Special ATPG to Correlate Test Patterns for Low-Overhead Mixed-Mode BIST

Madhavi Karkala Nur A. Touba

Computer Engineering Research Center
Dept. of Electrical and Computer Engineering
University of Texas, Austin, TX 78712

Hans-Joachim Wunderlich

Computer Architecture Lab
University of Stuttgart, Breitwiesenstr. 20-22
70565 Stuttgart, Germany

Abstract

In mixed-mode BIST, deterministic test patterns are generated with on-chip hardware to detect the random-pattern-resistant (r.p.r.) faults that are missed by the pseudo-random patterns. While previous work in mixed-mode BIST has focused on developing hardware schemes for more efficiently encoding a given set of deterministic patterns (generated by a conventional ATPG procedure), the approach taken in this paper is to improve the encoding efficiency (and hence reduce hardware overhead) by specially selecting a set of deterministic test patterns for the r.p.r. faults that can be efficiently encoded. A special ATPG procedure is described for finding test patterns for the r.p.r. faults that are correlated (have the same logic value) in many bit positions. Such test patterns can be efficiently encoded with one of the many "bit-fixing" schemes that have been described in the literature. Results are shown for different bit-fixing schemes which indicate dramatic reductions in BIST overhead can be achieved by using the proposed ATPG procedure to select which test patterns to encode.

1. Introduction

In built-in self-test (BIST), the test patterns that are applied to the circuit-under-test are generated with on-chip hardware. In order to minimize the overhead for BIST, the test pattern generation hardware must be compact. For this reason, pseudo-random testing is an attractive approach for BIST because a linear feedback shift register (LFSR), which has a simple compact structure, can be used to generate the test patterns. Unfortunately, however, the circuit-under-test may contain random-pattern-resistant (r.p.r) faults which have low detection probabilities and therefore limit the fault coverage that can be obtained with pseudo-random testing [Eichelberger 83].

One way to improve the fault coverage is to use *mixed-mode BIST* in which some deterministic patterns are generated with on-chip hardware to detect the r.p.r. faults that the pseudo-random patterns miss. This paper focuses on the problem of how to efficiently encode the deterministic patterns on-chip in order to minimize the overhead for mixed-mode BIST. Many different schemes for encoding deterministic test patterns on-chip have been developed. These schemes can be broadly categorized as follows:

1. <u>ROM-Based Schemes</u> - The simplest way to generate deterministic patterns on-chip is to store them in a ROM, however, the size of the required ROM is often prohibitive. Several compression techniques have been developed for reducing the size of the ROM [Agarwal 81], [Aboulhamid 83], [Dandapani 84], [Edirisooriya 92], [Dufaza 93].
2. <u>Reseeding Schemes</u> - Instead of storing the deterministic patterns themselves in a ROM, techniques have been developed for storing LFSR seeds that can be used to generate the deterministic patterns [Koenemann 91], [Hellebrand 92, 95a], [Venkataraman 93], [Zacharia 95, 96]. The LFSR that is used to generate the pseudo-random patterns is also used to generate deterministic *test cubes* (test patterns with unspecified inputs) by loading it with computed seeds. The number of bits that needs to be stored is reduced by storing a set of seeds instead of the deterministic patterns themselves.
3. <u>Counter-Based Schemes</u> - Several different techniques have been developed for designing a special counter that generates a set of deterministic patterns [Daehn 81], [Akers 89], [Dufaza 91, 95], [Kangaris 96ab], [Wunderlich 96], [Kiefer 97].
4. <u>Bit-Fixing Schemes</u> - "Bit-fixing" involves generating pseudo-random patterns and fixing the logic value of certain bit positions to cause the pseudo-random patterns to match deterministic test cubes. Bit-fixing has been found to be a very effective approach for generating deterministic test cubes for the r.p.r. faults. Several different bit-fixing schemes have been developed [Pateras 91], [Pomeranz 93a], [AlShaibi 94, 96], [Chatterjee 95], [Touba 95a, 95b, 96].

One characteristic that all of these schemes share is that the overhead depends on the set of patterns that is encoded. Some sets of patterns require less hardware to encode than others. While most previous work in mixed-mode BIST has focused on developing different schemes for more efficiently encoding a *given* set of deterministic patterns (generated by a conventional ATPG procedure), another approach for improving the encoding efficiency (and hence reduce hardware overhead) is to develop new ATPG procedures for specially *selecting* a set of deterministic test patterns for the r.p.r. faults that can be efficiently encoded with a particular hardware encoding scheme. Each r.p.r. fault is typically detected by a number of different test patterns. Thus, an additional degree of freedom in minimizing the hardware for mixed-mode BIST is in selecting which deterministic pattern to encode for each r.p.r. fault. How to make use of this degree of freedom is the subject of this paper.

Some previous work has been done in the area of developing ATPG procedures for improving encoding efficiency. Several techniques exist for minimizing the total number of deterministic test patterns that need to be encoded [Tromp 91], [Pomeranz 93b], [Kajihara 93]. These techniques are particularly effective for encoding schemes that involve storing the patterns in a ROM. In [Hellebrand 95b], a special ATPG procedure was described for improving the encoding efficiency for an encoding scheme where deterministic test cubes are generated by reseeding a multiple-polynomial LFSR. Decisions made during ATPG are guided by heuristics aimed at generating test cubes that have a large number of unspecified bits while keeping the overall number of test cubes small. Results show a dramatic improvement in the encoding efficiency with this approach. In [Reeb 96], an ATPG procedure was described for maximizing the number of unspecified bits in each deterministic test cube. In general, increasing the number of unspecified bits improves the encoding efficiency for most schemes, but it is not necessarily the most effective approach for improving the encoding efficiency.

This paper presents an ATPG procedure for improving the encoding efficiency for bit-fixing schemes such as those described in [Pateras 91], [Pomeranz 93a], [AlShaibi 94, 96], [Chatterjee 95], and [Touba 95a, 95b, 96]. Bit-fixing is very efficient for generating a set of test cubes that have the same specified logic value in particular bit positions (this will be referred to as "bit correlation"). For example, the test cubes *11011*, *11X00*, and *1X0X0*, are correlated in the 1st, 2nd, and 3rd bit positions, but not in the 4th and 5th. That is because all of the specified bits in the 1st and 2nd bit positions are 1's, and all of the specified bits in the 3rd position are 0's. However, the 4th and 5th bit positions have conflicts because some of the specified values are 1's and some are 0's. Note that the unspecified values (X's) don't matter. For a set of test cubes that is correlated in several bit positions, the correlated bit positions can be fixed to a particular logic value while the rest of the bits are pseudo-randomly generated. The amount of hardware required for generating a set of test cubes in a particular test length with bit-fixing depends on how correlated the test cubes are. The set of test cubes can be partitioned into subsets where the test cubes in each subset are correlated in a sufficient number of bit positions to enable them to be generated in a reasonable test length with bit-fixing. The bit-fixing hardware depends on how many such subsets there are. This paper presents an ATPG procedure for generating a set of test cubes for the r.p.r. faults in a way that maximizes the bit correlation among the test cubes. The set of test cubes obtained with this ATPG procedure can be used to design efficient bit-fixing hardware for detecting the r.p.r. faults.

Note that while the procedure in this paper is described for bit-fixing schemes in particular, its applications extend to other encoding schemes as well. ROM-based "store-and-generate" schemes (e.g., [Agarwal 81], [Aboulhamid 83]) store correlated test cubes in memory and generate the remaining bits by counters. Maximizing correlation reduces both memory size and test length. Some counter-based schemes work best for correlated patterns as well (e.g., [Wunderlich 96], [Kiefer 97]). In short, the contribution of this paper is a general ATPG tool that can be used to generate correlated test sets which are useful in many existing and potentially future test encoding schemes.

2. Partitioning Test Cubes for Bit-Fixing

The idea in bit-fixing is that for a set of test cubes that is correlated in several bit positions, the correlated bit positions can be fixed at a constant logic value while the rest of the bits are pseudo-randomly generated. Very little hardware is required to do this since the pseudo-random generator is already implemented in mixed-mode BIST, and bits can be fixed by simply ANDing (for fixing to 0) or ORing (for fixing to 1) them with a control signal. For a test cube with n specified bits, if k of them are fixed, then the probability of generating the test cube by randomly specifying the remaining n-k bits is $2^{-(n-k)}$. Thus the test length required to generate the test cube with bit-fixing depends on how large n-k is. The value of n-k must be small enough to allow the test cube to be generated within an "acceptable" test length. What is considered an acceptable test length depends on the particular test environment.

Given a set of test cubes for the r.p.r. faults and a constraint on the value of *n-k*, the test cubes can be partitioned into subsets such that each subset is correlated in a sufficient number of bit positions to allow the value of *n-k* for each test cube to satisfy the constraint. These subsets of test cubes will be referred to as "bit-fixing groups" in this paper. The value of *k* for each bit-fixing group is equal to the number of correlated bit positions among the test cubes in the bit-fixing group.

The amount of hardware required for bit-fixing depends on how many bit-fixing groups there are. So the task at hand is to find test cubes for the r.p.r. faults that are correlated in a way that minimizes the number of bit-fixing groups needed to satisfy a given constraint on *n-k*. The proposed strategy for doing this involves forming the bit-fixing groups one at a time. When forming each bit-fixing group, an attempt is made to maximize the number of test cubes that are included in each bit-fixing group under the constraint on *n-k*. The proposed procedure for forming each bit-fixing group is described below:

Input: Undetected r.p.r. faults and constraint on *n-k*

Output: Bit-fixing group, *BF-GROUP*, that satisfies constraint on *n-k*

Step 1: Perform ATPG for each r.p.r. fault using the procedure in [Reeb 96] to maximize the number of unspecified bits.

This step finds a test cube for each fault with a minimal number of specified bits (smallest *n*).

Step 2: Select test cube with the largest number of specified bits (largest *n*) as the initial test cube in the *BF-GROUP*.

Of the test cubes found in step 1, the one with the largest number of specified bits (largest *n*) corresponds to the hardest to encode. Thus, this test cube is used as the initial test cube in the bit-fixing group.

Step 3: Initialize the value of *kmax[f]* for each r.p.r. fault, *f*, to infinity.

A variable *kmax[f]* is associated with each r.p.r. fault. It keeps track of the largest number of correlated bit positions (*largest k*) that a test cube for fault *f* currently has. Since no test cube for fault *f* has been found yet, this value is initialized to infinity.

Step 4: Identify the set *K* of correlated bit positions among the test cubes in the *BF-GROUP*

Identify the set of bit positions, *K*, where all the test cubes in the bit-fixing group have compatible values. This is the maximum set of bit positions that can be fixed. Initially, there is only one test cube in the bit-fixing group (the one added in step 2), thus there are no conflicts so all bit positions are initially contained in the set *K*. As additional test cubes are added to the bit-fixing group, bit positions are removed from *K* when conflicts are introduced.

Step 5: For the r.p.r. fault *f* that is not covered by the *BF-GROUP* and has the largest value of *kmax[f]*, use the special ATPG procedure described in Sec. 3 to find a test cube for it that differs in the fewest number of bit positions from the set *K* of correlated bit positions.

The fault *f* which has the largest value of *kmax[f]*, is the one whose test cube may minimize the number of conflicts with the *BF-GROUP*. The value of *kmax[f]* is only an upper bound, so ATPG must be performed to find a test cube for the fault and compute the exact number of correlated bit positions. The special ATPG procedure in Sec. 3 is used to find the test cube that maximizes the number of correlated bit positions.

Step 6: If the number of correlated bit positions for the test cube obtained for fault *f* in Step 5 is equal to *kmax[f]*, then add it to *BF-GROUP*, otherwise update the value of *kmax[f]* and loop back to Step 5.

If the number of correlated bit positions, *k*, in the test cube is in fact equal to *kmax[f]*, then that test cube will minimize the number of conflicts with the set *K* of correlated bits in the current *BF-GROUP*, so it is added to the *BF-GROUP*. However, if *k* is less than *kmax[f]*, then it forms a new upper bound and thus the value of *kmax[f]* is set equal to *k*. The reason why the current value of *k* forms an upper bound on future values of *k* is the fact that the number of correlated bit positions between test cubes for a particular r.p.r. fault and the *BF-GROUP* can only decrease (*BF-GROUP* can only become less correlated when additional test cubes are added to it). If the value of *k* is less than *kmax[f]*, then the test cube for another fault may be more correlated, so the procedure loops back to step 5.

Step 7: If the *BF-GROUP* still satisfies the constraint on *(n-k)*, then loop back to step 4. Otherwise, remove the last test cube added to the *BF-GROUP* and stop.

The procedure stops when no more test cubes can be added to the bit-fixing group without violating the constraint on *n-k*. When the last test cube added to the bit-fixing group (in step 6) causes it to no longer satisfy the constraint, then that test cube is removed from the bit-fixing group so that the constraint is satisfied once again and the procedure stops.

Once a bit-fixing group has been formed by this procedure, bit-fixing hardware can then be designed (using one of the many schemes described in the literature) to fix the correlated bit positions in the bit-fixing group. The hardware can be simulated to determine the set of patterns that are applied to the circuit-under-test. Fault simulation can then be done for the patterns to determine which r.p.r. faults remain undetected. The procedure can then be repeated for those faults. This process continues until the desired fault coverage is achieved.

The key step in the partitioning procedure is step 5 in which a special ATPG procedure (which will be described in Sec. 3) is used to find a test cube that is maximally correlated with the existing set of test cubes in the bit-fixing group. By specially choosing the test cubes in step 5, as opposed to just using a given set of test cubes, the procedure is able to find a better partitioning of the test cubes resulting in reduced bit-fixing hardware.

3. Correlating ATPG Procedure

In this section, the ATPG procedure for finding correlated test cubes is presented. Given a set of correlated bit positions, the problem being addressed is to find a test cube for a particular fault that conflicts with as few of the correlated bit positions as possible. This ATPG task is different from *dynamic compaction* [Goel 79] where an attempt is made to find a test cube for a fault by specifying the don't cares (X's) in test cubes for others faults. Dynamic compaction looks for a test cube for a particular fault that has *no conflicts* with other test cubes, whereas the problem of interest here is to find a test cube for a particular fault that has the *fewest number of conflicts* with other test cubes.

3.1 Initial Input Assignments

The "Correlating ATPG" procedure presented here uses a PODEM [Goel 81] based algorithm in which the inputs corresponding to the correlated bit positions are assigned initial values. Normally the PODEM algorithm begins with all inputs having unassigned values (X's). However, in the Correlating ATPG procedure, the initial input assignments are made to begin in the part of the search space that would yield the most correlated test cube. If the fault can be detected by making further inputs assignments without backtracking on any of the initial input assignments (i.e., the correlated bit positions), then a test cube can be found with no conflicts in the correlated bit positions. In general, however, some backtracking on the initial input assignments will be necessary to detect the fault. The key to maximizing the bit correlation is to carefully select the order of the backtracking in order to minimize the number of initial assignments that are reversed.

3.2 Backtracking

Normally, backtracking in the PODEM algorithm is done in the reverse order in which the inputs are assigned (i.e., the last input assignment made is the one that is changed first). Backtracking in the Correlating ATPG procedure is done in the same way except for when backtracking on the initial input assignments (i.e., the correlated bit positions). The order in which backtracking is performed on the initial input assignments is determined by using structural heuristics aimed at minimizing the number of initial input assignments that need to be reversed.

Backtracking on the initial input assignments is required when one of the line values implied by the initial input assignments must be complemented in order to allow the fault to be provoked or sensitized to a primary output by subsequent input assignments. If the value implied at the fault site is the same value as the fault polarity (i.e., if a 1 (0) is implied at a stuck-at 1 (0) fault site), then one or more initial input assignments must be reversed in order to either complement the value implied at the fault site or to imply an *X* at the fault site such that subsequent input assignment can provoke the fault. Backtracing is done to determine which initial input assignments to reverse. When there is a choice on which gate input to set to a controlling value, decisions are made based on minimizing the total number of initial input assignments that need to be reversed. If the fault site cannot be sensitized to a primary output with additional input assignments (i.e., no "X-path" exists from the "D-frontier" to a primary output), then line justification decisions for creating an X-path are again based on minimizing the total number of initial input assignments that need to be reversed. These decisions can be quickly made using the controllability and observability cost functions described in the next subsection.

3.3 Controllability and Observability Cost Functions

In the Correlating ATPG procedure, the goal is to minimize the number of initial input assignments that are reversed. Thus, the cost of justifying a line to a particular logic value or observing a line is the number of initial input assignments that need to be reversed. Controllability and observability values are computed to reflect this cost and used to guide line justification decisions. These values are computed when the initial input assignments are made and their implications are determined. The controllability values are determined by traversing the circuit from the primary inputs to the primary outputs. If no value is implied on a line (i.e., it is an X), then both the 0-controllability and 1-controllability values for that line are 0 since it can be justified to either logic value without reversing any of the initial input assignments. If the value implied on a line is a 0 (1), then the 0-controllability (1-controllabilty) is set to 0 and the 1-controllability (0-controllability) is set to the number of initial input assignments that need to be reversed in order to complement the value implied on the line or to imply an X on the line. Once the controllability values have been computed, then the observability values can be determined by traversing the circuit from primary outputs to primary inputs and using the controllability values to determine the number of initial input assignments that need to be reversed in order to make the line observable. An example of computing controllability and observability values is shown in Fig. 1. *C0*, *C1*, and *O* denote the controllability-0, controllability-1, and observability values, respectively, for each line. Note that there is no initial assignment for the fourth input (i.e., it is an X) so there is no cost for subsequent assignments to that input.

When making line justification decisions in Correlating ATPG, the controllability and observability values based on the number of initial input assignments that need to be reversed are the primary criteria. Of course, in many cases these values will be 0 or multiple decisions will have the same value. In those cases, the conventional ATPG heuristics (to minimize ATPG runtime) or the heuristics in [Reeb 96] (to maximize don't cares, i.e., minimize *n*) can be used.

Consider the example in Fig. 2. The fault being targeted is the output of gate *G5* stuck-at 1. Conventional ATPG would begin with all inputs

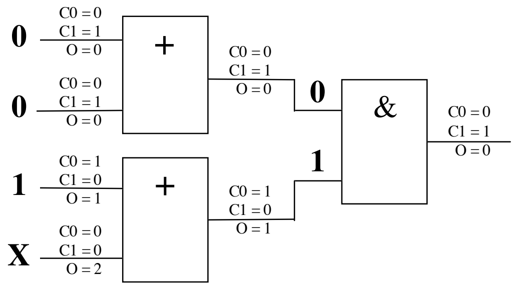

Figure 1. Controllability and Observability Values

initially unassigned (X's), however in Correlating ATPG, the initial input assignments correspond to the correlated bit positions. Implications based on the initial input assignments are made, and the controllability and observability values are computed based on the number of initial input assignments that need to be reversed as previously described. Since the value implied at the fault site is the same as the fault polarity, one or more of the initial input assignments must be reversed to justify a 0 at the fault site. Backtracing is done to determine which initial input assignments to reverse. Backtracing can be done through either gate *G3* or gate *G4*. Since the 0-controllability at the output of gate *G3* is less than the 0-controllability at the output of gate *G4*, backtracing is done through gate *G3*. Next there is a decision whether to backtrace through gate *G1* or gate *G2*. The 0-controllability values are equal for gate *G1* and gate *G2* because in either case, one input assignment will need to be reversed. In this case, a secondary criteria can be used in making the decision. For example, if the secondary criteria was to maximize the don't cares (X's), then backtracing would be done through gate *G1* since going through gate *G2* would require assigning a value to a currently unassigned input (in addition to reversing the input assigned to a 1).

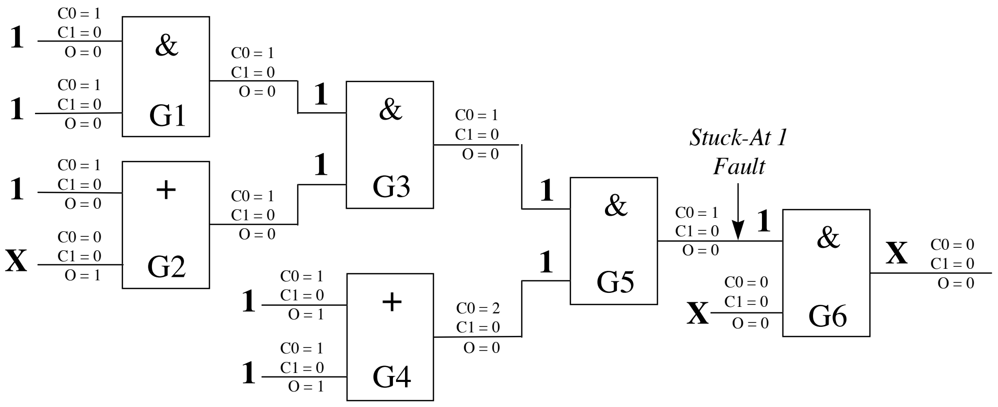

Figure 2. Example of Backtracking with Controllability and Observability Values

3.4 Post-Processing

The last step after a test cube that detects the fault has been found is to try to complement the value of any bit positions that conflict with the correlated bit positions. For each bit position that conflicts with a correlated bit position, the value is complemented and the resulting test cube is simulated to see if the fault is still detected. If the fault is no longer detected, then the bit position is returned to its previous value. Unlike the "maximal compaction" procedure described in [Pomeranz 93b], if it is possible to complement the bit, then the bit is left at the complemented value rather than making it an X. This is done to maximize the possibility of complementing other bits since the goal is to minimize the number of conflicts.

3.5 Backtracking Limit

The goal of the Correlating ATPG procedure is to maximize correlation as opposed to conventional ATPG procedures whose goal is to minimize execution time. One potential problem is that the heuristics used in the Correlating ATPG procedure may result in more backtracking. However, a limit can be placed on the backtracking based on the minimum amount of correlation that is acceptable. For the partitioning procedure described in Sec. 2, the constraint that is set on the value of *n-k* means that a test cube with less than a certain amount of bit correlation is not of interest. Thus, if the Correlating ATPG procedure backtracks on more than a certain number of initial input assignments, the procedure can be stopped since the fault will have to be covered by a different bit-fixing group anyway.

4. Experimental Results

Experiments were performed to compare "bit-fixing" hardware designed using the ATPG procedure described here with previous methods. Results are shown for all of the benchmark circuits for which previously published results exist. A comparison was made for two different bit-fixing hardware encoding schemes. The first bit-fixing hardware encoding scheme is the one described in [Pomeranz 93a] which uses "3-gate modules" for implementing the bit-fixing. Table 1 shows the results published in [Pomeranz 93a] compared with the results obtained using the proposed ATPG procedure.

The parameter *N* is the number of patterns applied in the pseudo-random test and also the number of patterns applied for each bit-fixing group. Three different values of *N* were used for each circuit. In each case, the number of bit-fixing groups is shown followed by the number of 3-gate modules required to implement the bit-fixing. The total test length is shown which is equal to the number of bit-fixing groups plus one (for the pseudo-random test) times the value of *N*. The fault coverage in all cases is 100% of detectable faults. In [Pomeranz 93a], test cubes obtained with COMPACTEST [Pomeranz 93b] (which uses heuristics to minimize the total number of test cubes) were used for selecting the bit positions to fix. For the proposed method, test cubes obtained using the Correlating ATPG procedure were used for selecting the bit positions to fix. As can be seen, a dramatic reduction in overhead is achieved by simply using different test cubes for selecting the bit positions to fix.

Table 1. Comparison with Published Results in [Pomeranz 93a]

Circuit		Parameter	[Pomeranz 93a]			3-Gate Mod. With Proposed ATPG		
Name	Num PI	N	Num Bit-Fixing Groups	Num 3-Gate Modules	Total Test Length	Num Bit-Fixing Groups	Num 3-Gate Modules	Total Test Length
C880	60	1024	2	24	3072	1	6	2048
		2048	1	13	8192	1	1	4096
		4096	2	20	12288	1	1	8192
C2670	233	1024	18	233	19456	5	71	6124
		2048	14	233	30675	4	65	10240
		4096	15	233	65536	4	63	20480
C3540	50	1024	3	6	4096	3	16	4096
		2048	2	4	6144	2	3	6144
		4096	2	3	12288	1	4	8192
C7552	207	1024	46	207	48128	7	165	8196
		2048	35	207	73728	6	153	14336
		4096	36	207	151552	5	144	24576

Table 2. Comparison with Published Results in [Touba 96].

Circuit		Bit-Fixing Sequence Generator in [Touba 96]			Bit-Fixing Sequence Generator using Proposed ATPG		
Circuit Name	Scan Size	LFSR Size	Seq. ID. Reg Size	Literal Count	LFSR Size	Seq. ID. Reg Size	Literal Count
s420	34	14	3	70	14	1	36
		10	4	70	10	3	59
s641	54	14	4	87	14	2	80
		9	6	109	9	5	98
s838	66	14	7	176	14	5	164
		12	7	199	12	6	183
s1196	32	14	4	71	14	4	69
		12	8	102	12	7	97
s5378	214	14	4	174	14	4	164
		12	9	367	12	8	332
C2670	233	16	5	334	16	4	313
		10	12	427	10	8	385
C7552	207	36	8	782	36	7	753
		17	13	828	17	11	806

A comparison was also made for the bit-fixing hardware encoding scheme described in [Touba 96] which selectively performs bit-fixing on a serial sequence of bits that is shifted into a scan chain. Table 2 shows the results published in [Touba 96] compared with the results obtained using the proposed Correlating ATPG procedure. For each circuit, the number of stages in the LFSR used to generate the pseudo-random patterns is shown followed by the size of the sequence ID register (which is equal to the number of bit-fixing groups) and the literal count of the multilevel bit-fixing sequence generation logic. In all cases, the total test length is 10,000 patterns and the fault coverage is 100% of detectable faults. The results published in [Touba 96] were obtained using a conventional ATPG procedure. By simply selecting a different set of test cubes to embed using the proposed Correlating ATPG procedure, a significant reduction in hardware overhead can be achieved. Note that both the amount of combinational logic (i.e., literal count) and more significantly the number of flip-flops required (i.e., Sequence ID Register Size) are reduced.

5. Conclusions

Traditional approaches for designing mixed-mode BIST hardware use structural information only indirectly in the form of identifying bit correlations in a given set of test cubes for the r.p.r. faults. The ATPG procedure described in this paper directly analyzes the circuit structure to find test cubes that lead to more efficient mixed-mode BIST hardware. The ATPG procedure described here can be used with any bit-fixing hardware encoding scheme to reduce BIST overhead. Results for two different bit-fixing schemes indicate that dramatic reductions in overhead can be achieved with the proposed ATPG procedure.

While the procedure in this paper is described for bit-fixing schemes in particular, it can easily be adapted for other test encoding schemes that benefit from correlated test cubes.

Acknowledgements

This material is based on work supported in part by the Defense Advanced Research Projects Agency (DARPA) under Contract No. DABT63-94-C-0045, in part by the National Science Foundation under Grant No. MIP-9702236, and in part by the Texas Advanced Research Program under Grant No. 1997-003658-369.

References

[Aboulhamid 83] Aboulhamid, M.E., and E. Cerny, “A Class of Test Generators for Built-In Testing,” *IEEE Transactions on Computers* , Vol. C-32, No. 10, pp. 957-959, Oct. 1983.

[AlShaibi 94] AlShaibi, M.F., and C.R. Kime, “Fixed-Biased Pseudorandom Built-In Self-Test for Random Pattern Resistant Circuits,” *Proc. of International Test Conference*, pp. 929-938, 1994.

[AlShaibi 96] AlShaibi, M.F., and C.R. Kime, “MFBIST: A BIST Method for Random Pattern Resistant Circuits,” *Proc. of International Test Conf.*, pp. 176-185, 1996.

[Agarwal 81] Agarwal, V.K., and E. Cerny, “Store and Generate Built-In Testing Approach,” *Proc. of FTCS-11*, pp. 35-40, 1981.

[Akers 89] Akers, S.B., and W. Jansz, “Test Set Embedding in a Built-In Self-Test Environment,” *Proc. of International Test Conference*, pp. 257-263, 1989.

[Chatterjee 95] Chatterjee, M., and D.K. Pradhan, “A Novel Pattern Generator for Near-Perfect Fault Coverage,” *Proc. of VLSI Test Symposium*, pp. 417-425, 1995.

[Daehn 81] Daehn, W., and J. Muncha, “Hardware Test Pattern Generation for Built-In Testing,” *Proc. of International Test Conference*, pp. 110-113, 1981.

[Dandapani 84] Dandapani, R., J. Patel, and J. Abraham, “Design of Test Pattern Generators for Built-In Test,” *Proc. of International Test Conf.*, pp. 315-319, 1984.

[Dufaza 91] Dufaza, C., and G. Cambon, “LFSR based Deterministic and Pseudo-Random Test Pattern Generator Structures,” *Proc. of European Test Conference*, pp. 27-34, 1991.

[Dufaza 93] Dufaza, C., C. Chevalier, and L.F.C. Lew Yan Voon, “LFSROM: A Hardware Test Pattern Generator for Deterministic ISCAS85 Test Sets,” *Proc. of Asian Test Symposium*, pp. 160-165, 1993.

[Dufaza 95] Dufaza, C., H. Viallon, and C. Chevalier, “BIST Hardware Generator for Mixed Testing Scheme,” *Proc. of European Design & Test Conf.*, pp. 424-430, 1995.

[Edirisooriya 92] Edirisooriya, G., and J.P. Robinson, “Design of Low Cost ROM Based Test Generators,” *Proc. of VLSI Test Symposium*, pp. 61-66, 1992.

[Eichelberger 83] Eichelberger, E.B., and E. Lindbloom, “Random-Pattern Coverage Enhancement and Diagnosis for LSSD Logic Self-Test,” *IBM Journal of Research and Dev.*, Vol. 27, No. 3, pp. 265-272, May 1983.

[Goel 79] Goel, P., and B.C. Rosales, “Test Generation and Dynamic Compaction of Tests,” *Proc. of International Test Conference*, pp. 189-192, 1979.

[Goel 81] Goel, P., “An Implicit Enumeration Algorithm to Generate Tests for Combinational Logic Circuits,” *IEEE Transactions on Computers*, Vol. C-30, No. 3, pp. 215-222, Mar. 1981.

[Hellebrand 92] Hellebrand, S., S. Tarnick, J. Rajski, and B. Courtois, “Generation of Vector Patterns Through Reseeding of Multiple-Polynomial Linear Feedback Shift Registers,” *Proc. of International Test Conference*, pp. 120-129, 1992.

[Hellebrand 95a] Hellebrand, S., J. Rajski, S. Tarnick, S. Venkataraman and B. Courtois, ”Built-In Test for Circuits with Scan Based on Reseeding of Multiple-Polynomial Linear Feedback Shift Registers,” *IEEE Trans. Comput.*, Vol. 44, No. 2, pp. 223-233, Feb. 1995.

[Hellebrand 95b] Hellebrand, S., B. Reeb, S. Tarnick, and H.-J. Wunderlich, ”Pattern Generation for a Deterministic BIST Scheme,” *Proc. of International Conference on Computer-Aided Design (ICCAD)*, 1995.

[Kajihara 93] Kajihara, S., I. Pomeranz, K. Kinoshita, S.M. Reddy, “Cost-Effective Generation of Minimal Test Sets for Stuck-at Faults in Combinational Logic Circuits,” *Proc. of the 30th Design Automation Conference*, pp. 102-106, 1993.

[Kangaris 96a] Kangaris, D., S. Tragoudas, A. Majumdar, “Deterministic Test Set Reproduction by a Counter,” *Proc. of Europ. Design & Test Conf.*, pp. 37-41, 1996.

[Kangaris 96b] Kangaris, D., and S. Tragoudas, “Generating Deterministic Unordered Test Patterns with Counter,” *Proc. of VLSI Test Symposium*, pp. 374-379, 1996.

[Kiefer 97] Kiefer, G., and H.-J. Wunderlich, “Using BIST Control for Pattern Generation,” *Proc. of International Test Conference*, 1997.

[Koenemann 91] Koenemann, B., “LFSR-Coded Test Patterns for Scan Designs,” *Proc. of European Test Conference*, pp. 237-242, 1991.

[Pateras 91] Pateras, S., and J. Rajski, “Cube-Contained Random Patterns and Their Application to Complete Testing of Synthesized Multi-level Circuits,” *Proc. of International Test Conference*, pp. 473-482, 1991.

[Pomeranz 93a] Pomeranz, I., and S.M. Reddy, “3-Weight Pseudo-Random Test Generation Based on a Deterministic Test Set for Combinational and Sequential Circuits,” *IEEE Trans. on Computer-Aided Design*, Vol. 12, No. 7, pp. 1050-1058, Jul. 1993.

[Pomeranz 93b] Pomeranz, I., L.N. Reddy, and S.M. Reddy, “COMPACTEST: A Method to Generate Compact Test Sets for Combinational Circuits,” *IEEE Transactions on Computer-Aided Design*, Vol. 12, No. 7, pp. 1040-1049, Jul. 1993.

[Reeb 96] Reeb, B., H.-J. Wunderlich, “Deterministic Pattern Generation for Weighted Random Pattern Testing,” *Proc. of European Design & Test Conference*, pp. 30-36, 1996.

[Touba 95a] Touba, N.A., and E.J. McCluskey, “Transformed Pseudo-Random Patterns for BIST,” *Proc. of VLSI Test Symposium*, pp. 410-416, 1995.

[Touba 95b] Touba, N.A., and E.J. McCluskey, “Synthesis of Mapping Logic for Generating Transformed Pseudo-Random Patterns for BIST,” *Proc. of International Test Conference*, pp. 674-682, 1995.

[Touba 96] Touba, N.A., and E.J. McCluskey, “Altering a Pseudo-Random Bit Sequence for Scan-Based BIST,” *Proc. of International Test Conf.*, pp. 167-175, 1996.

[Tromp 91] Tromp, G., “Minimal Test Sets for Combinational Circuits,” *Proc. of International Test Conference*, pp. 204-209, 1991.

[Venkataraman 93] Venkataraman, S., J. Rajski, S. Hellebrand, and S. Tarnick, “An Efficient BIST Scheme Based on Reseeding of Multiple Polynomial Linear Feedback Shift Registers,” *Proc. of Int. Conf. on Computer-Aided Design (ICCAD)*, pp. 572-577, 1993.

[Wunderlich 96] Wunderlich, H.-J., and G. Kiefer, “Bit-Flipping BIST,” *Proc. of International Conference on Computer-Aided Design (ICCAD)*, 1996.

[Zacharia 95] Zacharia, N., J. Rajski, and J. Tyszer, ”Decompression of Test Data Using Variable-Length Seed LFSRs,” *Proc. VLSI Test Sym.*, pp. 426-433, 1995.

[Zacharia 96] Zacharia, N., J. Rajski, J. Tyszer, and J.A. Waicukauski, “Two-Dimensional Test Data Decompressor for Multiple Scan Designs,” *Proc. of International Test Conference*, pp. 186-194, 1996.