An Efficient BIST Scheme Based on Reseeding of Multiple Polynomial Linear Feedback Shift Registers

Srikanth Venkataraman* Janusz Rajski* Sybille Hellebrand** Steffen Tarnick***

*MACS Laboratory, McGill University, Montreal, Canada - H3A 2A7
**University of Siegen, 57068 Siegen, Germany
***Max-Planck Society, 14469 Potsdam, Germany.

Abstract

In this paper we describe an optimized BIST scheme based on reseeding of multiple polynomial Linear Feedback Shift Registers (LFSRs). The same LFSR that is used to generate pseudo-random patterns, is loaded with seeds from which it produces vectors that cover the testcubes of difficult to test faults. The scheme is compatible with scan-design and achieves full coverage as it is based on random patterns combined with a deterministic test set. A method for processing the test set to allow for efficient encoding by the scheme is described. Algorithms for calculating LFSR seeds from the test set and for the selection and ordering of polynomials are described. Experimental results are provided for ISCAS-89 benchmark circuits to demonstrate the effectiveness of the scheme. The scheme allows an excellent trade-off between test data storage and test application time (number of test patterns) with a very small hardware overhead. We show the trade-off between test data storage and number of test patterns under the scheme.

1 Introduction

The increasingly stringent demands for high chip quality at affordable costs dictate Built-In Self Test (BIST) schemes to guarantee high fault coverage. Complete, or very high fault coverage is expected to be produced by a simple vector generator in an acceptable number of patterns. Techniques for Test Pattern Generation (TPG) on chip can be classified into the following groups: exhaustive testing, stored pattern testing, pseudo-random testing, weighted random testing and mixed-mode vector pattern generation. To justify the use of any of these techniques (or a combination of them) in the BIST environment, the following quality gauges have to be considered: test data storage, test application time, number of test patterns, hardware overhead and fault coverage.

Exhaustive testing involves generating all combinations of vectors for every output [1]. Although this guarantees the coverage of all combinational faults, it is not feasible for circuits with outputs driven by a large number of inputs and is not generally applicable to sequential circuits. Stored pattern testing is eminently successful in terms of fault coverage and test application time however it tends to produce extremely large test data volumes. Pseudo-random pattern testing with Signature Analysis using an LFSR can be used to generate a large number of tests using miniscule data storage (LFSR seeds and signatures) at a very small area overhead. However unacceptable fault coverage is obtained due to random pattern resistant logic structures (e.g., large fan-in).

Weighted random patterns characterize complete test sets in a compact form by signal probabilities for every input of the circuit [1]. Pseudo-random patterns biased to these probabilities maximize the coverage of hard to test faults, reducing the length of the test. Different faults may require different, conflicting values of signal probabilities [2]. This, combined with the objective to keep the length of the test set within reasonable limits and guarantee high fault coverage, could lead to a large volume of test data, and complex additional hardware.

Mixed-mode generators use LFSRs or other generators as a source of pseudo-random vectors which cover a large percentage of easily testable faults. Automatic test pattern generation is used to target random pattern resistant faults. A number of techniques have been developed to generate deterministic sequences of vectors [3, 4, 5]. Most of these techniques do not take advantage of the fact that test vectors are usually given in the form of testcubes with many unspecified inputs. They are therefore not suitable for scan design.

Reseeding of LFSRs [6, 7] has been proposed as a technique compatible with scan design. The same LFSR that is used to generate pseudo-random patterns is loaded with seeds from which it produces vectors that cover the testcubes of difficult to test faults. [7] examined the effectiveness of encoding of schemes based on reseeding and polynomial programming and proposed a scheme based on multiple feedback polynomials that offers the best trade-off between encoding efficiency and computational complexity.

While the work in [7] was mainly concentrated on the problem of encoding single testcubes, in this paper we propose a scheme which supports an efficient BIST implementation of the entire test set. The scheme involves compressing a deterministic test set into a set of seeds. These seeds are then used by the multiple polynomial LFSR to generate the original test set. A method for processing the test set so as to allow efficient encoding under the scheme is proposed. The problem of encoding the test set into seeds is discussed and an efficient algorithm for calculating LFSR

seeds from the test set is described. A method for balancing testcubes with respect to the characteristic polynomials of the multiple polynomial LFSR is discussed. Extensive experimental studies are performed on the ISCAS-89 circuits to demonstrate the effectiveness of the scheme.

2 An Optimized BIST Scheme Based on Multiple Polynomial LFSRs

The BIST approach followed in this paper assumes that mixed mode test generation is used to obtain full coverage for large scan based circuits. An LFSR is used to generate pseudo-random patterns that cover most of the faults in the circuit under test (CUT). The remaining hard to test faults are covered by deterministic testcubes which are stored in a compact representation and decoded for test application. In [7] it was shown that multiple polynomial LFSRs with partially programmable seeds allow a very efficient encoding of a deterministic testcube. To encode a testcube with s carebits s+4 bits are sufficient to keep the probability of not finding an encoding below 10^{-6}, whereas a classical reseeding approach would require $s + 19$ bits [6, 7]. With respect to a BIST implementation of the entire test the multiple polynomial architecture described in [7] can be optimized further. Before the details of the new architecture are described in section 2.2 the main features of multiple polynomial LFSRs with partially programmable seeds are summarized in section 2.1.

2.1 Reseeding of Multiple Polynomial LFSRs

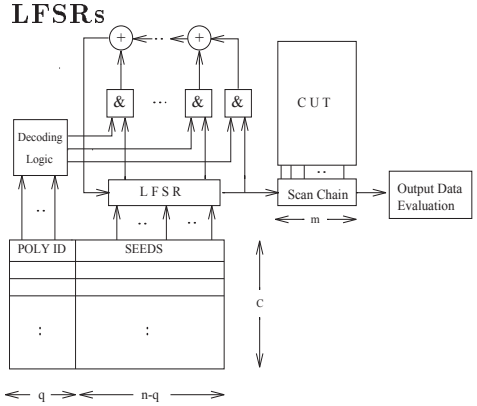

Figure 1: BIST scheme based on multiple polynomials.

Figure 1 illustrates encoding and decoding of deterministic testcubes using multiple polynomial LFSRs. An m-bit testcube is encoded into a n-bit word. The first q components are used to identify one out of 2^q different feedback polynomials of degree k and the remaining n-q bits provide the programmable part of the seed. In order generate a testvector from this information the seed is loaded into the LFSR and the feedback links are established according to the polynomial identifier. Subsequently, in m clock cycles the LFSR produces serially the bits of the test vector which are shifted into the scan chain. This test vector is then applied to the CUT.

To encode a given testcube $C = (c_0, \ldots, c_{m-1}) \in \{0, 1, x\}^m$ at most 2^q systems of linear equations have to be treated. For a fixed feedback polynomial $h(X) = X^k + \sum_{i=0}^{k-1} h_i X^i$ the LFSR produces an output sequence $(a_i)_{i \geq 0}$. The first m components of this sequence can be used as test patterns if $a_i = c_i$ holds for all specified bit positions $i \in S(C) := \{ i = 0, \ldots, m\text{-}1 \mid c_i \neq x\}$. By recursively applying the feedback equation $a_n = \sum_{i=0}^{k-1} h_i a_{n-k+i}$ to the equations $a_i = c_i$ a system of linear equations in the seed variables $a_0, \ldots, a_{k-1}$ is obtained. If the system has a solution (a_0*, ..., a_{k-1}*), the encoding of the testcube is given by the corresponding polynomial identifier and the seed (a_0*, ..., a_{k-1}*). If the system has no solution the next polynomial is tried.

In section 4 an efficient procedure to solve linear equations is described. Depending on the choice of the parameters n, k and q the scheme provides different BIST implementations ranging between the two extremes of fully programmable feedback polynomials with fixed seed values and reseeding of single-polynomial LFSRs. In [7] a probabilistic model is given to evaluate different implementations. It is shown that the general scheme with 16 polynomials practically achieves the encoding efficiency of fully programmable LFSRs. Thus s+4 bits are sufficient to keep the probability that no encoding can be found below 10^{-6}. The computational effort is comparable to reseeding of single polynomial LFSRs.

2.2 Implicit Encoding of Polynomials

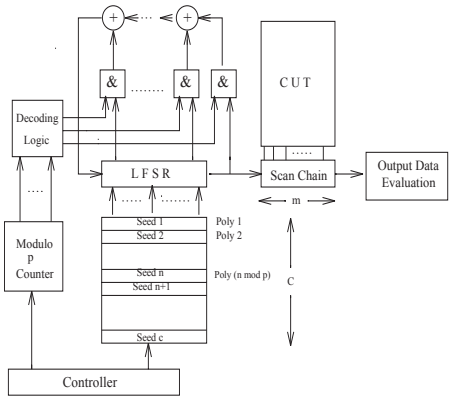

Figure 2: BIST scheme based on implicit encoding of polynomials.

As demonstrated in [7] the technique described in section 2.1 is very efficient with respect to the encoding of a single testcube. However, if the problem of encoding an entire test set is considered as a whole considerable optimizations can be obtained. The encoding of the required feedback polynomial can be done implicitly by ordering the testcubes. If the testcubes can be partitioned into groups of equal size such that each group is generated by one polynomial then it is possible to construct a sequence of testcubes corresponding to a periodic sequence of polynomials. In this case the switching from one polynomial to another can be controlled by a mod p counter, when p is the period of the sequence of polynomials. The BIST implementation is shown in figure 2.

With this approach s bits are sufficient to encode a

testcube with s carebits with a probability of failure below 10^{-6}. The fact that some testcubes might be generated by several polynomials can be used to construct a good ordering of the testcubes. In section 5 an ordering algorithm is presented. The scheme thus reduces the test data storage. However there is a possible increase in test application time due to uneven partitioning. If the difference in the size of any two partitions is greater than one then dummy testcubes have to be added to balance the arrangement resulting in increased test application time. The dummy testcubes can be regarded as additional random patterns. In practice this is not much of an overhead as shown by the experimental results. Further a higher computational effort is required as compared to the original scheme. Every testcube has to be analyzed against every polynomial as opposed to the original scheme where the number of polynomials analyzed for each testcube is only slightly more than one.

3 Compaction of Testcubes

This section describes a method to compact the set of testcubes. Compaction reduces the test application time and the memory required to store the encoded testcubes. This is achieved by reducing the number of testcubes in the test set and by improving the efficiency of encoding these cubes into seeds.

The length of the programmable LFSR to encode a set T of testcubes C_i, $i = 1, \ldots, c$ depends on the maximum number μ of specified bits in any testcube in T. The number of specified bits μ_i in any other testcube C_i is in general much smaller than μ. Encoding these testcubes into a word of size μ results in inefficient encoding of these testcubes. The difference $\mu - \mu_i$ is a measure of the encoding overhead on the cube C_i. Thus the encoding overhead O for the test set T is defined as :

$$O = 1 - \frac{1}{c\mu} \sum_{i=1}^{c} \mu_i, \qquad 0 \leq O \leq 1 \tag{1}$$

and the encoding efficiency E is defined as

$$E = \frac{1}{c\mu} \sum_{i=1}^{c} \mu_i = 1 - O, \qquad 0 \leq E \leq 1 \tag{2}$$

Compaction is done in two steps. First a simplification process is performed followed by a merging process as explained below.

The first step involves removing testcubes which contain other testcubes.

Definition 1 *Let $u = u_0 \ldots u_{m-1} \in \{0,1,x\}^m$ and $v = v_0 \ldots v_{m-1} \in \{0,1,x\}^m$ be two testcubes. We say that u is* contained *in v (or u is a* subcube *of v), $u \subseteq v$, if and only if $S(v) \subseteq S(u)$, and $\forall\ i \in S(v)\ v_i = u_i$ holds ($S(C)$ is defined in section 2.1).*

Obviously, a testcube u contained in a testcube v detects the same faults as v. If u, v $\in$ T and u $\subseteq$ v then v can be removed from T. This can be done by pairwise checking of testcubes for containment.

The second step involves merging consistent testcubes. The size of the programmable LFSR depends on the maximal number μ of specified bits in a testcube. Consistent testcubes can be merged if the number of specified bits in the resulting testcube is smaller than or equal to μ.

Let T = $\{C_i\}$ be a set of testcubes and let $\mu = max_{C \in T}\{s(C)\}$. Then s(C) $\leq \mu$ holds for all C $\in$ T and for most C $\in$ T the relation s(C) $\ll \mu$ is expected, where s(C) = |S(C)|.

Definition 2 *Let $u = u_0 \ldots u_{m-1} \in \{0,1,x\}^m$ and $v = v_0 \ldots v_{m-1} \in \{0,1,x\}^m$ be two testcubes. We call u and v* consistent, *$u \sim v$, if $\forall\ i \in S(v) \cap S(u)\ v_i = u_i$ holds.*

If two testcubes are consistent we can merge or intersect them in the following way.

Definition 3 *For two consistent testcubes u and v we define a* merging operation *(or* intersection*) $w = u \cap v$ with $S(w) = S(u \cap v) = S(u) \cup S(v)$ and $\forall i \in S(u) : w_i := u_i$ and $\forall i \in S(v) : w_i = v_i$.*

The objective is to merge pairwise consistent testcubes in such a way that the resulting number of testcubes is a minimum. This can be viewed as a graph theoretic problem. We define a graph G in such a way that the nodes correspond to the testcubes and two nodes are connected by an edge if the corresponding testcubes are consistent. The following example illustrates this point.

Example 1: Consider the set T=$\{C_1, C_2, C_3, C_4, C_5, C_6\}$ of testcubes with C_1=x1xx0x11xx, C_2=xxx10xx1xx, C_3=0x1xxxx0xx, C_4=xxx00xxxxx, C_5=xx1xxxx0xx, C_6=x101x1xxx0.Testcube C_5 can be dropped as it contains C_3. The graph in figure 3 shows the consistency relationship between the testcubes. □

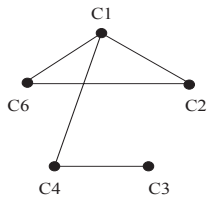

Figure 3: Graph G corresponding to T.

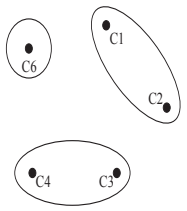

Figure 4: Partitioning of T.

The problem of finding a minimal number of resulting testcubes by merging pairwise consistent testcubes is equivalent to the problem of *partitioning the graph into cliques* [8]. The problem is an NP-complete problem and we have to use heuristic methods to solve it. The algorithm is shown in figure 5.

In MERGING the procedure MAXCLIQUE computes the largest subset M of T where each pair of testcubes of M is consistent. This corresponds to finding the largest clique in G. An additional constraint s($\cap$M) $\leq \mu$ has to be taken into account.

The following example illustrates the algorithm.

Example 1 (continued): The maximum value of specified bits in a testcube of T is $\mu = 5$. The largest clique in G consists of the nodes C_1, C_2 and C_6. However, since $s(C_1 \cap C_2 \cap C_6) = 7 > \mu$ so MAXCLIQUE has to examine the combinations $C_2 \cap C_6, C_1 \cap C_6$ and $C_1 \cap C_2$ where only $s(C_1 \cap C_2) = 5 \leq \mu$. For the remaining set T = $\{C_3, C_4, C_6\}$ the largest clique is M = $\{C_3, C_4\}$ with $s(C_3 \cap C_4) =$

5. The reduced set of testcubes is $C_1 \cap C_2$=x1x10x11xx, $C_3 \cap C_4$=0x100xx0xx and C_6=x101x1xxx0. A graphical representation of the partitioning of T is shown in figure 4. □

```
MERGING()
begin
  T_min := ∅
  repeat
      M := MAXCLIQUE(T)
      C := ∩ M
      T_min := T_min ∪ {C}
      T := T \ M
  until T = ∅
  return T_min
end
```

Figure 5: Algorithm for Merging Testcubes.

4 Computing Seeds from Testcubes

The test vectors are generated by loading the LFSR with precomputed seeds. The seeds can be computed by solving systems of linear equations selected by the *"care"* or specified bits of the testcube as described in section 2.1.

The output sequence of the LFSR can be expressed as a sequence of linear (XOR) functions (or symbolic expressions) of the seed $a(0) = (a_0, \ldots, a_{k-1})$ variables. As explained in section 2.1 computing the symbolic expressions involves using the LFSR feedback equation recursively. The expressions for the output sequence (a_i) for i = k to m-1 (testcube length) are generated in one forward pass using expressions $a_{i-k}, \ldots, a_{i-1}$, where k is the degree of the feedback polynomial.

The Gauss-Jordan elimination with maximum pivot strategy [9] is an efficient and stable method for solving a system of linear algebraic equations. This procedure is appropriately modified to obtain an algorithm to solve the system of linear equations in boolean variables. Since the variables only take a value of either 1 or 0, normalization to reduce diagonal elements to 1, in the elimination process, is not required. Further off-diagonal elements can be reduced to zero by simply adding equations.

Space and time optimizations are performed on the algorithm by compactly storing boolean equations in bit format and reducing the equations by simply performing bit operations (XORing). The algorithm requires N^2 XOR operations, where N is the number of equations.

5 Balancing Testcubes with Respect to Polynomials

As pointed out in section 2.2 an optimized BIST for a multiple-polynomial architecture is based on an implicit assignment of testcubes to polynomials given by the ordering of the testcubes. The best implementation is obtained, if a sequence of testcubes can be constructed which corresponds to a periodic sequence of polynomials with a small period. In this case the switch from one feedback polynomial to another can be controlled by a small cyclic counter. To increase the optimization potential it is useful to determine for each testcube all applicable feedback polynomials using the procedure described in section 4. This is illustrated by the following example.

Example 2: Let T = {C_1 = xx1x0xxx, C_2 = x1xx1xxx, C_3 = xxxxx00x, C_4 = xx1xxxx1} be the test set to be generated by a multiple-polynomial LFSR of degree 4. Let the LFSR operate according to the four polynomials p_1(X) = $X^4 + X^3 + 1$, p_2(X) = $X^4 + X^2 + 1$, p_3(X) = X^4 + X + 1, p_4(X) = $X^4 + X^2$ + X + 1. The seed values a_0 and a_1 are assumed to be fixed to $a_0 = 0$, $a_1 = 1$.

Solving the equations for the four testcubes for all possible polynomials provides a list P(C) of applicable polynomials for each testcube C = C_i, $i = 1, \ldots, 4$.

P(C_1) = {p_1,p_4}, P(C_2) = P(C_3) = {p_1,p_2,p_3,p_4} and P(C_4) = {p_2,p_3}.

This shows that in fact 2 polynomials are sufficient for a BIST implementation. If p_1 is used to generate C_1 and C_2, and p_2 to generate C_3 and C_4, then the sequence (C_1,C_2,C_3,C_4) implies the sequence (p_1,p_1,p_2,p_2) of polynomials. Reordering provides the periodic sequence (p_1,p_2,p_1,p_2) with minimal period 2 (the period cannot be further reduced, since P(C_1) and P(C_4) are disjoint). □

In fact it is not necessary to construct a 'perfectly' periodic sequence as demonstrated in example 2. Sequences of the form (p_1, ..., p_r, p_1, ..., p_r, ..., p_1, ..., p_i), i < r, can be implemented by controlling a mod-r counter such that it stops in the last cycle after i transitions. Furthermore sequences can be made periodic by reordering and inserting appropriate polynomials from the set P. For test application arbitrary seeds or seeds calculated for specific testcubes can be used for the inserted polynomials. However, this results in additional test time and additional storage overhead for such seeds. Therefore the number of additional polynomials should be kept as small as possible.

In general the following problem of *balancing polynomials* has to be solved :

Problem BP (Balancing Polynomials):

Let T = {C_1, ..., C_n} be a set of testcubes and P be a set of polynomials. For each testcube C ∈ T let P(C) ⊆ P denote the set of polynomials applicable to C. Find an assignment π: T ↦ P, such that:

(i) π(C) ∈ P(C) for all C ∈ T, (ii) The sequence ($\pi(C_1)$, ..., $\pi(C_n)$) can be transformed (by reordering and inserting polynomials) to a sequence of minimal period and minimal length.

An exact solution of BP efficient algorithms cannot be expected since the problem belongs to the class of NP-complete problems. The completeness of BP can be shown by reducing the problem *Hitting Set* [8] to BP. Due to the complexity of the problem BP a heuristic is used to balance the polynomials p ∈ P for a test set T. The basic idea is to attach to each polynomial p ∈ P a list T(p) of testcubes that can be generated by this polynomial and try to reduce these lists until small irreducible lists are obtained allowing a good BIST implementation. The initial lists contain all testcubes that can be generated by a polynomial. They are constructed from the lists P(C) of applicable polynomials, which have been determined for each C ∈ T, using the equation solving procedure of section 4, by

$$T(p) := \{ C \in T \mid p \in P(C) \}.$$

A list T(p) is reducible, if there is a testcube C $\in$ T(p) which can also be generated by another polynomial p' $\in$ P, p' $\neq$ p, i.e. C $\in$ T(p'). In this case C can be removed from T(p) to produce a reduced list T*(p) := T(p) $\setminus$ {C}. The strategy of the proposed algorithm is to reduce the largest list first in each step and to remove the testcube that can be generated by the largest number of polynomials.

Figure 6 gives the description of the algorithm. Input are the sets T, P and {P(C) | C $\in$ T} and output are the sets {T(p) | p $\in$ P}.

From the reduced lists T(p), p $\in$ P, provided by the algorithm of figure 6, the sequence of polynomials required for the BIST implementation can easily be constructed by appropriate ordering and inserting additional polynomials from P. Altogether w*M + (w-1)(|P| - M) seeds have to be stored, where w := max{|T(P)| : p $\in$ P} and M := |{p $\in$ P : |T(p)| = w}|.

```
BALANCE ();
begin
  for each p ∈ P do
    T(p) := {C ∈ T | p ∈ P(C)};
  while there is a p ∈ P with T(p) reducible do begin
    select p* ∈ P with T(p*) reducible and
      |T(p*)| = max{|T(p)| : T(p) reducible};
    select C* ∈ T(p*) with |P(C*)| maximal;
    T(p*) := T(p*) \ C*;
    P(C*) := P(C*) \ p*;
  end
end
```

Figure 6: Heuristic for balancing polynomials.

6 Experimental Results

Experiments were performed on the ISCAS 89 benchmark circuits [10] to study the BIST scheme. The benchmark circuits were fault simulated [11] for a varying number of random patterns. Faults not covered formed a set of hard to test faults. ATPG (Socrates [**?**]) was then used to target these faults. We then enlarged the number of don't cares in these test vectors, by determining which assignments were no longer needed, using a *bit stripping process*. The process consists of assigning to each testcube a set of unique faults which excludes those faults covered by any previous testcube. Next, each testcube's assignments were successively replaced by a don't care value and the testcube's fault coverage was resimulated. If the coverage remained unchanged, the replaced assignment was deemed unnecessary and removed. A second pass of fault simulation was used to guarantee that the testcube covers the assigned faults. The compaction process as explained in section 3 reduced the number of testcubes to be encoded and improved their encoding efficiency under the scheme. This reduced both the test application time and the memory required to store the encoded testcubes. Table 1 characterizes the compaction process in terms of the total number of testcubes and the encoding efficiency (E).

The compacted testcubes were encoded and balanced with respect to 16 polynomials as described in section 5. A balanced arrangement for a complete test set was used to obtain the seed data volume. Table 2 compares the test data volume under the scheme to that for stored pattern testing with signature analysis. The tradeoff curves between the storage requirements and random pattern lengths for complete test sets were plotted. Figure 7 shows these curves for some of the benchmark circuits.

Circuit	# Rand. Patterns	Unmerged		Merged	
		# Test cubes	(E)	# Test Cubes	(E)
s5378	1k	135	0.31	85	0.49
	2k	97	0.39	70	0.54
	5k	51	0.44	44	0.51
	10k	35	0.44	32	0.48
s9234	1k	313	0.28	152	0.51
	2k	284	0.27	139	0.49
	5k	231	0.32	116	0.55
	10k	196	0.34	107	0.54
s13207	1k	398	0.17	196	0.33
	2k	322	0.17	171	0.30
	5k	247	0.17	135	0.30
	10k	174	0.17	105	0.26
s35932	32	49	0.48	49	0.48
	64	34	0.42	34	0.42
	128	9	0.78	9	0.78
s38417	100k	333	0.23	91	0.80
	120k	308	0.23	86	0.78
	150k	271	0.27	91	0.77
	200k	244	0.27	78	0.79
	250k	210	0.34	88	0.76

Table 1: Statistics on the compaction of testcubes (ISCAS 89 Circuits).

7 Conclusions

A BIST scheme based on reseeding of multiple polynomial LFSRs was presented. The scheme offers an excellent trade-off of the following parameters: number of test patterns (test application time), hardware overhead and test data storage. Very small hardware overhead (LFSR, control circuitry and memory to store seeds) is required. The method is compatible with scan design and 100 % fault coverage is obtained. The test data is small (seeds for LFSR) and very small compared to stored patterns data. It is possible to trade-off testing time (number of test patterns) against test data. The compaction of the test set gives a reduction in the number of test patterns by a factor 2 on the average. The average encoding efficiency is around 0.6. Work is currently being done to further improve the encoding efficiency by concatenating cubes to encode multiple cubes in a seed.

The method is also useful with an external tester. It reduces the bandwidth of data transfer between tester and chip by sending in the seeds. The scheme can be extended for board level test, testing of Multi Chip Modules and boards populated with FPGAs and delay fault testing.

Circuit	# Rand. Patterns †	Seed volume	Stored Pattern ‡ data volume
s5378	1k	11008	57K
	2k	8960	
	5k	5632	
	10k	4096	
s9234	1k	19152	101K
	2k	17154	
	5k	14616	
	10k	13482	
s13207	1k	59175	343K
	2k	44973	
	5k	35505	
	10k	6615	
s35932	32	12650	128K
	64	8602	
	128	2277	
s38417	80k	20608	1.6M
	100k	19292	
	120k	18232	
	150k	16016	
	200k	13728	
	250k	12144	

Table 2: Test data volume (ISCAS 89 Circuits).

† The stored pattern data is calculated using a complete test set obtained using Socrates. ‡ The number of random patterns range from zero to those required for complete coverage or upto a point beyond which there is no significant increase in coverage.

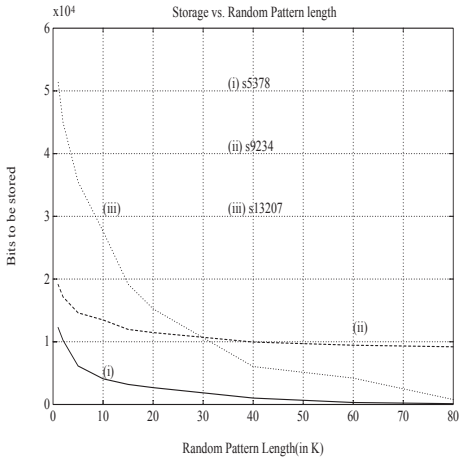

Figure 7: Tradeoff between storage and random pattern length, large circuits.

We started with fully specified or over-specified deterministic test vectors and reduced the number of unnecessary assignments through fault simulation. Another approach would be to identify necessary assignments during deterministic test pattern generation and produce minimally specified testcubes. This would give better results, however we would be trading-off test generation time to obtain a more compactly characterizable test set. Further reduction in the test data volume is possible if the deterministic test set to target the hard to test faults is compacted [13].

Acknowlegdements

This work was supported in parts by strategic grant MEF0045788 from the *Microelectronics Fund* of the Natural Science and Engineering Research Council of Canada, and the ESPRIT project 7107 ARCHIMEDES. SOCRATES was provided by the Technical University of Munich.

References

[1] P. Bardell, W.H. McAnney, J. Savir, "Built-In Test for VLSI", Wiley-Interscience, New York, 1987.

[2] H.J. Wunderlich, "Multiple Distributions for Biased Random Test Patterns", *Proc. IEEE Int. Test Conf.,* Washington D.C., 1988, pp. 236-244.

[3] W. Daehn and J. Mucha, "Hardware Test Pattern Generators for Built-In Test", *Proc. IEEE Int. Test Conf.*, 1981, pp. 110-113.

[4] S.B. Akers and W. Jansz, "Test Set Embedding in Built-In Self-Test Environment", *Proc. IEEE Int. Test Conf.,* Washington D.C., 1989, pp. 257-263.

[5] C. Dufaza and G. Cambon, "LFSR Based Deterministic and Pseudo-Random Test Pattern Generator Structures", *Proc. Europ. Test Conf.*, Munich, 1991, pp. 27-34.

[6] B. Koenemann, "LFSR-Coded Test Patterns for Scan Designs", *Proc. Europ. Test Conf.*, Munich, 1991, pp. 237-242.

[7] S. Hellebrand, S. Tarnick, J. Rajski and B. Courtois, "Generation of Vector Patterns Through Reseeding of Multiple-Polynomial Linear Feedback Shift Registers", *Proc. IEEE Int. Test Conf.*, Baltimore, Sept. 1992, pp. 120-129.

[8] Michael R. Garey and David S. Johnson, *"Computers and Intractability: A Guide to the Theory of NP-Completeness"*, W.H.Freeman, New York, 1979, p. 193 and p. 222.

[9] A. Ralston and P. Rabinowitz, "A First Course in Numerical Analysis", 2nd ed., McGraw Hill, New York, 1978, §9.3.1.

[10] F. Brglez, D. Bryan and K. Kozminski, "Combinational Profiles of Sequential Benchmark Circuits", *Proc. IEEE Int. Symp. Ciruits & Systems,* May 1989, pp. 1929-1934.

[11] F. Maamari and J. Rajski, "A Method of Fault Simulation Based on Stem Regions", *Proc. IEEE Trans. on CAD,* Vol. 9, No. 2, February 1990, pp. 212-220.

[12] M. Schulz and E. Auth, "Advanced Automatic Test Generation and Redundancy Identification Techniques", *Proc. FTCS-18,* Tokyo 1988, pp. 30-35.

[13] I. Pomeranz, L.N. Reddy and S.M. Reddy, "COMPACTEST: A Method to Generate Compact Test Sets for Combinational Circuits", *IEEE Int. Test Conf.*, 1991, pp. 194-203.