Delay-Fault Test Generation and Synthesis for Testability Under a Standard Scan Design Methodology

Kwang-Ting Cheng, *Member, IEEE,* Srinivas Devadas, *Member, IEEE,* Kurt Keutzer, *Member, IEEE*

Abstract—**We address the problems of test generation and synthesis aimed at producing VLSI sequential circuits that are delay-fault testable under a standard scan design methodology. We begin with theoretical results regarding the standard-scan delay testability of finite state machines (FSM's) described at the state transition graph (STG) level. We show that a one-hot coded and optimized FSM whose STG satisfies a certain property is guaranteed to be fully gate-delay-fault testable under standard scan. We extend this result to arbitrary-length encodings and develop a heuristic state assignment algorithm that results in highly gate-delay-fault testable sequential FSMs, which are also area-efficient, as evinced by results obtained on benchmark FSM circuits.**

We switch focus to the problem of delay test generation for large sequential circuits and modify a PODEM-based combinational test pattern generator for our purpose. The modifications involve a two time-frame expansion of the combinational logic of the circuit, and the use of backtracking heuristics tailored for our problem. We also employ a version of the scan shifting technique in our test pattern generator. We give an algorithm to determine an efficient ordering of the flip-flops in the scan-chain using the information derived from running a delay test generator on the circuit. Given an ordering of the flip-flops, we give an optimization algorithm that attempts to minimize the number of flip-flops to be made enhanced-scan so as to obtain a required level of delay-fault coverage. We show how the test vector sets derived using our test generator can be compacted by solving a clique covering problem. We present test generation, flip-flop ordering, flip-flop selection and test set compaction results on large benchmark circuits.

I. Introduction

WHILE a high degree of delay-fault coverage is desirable, a number of problems are associated with testing in delay-fault models. The first problem is creating logic that is highly testable for delay-faults. Synthesis for delay-fault testability of combinational circuits has recently been the subject of extensive investigation (e.g., [1]–[3]). Practical integrated circuit designs that are completely hazard-free robustly path-delay-fault testable were automatically synthesized using the techniques presented in [4], [5]. The work in [4], [5] assumes an enhanced-scan design methodology for the robust application of test vector pairs.

Another obstacle to delay-fault testing is the large vector sets required to test each path in (hazard-free) robust manner [6], [7]. This problem is addressed in [1] where a general robust path-delay-fault model is proposed which can potentially significantly reduce the test vector sets. Another alternative is to use delay fault models with a fewer number of faults such as the gate delay fault model. Results presented in [5] show that by using a gate delay-fault model rather than a path-delay-fault model can reduce the required number of test vectors to a manageable number.

A remaining important barrier to the practical use of delay-fault testing is that the application of a test vector pair to the combinational logic portion of a VLSI sequential circuit may require that the memory elements in the circuit be enhanced scan flip-flops, i.e., flip-flops that can store not just 1, but 2 bits of state. This is because an arbitrary vector pair cannot be applied to a sequential circuit under a non-scan or a standard scan design methodology. The use of an enhanced scan latch imposes an area penalty that is prohibitively expensive for many circuit designs. In this paper, we address the problem of test generation for, and the synthesis of, VLSI sequential circuits that are delay-fault testable under a standard scan design methodology.

There has been some previous work in the area of sequential delay testing. The various issues in the delay testing of sequential circuits were illustrated in [8], and the use of scan design rather than non-scan design was recommended. In [9], the problem of non-scan delay testing and finite-state machine (FSM) synthesis for delay testability was addressed. This work was limited to synthesizing testable finite state machines from state transition graph (STG) descriptions, hence the size of the circuits dealt with were small. Also, no guarantees of fault coverage were made. Designing stuck-open fault testable, standard scan-based sequential circuits was the subject of [10]. However, only circuits where each next state and output function were realized by a single complex gate were considered. A heuristic algorithm to arrange the flip-

Manuscript received March 20, 1991; revised August 17, 1992. This work was supported in part by the Defense Advanced Research Projects Agency under Contract N00014-87-K-0825, and in part by a grant from Analog Devices, Inc. This paper was recommended by Associate Editor S. Seth.

K.-T. Cheng is with the AT&T Bell Laboratories, Murray Hill, NJ 07974.

S. Devada is with the Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, Cambridge, MA. 01239.

K. Keutzer is with Synopsys, Mountain View, CA 94043-4033.

IEEE Log Number 9207273.

flops in a scan chain so a maximal number of given vector pairs can be applied was presented in [11].

We begin this paper with theoretical results regarding the standard-scan delay testability of FSM's described at the STG level (Section III). We show that *a one-hot coded and optimized FSM whose STG satisfies a certain property is guaranteed to be fully gate-delay-fault testable under standard scan*. We extend this result to arbitrary-length encodings and develop a heuristic state assignment algorithm that results in highly gate-delay-fault testable sequential FSM's, which are also area-efficient, as evinced by results obtained on benchmark FSM circuits.

We switch focus to the problem of delay test generation of large sequential circuits and modify a PODEM-based combinational test pattern generator for our purpose. The modifications involve a two time-frame expansion of the combinational logic of the circuit, and the use of backtracking heuristics tailored for our problem. Vector pairs that can be obtained by single-bit shifts can be applied at speed under standard scan. This notion is called *scan shifting* (called *transition shifting* in [12]). We first give an algorithm to determine the *efficient ordering of the flip-flops* in the scan chain that *uses the information derived from running a delay test generator on the circuit*. Given an ordering of the flip-flops, we give an optimization algorithm that attempts to *minimize the number of flip-flops to be made enhanced-scan* so as to obtain the required level of delay-fault coverage, under a partial enhanced-scan/standard-scan methodology. To address the problem of large vector sets we show how the test sets derived using our test generator can be compacted by solving a clique covering problem. We present test generation, flip-flop ordering, flip-flop selection and test set compaction results on circuits from the ISCAS sequential benchmark set in Section V.

II. Definitions and Notation

Analysis and synthesis of delay-fault testable circuits requires spanning the field of logic optimization, testing and timing analysis. In this section we try to provide a minimal amount of terminology from these fields.

2.1. Logic Synthesis

A **Boolean function** F of N variables is a mapping from $B^N = \{0, 1\}^N \rightarrow \{0, 1\}$. We model B^N as a Binary N-cube. A vertex $v \in B^N$ for which $F(v) = 1$ is a member of the *ON*-set. If $F(v) = 0$ then v is a member of the *OFF*-set.

A **literal** is a Boolean variable or its complement. We formally define a **cube** as a set of literals and we interpret the cube as a product of literals. For example $\{a, b, c\}$ is a cube, interpreted as $a \cdot b \cdot c$, which may be abbreviated to abc. A **minterm** is a cube in which every variable in the Boolean function appears. We may interpret the minterm as a **vertex** in the N-cube.

Minterms and cubes may be used to represent the values of a set of input variables: e.g., $x\bar{y}z$ is shorthand for $x = 1$, $y = 0$, and $z = 1$. In this way there is a natural correspondence between an input vector or input stimulus, a minterm and a vertex in the N-cube. This correspondence may be extended to cubes where unspecified values in the function are assumed to be undefined values. Thus, if a circuit C has inputs v, w, x, y, z, then applying the cube $x\bar{y}z$ to C is shorthand for applying $v = X$, $w = X$, $x = 1$, $y = 0$, and $z = 1$. (Here X denotes an unknown value.)

We define a **cover** as a set of cubes and we interpret the cover as a sum-of-products expression. For example $\{\{a, b, c\}, \{d, e, f\}\}$ is a cover, interpreted as $abc + def$. We say that a cube q **covers** a cube (or vertex) r, denoted $r \sqsubseteq q$, iff q is a subset of r ($q \subseteq r$). The use of the set notation $q \subseteq r$ is counter-intuitive and we will use $r \sqsubseteq q$ henceforth. We say that a cube (vector) q is *compatible* with a cube (vector) r iff there does not exist a literal $l \in q$ such that $\bar{l} \in r$. In other words, two vectors are compatible if their values do not conflict in any input. Given cubes (vectors) q and r, if q and r are compatible then the **cube intersection**, denoted $q \sqcap r$, is $q \cup r$ and if cubes q and r are not compatible then $q \sqcap r = \phi$. Again, the use of the set notation $q \cup r$ is counter-intuitive and we will use the more familiar $q \sqcap r$ henceforth. If a cube q covers only *ON*-set vertices of a Boolean function F then q is called an **implicant** of F. A **relatively essential vertex** or an **irredundancy test** of a cube q in a cover C is a vertex that is covered by q and is not contained in any other cube in C.

A **Boolean Network** η is a directed-acyclic graph (DAG) such that for each node n_i in η there is an associated cover F_i, and a Boolean variable y_i representing the output of F_i. As there is a one-to-one correspondence between a node and its cover we will use the terms interchangeably. There is a directed edge from n_i to n_j iff F_j explicitly depends on y_i or $\bar{y}_i$. Furthermore, some of the variables in η may be classified as *primary inputs* or *primary outputs*. We may directly implement a Boolean Network η by a combinational circuit $\mathcal{C}$ replacing each cover F_i in η by a NAND/NAND network and by replacing each edge set fanning out from a node in η by a **net**.

2.2. Delay Testing

In this section we introduce terminology that will allow us to discuss timing issues as well as fault models for temporal behavior. A **path** in a combinational circuit is an alternating sequence of vertices and edges, $\{v_0, e_0, \cdots, v_n, e_n, v_{n+1}\}$, where edge e_i, $1 \leq i \leq n$, connects the output of vertex v_i to an input of vertex v_{i+1}. For $1 \leq i \leq n$, v_i is a gate; v_0 is a primary input and v_{n+1} is a primary output. Each e_i is a net.

An **event** is a transition $0 \rightarrow 1$ or $1 \rightarrow 0$ at a gate. Consider a sequence of events, $\{r_0, r_1, \cdots, r_n\}$ occurring at gates $\{g_0, g_1, \cdots, g_n\}$ along a path, such that r_i occurs as a result of event r_{i-1}. The event r_0 is said to propagate along the path.

A circuit has a **gate-delay-fault** if there is one gate in

the circuit such that the output of the circuit is slow to make a $0 \rightarrow 1$ (or $1 \rightarrow 0$) transition because of an extra delay encountered at the gate. Each single gate-delay-fault is assumed to be so catastrophic as to cause a delay along any path through the gate to any output.

A circuit has a **path-delay-fault** if there exists a path from a primary input to a primary output via a set of gates and nets such that a primary input event is slow to propagate *along the path* to the primary output.

A two-pattern test $T = \langle v_1, v_2 \rangle$ is said to be a **general robust** delay test for a path π, if and only if, when π is faulty and the test T is applied, the circuit output is different from the expected state at sampling time, independent of the delays along gate input leads not on π [13].

A more stringent model is the **hazard-free robust** delay fault model, treated in [1], [4].[1] A robust test is said to be a hazard-free robust test if no hazards can occur on the tested path during the application of the test, regardless of gate delay values.

The vector $\langle v_1, v_2 \rangle$ is made up of $\langle i_1 @ s_1, i_2 @ s_2 \rangle$, where i_1 and i_2 are the values of the Primary Input (PI) lines, and s_1 and s_2 are the values of the Present State (PS) lines. The symbol @ denotes the concatenation of bit vectors. The constraint imposed by using a standard scan or a non-scan design methodology is now apparent: we require i_1 to produce s_2 as the next state if the machine is in s_1.

Given a sequential circuit with n flip-flops implemented under standard scan, our delay test methodology is as follows: We set the machine in test mode (by setting the Mode Switch high) and scan in state s_1 into the flip-flops (it requires n clock cycles). The machine is then clocked twice with appropriate primary input vectors i_1 and i_2. Before the machine is clocked the second time (in the words, before v_2 is applied), it has to be set in the normal mode to catch the fault effects from the functional logic. However, it can be set in either the normal mode or the test mode before the first clock pulse is applied. If the machine is in normal mode, then s_2 will be from the functional logic. Otherwise, the value of a flip-flop in s_2 will be obtained from its predecessor in the scan chain.

In this paper, we will be dealing with the general robust path and gate-delay-fault models. However, the suggested test generation procedure described in Section V should be considered as an approximate method for generating robust tests. To avoid using a multi-valued logic in test generation, the procedure may generate vector pairs that are hazardous.

III. Synthesizing Delay-Fault Testable Finite State Machines

3.1 Introduction

Our goal is to be able to synthesize a fully gate-delay-fault testable finite state machine (FSM) under a standard scan design methodology, or under a non-scan methodology. To this purpose we first show that given a State Transition Graph (STG) that satisfies a certain property, a one-hot coded and two-level minimized FSM derived from the STG will be fully testable under the gate-delay-fault model described in Section II. We show how any STG that does not satisfy the above property can be transformed into a STG that does satisfy the property by an appropriate addition of edges and states. Requiring the STG to satisfy this property is a sufficiency condition, and not necessary for a fully gate-delay-fault testable machine. We argue that most STGs will result in a fully gate-delay-fault testable machine after one-hot coding and two-level minimization.

One-hot codes may require too many bits and may be area-inefficient for large FSM's. We switch focus to arbitrary-length encodings (greater than or equal to the minimum-bit encoding). We give conditions that are necessary for the encoded and optimized two-level cover implementing the STG to be fully gate-delay-fault testable. We show that these conditions can be satisfied by appropriate modifications of the STG prior to encoding, and by constraining the encoding process.

It is possible to retain the gate-delay-fault testability under a multilevel logic implementation either in the one-hot coding case, or in the arbitrary encoding case, using a constrained algebraic factorization procedure to derive a multilevel network [4].

3.2. Terminology

A one-hot coding of a FSM with N_s states, implies a code of length N_s, with the code of each state having a 1 in a unique identifying position and 0's in each other position. After one-hot coding the FSM, the unused state codes are used as don't care conditions during two-level logic minimization. In a two-level implementation there is no sharing between the logic of the next state lines in a one-hot coded and minimized FSM; however, a cube (AND gate) may feed both a PO and an NS line.

A block diagram of a FSM is shown in Fig. 1(a). Its STG is shown in Fig. 1(b). The one-hot coded STG is shown in Fig. 2(a) with an appropriate don't care set appended, and a gate-level implementation of the combinational logic of the one-hot coded and logic minimized STG is shown in Fig. 2(b).

We are concerned with the gate-delay-fault testability of the gates in a two-level network. Given a particular AND gate, g, in a two-level network, we can detect a $0 \rightarrow 1$ delay fault on g, to a particular PO or NS line, by applying a vector v_1 in the *OFF*-set of the PO or NS line, and then applying a vector v_2 that is an irredundancy test for the cube corresponding to g for the PO or NS line. An irredundancy test implies that all AND gates that feed the PO or NS line are held at 0, except for g which is at 1. Applying $\langle v_2, v_1 \rangle$ will detect the $1 \rightarrow 0$ gate delay fault on g. Note that the above test may not be a robust test for the gate-delay-fault on g since the other gates may glitch to a 1 on the application of $\langle v_1, v_2 \rangle$. However it is in

[1]The hazard-free robust path-delay-fault model is simply called the robust path-delay-fault model in [4], [7].

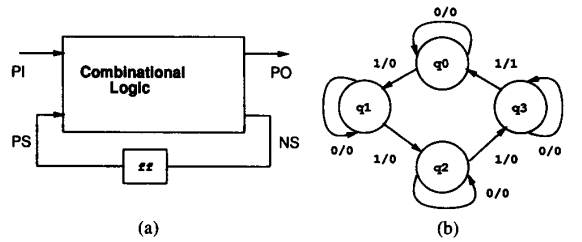

Fig. 1. A FSM and its STG.

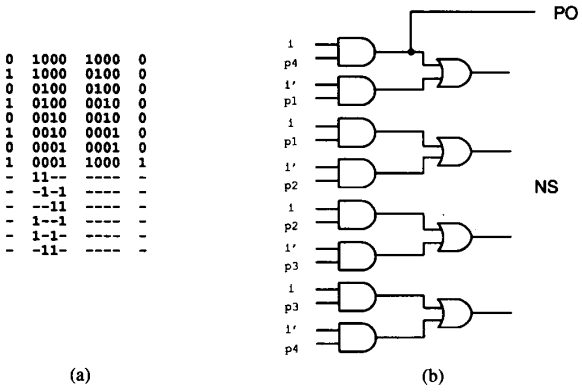

Fig. 2. One-hot coding a FSM.

keeping with our testing methodology outlined in Section II.

Testing any AND gate in a single-output two-level cover will simultaneously test the OR gate as well.

3.3. Testability of a One-Hot Coded and Two-Level Minimized FSM

Consider a one-hot coded and two-level minimized cover C derived from a STG G, representing a FSM M. Consider an AND gate g in C. We have four possible cases if we wish to test the AND gate g:

1) test g for a $1 \rightarrow 0$ gate delay fault to a PO line o;
2) test g for a $1 \rightarrow 0$ gate delay fault to a NS line n;
3) test g for a $0 \rightarrow 1$ gate delay fault to a PO line o;
4) test g for a $0 \rightarrow 1$ gate delay fault to a NS line n.

Even if all the AND gates feed the NS line (while also feeding the PO lines), we need to test the OR gates implementing the PO lines, which can be done by testing any one of the AND gates feeding the PO line, to the PO line.

For each of the cases above, we give the requirements on a $\langle i_1 @ s_1, i_2 @ s_2 \rangle$ that tests a particular AND gate g.

1) An irredundancy test for g at the PO o (there may be several), call it $i_1 @ s_1$, should go to a state s_2 which satisfies the property that $\exists i_2$ such that the value of o on the application of $i_2 @ s_2$ is 0.
   The state s_2 may not satisfy such a property. The original STG G can be modified so that each state $q \in G$ produces a 0 value for each output o on at least one input minterm i_2. Note that we do not require both 0 and 1 values on o to be produced by each state q, neither do we require that the minterm producing the 0 value for o be the same for different states.
   Also note that there may be several irredundancy tests for g and we really only need one irredundancy test to produce an s_2 that satisfies the above property. Making each state in a STG satisfy the property is an easy sufficiency condition. If we wish to relax this condition, we need to *predict* what the irredundancy tests are in the one-hot coded and minimized cover, and related them to input minterms and states—a difficult task.
2) An irredundancy test for g at the NS line n, $i_1 @ s_1$ will result in the machine going to a state s_n that is the state corresponding to the NS line, n. Recall that in a one-hot coding each NS line and PS line corresponds to a unique state in the FSM. Given that the irredundancy test produces a 1 at n, it means that the FSM has moved to s_n. We now require that $\exists i_2$ such that $i_2 @ s_n$ goes to a state $s_3 \neq s_n$. If $s_3 \neq s_n$, then the NS line n will have the value 0.
   The above requirement is easy to satisfy. For each state $q \in G$, we simply require that there be at least one input minterm i_2 that moves the machine to some state other than q. To put it differently, there should be no state p that only has "loop" edges, i.e., edges that result in the machine remaining in p.
   A strongly connected machine [14], where every state can be reached from every other state, will satisfy the above property. A STG that is not strongly connected can easily be modified to obtain a STG that is strongly connected.
3) Consider an irredundancy test for g at a PO o, namely $i_2 @ s_2$. We need s_1 to be produced by some input vector i_1 from some state s_1 such that the value of o is 0. In other words, we need, for each state q in the STG, at least one incoming edge which produces a value of 0 for each PO o (We do not require that the value of 1 be asserted for each PO o).
   Again, a STG can be easily modified to satisfy the above sufficiency condition.
4) Consider an irredundancy test for g corresponding to a NS line n, namely $i_2 @ s_2$. This vector results in the machine going to the state corresponding to the NS line n, namely s^n. We require that $s_2 \neq s^n$. Then we can reach s_2 with the value of the NS line n being 0, and apply i_2.
   The above requirement appears to be trivial, but in fact, is quite tricky. As mentioned earlier, it is difficult to predict what the irredundancy tests of each cube in the one-hot coded and minimized cover are going to be. We need the irredundancy tests for a cube and a NS line n to *not* correspond to s^n. If a loop edge of s^n becomes an irredundancy test for a NS line n, then the above condition is violated. One can envision a (complicated) two-level minimization strategy that monitors the irredundancy tests of

each cube and ensures that these tests do not correspond to loop edges. An easy way out is to split states in the machine so as to remove all loop edges. For instance, given a state graph fragment as follows:

$$i_1\ s_1\ \ q_1\ o_1$$

$$i_2\ q_1\ \ q_1\ o_2$$

$$i_3\ q_1\ \ s_2\ o_1$$

we can split q_1 into q_1 and q_2 transforming it to:

$$i_1\ s_1\ \ q_1\ o_1$$

$$i_2\ q_1\ \ q_2\ o_2$$

$$i_2\ q_2\ \ q_1\ o_2$$

$$i_3\ q_1\ \ s_2\ o_1$$

$$i_3\ q_2\ \ s_2\ o_1$$

where no loop edges exist. If q_1 and q_2 are encoded with different one-hot codes, then gate-delay-fault testability will be guaranteed.

It should be stressed that the above condition is a sufficient one and the likelihood of irredundancy tests corresponding only to loop edges in a typical machine is rather low.

Thus, it is feasible even though painstaking, to obtain a fully gate-delay-fault testable machine two-level circuit via one-hot coding and two-level minimization by appropriate modifications of the original STG. An example of applying the above transformations is shown in Fig. 3. In Fig. 3(a) we have an incompletely symbolic STG which satisfies the conditions 1 and 2 above. State **s1** does not satisfy condition 3, and an edge from state **s2** fanning out to **s1** is added to satisfy the property. Finally, state **s3** is split into **s3a** and **s3b** to satisfy condition 4. One-hot coding and minimizing the STG of Fig. 3(b) will result in a fully gate-delay-fault testable machine.

Empirical results indicate that these modifications are not necessary for high/full testability (cf. Section 4.2). Synthesis of multilevel circuits is treated in Section 3.6.

3.4 Testability of a FSM Under An Arbitrary Encoding

If a FSM has been encoded using, say a minimum-bit encoding, and then two-level minimized, we have a more complicated scenario as far as delay testing is concerned. AND gates may feed multiple NS lines, and there is no correspondence between the NS lines and states. However, an analysis can be performed and conditions necessary for gate-delay-fault testability can be derived.

As in the previous section, we will focus on the four different cases that occur while testing an AND gate g.

1) Same as in the one-hot coding case.
2) Consider an irredundancy test i_1 @ s_1 for a NS line n. This produces a 1 at the NS line n. Assume that i_1 @ s_1 moves the machine to state s_2. We require that $\exists\ i_2$ such that i_2 @ s_2 produces a 0 at the NS line n.

   A constrained encoding can satisfy the above requirement. For each next state line n (i.e., each bit in the encoding), we require an encoding such that each state fans out on some input minterm to a state that has a 0 in the bit corresponding to the NS line.
3) Same as in the one-hot coding case.
4) Consider an irredundancy test i_2 @ s_2 for a NS line n. We need s_2 to have a 0 at the bit position corresponding to the NS line n. Then we can reach s_2 by any means and apply i_2 to detect the $0 \rightarrow 1$ gate delay fault on g. Like in the one-hot coding case, this is a tricky requirement.

   If an irredundancy test is a loop edge, then the above condition is violated. One can remove loop edges as discussed earlier. In addition, we have to constrain the state encoding. If a state q has a 1 in a bit position, n, then if we can encode the fanin states of q (states from which we can reach q in one clock cycle) such that they all have a 0 in position n, the irredundancy tests will satisfy the above property. This may be a difficult constraint to satisfy for all states and all bit positions. Given that there are usually several irredundancy tests for a AND gate and several NS lines that a gate can be tested to, in practice this does not pose much of a problem.

   The use of scan shifting can further help in this case. A fanin state of q, namely s, need not have a 0 in the bit position n, if q can be obtained by shifting (a 0 or a 1 through) s. Further, loop edges will be allowed for states with the all 1's code and the all 0's code.

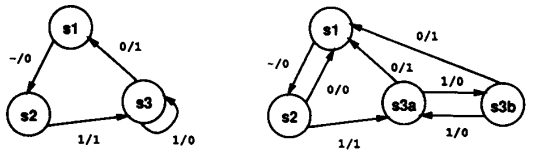

Fig. 3. Transformations to achieve testability under one-hot coding.

A STG that satisfies conditions 1 and 3 above is shown in Fig. 4(a). An encoding that satisfies conditions 2 and 4 is shown in Fig. 4(b).

3.5. Single-Output Versus Multiple-Output Minimization

After encoding a STG G, one can either perform a single-output two-level minimization or a multiple-output minimization. There are a number of different factors that are affected by this choice. In terms of circuit area, single-output two-level minimization has the potential to share only a small amount of logic in the two-level circuit, while, given an arbitrary encoding, multiple-output minimization can result in a significantly smaller number of

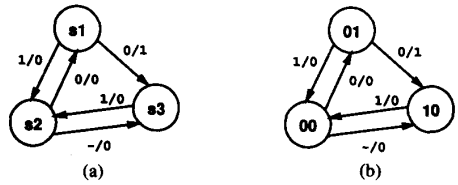

Fig. 4. Encoding that guarantees testability.

total product terms. Surprisingly, extensive experience has shown that algebraically factoring a single-output minimized two-level circuit most often results in a smaller circuit than algebraically factoring a multiple-output minimized two-level circuit, even though the latter is a smaller starting point. In terms of impact on testability, the number of cubes that feed a particular PO or NS line, will be larger in the case of the multiple-output minimized result and therefore the number of irredundancy tests per cube to a particular PO or NS line will be smaller, but there is also has the advantage that an AND gate corresponding to a cube can be tested to several NS and PO lines, as opposed to only one line. Even though the number of irredundancy tests for a cube in the multiple-output minimized case may be smaller, it should be noted that there will be at least one irredundancy test for each AND gate g to each PO or NS line g feeds, given that the cover is irredundant.

3.6. Multilevel Logic Implementations

The theory for maintaining the gate-delay-fault testability of a two-level network via a constrained algebraic factorization [4] can be used to advantage while synthesizing testable FSM's. We produce a FSM implemented using two-level logic that is standard-scan testable, and constrained algebraic factorization will maintain testability *and* test vector sets.

Performing a single-output two-level minimization initially prior to constrained factorization will ease the constraints that the factorization has to satisfy. Though the number of product terms in a single-output minimized circuit is typically significantly larger than a multiple-output minimized circuit, it appears that both serve as, more or less, comparable starting points for multilevel logic optimization [15], [4]. That is, the impact on the multilevel circuits' areas due to beginning from a different starting point is typically small.

IV. Heuristic State Assignment for High Testability

4.1. Testability Conditions and Minimal-Feedback Logic Structures

While the theory of the previous section can lead to a heuristic state assignment algorithm that maximizes delay testability, it is worthwhile to investigate one other property of a logic-level implementation of a sequential machine that guarantees delay-fault testability under standard scan, namely a *feedback-free* structure.

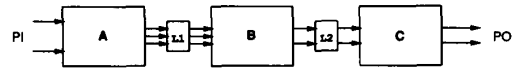

Fig. 5. A simple feedback-free circuit.

The circuit of Fig. 5 is fully gate-delay-fault testable, if A, B and C are gate-delay-fault testable individually, and if A and B produce all possible output combinations. Given a vector pair $\langle v_1, v_2 \rangle$ required to test a gate in B, we scan in v_1 into the flip-flops $L1$ and apply the appropriate primary input to A that produces v_2 at $L1$ one clock cycle later.

In [16], a state assignment algorithm devised toward producing a minimal-feedback structure was presented, in order to improve non-scan single stuck-at fault testability. The program SACRED [16] is guaranteed to find a feedback-free structure given a STG specification, if one exists. FSM's require some form of feedback. It is intuitive that if feedback is minimized that delay-fault testability will improve.

We can integrate the conditions presented in Sections 3.3 and 3.4 with the notion of minimizing feedback in an eventual logic-level implementation of a FSM to arrive at a heuristic algorithm for state assignment that maximizes delay-fault testability. In particular, we revisit the conditions required in the four cases of testing a gate in a one-hot coded and optimized cover, taking into account the fact that a next state line n can be made to not depend on all present state lines but rather on a minimal subset.

For each of the conditions 1–4 of Sections 3.3 and 3.4, we require an irredundancy test i_1 @ s_1 for a gate g or an irredundancy test i_2 @ s_2 for a gate g. Consider a next state line n that does not depend on a particular present state line p. In this case, *any* irredundancy test for any gate g feeding the next state line n, will have the bit position corresponding to p as a $-$ (*don't care*). This implies that we can choose the value of p to be 0 or 1 to satisfy conditions 1–4 of Section 3.4. For instance, in condition 2, *one of the two states corresponding to* s_1 (with p as a $-$) should go to a state s_2 on the application of i_1, such that $\exists\, i_2$ where i_2 @ s_2 produces a 0 at the next state line n. The condition has thus been relaxed. The other constraints are also relaxed in the case of obtaining multiple irredundancy tests for any particular gate.

Thus, a heuristic state assignment strategy tailored toward maximizing gate-delay-fault testability corresponds to the integration of the constraints given by conditions 2 and 4 of Section 3.4 into the algorithms of SACRED. During each step of the SACRED algorithm, codes are chosen based on minimizing the dependence of the next state lines on the present state lines, and attempting to satisfy the constraints posed by conditions 2 and 4. It should be noted that the program SACRED as presented in [16] had a different goal from minimizing the combinational dependence of the next state lines on the present state lines. Minimizing the sequential depth, i.e., the longest feedback path in the circuit was a primary objective.

TABLE I
RESULTS OF GATE-DELAY FAULT COVERAGE

Circuit	#input	#output	#state	gate-delay fault cov.	
				sacred	one-hot
shiftreg	1	1	8	100.0%	100.0%
tav	4	4	4	100.0%	100.0%
train11	2	1	11	95.9%	100.0%
bbara	4	2	10	96.8%	100.0%
cse	7	7	16	95.9%	99.7%
dk14	3	5	7	97.0%	100.0%
malati	4	4	24	99.4%	96.2%
s1	8	6	19	97.9%	99.6%
donfilex	1	1	24	96.9%	99.6%
tbk	6	3	32	98.4%	100.0

4.2. *Experimental Results*

In this section, we give empirical results on the testability of implementations using one-hot and SACRED encoding followed by two-level minimization and constrained algebraic factorization. The gate-delay-fault coverage is measured for a set of MCNC benchmark finite state machines. In our experiments no modifications to the STG were made. Gate-delay-fault coverage is obtained by the following: While gate-delay-faults remain undetected and paths through the fault have not been considered: 1) select an unconsidered path through the undetected gate-delay-fault, 2) generate a path-delay-fault vector pair for the path using the test generation procedure described in Section V, and 3) determine gate-delay-fault coverage due to the vector pair.

It is assumed that the required state s_2 in the second vector can be derived by either functional justification or scan shifting. The details of test generation strategy are given in Section V. Table I shows the percentages of testable gate-delay fault coverage of two different implementations. Since we do not modify the state transition graphs, the properties described in the previous sections may not be satisfied and, thus, 100% testability is not achieved in some examples. None of the SACRED implementations listed in Table I has a pure pipeline structure except *shiftreg*. Thus, coverage less than 100% is also expected in the SACRED encoding cases. However, as we expect, the gate-delay-fault coverage of both one-hot and SACRED implementations is always higher than 95%.

V. STRATEGIES FOR TEST GENERATION

5.1 *Introduction*

Test generation for delay-faults under standard scan design is similar to a two time-frame sequential circuit test generation problem [17]. In the first time-frame, all primary inputs and present state lines are fully controllable. In the second time-frame, only the primary inputs are fully controllable. There are two ways to produce the required state of the second time-frame: *functional justification* or *scan shifting*.

In functional justification, the values applied to the PI's and PS lines of the first time-frame not only sensitize proper paths for the given delay-fault but also produce the required state for the second time-frame after the clock is activated. In scan shifting, the required state of the second time-frame is produced using the test mode. The value of a present state line in the second time-frame is obtained from its predecessor in the scan chain by setting the mode switch to the scan mode before the clock is applied. Fig. 6 shows the concept. The scan shifting concept has been proposed in [12].

We assume the general robust fault model for test generation, but weaken it slightly. For a given path-delay-fault in the general robust delay fault model the required sensitization values on the side-inputs of gates along the path are listed in Table II [18]. For example, to sensitize a rising transition (i.e., 0 in time-frame I and 1 in time-frame II) from an AND gate's input to its output, the value 1 is required in time-frame II on each side-input while there is no requirement (*don't care* $-$) in time-frame I. However, for an OR gate, a stable 0 is required in both time-frames. To assign a stable value for a duration of two frames may require the use of a multi-valued logic. As a result the computational cost of test generation may increase. Therefore, we adopt an approximate sensitization requirement that is listed in Table III. Instead of requiring a stable value at the side-inputs of gates along the given path, it requires that these signals have the same final value at both time frames. The condition given in Table III is weaker than that given in Table II. A vector pair that sensitizes a path and satisfies the requirements given in Table III for side-inputs may cause hazards. Therefore, our procedure that is based on Table III should be considered as an approximate method for generating robust tests. The side-effect of this approximation is that hazards on side-inputs can potentially mask delay defects on the path under test.

5.2. *Generating Tests in Standard-Scan Circuits*

In a synchronous digital sequential circuit, a combinational sub-circuit between flip-flops naturally defines a combinational logic circuit. Given that observation it is natural to speak of the *ON*-set and *OFF*-set of the cone of logic feeding a flip-flop. In order to conduct functional justification efficiently, we first preprocess the circuit to enumerate the partial or complete (memory and CPU time permitting) *ON* and *OFF*-sets of each of the NS lines [19]. Cover enumeration is done via a PODEM-based enumeration algorithm. A limit on the number of cubes in each *ON* or *OFF*-set can be replaced. This limit is used to restrict the amount of memory and CPU time used for enumeration. Cover enumeration is generally very fast and full covers of moderately large circuits can be extracted easily.

Given the covers for all NS lines, the test generation for a given path and the fault type (rising or falling) follows three steps.

1) Generating a pattern (i_2 @ s_2) for time-frame II which satisfies all the value requirements given in

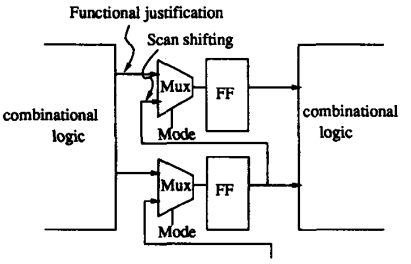

Fig. 6. Scan design structure.

TABLE II
SENSITIZATION REQUIREMENTS ON SIDE-INPUTS OF GATES ALONG A GIVEN PATH

transition on on-path input	Gate Type: AND or NAND TF-I	AND or NAND TF-II	OR or NOR TF-I	OR or NOR TF-II
Rising	–	1	A stable 0	
Falling	A stable 1		–	0

TABLE III
APPROXIMATE SENSITIZATION REQUIREMENTS ON SIDE-INPUTS OF GATES ALONG A GIVEN PATH

transition on on-path input	Gate Type: AND or NAND TF-I	AND or NAND TF-II	OR or NOR TF-I	OR or NOR TF-II
Rising	–	1	0	0
Falling	1	1	–	0

Table III and having as many *don't care* (–) entries in s_2 as possible. This step is done using a PODEM-like algorithm.

2) If a present state (PS) line in has a 1 (0) in s_2, the *ON*-set (*OFF*-set) of the corresponding NS line is picked. The intersection of the *ON* and *OFF*-sets of the NS lines with 0's and 1's gives the set of input vectors ($C(s_2)$) that would produce s_2.
3) Generating a pattern (i_1 @ s_1) for time-frame I which satisfies all the value requirement given in Table III and is covered by $C(s_2)$. If no such pattern exists, backtrack and repeat from Step 1.

The complexity of processing time-frame I is typically much higher than the traditional combinational circuit test generation for stuck-at-faults. It requires not only sensitizing a given path but also producing a specified state at the NS lines. This may result in a lot of backtracking. Step 2 described above, which is a very fast process, significantly reduces the amount of backtracking in time-frame I. It is particularly helpful in proving that a fault is redundant. One extra run of step 2 is required for each backtrack between time-frame I and II (i.e., when no test is found in step 3). In this case, we store the unjustifiable state to avoid unnecessary justification in the future.

The average test generation complexity using scan shifting is typically lower than that using functional justification. Test generation using scan shifting also follows a three-step procedure which is very similar to that using functional justification except that the second step is much simpler. In step 2, if a PS line has a 1 (0) in s_2, we would assign a 1 (0) in its predecessor in the scan chain at s_1. Test generation for time-frame I then starts with the partially assigned state.

A fault which is redundant assuming the use of functional justification may become testable using scan shifting. Similarly, a fault which is redundant assuming the use of scan shifting may become testable using functional justification. Since the test generation complexity using scan shifting is lower, we first try test generation in this mode. If a fault is identified to be redundant, the test generator will be switched to the functional justification mode and test generation will be attempted again.

5.2.1. Experimental Results We implemented the algorithm in the C language and used it to conduct experiments on the ISCAS sequential benchmark circuits. Table IV gives the circuit statistics and the total number of path/gate-delay-faults. We randomly selected 5000 paths (10 000 faults) for test generation if the circuit has more paths. To achieve real defect coverage in actual practice it would be more useful to select a set of paths that cover each gate in the circuit, or to test all critical paths.

In all experiments, we ran test generation for each fault in the fault list. No fault simulation was conducted.

For each circuit, we conducted our experiment in the following way: first, we ran the test generator assuming that the PS lines are fully controllable in both time frames. In other words, we assume the circuit is implemented using enhanced scan flip-flops that can store two bits of state. Table V shows the achievable fault coverage under the enhanced scan design environment. The percentages of detected and redundant path-delay-faults are listed. The last column gives the average run time per fault in CPU seconds on SUN4/260. We also compute the percentage of gate-delay-faults detected by the tests of the testable path-delay-faults. The number is listed in column 5 of Table V. If a fault is redundant in the enhanced scan design environment, it must be redundant in the standard scan design environment. Therefore, we only use the enhanced-scan testable faults in our standard-scan test generation experiments.

We then ran the test generator in scan shifting followed by functional justification mode. For a given fault, scan shifting is first tried. In this mode, the required value in a flip-flop of the second pattern is obtained from its predecessor in the scan chain as described above. If it is proven redundant in this mode, functional justification follows. In this mode, the required state of the second pattern is justified through the functional logic. Table VI gives the results we have obtained for standard-scan delay test generation. A redundant fault reported here is redun-

TABLE IV
ISCAS-89 SEQUENTIAL BENCHMARK CIRCUITS

Circuit	number of gates	number of flip-flops	number of gate-delay-faults	number of path-delay-faults
s27	10	3	34	56
s208	96	8	230	290
s298	119	14	272	462
s344	160	15	368	710
s382	158	21	364	800
s400	148	21	372	900
s420	196	16	462	738
s444	181	21	410	1070
s510	211	6	472	738
s526	194	21	434	820
s526n	193	21	436	816
s820	289	5	624	984
s832	287	5	620	1012
s953	395	32	880	2266
s1196	529	18	1122	5998
s1238	508	18	1080	5998
s1488	653	6	1334	1924
s1494	653	6	1322	1952
s5378	2779	179	6038	>10000
s9234	5597	228	11688	>10000

TABLE V
RESULTS OF TEST GENERATION ASSUMING ENHANCED SCAN DESIGN

Circuit	Path-delay-fault detected	Path-delay-fault redundant	gate-fault covered	ave. CPU sec. per fault
s27	89.3%	10.7%	100.0%	0.005
s208	100.0%	0.0%	100.0%	0.012
s298	76.2%	23.8%	100.0%	0.013
s344	90.6%	9.4%	98.6%	0.021
s382	88.0%	12.0%	100.0%	0.031
s400	80.3%	19.7%	99.2%	0.030
s420	100.0%	0.0%	100.0%	0.026
s444	67.6%	32.4%	98.8%	0.040
s510	100.0%	0.0%	100.0%	0.030
s526	86.3%	13.7%	100.0%	0.022
s526n	87.0%	13.0%	100.0%	0.022
s820	100.0%	0.0%	100.0%	0.046
s832	98.4%	1.6%	99.7%	0.050
s953	100.0%	0.0%	94.8%	0.054
s1196	59.5%	40.5%	99.3%	0.124
s1238	51.2%	48.8%	97.6%	0.133
s1488	99.6%	0.4%	100.0%	0.089
s1494	98.7%	1.3%	99.8%	0.089
s5378	79.8%	15.2%	75.7%	1.269
s9234	4.6%	95.4%	5.9%	0.694
average	82.9%	16.9%	93.5%	0.140

dant with respect to both modes. In the scan shifting mode, we assume the order of the flip-flops in the scan chain follows the same order of their appearance in the netlist file. The percentage of path-delay-fault coverage is about 64.9%. For those circuits that all path-delay-faults are considered for test generation, the percentage of gate-delay-faults covered by the test sequence of the detectable path-delay-faults is always higher than 80%. On an average, it takes about eight times more CPU seconds per fault during standard-scan test generation when compared to that required in an enhanced scan design environment.

TABLE VI
RESULTS OF TEST GENERATION USING BOTH SCAN SHIFTING (WITH DEFAULT SCAN-CHAIN ORDER) AND FUNCTIONAL-JUSTIFICATION

Circuit	Path-delay-fault detected	Path-delay-fault redundant	gate-fault covered	ave. CPU sec. per fault
s27	100.0%	0.0%	100.0%	0.00
s208	60.7%	39.3%	95.2%	0.04
s298	59.4%	40.6%	88.2%	0.04
s344	62.8%	37.2%	92.7%	0.20
s382	54.8%	45.2%	96.2%	0.34
s400	54.9%	45.1%	94.9%	0.30
s420	52.4%	47.6%	92.6%	0.16
s444	54.2%	45.6%	93.4%	0.46
s510	57.6%	42.4%	91.7%	0.21
s526	53.5%	46.5%	84.3%	0.44
s526n	53.7%	46.3%	84.2%	0.44
s820	64.9%	35.1%	87.3%	1.62
s832	64.8%	35.2%	86.8%	1.65
s953	56.6%	43.2%	88.7%	0.57
s1196	99.9%	0.01%	99.2%	0.26
s1238	99.9%	0.01%	97.5%	0.26
s1488	57.3%	42.7%	87.9%	0.89
s1494	57.1%	42.9%	80.0%	0.88
s5378	78.7%	3.4%	58.4%	8.12
s9234	54.3%	19.8%	5.7%	8.20
average	64.9%	32.9%	85.2%	1.25

5.3. *Efficient Ordering of Scan Flip-Flops*

In this section, we present an algorithm to compute an efficient ordering of the flip-flops in the scan-chain that maximizes the coverage of the delay faults. The algorithm uses the information derived from running a delay test generator on the circuit.

We first run the test generator under standard scan using functional justification. Notice that the order of the flip-flops in the scan chain is irrelevant to test generation under this mode. If a test pair is found, this fault would be detectable with any scan order. If the fault is not detectable using functional justification, we run test generation again in enhanced scan mode. The test generator attempts to have as many *don't care* (−) entries as possible in the present state lines in the second vector of the two vector sequence. Once the test pair $(v_1, v_2) = (i_1 @ s_1, i_2 @ s_2)$ is generated, a set of constraints on the scan ordering is computed and recorded. These constraints, if satisfied, guarantee that s_2 is obtainable by scan shifting of v_1 in standard scan. In general, if the value of flip-flop X is 0 (1) in s_1 and the value of flip-flop Y is 1 (0) in s_2, then the constraint is that flip-flop X cannot be the predecessor of flip-flop Y. For a circuit of n flip-flops, we can easily record the constraints by maintaining an $n \times n$ constraint matrix A. Initially all entries are zero. Given a test vector-pair for a fault f, if flip-flop X is not allowed to be the immediate predecessor of flip-flop Y in the scan-chain, then entry $A_{X,Y}$ is increased by one.

Consider a circuit of two inputs and three flip-flops. Suppose the target delay-fault is redundant in functional justification mode and the test pair generated in enhanced scan mode is $\langle -1 @ 0 - 1,0 - @101 \rangle$. The constraint

matrix A which is initialized to 0 will become:

$$\begin{bmatrix} - & 0 & 1 \\ 0 & - & 0 \\ 0 & 1 & - \end{bmatrix}.$$

We continue to update the constraint matrix until all faults in the fault list are processed. Given a particular set of test vector-pairs, the final value of $A_{i,j}$ can be interpreted as the number of faults that will not be detected by scan shifting if flip-flop i is the immediate predecessor of flip-flop j in the scan-chain. Notice that a delay-fault may have more than one test. Here we just consider one test during the construction of the constraint matrix. Therefore, even if the constraints derived from the test are not completely satisfied, the fault may still be detected by a test other than the one that was used to construct the constraint matrix. Hence, the value of $A_{i,j}$ is simply the upper-bound of the number of undetectable faults by scan shifting if flip-flop i is the predecessor of flip-flop j under the vector set used to construct the matrix. Once all faults are processed, we determine a scan ordering to satisfy most constraints.

Having constructed the constraint matrix the routine DETERMINE_ORDER in Fig. 7 tries to find a scan ordering such that a minimum number of constraints are violated. It selects the flip-flops one by one to form the scan chain. The first flip-flop (head) of the scan-chain, which is not a successor of any flip-flop is determined first. The natural flip-flop to choose to start the chain is the one that would violate the largest number of constraints if it was successor of any other flip-flop in the scan-chain; however, as we don't yet know what the scan-chain will be, we cannot compute this exactly. As a heuristic we choose as head of the scan chain a flip-flop that has the largest column sum of the constraint matrix A. We then repeatedly determine the next flip-flop Y of the scan-chain that has the minimum value of $A_{i,Y}$ where i is the previously selected flip-flop.

Example: Consider a circuit of two inputs, three flip-flops and four delay-faults. Suppose all four faults cannot be detected in functional justification mode and the vector pairs generated under enhanced scan are given as in Table VII.

The order of the scan-chain has nothing to do with the values of i_1 and i_2. The constraint matrix that is initialized to zero becomes:

$$\begin{bmatrix} - & 0 & 1 \\ 1 & - & 1 \\ 1 & 0 & - \end{bmatrix}, \begin{bmatrix} - & 0 & 1 \\ 2 & - & 2 \\ 1 & 0 & - \end{bmatrix}, \begin{bmatrix} - & 0 & 1 \\ 2 & - & 2 \\ 1 & 0 & - \end{bmatrix}$$

and

$$\begin{bmatrix} - & 1 & 1 \\ 2 & - & 2 \\ 2 & 0 & - \end{bmatrix}$$

respectively, after processing each fault.

Routine DETERMINE_ORDER selects flip-flop 1 as the head of the scan-chain because it has the largest column sum (equal to 4) of the constraint matrix. Since $A_{1,2} = A_{1,3} = 1$ and column sum of flip-flop 3 (equal to 3) is larger than that of flip-flop 2 (equal to 1), flip-flop 3 is selected as the next element of the scan-chain. The last element of the scan chain is then flip-flop 2. In this scan order, the required state of vector 2 (s_2) can be generated from s_1 by scan shifting the tests for fault 2, 3 and 4 with the given vector pairs shown above. Notice that fault 1 may still be detected if it has other tests beside the one given in the table above. Note also that if the order of the flip-flops in the scan-chain is flip-flop 1, 2 and 3, only fault 3 can be detected by its test vector pair given above.

```
DETERMINE_ORDER()
{
  F: SET OF FLIP-FLOPS WHOSE ORDER IS NOT DECIDED YET. INITIALLY, IT CONTAINS ALL
  FLIP-FLOPS.
  SELECT FLIP-FLOP X WHICH HAS THE LARGEST COLUMN SUM OF A AS THE HEAD OF THE
  SCAN CHAIN (COLSUM(X) = ∑_{1≤i≤n, i∈F} A_{i,X});
  CURRENT_TAIL = X;
  REMOVE X FROM F;
  WHILE(F IS NOT EMPTY)
  {
    SELECT FLIP-FLOP Y AS THE NEXT ELEMENT IN THE SCAN CHAIN SUCH THAT Y HAS THE
    MINIMUM NUMBER OF CONFLICTS AS DETERMINED BY THE CONSTRAINT MATRIX, I.E.,
      ∀i ∈ F
        1. A_{current_tail,Y} < A_{current_tail,i} OR
        2. A_{current_tail,Y} == A_{current_tail,i} AND COLSUM(Y) ≥ COLSUM(i)
    CURRENT_TAIL = Y;
    REMOVE Y FROM F;
  }
}
```

Fig. 7. Algorithm for determining scan-chain order.

TABLE VII
VECTORS DETECTING FOUR FAULTS

Fault	i_1	s_1	i_2	s_2
1	00	000	01	1 − 1
2	−0	− 1 −	01	0 − 0
3	− −	1 − 0	− −	0 − 1
4	10	110	10	10−

5.3.1. Experimental Results Table VIII gives the results for two different orderings of the scan-chain: one follows the same order of their appearance in the netlist file (called *default order*) and the other determined by our algorithm give in Section 5.3, called *efficient order*.

Since a fault that is redundant in the enhanced-scan circuit must be redundant in both functional justification and scan shifting modes, we only use the enhanced-scan testable faults in our standard-scan test generation experiments. The fault coverage is computed as the detected path-delay-faults over total enhanced-scan testable faults.

In all examples except one (s510), the fault coverage of the best scan order is higher than that of the default scan order. In average of 20 circuits, the former is about 20% higher than the latter. In the case of s510, almost all entries of the constraint matrix have similar values. Thus, different scan orderings will give similar fault coverages. The time given under column "efficient order" includes

TABLE VIII
PATH-DELAY-FAULT COVERAGE AND RUN TIME USING BOTH SCAN SHIFTING (WITH EITHER DEFAULT OR EFFICIENT SCAN-CHAIN ORDER) AND FUNCTIONAL JUSTIFICATION

	default order		efficient order	
Circuit	fault coverage	ave. CPU sec. per fault	fault coverage	ave. CPU sec. per fault
s27	100.0%	0.00	100.0%	0.00
s208	60.7%	0.04	69.0%	0.05
s298	59.4%	0.04	82.1%	0.05
s344	62.8%	0.20	92.7%	0.21
s382	54.8%	0.34	95.6%	0.17
s400	54.9%	0.30	95.3%	0.19
s420	52.4%	0.16	85.0%	0.15
s444	54.2%	0.46	96.4%	0.11
s510	57.6%	0.21	56.4%	0.23
s526	53.5%	0.44	78.1%	0.45
s526n	53.7%	0.44	78.0%	0.45
s820	64.9%	1.62	66.7%	0.49
s832	64.8%	1.65	66.4%	0.51
s953	56.6%	0.57	100.0%	0.09
s1196	99.9%	0.26	99.9%	0.29
s1238	99.9%	0.26	99.9%	0.29
s1488	57.3%	0.89	61.4%	0.94
s1494	57.1%	0.88	57.3%	1.03
s5378	78.7%	8.12	100.0%	5.39
s9234	54.3%	8.20	100.0%	5.92
average	64.9%	1.25	83.8%	0.85

the time for determining the scan order. I.e., the average CPU time per fault is computed as: (total time for determining the scan order + total test generation time for the circuits using the efficient scan order)/total faults. In the step of determining the scan ordering, we did not run the test generator in the functional justification mode to save run time for this step. In this step, for each enhanced-scan testable fault, a test pair is generated under the enhanced scan mode and the test pair is used for constructing the constraint matrix. Because the CPU time consumed by routine DETERMINE_ORDER ( ) is very small, the time reported under column "efficient order" is very close to the sum of the test generation time under enhanced scan and the test generation time under standard scan using the efficient scan order.

The test generation time in scan shifting mode is significantly faster than that in functional justification mode. If the efficient scan order is used, more faults are detected in scan shifting mode and, thus, less faults need to be tried under functional justification. This explains why the run time of the efficient scan order is about 30% faster.

5.4. Selecting Flip-Flops to be Made Enhanced-Scan

Even with an efficient order of the flip-flops in the scan chain, a certain percentage of delay-faults that are detectable in the enhanced scan design environment remain undetectable in the standard scan design environment. If the fault coverage in standard scan environment is not acceptable, a *partial-enhanced-scan* design methodology can be used. In partial-enhanced-scan design, a subset of flip-flops is selected and made enhanced scan. The corresponding present state lines of the enhanced-scan flip-flops are fully controllable in both time-frames in test generation. Many of the standard-scan undetectable faults will become detectable in the partial-enhanced-scan environment and a required level of delay-fault coverage may be achieved.

Given a set of faults that are testable under enhanced-scan but are redundant in either functional justification mode or scan shifting mode with a given ordering of flip-flops, the proposed method attempts to minimize the number of flip-flops to be made enhanced-scan so as to detect all or a specified percentage of the given delay-faults.

In the following, we first give the formulation of the problem. Since finding the optimal solution requires solving an *NP*-complete problem, we resort to heuristics to find an efficient solution.

Let us denote the *ON*-set and *OFF*-set of a flip-flop ff_i as $ON(ff_i)$ and $OFF(ff_i)$ and the value of a flip-flop ff_i on vector j as $v_j(ff_i)$. The set of flip-flops that is neither enhanced-scan (i.e., is standard-scan) nor the head of the scan chain is denoted as *SS*. In the partial-enhanced-scan design environment, if a given fault f is detectable in the functional justification mode, it means that there exists a vector pair (v_1, v_2) that satisfies the following properties:

(1.1) the vector pair sensitizes the necessary paths for the detection of the given delay-fault;

(1.2) $$v_1 \sqcap \prod_{\forall ff_i \in SS \,\&\, v_2(ff_i)=1} ON(ff_i) \sqcap \prod_{\forall ff_j \in SS \,\&\, v_2(ff_j)=0} \cdot OFF(ff_j) \neq \varnothing.$$

The second (third) term in **(1.2)** is the cube intersection of the *ON*-sets (*OFF*-sets) of those flip-flops that are not enhanced-scan and their value in v_2 is 1 (0). If these three cubes are compatible (i.e., their cube intersection is not empty), then it will be possible to generate vector 2 through functional justification.

For a given scan ordering, we denote the predecessor of a flip-flop ff_i as $P(ff_i)$. If a given fault f is detectable in the scan shifting mode, it means that there exists a vector pair that satisfies the following properties:

(2.1) the vector pair sensitizes the proper paths for the given delay-fault;

(2.2) $\forall ff_i \in$ SS and $v_2(ff_i) = 0$, $v_1(P(ff_i)) =$ 0 or $-$;

(2.3) $\forall ff_i \in$ SS and $v_2(ff_j) = 1$, $v_1(P(ff_j)) =$ 1 or $-$.

Property **(2.2)** simply means that for every standard-scan flip-flop whose required value in v_2 is 0, the value of its predecessor in the scan-chain in v_1 must be either 0 or don't-care. Property **(2.3)** has a similar meaning.

The procedure of selecting flip-flops for partial enhanced scan is summarized as follows:

FOR EACH FAULT (f) THAT IS ENHANCED-SCAN DETECTABLE BUT STANDARD-SCAN UNDETECTABLE {
  GENERATE A SET OF VECTOR PAIRS, $T(f)$, THAT DETECT f UNDER ENHANCED-SCAN;

FOR EACH VECTOR PAIR $i \in T(f)$ OBTAIN A MINIMAL SET OF FLIP-FLOPS $FF_i^{fj}(f)$ FOR ENHANCED-SCAN SUCH THAT **(1.2)** IS SATISFIED; /* *In this case, the required state of vector 2 can be produced through functional justification.* */

FOR EACH VECTOR PAIR $i \in T(f)$, OBTAIN A MINIMAL SET OF FLIP-FLOPS $FF_i^{ss}(f)$ FOR ENHANCED-SCAN SUCH THAT **(2.2)** AND **(2.3)** ARE SATISFIED; /* *In this case, the required state of vector 2 can be produced through scan shifting.* */

}FIND A MINIMAL SET (ES) OF FLIP-FLOPS FOR ENHANCED-SCAN SUCH THAT, FOR EVERY FAULT OR A GIVEN PERCENTAGE OF FAULTS, AT LEAST ONE SET OUT OF THE SETS $FF_i^{fj}(f)$ AND $FF_i^{s}(f)$ IS COVERED BY ES.

If the set of the enhanced-scan flip-flops ES covers set $FF_i^{fj}(f)$, then vector pair i, that would detect fault f, can be produced through functional justification. Similarly, if it covers $FF_i^{ss}(f)$, then the corresponding vector pair can be produced through scan shifting. If ES covers either one of them for some vector pair i, the fault will be detectable in this partial-enhanced-scan environment.

The complexity of computing $FF_i^{fj}(f)$ is very high while that of $FF_i^{ss}(f)$ is relatively easy. Thus, in our experiments, we only computed FF_i^{ss} and computed only one test vector pair per fault (i.e., restricted the size of $T(f)$ to 1).

Example: Consider a circuit of two inputs, five flip-flops (a, b, c, d and e) and four enhanced-scan detectable but standard-scan undetectable faults. The vector pairs generated under enhanced-scan for these four faults are listed in Table IX. Here only one vector pair is generated under enhanced scan ($n = 1$) and only FF_1^{ss} is computed. Assume the scan ordering is: scan_in $\rightarrow a \rightarrow b \rightarrow c \rightarrow d \rightarrow e \rightarrow$ scan_out. Hence, $P(b) = a$, $P(c) = b$, $\cdots$ etc.

The sets of $FF_1^{ss}(f1)$, $FF_1^{ss}(f2)$, $FF_1^{ss}(f3)$ and $FF_1^{ss}(f4)$ are $\{c, d\}$, $\{c\}$, $\{d\}$, $\{b, d\}$. If flip-flops b, c and d are selected to be enhanced-scan, all four faults will be detected in the partial-enhanced-scan environment using scan shifting. If only flip-flops c and d are selected, $f1, f2$ and $f3$ will be covered. In this case, fault $f4$ may still be covered if (1) vector $\langle 1 - @11000 \rangle \sqcap ON(a) \sqcap OFF(b) \neq \varnothing$ or (2) there are other test pairs for $f4$ that satisfies either **(1.1)** and **(1.2)** or **(2.1)**, **(2.2)** and **(2.3)**.

Once the sets FF_1^{ss} are computed, the selection of a set of flip-flops to cover the largest number of faults is equivalent to a set covering problem [20] (an NP-complete problem). The following heuristic is used.

1) Compute the frequency of every flip-flop unselected thus far. The frequency is the number of faults f_i whose set $FF_1^{ss}(f_i)$ contains that flip-flop.
2) Select the flip-flop having the highest frequency.
3) Eliminate the faults whose set FF_1^{ss} are covered by the current set of selection and estimate the fault coverage.
4) If the estimated fault coverage is lower than the required level, go to step 1.

TABLE IX
VECTORS DETECTING FOUR FAULTS

Fault	i_1	$s_1(abcde)$	i_2	$s_2(abcde)$
$f1$	-0	00000	-0	$1-11-$
$f2$	$--$	$-1-1-$	$--$	$0-0-1$
$f3$	11	$1-10-$	10	$0-100$
$f4$	$1-$	11000	$1-$	$10-1-$

The user first specifies the required delay-fault coverage. During this process, the fault coverage is estimated as the number of faults whose FF_1^{ss} are covered by the current set of selected enhanced-scan flip-flops plus the number of standard-scan detectable faults divided by the total number of enhanced-scan detectable faults. Since a fault whose FF_1^{ss} is not covered by the enhanced-scan flip-flops is not necessarily partial-enhanced-scan undetectable, this estimation is a lower bound of the coverage. To compute the final fault coverage, test generation of those uncovered faults in the partial-enhanced-scan environment should be re-run.

5.4.1 Experimental Results Table X shows the results of partial-enhanced-scan by specified 100% and 95% coverage as the desirable coverage. Notice that the fault coverage is computed as the detected faults over total enhanced-scan testable faults. The faults that are redundant under enhanced-scan are excluded from consideration. The order of the flip-flops in the scan-chain is assumed the efficient order determined by the method described in Section 5.3. Table X gives the number and percentage of selected enhanced-scan flip-flops. Once the flip-flops are selected, the test generation is re-run assuming the selected flip-flops are enhanced-scan and the remaining flip-flops are standard-scan. The final fault coverage is also listed in the table.

5.5. Computing Test Vector Sets

5.5.1. Introduction In order to reduce test application time it is describe to produce a test vector set that is as compact as possible. While fault simulation can be used to dynamically compact a test vector set, we take a more global approach to test vector compaction (within a single output of a function). In hazard-free robust path-delay-fault testing only a single path is sensitized by each test vector pair and their is no room for vector compaction. However, for the general robust or non-robust path-delay-fault model, multiple paths may be sensitized. The fact that multiple paths may be simultaneously sensitized allows for vector compaction. During the test generation process, vector pairs are produced that detect path delay faults in such a way that the vectors have a maximal number of undefined values. Recall from Section II that two vectors are compatible if their values do not conflict in any input. Given this compatibility relation a compatibility graph can be generated and a clique covering algorithm can be applied to compact compatible vectors. This will be discussed in detail in Section 5.5.4.

TABLE X
RESULTS OF PARTIAL-ENHANCED-SCAN

		Selection Criterion					
		100%			95%		
Circuit	Total Flip-Flops	Selected FFs #	Selected FFs %	Final fault covg.	Selected FFS #	Selected FFS %	Final fault covg.
s27	3	0	0%	100%	0	0%	100%
s208	8	7	88%	100%	7	88%	100%
s298	14	5	36%	100%	2	14%	99.4%
s344	15	9	60%	100%	5	33%	98.3%
s382	21	4	19%	100%	2	10%	96.6%
s400	21	4	19%	100%	2	10%	96.3%
s420	16	15	94%	100%	13	81%	95.8%
s444	21	4	19%	100%	2	10%	97.8%
s510	6	5	83%	100%	5	83%	100%
s526	21	4	19%	100%	3	14%	98.4%
s526n	21	4	19%	100%	3	14%	98.4%
s820	5	4	80%	100%	4	80%	100%
s832	5	4	80%	100%	4	80%	100%
s953	32	0	0%	100%	0	0%	100%
s1196	18	0	0%	100%	0	0%	100%
s1238	18	0	0%	100%	0	0%	100%
s1488	6	5	83%	100%	5	83%	100%
s1494	6	5	83%	100%	5	83%	100%
s5378	179	0	0%	100%	0	0%	100%
s9234	228	0	0%	100%	0	0%	100%
average	—	—	39%	100%	—	34%	99.1%

Vector pairs produced from test generation using scan shifting cannot be compacted with vector pairs produced using functional justification, since the two test application methods are fundamentally different, so we will consider the compaction of scan shifting vectors and functionally justified vectors individually. We will describe our compaction strategy for path-delay-faults and gate-delay-faults. We first show the correctness of our compaction strategy for circuits implemented under standard scan using the scan shifting approach and then will discuss the functional justification case.

5.5.2. Correctness of Compaction Strategy We now show in detail how vector pairs produced by scan shifting can be compacted.

Theorem 5.1: Let $(i_{k1} @ s_{k1}, i_{k2} @ s_{k2})$ and $(i_{l1} @ s_{l1}, i_{l2} @ s_{l2})$ be two scan-shift vector pairs detecting the path delay faults π_k and π_l respectively. If i_{k1}, is compatible with i_{l1}, s_{k1} is compatible with s_{l1} and i_{k2} is compatible with i_{l2} then $(i_{k1} \sqcap i_{l1} @ s_{k1} \sqcap s_{l1}, i_{k2} \sqcap i_{l2} @ \mathit{shift}(s_{k1} \sqcap s_{l1}))$ can be applied as a scan-shift vector pair that detects path-delay-faults on both π_k and π_l.

Proof: Since s_{k1} is compatible with s_{l1} for any kind of bit-wise shift (e.g., shift left by 1), $\mathit{shift}(s_{k1})$ is compatible with $\mathit{shift}(s_{l1})$. We know that $\mathit{shift}(s_{l1}) \sqsubseteq s_{l2}$ and $\mathit{shift}(s_{k1}) \sqsubseteq s_{k2}$ because $(i_{k1} @ s_{k1}, i_{k2} @ s_{k2})$ and $(i_{l1} @ s_{l1}, i_{l2} @ s_{l2})$ are each scan-shifted vector pairs. This means that $\mathit{shift}(s_{k1}) \sqcap \mathit{shift}(s_{l1}) \sqsubseteq s_{k2}$ and $\mathit{shift}(s_{k1}) \sqcap \mathit{shift}(s_{l1}) \sqsubseteq s_{l2}$. This in turn implies that $\mathit{shift}(s_{k1} \sqcap s_{l1}) \sqsubseteq s_{k2}$ and $\mathit{shift}(s_{k1} \sqcap s_{l1}) \sqsubseteq s_{l2}$. Therefore, given that i_{k1} is compatible with i_{l1} and i_{k2} is compatible with $i_{l2}(i_{k1} \sqcap i_{l1} @ s_{k1} \sqcap s_{l1}, i_{k2} \sqcap i_{l2} @ \mathit{shift}(s_{k1} \sqcap s_{l1}))$ is a scan-shift vector pair that detects path delay faults on both π_k and π_l. ■

We now show that the vector pairs produced by functional justification can also be compacted. First some notation: Given a vector pair $(i_{k1} @ s_{k1}, i_{k2} @ s_{k2})$ $\mathit{next\_states}(s_{k1})$ denotes the next states reached on the application of the input i_{k1} when the machine is in any of the states given by s_{k1}.

Theorem 5.2: Let $(i_{k1} @ s_{k1}, i_{k2} @ s_{k2})$ and $(i_{l1} @ s_{l1}, i_{l2} @ s_{l2})$ be two functional-justification vector pairs detecting the path delay faults π_k and π_l respectively. If i_{k1} is compatible with i_{l1}, s_{k1} is compatible with s_{l1} and i_{k2} is compatible with i_{l2}, then $(i_{k1} \sqcap i_{l1} @ s_{k1} \sqcap s_{l1}, i_{k2} \sqcap i_{l2} @ \mathit{next\_states}(s_{k1} \sqcap s_{l1}))$ is a functional-justification vector pair that detects path-delay-faults on both π_k and π_l.

Proof: Denote $\mathit{next\_states}(s_{k1})$ as the next states reached when the machine is in any of the states given by s_{k1} on the application of input i_{k1}, and $\mathit{next\_states}(s_{l1})$ as the next states reached when the machine is in the states s_{l1} on the application of input i_{l1}. Since the vector pairs can be functionally justified, we know that $\mathit{next\_states}(s_{k1}) \sqsubseteq s_{k2}$ and $\mathit{next\_states}(s_{l1}) \sqsubseteq s_{l2}$. This means that $\mathit{next\_states}(s_{k1}) \sqcap \mathit{next\_states}(s_{l1}) \sqsubseteq s_{k2}$ and $\mathit{next\_states}(s_{k1}) \sqcap \mathit{next\_states}(s_{l1}) \sqsubseteq s_{l2}$. This in turn implies that $\mathit{next\_states}(s_{k1} \sqcap s_{l1}) \sqsubseteq s_{k2}$ and $\mathit{next\_states}(s_{k1} \sqcap s_{l1}) \sqsubseteq s_{l2}$, where $\mathit{next\_states}(s_{k1} \sqcap s_{l1})$ corresponds to the next states reached on the application of input $i_{k1} \sqcap i_{l1}$ when the machine is in the states $s_{k1} \sqcap s_{l1}$. We know that $\mathit{next\_states}(s_{k1})$ is compatible with s_{l1}, since i_{k1} is compatible with i_{l1} and s_{k1} is compatible with s_{l1} (the machine has to go to *some* state). Therefore, given that i_{k2} is compatible with $i_{l2}(i_{k1} \sqcap i_{l1} @ s_{k1} \sqcap s_{l1}, i_{k2} \sqcap i_{l2} @ \mathit{next\_states}(s_{k1} \sqcap s_{l1}))$ is a functional-justification vector pair that detects both π_k and π_l.

5.5.3. Compacting Delay-Fault Tests In order to compact path-delay-fault tests, we simply use the clique covering approach described in Section 5.5.4.

Compacting gate delay fault tests is a two-step process. We only require a test for *some* path that passes through the gate. Therefore, we pick a minimum set of testable paths that cover all the gates in the circuit in polynomial time using the algorithm developed in [21]. This gives us a set of paths, for which we have the test vector pairs. We then use the clique covering approach described in the next section to compact the test vector pairs.

5.5.4. Clique Covering We compact vectors by solving the clique covering problem [22] on an undirected graph built as follows: Each vertex V_i in the graph corresponds to a pair of test vectors (cubes). We have an edge $\langle V_i, V_j \rangle$ in the graph if the vector pairs V_i and V_j are compatible, according to the definition given in Section 5.5.1. We then proceed to find a clique covering of the graph. Recall that a clique is a fully connected subgraph in a graph with undirected edges between vertices. Clique covering corresponds to selecting a minimum number of cliques that cover all the vertices in the graph. Each clique corresponds to a vector pair that is covered by all the ver-

TABLE XI
TEST COMPACTION RESULTS OF ISCAS-89 BENCHMARK CIRCUITS

Circuit	Before Compaction		After Compaction	
	Scan	Justify	Scan	Justify
s27	100	0	10	0
s208	396	20	54	6
s298	518	60	65	13
s344	1150	42	135	16
s382	1336	10	157	3
s400	1372	6	161	2
s420	1154	108	125	14
s444	1382	12	143	4
s510	616	188	101	55
s526	1086	30	110	5
s526n	1082	30	108	5
s820	1028	284	183	86
s832	1064	252	190	72
s953	4532	0	572	0
s1196	7396	20	1310	10
s1238	7306	20	1212	10
s1488	1962	398	301	95
s1494	1774	442	265	117
s5378	15020	74	929	23
s9234	1452	0	244	0

tices (cubes, or vector pairs) in the clique. This vector is compatible with all the cubes in the clique and may therefore be used to simultaneously test all the faults associated with the vector pairs in the clique.

The above strategy works for V_i's that are composed of pairs of cubes, because cube compatibility is a transitive relation. That is, if c_1 is compatible with c_2 and c_2 is compatible with c_3, then c_1 is compatible with c_3. However, the same is *not* true of covers i.e., sets of cubes. If we generate more than one test pair per path delay fault, then our V_i's may be composed of several cube pairs; we only have to select a vector that is contained in any one of the cube pairs. Therefore, during clique covering, a check has to be made to see if the *sets of cubes*, corresponding to vertices in the selected clique, are compatible.

Motivated by the fact that clique covering is an NP-complete problem [22], we resort to efficient heuristics. We use a greedy heuristic that begins with a seed node, "grows" the node to a maximal clique, selects the clique, deletes all the vertices corresponding to the clique and iterates until all the vertices are selected.

5.5.5. Experimental Results Compaction results using the clique covering strategy are presented in Table XI. As can be seen, factors of up to 10 have been achieved by virtue of a global optimization strategy toward test compaction. The vector set for compaction is derived by our test generator running under standard scan for all path-delay faults. For s5238 and s9234 which have more paths, 5000 paths are randomly selected for consideration. For each fault, scan shifting is first attempted. The efficient scan ordering determined by the method described in Section 5.3 is assumed. If the fault is redundant in scan shifting, functional justification is further attempted.

VI. CONCLUSIONS

It is our belief that one of the principal obstacles to the wider practical application of test generation for faults in path-delay-fault and gate-delay-fault models is the inability to delay-test a sequential circuit using a standard-scan methodology. In this paper we have addressed this problem, and given a variety of techniques that can be used to generate delay tests for sequential circuits. Furthermore, we have shown how various synthesis techniques can be applied to improve the delay-fault testability of circuits in a standard scan methodology. It appears that entire integrated circuits of moderate complexity can be synthesized for high (nearly 100%) gate-delay-fault coverage, using our techniques.

ACKNOWLEDGMENT

This work was stimulated by valuable discussions with Vijay Iyengar, Ken Wagner, and Tom Williams.

REFERENCES

[1] A. Pramanick and S. Reddy, "On the design of path delay fault testable combinational circuits," in *Proc. 20th Fault Tolerant Computing Symp.* pp. 374–381, June 1990.
[2] S. Kundu, S. M. Reddy, and N. K. Jha, "On the design of robust multiple fault testable CMOS combinational logic circuits," in *Proc. Int. Conf. Computer-Aided Design*, pp. 240–243, Nov. 1988.
[3] K. Roy, K. De, J. A. Abraham, and S. Lusky, "Synthesis of delay fault testable combinational logic," in *Proc. Int. Conf. Computer-Aided Design*, pp. 418–421, Nov. 1989.
[4] S. Devadas and K. Keutzer, "Synthesis and optimization procedures for robustly delay-fault testable logic circuits," in *Proc. 27th Design Automation Conf.*, pp. 221–227, June 1990.
[5] S. Devadas and K. Keutzer, "Design of integrated circuits fully testable for delay faults and multifaults," in *Proc. Int. Test Conf.*, pp. 284–293, Oct. 1990.
[6] G. L. Smith, "A model for delay faults based on paths," in *Proc. Int. Test Conf.*, pp. 342–349, Sept. 1985.
[7] S. Devadas and K. Keutzer, "Necessary and sufficient conditions for robust delay-fault testability of logic circuits," in *Sixth MIT Conf. on Advanced Research on VLSI*, pp. 221–238, Apr. 1990.
[8] Y. K. Malaiya and R. Narayanswamy, "Testing for timing failures in synchronous sequential integrated circuits," in *Proc. Int. Test Conf.*, pp. 560–571, Oct. 1983.
[9] S. Devadas, "Delay test generation for synchronous sequential circuits," in *Proc. Int. Test Conf.*, pp. 144–152, Aug. 1989.
[10] B. Park and P. R. Menon, "Design of scan-testable CMOS sequential circuits," in *Proc. Int. Test Conf.*, pp. 369–376, Sept. 1990.
[11] W. Mao and M. Ciletti, "Arrangement of latches in scan-path design to improve delay fault coverage," in *Proc. of Int. Test Conf.*, pp. 387–393, Sept. 1990.
[12] J. A. Waicukauski, E. Lindbloom, B. Rosen, and V. Iyengar, "Transition fault simulation," *IEEE Design and Test*, pp. 32–38, Apr. 1987.
[13] C. J. Lin and S. M. Reddy, "On delay fault testing in logic circuits," *IEEE Trans. Computer-Aided Design*, vol. CAD-6, pp. 694–703, Sept. 1987.
[14] F. J. Hill and G. R. Peterson, *Introduction to Switching Theory and Logical Design*. New York: Wiley, 1981.
[15] G. D. Hachtel, R. M. Jacoby, K. Keutzer, and C. R. Morrison, "On the relationship between area optimization and multifault testability of multilevel logic," in *Int. Conf. Computer-Aided Design*, pp. 422–425, Nov. 1989. Also appeared as "On properties of algebraic transformations and the synthesis of multifault-irredundant circuits," *IEEE Trans. Computer-Aided Design*, pp. 313–321, Mar. 1992.
[16] K-T. Cheng, and V. D. Agrawal, "Design of sequential machines for efficient test generation," in *Proc. Int. Conf. Computer-Aided Design*, pp. 358–361, Nov. 1989.

[17] G. R. Putzolu and J. P. Roth, "A heuristic algorithm for the testing of asynchronous circuits," *IEEE Trans. Computers*, vol. C-20, pp. 639–647, June 1971.
[18] C. Lin, S. M. Reddy, and S. Patil, "An automatic test pattern generator for the detection of path delay faults," in *Proc. Int. Conf. Computer-Aided Design*, pp. 284–287, 1987.
[19] A. Ghosh, S. Devadas, and A. R. Newton, "Test generation for highly sequential circuits," in *Proc. Int. Conf. Computer-Aided Design*, pp. 362–365, Nov. 1989.
[20] A. V. Aho, J. E. Hopcroft, and J. D. Ullman. *The Design and Analysis of Computer Algorithms*. Englewood Cliffs, NJ: Prentice-Hall, 1974.
[21] W.-N. Li, S. M. Reddy, and S. Sahni, "On path selection in combinational logic circuits," *IEEE Trans. Computer-Aided Design*, vol. 8, pp. 56–63, Jan. 1989.
[22] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-completeness*. San Francisco, CA: Freeman, 1979.

Kwang-Ting (Tim) Cheng (S'88–M'88), for a photograph and a biography, please see page 24 of the January 1993 issue of this TRANSACTIONS.

Srinivas Devadas (S'87–M'88), for a photograph and a biography, please see page 598 of the May 1993 issue of this TRANSACTIONS.

Kurt Keutzer (S'82–M'84) received the B.S. degree in mathematics from the Maharishi International University in 1978 and the M.S. and Ph.D. degrees in computer science from Indiana University in 1981 and 1984, respectively.

In 1984 he joined AT&T Bell Laboratories, where he worked to apply various computer science disciplines to practical problems in computer-aided design. In 1991 he joined Synopsys Inc., Mountain View, CA, where he continues his research. His efforts have led to two Design Automation Conference best paper awards, a distinguished paper citation from the International Conference on Computer-Aided Design, and the inclusion of a paper in the anthology *Twenty-Five Years of Electronic Design Automation*.

Dr. Keutzer serves on the editorial boards of the IEEE TRANSACTIONS ON COMPUTER-AIDED DESIGN, the new journal *Formal Methods in System Design, and Bowker's CADCAM Abstracts*, a reference publication. He is a member of the executive committees of the International Conference on Computer-Aided Design and the International Conference on Computer Design. He also serves on several other technical program committees.