Designing Self-Testable Cellular Arrays

Cheng-Wen Wu and Shuye-Kung Lu
Department of Electrical Engineering
National Tsing Hua University
Hsinchu, TAIWAN 30043

Abstract
We present design-for-testability techniques and built-in self-test structures for cellular arrays based on the M-testability condition, which results in the minimal number of tests. Our technique applies to arrays with arbitrary dimensions and various connections. A systolic array multiplier is given as an example, showing an overhead of only 4% for making it M-testable. Our method compares favorably with that based on pl-testability. It reduces drastically the testing costs for circuits realized as cellular arrays.

1. Introduction
In our previous paper \cite{1}, conditions for pseudoxhaustively testing an iterative logic array (ILA) of combinational cells with a minimum number of tests, and techniques for designing such arrays, were proposed. We call an array testable with a minimal test set—equivalent to a minimal exhaustive test set of a single cell—an M-testable array. The purpose of this paper is to present efficient built-in self-test (BIST) structures for M-testable ILAs, and design-for-testability (DFT) techniques for general ILAs based on our M-testability conditions.

Friedman \cite{2} presented sufficient conditions for testing linear ILAs with a constant number of tests (independent of the number of cells in the array), which he called C-testable. The notion of I-testable and CI-testable was defined by Sridhar and Hayes \cite{3}. They discussed the design of self-testing bit-sliced CPUs based on I-testability, whereas identical test responses are obtained from every cell in a functionally correct array, allowing verification of test responses using simple equality checkers. Test pattern generation and distribution, however, is a complicated problem in their design. Abouhamid and Cerny followed their work and proposed a BIST structure, also for 1-dimensional unilateral ILAs \cite{4}. They showed that all C-testable one-dimensional ILAs also are pl-testable, which means partitioned I-testable. It is a property yielding, in many linear-array cases, relatively simple built-in-testing structures, both for the test generator and for the response verifier. However, their technique is not appropriate for 2-D arrays.

In this paper we propose BIST structures based on the M-testability conditions. M-testability is stronger than C-testability; it results in the minimal number of test patterns. We also show that its corresponding BIST structures require smaller hardware overhead as compared to those based on pl-testability: it has a much simpler and more regular test generator, and an equally compact response verifier with simpler routing. Moreover, our technique applies to ILAs with an arbitrary dimension, and even to arrays with various other connection types, e.g., hexagonal or octagonal ones.

Although it is more stringent, we can make non-M-testable arrays easily testable by using a truth-table augmentation technique. To illustrate our approach, we give a systolic array multiplier as an example, and show that an overhead of only 4% is sufficient to make it M-testable.

2. Review of M-testable Arrays
A cell is a combinational machine \((\Sigma, \Delta, f)\), where \(f: \Sigma \rightarrow \Delta, \Sigma = \{0, 1\}^l\) and \(\Delta = \{0, 1\}^0\) for \(l, 0 \in \mathbb{N}\). An ILA is an array of cells. We use the terms array and ILA interchangeably. An ILA is homogeneous when it consists of functionally identical cells, otherwise it is heterogeneous. Unless otherwise specified, arrays are assumed to be homogeneous. For homogeneous arrays, \(\Sigma = \Delta\), so \(f = 0 \equiv w\), which denotes the input or output word length. We therefore may say that an array is 2\(^{-}\)-testable when it is M-testable.

To test an array, we choose to verify the function of every cell in it. A cell may fail in an arbitrary manner, as long as it remains combinational. Verifying a cell function involves generating inputs for the cell (i.e., controlling the cell), and propagating faults from the cell (i.e., observing the cell). This model is called the cell fault model, and it has been used as the cell-level fault model by most researchers dealing with ILA testing. We also assume that there is at most one faulty cell in the array. That is, the single cell fault model is adopted. A complete or exhaustive input sequence \(\sigma\) for a cell is an input sequence consisting of all possible input combinations for the cell, i.e., \(\sigma = \sigma_1 \sigma_2 \cdots \sigma_k\), where \(\forall \sigma_i \in \Sigma, i \in \{1, 2, \cdots, k\}\). A complete output sequence \(\delta = \delta_1 \delta_2 \cdots \delta_k\) is defined analogously. A minimal complete sequence is a shortest such sequence. We may denote an output sequence as \(\delta = f(\sigma)\).

Fig. 1: A 2-dimensional \(+45^\circ\) tessellation.

THEOREM 1 \cite{1}: A \(k\)-dimensional ILA is M-testable if it has a bijective cell function, where \(k\) is an arbitrary positive integer.

A 2-D example is shown in Fig. 1. Let \(w\) denote the input word length of a cell. Then M-testability
stands for $2^n$-testability, i.e., the whole array can be pseudoexhaustively verified with only $2^n$ tests regardless of how many cells are there in the array. We may also denote that $M = 2^n$. Let $\sigma = \sigma_1 \sigma_2 \cdots \sigma_M$ be a minimal complete input sequence. Then any permutation of $\sigma$ is also a minimal complete input sequence. Let $\pi$ be the set of all $M!$ such permutations. Since $f$ is a bijection, $\pi$ also is the set of all minimal complete output sequences. Let $f(\sigma)$ denote the output sequence that corresponds to the input sequence $\sigma$. Given a complete sequence $\sigma$, define complete sequence $\sigma'$ inductively as follows: $\sigma' = \sigma$, and $\sigma' = f(\sigma')$, $i > 1$, where $\sigma' = (l_i, J_i)$. That is, given the minimal complete sequence $\sigma'$, we construct, for each cell in the array, a compatible minimal complete input sequence. The construction begins by applying $\sigma'$ to cell$_{11}$ obtaining minimal complete sequence $\sigma'' = (l_2, J_2)$, which in turn is fed to cell$_{21}$ and cell$_{22}$, obtaining minimal complete sequence $\sigma'' = (l_3, J_3)$. We then apply $\sigma''$ to cell$_{31}$, cell$_{22}$, and cell$_{13}$, obtaining minimal complete sequence $\sigma'' = (l_4, J_4)$. Reiterating this process, we construct a minimal complete input sequence for every cell in the array. All cells whose indices sum to the same number, i.e., those lie in the same 45° line, receive the same minimal complete input sequence (see Fig. 1). This pattern is called a 445° tessellation [5]. Using this pattern, any fault is propagated to some observable primary output. A fault results in an input change to the cell's horizontal and/or vertical neighboring cell. Since $f$ is a bijection, this input change is propagated to an output change. Bijectivity ensures that the change continues to ripple to some external output; the propagating paths cannot mask each other.

3. BIST Structures

We will discuss two types of ILAs: M-testable arrays and non-M-testable ones. For non-M-testable arrays, a truth-table augmentation approach is used to modify the array so that it becomes M-testable.

Based on the test pattern tessellation shown in Fig. 1, a BIST structure is given in Fig. 2, where the array is assumed to be M-testable. In the figure, extra circuit elements and wires are highlighted, and only the test pattern generation and routing mechanism is shown. The test generator is marked $G$, which is simply a binary counter that counts from 0 to $2^n - 1$. The multiplexers are marked $M$, which are controlled by a mode-selection signal that indicates whether the circuit is in test or normal mode. If it is in test mode, all multiplexers take the inputs which are highlighted. Therefore, if the generator generates a minimal complete input sequence $(l_i, J_i)$ for cell$_{11}$, all other cells subsequently receive their own minimal complete input sequences according to Thm. 1. When it is in normal mode, all multiplexers take the inputs which are drawn in thin line segments, i.e., the test paths are disabled, and the array returns to normal operation. The performance penalty is negligible; the multiplexers can be realized with pass transistors or transmission gates. For non-M-testable arrays, extra inputs and outputs to the cells are introduced. The technique will be shown later.

Fig. 3: A BIST structure with response evaluator.

To make the discussion complete, we need to consider the evaluation of the responses on chip. A BIST structure, including general response analysis modules, is shown in Fig. 3. In the figure, the modules marked $A$ are response analyzers. Due to the special characteristics of the test patterns—a minimal complete input sequence for every cell—there are novel techniques for response analysis. The first one is the check--sum method: each $A$ module in Fig. 3 is replaced by an accumulator, i.e., a binary adder. Since the output sequence is minimal complete, the accumulated output becomes

$$\sum_{i=1}^{k} \delta_i = \frac{2^w(2^w - 1)}{2} = 2^{2w-1} - 2^{w-1}.$$ 

For example, if $w = 3$ then the accumulated result is 11100. Judging from the bit positions of zeros and ones, one can determine immediately whether the output sequence is faulty or not. This approach has the advantage that fault coverage is high. All single-bit errors can be detected. The only undetected errors are those multi-bit errors corresponding to different permutations of the correct output sequence. It is rare however that a physical failure will cause such a logical effect in real world. The price to pay for the high fault coverage is extra hardware for the binary adders, and its low evaluation speed.

Another method is the use of parity checkers. A minimal complete sequence implies that the number of ones and zeros in the sequence, at each bit position, must be equal. That is, an even parity must be true for every output bit. This approach is faster and uses much less hardware than the previous one but it can not detect any output line stuck-at-0 or stuck-at-1, which can have a relatively high probability to occur.
There are alternatives in the way the outputs of the A modules are collected, and there is a tradeoff between hardware/time overhead and the degree of compression of the output data. Fig. 3 depicts just one of the alternatives. The outputs of the A modules, for example, can be sent to a shift register which performs parallel-to-serial conversion. The result is then shifted out sequentially for external go/no-go judgement. Another approach is to OR the outputs of the A modules, bit by bit, and send the 1-bit final result out, which is a true go/no-go signal. The approach completely excludes the need of external test equipments. Of course a simple, inexpensive tester can be used to replace both the hardware overhead and the degree of potential reliability problem introduced by the test distribution wires on chip. This would reduce the hardware overhead to a minimum, and eliminate the potential reliability problem introduced by that extra hardware.

![Figure 4: Parallel sorting array.](image)

**4. Applications**

To illustrate our approach, we show two practical examples: a parallel sorting array and a systolic array multiplier.

**Example 1:** A 2-D array for parallel sorting, together with its cell function, is shown in Fig. 4(a) [6]. The array can be made rectangle by placing dummy cells to fill the lower left corner. The cell function is not a bijection. For example, for the original function, the (in boldface characters) and the modified cell functions are given in Fig. 4(b). The modified function clearly is bijective. The modification is based on the following rule. If there are two domain values to their new output bit. We maintain the original cell function in table entries where the added input is said to be reflexive to the upper half and the lower half of the table is denoted \( \alpha \). If \( \alpha \) is a feasible assignment of values for the entries of the augmented truth table (as defined in Ex. 1), then \( \alpha \) (or \( \alpha \) resp.) denotes \( \alpha \) confined to the input (output resp.) variable \( \alpha \) (or \( \alpha \) resp.) assigns only the values for \( \alpha \) (or \( \alpha \) resp.). The assignment \( \alpha \) confined to the upper half of the table is denoted \( \alpha \); the assignment confined to the lower half is \( \alpha \). An assignment \( \alpha \) for an input or output variable \( \alpha \) is said to be reflexive if in the augmented truth table, the values assigned by \( \alpha \) are one-to-one corresponding to those assigned by \( \alpha \). It is complementarily reflexive, or simply complementary if the assigned values by \( \alpha \) are one-to-one corresponding to

\[
\begin{align*}
\hat{x} &= x + y \\
\hat{y} &= x \cdot y
\end{align*}
\]

The modified function is more complicated:

\[
\begin{align*}
\hat{x} &= (x + y) \cdot z + x \\
\hat{y} &= (x + y) \cdot z + x \\
\hat{z} &= (x + z) \cdot y + x \\
\end{align*}
\]

It looks like the overhead can easily reach above 75%, which of course is untolerable. The real overhead however can be made much lower than it seems by an alternative output assignment of the augmented truth table. There are many possible ways of assigning values to the expanded portion of the truth table, as long as the bijectivity condition is satisfied. Depending on the assignment of the values, simpler function may be obtained, resulting in lower hardware complexity. Let the original cell’s MOS implementation be as shown in Fig. 5(a), which has 10 transistors. We may use, for example, the output assignment as given in Fig. 5(b), which comes up with the following equations:

\[
\begin{align*}
\hat{x} &= (x + y) \cdot z + (x + y) \\
\hat{y} &= x \cdot y \cdot z + x \cdot y \\
\hat{z} &= x
\end{align*}
\]

The binary case in this example is the worst case. Real applications however are rarely binary. The numbers to be compared can be, for example, 8-bit or 16-bit ones. In the modified function, however, there will still be only one extra input bit and one extra output bit. The reason is that no 3 different inputs map to the same output in the original function.

Without loss of generality, we assume that there are no more than two identical entries in the output part of the original truth table, i.e., only one extra input bit (2) and one extra output bit (1) are added. We also assume that the truth tables are fully expanded, i.e., there are no don’t-care terms. In reality, don’t-cares can be assigned in an arbitrary way. We of course will assign them such that the number of identical output words is as small as possible (or even disappear). If \( \alpha \) is a feasible assignment of values for the entries of the augmented truth table (as defined in Ex. 1), then \( \alpha \) (or \( \alpha \) resp.) denotes \( \alpha \) confined to the input (output resp.) variable \( \alpha \) (or \( \alpha \) resp.) assigns only the values for \( \alpha \) (or \( \alpha \) resp.). The assignment \( \alpha \) confined to the upper half of the table is denoted \( \alpha \); the assignment confined to the lower half is \( \alpha \). An assignment \( \alpha \) for an input or output variable \( \alpha \) is said to be reflexive if in the augmented truth table, the values assigned by \( \alpha \) are one-to-one corresponding to those assigned by \( \alpha \). It is complementarily reflexive, or simply complementary if the assigned values by \( \alpha \) are one-to-one corresponding to

\[
\begin{align*}
\hat{x} &= x + y \\
\hat{y} &= x \cdot y
\end{align*}
\]
EXAMPLE 2: A 2-D array for pipelined carry-save multiplication is shown in Fig. 6(a) [7]. The cell functions are given as follows:

\[
\begin{align*}
\hat{a} &= a \\
\hat{b} &= b \\
\hat{s} &= s \oplus c \oplus a \cdot b \\
\hat{c} &= s \cdot c + c \cdot a \cdot b + s \cdot a \cdot b \\
\end{align*}
\]

where \( a \) and \( b \) represent multiplier and multiplicand bits, \( s \) is the summand bit, and \( c \) is the carry bit. Their respective output bits are denoted \( \hat{a}, \hat{b}, \hat{s}, \text{ and } \hat{c} \), which are propagated to the next stage. From the truth table we see that the cell function is not bijective. For example, \((a, b, s, c) = (0, 0, 0, 1)\) and \((0, 0, 1, 0)\) map to the same output \((0, 0, 1, 0)\).

We now add an extra input \( z \) (in the 3rd dimension) and a corresponding output \( \hat{z} \) to the cell, and modify the cell function as given in Fig. 6(b). The modified function clearly is bijective, hence the BIST structures as depicted in Section III can be adopted. The new cell functions are:

\[
\begin{align*}
\hat{a} &= a \\
\hat{b} &= b \\
\hat{s} &= s \oplus c \oplus a \cdot b \\
\hat{c} &= (s \cdot c + c \cdot a \cdot b + s \cdot a \cdot b \cdot z) \oplus (s \cdot c + a \cdot b + s \cdot a \cdot b \cdot z) \\
\hat{z} &= c \quad \text{(or } \hat{z} = s) \\
\end{align*}
\]

The function modification (or truth-table augmentation) is based on our desire to make the output assignment reflexive or complementary. The assignment of variables \( \hat{a}, \hat{b}, \text{ and } \hat{s} \) are reflexive, and that of \( \hat{c} \) is complementary. The extra output \( \hat{z} \) can be assigned any element in \([s, c]\). The MOS implementation has an overhead of only 4%. In the modified function, there are only one extra input bit and one extra output bit need to be added. The reason is that no 3 different inputs map to the same output in the original function.

The method also is applicable to the non-restoring array dividers [8], in which the cell function is slightly different from that of the multipliers shown above, but is still non-bijective. To make them bijective, we can apply our truth-table assignment algorithm in a similar way. With an extra transistor, the overhead also is 4%.

We compare our approach with that of Aboulhamid and Cerný [4], which is based on pl-testability. Their approach is not feasible for 2-D arrays, since response evaluation would become too complicated. If we consider just linear arrays, then our approach requires only one response analyzer at the rightmost output (see Fig. 3), which is a drastically reduction in hardware as compared to theirs. Our approach can easily be applied to arrays that do not satisfy the M-testability condition. This would be hard for arrays that are not pl-testable.

REFERENCES