JOURNAL OF ELECTRONIC TESTING: Theory and Applications 21, 631–649, 2005

The Coupling Model for Function and Delay Faults

JOONHWAN YI
Telecommunication Research Center, Samsung Electronics Corporation, Suwon, Kyunggi-Do, Republic of Korea
joonhwan.yi@samsung. com

JOHN P. HAYES
Department of Electrical Engineering and Computer Science, University of Michigan, 1301 Beal Avenue, Ann Arbor, MI 48109, USA
jhayes@eecs.umich.edu

Received June 11, 2003; Revised July 6, 2005

Editor: V. Agrawal

Abstract. We propose a high-level fault model, the coupling fault (CF) model, that aims to cover both functional and timing faults in an integrated way. The basic properties of CFs and the corresponding tests are analyzed, focusing on their relationship with other fault models and their test requirements. A test generation program COTEGE for CFs is presented. Experiments with COTEGE are described which show that (reduced) coupling test sets can efficiently cover standard stuck-at-0/1 faults in a variety of different realizations. The corresponding coupling delay tests detect all robust path delay faults in any realization of a logic function.

Keywords: delay faults, fault modeling, functional faults, test generation

1. Introduction

Gate-level faults, such as the standard *stuck-at line* (SSL) and path delay fault models, represent faulty behavior associated with logic gates and their interconnections. Consequently, they are not well suited to designs that contain higher-level modules whose implementation details are unavailable. In such cases, we need to check the faulty behavior of a module with respect to the module's overall specifications, and high-level fault models can play an important role. In addition, high-level testing can be used to correct testability problems in early design stages [19], or to test multiple implementations of a high-level design [14]. A module's specification is typically in the form of a Boolean expression, a truth table, a binary decision diagram, or a behavioral Verilog code description.

Several higher-level fault models have been proposed for realization-independent functional testing of arbitrary combinational modules, including the cell and pin fault models. A *cell fault* [20] implicitly models all defects that alter a module's specification and so provides a high degree of realization independence. This model can only be applied to very small modules, however, because it requires an exhaustive test set comprising all possible input vectors (patterns). For example, in the case of the n-bit *ADDER* module shown in Fig. 1(a), 2^{2n+1} tests are required to detect all cell faults. The *pin fault* model considers stuck-at-0/1 faults occurring at

This research was sponsored in part by the U.S. National Science Foundation under Grants No. CCR-9872066 and CCR-0073406.
A shorter version of this paper was published in the Proceedings of European Test Workshop, 2001. The single coupling fault (CF) and coupling delay fault (CDF) models of the earlier paper are extended here to the general CF and CDF models, and more analysis and discussion are presented.

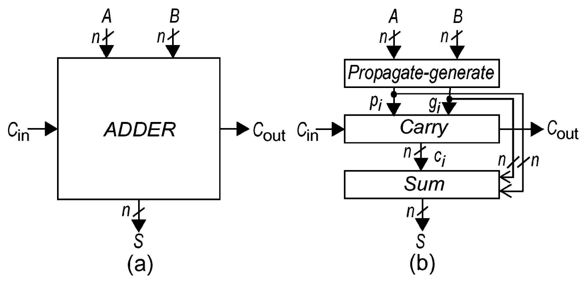


Fig. 1. (a) An n-bit adder module. (b) High-level structure of a carry-lookahead adder [14].

the module boundary, and has weak correlation with the circuit's physical faults. For *ADDER*, just two tests, the all-0 and all-1 vectors, suffice to detect the $6n+4$ possible pin faults. The SSL model lies somewhere between cell and pin faults, but is realization-dependent. To detect all SSL faults in a two-level implementation (not a practical design) of the n-input adder, 2^{2n+1} tests, that is, exhaustive testing, is required. However, $2n+2$ tests are sufficient to detect SSL faults in various practical adder implementations [2].

Another interesting approach to high-level testing is based on the concept of a *universal test set* [1, 4]. It exploits the unateness (monotonicity) property of a module's variables, and is composed of functionally-defined minimal true and maximal false tests. A universal test set can detect both single and multiple stuck-line (MSL) faults in realizations with minimal restrictions on their structure [14]. The size of the universal test set is small for functions that are fully or partially unate, but it becomes exhaustive for binate functions. For example, the universal test set is exhaustive for the *ADDER* module of Fig. 1(a) because the S (sum) outputs are binate in all $2n+1$ input variables. If a circuit is decomposed into suitable submodules and universal test sets are applied to the submodules, as in the RIBTEC technique [14], the resulting functional tests achieve a high degree of realization independence with $O(n^k)$ tests for some small k. The modular n-input carry look-ahead adder shown in Fig. 1(b), for instance, requires far fewer than 2^{2n+1} tests—it requires 44 tests when $n = 16$.

Few high-level delay fault models have been proposed so far. Pomeranz and Reddy [18] introduced the *gross* and *function-robust* delay fault models. As we show later in Section 4, the gross delay model covers all path delay faults. However, it requires all adjacent vector pairs as tests, which results in huge test sets ($n2^n$ tests for an n-input module). The function-robust delay fault model covers all robust path delay faults. Using the powerful functional fault coverage of a universal test set, Sparmann et al. [24] proposed the universal delay test set which, however, is only applicable to unate functions. Psarakis et al. [21] studied delay testing using what they call the *realistic sequential cell fault model* (RS-CFM). Their approach targets non-specific "sequential" faults such as stock-open faults in special classes of circuits (hazard-free iterative logic arrays). The RS-CFM model is not suitable for path delay fault testing because it is defined for an iterative logic array in which at most a single cell (or module) is faulty.

Digital systems are susceptible to both functional and timing faults due to increasing operating speeds and decreasing process feature sizes. This paper proposes a fault model, the *coupling fault* (CF) model, that covers both functional and timing faults in an integrated way, and is applicable to fairly large modules without imposing significant design constraints. A CF models (static) functional faults, but is motivated by (dynamic) transition effects, and so is easily extendible to a delay fault model. The corresponding coupling test sets share some properties with universal test sets, but are not necessarily exhaustive for binate functions. They also achieve high SSL fault coverage for a wide range of implementations. A pair of adjacent coupling tests (CTs) constitutes a *coupling delay test*, so the combinational CF model naturally extends to the sequential CF model. We show that coupling delay tests, which happen to correspond to the single input change (SIC) tests used in [21] and elsewhere, achieve high coverage of path delay faults. We also show that the *coupling delay test set* (CDTS) for a function z is a subset of other high-level delay test sets for z, e.g. the gross and function-robust delay test sets.

The size of a high-level test set is usually much larger than that of a realization-dependent one to assure good fault coverage for many realizations. Often this large size limits the application of high-level test sets to a small portion of a circuit. The interface region around the intellectual property (IP) circuits in a system-on-a-chip (SOC) is a good candidate for high-level delay testing because its implementation details are not known to either the SOC or the IP designers. For example, consider a high-level combinational module *CUT* at the interface of two *IP* circuits IP_1 and IP_2 in an SOC C, as shown in Fig. 2. *CUT* is composed of two modules, say M_1 and M_2, which are included in IP_1

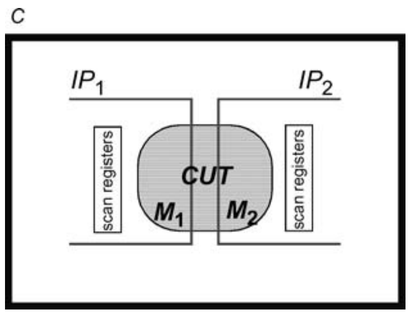


Fig. 2. A combinational module *CUT* at the interface of two *IP* circuits IP_1 and IP_2 in an SOC *C*.

and IP_2, respectively. Note that none of the designers of IP_1 and IP_2 knows the gate-level implementations of *CUT* fully because M_1 (M_2) is a high-level module to the designer of IP_2 (IP_1). It is thus very hard to detect faults in *CUT* using gate-level fault models. Design-for-testability (DFT) techniques [7, 13] can be used if the delay penalty due to DFT circuits is tolerable. Otherwise, high-level testing approaches appear very attractive if the scan design at IP circuit boundaries and high-level descriptions of $M_1$and M_2 are known, which constitutes a very limited disclosure in the case of IPs.

Section 2 of this paper formalizes coupling faults and tests. Functional testing is examined in Section 3, while Section 4 considers delay testing. A coupling-test generation program COTEGE is presented in Section 5. Finally in Section 6, some experimental results on the fault coverage of coupling tests are presented.

2. Coupling Faults and Tests

Intuitively, a CF alters output values in response to changes occurring on one or more inputs of a logic function. The simplest case is a *single CF* which is defined in terms of a single input/output signal pair. In contrast, a *general CF* is defined in terms of multiple inputs and a single output signal. Mainly single CFs are considered in this paper, and "CF" means a single CF unless otherwise noted.

Consider the module M in Fig. 3 which realises m Boolean functions $Z = \{z_1, z_2, \ldots, z_m\}$ with n input variables $X = \{x_1, x_2, \ldots, x_n\}$. The normal "coupling relationship" between any x_i in X and any z_j in Z can make z_j change from b to its complement b', denoted $b \rightarrow b'$, when x_i undergoes the change $a \rightarrow a'$ in some input vector v applied to M; see Fig. 3(a). Any change to this relationship is modeled by a CF denoted by $x_i \mid z_j$. That is, when the transition $a \rightarrow a'$ occurs in input x_i, the fault $x_i \mid z_j$ in M either makes z_j stable at b or b', or else produces the inverted transition $b' \rightarrow b$; see Fig. 3(b). Each of these possible faulty behaviors is revealed by some test vector that can propagate a signal transition from x_i to z_j when M is fault-free. All such vectors are represented by the Boolean difference of z_j with respect to x_i, which is defined as follows.

$$\frac{dz_j}{dx_i} = z_j(x_1, \ldots, x_i = 0, \ldots, x_n) \oplus z_j(x_1, \ldots, x_j = 1, \ldots, x_n) \qquad (1)$$

Any vector v making $\frac{dz_j}{dx_i} = 1$ can be affected by a CF of the form $x_i \mid z_j$, leading to the following definition.

Definition 1 (Coupling Fault). A coupling fault $x_i \mid z_j$ from x_i to z_j in module M changes the value of z_j for at least one input vector satisfying $\frac{dz_j}{dx_i} = 1$

Observe that this definition is realization-independent and combinational (static); detection of a CF requires a single test vector not a vector-pair.

We refer to the set of all test vectors for a CF set F as its *coupling test set* (CTS). We also follow the common practice of using a Boolean function $CTS_F(X)$ to

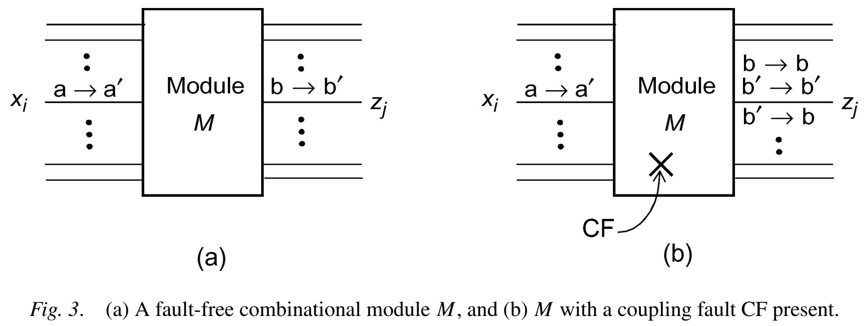


Fig. 3. (a) A fault-free combinational module M, and (b) M with a coupling fault CF present.

represent test sets, where $CTS_F(v) = 1$ if and only if v is a test for some fault in F. Each member of the CTS thus corresponds to a minterm of $CTS_F(X)$. To detect $x_i \mid z_j$, we need to apply all input vectors satisfying $\frac{dz_j}{dx_i} = 1$. Therefore the set of all CTs for $x_i \mid z_j$is given by

$$CTS_{x_i \mid z_j}(X) = \frac{dz_j}{dx_i} \tag{2}$$

Each function $z_j(X)$ in Z has n CFs $x_1 \mid z_j, x_2 \mid z_j, \ldots, x_n \mid z_j$, so the CTS for all such faults F in Z is

$$CTS_F(X) = \sum_i \sum_j \frac{dz_j}{dx_i}$$

The n-input identity function I_n is composed of two minterms separated by the maximum Hamming distance n. For example, $I_4 = abcd + a'b'c'd'$ and $CTS_{a \mid I_4}(a, b, c, d)$ can be represented by

$$\frac{dI_4}{da} = bcd + b'c'd' \tag{3}$$

Hence the corresponding CTS is {1111, 0111, 1000, 0000}. The complete CTS for I_4 consists of the 10 vectors indicated by shading in Fig. 4.

The coupling test set $CTS_{x_i \mid z_j}(X)$ is independent of x_i since $\frac{dz_j}{dx_i}$ is independent of x_i. This implies that if $v = x_1 \ldots x_{i-1} 0 x_{i+1} \ldots x_n$ is a CT, so is the *adjacent CT* $v^* = x_1 \ldots x_{i-1} 1 x_{i+1} \ldots x_n$ obtained by complementing x_i. Note that $v = x_i' \cdot s$ and $v^* = x_i \cdot s$, where s satisfies $\frac{dz_j}{dx_i} = 1$. Thus, $z_j(v^*) = z_j'(v)$ from the definition of the Boolean difference. For example, $abcd$ is an adjacent CT to the CT $a'bcd$ for input a in I_4. Conversely, if two input vectors v and v^* are adjacent and yield different outputs, then v and v^* are adjacent

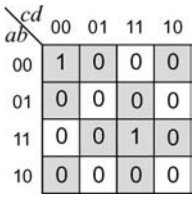


Fig. 4. The four-input identity function $I_4 = abcd + a'b'c'd'$ with coupling tests shaded.

CTs. Clearly, all CTs lie on the boundary between the 0's and 1's in the K-map of the target function z_j; see Fig. 4. It also follows that the CTs completely define z_j.

A pair $\langle v, v^* \rangle$ of adjacent CTs for input x_i and output z_j can serve to detect a delay fault that causes excessive delay in propagating a signal transition on x_i to z_j. The consecutive application of v and v^* activates a signal transition on x_i which propagates through the module to z_j. As in conventional delay testing, we can check for an associated path delay fault by monitoring z_j after applying $\langle v, v^* \rangle$.

Definition 2 (Coupling Delay Fault). A module M contains a coupling delay fault $f = x_i - z_j$ from x_i to z_j if a signal transition on x_i fails to propagate to z_j within a specified time. The corresponding coupling delay test set (CDTS) is the set of all adjacent CT pairs $\langle v, v^* \rangle$ of $x_i \mid z_j$.

Suppose that the probability of a vector v of z being a minterm (or maxterm) is 2^{-1}. Vector v has n adjacent vectors. If v is not a CT, all n adjacent vectors must produce the same output value $z(v)$. The probability that v is not a CT is then $(2^{-1})^n = 2^{-n}$, that is, the probability that a vector of z is a CT is $1 - 2^{-n}$ Since there are 2^n possible vectors, we conclude that the average size of CTS_z is $2^n \cdot (1 - 2^{-n}) = 2^n - 1$.

Although the average size of a CTS is large, many useful functions require small CTSs. The elementary n-input (gate) functions, i.e. AND, NAND, OR, and NOR, require $n + 1$ CTs which are identical to the minimal SSL fault test sets. This implies that test generation for CFs reduces to the standard ATPG problem when applied to gate-level circuits. Also, as shown in Section 6, many arithmetic functions (adders, ALUs and comparators) have CTSs of reasonable size.

Table 1 shows various coupling test sets for some elementary (gate) functions, including the identity function. In the XOR case, the CTS is exhaustive; a two-level XOR circuit (usually impractical) requires all input vectors to detect all SSL faults. However, a practical n-input XOR tree circuit requires only four tests for any n [11]. In the case of the identity function I_n, the CTS size increases linearly with n and is identical to the SSL fault test set for a two-level sum-of-products (SOP) implementation of I_n. Observe that the universal test set for I_n is exhaustive because I_n is fully binate. Using the Synopsys Design Compiler, we synthesized two different implementations of the 20-input identity function I_{20} as shown in Fig. 5; one is optimized for low area and the other for high speed. The conventional ATPG

Table 1. Coupling test sets and their sizes for some basic Boolean functions.

Function $z(x_1, \ldots, x_n)$		CTS_z	Test set size: Coupling test set	Test set size: Coupling delay test set	Test set size: Minimal SSL fault test set
AND	$n = 2$	$x_1 + x_2$	3	4	3
	n	$x_2x_3 \ldots x_n + x_1x_3 \ldots x_n + \cdots + x_1x_2 \ldots x_{n-2}x_n$ $+x_1x_2 \ldots x_{n-1}$	$n + 1$	$2n$	$n + 1$
OR	$n = 2$	$x'_1 + x'_2$	3	4	3
	n	$x'_2x'_3 \ldots x'_n + x'_1x'_3 \ldots x'_n + \ldots + x'_1x'_2 \ldots x'_{n-2}x'_n$ $+x'_1x'_2 \ldots x'_n-1$	$n + 1$	$2n$	$n + 1$
XOR	$n = 2$	1	4	8	4
	n	1	2^n	$n2^n$	4 (XOR tree)
Identity	$n = 4$	$x_2x_3x_4 + x'_2x'_3x'_4 + x_1x_3x_4 + x'_1x'_3x'_4 + x_1x_2x_4$ $+x'_1x'_2x'_4 + x_1x_2x_3 + x'_1x'_2x'_3$	10	16	10 (SOP)
	n	$x_2x_3 \ldots x_n + x'_2x'_3 \ldots x'_n + \cdots + x_1x_2 \ldots x_{n-1} + x'_1x'_2 \ldots x'_{n-1}$	$2n + 2$	$4n$	$2n + 2$ (SOP)

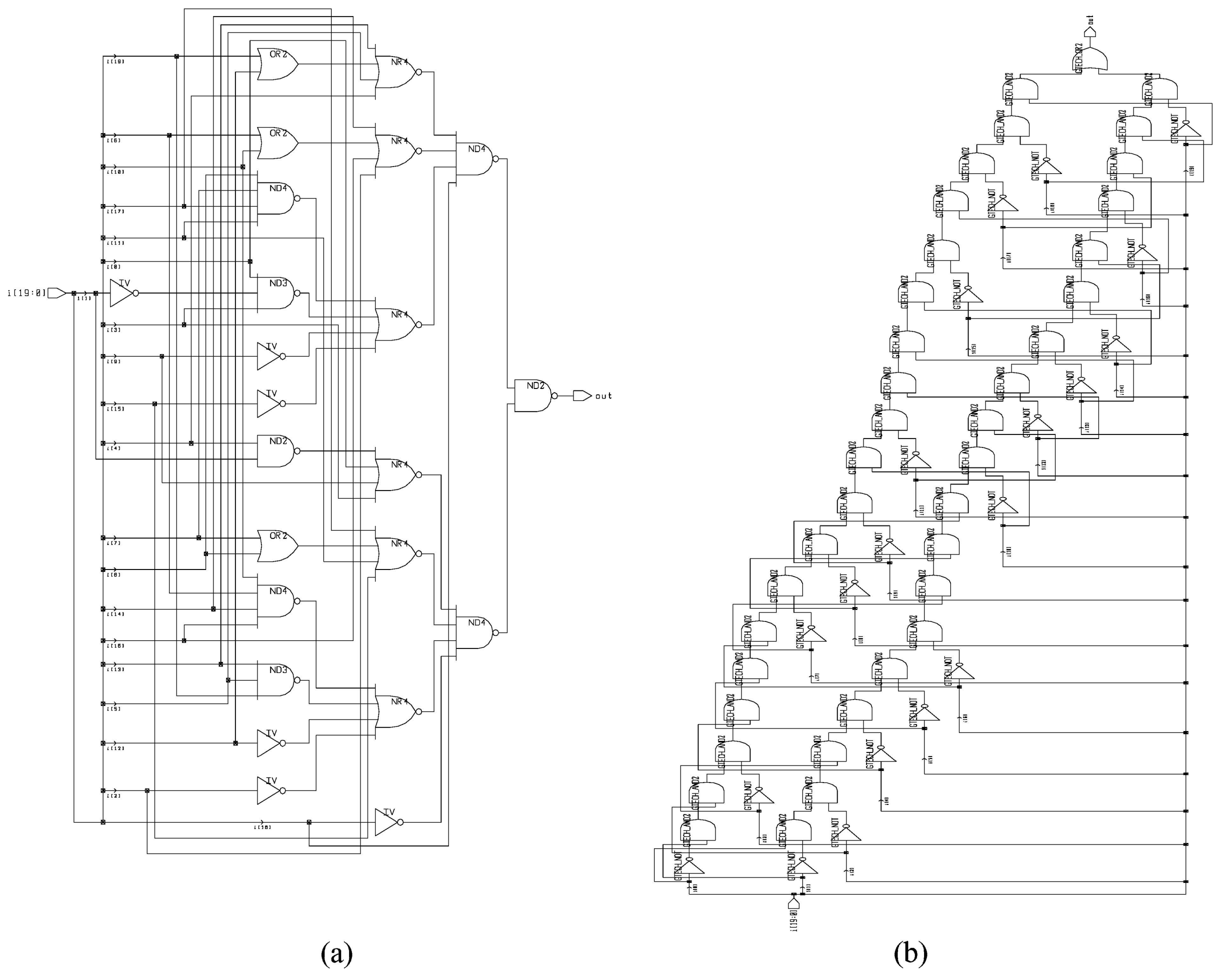


Fig. 5. Two gate-level realizations of the 20-input identity function I_{20} synthesized for (a) high speed, and (b) low area.

tool ATALANTA [16], generates 42 tests for both implementations; these tests happen to be identical to the CTs for I_{20}.

2.1. General Coupling Faults and Tests

Consider the module M shown in Fig. 3 which realizes m Boolean functions $Z = \{z_1, z_2, \ldots, z_m\}$ with n input variables $X = \{x_1, x_2, \ldots, x_n\}$. A general CF $X_i \mid z_j$ blocks the coupling between a set of inputs $X_i = \{x_{i1}, x_{i2}, \ldots, x_{ik}\} \subseteq X$ and an output z_j in M. Assume that z_j changes $b \rightarrow b'$ when a subset X_i of X changes from $A = l_1 l_2 \ldots l_h \ldots l_k$ to $B = l'_1 l'_2 \ldots l'_h \ldots l'_k$ where l_h is either x_{ih} or x'_{ih}. If M has CF $X_i \mid z_j$ then z_j remains at b or b', or else it changes $b' \rightarrow b$ when X_i changes $A \rightarrow B$. A general CF and its test set are similarly defined as follows.

Definition 3 (General Coupling Fault). A general coupling fault $X_i \mid z_j$ from a set of inputs X_i to an output z_j in module M changes the value of z_j for at least one input vector satisfying

$$G = \frac{d}{dx_{i1}}\left(\frac{d}{dx_{i2}}\left(\cdots\left(\frac{dz_j}{dx_{ik}}\right)\right)\right) = 1 \qquad (4)$$

It follows that $CTS_{X_i \mid z_j} = G$ where $CTS_{X_i \mid z_j}$ is the CTS for $X_i \mid z_j$. A general CF $X_i \mid z_j$ is called a *k-input CF* because X_i is composed of k inputs. For example, a single-input CF is a single CF. The CTS for all k-input CFs of z_j contains all minterms v and maxterms v^* of z_j where v and v^* are separated by Hamming distance k. That is, the general CTS for z_j contains all minterms and maxterms separated by Hamming distances from 1 to n, which results in an exhaustive test set. Therefore, general CFs can model any physical defect that changes the function under test, like the cell fault model [20]. They provide strong realization independence but their application is restricted to small functions.

Consider a minterm-maxterm pair $t = \langle v, v^* \rangle$ where v and v^* are tests for $X_i \mid z_j$ and they are different in every input of X_i. Then t can serve to detect a delay fault that causes excessive delay in signals that propagate from X_i to z_j.

Definition 4 (General Coupling Delay Fault). A module M contains a general coupling delay fault X_i–z_j from X_i to z_j if any signal transitions on X_i fail to propagate to z_j within a specified time.

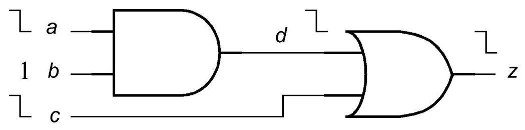


Fig. 6. The application of a test $t = \langle abc, a'bc' \rangle$ to a gate-level implementation of $z = ab + c$.

The corresponding general CDTS is the set of all minterm-maxterm pairs $\langle v, v^* \rangle$ where v and v^* are tests for $X_i \mid z_j$ and they are different in every input of X_i. A general coupling delay fault (CDF) X_i–z_j is called a *k-input CDF*. The CDTS for all k-input CDFs of z_j contains all minterm-maxterm pairs $\langle v, v^* \rangle$ separated by Hamming distance k. Thus, the general CDTS for z_j is composed of all possible adjacent and non-adjacent minterm-maxterm pairs of z_j.

A multiple-input CDT t propagates signal transitions through multiple input-output paths. Some single-input CDFs may be detected by t and thus some single-input CDTs can be replaced by t. For example, consider path delay faults in the implementation of function $z = ab + c$ appearing in Fig. 6. A two-input CDT $t = \langle abc, a'bc' \rangle$ propagates falling signal transitions on a and c to z, and the latest signal transition on an input of OR gate will change z from 1 to 0. Therefore, t detects slow-to-fall delay faults associated with both paths adz and cz. Since two single CDTs $t_a = \langle abc', a'bc' \rangle$ and $t_b = \langle a'bc, a'bc' \rangle$ also detect these two faults, t_a and t_b can be eliminated from the general CDTS of z. The relationship between single- and multiple-input CDTs is discussed further in [25].

3. Properties of Coupling Tests

In this section, coupling tests are related to test sets for SSL faults and high-level functional fault models. We derive a design constraint that ensures full coverage of SSL faults in a two-level circuit by the corresponding CTS.

A sum of products (SOP) *cover E* of z is a set of implicants $q_1, q_2, \ldots, q_k$ of z such that $z = q_1 + q_2 + \cdots + q_k$. Each implicant q_i corresponds to an AND gate in a two-level AND-OR realization of z. An implicant q_i of E is *relatively essential* if the result of deleting q_i from E covers fewer minterms of z than E. A cover is *non-redundant* if all implicants in the cover are relatively essential. Consider an implicant $q = l_1 l_2 \ldots l_i \ldots l_k$ in a non-redundant cover E of z. Literal l_i in q is a *prime literal* [12] if the product term $r = l_1 l_2 \ldots l_{i-1} l_{i+1} \ldots l_k$

obtained by deleting l_i from q is not an implicant of z; clearly, all literals in a prime implicant are prime. An input vector v is *relatively essential* [12] in q if $q(v) = 1$ but $q^+(v) = 0$ for some other implicant $q^+ \neq q$ in E. A relatively essential implicant contains at least one relatively essential vector. A cover corresponding to a non-redundant SOP realization of z is composed of relatively essential prime implicants.

We now specify a design constraint to ensure that the CTS for z detects all SSL faults in any nonredundant SOP realization C of z. Let E be the cover corresponding to C. Although we only consider SOP covers explicitly, the results hold for product-of-sums (POS) covers by the duality principle.

Lemma 1. *A test set T detects all SSL faults in a non-redundant SOP circuit C if T includes*

(1) *At least one relatively essential vector v of every prime implicant q in E, and*
(2) *At least one false vector w adjacent to a minterm of q for every literal l_i in q.*

Proof: Let AND gate G in C realize q and let input i of G correspond to l_i. The proof is done by showing that v detects the fault i stuck-at-0 and w detects i stuck-at-1. First, consider i stuck-at-0. Input v activates this fault by setting all inputs of G to 1 and propagates an error to z by setting the output of every other AND gate $G^* \neq G$ to 0. Now consider i stuck-at-1. Since w is adjacent to a minterm of q at l_i, w sets i to 0 (fault activation) and all other inputs of G to 1. In addition, w forces the output of G^* to 0 because w is a maxterm. That is, w activates i stuck-at-1 and propagates a faulty value to z. □

Every false vector adjacent to a minterm is a CT, that is, a CTS always satisfies condition (2) of Lemma 1. To meet condition (1), a CTS must contain at least one relatively essential vector from every prime implicant q in E, which implies that q must have at least one relatively essential vector (minterm) that is adjacent to a false vector (maxterm). This leads to the following sufficient condition for the CTS to detect all SSL faults in a two-level circuit

Theorem 1. *Let C be a non-redundant two-level circuit realizing an SOP cover E of z. The coupling test set CTS_Z detects all SSL faults in C if every prime implicant in E has at least one relatively essential vector that is adjacent to a false vector.*

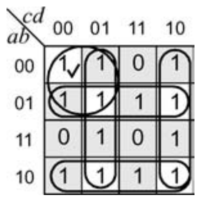


Fig. 7. Non-redundant cover $E = \{ab', a'b, cd', c'd, a'c'\}$ for the 4-input function $(abcd + a'b'cd + abc'd')'$ with coupling tests shaded.

A two-level circuit that is testable for robust path delay faults [8] satisfies the design constraint specified in Theorem 1. Although it appears to be uncommon, some functions have non-redundant covers that fail to satisfy this constraint. For example, I_4 has some non-redundant POS covers that violate the constraint although all minimal SOP and POS covers satisfy it. Another example is shown in Fig. 7. Here the non-redundant cover $E = (ab', a'b, cd', c'd, a'c')$ is not fully testable by the function's CTS. The checked vector $a'b'c'd'$ is an essential SSL test for the two-level circuit realizing E but is not a CT. However, there is another minimal POS cover $(a' + b' + c' + d', a + b + c' + d', a' + b' + c + d)$ for this function that meets the foregoing constraint. We also found a 6-input function all of whose non-redundant covers violate this constraint:

$$\begin{aligned} z = {} & ab'ef + a'bef + cd'ef + c'def + acef \\ & + c'd'ef' + cdef' + c'd'e'f + cde'f \\ & + a'b'c'de'f' + a'b'cd'e'f' + abcd'e'f' \end{aligned}$$

The coupling test set CTS_Z for z has at least one relatively essential vector from 11 out of 12 prime implicants, which implies that most SSL faults are detected by CTS_Z. In fact, CTS_Z detects 91 faults out of 92 possible (collapsed) faults in all eight possible SOP realizations of z.

Most practical circuits are not SOP realizations. A *test-set preserving* logic transformation [3] L for a fault type F transforms a circuit N_1 to N_2 denoted $L(N_1) = N_2$ such that a test set for N_1 detects all faults of type F in N_2. Some important logic synthesis techniques such as algebraic factorization and resubstitution are test-set-preserving for SSL faults [22].

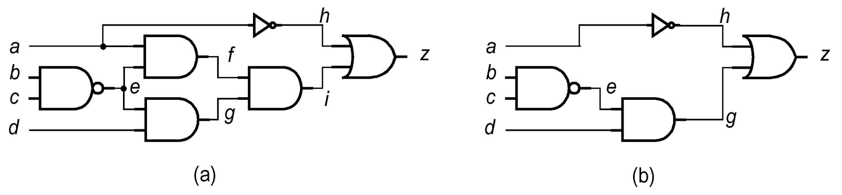


Fig. 8. (a) A non-BIP implementation C_0 of the function $z = a' + b'd + c'd$, and (b) the BIP implementation C_0^* of z obtained by removing gates and lines from C_0.

Let C be a two-level realization of a function z that meets the design constraint given in Theorem 1. Then, the coupling test set for z also detects all SSL faults in a multiple-level circuit derived from C by test-set-preserving synthesis techniques. As will be seen in our experimental results, the coupling test sets show high SSL fault coverage for a wide range of multilevel implementations.

CFs are also related to other existing fault models. Although pin faults are weak, as observed in Section 1, pin fault tests sometimes detect hard-to-detect (random-test resistant) faults. For example, the two all-0 and all-1 pin fault tests for an n-bit adder sensitize the long internal path from C_{in} to C_{out} used for carry propagation. In general, we can rewrite (2) as

$$CTS_{x_i\,|\,z_j}(X) = x_i\frac{dz_j}{dx_i} + x_i'\frac{dz_j}{dx_i} \tag{5}$$

Since $x_i\frac{dz_j}{dx_i}$ represents all tests that detect x_i stuck-at-0 via z_j and $x_i'\frac{dz_j}{dx_i}$ represents all tests that detect x_i stuck-at-1 via z_j, $CTS_{x_i\,|\,z_j}(X)$ must contain all tests for the SSL faults on x_i. Hence $CTS_{x_i\,|\,z_j}(X)$ includes all tests for pin faults.

Finally, we establish a relationship between coupling tests and the universal test set (UTS) [1, 4] for unate functions. The UTS covers all SSL faults in realizations that meet minor and practical constraints on inversion parity, such as the balanced inversion parity (BIP) constraint, and the slightly more restrictive unate-gate network constraint [14]. A *BIP network* is a gate-level circuit in which, whenever possible, all paths from any input variable to an output have the same inversion parity—the inversion parity of a path is N modulo 2, where N is the total number of inversions on the path.

Consider a function z whose positive and negative cofactors with respect to a variable x are respectively denoted by $z\,|_x$ and $z\,|_{x'}$ hence $z = x \cdot z\,|_x + x' \cdot z\,|_{x'}$. Function z is positive unate in x if $z\,|_x \supseteq z\,|_{x'}$ and negative unate if $z\,|_x \subseteq z\,|_{x'}$. Otherwise, z is binate in x. Function z is unate if all its variables are unate; otherwise, z is binate. It is shown in [14] that any logic function z can be realized such that all paths from a unate variable of z have the same inversion parity, while at least one path from a binate variable must have a different inversion parity from other paths. BIP realizations cover a broad range of implementations and tend to be minimal or near-minimal in gate count because any inversion that violates the BIP condition can be easily removed without adding gates. The authors of [14] report that all synthesized (gate-level) circuits used in their experiments meet the BIP condition with no design changes. For example, Fig. 8(a) is a non-BIP implementation C_0 of the function $z = a' + b'd + c'd$ because there is an even inversion path *afiz* from a to z, and z is negative unate in a. The BIP implementation C_0^* of z shown in Fig. 8(b) is obtained by removing the line between a and f two redundant gates from C_0.

An input vector v_i is *greater than or equal to* another vector v_j, denoted $v_i \geq v_j$, if v_i has 1 in every bit position where v_j does. For example, $1110 \geq 0100$. The minimal true vectors (v_i) of z are input vectors such that $z(v_j) = 1$ for all $v_j \geq v_i$, but $z(v_j) = 0$ for all $v_j < v_i$. Maximal false vectors are defined similarly. *Expanded vectors* contain all input literals, both true and complemented, appearing in a function's minimal SOP expression [1]. For example, consider a function $z(a, b, c) = ab + a'c'$ that is binate in a, positive unate in b, and negative unate in c. The expanded vector of $abc = 111$ is $aa'bc' = 1010$.

The UTS for a function z is composed of all its maximal false and minimal true expanded vectors. It is shown in [4] that the UTS is minimal in the sense that no test set detects all SSL faults in a non-redundant two-level realization of a positive unate function with fewer tests than the UTS.

Theorem 2. *The coupling test set CTS_z for a unate function z contains the universal test set UTS_z for z, that is, $CTS_z \supseteq UTS_z$.*

Proof: Without loss of generality, assume that z is positive unate in all its variables. Then the UTS for z is composed of all maximal false and minimal true vectors of z, and every minterm m of z must be expressible as a product containing at least one uncomplemented literal. Assume that m is a minimal true vector that has no adjacent false vector. Then m is surrounded by true vectors, so an adjacent true vector m^* must be obtained by complementing an uncomplemented literal from m. Since z is positive unate, $m > m^*$. This contradicts to the assumption that m is a minimal true vector. Therefore, m must have an adjacent false vector. Similarly, every maximal false vector is also adjacent to at least one true vector. All adjacent false and true vectors are CTs, so all maximal false and minimal true vectors are CTs. □

This result implies that the CTS detects all SSL faults in BIP-type realizations of a unate function. Typically, only small portions of useful circuits are unate. In the case of binate functions, there is no obvious relation between the CTS and the UTS. While the UTS is always exhaustive for fully binate functions, the CTS can be smaller, as demonstrated by the identity function I_n.

Theorem 2 also implies that a CTS may contain some unnecessary or redundant tests for BIP-type realizations. From the definition of a UTS, any true (false) expanded vector that is greater (less) than some other true (false) expanded vector is an unnecessary test for BIP circuits, regardless of the function's unateness. Let a CT v be true if v is a true vector and false if v is a false vector. The expanded form of a CT is called an *expanded CT*. Then for BIP circuits, a CTS can be reduced by removing all true expanded CTs greater than other true expanded CTs, and all false expanded CTs less than other false expanded CTs.

Definition 5 (Reduced Coupling Test Set). The reduced CTS for a function z is the set of minimal true and maximal false expanded CTs of z.

As will be seen in Section 6, reduced CTSs are used in our experiments, and they show high SSL fault coverage for a wide range of practical implementations.

4. Properties of Coupling Delay Tests

Path-based models are widely accepted as realistic models of delay faults in gate-level logic circuits. The conventional path delay model [23] associates faults with single paths. On the other hand, some approaches [9, 12] associate delay faults with both single and multiple paths. In this section, we relate the coupling delay fault model to the conventional path [23] and so-called primitive [12] delay fault models.

A *single path* p is an alternating sequence of lines and gates from a primary input to a primary output. Consider a gate G lying on p. An input i of G is called an *on-path input* of p if i is on p; otherwise, i is an *off-path input*. A *multiple path* P is a set of single-paths that share the same primary output [12]. A (single) *path delay fault* $f_p = \uparrow p$ ($\downarrow p$) increases circuit delay beyond some acceptable threshold when a signal transition is propagated through p, and causes a rising (falling) transition at the primary output of p. Tests for path delay faults are divided into two main categories: robust and non-robust. The application of a *robust test* t for f_p will cause an incorrect output to be measured in the presence of f_p, independent of the delays in the rest of the circuit under test; f_p is then called a *robust* (*path*) *delay fault*. All other tests are *non-robust* and the corresponding faults are *non-robust delay faults*. A path p is *static sensitized* by a vector v if v sets all off-path inputs of p to non-controlling values. If $t = \langle v_1, v_2\rangle$ is a test for f_p,p is static sensitized by v_2 and a proper signal transition is initiated at the input of p.

For example, $t_n = \langle a'b, ab\rangle$ is a robust test for fault $\downarrow adgh$ in the circuit of Fig. 9(a). The signal edge with an arrow represents the signal transition propagated through the path under test. As can be seen in Fig. 9(a), there are two possible output waveforms. In either case, if fault $\downarrow adgh$ is present, t_n detects the fault because the sampling time will precede the edge with the arrow, and the sampling value is 1 rather than the expected value 0. On the other hand, t_n is also a non-robust test for a fault $\downarrow adfh$ as shown in Fig. 9(b). If the output is sampled at time t_1 or t_3, the fault is detected. However, if the output is sampled at time t_2, the fault is not detected. The glitch at the output h exists under the assumption that paths *acfh* and *adgh* are fault-free. If either of these two paths is faulty, no falling edge at h can be propagated through the path under test *adfh* and thus t_n is invalidated.

The on- and off-path signal conditions for robust and non-robust delay tests are summarized in Table 2, which is adapted from tables in [5] and [17]. According to these conditions, a robust test causes a signal transition whose initial and final values are different at on-path inputs and thus at the path output. That is,

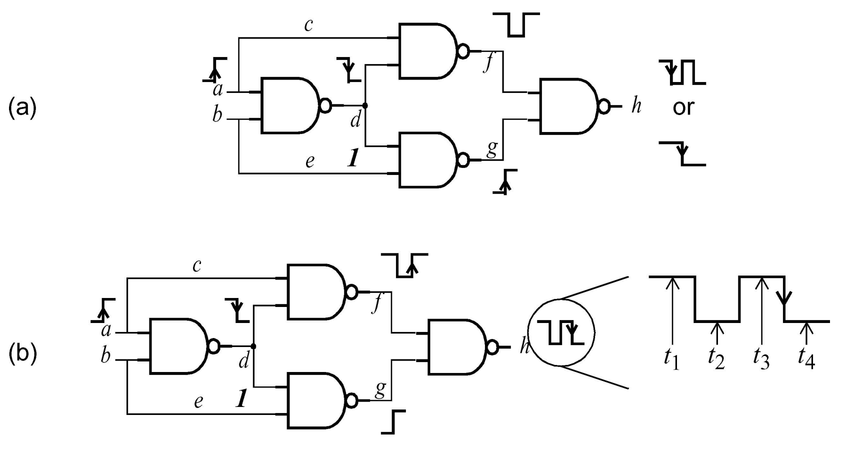


Fig. 9. A realization of the 2-input XOR function. Test $\langle a'b, ab\rangle$ is (a) a robust test for $\downarrow$ *adgh* and (b) a non-robust test for $\downarrow$ *adfh*.

two vectors in a robust test must yield different output values.

A non-robust test t is said to be *validatable non-robust* (VNR) if and only if there exists a set of tests T including t such that no element in T is a robust test for the target fault f_p, and the correct response to T implies that f_p is not present in the circuit under test [12]. Robust and VNR tests guarantee detection of the target fault, while non-robust tests do not. Both robust and non-robust delay faults are called singly-testable (ST) faults in [9]. An *ST-dependent* fault is a path delay fault that does not cause excessive delay in the absence of a certain ST fault [9]. Thus, an ST-dependent fault does not need to be tested.

It is well-known that every detectable path delay fault has a *single input change* (SIC) test t [9, 23] in which the two input vectors of t differ only in a single input variable.

Lemma 2. *The set T_{SIC} of all SIC tests is necessary and sufficient to detect all (single) path delay faults in any realization of a function z.*

Table 2. Delay test classification in gate-level circuits. (*cv* and *ncv* denote controlling and non-controlling values, respectively. X denotes don't care values including glitches).

	On-path Input	
Off-path input	$cv \to X \to ncv$	$ncv \to X \to cv$
$X \to ncv$	Robust	Non-robust
Stable *ncv*	Robust	Robust

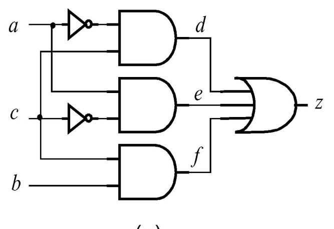
Proof: *Sufficiency*: By way of contradiction, assume that a path delay fault $f_p = \uparrow p$ or $\downarrow p$ only has a multiple-input-change test $t = \langle v_1, v_2\rangle$. This implies that at least one input x_k in t changes, where x_k is not the primary input of p. Let b_1 and b_2 denote the values of x_k in v_1 and v_2, respectively, where $b_2 = b_1'$. Then, v_2 must set all off-path inputs to non-controlling values. If b_2 makes every off-path input have a non-controlling value, the test $t^* = \langle v_1^*, v_2\rangle$ is also a test for f_p where v_1^* is obtained by replacing b_1 with b_2 in v_1. Similarly, we can make all other inputs except for the primary input of p stable. That is, f_p has an SIC test, which is contradictory to the initial assumption. So, every path delay fault has an SIC test and thus T_{SIC} detects all path delay faults in any realization of z.

Necessity: This part of the proof follows from the fact that the sum-of-minterms realization of z requires all SIC tests to detect all target faults. □

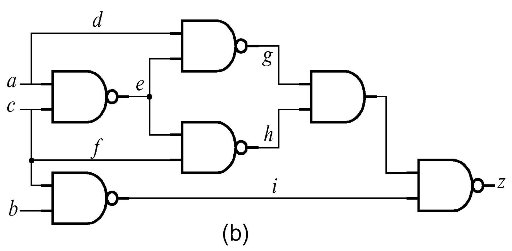
Note that the necessary test set for gross delay faults is the set T_{SIC} of all SIC tests [19]. The CDTS T_{CDF} is the set of SIC tests of which the two vectors yield different outputs, that is, $T_{\text{CDF}} \subseteq T_{\text{SIC}}$. Also the test set T_{FR} for function-robust delay faults contains all SIC tests that yield different outputs [19], that is, $T_{\text{CDF}} \subseteq T_{\text{FR}}$.

Theorem 3. *The CDTS for a function z detects all robust path delay faults in any realization of z.*

Proof: Consider a robust path delay fault $f_r = \uparrow p$ ($\downarrow p$) in a realization of z. Then f_R must have an SIC test $t_R = \langle v_1, v_2\rangle$ where v_1 and v_2 differ only in p's primary input. Note that t_R propagates a signal transition



(a)



(b)

Coupling delay test set $T_D(z)$

t_1: <*0*10, *1*10>,	t_2: <*1*10, *0*10>
t_3: <*1*01, *0*01>,	t_4: <*0*01, *1*01>
t_5: <1*0*1, 1*1*1>,	t_6: <1*1*1, 1*0*1>
t_7: <00*0*, 00*1*>,	t_8: <00*1*, 00*0*>
t_9: <01*0*, 01*1*>,	t_{10}: <01*1*, 01*0*>
t_{11}: <10*1*, 10*0*>,	t_{12}: <10*0*, 10*1*>

(c)

Path delay faults in realization (b)

Robust:	↑adgz (t_1), ↓adgz (t_2), ↑cfhz (t_7, t_{10}), ↓cfhz (t_8), ↓aehz (t_4), ↓cegz (t_{12}), ↑biz (t_5), ↓biz (t_6), ↓ciz (t_{10})
NR:	↑aehz (t_3), ↓aegz (t_4), ↑cegz (t_{11}), ↓cehz (t_{12})
ST-dependent:	↑ciz (t_9)
Redundant:	↑aegz, ↑cehz

(d)

Fig. 10. (a) A 2-level BIP realization of $z = ac' + a'c + bc$, (b) a multi-level BIP realization of z, (c) coupling delay test set T_D(z) for z, and (d) path delay faults and their tests (in parentheses) for realization (b).

through p and causes a rising (falling) transition on the path output. Test t_R is robust, so v_1 and v_2 must always yield different outputs. Since t_R is an SIC test and v_1 and v_2 yield different output values, t_R is a coupling delay test. Therefore the CDTS for z detects f_R □

It is worth noting that, in practical designs, the CDTS contains many non-robust tests. For example, Fig. 10 shows two realizations of the 3-input function $z = ac' + a'c + bc$ and the coupling delay test set $T_D(z)$. In the 2-level realization of Fig. 10(a), all 12 path delay faults are robust and are detected by $T_D(z)$. In the multi-level realization of Fig. 10(b), there are nine robust and four non-robust (NR) path delay faults with two redundant and one ST-dependent faults. All of these faults are detected by $T_D(z)$ as shown in Fig. 10(d). Although it is not necessary to test the ST-dependent fault $\uparrow ciz$, $T_D(z)$ has a test $t_9 = \langle 010, 011\rangle$ that propagates a rising signal transition through path ciz to z, which may detect a fault in other realizations.

We now extend the result of Theorem 3 to transition faults and stuck-open faults. A (single) transition fault $f = \uparrow l$ ($\downarrow l$) on a line l in a circuit C causes excessive delays so that a rising (falling) signal transition on l fails to propagate to any output z_j. Let P_l be a partial path from l to z_j. A transition fault test $t = \langle v_1, v_2\rangle$ for f is composed of two input vectors v_1 and v_2 where v_2 static sensitizes P_l and t initiates a proper signal transition, that is a rising (falling) signal transition, on l. To make t robust, all on- and off-path inputs of P_l must meet the conditions given in Table 2. This implies that v_1 and v_2 must yield different outputs if t is robust. Similarly to Lemma 2, we can prove that a robust transition fault has an SIC test. Because two vectors v_1 and v_2 in a robust test $t = \langle v_1, v_2\rangle$ yield different outputs, we can easily show that the CDTS of a function module M detects all robust transition faults in any realization of M.

Theorem 4. *The CDTS of a function z detects all robust transition faults in any realization of z.*

Consider a robust test t for a path delay fault $f = \uparrow p$. Then, t is a robust test for a transition fault associated with a line l on p. For example, robust test $t_1 = \langle 010, 110\rangle$ for path delay fault $\uparrow adgz$ in Fig. 10(b) is also a robust test for transition faults $\uparrow a$, $\uparrow d$, $\downarrow g$, and $\uparrow z$. The robust path delay tests (t_1, t_2, t_4, t_5, t_6, t_7, t_8, t_{10}, t_{12}) in Fig. 10(d) robustly detect all transition faults except $\uparrow e$ in Fig. 10(b). The transition fault $\uparrow e$ is robustly detected by two CDTs t_3 and t_{11} as can be seen in Fig. 11.

A stuck-open fault f at a transistor tr of a gate G in a CMOS logic circuit can adversely affect the delay of G in that a signal transition through tr can fail to propagate to an output z_j through a partial path P_l in the required time. Thus a test t for f is also composed of two input vectors. To make t robust, all on- and off-path inputs of P_l must meet the conditions given in

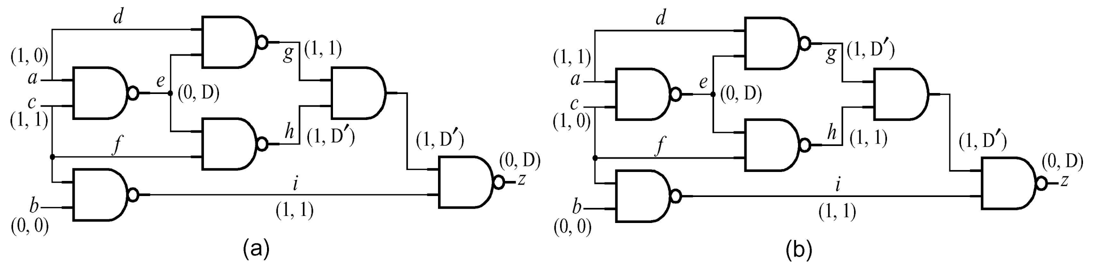


Fig. 11. Demonstration that transition fault $\uparrow e$ in Fig. 10(b) is robustly detected by two CDTs; (a) $t_3 = \langle(101, 001)\rangle$ and (b) $t_{11} = \langle(101, 100)\rangle$.

Table 2. This implies that v_1 and v_2 must yield different outputs if t is robust. As in the above discussion on the transition fault coverage of CDTSs, we can easily show that every stuck-open fault in a functional module M has an SIC test, and thus the CDTS of M detects all robust stuck-open faults in any realization of M.

Theorem 5. *The CDTS of a function z detects all robust stuck-open faults in any realization of z.*

Next we derive a sufficient condition for a CDTS to detect primitive delay faults in two-level circuits. A *multiple-path delay fault* [12] $F_P = \uparrow P$ ($\downarrow P$) represents the condition in which every single path $p \in P$ has a fault $\uparrow p$ ($\downarrow p$). Multiple path P is static sensitized by a vector v if v sets all side inputs of P to noncontrolling values. F_P is *primitive* if P is static sensitizable, and there is no sub-path of P that is similarly static sensitizable. Primitive delay faults include both robust and non-robust (single and multiple) path delay faults. If all primitive delay faults in a circuit C have either a robust or a VNR test, these tests form a necessary and sufficient test set to guarantee that C performs correctly at or below the test clock frequency [12].

We restrict our discussion to single-output circuits here. As defined in Section 3, a literal l in an implicant q of a function z is prime if the result of deleting l from q is no longer an implicant of z. Consider a non-redundant two-level single-output circuit C realizing a cover E of z. Then, there are only two types of primitive delay faults according to [12]: $f_1 = \downarrow P$, where multiple-path P starts from a primary input associated with a prime literal l_i in an implicant $q \in E$, and $f_2 = \uparrow p$, where p is a single path in P, i.e., $p \in P$. Let $v_2 = l_1 l_2 \ldots l_i \ldots l_n$ be an input vector such that $q(v_2) = 1$ and $v_1 = l_1 l_2 \ldots l_i' \ldots l_n$ be the result of replacing literal l_i with l_i' from v_2. It is shown in [12] that the vector-pair $t_m = \langle v_2, v_1\rangle$ robustly detects f_1 when $E(v_1) = 0$. It is also shown that $t_s = \langle v_1, v_2\rangle$ is a robust or a VNR test for f_2 when v_2 is a relatively essential vector of q and $E(v_1) = 0$. In other words, t_m is composed of a true vector v_2 and the adjacent false vector v_1; thus v_1 and v_2 form a CT pair. If C meets the conditions of Theorem 1, there exists at least one relatively essential vector v_2 in an implicant $q \in E$ that is adjacent to a false vector v_1, so that $\langle v_1, v_2\rangle$ detects f_2. Note that v_1 and v_2 are adjacent CTs.

Theorem 6. *Consider a non-redundant two-level circuit C realizing a cover E of a single-output function z. The CDTS for z contains a robust or a VNR test for every primitive delay fault in C if and only if each implicant in E has a relatively essential vector adjacent to a false vector.*

The synthesis techniques algebraic factorization and resubstitution preserve primitive fault testability and primitive fault test sets [12]. Using this fact, we can extend Theorem 6 to multiple-level circuits. Hence, the coupling delay test set for z also detects all primitive delay faults in a multi-level circuit derived from C by using algebraic factorization and resubstitution.

5. Coupling Test Generation

COTEGE (COupling TEst GEnerator) is a test generation program for coupling faults that we developed for combinational modular circuits whose functional behavior is specified hierarchically as in RIBTEC [14]. As illustrated in Fig. 12, the combinational circuit of interest C can be a sub-circuit (e.g. an adder) of a sequential circuit partitioned by registers, and each circuit C is further partitioned into modules, e.g. a carry module of an adder. The CTS for each module is computed by COTEGE with respect to the module boundaries. Then

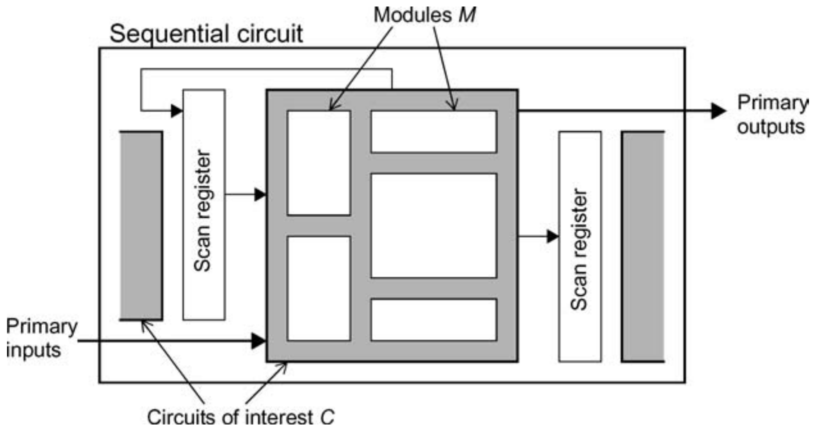


Fig. 12. Hierarchical specification of a circuit.

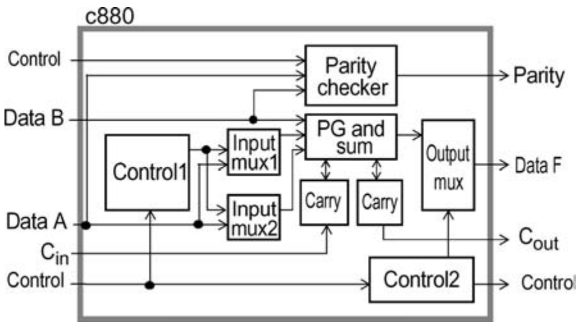


Fig. 13. Modular design of the c880 benchmark, an 8-bit ALU [14].

a test set T for C is generated with respect to the inputs of C by using high-level techniques [14] based on PODEM [10]. Thus while T is dependent on the modular interconnection structure of C, it is independent of each module's internal realization. Note that some CTs for an embedded module m may not be applicable by T because the inputs of m depend on the functions of preceding modules. For example, Fig. 13 shows the modular structure of an ISCAS-85 benchmark circuit c880 used in our experiments. Circuit c880 is an 8-bit ALU, and is partitioned into various modules including carry, control, and parity-check modules.

COTEGE reads behavioral Verilog code defining a target circuit in sum-of-products form and generates a test output file which can be directly used by a conventional SSL fault simulator FSIM [16]. Fig. 14 gives a pseudocode description of COTEGE. The CTS CTS_k for each type of module M_k is generated by procedure $CTgen()$ whose pseudocode is shown in Fig. 15. The reduced CTS CTS_k^R is obtained from CTS_k by removing all non-maximal false and non-minimal true CTs. Then for each module CT $t_{ij} \in CTS_k^R$, a high-level ATPG technique HL-PODEM [14] computes the test pt defined at inputs of the circuit under test C. The fault simulation step removes any untested CFs that are detected by pt. After all CFs are detected, reverse-order fault simulation is performed for test compaction purposes.

The procedure HL-PODEM regards module input values of t_{ij} and faulty responses r_{ij} at the outputs of module M_k as objectives in the standard PODEM algorithm, and computes an input vector pt that applies t_{ij} and propagates the faulty responses to a primary output. CT t_{ij} is uncontrollable if no input vector of C applies t_{ij} to M_k. Since M_k usually has multiple outputs and t_{ij} can be a CT for the CF between another input-output pair, there can be several possible faulty responses r_{ij} for t_{ij}. CT t_{ij} is unobservable if the faulty value at the target module output z_j cannot be propagated to a primary output of C for at least one of all those faulty responses.

For every CT t_{ij} that is uncontrollable and/or unobservable, HL-PODEM keeps replacing t_{ij} with a set T_C of covered CTs by t_{ij} according to the universal test method of RIBTEC [14] until there exists no such T_C. Two vectors t_a and t_b are said to be *comparable* if either $t_a \geq t_b$ or $t_a \leq t_b$; otherwise, they are *incomparable*. Let e_a denote the expanded vector of t_a. If t_{ij} is a true vector, T_C is the largest set e_k of incomparable expanded true CTs that contains a CT $e_k \leq e_h$ for any true expanded CT $e_h > e_{ij}$. That is, every expanded CT e_h greater than e_{ij} has an expanded CT $e_k \leq e_h$. Hence, it

```
procedure COTEGE(C) {                    // $C$ = circuit and $M_k$ = module
// $x_i$ = module input, $z_j$ = module output, and $t_{ij}$ = module coupling test for CF $x_i|z_j$.
for each type of modules $M_k$
    $CTS_k$ = CTgen($z_j$);// $CTS_k$ is the CTS for $M_k$
    $CTS_k^N$ = $CTS_k$ - {all maximal false and minimal true CTs};
    $CTS_k^R$ = $CTS_k$ - $CTS_k^N$;// $CTS_k^R$: reduced CTS for $M_k$
    $S_{IO}$ = all pairs { ($t_{ij}$, $r_{ij}$) } for each CT $t_{ij}$ and faulty response $r_{ij}$ in $CTS_k^R$;
end
while ($S_{IO}$ is not empty) begin
    Select a pair ($t_{ij}$, $r_{ij}$) from $S_{IO}$
    $S_{IO}$ = $S_{IO}$ - ($t_{ij}$, $r_{ij}$);
    // HL-PODEM determines whether $t_{ij}$ is observable and controllable with the faulty output $r_{ij}$
    // pt = test defined at inputs of C
    pt = HL-PODEM( ($t_{ij}$, $r_{ij}$) );
    if (pt exists) then
        $T = T \cup \{pt\}$;// The final test set
        $S_{CF} = S_{CF} \cup \{x_i|z_j\}$;// $S_{CF}$ = Set of detected CFs
        FaultSimulate (pt, $S_{IO}$);// Remove other pair from $S_{IO}$ that are covered by pt
    end
end
Reverse-Order-FaultSimulate(T);// Test set compaction
return T and $S_{CF}$;
} // end of COTEGE();
```

Fig. 14. Pseudocode for the COTEGE program which computes a high-level test set for a modular circuit based on the coupling fault model.

```
procedure CTgen() {
for each product term $P_i$ of a SOP B
    { $G^T(k)$, $G^F(k)$ } = build_G($P_i$);// $G^T(k)$: Set of true vectors that have k 1's.
                                      // $G^F(k)$: Set of false vectors that have k 1's.
for $0 \leq k \leq n - 1$     // n: the number of inputs of B
    $V_{k,t} \in G^T(k)$; $V_{k+1,f} \in G^F(k+1)$;
    if $V_{k,t}$ and $V_{k+1,f}$ have one-bit difference then $CTS^C = CTS^C \cup \{V_{k,t}, V_{k+1,f}\}$;// $CTS^C$: the CTS
    $V_{k,f} \in G^F(k)$; $V_{k+1,t} \in G^T(k+1)$;
    if $V_{k,f}$ and $V_{k+1,t}$ have one-bit difference then $CTS^C = CTS^C \cup \{V_{k,f}, V_{k+1,t}\}$;
end
return $CTS^C$;
} // end of CTgen();
```

Fig. 15. Pseudocode of $CT_{gen()}$ which generates all CTs for a given logic function.

is not necessary to apply e_h if all tests in T_C are applied to a BIP network for stuck-at fault detection. Similarly, if t_{ij} is a false vector, T_C is the largest set e_k of incomparable expanded false CTs that contains a CT $e_k \geq e_h$ for any false expanded CT $e_h < e_{ij}$. For example, consider the 3-input function z whose expanded truth table is shown in Table 3. All eight input vectors are CTs and the reduced CTS for z is composed of the four shaded tests. If (expanded) test 0101 is either uncontrollable or unobservable, then $T_C = \{0111\}$ replaces 0101.

Computing tests via the Boolean difference is known to be hard. In Larrabee's ATPG method [15], for example, the Boolean difference is used only for a small number of hard-to-detect faults after random tests are

Table 3. Expanded truth table of the function $z = ab + a'c'$.

Input vector abc	Expanded vector $aa'bc'$	z	
000	0101	1	Min. true CT
001	0100	0	
010	0111	1	
011	0110	0	Max. false CT
100	1001	0	Max. false CT
101	1000	0	
110	1011	1	
111	1010	1	Min. true CT

generated to detect easy-to-detect faults. The representation and manipulation of Boolean functions can be implemented in several ways: sum-of-products, conjunctive normal form, or binary decision diagrams (BDDs). As we show in [25], the complexity of computing the Boolean difference can be greatly reduced by using BDDs. However, in the current version of COTEGE, the CT generation procedure $CTgen()$ exhaustively searches for CTs by using the properties of adjacent CTs. In other words, all vector pairs separated by Hamming distance 1 are examined if they have different outputs. If so, the two vectors are added to the CTS. The size of a function that can be efficiently handled by COTEGE is also restricted by test set size. In our experiments, we limit the number of module inputs to at most 20. Note that some functions still have too many tests with 20 inputs. For example, the XOR function with 20 inputs has an exhaustive set of 2^{20} tests, so we limit XOR modules to 6 inputs.

6. Experimental Results

We applied COTEGE to some representative and fairly big combinational circuits on a 400 MHz SUN Ultra5 workstation. The test circuits for the experiments include two ISCAS-85 benchmarks (c880 and c7552), three datapath circuits (a 16-bit carry lookahead adder CLA, a 16-bit ALU, and a 16-bit magnitude comparator), and two fully binate circuits (the 20-input identity function l_{20} and a 12-bit address comparator). The ISCAS-85 and datapath circuits have the module structure used in [14], and are composed of modules containing as many as 17 inputs and 8 outputs. The two binate circuits are viewed as single modules. Fig. 13 shows the modular design we used for c880.

Table 4 compares the size of the reduced CTS generated by COTEGE to that of the UTS generated by RIBTEC for each example circuit. As can be seen, the size differences between the CTSs and UTSs are negligible for the CLA, ALU, magnitude comparator, c880, and c7552. These circuits have highly unate modules, so their UTSs are also quite small. As expected, for the two fully binate functions the CTS is much smaller than the UTS, which is, of course, exhaustive. We also generated test sets for SSL faults using a conventional (non-hierarchical) ATPG tool ATALANTA [16] and two different gate-level implementations of each circuit; these implementations were synthesized either for low area or high speed by the Synopsys Design Compiler. As Table 4 shows, the ratio of the size of the CTS to that of the SSL test set ranges from 1 to 8. Interestingly, the CTS and SSL test sets for l_{20} are identical for both implementations.

Conventional SSL fault simulation was performed on the synthesized implementations of each circuit with the above test sets. Note that we do not restrict the synthesis techniques to test-set preserving ones. Table 5 summarizes the results. For example, the coupling and universal test sets achieve 100% fault coverage for the 16-bit CLA circuit optimized for low area (listed under Area) and high speed (listed under Speed). The test set generated by ATALANTA for the low-area design achieves 100% fault coverage, but it misses 39 faults in the high-speed design, which results in 93% fault

Table 4. Size of the test sets generated by COTEGE, RIBTEC and ATALANTA (CPU time is in seconds.).

						ATALANTA (A)					
	COTEGE (C)		RIBTEC (R)			Low-area realization			High-speed realization		
Circuits	CPU time	Number of tests	CPU time	Number of tests	Test set size ratio C/R	CPU time	Number of tests	Ratio C/A	CPU time	Number of tests	Ratio C/A
16-bit CLA	282	48	1,254	48	1	0.05	24	2.0	0.13	30	1.6
16-bit ALU	2.28	87	4.32	88	0.99	0.05	31	2.8	0.83	35	2.5
16-bit comparator	16.3	110	12.0	114	0.96	0.38	52	2.1	0.32	69	1.6
c880	597	208	410	226	0.92	0.15	46	4.5	0.17	64	3.3
c7552	1,162	209	1,159	213	0.98	0.35	52	4.0	0.92	65	3.2
20-input identity function	27.7	42	–	2^{20}	0.0004	0.03	42	1.0	0.03	42	1.0
12-bit address comparator (16 inputs)	5.40	390	–	2^{16}	0.006	0.07	48	8.1	0.08	53	7.4

Table 5. SSL fault coverage of reduced CTSs and UTSs for various gate-level realizations. The numbers of undetected SSL faults are in parentheses.

	16-bit CLA		16-bit ALU		16-bit comparator		c880		c7552		20-input identity func.		12-bit addr. comparator	
Test set	Area	Speed	Area	Speed	Area	Speed	Area	Speed	Area	Speed	Area	Speed	Area	Speed
COTEGE tests (CTSs) for modular circuit	100%	100%	100%	99.2% (7)	99.1% (4)	100%	100%	99.8% (2)	99.8% (4)	98.5% (35)	100%	100%	100%	100%
RIBTEC tests (UTSs) for modular circuit	100%	100%	100%	99.2% (7)	99.1% (4)	100%	100%	99.8% (2)	99.8% (4)	98.5% (35)	100%	100%	100%	100%
ATALANTA tests for low-area realization	100%	93.0% (39)	100%	97.7% (17)	96.3% (17)	91.9% (42)	100%	95.9% (36)	99.8% (4)	96.6 (77)	100%	100%	100%	96.1% (6)
ATALANTA tests for high-speed realization	98.5% (7)	100%	100%	99.2% (7)	98.3% (8)	100%	99.7% (2)	99.8% (2)	99.6% (7)	98.5% (35)	100%	100%	98.0% (3)	100%

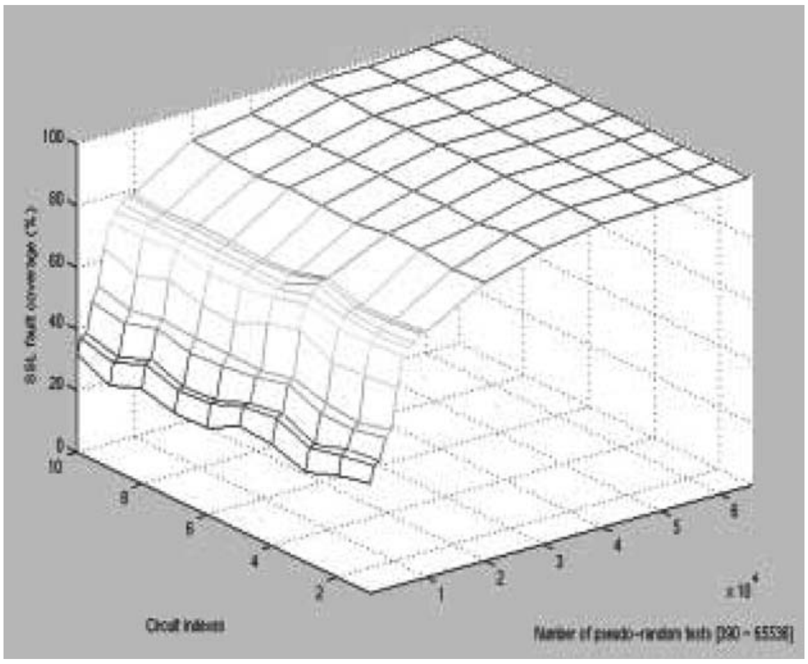


Fig. 16. SSL fault coverage of pseudorandom test sets for 10 different realizations of the address comparator.

coverage. As can be seen from Table 5, both the coupling and universal test sets detect all SSL faults in all gate-level implementations, while the realization-dependent SSL test sets for one design miss some faults in the other design. Note that the CTSs for the two binate functions detect all SSL faults in all gate-level implementations with fewer tests than the UTSs. Also, note that the high-level test sets (coupling and universal test sets) for the 16-bit magnitude comparator detect some hard-to-detect faults in the low-area design which the conventional ATPG tool (ATALANTA) aborts. In general, the SSL fault coverage of the high-level test sets is better than that of the realization-dependent test sets.

Pseudorandom testing can also provide a high degree of realization independence at the cost of a very large number of tests. Fig. 16 shows the SSL fault coverage of (pseudo) random test sets for 10 different implementations of the 12-bit address comparator. The fault coverage figures of the random test sets for all implementations are almost the same, as expected. The SSL fault coverage of 390 random vectors is 20 to 30% in all cases. To achieve coverage above 90%, more than 20K random test vectors are required; to achieve 100% fault coverage, more than 61K random vectors were required. Thus we need a near-exhaustive random test set to achieve 100% SSL fault coverage for any realization of the address comparator, while 390 CT vectors provide 100% SSL fault coverage for all the realizations considered.

The coupling delay test sets for two fully binate functions (I_{20} and the 12-bit address comparator) and four ISCAS-89 benchmark circuits were generated and the results are shown in Table 6. Each circuit is regarded as a single module in this experiment. For the two binate functions, the low-area implementations used in previous experiments are used to compute the number of paths and path delay faults. The number of path delay faults is the maximum number of path delay tests. The number of coupling delay tests was found to be from one to 65 times larger than that of the path delay tests. We propose a hierarchical coupling delay test generation method and a fault model in [25] to address this test-set size problem. Nevertheless, the size of the coupling delay test sets is much less than that of the corresponding gross delay test set, that is, the set of all SIC tests. (Recall that the number of SIC tests for an n-input function is $n \cdot 2^n$.) The coupling delay tests also have the advantage of detecting all robust path delay faults in any realization.

Table 6. Coupling delay test generation results for two fully binate functions and four ISCAS-89 benchmark circuits.

				Coupling delay testing			
Circuit	Description	No. of inputs	No. of path delay faults	No. of faults	No. of tests	CPU time	No. of gross delay tests
I_{20}	20-input identity	20	80	80	80	10.89 sec	20.9×10^6
addrcomp	12-bit address comparator	16	142	790	790	1.09 sec	1.05×10^6
s298	Traffic light controller	17	462	1856	672	0.3 sec	2.23×10^6
s400		24	896	83,202	51,870	54.86 sec	403×10^6
s386	Controller	13	414	6,000	3,084	1.15 sec	0.106×10^6
s1494		14	1,952	186,398	55,830	316.54 sec	0.229×10^6

7. Conclusions

A high-level fault model called the coupling fault (CF) model has been proposed to address some testing problems that cannot be efficiently handled at the gate level. The basic properties of CFs and their tests have been explored, one of which is the ability to cover both functional and delay faults in a uniform and realization-independent manner. A test generation program COTEGE has been developed for CF detection. Our experiments with COTEGE show that (reduced) coupling test sets efficiently cover standard, low-level faults in a variety of realizations using fewer tests than other high-level test sets. The corresponding coupling delay tests (CDTs) detect all robust path delay faults in any realization of a function. CDT sets tend to be larger than typical path delay test sets, but their size can be reduced by employing hierarchical methods [25].

References

1. S.B. Akers, "Universal Test Sets for Logic Networks," *IEEE Trans. on Computers*, vol. C-22, no. 9, pp. 835–839, 1973.
2. H. Al-Asaad, J.P. Hayes, and B.T. Murray, "Scalable Test Generators for High-Speed Datapath Circuits," *Journal of Electronic Testing*, vol. 12, pp. 111–125, 1998.
3. M.J. Batek and J.P. Hayes, "Test-Set Preserving Logic Transformations," in *Proc. Design Automation Conf.*, pp. 454–458, 1992.
4. R. Betancourt, "Derivation of Minimum Test Sets for Unate Logic Circuits," *IEEE Trans. on Computers*, vol. C-20, no. 11, pp. 1264–1269, 1971.
5. K.-T. Cheng, A. Krstic, and H-C Chen, "Generation of High Quality Tests for Robustly Untestable Path Delay Faults," *IEEE Trans. on Computers*, vol. 45, no. 12, pp. 1379–1392, 1996.
6. K. De, "Test Methodology for Embedded Cores which Protects Intellectual Property," in *Proc. VLSI Test Symposium*, 1997, pp. 2–9
7. B.I. Dervisoglu, "A Unified DFT Architecture for Use with IEEE 1149.1 and VSIA/IEEE P1500 Compliant Test Access Controllers," in *Proc. Design Automation Conference*, pp. 53–58, 2001.
8. S. Devadas and K. Keutzer, "Synthesis of Robust Delay-Fault-Testable Circuits: Theory," *IEEE Trans. on CAD*, vol. 11, no. 1, pp. 87–101, 1992.
9. M. Gharaybeh, M.L. Bushnell, and V.D. Agrawal, "Classification and Test Generation for Path-Delay Faults Using Single Stuck-at Fault Tests," *Journal of Electronic Testing*, vol. 11, pp. 55–67, 1997.
10. P. Goel, "An Implicit Enumeration Algorithm to Generate Tests for Combinational Logic Circuits," *IEEE Trans. on Computers*, vol. C-30, no. 3, pp. 215–222, 1981.
11. J.P. Hayes, "On Realizations of Boolean Functions Requiring a Minimal or Near-Minimal Number of Tests," *IEEE Trans. on Computers*, vol. C-20, no. 12, pp. 1506–1513, 1971.
12. W. Ke and P.R. Menon, "Synthesis of Delay-Verifiable Combinational Circuits," *IEEE Trans. on Computers*, vol. 44, no. 2, 1995.
13. H. Kim and J.P. Hayes, "Delay Fault Testing of Designs with Embedded *IP* Cores," in *Proc. VLSI Test Symposium*, 1999, pp. 160–167.
14. H. Kim and J.P. Hayes, "Realization-Independent ATPG for Designs with Unimplemented Blocks," *IEEE Trans. on CAD*, vol. 20, no. 2, pp. 290–306, 2001.
15. T. Larrabee, "Efficient Generation of Test Patterns Using Boolean Difference," in *Proc. International Test Conf.*, pp. 795–801, 1989.
16. H.K. Lee and D.S. Ha, "On the Generation of Test Patterns for Combinational Circuits," Tech. Report 12–93, EE Dept., Virginia Polytechnic Institute and State University, 1993.
17. C.J. Lin and S.M. Reddy, "On Delay Fault Testing in Logic Circuits," *IEEE Trans. on CAD*, vol. CAD-6, no. 5, pp. 694–703, 1987.
18. I. Pomeranz and S.M. Reddy, "On Testing Delay Faults in Macro-Based Combinational Circuits," in *Proc. International Conf. on CAD*, pp. 332–339, 1994.
19. I. Pomeranz and S.M. Reddy, "Functional Test Generation for Delay Faults in Combinational Circuits," in *Proc. International Conf. on CAD*, pp. 687–694, 1995.
20. M. Psarakis, D. Gizopoulos, and A. Paschalis, "Test Generation and Fault Simulation for Cell Fault Model Using Stuck-at Fault Model Based Test Tools," *Journal of Electronic Testing*, vol. 13, pp. 315–319, 1998.

21. M. Psarakis et al., “Sequential Fault Modeling and Test Pattern Generation for CMOS Iterative Logic Arrays,” *IEEE Trans. on Computers*, vol. 49, no. 10, pp. 1083–1099, 2000.
22. J. Rajski and J. Vasudevamurthy, “The Testability-Preserving Concurrent Decomposition and Factorization of Boolean Expressions,” *IEEE Trans. on CAD*, vol. 11, no. 6, pp. 778–793, 1992.
23. G.L. Smith, “Model for Delay Faults Based Upon Paths,” in *Proc. International Test Conf.*, pp. 342–349, 1985.
24. U. Sparmann, H. Muller, and S.M. Reddy, “Universal Delay Test Sets for Path Delay Faults,” *IEEE Trans. on VLSI Systems*, vol. 7, no. 2, pp. 156–166, 1999.
25. J. Yi, “High-Level Function and Delay Testing for Digital Circuits,” Ph.D Thesis, University of Michigan, 2002.

Joonhwan Yi received the B.S degree in electronics engineering from Yonsei University, Seoul, Korea, in 1991, and the M.S. and Ph.D degrees in electrical engineering and computer science from the University of Michigan, Ann Arbor, in 1998 and 2002, respectively.

From 1991 to 1995, he was with Samsung Electronics, Semiconductor Business, Korea, where he was involved in developing application specific integrated circuit cell libraries. In 2000, he was a summer intern with Cisco, Santa Clara, CA, where he worked for path delay fault testing. Since 2003, he has been with Samsung Electronics, Telecommunication Network, Suwon, Korea, where he is working on system-on-a-chip design. His current research interests include C-level system modeling for fast hardware and software co-simulation, system-level power analysis and optimization, behavioral synthesis, and high-level testing.

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, Urbana-Champaign, 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 and software development. From 1972 to 1982 he was a faculty member of the Departments of Electrical Engineering– Systems and Computer Science of the University of Southern California, Los Angeles. Since 1982 he has been with the Electrical Engineering and Computer Science Department of the University of Michigan, Ann Arbor, where he holds the Claude E. Shannon Chair in Engineering Science.

Professor Hayes was the Founding Director of the University of Michigan’s Advanced Computer Architecture Laboratory (ACAL). He has authored over 225 technical papers, several patents, and five books, including *Introduction to Digital Logic Design* (Addison-Wesley, 1993), and *Computer Architecture and Organization*, (3rd edition, McGraw-Hill, 1998). He has served as editor of various technical journals, including the *Communications of the ACM*, the *IEEE Transactions on Parallel and Distributed Systems* and the *Journal of Electronic Testing*. Professor Hayes is a fellow of both IEEE and ACM, and a member of Sigma Xi. He received the University of Michigan’s Distinguished Faculty Achievement Award in 1999 and the Humboldt Foundation’s Research Award in 2004. His current teaching and research interests are in the areas of computer-aided design, verification, and testing; VLSI circuits; fault-tolerant embedded systems; ad-hoc computer networks; and quantum computing.