# A Novel Reseeding Technique for Accumulator-based Test Pattern Generation

X. Kavousianos
Computer Technology Institute
Patras,
Greece
kabousia@ceid.upatras.gr

D. Bakalis
Dept of Comp. Engineering &
Informatics, University of Patras,
Greece
bakalis@cti.gr

D. Nikolos
Dept of Comp. Engineering &
Informatics, University of Patras,
Greece
nikolosd@cti.gr

## ABSTRACT

In this paper we present a novel reseeding technique for accumulator-based Test Pattern Generation suitable for circuits with hard-to-detect faults. Storing the seeds is not necessary since the seeds are generated on-the-fly by inverting the logic value of some of the bits of the accumulator's register. The proposed technique achieves complete fault coverage with shorter test sequences and requires less hardware for its implementation than the corresponding already-known techniques. Furthermore, our technique does not affect the system performance since the logic required for its implementation is not inserted in the critical path.

## 1. INTRODUCTION

The advances of semiconductor process technology force IC companies to move towards very deep submicron integrated circuit technology for taking advantage of the increased functionality, higher speeds and decreased costs that it offers. Very deep submicron ICs although capable of offering increased speeds and integration of millions of gates require new and effective test methodologies in order to be tested adequately and cost effectively.

Built-In Self-Test (BIST) is becoming a very attractive Design For Testability (DFT) strategy since it reduces external testing requirements. BIST tries to incorporate in the same IC the Circuit Under Test (CUT) and its tester enabling in this way the chip to test itself. Although this leads to increased implementation area, this DFT method is becoming more and more attractive since it decreases the time to market, it often leads to higher testing quality and it cuts down the cost effectively [7].

The quality of a BIST scheme depends on: (a) the Test Pattern Generator (TPG), a circuit that produces the patterns applied to the CUT, and (b) the Test Response Verifier, a circuit which captures the responses of the CUT, compacts them to one single pattern called signature and compares this against the signature of a fault-free CUT.

One way to perform test pattern generation and test response compaction is by means of linear feedback shift registers (LFSRs) and multiple-input linear feedback shift registers (MISRs) [1], [3]. In this way high fault coverage can be achieved without using expensive external test equipment. However, modifications to the registers of the circuit must be made resulting to additional hardware overhead and possible system performance degradation due to the additional multiplexers in the signal path.

The possible system performance degradation and the area required for BIST can be minimised if some of the original building blocks of the circuit are utilised to generate patterns and/or to compact test responses. General purpose computing structures as well as digital signal processing circuits' datapaths and many other circuits contain adders, subtractors and/or accumulators. The suitability of these modules for test pattern generation [4]-[6], [10]-[12], [14], [15] and test response compaction [2], [6], [8], [9], [12], [13] in test per-clock BIST schemes has been investigated. In many cases the test set length to achieve complete fault coverage can be prohibitively large. To cope with this problem, three approaches have been proposed [4], [14], [15]. The first one [14] presents a method for suitably choosing the seed and the constant value of the accumulator such that the cardinality of the test set for a set of hard-to-detect faults to be minimised. However the cardinality of the test set still remains large. The second approach [15] is suitable for easily-random testable circuits and it is based on reseeding. Forward and reverse fault simulation is used to find windows of ineffective test patterns and determine the seeds of the accumulator. Recently, a method was presented [4] that, based on Genetic Algorithms, computes the initial values for general functional modules, such as adders and multipliers, so that they are able to produce test patterns with complete fault coverage. However, in the case of circuits with many hard-to-detect faults a large number of seeds must be used. Therefore the hardware overhead can be very large considering the ROM that must be used to store the various seeds and the necessary control module. Furthermore, in order to be able to load the various seeds to the registers of the TPG, multiplexers have to be inserted. In the case that these registers are in the critical path this results in system performance degradation.


GLSVLSI 2001, West Lafayette, Indiana, USA

In this paper we present a novel reseeding technique for accumulator-based test pattern generation. The accumulator beginning from an initial seed and a constant value produces a new test pattern at each clock cycle by the "add" operation. A new seed is produced on-the-fly by inverting the logic value of some of the bits of the accumulator's register. The proposed technique achieves complete fault coverage with less hardware overhead and shorter test sequences than the accumulator-based method given in [4] for all circuits with hard-to-detect faults. Furthermore, the logic required by our technique is not inserted in the critical path, therefore it does not impact the system performance.

The remaining of the paper is organised as follows: Sections 2 and 3 present respectively the architecture and the reseeding algorithm for the proposed TPG. In Section 4 the effectiveness of the proposed technique is evaluated with experimental results and comparisons are made with the other already-known techniques. Conclusions are given in Section 5.

## 2. PROPOSED ARCHITECTURE

The accumulator is utilised as a test pattern generator for testing blocks physically connected to it. The architecture of the proposed TPG is shown in Figure 1. The accumulator consists of a *k*-bit adder and a *k*-bit register. It has been reported in [10] that accumulators with stored carry feedback perform better than the accumulators without carry feedback. For that reason we suppose that the carry out of the adder is stored in a flip-flop and is used in the next clock cycle. After being initialised, the accumulator adds the constant value, the current contents of the register and the content of the flip-flop and stores the result back to the register.

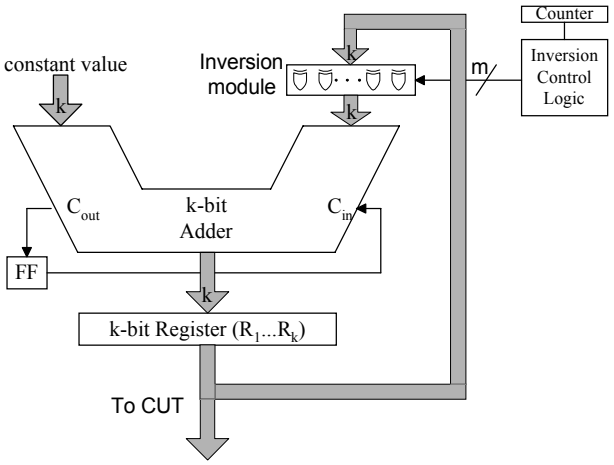



**Figure 1. Proposed TPG circuit**

In order to produce a new seed, some of the bits of the register are inverted before added with the constant value. For that purpose, we insert the "Inversion Module" between the inputs of the adder and the outputs of the register. The inversion module consists of a series of exclusive-OR (XOR) gates. As will be shown in experimental results, only a few inputs must be inverted at some time during testing mode. Therefore, *m* XORS, with $m<k$ are sufficient (see Figure 2). The counter and the inversion control logic decide which bits will be inverted and when.

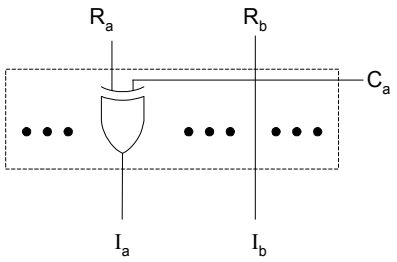



**Figure 2. Inversion module**

Let $I_i$, $1 \le i \le k$, be the input of the adder driven by cell $R_i$ of the accumulator's register R directly or through a XOR gate. $C_a$, is 1 only when during test mode $R_a$ must be inverted. During the normal mode of operation $C_a$ has the value 0. The inversion control logic is responsible for producing all control lines $C_a$. It receives the output of a counter, which counts the vectors generated by the accumulator, and sets the *m* control lines $C_a$ to either 0 or 1 depending on the number of the current vector.

For example consider that the circuit under test has 4 inputs and we have a 4-bit accumulator with initial seed 0111 and constant value 0111. The sequence of test vectors that are generated by the accumulator is given in Figure 3a. Consider that our circuit has 3 hard-to-detect faults that are tested by the vectors x011, 0001 and 1001, where x denotes don't care. Also consider that all the easy to detect faults are detected by the first two vectors of the sequence generated by the accumulator. We can see that, without reseeding, the first twelve vectors are sufficient for testing our circuit. Now suppose that we invert the value of $R_2$ line during the addition for the generation of vector 2. Then vector 0011 will be generated which covers the desirable vector x011. In the same way, the inversion of the value of lines $R_2$ and $R_4$ at the next addition, generates the vector 0001. The next vector that will be generated by the accumulator is the vector 1001. We can easily see that all the faults are now detected by the first 5 vectors produced by the reseeded accumulator. The implementation, that is, the counter, the inversion control logic and the inversion module are shown in Figure 3b. We assume that in normal mode the counter remains at the 00 state (vector 0), so $C_2=C_4=0$. It is obvious that when the accumulator is used to test more than one circuits, the inversion control logic can be easily designed in such a way so as to generate the *m* control signals taking into account all the under test circuits.

In the above example, the selection of bits $R_2$, $R_4$ to be inverted is a crucial factor affecting the hardware overhead of the proposed architecture. Therefore, in section 3 we present an efficient algorithm for selecting those register bits, which will minimise the overall hardware overhead.

## 3. RESEEDING ALGORITHM

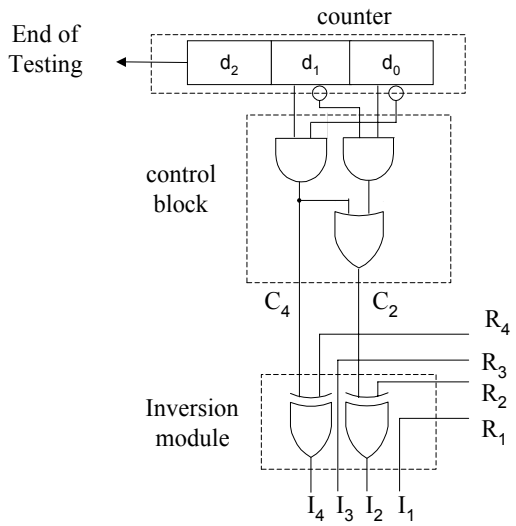
According to the proposed method, the test session consists of the parts $P_0$, $P_1$, $P_2$, $P_3$,... as shown in Figure 4. Each one of these parts consists of vectors produced by successive additions of the constant value and the current value of the register. The last vector of each part, except $P_0$, is produced by adding the constant value and the current content of the register with some bits inverted (shaded areas in Figure 4).

Let MV be the maximum number of vectors to be generated

for the detection of the hard-to-detect faults. Given the value of MV, the main objective of the algorithm is to minimise the required hardware (inversion module and inversion control logic).

| Vector | Accumulator Sequence $R_4R_3R_2R_1$ | Accumulator Sequence $R_4R_3R_2R_1$ |
|---|---|---|
| 0 | **0 1 1 1** | **0 1 1 1** |
| 1 | **1 1 1 0** | **1 1 1 0** |
| 2 | 0 1 0 1 | Reseeding **0 0 1 1** |
| 3 | 1 1 0 1 | Resseding **0 0 0 1** |
| 4 | 0 1 0 0 | **1 0 0 1** |
| 5 | 1 1 0 0 | |
| 6 | **0 0 1 1** | |
| 7 | 1 0 1 1 | |
| 8 | 0 0 1 0 | |
| 9 | 1 0 1 0 | |
| 10 | **0 0 0 1** | |
| 11 | **1 0 0 1** | |
| 12 | 0 0 0 0 | |
| 13 | 1 0 0 0 | |
| 14 | 1 1 1 1 | |
| . | . | |
| . | . | |
| . | . | |

**(a)**




**(b)**

**Figure 3. Example circuit.**

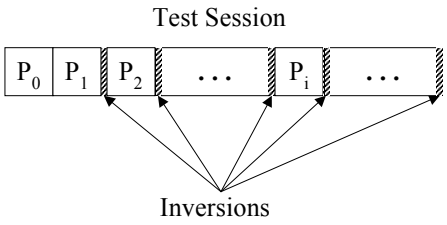



**Figure 4. Test session**

The flowchart of the algorithm is given in Figure 5. The first step is to select the constant value as proposed in [10], in order the accumulator to be able to generate a sequence of vectors with period $2^k$-1. The initial seed of the state register is set for simplicity to 00...0.

The set $P_0$ is calculated by dropping all easy-to-detect faults. Specifically, successive additions are performed until a number of *T* successive vectors fail to cover any additional faults. All vectors applied excluding the *T* last ones form set $P_0$. The remaining $f_0$ undetected faults are considered as hard-to-detect. For each hard-to-detect fault at most *Q* test vectors are extracted via Automatic Test Pattern Generation. Each test vector has as many don't care bits as possible.

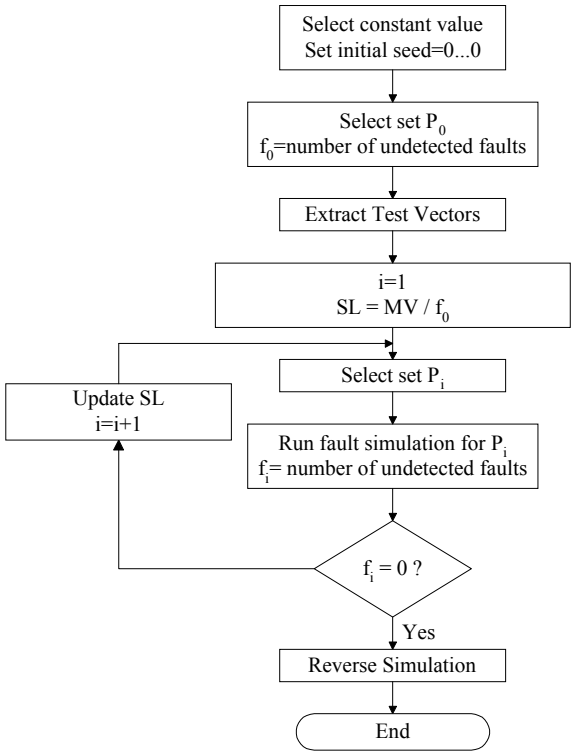



**Figure 5. Flowchart.**

So far the first $|P_0|$ vectors of the test sequence are determined. We must now cover the remaining $f_0$ faults with at most MV vectors. In the worst case $f_0$ reseedings will be necessary, and in order to produce at most MV vectors, each reseeding must be performed at every MV/$f_0$ vectors or sooner. Therefore, we introduce the factor SL which is the maximum number of successive additions before a reseeding operation.

**Table 1: The results of the proposed technique for two different values of MV.**

| Circuit | Solution 1 | | | | Solution 2 | | | |
|---|---|---|---|---|---|---|---|---|
| | *MV* | *Number of Vectors* | *Inversion bits* | *Area Overhead (gate equivalents)* | *MV* | *Number of Vectors* | *Inversion bits* | *Area Overhead (gate equivalents)* |
| ***c2670*** | 10000 | 4535 | 42 | 472 | 5000 | 2482 | 48 | 554 |
| ***c7552*** | 5000 | 4377 | 147 | 1070 | 4000 | 3646 | 145 | 985 |
| ***s420*** | 5000 | 2516 | 12 | 192 | 2000 | 1712 | 11 | 226 |
| ***s641*** | 5000 | 2112 | 12 | 95 | 3000 | 2423 | 11 | 80 |
| ***s713*** | 9000 | 5490 | 8 | 80 | 4000 | 1872 | 13 | 88 |
| ***s820*** | 10000 | 4827 | 5 | 38 | 3000 | 2787 | 7 | 72 |
| ***s838*** | 6000 | 2272 | 29 | 766 | 4000 | 3095 | 34 | 734 |
| ***s953*** | 9000 | 8412 | 3 | 47 | 4000 | 2449 | 7 | 137 |
| ***s1196*** | 10000 | 6279 | 6 | 112 | 5000 | 4415 | 6 | 167 |
| ***s1238*** | 7000 | 4082 | 7 | 137 | 5000 | 4757 | 6 | 104 |
| ***s1423*** | 3000 | 2468 | 13 | 93 | 2000 | 1511 | 21 | 117 |

Initially SL is set to $MV/f_0$ and its value is updated after each reseeding in order to take into consideration a number of vectors possibly left unused by the reseeding, as well as possible additional faults detected even if they were not predicted. In order to select the test vectors of set $P_i$ (i=1,2,...), we perform SL successive additions. Among the SL accumulator vectors, the one resulting in less hardware overhead must be selected. After each addition the content of the register R is compared with each of the ATPG extracted vectors. If it is compatible with at least one of them, then the corresponding fault (faults) has been covered without any inversions, and it is dropped. If it is not compatible with any one of them, then some inversions have to be done at the previous state of the register, in order the current state (after the addition) to be compatible with at least one test vector. Suppose that set $P_i$ consists of states R(1), R(2), …, R(SL) and let R(j-1), R(j) be the previous and current state of the register respectively. Then we have R(j)=R(j-1) + ConstantValue + Carry. Lets assume that R(j) must be made compatible with test vector *T*. Then we compare R(j) and *T* at each bit position, starting from the least significant bit to the most significant one. If at the *q* bit we have $R_q(j)=T_q$ or $T_q$ = X (don't care bit) then we move to the next bit *q+1*. If $R_q(j)\neq T_q$ then we invert the corresponding bit $R_q(j-1)$. We perform again the addition and we get the new state R(j) which is compatible with test vector *T* at the *q* least significant bits. Note that after the inversion of bit *q*, the *q-1* less significant ones remain unaffected. The above procedure is repeated until we compare all bits of *R* and *T*.

We apply the above procedure for all states R(1), R(2), …, R(SL) and for all test vectors, and we select the test vector and the state which results to the least additional hardware overhead, that is the least additional XOR gates. This state is considered as the last state of $P_i$.

The next step is to run fault simulation and drop all faults detected by $P_i$. Let $f_i$ be the number of the remaining undetected faults. Since, in general, $|P_i|<SL$, we adjust the value of SL as $(MV-\sum_{j=1}^{i}|P_j|)/f_i$. Then the algorithm continues with the next set $P_{i+1}$ until all faults are covered.

Some of the easily-testable faults that were covered with the test vectors of set $P_0$ can also be covered with some test vectors of the sets $P_1$, $P_2$,... For that reason some of the first test vectors of the test sequence can be redundant. In order to minimise the cardinality of the test set, reverse simulation [15] is performed and the initial seed is adjusted so as to exclude these redundant test vectors.

We assume that the initial seed of the register is set via initialisation of the register. When the initialisation state of the register is determined by other operations of the system (e.g. in normal mode) then the algorithm considers as initial seed this initialisation state and the reverse simulation is not performed. This is also valid when more than one circuits are to be tested by the accumulator with a single initialisation seed. Therefore, no loading of the register is needed, and thus no multiplexers are inserted for making the register accessible.

Concerning the constant value, we note that the selection is done as in [10] and depends only on the length of the accumulator, *k*. Moreover, it is the same for all TPG sessions of the accumulator. Therefore, an initialisation operation can be sufficient to set the initial seed to the register containing the constant value.

## 4. EXPERIMENTAL RESULTS

In order to validate the effectiveness of the proposed technique, we have implemented the algorithm of Section 3 in C programming language and performed several simulations. For comparison reasons, we have used the ISCAS'85 and ISCAS'89 benchmark circuits that were used in [4] and require more than 1 seed.

In Table 1 we present results for two different values of MV for each benchmark circuit. The constant value of the accumulator is selected randomly. The third and seventh columns indicate the number of vectors required for achieving complete fault coverage. The fourth and eighth columns present the number of inputs of the adder where XOR gates must be inserted (this constitutes the Inversion Logic module). Finally, in the fifth and ninth columns we present the hardware overhead required. This includes the hardware overhead of the XOR gates and the inversion control logic. The hardware overhead is measured in terms of gate equivalents assuming that 1 gate equivalent is equal to a 2-input NAND gate. We remind that MV

**Table 2: Comparison of the proposed technique with the one presented in [4]**

| Circuit | Number of Vectors | | | Area overhead | | | |
|---|---|---|---|---|---|---|---|
| | | | | Functional BIST [4] | | | Proposed technique (gate equiv.) |
| | Functional BIST [4] | Proposed technique | Reduction | Multiplexers (gate equiv.) | ROMbits | Control Logic | |
| **c2670** | 10179 | 2482 | 75.6% | 559 | 15378 | H | 554 |
| **c7552** | 4000 | 3646 | 8.9% | 497 | 26496 | H | 985 |
| **s420** | 5510 | 1712 | 68.9% | 82 | 476 | H | 226 |
| **s641** | 4475 | 2423 | 45.9% | 130 | 540 | H | 80 |
| **s713** | 9082 | 1872 | 79.4% | 130 | 540 | H | 88 |
| **s820** | 5311 | 2787 | 47.5% | 55 | 138 | H | 72 |
| **s838** | 6694 | 2272 | 66.1% | 158 | 1452 | H | 766 |
| **s953** | 7871 | 2449 | 68.9% | 108 | 270 | H | 137 |
| **s1196** | 10000 | 4415 | 55.9% | 77 | 256 | H | 167 |
| **s1238** | 7356 | 4757 | 35.3% | 77 | 256 | H | 104 |
| **s1423** | 3100 | 1511 | 51.3% | 218 | 546 | H | 117 |

denotes the maximum number of test vectors devoted for detecting the hard-to-detect faults. From Table 1 we can see that the number of the required test vectors for all faults never exceeds the value of MV.

We can also see that, in most cases, only a very small portion of the adder inputs require inversion. Furthermore, a big MV value results in more total number of vectors but in less hardware overhead whereas a small MV value results in less total number of vectors but with more hardware overhead. The exceptions in this rule (see circuits s641, s820 and s1238) are due to the random way of selecting the constant value, the reverse fault simulation and the extracted ATPG test vectors.

Let us now compare the proposed technique with the accumulator-based technique presented in [4] (see Table 2). Columns 2 and 3 present the total number of vectors required in these two techniques. Column 4 presents the reduction percentages. It is quite obvious that the proposed technique outperforms the one in [4].

The functional BIST approach [4] requires: (a) multiplexers to both input and state registers to make them accessible, (b) a ROM for storing the seeds and (c) a counter and a logic block controlling the reseeding operation. The area overhead, in equivalent gates, of the multiplexers and the number of ROM bits for storing the seeds are given in columns 5 and 6 respectively. The area of the logic block controlling the reseeding operation is expected to be comparable to the area required by the control module of the proposed technique, but it can not be estimated since we do not have enough information about the reseeding process. For that reason, in Column 7 we estimate it as H. The area of the proposed technique is given in Column 8. It includes all the necessary logic expect for the counter which is ignored in both techniques. We observe that in most cases the area overhead of our method is even less than the area overhead required only for the multiplexers in [4]. If we also consider the area for the ROM and the controlling logic of [4], then the area overhead of the proposed technique is drastically less than the area overhead of [4].

We have to remind that the reseeding applied in [4] may affect system performance while our method does not affect system performance because the additional logic is added to a non-critical path of the system.

## 5. CONCLUSIONS

In this paper we have proposed a new reseeding technique for accumulator-based Test Pattern Generation suitable for circuits with hard-to-detect faults. Reseeding takes place on-the-fly by inverting the logic value of some of the bits of the accumulator's register. With experimental results we have shown that the hardware required for the implementation of the proposed technique is lower than that required by the other accumulator-based reseeding techniques, the test set length is shorter and the system performance is not affected. Even better results are expected by: (a) making experiments with various constants and choosing the best solution, and (b) improving the reseeding algorithm targeting minimal hardware implementation cost.

## 6. REFERENCES


[1] Agrawal V., Kime Ch. and Saluja K., "A Tutorial on Built-In Self-Test, Part 1: Principles", IEEE Design and Test of Computers, pp. 73-82, March 1993.

[2] Bakalis D., Nikolos D. and Kavousianos X., "Test Response Compaction by An Accumulator Behaving as a Multiple-Input Non-Linear Feedback Shift Register", Proc. of International Test Conference, pp. 804-811, Atlantic City, NJ, USA, 2000.

[3] Bardell P., McAnney W. and Savir J., Built-In Test for VLSI: Pseudorandom Techniques, John Wiley & Sons, 1987.

[4] Chiusano S., Prinetto P. and Wunderlich H., "Non-Intrusive BIST for Systems-on-a-Chip", Proc. of International Test Conference, pp. 644-651, Atlantic City, NJ, USA, 2000.

[5] Gupta S., Rajski J. and Tyszer J., "Arithmetic Additive Generators of Pseudo-Exhaustive Test Patterns", IEEE Trans. on Computers, vol. 45, no. 8, August 1996.

[6] Mukherjee N., Kassab M., Rajski J. and Tyszer J., "Arithmetic Built-In Self-Test for High Level Synthesis", Proc. of VLSI Test Symposium, pp. 132-139, Princeton, NJ, USA, 1995.

[7] Nadeau B.-Dostie, *Design for At-Speed Test, Diagnosis and Measurement*, Kluwer Academic Publishers, 2000.

[8] Rajski J., Tyszer J., "Test Response Compaction in Accumulators with Rotate Carry Adders", IEEE Trans. on CAD, vol. 12, no.4, pp. 531-539, April 1993.

[9] Rajski J., Tyszer J., "Accumulator-Based Compaction of Test Responses", IEEE Trans. on Computers, vol.42, no.6, pp. 643-650, June 1993.

[10] Stroele A. P., "BIST Pattern Generators Using Addition and Subtraction Operations", Journal of Electronic Testing: Theory and Applications, vol. 11, pp. 69-80, 1997.

[11] Stroele A. P., "Arithmetic Pattern Generators for Built-In Self-Test", Proc. of International Conference on Computer Design, pp. 131-134, Austin, TX, USA, 1996.

[12] Stroele A. P., "Synthesis for Arithmetic Built-In Self-Test", Proc. of VLSI Test Symposium, pp. 165-170, Montreal, Canada, 2000.

[13] Stroele A. P., "Test Response Compaction Using Arithmetic Functions", Proc. of VLSI Test Symposium, pp. 380-386, Princeton, NJ, USA, 1996.

[14] Stroele A. P. and Mayer F., "Test Length Reduction for Accumulator-based Self-Test", Proc. of International Symposium on Circuits and Systems, pp. 2705-2708, New York, NY, USA, 1997.

[15] Stroele A. P. and Mayer F., "Methods to Reduce Test Application Time for Accumulator-based Self-Test", Proc. of VLSI Test Symposium, pp. 48-53, Monterey, CA, USA, 1997.