VLSI Design
1994, Vol. 1, No. 4, pp. 285–298
Reprints available directly from the publisher
Photocopying permitted by license only

Optimal Testing and Design of Adders†

MICHAEL J. BATEK and JOHN P. HAYES
Advanced Computer Architecture Laboratory, Department of Electrical Engineering and Computer Science, University of Michigan

On-the-fly calculations of area and performance are a typical part of the computer-aided iterative design process in VLSI, which aims at a satisfactory tradeoff of various conflicting objectives, among which are test-generation time and test-set size. However, determining test sets on-the-fly as one circuit is transformed into another is extremely difficult. Our goal is to add a test dimension to the design optimization process that complements methods concerned with area and performance optimization. We define a set of logic transformations that result in easily computed changes to test sets. Test-set preserving (TSP) transformations preserve a combinational circuit's test sets, while test-set altering (TSA) transformations introduce a minimum number of tests needed to maintain completeness. We illustrate our approach with a family of adders that share area-efficient tree structures and differ in the amount of carry-lookahead used to accelerate carry computation. Members include the ripple-carry adder, which has no lookahead, and the standard carry-lookahead adder, which exploits lookahead across all inputs. It is straightforward to derive area and performance measures for this class of adders. Given an n-bit adder with lookahead degree k, we determine a sequence of circuit transformations that produce the adder of degree k^2 and test sets of minimum size. Optimal test sets of size $k(\log_k n + 1) + 2$ result for arbitrary n and k, which improve significantly upon previously reported tests.

Key Words: *Test-set optimization; Logic transformations; Adder design; Tree-structured adders; Testing; Logic synthesis*

Logic designers and automated design tools for VLSI both have the same goals—to obtain circuits that represent the best tradeoff among conflicting design objectives. Although design objectives vary in priority, traditionally the most important are area (measured by gate count) and performance (measured by worst-case delay), both of which should be minimized. Since it is usually straightforward to assess the impact of small changes to a design on area and performance, iterative improvement has become the predominant strategy for solving multidimensional design objectives. However, determining sets of input test patterns as a circuit is transformed into another is extremely complex, so generation of such test sets is normally separated from the design process. Yet there exist empirical results demonstrating close relationships among the test sets for various implementations of the same function. For instance, Davé and Patel have shown that a complete test set for a two-level circuit almost always detects all single stuck-line (SSL) faults in other realizations of the same circuit [7].

Others have circumvented the problem of on-the-fly test-set computation by employing circuit transformations that preserve the complete test sets of the initial circuit [3, 8, 9, 12], that is, the test sets for the initial circuit are sufficient to completely test all transformed circuits. We formalized the concept of test-set preserving (TSP) transformations in [3] and showed how they can explain analytically the test-set relationships found by Davé and Patel in adder design [7]. A drawback to these approaches is that the complete test sets for the initial circuit can be much larger than those necessary to test transformed designs. For example, the minimal n-bit two-level adder requires at least $c2^n$ tests, where c is a constant; this is equivalent to $\Omega(2^n)$ tests in complexity notation. We show in [3] that the two-level adder can be transformed into the ripple-carry adder with TSP transformations only. However, the ripple-carry ad-

†This research was supported by the National Science Foundation under Grant No. MIP-9200526.

der requires at most $O(1)$ tests, far fewer than the two-level adder.

One might think that it is straightforward to determine test patterns that are rendered unnecessary by a circuit transformation and remove them. However, a test that is no longer required for a transformed subcircuit cannot be removed without determining if the test is needed by the overall circuit. We therefore introduce test-set altering (TSA) transformations, which add a minimum number of tests to maintain complete test sets for SSL faults in transformed designs. TSA transformations enable test-set computation to take place in tandem with circuit transformation. The test sets for transformed designs are the union of the initial circuit's test sets with tests added by subsequent TSA transformations. TSA transformations also enlarge the design space. Sequences of transformed designs are no longer restricted to those monotonically decreasing in test set size, and initial circuits need no longer be in two-level or other canonical forms.

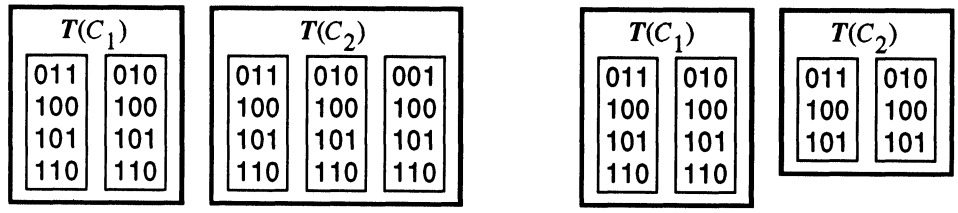
We apply TSA transformations to adder design, extending our earlier work on TSP transformations [3]. The target circuits are a family of regularly structured adders $\{\mathfrak{A}_n^k\}$, where n represents the size of the adder in bits and k the degree of carry-lookahead, that is, the number of input bits used to compute carries. $\{\mathfrak{A}_n^k\}$ includes a large number of adder designs that represent different solutions to area- and performance-based design objectives. For instance, $\mathfrak{A}_n^1$ is the ripple-carry adder, which is efficient in area but not in speed. Improvements in speed can be made at the expense of area and fanout; such improvements lead to carry-lookahead, carry-select, or carry-skip adders [1, 4, 5, 6, 10, 11, 15]. We restrict the class $\{\mathfrak{A}_n^k\}$ to area-efficient adders whose area complexity measured by gate count is $O(kn)$. The carry-select and carry-skip adders do not meet this bound; they also require larger test sets. The fastest but largest adder in $\{\mathfrak{A}_n^k\}$ is $\mathfrak{A}_n^n$, which consists of a constant number of logic levels.

It is a simple task to derive area and performance measures for the family $\{\mathfrak{A}_n^k\}$. Testing requirements are far more difficult to ascertain and presently there are only limited results. For example, it is well known that the n-bit ripple-carry adder is C-testable, that is, it can be completely tested for all SSL faults with a constant number of tests. Becker [4] determined that $O(\log_2 n)$ is an upper bound on the number of tests for the SSL faults in the carry-lookahead adder of Brent and Kung [5], however, Becker's test sets are not of minimal size, as we will demonstrate. Abraham and Gajski discuss a more general procedure that results in test sets that are $O(m)$ in size for a circuit composed of m modules [1]. This upper bound is also not tight because it does not account for the potential parallelism among tests.

In the following section, we present the notation necessary for transforming circuits and test sets. The structure of the family of adders $\{\mathfrak{A}_n^k\}$ is then examined, and the procedures for transforming the adder $\mathfrak{A}_n^k$ into $\mathfrak{A}_n^{k^2}$ that produce test sets of minimum size are detailed. We obtain test sets $k(\log_k n + 1) + 2$ in size, which we show is optimal for $k = n$. Following this, we discuss the relationship of other adder types of $\{\mathfrak{A}_n^k\}$.

TEST-SET ALTERING LOGIC TRANSFORMATIONS

In our earlier work [3], we defined a general set of test-set preserving (TSP) transformations for combinational logic circuits. A TSP transformation is a function $\mathfrak{I}$ that takes a circuit (or a corresponding expression) C_1 and returns the circuit $\mathfrak{I}(C_1) = C_2$. The SSL fault test sets $\boldsymbol{T}(C_1)$ for the initial circuit are related to the test sets $\boldsymbol{T}(C_2)$ for the transformed circuit by equality $\boldsymbol{T}(C_1) = \boldsymbol{T}(C_2)$, inclusion $\boldsymbol{T}(C_1) \subseteq \boldsymbol{T}(C_2)$, or covering $\mathbf{T}(C_1) \subseteq_c \boldsymbol{T}(C_2)$. Equality and inclusion relationships are straightforward; $\boldsymbol{T}(C_1)$ *covers* $\boldsymbol{T}(C_2)$ if and only if for any test set $T_1 \in \boldsymbol{T}(C_1)$ there exists some test set $T_2 \in \boldsymbol{T}(C_2)$ such that $T_2 \subseteq T_1$. In Figure 1a, $\boldsymbol{T}(C_1)$ is included in $\boldsymbol{T}(C_2)$. $\boldsymbol{T}(C_1)$ also covers $\boldsymbol{T}(C_2)$ by implication. Figure 1b illustrates the covering relationship that allows us to relate the



(a) $\boldsymbol{T}(C_1) \subseteq \boldsymbol{T}(C_2)$, $\boldsymbol{T}(C_1) \subseteq_c \boldsymbol{T}(C_2)$

(b) $\boldsymbol{T}(C_1) \not\subseteq \boldsymbol{T}(C_2)$, $\boldsymbol{T}(C_1) \subseteq_c \boldsymbol{T}(C_2)$

FIGURE 1 Examples of inclusion relationships among sets of test sets.

smaller test sets $\boldsymbol{T}(C_2)$ to $\boldsymbol{T}(C_1)$. Each of these relationships is transitive, hence a sequence of TSP transformations is also TSP.

A TSP sequence relates test sets that are monotonically decreasing in size—the initial test sets include all tests for the transformed circuits. As stated earlier, the initial test sets can be several orders of magnitude larger than necessary to test transformed designs. Hence our approach to this problem is to examine transformations that add a minimum number of tests to maintain completeness for the test sets of the transformed circuits. For instance, we would want to add the test $t = 110$ to $\boldsymbol{T}(C_1)$ if we transformed C_2 into C_1 in Figure 1b.

Our starting point is the four TSP transformations, fanout-free, De Morgan, extraction and resubstitution, defined in [3], which were shown to suffice for adder design. Combinations of these transformations produce transformations such as "flattening," used elsewhere in the literature. We now characterize the test-set effects of the inverses of these TSP transformations. In the case of fanout-free and De Morgan transformations, $\boldsymbol{T}(C_1) = \boldsymbol{T}(C_2)$, from which it follows that $\boldsymbol{T}(C_2) = \boldsymbol{T}(C_1)$ and their inverses are TSP. Typical examples of fanout-free and De Morgan transformations are $\mathfrak{F}((x_1 + x_2 + x_3)) = ((x_2 + x_2) + x_3)$ and $\mathfrak{M}((x_1 + \bar{x}_2)) = \overline{(\bar{x}_1 x_2)}$, respectively. The inverses of these transformations $\mathfrak{F}^{-1}$ and $\mathfrak{M}^{-1}$ are basically of the same type. However, in the case of extraction and resubstitution, $\boldsymbol{T}(C_1) \subseteq_c \boldsymbol{T}(C_2)$, which does not imply $\boldsymbol{T}(C_2) \subseteq_c \boldsymbol{T}(C_1)$; hence the inverse transformations, distribution and substitution respectively, are not TSP. A transformation $\mathfrak{T}(C_1) = C_2$ is *test-set altering* if $\boldsymbol{T}(C_1) \supseteq \boldsymbol{T}(C_2)$ or $\boldsymbol{T}(C_1) \supseteq_c \boldsymbol{T}(C_2)$.

The *distributive transformation* is defined as the inverse of the extraction transformation, that is $\mathfrak{D} = \mathfrak{E}^{-1}$, and corresponds to the distributive law of Boolean algebra. It aids in converting (flattening) a circuit to two-level form. Figure 2 illustrates the distributive transformation $\mathfrak{D}((x_1^1 + x_2^1)(x_1^2 + x_2^2)) = ((x_1^1 x_1^2) + (x_1^1 x_2^2) + (x_2^1 x_1^2) + (x_2^1 x_2^2))$ and the corresponding test-set transformation. Distribution has the general form $\mathfrak{D}(S) = P$, where $S = ((s_1^+)(s_2^+) \ldots (s_m^+))$ and $P = ((p_1^*) + (p_2^*) + \ldots + (p_q^*))$. The expressions (E^+) and (E^*) denote an arbitrary number of sum and product inputs, respectively. The number of product terms q in P is $n_1 n_2 \ldots n_m$ where n_i is the number of inputs to the sum (s_i^+). Inputs to (s_i^+) are labeled $x_1^i x_2^i \ldots x_{n_i}^i$, and are subject to minor independence constraints [3]. The minimum and complete test sets are related by $\boldsymbol{T}(S) \supseteq_c \boldsymbol{T}(P)$, hence distribution is TSA.

Since S and P are two-level circuits, we only need to consider faults on each input x_j^i. Consider the stuck-at-1 fault $x_j^i/1$ on an input x_j^i of (s_i^+). The test for this fault in S requires $(s_i^+) = 0$, or $x_1^i x_2^i \ldots x_n^i = 00 \ldots 0 = b_{n_i}$. The fault is propagated to the output of S if and only all $(s_k^+) = 1$ for $k \neq i$, or $x_1^k x_2^k \ldots x_{n_k}^k = dd \ldots 1 \ldots d = r_{n_k}$ (d denotes don't care). Hence $b_2 r_2 = 0011$ is the first test in Figure 2 that detects the faults $x_1^1/1$ and $x_2^1/1$. The corresponding branch faults of x_j^i in P can be detected in parallel if all $x_j^k = 1$ in (s_k^+) for $k \neq i$, or $x_1^k x_2^k \ldots x_{n_k}^k = 11 \ldots 1 = a_{n_k}$. Thus for all of the sum terms in S, we add a set of m tests, denoted by $T_1(P) = \cup_{i=1}^m (a_{n_1} \ldots a_{n_{i-1}} b_{n_i} a_{n_{i+1}} \ldots a_{n_m})$. $T_1(P) = b_2 a_2 \cup a_2 b_2 = 0011 \cup 1100$ for the example in Figure 2. However, $\boldsymbol{T}(S)$ contains $T_1(P)$ due to the patterns chosen for r_2; in this instance of $\boldsymbol{T}(S)$, no tests are added.

Next consider the stuck-at-0 fault $x_j^i/0$ on an input x_j^i of (s_i^+). The test for this fault in S requires $x_j^i = 1$ and $x_{j'}^i = 0$ for all $j' \neq j$, or $x_1^i x_2^i \ldots x_j^i \ldots x_{n_i}^i = 00 \ldots 1 \ldots 0$ and the same propagation conditions as before. Let W_{n_i} represent the n_i patterns $100 \ldots 0 \cup 0100 \ldots 0 \cup \ldots \cup 00 \ldots 01$. The tests for the faults $x_j^i/0$ in S, where i is constant, are $r_{n_1} \ldots r_{n_{i-1}} W_{n_{i+1}} r_{n_{i+1}} \ldots r_{n_m}$. For instance, the tests for $x_j^1/0$ in Figure 2 are $W_2 r_2 = (10 \cup 01) r_2 = 1010 \cup 0101$; here different choices for r_2 are made for each test. The corresponding branch stuck-at-0 faults in P may not be detected in parallel, hence a test must be

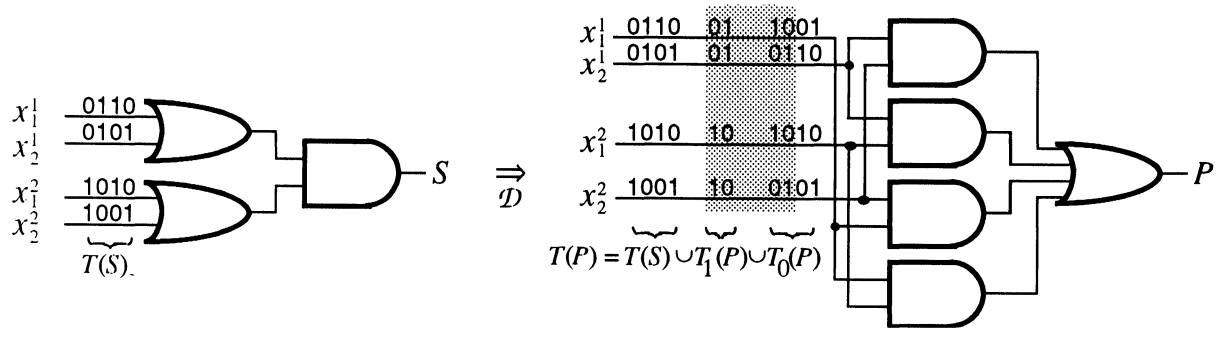


FIGURE 2 The transformation $\mathfrak{D}((x_1^1 + x_2^1)(x_1^2 + x_2^2)) = ((x_1^1 x_1^2) + (x_1^1 x_2^2) + (x_2^1 x_1^2) + (x_2^1 x_2^2))$.

added for each of the fanout branches. These tests are $T_0(P) = W_{n_1}W_{n_2} \ldots W_{n_m}$. $T_0(P) = W_2W_2 = (10 \cup 01)(10 \cup 01) = 1010 \cup 1001 \cup 0110 \cup 0101$ for the example in Figure 2. The shaded patterns represent tests already present in $T(S)$. Hence only two patterns are added for this form of distribution. The general test-set transformation for distribution is therefore of the form

$$\boldsymbol{T}(P) = \boldsymbol{T}(S) \cup T_1(P) \cup T_0(P)$$

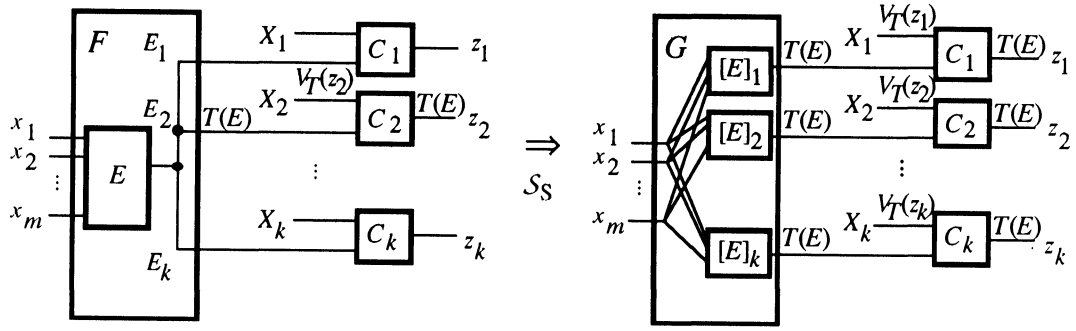
The *substitution transformation* is defined as the inverse of the resubstitution transformation, i.e., $\mathcal{S} = \mathcal{R}^{-1}$. Substitution replicates circuitry and moves fanout toward the primary inputs. We identify two forms of the substitution transformation, which are illustrated in Figure 3; these correspond to the two forms of resubstitution [3]. The test sets $\boldsymbol{T}(F)$ must be transformed so that the tests $T(E)$ are applied to k copies of subcircuit E. The form of the test set transformation depends on the type of substitution. In the case of *space-disjoint* substitution (Figure 3a), the outputs of F are controlled by disjoint sets of inputs X_i. Hence all copies of E can be tested in parallel. The change in the propagation of the tests $T(E)$ is illustrated in Figure 3a. Let $V_T(z_i)$ denote the values of X_i such that $z_i = E_i$. The corresponding test-set transformation is

$$\boldsymbol{T}(G) = \boldsymbol{T}(F) \cup T(E)(V_T(z_1)V_T(z_2) \ldots V_T(z_k))$$

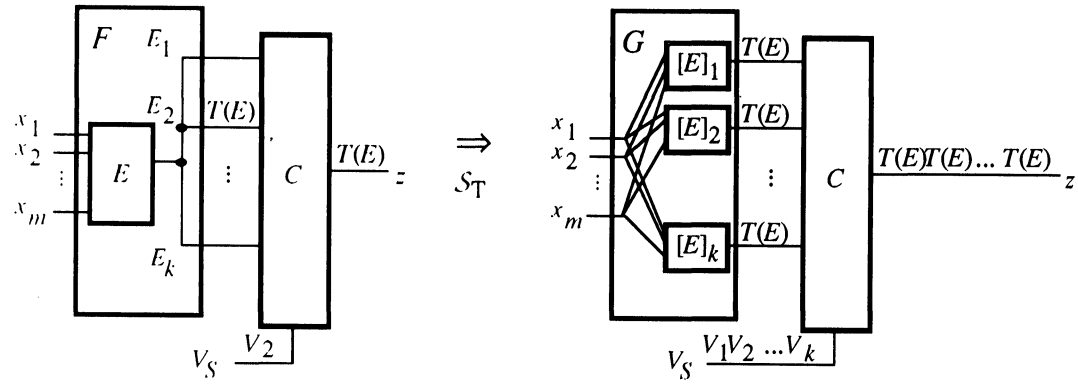
In *time-disjoint* substitution, the output $z = E_i$ for disjoint sets of values V_i of the select input V_S. Faults in different copies $[E]_i$ may not be detected in parallel since each value of V_S sensitizes only a single path from E to z. Hence tests are required to test each $[E]_i$ individually. This is reflected in Figure 3b by the sequential time-multiplexing of the tests $T(E)$ by the sets of values $V_1, V_2, \ldots, V_k$. The test set transformation for time-disjoint substitution is:

$$\boldsymbol{T}(G) = \boldsymbol{T}(F) \cup T(E)(V_1 \cup V_2 \cup \ldots \cup V_k)$$

The tests added by TSA transformations are expressed in terms of the module's inputs and outputs. In general, transformed modules are embedded within a circuit and these tests must be propagated and justified to insure that they detect new faults in the transformed circuitry. For the adder, a few justification and propagation functions V_J and V_P suffice to describe the conditions necessary to apply transformations to embedded modules. For arbitrary circuits, conventional testing techniques can be used.



(a) Space-disjoint substitution: $z_i = E_iX_i$.



(b) Time-disjoint substitution: $z = E_1V_1 + E_2V_2 + \cdots + E_kV_k$, $V_i \cap V_j = \emptyset$.

FIGURE 3 The two types of substitution transformations.

ADDER STRUCTURE

The structure common to all adders in $\{\mathfrak{A}_n^k\}$ is referred to by Unger [15] as the combinational iterative tree (CIT), and is applicable to any circuit that can be described iteratively. Each node in the tree computes an output function that characterizes the primary inputs feeding that node. In the case of the adder, three types of functions are computed: the carry function c_i, the carry-generate function g_i, and the carry-propagate function p_i. The adder's behavior is defined recursively by the carry equation $c_i = a_ib_i + a_ic_{i-1} + b_ic_{i-1}$ and sum equation $s_i = a_ib_ic_{i-1} + \bar{a}_i\bar{b}_ic_{i-1} + \bar{a}_ib_i\bar{c}_{i-1} + a_i\bar{b}_i\bar{c}_{i-1}$, or, equivalently, $s_i = a_i \oplus b_i \oplus c_{i-1}$. This iterative description gives rise immediately to the n-bit ripple-carry adder. The ripple-carry adder's tree structure is a linear chain, i.e., the n nodes in the tree have in-degree one, and each computes the carry c_i and sum s_i functions for a single pair of input bits a_i, b_i. Figure 4 depicts the tree structure inherent in this circuit, which we exploit in the circuit transformation process.

The carry-generate and carry-propagate functions are defined as $g_i = a_ib_i$ and $p_i = a_i + b_i$, respectively, and allow us to express the carry function as $c_i = g_i + p_ic_{i-1}$, where $g_0 = c_0$, the input carry. We can show inductively by unfolding the expression for c_i that

$$c_i = g_i + p_ic_{i-1} = g_i + p_ig_{i-1} + p_ip_{i-1}g_{i-2} + \ldots + p_ip_{i-1}\ldots p_1c_0 \quad (1)$$

Let the functions $g_{j,i}$ and $p_{j,i}$ denote the generate and propagate functions for the inputs a_i, b_i, a_{i-1}, b_{i-1}, $\ldots$, a_j, b_j. Let $g_{i,i} = g_i$ and $p_{i,i} = p_i$ for convenience. We can also show by induction that:

$$g_{j,i} = g_i + p_ig_{j,i-1} = g_i + p_ig_{i-1} + p_ip_{i-1}g_{i-2} + \ldots + p_ip_{i-1}\ldots p_{j+1}g_j \quad (2)$$

$$p_{j,i} = p_ip_{j,i-1} = p_ip_{i-1}\ldots p_j \quad (3)$$

Note that $g_{0,i} = c_i$.

The adder $\mathfrak{A}_n^k$ computes equations (1–3) and the sum functions s_i using four distinct logic stages, each consisting of a single module type GP_1, GP_k, C_{k-1} or S. The modules GP_1, GP_k, C_{k-1} and S compute g_i and p_i, equations (2) and (3) with $i - j = k$, equation (1), and s_i, respectively. In the $k = 1$ case, $\mathfrak{A}_n^1$ consists of GP_1, C_1 and S modules only. The first stage of $\mathfrak{A}_n^k$ contains n GP_1 modules that compute g_i, p_i for $1 \le i \le n$. The second stage of the adder consists of a regular k-ary tree of GP_k modules, which compute the functions $g_{j,i}$, $p_{j,i}$, where $i - j = k$. A regular k-ary tree has $\log_k n$ levels, where n is an integral power of k, interior nodes of degree $k + 1$, and n/k leaf nodes. There are $\Sigma_{i=0}^{\log_k n} k^i = (n - 1)/(k - 1)$ GP_k modules in the tree, each consisting of $k + 1$ gates. In the first level of the tree, $g_{j,i}$ and $p_{j,i}$ are calculated for $i = k, 2k, \ldots, n$ and $j = i - k + 1$ from the inputs g_n, p_n, g_{n-1}, p_{n-1}, $\ldots$, g_1, p_1 computed by the first stage. At the m-th level, $g_{j,i}$ and $p_{j,i}$ are calculated for $i = k^m, 2k^m, \ldots, n$ and $j = i - k^m + 1$ from the outputs of the previous level. For instance, the functions $gp_{1,4}$, $gp_{5,8}$, $\ldots$, $gp_{13,16}$ are computed in the second level of the adder $\mathfrak{A}_{16}^2$ shown in Figure 5. At the $\log_k n$-th level, $g_{j,i}$ and $p_{j,i}$ are computed for $i = k^{\log_k n} = n$ and $j = 1 - k^{\log_k n} + 1 = 1$.

The third stage of the adder is a divergent regular k-ary tree that computes the carry functions c_1, c_2, $\ldots$, c_n from the generate and propagate functions $g_{j,i}$ and $p_{j,i}$ of the GP_k tree and the input carry c_0. If the inputs to the module C_{k-1} are labeled g_{k-1}, p_{k-1}, $\ldots$, g_1, p_1, c_0, then C_{k-1} calculates the $k - 1$ carry functions c_{k-1}, c_{k-2}, $\ldots$, c_1 described by equation (1). The C_{k-1} tree contains $(n - 1)/(k - 1)$ modules of $\Sigma_{i=2}^{k} i = k(k + 1)/2 - 1$ gates each. The first level of the tree is a single module that computes the $k - 1$ carry functions $c_{n/k}$, $c_{2n/k}$, $\ldots$, $c_{(k-1)n/k}$ from the functions $g_{j,i}$, $p_{j,i}$ computed in level $\log_k n - 1$ of the GP_k tree. For example, the module C_1 computes the carry $c_{n/k} = c_8$ in the adder $\mathfrak{A}_{16}^2$ in Figure 5. The m-th level of the C_{k-1} tree consists of k^m modules that compute the carries from $g_{j,i}$, $p_{j,i}$ in level $\log_k n - m$ of the GP_k tree and the carry signals determined in previous levels of the C_{k-1} tree. After level m, the k^m carries c_0, c_{n/k^m}, c_{2n/k^m}, $\ldots$, $c_{(k^m-1)n/k^m}$ are avail-

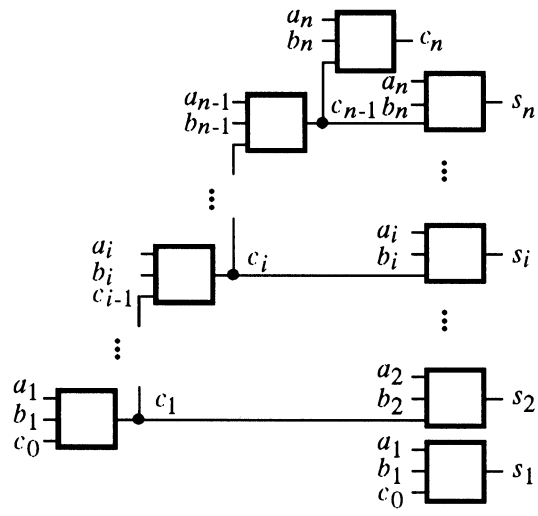


FIGURE 4 Tree structure of a ripple-carry adder.

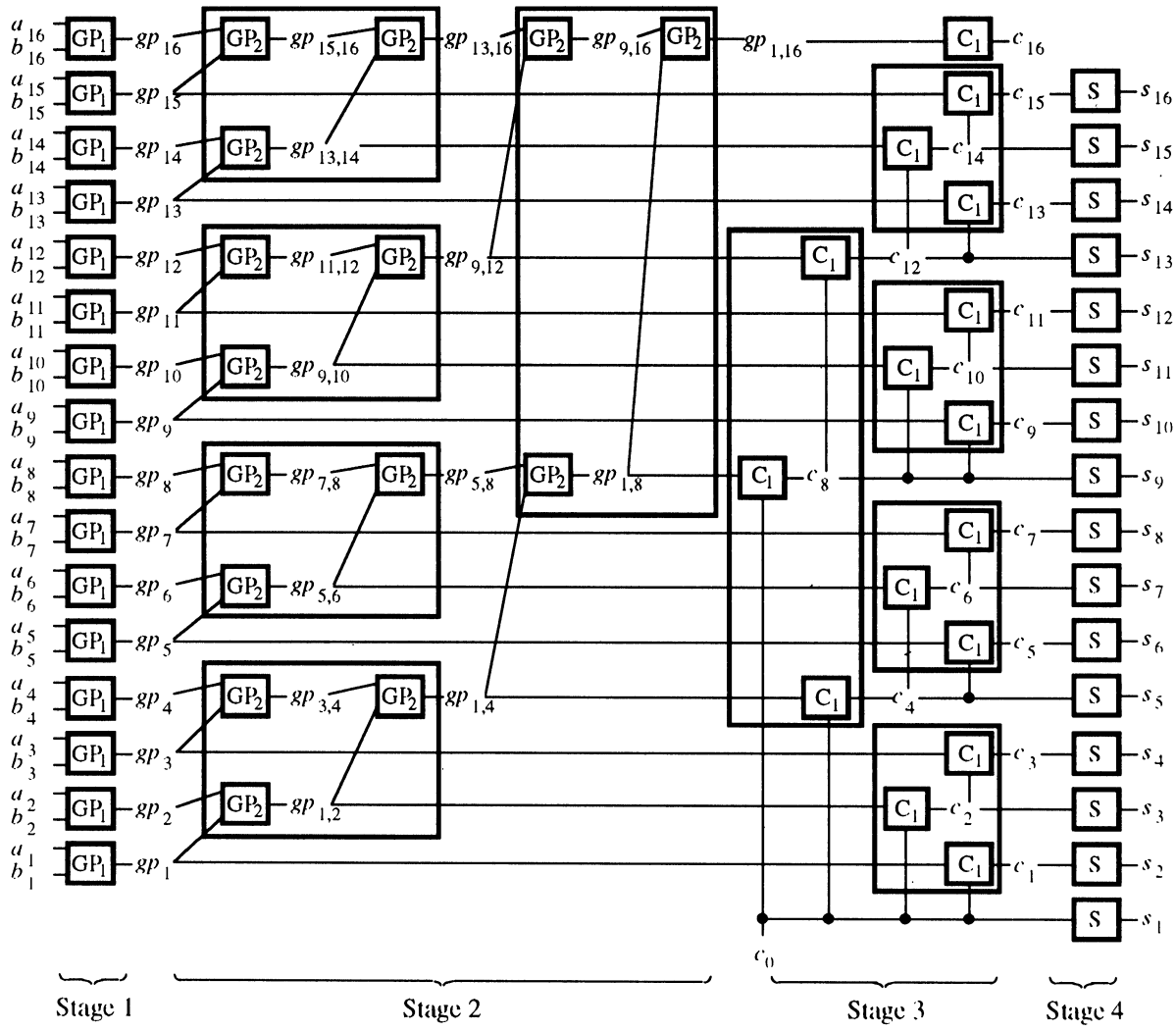


FIGURE 5 $\mathfrak{A}_{16}^2$: 16-bit adder of degree 2.

able. In the last level of the tree, $k^{\log_k n} = n$ and the available carries are $c_0, c_1, c_2, \ldots, c_{n-1}$. The overflow signal c_n is produced by $c_n = g_{1,n} + p_{1,n}c_0$ for all n and k.

In the fourth stage of the adder, the sum s_i is computed by the module S, for $1 \le i \le n$. We assume a specific implementation of S consisting of two 2-input exclusive-or modules. These modules may be realized in any form, as exhaustive testing (4 tests) is required to detect faults in all possible implementations. Figure 6 depicts the adder $\mathfrak{A}_{16}^4$, a quaternary structure present in the adder for the Advanced Micro Devices AM29050 microprocessor described by Lynch and Swartzlander [11].

Area and performance measures for this family of adders are easy to derive, which allow us to find optimal values of k. The number of modules in $\mathfrak{A}_n^k$ is $M(n, k) = n + (n - 1)/(k - 1) + (n - 1)/(k - 1) + n + 1 = ((2n + 1)k - 3)/(k - 1)$. When each term is multiplied by the number of gates per module, we have a gate count of

$$A_1(n, k) = \frac{(n - 1)k^2 + (23n + 1)k - (20n + 4)}{2(k - 1)} \quad (4)$$

which serves as a measure of area cost. The minimum number of gates occurs for $k = 3$, independent of n. The gate complexity of $O(kn)$ is superior to those for other adder types. For instance, the class of adders $\{\mathfrak{A}_n^k\}$ based on Abraham and Gajski's design [11] requires $O(kn\log_k n)$ gates.

We can obtain a second measure of area cost by considering the layout area. If we assume that a gate

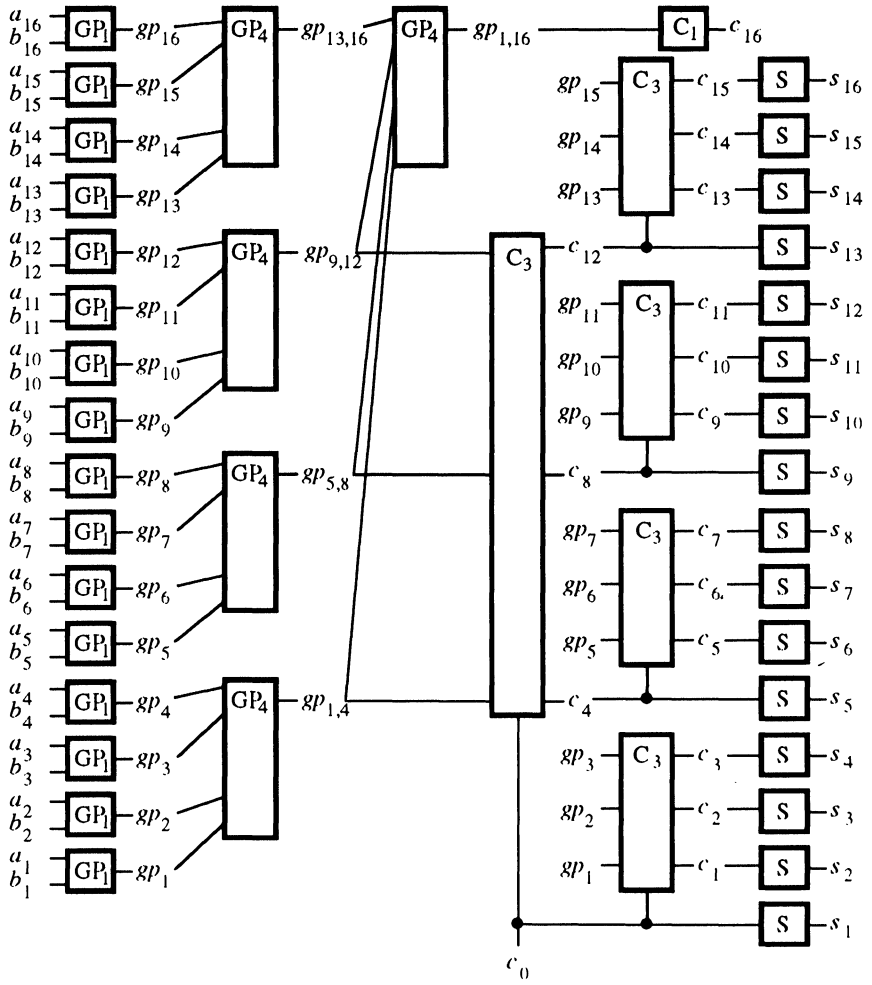


FIGURE 6 $\mathfrak{A}_{16}^4$: 16-bit adder of degree 4.

with m inputs occupies an area C proportional to m, the adder's layout area is

$$A_2(n, k) = \frac{C((n-1)k^3 + 15(n-1)k^2 + (194n + 22)k - 186n - 30)}{6(k-1)} \quad (5)$$

Based on this measure, the minimum area for an adder $\mathfrak{A}_n^k$ occurs when $k = 2$, even though the gate count $A_1(n, 2) > A_1(n, 3)$. Turning to performance measures, the longest delay in terms of gate-path length is

$$D(n, k) = 4\log_k n + 3 \quad (6)$$

The minimum delay occurs for $k = n$.

Using the above measures for area and delay, we have examined the optimization functions A_2D and A_2D^2, which commonly represent the "cost" of a VLSI implementation [13, 14]. Numerical solutions of these objective functions yield optimal values of k in between 5.7 and 6.2 due to a slight dependence on n. Hence both the A_2D and A_2D^2 objectives favor small amounts of lookahead k relative to n.

The adders in $\{\mathfrak{A}_n^k\}$ have a well-defined structure; consequently, estimates of their area and delay are easy to establish. Even though test generation is predominantly a structure-based process, characterizing the testing requirements for the adders in $\mathfrak{A}_n^k$ is not straightforward. We proceed to show how this can be done by applying the theory of TSA transformations described above.

CIRCUIT AND TEST-SET TRANSFORMATION FOR ADDERS

The degree k of an adder $\mathfrak{A}_n^k$ directly affects area and delay, as indicated by equations (4–6) in the previous

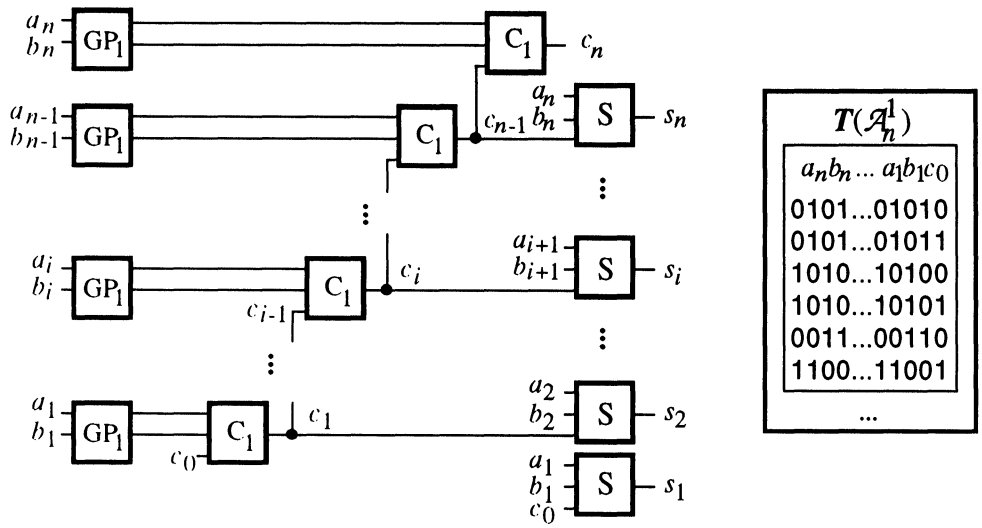


FIGURE 7 Initial ripple-carry adder $\mathfrak{A}_n^1$ and its test set.

section. However, the impact of k on the test sets of the adder is not as easily determined. We now show how to transform the adder $\mathfrak{A}_n^k$ into $\mathfrak{A}_n^{k^2}$ and produce test sets of minimum size. The procedure may be generalized to transform $\mathfrak{A}_n^k$ into $\mathfrak{A}_n^{k^m}$ for $m \geq 2$.

The initial circuit $\mathfrak{A}_n^1$ is a version of the ripple-carry adder of Figure 4. The function $c_i = a_ib_i + a_ic_{i-1} + b_ic_{i-1}$ is computed in the form $c_i = g_i + p_ic_{i-1}$. This form allows us to cast the ripple-carry adder into the structure defined in the previous section. Figure 7 depicts the structure of $\mathfrak{A}_n^1$ and an initial test set, which is a minimal test set for all SSL faults. The ripple-carry adder of Figure 7 does not have a GP_1 tree since the inputs required for the linear C_1 tree are provided by the functions g_i, p_i in the first stage.

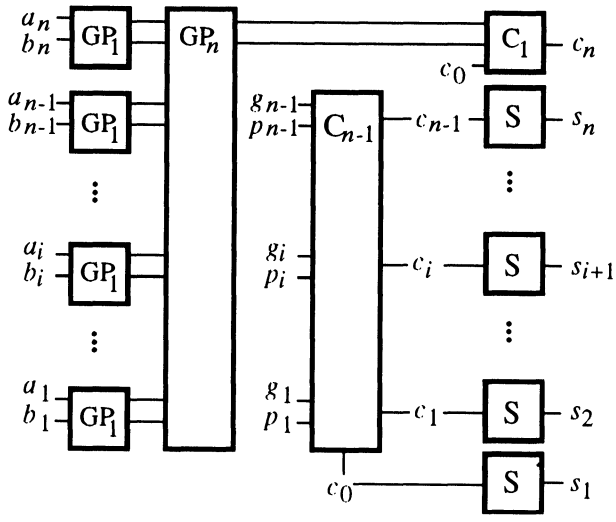


FIGURE 8 Final transformed circuit $\mathfrak{A}_n^n$.

The final transformed circuit $\mathfrak{A}_n^n$ results from repeatedly applying the procedure below that transforms $\mathfrak{A}_n^k$ into $\mathfrak{A}_n^{k^2}$. The final circuit consists of two single-node trees GP_n and C_{n-1}, illustrated in Figure 8. Incidentally, both $\mathfrak{A}_n^1$ and $\mathfrak{A}_n^n$ can be transformed into the minimal two-level adder by applying the specific TSA transformations that are the inverses of the TSP sequences for adders [3]. There are several potential sequences that transform $\mathfrak{A}_n^1$ into $\mathfrak{A}_n^n$, depending on the increment of k. Figure 9 depicts several of these, where $\Rightarrow$ denotes a transformation of $\mathfrak{A}_n^k$ into $\mathfrak{A}_n^{k^2}$. (When $k = 1$, $\Rightarrow$ denotes the transformation of $\mathfrak{A}_n^k$ into $\mathfrak{A}_n^{mk}$, for some integer $m > 1$.)

The procedure for transforming $\mathfrak{A}_n^k$ into $\mathfrak{A}_n^{k^2}$ is marked by a transformation of the modules GP_k and C_{k-1} into the modules GP_{k^2} and C_{k^2-1}, respectively. The straightforward process of grouping the modules GP_k and C_{k-1} that compose GP_{k^2} and C_{k^2-1} is illustrated in Figure 5 for $\mathfrak{A}_{16}^2$, which is to be transformed into $\mathfrak{A}_{16}^4$. We now describe the general procedures for transforming C_{k-1} into C_{k^2-1} and GP_k into GP_{k^2}. These are applied to modules of the circuit from right to left, that is, from primary outputs to primary inputs.

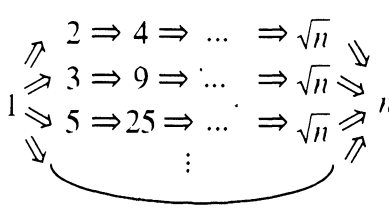


FIGURE 9 Values of k for some typical adder transformation sequences.

Tests are specified in algebraic terms using two functions V_J and V_P we call value-justify and value-propagate, respectively. Let $V_P(c_j)$ be defined as the values of the primary inputs that propagate the responses to the tests $T(C_i)$ for module C_i to the signal c_j. For example, $V_P(c_3) = a_3b_3 = 01$ or $10 = \{^{01}_{10}\}$ in Figure 5. Let $V_J(T(C_i))$ denote the values of the primary inputs that justify the tests $T(C_i)$ applied to module C_i. In the following tables, we simplify these expressions as much as possible to describe the form of the tests but avoid lengthy descriptions of easily computed test sets.

The goal of the circuit transformations in Table I is to move all fanout internal to C_{k^2-1} to its periphery, which is accomplished by steps 1 and 2, and subsequently apply distribution to produce C_{k^2-1} in two-level form. Table I also defines the TSA transformations that accompany each of the preceding circuit transformations. These steps are illustrated in Figure 10. In step 1, the space-disjoint substitution transformation $\mathcal{S}_{S,1}$ removes fanout on the carry signals $c_k, c_{2k}, \ldots, c_{(k-1)k}$ by replicating C_{k-1} $k-1$ times. The expression for the transformed test sets can be simplified; several of the added tests are implied by the simplified expression. In step 2, a second substitution transformation $\mathcal{S}_{S,2}$ produces $k(k-1)$ GP_k modules, one for each carry function c_{k^2-1} through c_k. For example, in Figure 5 two GP_2 modules are produced for c_3 and c_2. Substitution requires the addition of $O(k^2)$ tests for all copies of the repeated subcircuit GP_k. These tests are of the form $V_P(c_{k^2-1})V_J(T(GP_k))V_P(c_{k(k-1)}) = \{^{01}_{10}\} \cdots \{^{01}_{10}\}$ $V_J(T(GP_k)) \left\{\begin{smallmatrix}01\\10\\11\end{smallmatrix}\right\} \cdots \left\{\begin{smallmatrix}01\\10\\11\end{smallmatrix}\right\}$.

After some fanout-free transformations are applied, which do not affect test sets, the distribution transformation $\mathcal{D}_1$ reduces the four-level circuit to the two-level form C_{k^2-1}. The $\mathcal{D}_1$ transformation is applied $k-1$ times; each application introduces $O(k)$ tests. These tests are specified in terms of the inputs to C_{k^2-1}. After all C_{k^2-1} modules have been produced by this procedure, $O(k^2 \log_{k^2} n)$ tests have been introduced. Most of these new tests can be merged with the tests for $\mathcal{C}_n^k$.

Table II describes the transformation of the modules GP_k to GP_{k^2} and the test sets $\boldsymbol{T}(GP_k)$ to $\boldsymbol{T}(GP_{k^2})$, which proceeds from primary outputs toward primary inputs. Figure 11 illustrates the procedure. In step 1 the substitution transformation $\mathcal{S}_{T,1}$ moves the fanout from the $k-1$ gates of the form $p_{mk+1,(m+1)k}$ in GP_k to the inputs of GP_{k^2} so that the subcircuits g_{1,k^2} and p_{1,k^2} may be constructed independently. $T(p_{k+1,2k})$ consists of $k+1$ tests which are propagated separately by both $V_P(g_{1,k^2})$ and $V_P(p_{1,k^2})$. A fanout-free transformation produces p_{1,k^2} from $((p_{1,k})(p_{k+1,2k}) \cdots (p_{(k-1)k+1,k^2}))$. The module GP_{k^2} is completed by applying $\mathcal{D}_2$ $k-1$ times in step 2. Substep 2.*i* results in k additional tests, hence step 2 contributes $(k-1)k$ tests to $\boldsymbol{T}(GP_{k^2})$. After

TABLE I
Circuit and Test-Set Transformations for the C_{k^2-1} Tree

Step	Circuit Transformation	Test-set Transformation
1	$\mathcal{S}_{S,1}(C_{k-1}) = [C_{k-1}]_1, \ldots, [C_{k-1}]_{k-1}$	$\boldsymbol{T}(C_{k^2-1}) \Rightarrow \boldsymbol{T}(C_{k^2-1}) \cup V_P(c_{k^2-1})V_J(T(C_{k-1}))$ $= \left\{\begin{smallmatrix}01\\10\end{smallmatrix}\right\} \ldots \left\{\begin{smallmatrix}01\\10\end{smallmatrix}\right\} V_J(T(C_{k-1}))$
2	$\mathcal{S}_{S,2}(GP_k) = [GP_k]_1, \ldots, [GP_k]_{k(k-1)}$	$\boldsymbol{T}(C_{k^2-1}) \Rightarrow \boldsymbol{T}(C_{k^2-1})$ $\cup V_P(c_{k^2-1})V_J(T(GP_k))V_P(c_{k(k-1)})$ $\cup V_P(c_{k(k-1)-1})V_J(T(GP_k))V_P(c_{k(k-2)})$ $\cup \ldots \cup$ $\cup V_P(c_{2k-1})V_J(T(GP_k))V_P(c_k)$
3	$\mathcal{D}_1((p_{(k-1)k} \cdots p_{k+1}(g_k + \ldots + p_k p_{k-1} \ldots p_1 c_0))) =$ $((p_{(k-1)k} \cdots p_{k+1} g_k) + \ldots + (p_{(k-1)k} \cdots p_{k+1} p_k \cdots p_1 c_0))$	$\boldsymbol{T}(C_{k^2-1}) \Rightarrow \boldsymbol{T}(C_{k^2-1})$ $\cup V_J(g_{(k-1)k} p_{(k-1)k} \cdots g_{k+1} p_{k+1}\, g_k p_k \ldots c_0)$ $= V_J($ $d0d1 \ldots d1\ d1 \ldots d11$ $\cup\ d1d0 \ldots d1\ d1 \ldots d11$ $\cup \ldots \cup$ $\cup\ d1d1 \ldots d0\ d1 \ldots d11)$

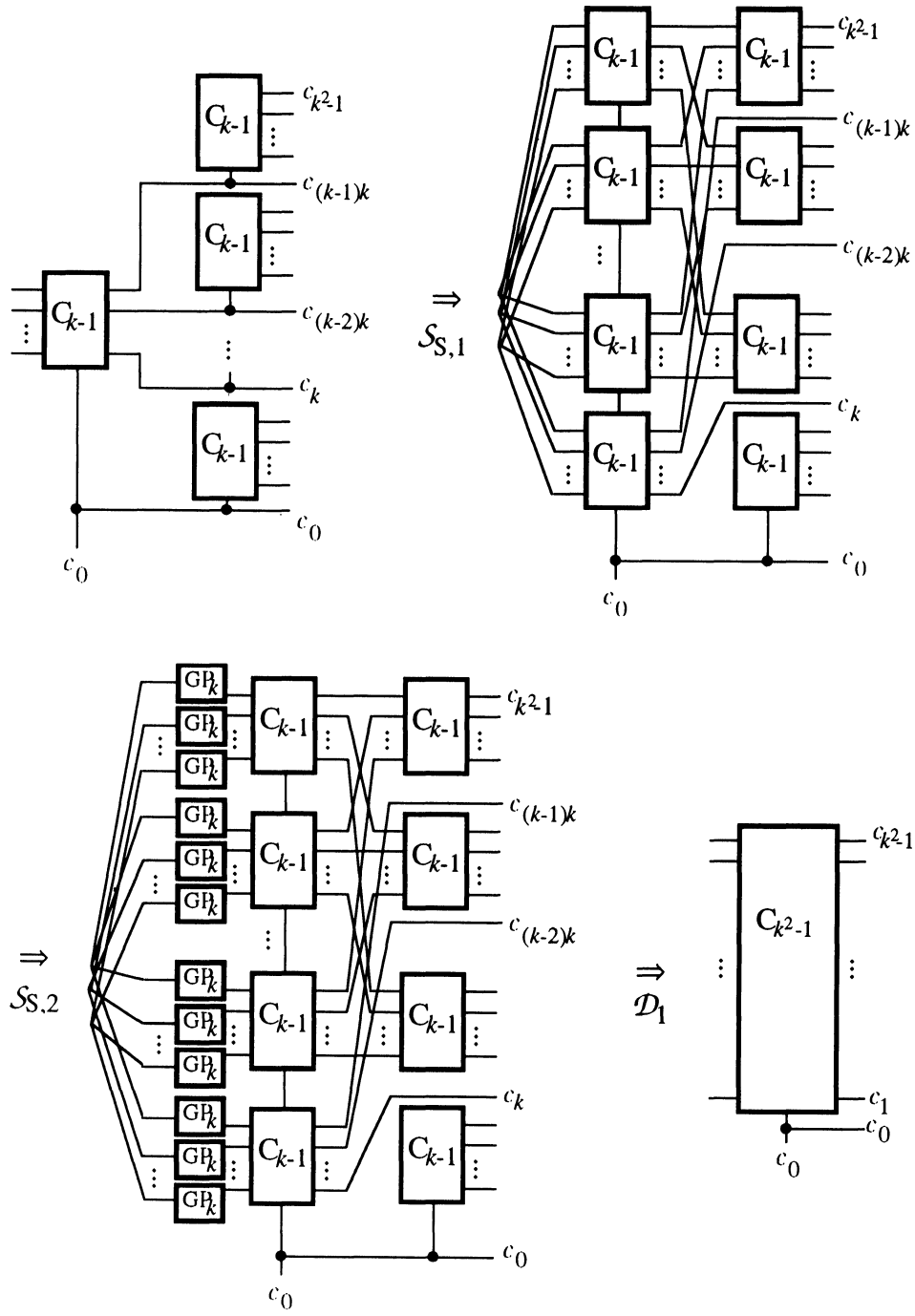


FIGURE 10 Transformation steps 1 through 3 for C_{k^2-1}.

producing all GP_{k^2} modules, $O(k^2 \log_{k^2} n)$ tests have been added. Again, most of these new tests can be merged with the tests for $\mathfrak{A}_n^k$, resulting in $O(k^2 \log_{k^2} n)$ tests for $\mathfrak{A}_n^{k^2}$.

After transformation to the adder $\mathfrak{A}_n^{k^2}$ is complete, the added tests must be combined and merged. Rather than describing the complex procedure for doing so, we illustrate the test sets that result from transforming an adder along the path $\mathfrak{A}_n^1 \Rightarrow \mathfrak{A}_n^2 \Rightarrow \mathfrak{A}_n^4 \Rightarrow \cdots \Rightarrow \mathfrak{A}_n^n$. We use the notation T^m to denote the test-by-test concatenation of m copies of a test set T. For the test set $T = \{t_1, t_2, \ldots, t_k\}$, $T^m = \{t_1^m,$

TABLE II
Circuit and Test-Set Transformations for the GP_{k^2} Tree

Step	Circuit Transformation	Test-set Transformation
1.1	$\mathcal{S}_{T,1}(p_{k+1,2k}) = [p_{k+1,2k}]_1, [p_{k+1,2k}]_2, \cdots$	$\boldsymbol{T}(GP_{k^2}) \Rightarrow \boldsymbol{T}(GP_{k^2}) \cup T(p_{k+1,2k})V_P(p_{1,k^2}) \cup T(p_{k+1,2k})P(g_{1,k^2})$
1.2	Replace k and $2k$ with $2k$ and $3k$ in substep 1.1	
...		
1.k–1	Replace k and $2k$ with $(k-1)k$ and k^2 in substep 1.1	
2.1	$\mathcal{D}_2((p_{k^2}\cdots p_{k+1}(g_k + p_k g_{k-1} + \cdots + p_k p_{k-1}\cdots p_1 c_0))) = ((p_{k^2}\cdots p_{k+1} g_k) + \cdots + (p_{k^2}\cdots p_{k+1} p_k p_{k-1}\cdots p_1 c_0))$	$\boldsymbol{T}(GP_{k^2}) \Rightarrow \boldsymbol{T}(GP_{k^2}) \cup V_J(g_{k^2}p_{k^2}\cdots g_{k+1}p_{k+1}\ g_k p_k \cdots g_1 p_1)$ $= V_J(\ 1dd0\ldots d0\ 1d\ldots 1d \cup d01d\ldots d0\ 1d\ldots 1d \cup \ldots \cup d0\ldots d01d\ 1d\ldots 1d)$
2.2	Replace k with $2k$ in substep 2.1	
...		
2.k–1	Replace k with $(k-1)k$ in substep 2.1	

$t_2^m, \ldots, t_k^m\}$. For example, if $T = 010 \cup 100$, $T^2 = 010010 \cup 100100$. The following expresses the tests for the initial ripple-carry adder $\mathfrak{A}_n^1$ of Figure 7.

$$\boldsymbol{T}(\mathfrak{A}_n^1) = (010 \cup 100 \cup 011 \cup 101 \cup 110 \cup 001)^n$$

$$|\boldsymbol{T}(\mathfrak{A}_n^1)| = 6$$

Table III recursively expresses the test sets $\boldsymbol{T}(\mathfrak{A}_n^k)$ that result from applying the procedures of Tables I and II with various values of k.

The above analysis shows that the test-set size for $\mathfrak{A}_n^k$ is $|\boldsymbol{T}(\mathfrak{A}_n^k)| = k(\log_k n + 1) + 2$. Becker [4] has shown that Brent and Kung's adder [5], which is a simplified version of $\mathfrak{A}_n^2$, can be tested for SSL faults

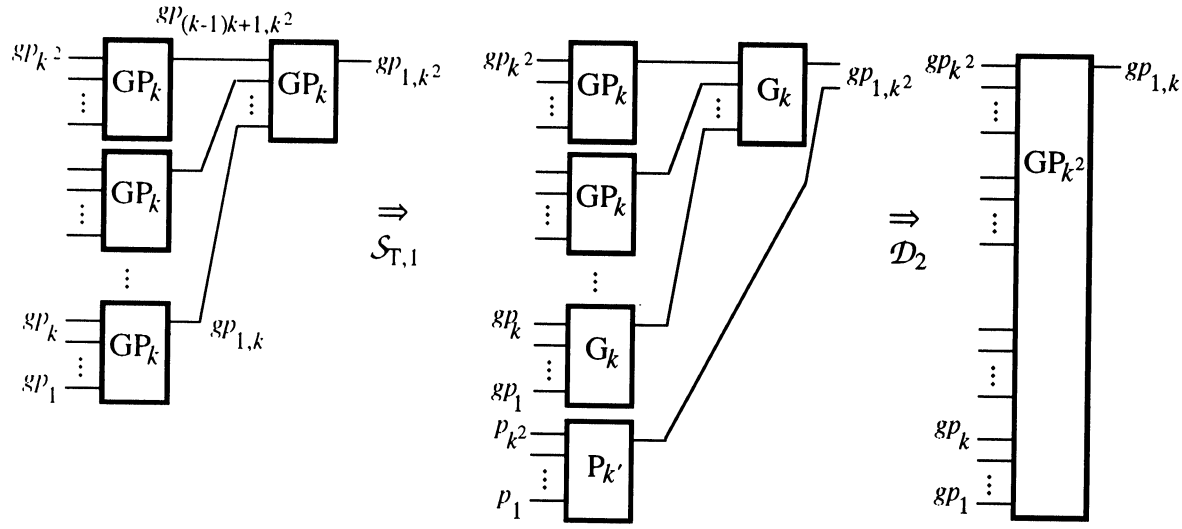


FIGURE 11 Transformation steps 1 and 2 for GP_{k^2}.

TABLE III
Test Sets for the Adders $\mathfrak{A}_n^2$, $\mathfrak{A}_n^4$ and $\mathfrak{A}_n^n$

k	Test Set $T(\mathcal{A}_n^k)$	a_nb_n ...	$a_{3n/4}b_{3n/4}$ ...	$a_{n/2}b_{n/2}$ ...	$a_{n/4}b_{n/4}$ ...	$a_1b_1c_0$	$\lvert T(\mathcal{A}_n^k)\rvert$
2	$(T(\mathcal{A}_{n/2}^2))^2$	$\cup \left\{\begin{smallmatrix}01\\10\end{smallmatrix}\right\} \ldots \left\{\begin{smallmatrix}01\\10\end{smallmatrix}\right\}$ 00 11 ...				111	$2\log_2 n + 4$
		$\cup \left\{\begin{smallmatrix}01\\10\end{smallmatrix}\right\} \ldots$		$\left\{\begin{smallmatrix}01\\10\end{smallmatrix}\right\}$ 00 11 ...		111	
4	$(T(\mathcal{A}_{n/4}^4))^4$	$\cup \left\{\begin{smallmatrix}01\\10\end{smallmatrix}\right\} \ldots \left\{\begin{smallmatrix}01\\10\end{smallmatrix}\right\}$ 00 11 ...				111	$4\log_4 n + 6$
		$\cup \left\{\begin{smallmatrix}01\\10\end{smallmatrix}\right\} \ldots$		$\left\{\begin{smallmatrix}01\\10\end{smallmatrix}\right\}$ 00 11 ...		111	
		$\cup \left\{\begin{smallmatrix}01\\10\end{smallmatrix}\right\} \ldots$		$\left\{\begin{smallmatrix}01\\10\end{smallmatrix}\right\}$ 11 0			
		$\cup \left\{\begin{smallmatrix}01\\10\end{smallmatrix}\right\} \ldots$			$\left\{\begin{smallmatrix}01\\10\end{smallmatrix}\right\}$ 11 0		
n	$\left\{\begin{smallmatrix}01\\10\end{smallmatrix}\right\}^1 T(\mathcal{A}_{n-1}^{n-1})$	$\cup$ 110 $\cup$					$2(n+1)$
	$\left\{\begin{smallmatrix}01\\10\end{smallmatrix}\right\}^0 T(\mathcal{A}_{n-1}^{n-1})$	$\cup$ 0011 ...				111	

by $8\log_2 n - 6$ tests. However, our analysis demonstrates that $|\boldsymbol{T}(\mathfrak{A}_n^2)| = 2\log_2 n + 4$, roughly a quarter of the number of tests determined by Becker. We now show that the final test set $\boldsymbol{T}(\mathfrak{A}_n^n)$ in Table III is optimal.

Theorem 1: $|\boldsymbol{T}(\mathfrak{A}_n^n)| = 2(n + 1)$

Proof: The test set $\boldsymbol{T}(\mathfrak{A}_n^n)$ in Table III is guaranteed to be complete by our TSA analysis because we have added tests to maintain completeness for each TSA transformation. Hence $|\boldsymbol{T}(\mathfrak{A}_n^n)| \leq 2(n + 1)$. We proceed to show that $2(n + 1)$ is also a lower bound for $|\boldsymbol{T}(\mathfrak{A}_n^n)|$. Consider the subcircuit that computes the carry $c_n = g_n + p_n g_{n-1} + \cdots + p_n p_{n-1} \cdots p_1 c_0$ in $\mathfrak{A}_n^n$ (modules GP_n and C_1 in Figure 8), which consists of $n + 1$ terms labeled in a right-to-left ordering by c_n^i, that is, $c_n^i = p_n^i p_{n-1}^i \cdots p_i^i g_{i-1}$, where $1 \leq i \leq n$. Any complete test set must detect the faults $g_{i-1}/0$, $g_{i-1}/1$, and $p_j^i/1$, $i \leq j \leq n$, for each of these product terms. The tests for the faults $g_{i-1}/0$, $g_{i-1}/1$, and $p_j^i/1$ are presented in Table IV in the form of a fault table. A complete set of tests for the subcircuit computing c_n is the union of the tests obtained by varying i and j over the ranges specified in the table.

The tests for $g_{i-1}/0$, $1 \leq i \leq n + 1$, which form row 1 of Table IV, are pairwise disjoint, and the pattern $a_nb_nc_{n-1} = \left\{\begin{smallmatrix}01\\10\end{smallmatrix}\right\}$ 1 or 110 makes them disjoint from all other tests. Hence these $n + 1$ tests are essential and must be in any complete test set for $\mathfrak{A}_n^n$. The tests for the faults $g_{i-1}/1$, $2 \leq i \leq n + 1$, contain the test $a_nb_n \ldots c_0 = \left\{\begin{smallmatrix}01\\10\end{smallmatrix}\right\}\left\{\begin{smallmatrix}01\\10\end{smallmatrix}\right\} \ldots \left\{\begin{smallmatrix}01\\10\end{smallmatrix}\right\}0$, which detects $g_0/1 = c_0/1$. The faults $g_{i-1}/1$, $2 \leq i \leq n + 1$, dominate the fault $g_0/1$ and can be removed from the fault table. The test $\left\{\begin{smallmatrix}01\\10\end{smallmatrix}\right\}\left\{\begin{smallmatrix}01\\10\end{smallmatrix}\right\} \ldots \left\{\begin{smallmatrix}01\\10\end{smallmatrix}\right\}0$ is disjoint

TABLE IV
Fault Table for the Subcircuit of $\mathfrak{A}_n^n$ Computing c_n

Fault	Tests: a_nb_n	$a_{n-1}b_{n-1}$	...	$a_{j+1}b_{j+1}$	a_jb_j	$a_{j-1}b_{j-1}$	...	a_ib_i	$a_{i-1}b_{i-1}$	$a_{i-2}b_{i-2}$	...	a_1b_1
$g_{i-1}/0$, $1 \leq i \leq n+1$	$\left\{\begin{smallmatrix}01\\10\end{smallmatrix}\right\}$	$\left\{\begin{smallmatrix}01\\10\end{smallmatrix}\right\}$	...				...	$\left\{\begin{smallmatrix}01\\10\end{smallmatrix}\right\}$	110	*dd*	...	*dd*
$g_{i-1}/1$, $1 \leq i \leq n+1$	$\left\{\begin{smallmatrix}01\\10\end{smallmatrix}\right\}$	$\left\{\begin{smallmatrix}01\\10\end{smallmatrix}\right\}$	...				...	$\left\{\begin{smallmatrix}01\\10\end{smallmatrix}\right\}$0	*dd*	*dd*	...	*dd*
$p_j^i/1$, $1 \leq i \leq n$, $i \leq j \leq n$	$\left\{\begin{smallmatrix}01\\10\end{smallmatrix}\right\}$	$\left\{\begin{smallmatrix}01\\10\end{smallmatrix}\right\}$	...	$\left\{\begin{smallmatrix}01\\10\end{smallmatrix}\right\}$	00	$\left\{\begin{smallmatrix}01\\10\\11\end{smallmatrix}\right\}$	...	$\left\{\begin{smallmatrix}01\\10\\11\end{smallmatrix}\right\}$	110	*dd*	...	*dd*

from all others, hence it too is essential. The tests for $p_j^i/1$, where i is constant, may be combined into a single test of the form $a_n b_n \ldots c_0 = \{\substack{01\\10}\}\{\substack{01\\10}\} \ldots \{\substack{01\\10}\}0011 \ldots 111$. (Those for different i are disjoint.) This results in n tests since $1 \leq i \leq n$. Hence the fewest tests required to detect all SSL faults in the subcircuit computing c_n is $n + 1 + 1 + n = 2(n + 1)$. Thus $|\boldsymbol{T}(\mathfrak{A}_n^n)| \geq 2(n + 1)$ and $|\boldsymbol{T}(\mathfrak{A}_n^n)| = 2(n + 1)$ follows. □

Now that we have a well-defined measure $|\boldsymbol{T}(\mathfrak{A}_n^n)|$ for the testing dimension of a circuit, it is possible to perform tradeoff analysis using test-set size as a parameter in the objective function. For example, we might choose the objective function $A_2(n, k)/|T(\mathfrak{A}_n^k)|$, which is analogous to the area-delay objective used in [14]. It is easily shown that the value of k that minimizes A_2/T occurs between 7.4 and 9.2 for values of n ranging from 16 to 1024. A counterintuitive result of this analysis is that the larger values for k occur for the smaller values of n, meaning that as the number of input bits increases, the optimal amount of carry-lookahead decreases.

EXTENSION TO OTHER ADDER TYPES

Other adders that do not have the structure described in section III require more area and, in turn, larger test sets. These include the carry-select and conditional sum adders, among others [6, 10]. As mentioned previously, the class of adders $\{\mathfrak{AG}_n^k\}$ based on Abraham and Gajski's design [1] requires $O(kn\log_k n)$ gates, which is greater than the bound of $O(kn)$ for $\{\mathfrak{A}_n^k\}$. We show briefly how $\{\mathfrak{AG}_n^k\}$ relates to the class $\{\mathfrak{A}_n^k\}$. Figure 12a illustrates $\mathfrak{AG}_8^2$, redrawn in a form that resembles $\mathfrak{A}_8^2$. $\mathfrak{AG}_8^2$ contains the first two stages of $\mathfrak{A}_8^2$ and part of the third stage.

$\mathfrak{AG}_8^2$ is transformed into $\mathfrak{A}_8^2$ using the TSP transformations $\mathfrak{F}$, $\mathfrak{E}$ and $\mathfrak{R}_S$, hence the test sets for $\mathfrak{AG}_8^2$ must be no smaller than the test sets for $\mathfrak{A}_8^2$. The fanout-free and extraction transformations $\mathfrak{F}$ and $\mathfrak{E}$ transform a connected pair of GP_2–C_1 modules into a pair of C_1–C_1 modules. The first of these C_1 modules is present in the partial C_1 tree of $\mathfrak{A}_8^2$; hence the application of the resubstitution transformation $\mathfrak{R}_S$ that combines these replicated subcircuits. The second C_1 module becomes part of the C_1 tree. This sequence is repeated level by level for other GP_2–C_1 pairs, producing the adder of Figure 12b. Similar sequences exist for other forms of Abraham and Gajski's adder; their test sets can be determined by generalizing this example.

CONCLUSIONS

We have shown for the first time that it is possible to generate optimal test sequences on-the-fly while exploring the design space for adders. We have done this by defining test-set-altering transformations that complement the circuit transformation rules typically used in CAD-based design. This makes it possible to treat testing on an equal footing with area and delay in making VLSI design tradeoffs. Currently, we are investigating the applicability of our approach to a broader class of designs that can be described

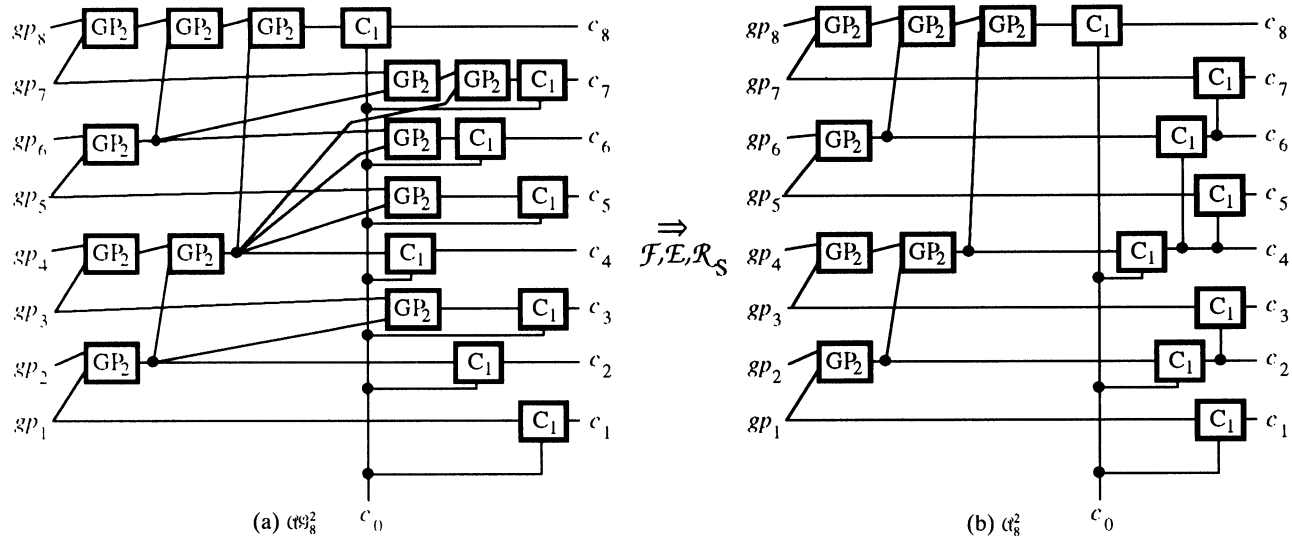


FIGURE 12 TSP transformation for the Abraham-Gajski adder $\mathfrak{AG}_8^2$.

iteratively. It appears possible to extend our work to these circuit types, making it feasible to explore the rich design space of VLSI circuits and obtain optimal test sets at the same time.

References

[1] J. Abraham and D. Gajski, "Easily testable, high-speed realizations of register-transfer-level operations," in *Proceedings of the 10th Fault-Tolerant Computing Symposium,* 1980, pp. 339–344.

[2] M. Abramovici, M. Breuer and A. Friedman, *Digital Systems Testing and Testable Design,* Computer Science Press, New York, 1990.

[3] M. Batek and J. Hayes, "Test-set preserving logic transformations," in *Proceedings of the 26th Design Automation Conference,* 1992, pp. 454–458.

[4] B. Becker, "Efficient testing of optimal time adders," *IEEE Transactions on Computers,* Vol. C-37, September 1988, pp. 1113–1121.

[5] R. Brent and H. Kung, "A regular layout for parallel adders," *IEEE Transactions on Computers,* Vol. C-31, March 1982, pp. 260–264.

[6] J. Cavanagh, *Digital Computer Arithmetic: Design and Implementation,* McGraw-Hill, New York, 1984.

[7] U. Davé and J. Patel, "A functional-level test generation methodology using two-level representations," in *Proceedings of the International Conference on CAD,* 1989, pp. 422–425.

[8] S. Devadas and K. Keutzer, "Design of integrated circuits fully testable for delay-faults and multifaults," in *Proceedings of the International Test Conference,* 1990, pp. 284–293.

[9] G. Hachtel et al., "On properties of algebraic transformations and the multifault testability of multilevel logic," in *Proceedings of the International Conference on Computer-Aided Design,* 1989, pp. 422–425.

[10] K. Hwang, *Computer Arithmetic: Principles, Architecture, and Design,* Wiley, New York, 1979.

[11] T. Lynch and E. Swartzlander, Jr., "A spanning tree carry lookahead adder," *IEEE Transactions on Computers,* Vol. C-41, August 1992, pp. 931–939.

[12] J. Rajski and J. Vasudevamurthy, "The testability-preserving concurrent decomposition and factorization of boolean expressions," *IEEE Transactions on Computer-Aided Design,* Vol. 11, June 1992, pp. 778–793.

[13] C. Thompson, "Area-time complexity for VLSI," in *Proceedings of the ACM Symposium on the Theory of Computation,* 1979, pp. 81–88.

[14] J. Ullman, *Computational Aspects of VLSI,* Computer Science Press, Rockville, Maryland, 1984.

[15] S. Unger, "Tree realizations of iterative circuits," *IEEE Transactions on Computers,* Vol. C-26, April 1977, pp. 365–383.

Biographies

MICHAEL J. BATEK received a B.S. degree in Computer Engineering (Honors) from the University of Illinois at Champaign/Urbana in 1987 and a M.S.E. degree in Computer Science and Engineering in 1992 from the University of Michigan at Ann Arbor. From 1989 to the present, he has been a research assistant at the University of Michigan, working in the areas of testing and logic synthesis while pursuing the Ph.D. degree. His other areas of interest include high-level and language-based test generation and computer architecture.

JOHN P. HAYES received the B.E. degree from the National University of Ireland, Dublin, and the M.S. and Ph.D. degrees from the University of Illinois, all in electrical engineering. He is currently a Professor of Electrical Engineering and Computer Science at the University of Michigan, Ann Arbor. He teaches and conducts research in the areas of computer-aided design and testing, computer architecture, VLSI design, and fault-tolerant computing. He is the author of several books including *Introduction to Digital Logic Design* (Addison-Wesley, 1993). Hayes is a Fellow of IEEE and a member of ACM and Sigma Xi.