Effective Built-In Self-Test for Booth Multipliers

DIMITRIS GIZOPOULOS
ANTONIS PASCHALIS
Institute of Informatics & Telecommunications, NCSR Demokritos
YERVANT ZORIAN
LogicVision

Module generators provided by library vendors supply chip designers with optimized Booth multipliers, which are widely used as embedded cores in both general-purpose data path structures and specialized digital signal processors. Designers frequently use Booth multipliers in area- and speed-critical parts of complex ICs. Compared to standard array multipliers, Booth multipliers are faster and require less area, and their regular structure (see box on page 106) facilitates efficient implementation and testing in VLSI devices. Therefore, the Booth architecture is the basis of most embedded multipliers produced by automatic synthesis tools.1

Testing these deeply embedded multipliers requires an effective BIST scheme that can be easily synthesized along with the multiplier by the module generator. We have developed a BIST scheme for Booth multipliers that fulfills this requirement.

The problem
Achieving high controllability and observability by using scan techniques, engineers can test Booth multipliers efficiently with external testers if the multipliers have either linear or C testability. Linear testability means the number of test vectors increases linearly with the size of the multiplier operands. C testability means the number of test vectors is constant irrespective of multiplier operand size. Several design-for-testability (DFT) approaches for making Booth multipliers linear- or C-testable have appeared in the literature. Two of these approaches assume specific implementations of the multiplier cells, using a specific silicon compiler.2,3 Another approach is independent of specific multiplier cell implementations.4 All three approaches use DFT modifications to make the multiplier C-testable and provide a small test set regardless of operand lengths.

Externally testing Booth multipliers using scan techniques is not trivial at speed, a demand in today’s high-speed ICs. Furthermore, scan design is not always cost-effective in high-complexity designs with many deeply embedded cores. In such cases, built-in self-test (BIST), a method that puts the tester on the chip, provides an excellent solution.

The low controllability and observability of multiplier modules deeply embedded in complex ICs impose serious testability problems.5 For embedded Booth multipliers, as well as other embedded cores such as RAMs,6 ROMs,7 and FIFOs,8 an efficient BIST scheme is the best solution. It permits at-speed testing, provides very high fault coverage, and drives down testing costs for the overall IC.

Adapting the previously mentioned DFT approaches to BIST schemes is very expensive in terms of hardware implementation because of the irregularity of input test sets and their output responses. Therefore, a BIST scheme aimed specifically at relieving the difficulties of testing embedded multiplier modules is necessary. An effective BIST scheme for Booth multipliers must satisfy the following requirements:

- To avoid performance degradation in carefully optimized designs, the scheme must not apply DFT modifications to the multipliers.
- To avoid an increase in test vector length, the scheme must not use DFT modifications.
- To provide high fault coverage, the scheme must provide a test set that is independent of the operand size.
- To be cost-effective, the scheme must be implemented in a way that minimizes the area overhead.

Booth multipliers, widely used as embedded cores in general-purpose data path structures and specialized digital signal processors, pose serious testability problems. This generic BIST scheme does not require DFT modifications in the multiplier structure, guarantees fault coverage higher than 99%, and can be adopted by any module generator.
Booth multiplier operation

Consider the multiplication of numbers X and Y, in two's-complement representation, with sizes \( N_x \) and \( N_y \) respectively. Standard (nonrecoded) \( N_x \times N_y \)-array multipliers calculate one \( N_y \)-bit-wide partial product for each of the \( N_x \) bits of multiplier operand \( X \). An array of 1-bit adders (half adders and full adders) implements the addition of these \( N_x \) partial products. Modified Booth multipliers reduce the number of partial products that must be added together by recoding one of the two operands. The multiplier treats groups of \( k \) bits (\( k \geq 2 \)) of recoded operand \( X \) together to produce one (instead of \( k \)) partial product. The total number of partial products thus decreases from \( N_x \) to \( \left\lceil \frac{N_x}{k} \right\rceil \), and multiplication significantly accelerates.

The Booth multiplier with 2-bit recoding (\( k = 2 \)) is the architecture used in practice. In this case, recoded operand \( X \) is divided into groups of 2 bits (\( X_{2j}, X_{2j+1}, j = 0, 1, ..., \left\lceil \frac{N_x}{2} \right\rceil - 1 \)). Figure A shows an example of the overall architecture of an \( 8 \times 4 \)-bit multiplier. The first part of the multiplier (Figure A1) calculates the partial products, and the second (Figure A2) sums them. The multiplier consists of three types of functional cells: \( r \) (recoding) cells, \( pp \) (partial-product) cells, and adders.

The \( r \) cells perform recoding. They receive a group of 2 bits (\( X_{2j}, X_{2j+1}, j = 0, 1, ..., \left\lceil \frac{N_x}{2} \right\rceil - 1 \)) of recoded operand \( X \), along with the most significant bit of the previous group (\( X_{2j-1} \)). The \( r \) cells produce a set of recoding signals (\( \text{Sign}_j, \text{One}_j, \text{Two}_j \)) representing the operation that must be performed over nonrecoded operand \( Y (\{+0Y, -0Y, +1Y, -1Y, +2Y, -2Y\}) \) at row \( j \), as shown in Table A. The \( X_{2j-1} \) input for the \( r \) cell of row 0 (bottom row) is always 0, so this cell is reduced to a simpler two-input cell instead of a three-input cell, as shown in Figure A1. The 2-bit recoding scheme (\( k = 2 \)) produces all partial products with trivial and therefore fast operations such as inversion (to generate \(-Y\)) and shifting (to generate \(2Y\)). This is

![Figure A. Modified Booth multiplier: \( r \) cells and \( pp \) cells (1); adder array (2). \( F \): full; \( H \): half](image-url)
the reason that in practice we prefer 2-bit recoding over larger schemes such as 3-bit recoding, which requires nontrivial operations such as the production of $3Y$.

The $pp$ cells calculate the partial products that must be added together to form the final product. The $pp$ cell at row $j$ ($j = 0, 1, \ldots, \lceil N/2 \rceil - 1$) and column $i$ ($i = 0, 1, \ldots, N_j$) receives the three recoded signals from the corresponding recoding cell ($r$ cell) of row $j$ ($Sign_j$, $One_j$, $Two_j$), along with a pair of bits of the nonrecoded $Y$ operands ($Y_i$, $Y_{i-1}$) to calculate the partial product bit at this position ($pp_{j,i}$). The five-input, single-output combinational function realized by $pp$ cells is

$$pp_{j,i} = One_j \left( Sign_j X_i + Sign_j X_{i-1} \right) + Two_j \left( Sign_j X_{i-1} + Sign_j X_{i-2} \right)$$

The $pp$ cells of columns 0 and $N_j$ receive only 1 bit of the $Y$ operand ($Y_0$ and $Y_{N_j-1}$ respectively), so they can be reduced to simpler four-input cells, instead of five-input cells.

The $add$ cells (full and half) add up the partial products produced by the $pp$ cells. The $add$ cells in the array connect in a carry-propagate fashion. The $add$ cells in the bottom row are half $add$ers; the remaining are full $add$ers.

Since the Booth algorithm uses numbers in two’s-complement representation, the multiplier must perform a sign extension for the correct addition of the partial products. In our case, the multiplier performs the sign extension by adding a constant value to the $add$ers producing the last $N_y$ bits of the product. This action adds +1 at the rightmost half $add$er. It also connects the sum outputs of the full $add$ers at column $N_j$ to an input of the full $add$ers at their right and inverts them to give the corresponding product bit (see Figure A2).

The BIST architecture proposed in this article is applicable not only to the Booth multiplier structure presented here. We can apply it with equal efficiency to Booth multiplier architectures using other recoding schemes (for example, four recoding signals from the $r$ cells instead of three) or using a carry-save instead of a carry-propagate $add$er array. It is also applicable to nonrecoded standard-array multipliers with either carry-save or carry-propagate architecture.

### Table A. The $r$-cell truth table.

<table>
<thead>
<tr>
<th>$X_{2i+1}$</th>
<th>$X_{2i}$</th>
<th>$X_{2i-1}$</th>
<th>$Sign_j$, $One_j$, $Two_j$</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0, 1, 1</td>
<td>+0$Y$</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0, 1, 0</td>
<td>+1$Y$</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0, 1, 0</td>
<td>+1$Y$</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0, 0, 1</td>
<td>+2$Y$</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1, 0, 1</td>
<td>-2$Y$</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1, 1, 0</td>
<td>-1$Y$</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1, 0, 0</td>
<td>-1$Y$</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0, 0, 0</td>
<td>0$Y$</td>
</tr>
</tbody>
</table>
proach, our BIST scheme is superior in that it guarantees very high fault coverage with a small test set and requires simple BIST hardware. Since our scheme uses only regular blocks (counter, multiplexers, and accumulator), greatly simplifying the BIST synthesis process, it is easily adaptable to any module generator. The scheme does not require DFT modifications in the multiplier structure.

Test strategy and fault model

Our test strategy is based on pseudoexhaustive testing at the cell level—that is, applying all possible input combinations to all the multiplier’s cells. We assume that only one cell can be faulty at a time and that only combinational faults can occur. Adopting this fault model, we cover all detectable combinational faults (faults that can change the single faulty cell’s function to any other combinational function that can impact the multiplier’s correct operation). The set of faults included in the fault model is a superset of the set of all single stuck-at faults. It also includes any other type of single or multiple fault that can appear in the single faulty cell and change its function to a different combinational function.

Each type of cell (see box for description of cell types) requires the following sets of input combinations:

- **r cells.** An exhaustive r-cell test requires the eight different input combinations of its three inputs \(X_{i-1}, X_p, X_{i+1}\). The r cell of row 0, which always receives \(X_{i-1} = 0\), is reduced to a simpler, nonredundant two-input cell. Its exhaustive testing requires all four combinations of its two inputs.

- **pp cells.** Each pp cell receives from the corresponding r cell a set of three recoded signals that represent a recoded digit (+0, –0, +1, –1, +2, –2) and a pair of bits of the nonrecoded operand \(Y_i\). An exhaustive pp-cell test requires all combinations of each recoded digit with each of the four different combinations of \(Y_i\). The pp cells of column 0 receive only bit \(Y_0\), and the pp cells of column \(N_i\) receive only bit \(Y_{N_i-1}\). An exhaustive test of these pp cells requires all combinations of each recoded digit with each of the two binary values of input \(Y_0\) (respectively \(Y_{N_i-1}\)).

- **adders.** An exhaustive test for a full adder requires the eight input combinations of its three inputs. An exhaustive test of a half adder requires the four input combinations of its two inputs.

This fault model makes our test strategy independent of specific cell implementations. We can use any internal realization of the multiplier cells. Our strategy is valid, and every pp cell will receive all required combinations, even if the recoded digits produced by the recoding cells are not represented with three signals (Sign, One, Two) but with a different encoding scheme. (A different encoding scheme, for example, might use four signals representing the recoded digits +1, –1, +2, –2, and 0 to all four signals meaning recoded digit +0.) Thus, the proposed methodology detects all detectable combinational cell faults.

Proposed BIST scheme

Designers usually use optimized layouts of multiplier designs produced by module generators. DFT modifications in a multiplier structure may add extra hardware overhead and lead to performance degradation. Our BIST scheme avoids such modifications by adding a test pattern generator and an output data evaluator on the periphery of the multiplier, as shown in Figure 1. The figure also shows the input data registers for the two operands \(X\) and \(Y\). The architecture uses the two sets of multiplexers at the top and left sides of the multiplier array to select between normal and BIST multiplier inputs. The propagation delay of multiplier blocks usually determines the system clock period. Therefore, to avoid affecting system performance, we can place the multiplexers before the input registers, provided that the multiplier inputs come from faster modules. As a result, the BIST architecture imposes virtually no delay overhead. The proposed BIST structure makes a complex BIST controller unnecessary.

Test pattern generation. The test pattern generation requirements of BIST differ greatly from those of external testing strategies. Conventional external testing requires the fewest possible test vectors. Because they are stored in tester memory and often applied at lower frequencies than the cir-
circuit’s operating frequency, a greater number of test vectors increases testing time and cost. In contrast, BIST test vectors are applied at the normal operating (system) speed and are produced (and not stored) on chip. Therefore, BIST test vectors must be highly regular (with repetitive patterns and correlation) so that small machines can generate them.

The proposed test pattern generator for an \(N_x \times N_y\)-bit Booth multiplier is an 8-bit counter that generates 256 values (or a maximum-length 8-bit LFSR that generates 255 values). During BIST, the multiplier repeatedly uses four counter output signals as \(X\) inputs, and the remaining four counter outputs as \(Y\) inputs, as shown in Figure 1. We denote the counter outputs used as \(X\) inputs \(X_{\text{BIST0}}, X_{\text{BIST1}}, X_{\text{BIST2}}, \) and \(X_{\text{BIST3}}\). We denote the counter outputs used as \(Y\) inputs \(Y_{\text{BIST0}}, Y_{\text{BIST1}}, Y_{\text{BIST2}}, \) and \(Y_{\text{BIST3}}\) -

During BIST, the multiplier array receives \(X = X_{\text{BIST}} \mod 4\) for every \(0 \leq i \leq N_x - 1\), and \(Y = Y_{\text{BIST}} \mod 4\) for every \(0 \leq i \leq N_y - 1\). This process is accomplished by the two sets of multiplexers in Figure 1. During application of the 256 test vectors, all the multiplier’s cells are tested exhaustively with all input combinations that can appear during normal mode, except the \(N_y - 4\) rightmost half adders of the bottom row. These receive three of their four possible input combinations. This result is valid for any length of multiplier operands. The input combinations of the three types of cells can be repeatedly applied simultaneously to multiple cells of the same type in both the horizontal and vertical dimensions.

The only exception is the half adders, which cannot receive input combination \(c_n = 1\) for any \(n\) repeatedly. If we apply this combination to one half adder, no other half adder can receive it because \(c_n = 1\) cannot be reproduced in the row. Therefore, we need a number of test patterns that increases linearly with the length of the \(Y\) operand.

Let’s examine how the test patterns produced by the 8-bit counter apply all possible input combinations to every multiplier cell except the set of half adders just mentioned.

- **\(r\) cells.** The \(r\) cell of row \(j\) \((j = 0, 1, \ldots, [N_y/2] - 1)\) receives a group of 3 successive bits of the \(X\) operand \((X_{2j-1}, X_{2j}, X_{2j+1})\) and produces the group of recoding signals \((\text{Sign}_j, \text{One}_j, \text{Two}_j)\). Since the 8-bit counter exhaustively covers groups of 4 successive bits of the \(X\) operand, it also exhaustively covers groups of 3 successive bits. Therefore, the \(r\) cells are exhaustively tested. So too is the reduced two-input \(r\) cell of row 0.

- **\(pp\) cells.** The \(pp\) cell at row \(j\) and column \(i\) \((pp_{ji})\) receives the group of recoding signals \((\text{Sign}_i, \text{One}_i, \text{Two}_i)\) from the corresponding recoding cell of row \(j\) along with 2 bits of the \(Y\) operand \((Y_j, Y_{j+1})\). For each value of the group of recoding signals, the counter provides all four combinations of signals \(Y_j, Y_{j+1}\) to the cell. Since the counter also exhaustively covers 2-bit groups. Since all possible values of \((\text{Sign}_i, \text{One}_i, \text{Two}_i)\) are produced, each \(pp\) cell is tested with respect to the adopted fault model. This is also true for the reduced four-input \(pp\) cells of columns 0 and \(N_y\).

- **adders.** All adders, except the \(N_y - 4\) half adders of the bottom row, are exhaustively tested with all eight (full adders) or four (half adders) combinations of their inputs. This is due to the repetitive nature of the 256 patterns, which apply the same input combinations to many adders at the same time.

Apart from applying every possible input combination to all the multiplier’s cells, we must be sure that all faulty cell outputs propagate toward primary outputs and thus disclose the faults. Let’s consider all cell types that can be faulty and how faulty cell outputs propagate toward primary outputs.

- **\(r\) cells.** If the single faulty cell is the \(r\) cell of row \(j\) \((j = 0, 1, \ldots, [N_y/2] - 1)\), at least one of its three outputs \((\text{Sign}_j, \text{One}_j, \text{Two}_j)\) will be faulty. For every combination of these three signals, all four different combinations of signals \(Y_j, Y_{j+1}\) are applied to the \(pp\) cells of the same row. Therefore, the fault will propagate to the single output of these cells for at least one combination of \(Y_j, Y_{j+1}\). We can easily verify this by examining the function realized by the \(pp\) cells. If \(pp_{ji} (i = 0, \ldots, N_x)\) is the least significant \(pp\) line that carries the fault, the fault propagates to primary output \(P_{2ju}\) through a chain of adders (following the sum lines), and the fault is detected. Figure A (box) verifies this propagation.

- **\(pp\) cells.** Say that the single faulty cell is a \(pp\) cell at row \(j\) \((j = 0, 1, \ldots, [N_y/2] - 1)\) and column \(i\) \((i = 0, \ldots, N_y)\). Since the single output of these cells drives an \(adder\) input, the fault propagates through a chain of adders (following the sum lines) at primary output \(P_{2ju}\) and thus is detected.

- **adders.** When the single faulty cell is one of the \(adders\) (either half or full), the fault propagates through a chain of adders, again following sum lines toward primary outputs.

**Output data compaction and fault coverage.** As mentioned earlier, we adopted accumulator-based output data compaction as proposed in Rajski and Tyszer. Because multiplier units are usually accompanied by accumulators in data path blocks, this solution requires little extra hardware for response verification. Even if we synthesize a dedicated accumulator for output data compaction, it does not require much design effort due to its inherent regularity.

Since the BIST scheme’s signature resides in the accumulator, which is a well-accessed register, it can easily be validated by comparison with the fault-free signature inside or outside the chip. Researchers have analyzed the benefits of accumulator-based compaction extensively.
We extensively simulated the proposed BIST architecture for a 16x16 Booth multiplier with respect to all single stuck-at faults, using the Cadence Design Framework fault simulation tool (the Verifault stuck-at fault simulator). We explored different compaction schemes based on accumulators and a classical MISR. We also considered the single stuck-at faults in the compactor itself. We used the Espresso minimizer to extract the gate-level implementations of the multiplier cells from their logic functions. Table 1 summarizes the fault simulation results.

We analyze the results shown in Table 1 as follows:

- The fault coverage of the proposed scheme before compaction is 99.8%. The undetected faults are in the half-adder cells that receive three of their four input combinations.
- The use of a plain accumulator without rotate-carry provides poor fault coverage—far below the 99% target.
- An accumulator with a rotate-carry adder provides much better results, above the 99% target, with aliasing of 0.6%. The faults undetected due to this aliasing are ppcell faults.
- The new accumulator-based compaction scheme provides a 99.7% fault coverage, with only 0.1% aliasing. This increase in fault coverage indicates that the new compaction scheme is promising and requires further study.
- A classical MISR achieves a zero aliasing solution. The negligible difference in fault coverage between this technique and the multiple rotate-carry scheme proves the latter’s effectiveness. The difference is even smaller for larger multipliers (32 or 64 bits).

The accumulator-based compaction scheme uses multiple rotate-carry loops, as shown in Figure 2 for a 32-bit accumulator. The accumulator consists of four 8-bit segments. Each segment’s carry-out signal is fed into its carry-in input and XORed with the previous segment’s carry-out signal. Memory elements denoted F in the figure control the feedback loops. During normal operation, the memory elements are set to 0.

**Table 1. Simulation results for the 16 x 16 Booth multiplier.**

<table>
<thead>
<tr>
<th>Compaction scheme</th>
<th>Fault coverage (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Before compaction</td>
<td>99.8</td>
</tr>
<tr>
<td>Plain accumulator (no rotate-carry)</td>
<td>95.5</td>
</tr>
<tr>
<td>Single rotate-carry</td>
<td>99.2</td>
</tr>
<tr>
<td>Multiple rotate-carrys (Figure 2)</td>
<td>99.7</td>
</tr>
<tr>
<td>MISR</td>
<td>99.8</td>
</tr>
</tbody>
</table>

**Table 2. Hardware overhead (%).**

<table>
<thead>
<tr>
<th>Multiplier size (bits)</th>
<th>Single rotate</th>
<th>Multiple rotate</th>
<th>No adder Single rotate</th>
<th>Multiple rotate</th>
</tr>
</thead>
<tbody>
<tr>
<td>16 x 16</td>
<td>4.71</td>
<td>5.95</td>
<td>13.34</td>
<td>14.68</td>
</tr>
<tr>
<td>32 x 32</td>
<td>2.02</td>
<td>2.73</td>
<td>6.71</td>
<td>7.46</td>
</tr>
<tr>
<td>64 x 64</td>
<td>0.91</td>
<td>1.29</td>
<td>3.37</td>
<td>3.77</td>
</tr>
</tbody>
</table>

**Figure 2. Multiple rotate-carry scheme. F: feedback memory elements**

Finally, we provide an estimate in gate equivalents of the hardware cost imposed by our BIST scheme. We assume that a full adder equals 10 gates, a half adder equals 5 gates, an r cell equals 8 gates, a ppcell equals 14 gates, a flip-flop equals 10 gates, a multiplexer equals 3 gates, and a 2-input NAND gate equals 1 gate.

The multiplier’s original design consists of the following: $N/2$ r cells, $N/2 \times (N + 1)$ pp cells, $(N/2 - 1) \times (N + 1)$ full adders, $N + 1$ half adders, and $2 \times (N + N_r)$ flip-flops for the three registers. If an accumulator is already part of the circuit in the original hardware, we add $N_r + N$ full adders, assuming that the accumulator’s adder is in ripple-carry form. For the original design (assuming that the accumulator already exists), the total is $30N_r + 25N + 3[2N/2] + 24[3N/2] - 5$ gate equivalents.

The accumulator-based compaction scheme uses multiple rotate-carry loops, as shown in Figure 2 for a 32-bit accumulator. The accumulator consists of four 8-bit segments. Each segment’s carry-out signal is fed into its carry-in input and XORed with the previous segment’s carry-out signal. Memory elements denoted F in the figure control the feedback loops. During normal operation, the memory elements are set to 0.

**Hardware and delay cost.** Finally, we provide an estimate in gate equivalents of the hardware cost imposed by our BIST scheme. We assume that a full adder equals 10 gates, a half adder equals 5 gates, an r cell equals 8 gates, a ppcell equals 14 gates, a flip-flop equals 10 gates, a multiplexer equals 3 gates, and a 2-input NAND gate equals 1 gate.

The multiplier’s original design consists of the following: $N/2$ r cells, $N/2 \times (N + 1)$ pp cells, $(N/2 - 1) \times (N + 1)$ full adders, $N + 1$ half adders, and $2 \times (N + N_r)$ flip-flops for the three registers. If an accumulator is already part of the circuit in the original hardware, we add $N_r + N$ full adders, assuming that the accumulator’s adder is in ripple-carry form. For the original design (assuming that the accumulator already exists), the total is $30N_r + 25N + 3[2N/2] + 24[3N/2] - 5$ gate equivalents.

The extra hardware for BIST test pattern generation consists of the 8-bit LFSR (92 gate equivalents) and $N_r + N$ multiplexers, totaling $3N + 3N_r + 92$ gate equivalents.

The extra hardware for BIST output data compaction consists of 1 flip-flop (10 gate equivalents) when a single rotate-carry is used. In addition, there are $1 + [2(N_r + N)/8]$ flip-flops and $[N_r + N]/8$ XOR gates—a total of $10 + 13[2(N_r + N)/8]$ gate equivalents. Table 2 gives the hardware overhead for 16x16, 32x32, and 64x64-bit multipliers. Obviously,
Because we apply the BIST hardware only to the multiplier’s periphery, our scheme causes no internal performance degradation. For multiple embedded multipliers in a complex IC, more than one multiplier module can share the BIST hardware. We can implement the same scheme without extra effort in pipelined Booth multipliers.

Its comprehensive cellular fault model makes the BIST scheme generic and applicable to any Booth multiplier. Thus, the scheme is suitable for use in any multiplier core generator. AT&T’s standard cell library has adopted it for macrocell generation of multipliers, providing automatic synthesis of the BIST hardware of any size along with the Booth multiplier itself. We are currently investigating the scheme’s applicability in treelike (Wallace) multipliers.

References

Dimitris Gizopoulos is a researcher at the Institute of Informatics and Telecommunications (I&I) of NCSR Demokritos, Athens, where he works on VLSI design and testing projects. Additional research interests include computer architecture and fault-tolerant computing. He serves as vice chair of the Technical Meetings Group of the IEEE Test Technology Technical Committee. Gizopoulos graduated in computer engineering and informatics from the University of Patras, Greece. He received the PhD from the Department of Informatics, University of Athens, under a research assistantship from the I&I, NCSR Demokritos. He is a member of the IEEE and the Computer Society.

Antonis Paschalis is the head of the Digital Design and Testing Group and the project leader of various R&D projects at the Institute of Telecommunications and Informatics of NCSR Demokritos. He is a member of the editorial board of the Journal of Electronic Testing: Theory and Applications and has participated in several conference organizing and program committees and in the European Test Technology Technical Committee Support Group. Paschalis received the BSc in physics and the MSc and PhD in computer science, all from the University of Athens. He is a member of the IEEE and the Computer Society.

Yervant Zorian is the chief technology advisor for LogicVision, Inc., in San Jose, California. He is editor-in-chief of IEEE Design & Test, after serving for four years as associate editor-in-chief. He also chairs the IEEE Computer Society Test Technology Technical Committee. Zorian received an MS degree in computer engineering from the University of Southern California and a PhD in electrical engineering from McGill University in Canada. He is a member of the IEEE and the Computer Society.

Send correspondence to Dimitris Gizopoulos, Institute of Informatics & Telecommunications, NCSR Demokritos, 153 10 Aghia Paraskevi, Athens, Greece; dgizop@iit.demokritos.gr.