JOURNAL OF ELECTRONIC TESTING: Theory and Applications 11, 197–209 (1997)
© 1997 Kluwer Academic Publishers. Manufactured in The Netherlands.

Testability Properties of Divergent Trees*

R.D. (SHAWN) BLANTON
Center for Electronic Design Automation, ECE Department, Carnegie Mellon University, Pittsburgh, PA 15213-3890
blanton@ece.cmu.edu

JOHN P. HAYES
Advanced Computer Architecture Laboratory, EECS Department, University of Michigan, Ann Arbor, MI 48109-2122
jhayes@eecs.umich.edu

Received March 20, 1996; Revised June 6, 1997

Editor: A. Paschalis

Abstract. The testability of a class of regular circuits called divergent trees is investigated under a functional fault model. Divergent trees include such practical circuits as decoders and demultiplexers. We prove that uncontrolled divergent trees are testable with a fixed number of test patterns (C-testable) if and only if the module function is surjective. Testable controlled trees are also surjective but require sensitizing vectors for error propagation. We derive the conditions for testing controlled divergent trees with a test set whose size is proportional to the number of levels p found in the tree (L-testability). By viewing a tree as overlapping arrays of various types, we also derive conditions for a controlled divergent tree to be C-testable. Typical decoders/demultiplexers are shown to only partially satisfy L- and C-testability conditions but a design modification that ensures L-testability is demonstrated.

Keywords: fault detection, fault modeling, regular circuits, interactive logic arrays, structured circuits, test generation

1. Introduction

It has long been recognized that regular logic circuits, which are constructed from identical modules that are interconnected in a uniform fashion, are easier to test than irregular circuits. For example, the circuit of Fig. 1(a) is an 8-bit ripple-carry adder composed of eight identical modules (full adders) connected as a one-dimensional array. This adder is testable for all single stuck-line faults [1] with eight test patterns. These eight tests also detect any fault that alters the function of any single module in the circuit. This testing property is true for an n-bit adder as well, that is, eight tests are required for a ripple-carry adder regardless of the number of modules in the array, a property known as C-testability [2].

The one-dimensional array adder of Fig. 1(a) is a special case of a type of regular circuit called a tree. In general, *trees* are combinational logic circuits that have at most one path between any two input/output ports. A path can contain single or multiple lines (buses). The modules or cells used to construct a tree circuit can have internal reconvergent fanout, but fanout is not allowed among the modules. Figure 1(b) illustrates an 8-bit parity tree constructed from 2-bit EXCLUSIVE-OR modules. For this tree, the size of the buses that interconnect the modules is one, and between any two

*This research was supported by the National Science Foundation under Grant No. MIP–9503463.

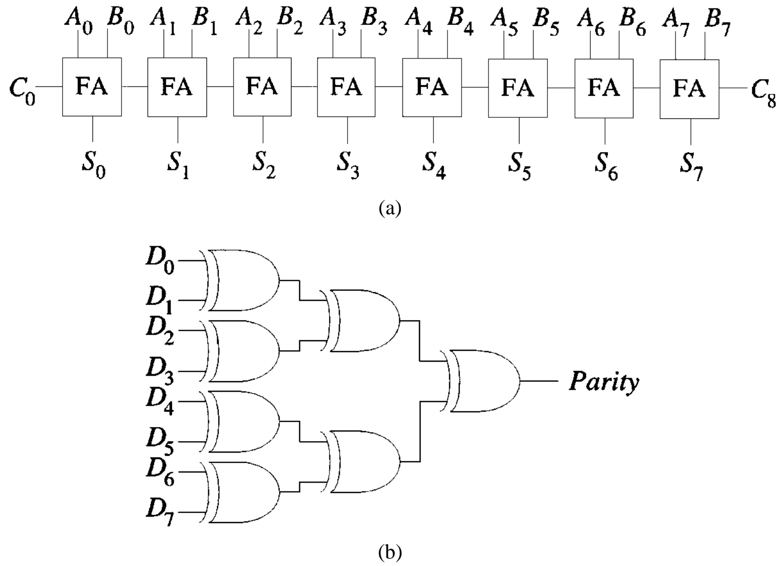


Fig. 1. Examples of regular tree circuits: (a) an 8-bit ripple-carry adder and (b) an 8-bit parity circuit.

module ports there is only one path. One can easily see how the one-dimensional array structure of the adder results if the parity tree is restricted to a single module per level.

It is useful to extend the notion of a tree to make a distinction between data and control inputs. A tree with both data and control lines is called a *controlled tree*. The 3-to-8 decoder circuit constructed from 1-to-2 decoder (Dcd) modules and the demultiplexer circuit constructed from 1-to-2 demultiplexer (Dmx) modules of Fig. 2 are examples of such a circuit. Tree circuits can be convergent, divergent, or neither, that is, the number of signal lines can increase, decrease or remain the same as one moves from the primary inputs to outputs. The parity tree is an example of a tree that converges, while the decoder and demultiplexer trees are examples of divergent trees. The ripple-carry adder is a tree that neither diverges or converges.

In general, regular circuits are easy to test because the testing requirements for the circuit's modules are identical and the regular interconnections allow tests for one module to be used on other modules. They also have other advantages. For example, because regular circuits are made by connecting identical modules in

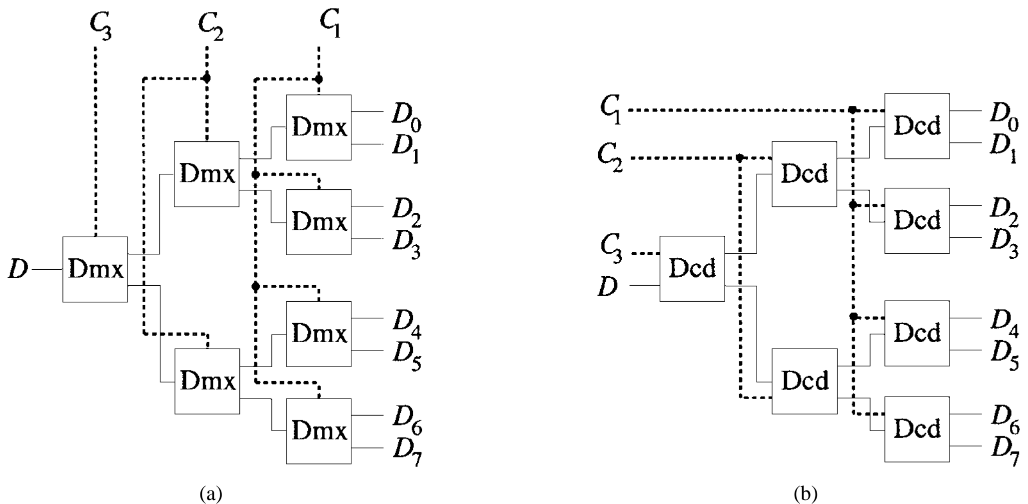


Fig. 2. Examples of divergent tree circuits: (a) a 1-to-8 demultiplexer and (b) a 1-to-8 decoder.

a regular fashion, large circuits are easy to build and have fewer design errors. The interconnection structure of regular circuits is also ideal for layout in a VLSI environment. As a result, they are used a great deal in large designs. For example, two-dimensional arrays and trees in the form of storage arrays, decoders, and multiplexers can be found in all memory designs.

The testing properties of one-dimensional arrays have been studied extensively [2–4]. In [2], the concept of C-testability was introduced, and conditions required for arrays without vertical outputs to be C-testable are presented. In [3], necessary and sufficient conditions for the C-testability of unilateral and bilateral one-dimensional arrays with vertical outputs are presented. The authors of [4] present similar testing conditions to those in [3] along with conditions for locating faulty modules within the array. The testability of two-dimensional arrays has been similarly investigated. For example, various sets of sufficient conditions for the C-testability of two-dimensional arrays are presented in [5–7]. In [8] and [9], C-testable two-dimensional array designs for multiplier and divider circuits respectively, are discussed. In [10], an automatic test pattern generator called NCUBE for two-dimensional arrays is described. The testing characteristics of systolic arrays have also been investigated in [11, 12]. Systolic arrays are one- and two-dimensional arrays and trees with buffer memory added to the circuit's modules.

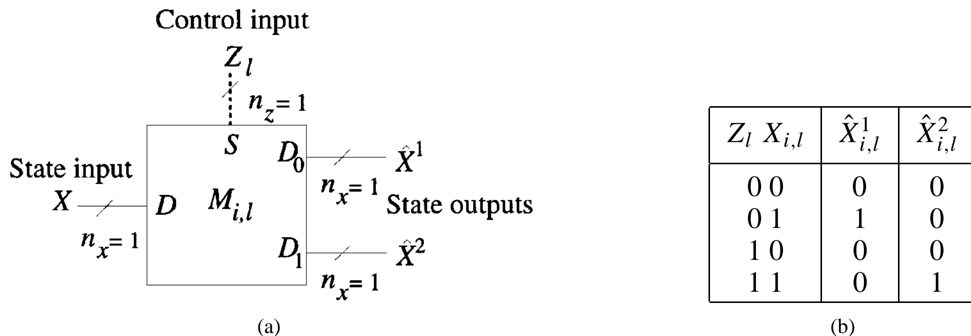
Some research has addressed the testability of convergent trees [13–18], but there appears to be no work on divergent trees. In this paper, we examine the testing properties of divergent tree circuits like the decoder and demultiplexers of Fig. 2. The rest of this paper is organized as follows. Section 2 presents our notation for describing the function and structure of divergent trees. The IP fault model adopted is then defined and illustrated. The testability of uncontrolled and controlled divergent trees is analyzed under this fault model in Section 3. Section 4 presents conditions for testing divergent trees with very few test patterns, while Section 5 describes design methods for ensuring that these conditions are satisfied. Finally, Section 6 summarizes our results.

2. Function and Structure

An n-ary p-level divergent tree $D(n, p)$ is constructed from a set of identical modules arranged in p levels. Each module $M_{i,l}$ has a single state input bus X, an optional level-control input bus Z_l, and n state output buses denoted $\hat{X}^1_{i,l}, \hat{X}^2_{i,l}, \ldots, \hat{X}^n_{i,l}$; see Fig. 3(a). (The subscripts i, l will only be used when the corresponding module name is ambiguous.) The word size for X and each $\hat{X}^j$ is n_x and the word size of Z_l is n_z. Thus, the set of values that can be assigned to X is $I_X = \{0, 1, \ldots, 2^{n_x} - 1\}$, where each n_x-bit value is denoted by a decimal number. Similarly, the set of values assignable to Z_l is $I_Z = \{0, 1, \ldots, 2^{n_z} - 1\}$. The set of possible output values that can be produced at each state output $\hat{X}^j$ is also I_X.

The n state output functions $\hat{X}^j$ for $1 \leq j \leq n$ of the divergent tree module can be described by a composite truth table containing n output columns labeled $\hat{X}^1, \hat{X}^2, \ldots, \hat{X}^n$. Each row r_k and column $\hat{X}^j$ entry specifies an input pattern $ip_k = (v_0, v_1)$ and an output value v_k. Here ip_k denotes a control value v_0 and a state input value v_1, while v_k denotes the state output value produced at $\hat{X}^j$ when ip_k is applied. Row k of the truth table thus defines the fault-free mapping denoted $ip_k \rightarrow v_k$ at state output $\hat{X}^j$ or, equivalently, $\hat{X}^j(ip_k) = v_k$.

A *controlled* n-ary p-level divergent tree $CD(n, p)$ has a control input bus Z_l for each $M_{i,l}$ in level l, where



Z_l $X_{i,l}$	$\hat{X}^1_{i,l}$	$\hat{X}^2_{i,l}$
0 0	0	0
0 1	1	0
1 0	0	0
1 1	0	1

Fig. 3. (a) A 1-to-2 demultiplexer module $M_{i,l}$ and (b) the truth table for $M_{i,l}$.

$1 \leq l \leq p$. A controlled divergent tree is *globally controlled* if all Z_l are connected together. A divergent tree is *uncontrolled* if no Z_l is present. If a divergent tree has n^{p-l} modules on every level l, then the tree is *complete*, otherwise it is *incomplete*. Here, we only consider complete trees, but our results apply to incomplete trees as well. Figure 3 illustrates our notation for a module that represents a 1-to-2 demultiplexer.

We employ a special case of the *input pattern* (IP) fault model [19]. The IP fault model assumes only a single module in the circuit can be faulty. All modules are assumed to be combinational, and the function of a faulty module is assumed to remain combinational, that is, no sequential behavior is allowed. No restriction is placed on the type or size of the module, making it possible to apply the IP model to circuits described at different design levels.

A fault f_i that affects output $\hat{X}^k$ can change the response to an input pattern ip_j from v_j to v'_j; we denote this change by $ip_j \rightarrow (v_j, v'_j)^k$ and refer to it as a single *input pattern* fault, or simply as an IP fault. The pair of distinct state values (v_k, v'_k) denoting the good and faulty output values is the *error* corresponding to the IP fault. The IP fault model allows a single module to be affected by a single IP fault or a set of such faults (a *multiple IP fault*) $F = \{f_1, f_2, \ldots, f_q\}$. It is important to note that a multiple IP fault is not restricted to a single state output $\hat{X}^k$.

The IP fault model can be customized for the testing application of concern. For example, in the special case where all single and multiple IP faults are assumed possible, the IP model becomes equivalent to the well-known *cell fault model* [20], where all modules are required to be tested pseudo-exhaustively. This special case of the IP fault model is advantageous in situations where the implementation details of the circuit modules are unknown.

The characteristics of the IP fault model are illustrated by the following example. Figure 4 shows truth tables for all 16 possible Boolean functions of an uncontrolled divergent tree module $M_{i,l}$ with $n = 2$ and $n_x = 1$. Assume both the $\hat{X}^1$ and $\hat{X}^2$ outputs of $M_{i,l}$ implement the NOT function, which is defined by column 12 of Fig. 4. The remaining columns represent all the possible faulty functions allowed by the IP fault model. For example, the faulty function of column 11 is easily described by the multiple IP fault $F = \{1 \rightarrow (0, 1)^1, 0 \rightarrow (1, 0)^2, 1 \rightarrow (0, 1)^2\}$. Notice that columns 0, 4, 8, 13, 14, and 15 are the only faulty functions possible under the single stuck-line (SSL) fault model. (Under the SSL fault model [1], a single signal line in the circuit can either become permanently fixed (stuck) at a logical 1 or 0 value.) Thus, the IP fault model is more general than the widely-used SSL model because it allows for many more faulty behaviors.

In this paper, we assume that a tree module can be affected by any single or multiple IP fault (i.e., the cell fault model). This means a faulty divergent tree module can produce an erroneous output value at one or more of its n state outputs for one or more input patterns. Since error masking is not possible in a divergent tree (due to the absence of reconvergence), the set of (single and multiple) IP faults that affect a single state output X^j for $1 \leq j \leq n$ are dominated by the set of all possible IP faults. Thus, we need only to explicitly consider IP faults that affect a single output $\hat{X}^j$ for $1 \leq j \leq n$. Moreover, the set of all multiple IP faults that affect a single state output $\hat{X}^j$ dominate the set of all single IP faults that affect the same output $\hat{X}^j$. Thus, adoption of the cell fault model only requires us to consider the set of all possible single IP faults that affect each state output.

The number of possible single IP faults for an n-output divergent tree module is $nK(W - 1)$, where $K = 2^{n_x + n_z}$ and $W = 2^{n_x}$. Note that this number of faults is much smaller than the $2^{Knn_x} - 1$ possible faulty module functions. For example, the demultiplexer module of Fig. 3 has $2^8 - 1 = 255$ different functional faults. The set of eight single IP faults for the two state outputs $\hat{X}^1$ and $\hat{X}^2$ are shown in Fig. 5.

Input	Output columns ($\hat{X}^1, \hat{X}^2$)															
X	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	00	00	00	00	01	01	01	01	10	10	10	10	**11**	11	11	11
1	00	01	10	11	00	01	10	11	00	01	10	11	**00**	01	10	11

Fig. 4. The 16 possible Boolean functions of a 1-bit uncontrolled divergent tree module.

$\hat{X}^1$ IP faults	$\hat{X}^2$ IP faults
$Z_l X \rightarrow (v_k, v'_k)^j$	$Z_l X \rightarrow (v_k, v'_k)^j$
$00 \rightarrow (0, 1)^1$	$00 \rightarrow (0, 1)^2$
$01 \rightarrow (1, 0)^1$	$01 \rightarrow (0, 1)^2$
$10 \rightarrow (0, 1)^1$	$10 \rightarrow (0, 1)^2$
$11 \rightarrow (0, 1)^1$	$11 \rightarrow (1, 0)^2$

Fig. 5. The eight IP faults for the $\hat{X}^1$ and $\hat{X}^2$ state outputs of the 1-to-2 demultiplexer module of Fig. 3.

Module D_1		
X	$\hat{X}^1$	$\hat{X}^2$
0	3	0
1	1	1
2	2	2
3	0	0

(a)

Module D_2		
X	$\hat{X}^1$	$\hat{X}^2$
0	0	2
1	1	0
2	2	3
3	3	1

(b)

Fig. 6. Truth tables for two uncontrolled divergent modules: (a) D_1 which is not surjective and (b) D_2 which is surjective.

3. Tree Testability

We now examine the testing properties of tree circuits. A tree is *testable* for all single IP faults (and, therefore testable under the cell fault model) if each module in the tree can have every possible module input pattern applied to its inputs, and any resulting error can be propagated to the tree's output. For uncontrolled trees, we derive both necessary and sufficient conditions for testability. We then extend our fault model to controlled tree circuits and derive the necessary and sufficient conditions for controlled divergent tree testability.

Uncontrolled Trees. We begin by defining the surjective property for a divergent tree module and then show how this property is both necessary and sufficient for an uncontrolled divergent tree to be testable.

Definition 1. A divergent tree module is *surjective* if the set of output values produced at each $\hat{X}^j$ is I_X, for $1 \leq j \leq n$. In other words, there is at least one occurrence of every possible state output value in each output column of the module's truth table.

Figure 6 shows two examples of binary ($n = 2$) divergent tree modules. The module of Fig. 6(a) is not surjective because state value 3 is absent from the $\hat{X}^2$ output column. The second module of Fig. 6(b) is surjective because both the $\hat{X}^1$ and $\hat{X}^2$ output columns contain all members of the set $I_X = \{0, 1, 2, 3\}$. Notice that surjectivity in an uncontrolled divergent module implies that the state output functions are information lossless. This means that the output values produced by a module uniquely determines the input value applied. This is true for the uncontrolled divergent tree module since it has the same number of signal lines for the state input X and each state output $\hat{X}^j$.

Theorem 1. *An n-ary p-level uncontrolled divergent tree $D(n, p)$ is testable if and only if $M_{i,l}$ is surjective.*

Proof: First, we show that the surjective property is necessary. Assume $D(n, p)$ is testable but $M_{i,l}$ is not surjective. Then there is a state value $v_j \in I_X$ that cannot be generated as an output value at some state output $\hat{X}^j$. As a result, v_j cannot be applied to the internal module $M_{j,1}$ connected to $\hat{X}^j_{1,1}$. Hence, $D(n, p)$ cannot be fully tested unless $M_{i,l}$ is surjective.

Now we show that the surjective property is sufficient for testability. Assume the divergent tree module is surjective. Then every state output function is surjective, so error propagation from the state input X to each state output $\hat{X}$ is guaranteed. We now use induction on the number of levels p to show that each module input pattern ip_j can be applied to all modules in $D(n, p)$ using 2^{n_x} test patterns. Obviously, all 2^{n_x} input patterns can be applied to the 1-level tree $D(n, 1)$ with 2^{n_x} tests. By the inductive hypothesis, assume 2^{n_x} tests are sufficient for a $(p - 1)$-level tree $D(n, p - 1)$. A p-level tree can be viewed as a $(p - 1)$-level tree with an additional level of modules connected to the outputs of $D(n, p - 1)$. $D(n, p)$ is testable with the same 2^{n_x} tests used for $D(n, p-1)$ because the level-1 surjective modules of $D(n, p - 1)$ produce all 2^{n_x} input patterns for the new level-1 modules of $D(n, p)$. Thus by the principle of induction, all uncontrolled divergent trees $D(n, p)$ with $p \geq 1$ can have all module input patterns applied to every module with 2^{n_x} tests. □

Example 1. Reconsider the truth tables for the divergent tree modules of Fig. 6. Module D_1 is not surjective and therefore does not satisfy the conditions of Theorem 1. Figure 7(a) shows how the state output value 3, which is missing from $\hat{X}^2$ in Fig. 6(a), prevents testability of a 3-level uncontrolled divergent

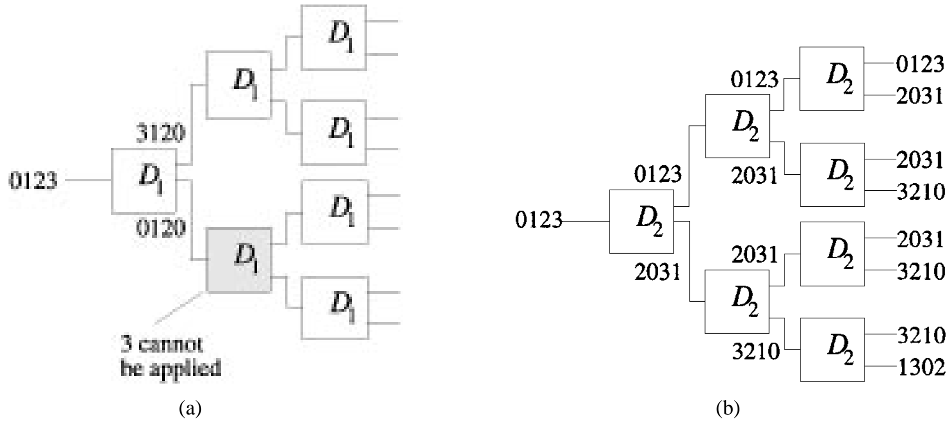


Fig. 7. Three-level uncontrolled divergent trees constructed from the modules defined in Fig. 6: (a) D_1 and (b) D_2.

tree. Module D_2 is surjective and therefore satisfies Theorem 1. Figure 7(b) shows the $2^{n_x} = 2^2 = 4$ test patterns that completely test a 3-level uncontrolled divergent tree constructed from D_2 modules. These four test patterns also test an arbitrarily large tree of p levels.

Controlled Trees. The testing of a controlled tree is complicated by faults affecting the control input buses $Z_1, Z_2, \ldots, Z_p$. We assume that each control input bus Z_l, $1 \leq l \leq p$, can be affected by a set of "stuck-bus" faults, each of which is denoted Z_l/v_z, where $v_z \in I_Z$ is some control input value. A *control input* fault Z_l/v_z causes the control input bus Z_l of all modules $M_{i,l}$ in level l to be permanently fixed at v_z. Because it affects more than one module, the fault Z_l/v_z is not covered by our fault model. The inclusion of control input faults also subsumes SSL faults on the lines making up Z_l. Thus, we define a controlled divergent tree to be completely testable if each module $M_{i,l}$ in the tree can be tested for all its IP faults, and each level l can be tested for all its control input faults of the form Z_l/v_z. A single-fault assumption is still made, in that only a single functional fault affecting a module or a single control input fault can occur.

We now turn our attention to the testability of controlled divergent trees. We define the sensitizing property for these trees and then present conditions for testability.

Definition 2. A *sensitizing* vector for an error (v_a, v_b) on state input X of a divergent tree module is a control input value $v_0 \in I_Z$ such that $\hat{X}^j(v_0, v_a) \neq \hat{X}^k(v_0, v_b)$ for some $j \neq k$. In other words, v_0 is a control value that causes at least two state outputs to produce different values when the error (v_a, v_b) is present on the state input.

Theorem 2. *An n-ary p-level controlled divergent tree $CD(n, p)$ is testable if and only if $M_{i,l}$ is surjective and there exists a sensitizing vector for every possible error (v_a, v_b).*

Here, we sketch the proof of Theorem 2. The surjective condition, similar to Theorem 1, is necessary and sufficient for applying every input pattern to every module in an arbitrarily large tree. The sensitizing vector condition is required for propagating all possible errors to an output from any module within the tree.

Figure 8 shows a truth table for a 1-to-2 decoder module function. A controlled divergent tree constructed from these modules is indeed testable because the errors (1, 0) and (0, 1) both have the sensitizing vectors $Z_l = 1$ and $Z_l = 0$, and each $\hat{X}^j$ is surjective, that is, the values $I_X = \{0, 1\}$ appear in the output columns $\hat{X}^1$ and $\hat{X}^2$. Also note that any 1-bit, 1-to-n decoder will always be testable under the functional fault model since, by definition, each module output will be surjective and the required sensitizing vectors must exist.

Z_l X	$\hat{X}^1$	$\hat{X}^2$
0 0	0	1
0 1	1	1
1 0	1	0
1 1	1	1

Fig. 8. Truth table for the 1-to-2 decoder module.

4. Efficient Testing

We next examine two special properties called level-testability and C-testability that greatly reduce the number of tests for a controlled divergent tree.

L-testability. Consider the problem of testing a tree for all IP faults using the smallest possible test set. The size of the required test set can be exponential in the number of levels in the tree. For example, consider the demultiplexer module defined in Fig. 3. A p-level demultiplexer constructed from these modules requires 2^{p-1} different test patterns just for the set of SSL faults [15].

A tree is *level-testable* (L-testable) if all l-level modules $M_{i,l}$ can be simultaneously tested for any IP fault $ip_k \rightarrow (v_k, v_k')^j$. Trees that are L-testable have a test set whose size is bounded by $W \cdot p$, where W is a constant typically equal to the number of possible input patterns.

Theorem 3. *An n-ary p-level controlled divergent tree $CD(n, p)$ is L-testable if and only if* (1) *there is a sensitizing vector for every error* (v_a, v_b); (2) *for each $v_k \in I_X$ there is an input pattern ip_k that produces v_k at each state output $\hat{X}^j$, for* $1 \leq j \leq n$; *and* (3) *for each control value $v_1 \in I_Z$ there is a $v_2 \in I_Z$ such that subfunctions $\hat{X}^j(Z_l = v_1) \neq \hat{X}^k(Z_l = v_2)$, for some $1 \leq j, k \leq n$.*

The proof of Theorem 3 is similar to that of Theorem 1. The sensitizing condition of Theorem 3 insures error propagation while condition (2) is required for simultaneously applying any given input pattern ip_k to all modules on any single level. This capability accompanied by the sensitizing condition (1) allows all level-l (for any l) modules to tested simultaneously for all $n(2^{n_x} - 1)$ possible single IP faults. Condition (3) is both necessary and sufficient for testing control input faults.

Example 2. A truth table for a 1-to-2 decoder function different from Fig. 8 is shown in Fig. 9. This decoder module partially satisfies the conditions of Theorem 3. Condition (1) is satisfied in that sensitizing vectors exist for the errors (0, 1) and (1, 0). Condition (3) is satisfied as well in that the subfunctions $\hat{X}^1(Z_l = 0)$ and $\hat{X}^2(Z_l = 1)$ are not equal, that is, $\hat{X}^1(Z_l = 0) \neq \hat{X}^2(Z_l = 1)$. But condition (2) is only partially satisfied. This condition states that for each $v_k \in I_X = \{0, 1\}$ there must be an input pattern ip_k that produces v_k at $\hat{X}^1$ and $\hat{X}^2$. This is true for $v_k = 1$ but not for $v_k = 0$, that is, we have two input patterns, namely $(Z_l, X) = (0, 0)$ and $(0, 1)$, that produce 1 at both state outputs but no input pattern that produces 0 at both state outputs.

Z_l X	$\hat{X}^1$	$\hat{X}^2$
0 0	1	0
0 1	1	1
1 0	0	1
1 1	1	1

Fig. 9. Truth table for a 1-to-2 decoder module.

IP faults affecting $\hat{X}^1$	IP faults affecting $\hat{X}^2$	Are the faults L-testable?
$01 \rightarrow (1,0)^1$	$01 \rightarrow (1,0)^2$	yes
$11 \rightarrow (1,0)^1$	$11 \rightarrow (1,0)^2$	yes
$00 \rightarrow (1,0)^1$	$00 \rightarrow (0,1)^2$	no
$10 \rightarrow (0,1)^1$	$10 \rightarrow (1,0)^2$	no

Fig. 10. L-testability of the IP faults of the 1-to-2 decoder function defined in Fig. 9.

Since the testability of control faults only depends on conditions (1) and (3) of Theorem 3, we can conclude that a decoder of any size composed of the 1-to-2 decoder modules is testable for all control faults. We can also state that all IP faults involving the state input value 1 are L-testable, but the IP faults requiring the state input value 0 are not. Figure 10 summarizes the L-testability of IP faults in this case.

Example 3. RTRAM is a RAM architecture designed to improve both testability and performance [21]. It utilizes a complex decoder that has additional modes of operation beyond the normal decode function. Figure 11 shows the gate-level implementation of a decoder used in RTRAM. In [21], the authors show that the decoder is stuck-fault testable with a test set whose size is exponential in the number of levels. Analysis of the decoder implementation indicates that its function completely meets the L-testability conditions of Theorem 3. The decode function provides the sensitizing vectors and there exist many input patterns that produce a 0 and a 1 at both outputs; for example, $000000 \rightarrow 00$ and $110111 \rightarrow 11$. This means that the

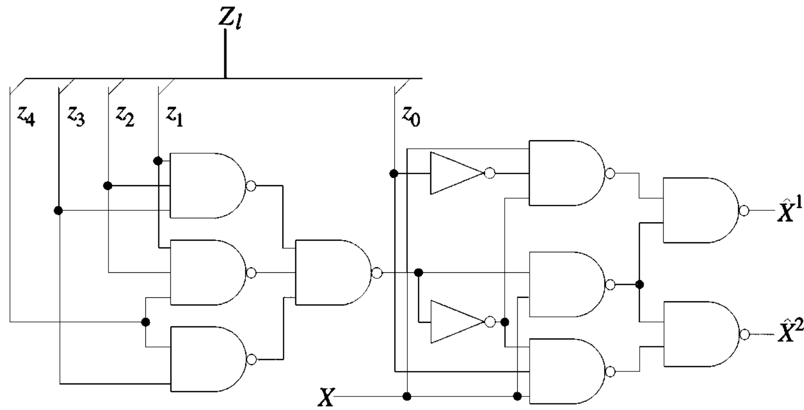


Fig. 11. Gate-level implementation of the RTRAM decoder module [21].

$2^7 = 128$ IP faults of each l-level decoder module are testable with 64 patterns.

C-testability. A closer examination of the divergent tree structure illustrated in Fig. 2 shows that it can be viewed as n^{p-1} arrays of n different *types*. For each array type j, the module input $X_{i,l}$ serves as the array's state input while the control input Z_l serves as the array's vertical input; the output $\hat{X}^j_{i,l}$ is the array's state output. Figure 12 shows a covering of the demultiplexer circuit of Fig. 2(a) with four arrays of two different types. Obviously a binary ($n = 2$) divergent tree module has two array types while an n-ary output module has n array types. Viewing trees as overlapping arrays allows the testing properties of array circuits to be generalized to divergent trees.

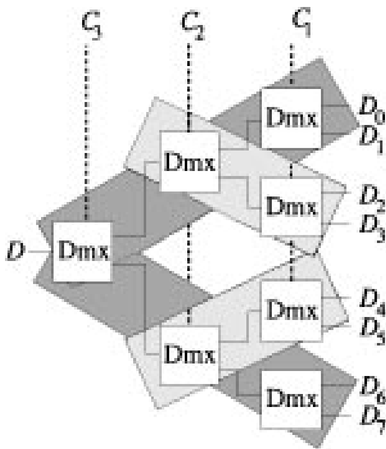


Fig. 12. A covering of a 3-level demultiplexer tree by four arrays of two different types.

An IP fault is *C-testable* in an array of arbitrary size if it can be tested with k test patterns, where k is some constant. Friedman [2] showed that an IP fault $ip_1 \rightarrow (v_1, v_1')$ is C-testable in a one-dimensional array if there is a sequence of module input patterns that regenerates ip_1 at an arbitrarily large number of modules and can propagate the error (v_1, v_1') from each of those modules. The following lemma gives a bound on test set size when faults are C-testable in the arrays within the divergent tree.

Lemma 1. *An n-ary p-level controlled divergent tree $CD(n, p)$ can be tested with at most kn^{p-1} test patterns if each type-j array is C-testable, where $1 \leq j \leq n$.*

Proof: If the conditions of the lemma are satisfied then the module is surjective. Hence, any module in $CD(n, p)$ can have any input pattern ip_k applied, and therefore $CD(n, p)$ can be tested by testing each array within the tree separately. Since each array type is C-testable, the bound on the number of test patterns is kn^{p-1}, where is k is some constant. Also note that any control-input fault is covered by testing the arrays contained in the tree. □

Because an error at the input of a divergent tree module can be propagated to one or more of the module's n state outputs, the conditions for C-testability of arrays within a controlled divergent tree are less restrictive than the single array case. Thus, determining array C-testability for divergent trees should consider all n state outputs. This point is illustrated in the following example.

Example 4. Reconsider the demultiplexer module. Its truth table and IP faults are shown again in

Z_l X	$\hat{X}^1$	$\hat{X}^2$
0 0	0	0
0 1	1	0
1 0	0	0
1 1	0	1

(a)

$\hat{X}^1$ IP faults	$\hat{X}^2$ IP faults
$Z_l X \rightarrow (v_k, v_k')^j$	$Z_l X \rightarrow (v_k, v_k')^j$
$00 \rightarrow (0, 1)^1$	$00 \rightarrow (0, 1)^2$
$01 \rightarrow (1, 0)^1$	$01 \rightarrow (0, 1)^2$
$10 \rightarrow (0, 1)^1$	$10 \rightarrow (0, 1)^2$
$11 \rightarrow (0, 1)^1$	$11 \rightarrow (1, 0)^2$

(b)

Fig. 13. (a) Truth table for the 1-to-2 demultiplexer module and (b) IP faults for the 1-to-2 demultiplexer module.

Figs. 13(a) and (b), respectively. Inspection of the truth table indicates that any control value applied to Z_l will propagate any error to either $\hat{X}^1$ or $\hat{X}^2$. Thus, an IP fault $ip_i \rightarrow (v_i, v_i')^j$ is C-testable for a type-j array if ip_i can be repetitively applied along an array of type j. The truth table reveals that all but two of the eight IP faults are C-testable in their corresponding array types; the two exceptions are $11 \rightarrow (0, 1)^1$ and $01 \rightarrow (0, 1)^2$. But note that all internal tree modules (all modules not on the output level p) are tested for these two faults when the other six faults are tested. For example, testing the $\hat{X}^2$-type arrays for $11 \rightarrow (1, 0)^2$, also tests each internal module for the fault $11 \rightarrow (0, 1)^1$. The number of p-level modules not tested for this fault is equal to the number of modules not contained in an array of type $\hat{X}^2$, that is, 2^{p-2}. Thus, an additional 2^{p-2} tests are needed to cover these modules. Similarly, any internal module fault $01 \rightarrow (0, 1)^2$ is detected by test patterns for the fault $01 \rightarrow (1, 0)^1$. Hence, an additional 2^{p-2} separate tests are required for the p-level modules and the fault $01 \rightarrow (0, 1)^2$. Thus, a test set for a p-level demultiplexer is bounded by $2 \cdot 2^{p-2} + 6 \cdot 2^{p-1} = 7 \cdot 2^{p-1}$. For the 3-level demultiplexer of Fig. 12, the bound is $7 \cdot 4 = 28$ tests.

In Section 3, we found that an uncontrolled divergent tree that is testable is also C-testable. This is in contrast to a controlled divergent tree which can easily be testable without being C-testable. But under certain conditions a controlled divergent tree is C-testable. First, the arrays within a divergent tree must all be C-testable if the divergent tree is to be C-testable. Second, all l-level modules (for $1 \leq l \leq p$) must have the same control value applied when the divergent tree is being tested. These conditions are formally stated in the following theorem.

Theorem 4. *An n-ary p-level controlled divergent tree $CD(n, p)$ is C-testable if and only if each type-j array is C-testable for $1 \leq j \leq n$ and all input patterns simultaneously applied to a given level l have identical control values.*

The proof of Theorem 4 is very similar to the convergent tree case proven in [18] and is therefore not presented here.

Example 5. In Example 4, we found that any pattern of control values will propagate errors to the demultiplexer tree outputs. Thus any set of input patterns that can be repetitively applied along each of the two array types that have identical control values will be C-testable. Input patterns 10 and 00 are two patterns that satisfy these conditions. Thus four of the possible eight IP faults associated with the demultiplexer module (also any of the decoder modules) are C-testable. The two test patterns that test for the faults $\{00 \rightarrow (0, 1)^1, 00 \rightarrow (0, 1)^2, 10 \rightarrow (0, 1)^1, 10 \rightarrow (0, 1)^2\}$ are shown in Fig. 14. The second test pattern shown in Fig. 14 illustrates error propagation for a faulty module (shaded) affected by the IP fault $00 \rightarrow (0, 1)^2$.

5. Design for Testability

The conditions of Theorem 3 suggest how to introduce L-testability into divergent tree circuits. Theorem 3 states that a controlled divergent tree is L-testable if the required sensitizing vectors exist, and for each state value $v_k \in I_X$ there is an input pattern ip_i that produces v_k at each state output $\hat{X}^j$. By adding a single control line or utilizing an unused control value $v_z \in I_Z$, the conditions of Theorem 3 can be easily satisfied.

Example 6. Consider testing a 1-to-1024 decoder implemented as a 9-level controlled divergent tree of 1-to-2 decoder modules; large decoders of this sort are commonly found in memory chips. The decoder has to be exhaustively tested to detect all SSL faults [15],

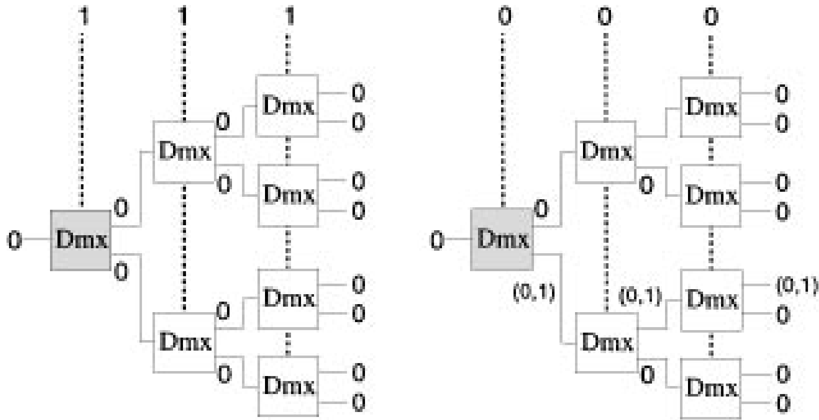


Fig. 14. Test patterns for the C-testable IP faults of the demultiplexer module.

hence a 1-to-1024 decoder requires $2^{10} = 1024$ test patterns. The 1-to-2 decoder module does have the required sensitizing vectors for L-testability, but is not L-testable because the state value 1 can only be applied to one module at a time in any level. This implies that only one module per level can be tested for a fault that requires the state value 1 for fault sensitization.

Adding a single control line *Test* to the 1-to-2 decoder module allows its function to be altered so that a 1 can be simultaneously generated at both the $\hat{X}^1$ and $\hat{X}^2$ state outputs. This modification results in the state table shown in Fig. 15. Notice that the input pattern $(Test, Z_l, X) = (1, 1, 1)$ produces 1 at both state outputs. A decoder built from these L-testable decoder modules requires Wp tests, where $W = 8$ is the number of test patterns required by a single module. Hence, a 1-to-1024 decoder composed of L-testable modules requires $9 \cdot 8 = 72$ test patterns, a reduction of nearly 93%. The delay associated with a 9-level decoder may be unacceptable for some applications. This delay can be reduced while preserving testability by using larger decoder modules, which can be made L-testable in the same way.

A gate-level implementation of the unmodified decoder function is shown in Fig. 16(a). A gate-level

		Test Z_l			
		00	01	10	11
X	0	00	10	00	01
	1	*dd*	*dd*	*dd*	11

Fig. 15. Truth table of the L-testable 1-to-2 decoder.

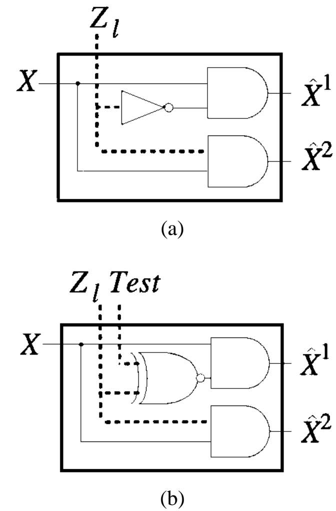


Fig. 16. Gate-level implementations of (a) the unmodified and (b) L-testable decoder modules.

implementation of the L-testable decoder function is shown in Fig. 16(b). Notice, that L-testability is achieved by simply replacing the NOT gate of Fig. 16(a) with an EXCLUSIVE-NOR gate. The L-testable decoder design presented here is similar to the ad hoc design presented in [15]. Here, a gate in the original decoder implementation is replaced with an EXCLUSIVE-NOR gate, introducing somewhat less overhead then the additional EXCLUSIVE-OR gate used in [15].

We show next how our DFT approach can enhance the testability of a practical circuit. A set of registers called a register file is often used in processor datapaths

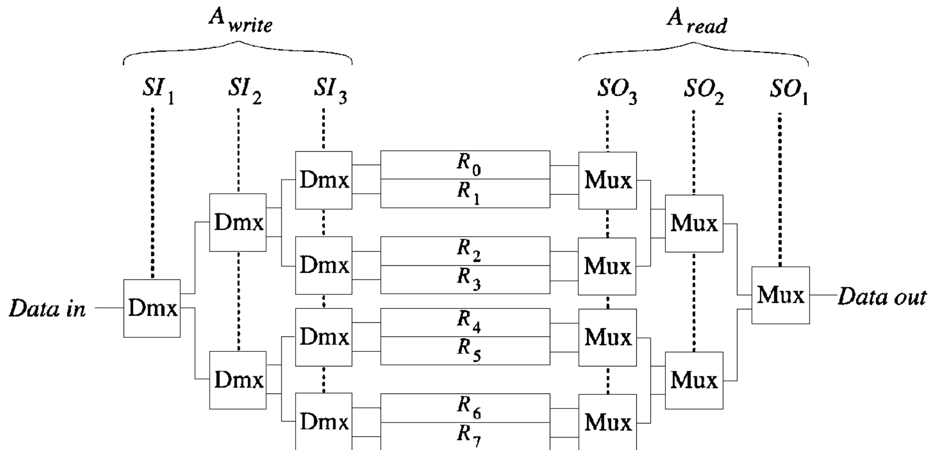


Fig. 17. A register file containing eight registers.

for data storage. It is typically implemented by arrays of D-type latches, with one or more I/O access ports connected to uni- or bidirectional lines [22]. The example in Fig. 17 contains eight registers, and uses multiplexer and demultiplexer trees to implement separate input and output ports. The register R_i with address $i = A_{\text{read}}$ can be read out through the multiplexer tree while, at the same time, the register R_j with address $j = A_{\text{write}}$ can be written into using the demultiplexer tree.

Using a similar DFT technique described in [18, 23], the multiplexer part of Fig. 17 can be made L-testable. To address the testability of the registers, we must again extend our fault model to sequential array circuits. In general, a sequential module is a finite-state machine whose memory outputs are all primary or serve as inputs to a neighboring module. To extend our functional fault model, we assume a sequential module can change its next-state (transition) function to any other, as long as the number of memory states does not increase. Thus testing a sequential array requires that the next-state function of each module be verified.

We will only consider the simplest case where the sequential modules are D latches [3]. Testing a register with this fault model is easy since each module only has two states. Its next-state function can be tested by verifying that a 0 and 1 can be written into each latch with present states of either 0 or 1. Since the data values applied to each latch are independent, the next-state function of all latches can be tested in parallel. Thus, registers are C-testable for this extension of our fault model.

When the multiplexer, demultiplexer, and registers are combined to form a register file as shown in Fig. 17, the testability of these circuits under the functional fault model is preserved, but testing is limited to a single register at a time. Suppose we use an L-testable demultiplexer and multiplexer circuit in Fig. 17. The demultiplexer cannot be "L-tested" because its data outputs cannot be simultaneously observed, that is, each data output must be written into a single register at a time. Similarly for the multiplexer, applying a single L-test requires eight separate writes into the register file. The L-testability characteristics of these modified trees can be preserved when combined to form a register file if all registers can be written into simultaneously, that is, all can be clocked at the same time. Figure 18

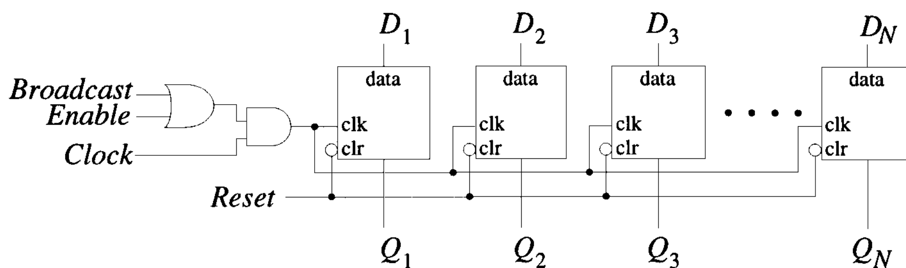


Fig. 18. An N-bit register viewed as a one-dimensional array of D-type latches.

shows a signal called *Broadcast* which is used to bypass the gated enable (*Enable*) signal of the register. If connected to registers $R_0, R_1, \ldots, R_7$, *Broadcast* allows the simultaneous writing of all registers. With this feature, the registers simply act as transparent buffers between the divergent and convergent tree structures, hence they do not affect fault sensitization or error propagation. Furthermore, since all registers can be written into simultaneously, they can all be tested in parallel. If each register has the same test pattern applied, than any single error resulting from a faulty register module will be propagated to the multiplexer outputs since the multiplexer is L-testable. Thus, the modified register file is L-testable.

6. Conclusion

We have investigated a class of regular circuits called divergent trees which include such practical circuits as decoders and demultiplexers. Because of their regular structure, the testability of divergent tree circuits of arbitrary size can be determined solely from the tree module's function. Uncontrolled divergent trees are testable with a fixed number of test patterns (C-testable) if and only if the module function is surjective. Testable controlled trees also have to be surjective but sensitizing vectors are required for error propagation. Controlled divergent trees can also be L-testable or C-testable if other conditions are satisfied. We found that decoders/demultiplexers only partially satisfy the requirements for L- and C-testability but can be modified to enhance their testability. Complete L-testability of a demultiplexer was achieved using gate replacement in the original gate-level design. The utility of the L-testable demultiplexer was further illustrated in the design of a small register file.

References

1. M. Abramovici, M.A. Breuer, and A.D. Friedman, *Digital Systems Testing and Testable Design*, IEEE Press, Piscataway, NJ, 1990.
2. A.D. Friedman, "Easily Testable Iterative Systems," *IEEE Transactions on Computers*, Vol. 22, No. 12, pp. 1061–1064, Dec. 1973.
3. T. Sridhar and J.P. Hayes, "Design of Easily Testable Bit-sliced Systems," *IEEE Transactions on Computers*, Vol. 30, No. 11, pp. 842–854, Nov. 1981.
4. R. Parthasarathy and S. Reddy, "A Testable Design of Iterative Logic Arrays," *IEEE Transactions on Circuits and Systems*, Vol. 28, No. 11, pp. 1037–1045, Nov. 1981.
5. W. Cheng and J.H. Patel, "Testing in Two-dimensional Iterative Logic Arrays," *Proc. of the 16th International Symposium on Fault-Tolerant Computing*, Oct. 1986, pp. 76–81.
6. H. Elhuni, A. Vergis, and L. Kinney, "C-Testability of Two-dimensional Iterative Arrays," *IEEE Transactions on Computer-Aided Design*, Vol. 5, No. 4, pp. 573–581, Oct. 1986.
7. C. Wu and P. Cappello, "Easily Testable Iterative Logic Arrays," *IEEE Transactions on Computers*, Vol. 39, No. 5, pp. 640–652, May 1990.
8. A. Takach and N. Jha, "Easily Testable Gate-level and DCVS Multipliers," *IEEE Transactions on Computer-Aided Design*, Vol. 10, No. 7, pp. 932–942, July 1991.
9. Q. Tong and N. Jha, "Design of C-Testable DCVS Binary Array Dividers," *IEEE Journal of Solid-State Circuits*, Vol. 26, No. 2, pp. 134–141, Feb. 1991.
10. A. Chatterjee and J. Abraham, "NCUBE: An Automatic Test Generation Program for Iterative Logic Arrays," *Proc. of International Conference on Computer-Aided Design*, Nov. 1988, pp. 428–431.
11. H. Elhuni and L. Kinney, "Techniques for Testing Hex Connected Systolic Arrays," *Proc. 1986 International Test Conference*, Sept. 1986, pp. 1024–1033.
12. J.H. Kim, "On the Design of Easily Testable and Reconfigurable Systolic Arrays," *Proc. International Conference on Systolic Arrays*, 1988, pp. 1024–1033.
13. W.T. Cheng, *Testing and Error Detection in Iterative Logic Arrays*, Ph.D. thesis, University of Illinois at Urbana-Champaign, 1985.
14. J. Abraham and D. Gajski, "Design of Testable Structures Defined by Simple Loops," *IEEE Transactions on Computers*, Vol. 30, No. 11, pp. 875–883, Nov. 1981.
15. D. Bhattacharya and J.P. Hayes, "Designing for High-level Test Generation," *IEEE Transactions on Computer-Aided Design*, Vol. 9, No. 7, pp. 752–766, July 1990.
16. F. Lombardi and D. Sciuto, "Constant Testability of Combinational Cellular Tree Structures," *Journal of Electronic Testing: Theory and Applications*, Vol. 3, No. 5, pp. 139–148, May 1992.
17. R.D. Blanton and J.P. Hayes, "Efficient Testing of Tree Circuits," *Proc. of the 23rd International Symposium on Fault-Tolerant Computing*, June 1993, pp. 176–185.
18. R.D. Blanton and J.P. Hayes, "Testability of Convergent Tree Circuits," *IEEE Transactions on Computers*, Vol. 45, No. 8, pp. 950–963, Aug. 1996.
19. R.D. Blanton and J.P. Hayes, "Properties of the Input Pattern Fault Model," *Proc. of 1997 International Conference on Computer Design*, Oct. 1997.
20. W.H. Kautz, "Testing for Faults in Cellular Logic Arrays," *Proc. 8th Symposium on Switching Automata Theory*, 1967, pp. 161–174.
21. D.K. Pradhan and N.R. Kamath, "RTRAM: Reconfigurable Testable Multi-bit RAM Design," *Proc. of 1988 International Test Conference*, Sept. 1988, pp. 263–278.
22. Texas Instruments, *TTL Data Book*, Vol. 2, Dallas, Texas, 1985.
23. R.D. Blanton, *Design and Testing of Regular Circuits*, Ph.D. thesis, University of Michigan, 1995.

Shawn Blanton is an assistant professor in the Department of Electrical and Computer Engineering at Carnegie Mellon University where he is a member of the Center for Electronic Design Automation and principal investigator of the SEMATECH Design for Testability

Cost Model Project. He received the Bachelor's degree in engineering from Calvin College in 1987, a Master's degree in Electrical Engineering in 1989 from the University of Arizona, and a Ph.D. degree in Computer Science and Engineering from the University of Michigan, Ann Arbor in 1994. His research interests include the computer-aided design of VLSI circuits and systems; fault-tolerant computing and diagnosis; verification and testing; and computer architecture. He has worked on the design and test of complex digital systems with General Motors Research Laboratories, AT&T Bell Laboratories, and Intel. Dr. Blanton is the recipient of National Science Foundation Career Award and is a member of IEEE and ACM.

John P. Hayes is Professor of Electrical Engineering and Computer Science at the University of Michigan, Ann Arbor, where he teaches and does research in the areas of computer architecture; computer-aided design, verification and testing; VLSI design; and fault-tolerant embedded systems. He received the B.E. degree from the National University of Ireland, Dublin, in 1965, and the M.S. and Ph.D. degrees from the University of Illinois, Urbana-Champaign, in 1967 and 1970, respectively, all in Electrical Engineering. While at the University of Illinois, he participated in the design of the ILLIAC III computer. In 1970 he joined the Operations Research Group at the Shell Benelux Computing Center in The Hague, where he worked on mathematical programming. From 1972 to 1982 Dr. Hayes was a faculty member of the Departments of Electrical Engineering-Systems and Computer Science of the University of Southern California, Los Angeles. He joined the University of Michigan in 1982. He was the founding director of the University of Michigan's Advanced Computer Architecture Laboratory. He was Technical Program Chairman of the 1977 International Conference on Fault-Tolerant Computing, Los Angeles, and the 1991 International Computer Architecture Symposium, Toronto. Dr. Hayes is the author of numerous technical papers and five books, including *Hierarchical Modeling for VLSI Circuit Testing*, (Kluwer, 1990; coauthored with D. Bhattacharya), *Introduction to Digital Logic Design*, (Addison-Wesley, 1993), and *Computer Architecture and Organization* (3rd ed., McGraw-Hill, 1998). He has served as editor of various technical journals, including the *IEEE Transactions on Parallel and Distributed Systems* and the *Journal of Electronic Testing*. Dr. Hayes is a Fellow of IEEE and a member of ACM and Sigma Xi.