# A DYNAMIC PROGRAMMING APPROACH TO THE TEST POINT INSERTION PROBLEM

*Balakrishnan Krishnamurthy*

Computer Research Laboratory
Tektronix Laboratories
Beaverton, OR 97077

*ABSTRACT*

The *test point insertion problem* is that of selecting $t$ nodes in a combinational network as candidates for inserting observable test points, so as to minimize the number of test vectors needed to detect all single stuck-at faults in the network. In this paper we describe a dynamic programming approach to selecting the test points and provide an algorithm that inserts the test points optimally for fanout-free networks. We further extend this algorithm to general combinational networks with reconvergent fanout. We also analyze the time complexity of the algorithm and show that it runs in $O(n \cdot t)$ time, where $n$ is the size of the network and $t$ is the number of test points to be inserted.

As a side result we show that we can compute minimal test sets for a restricted class of networks that includes fanout. This extends previous results which were limited to fanout-free networks.

## 1. Introduction

The *test point insertion problem* is that of determining a set of $t$ nodes in a network, as candidates for inserting observable test points, so as to minimize the number of test vectors needed to test the resulting network for all single stuck-at faults. In this paper we will only consider combinational networks. Even for combinational networks the task of determining the optimal nodes for test point insertion becomes intractable (as we shall explain later). In fact, most such optimization problems are computationally intractable for general networks. Our approach in this paper, as has been that of [1, 3, 7] is to solve such optimization problem optimally for fanout-free networks and then lift the solution to the general class of networks.

In this paper we will demonstrate an efficient dynamic programming solution to the test point insertion problem that provides provably optimal results for fanout-free networks. Furthermore, the solution can be extended to networks with fanout as well as reconvergence, in which case the optimality of the solution is no more guaranteed. We point out that the complexity of the algorithm (for both fanout-free as well as general networks) is proportional to the product of the size of the network and $t$, the number of test points to be inserted.

It should be pointed out that unlike in many other testing related problems the test point insertion problem is both unsolved and important for fanout-free networks. Not only is the general problem intractable, but it is even more difficult to evaluate the goodness of a specific choice of test points in a specific non-fanout-free network! There is no way of telling if the suggested test points minimize the required number of tests, or even what the required number of tests is in a general network! The best we can do is to ensure that our solution to the general network does as well as possible within fanout-free subcomponents. Thus, the only viable solution in a pragmatic sense is to devise an optimal solution for the fanout-free case and lift it to the fanout case. This is precisely what we do in this paper.

The test point insertion problem was originally proposed in 1973 by Hayes and Friedman [7]. They proposed a labeling scheme and modeled the test point insertion problem as an integer programming problem, and they suggested some heuristics for determining the integer programming solution. It is now known that solving an integer programming problem is just as hard as the test point insertion problem. Even in the fanout-free case, the resulting integer programming problem does not seem any easier. Further, we mention in this paper a subtle point that the authors of [7] overlooked and we point out that this significantly adds to the complexity of the problem.

One of the major contributions of [7] is an explicit algorithm for computing the minimum number of test vectors needed to detect all single stuck-at faults in a fanout-free combinational network. Their proposed labeling scheme could be shown to be exact for fanout-free networks. In fact, this labeling scheme can be used to generate a minimal test sets itself (see [5]). If minimal test sets can be computed for fanout-free networks, why is it difficult to determine optimal nodes for test point insertion in a fanout-free network? The main reason is that as soon as we insert a test point the network no longer remains fanout-free. In fact, for this very reason, it is relatively easy to insert one test point into a fanout-free network. We will provide a simple algorithm to do this.

We pointed out in the last paragraph that even a fanout-free network fails to be fanout-free as soon as we insert test points into the network. However, even though there is fanout in the resulting network, it is of a special type. In Section 4 we show that the minimal test set size for such networks can be computed efficiently. We require this result for our subsequent proofs of optimality of the algorithms suggested in this paper.

We should mention another major difficulty in the test point insertion problem. Consider the network of Figure 1. This network requires 10 test vectors to detect all single stuck-at faults. If we were to insert one test point the best place to insert it would be at the output of gate $A$. That will reduce the test set size to 9. On the other hand, if we were to insert two test points the optimal places to insert them would be at the outputs of gates $B$ and $C$. The test set size for this network will be 6. We realize from this example that we cannot insert test points one at a time, preserving the previously inserted test points. We need to take into account the total number of test points to be inserted before we insert the very first test point. That makes the task all the more difficult.

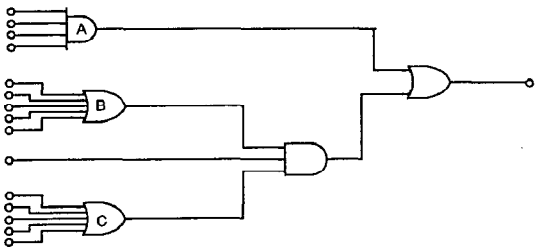


Figure 1

Let us make a few comments about the theoretical and practical implications of the results of this paper. Theoretically, the test point insertion problem, even for fanout-free networks, has remained open for over a decade. In addition, the problem of computing the minimum test set size for multiple output non-reconvergent networks is an open theoretical problem. We make some progress in this paper towards a solution to that problem.

On the practical side, test set size has been proposed as a general testability analysis mechanism, and recent results have indicated its usefulness for analyzing testability and in test generation ([3, 9]). In light of this, the test point insertion problem becomes of utmost importance in enhancing testability. Of course, we do not claim that solving the test point insertion problem optimally for fanout-free networks is of immediate practical significance. But, the ability to lift the solution to arbitrary networks does render it of practical value. We offer in this paper a dynamic programming approach to the test point insertion problem that provides optimal results for fanout-free networks and lifts to general networks without any degradation in the time complexity of the algorithm.

In Section 2 we review some of the background material pertinent to the test point insertion problem. In Section 3 we present a simple solution to the single test point insertion problem and show that it is optimal for fanout-free networks. In Section 4 we show that the minimum test set size for a restricted class of networks with fanout can be computed efficiently. In Section 5 we present a dynamic programming solution to the general test point insertion problem and show that it is optimal for fanout-free networks. Finally, in Section 6 we conclude with a few comments on the complexity of these algorithms and how one can lift this algorithm to an arbitrary combinational network with reconvergent fanout.

## 2. Background

In this section we will first point out certain established complexity results that puts this work in proper perspective. We will then proceed to review some of the existing techniques for test set size enumeration that we will use in this paper.

It is well known that determining whether a specific stuck-at fault in a given network is detectable or not is a difficult problem. More precisely, it is NP-complete ([6]). Consequently, determining whether a given network is irredundant or not is also NP-complete. More surprisingly, the problem of determining the size of a minimum test set (that detects all single stuck-at faults) for a given irredundant network is at least as hard. More precisely, it is NP-hard ([8]). All of these complexity results use the fact that the Boolean Satisfiability problem (see [6]) is embedded in all test generation problems particularly due to reconvergent fanout. In fact, for fanout-free networks we can not only compute the minimal test set size but also compute all possible minimal test sets for the network (see [5]). This raises a natural question about networks that contain fanout but no reconvergence, i.e., multiple output non-reconvergent networks. Can a minimal test set be efficiently computed for such networks. This remains an open problem.

For the purposes of this paper we need to be familiar with test set size enumeration for fanout-free networks. That is, given a fanout-free network how can we compute the minimum number of test vectors needed to detect all single stuck-at faults in the network? A labeling method was suggested in [7] and a generalization of this approach has been used in [3] to develop an approximation technique for estimating the test set size for arbitrary networks. Our exposition, terminology and notation closely follows the test counting technique of [3]. Similar techniques have also been developed in [1, 2] under a slightly different context.

To describe the test counting technique, let us first define the concepts of *logic value*, *sensitivity value* and *test value*. Given a test vector $t$ for a circuit $C$ we can perform logic simulation and determine the Boolean value of 1 or 0 on each lead $A$ in $C$. This value is called the *logic value* on $A$. Further, we could perform fault simulation and determine, for each lead $A$ in $C$, whether or not $t$ detects the "appropriate" stuck-at fault on $A$. We denote the result using the symbols "+" or "-" to denote that the corresponding stuck-at fault at that lead is detected or undetected, respectively, and we call these values

the *sensitivity values*. Thus, a logic value of 1 and a sensitivity value of + on a lead $A$ in $C$ indicates that the test $t$ yields a 1 on $A$ and that if $A$ were stuck-at 0, the value at some primary output of $C$ would change. The *test value* on a lead $A$ is one of the following four possible values: $0^+, 1^+, 0^-, 1^-$; and is an obvious combined representation of the lead's logic and sensitivity values.

Having described the test values imposed by a test $t$ on each lead $A$ of $C$, let us turn to the definition of test counts. Given a set of test vectors $T$ for $C$, we associate with each lead $A$ in $C$, four *test counts* denoted by $a_0^+$, $a_0^-$, $a_1^+$, and $a_1^-$. Given below are the definitions for these test counts.

$a_0^+$ = number of test vectors in $T$ that yield a test value of $0^+$ on $A$;

$a_0^-$ = number of test vectors in $T$ that yield a test value of $0^-$ on $A$;

$a_1^+$ = number of test vectors in $T$ that yield a test value of $1^+$ on $A$;

$a_1^-$ = number of test vectors in $T$ that yield a test value of $1^-$ on $A$.

The test counting technique is a method for computing lower bounds for the test counts at each lead of $C$ without knowing $T$, and, of course, without performing logic or fault simulation. Instead, the technique does make use of the set of faults that $T$ detects. Thus, given $C$ and a set of faults $F$, the test counting technique provides a lower bound for the test counts, and these lower bounds are applicable to any test set $T$ that detects all the faults in $F$. In particular, the lower bounds are applicable to the minimal test set. Thus the claim made in [3], that test counting provides a lower bound for the minimum number of tests needed to detect all the faults in $C$.

The test counting technique proceeds by satisfying a collection of inequalities, called *constraints*, relating the various test counts. These inequalities come in two varieties. The first comes from the fault set, one for each fault in $F$. For example, the inequality corresponding to a stuck-at-1 fault on a lead $A$ is:

$$a_0^+ \geq 1$$

The second class of inequalities comes from the set of constraints associated with each type of gate. Let us briefly indicate the nature of one such constraint. Consider a three input *AND* gate with input lead $A$, $B$ and $C$ and output lead $D$. Clearly, whenever a test vector yields a test value of $0^+$ on lead $A$ the test value on lead $C$ must be $1^-$. Similarly, whenever a test vector yields a test value of $0^+$ on lead $B$ the test value on lead $C$ must be $1^-$. Further, whenever a test vector yields a test value of $1^-$ on lead $d$ the test value on lead $C$ must be $1^-$. Finally, the above three situations are mutually exclusive, i.e., no two of those three situations could occur in the same test vector. From this argument we can conclude:

$$c_1^- \geq a_0^+ + b_0^+ + d_1^-$$

In this paper we will neither require the general test counting technique not the details of the large number of constraints enumerated in [4]. Instead we will, at this point, describe an adequate procedure for computing the required test set size of fanout-free networks. For this purpose we will require only the sensitive test counts. We will also assume that the network is made up of two input *AND* and *OR* gates only. A generalization of this algorithm to larger number of inputs to the gates and to inverters is rather obvious. The algorithm is given below:

*Algorithm 1*:

Input: A fanout-free network.

Output: The number of test vectors required to test the network for all single stuck-at faults, and a set of test counts for each lead in the network.

Procedure:

1. Associate two variables $a_0^+$ and $a_1^+$ with each lead $A$ in the network.
2. For each lead $A$ initialize $a_0^+$ and $a_1^+$ to 1 to indicate that the stuck-at-1 fault and the stuck-at-0 fault on that lead is to be detected.
3. Starting from the inputs and working towards the outputs consider each gate one at a time. Let the input leads of the gate be $A$ and $B$ and the output lead of the gate be $C$. Depending on the type of gate do:

   *AND*: Set $c_0^+ := a_0^+ + b_0^+$;
   $c_1^+ := MAX(a_1^+, b_1^+)$;

   *OR*: Set $c_1^+ := a_1^+ + b_1^+$;
   $c_0^+ := MAX(a_0^+, b_0^+)$;
4. Compute the largest value of $a_0^+ + a_1^+$ over all leads $A$ in the network. This is the required number of test vectors.

Let us illustrate this algorithm using an example. We indicate the values of the two test counts $a_0^+$ and $a_1^+$ associated with a lead $A$ by a vector of two elements in their respective order. Consider the network of Figure 2A. Applying the above algorithm on the network yields the test counts shown in that figure. For example, the test counts for lead $A$ are: $a_0^+$=1 and $a_1^+$=3. This indicates that in any complete test set for this network there will be three vectors that will detect a stuck-at-0 fault on lead $A$.

The following theorem is implicit in the work of [7] and is explicitly stated in [3]:

*Theorem 1*: The above algorithm provides a lower bound on the number of test vectors needed to detect a specified set of stuck-at faults in a combinational network. Furthermore, for fanout-free networks the derived lower bound is optimal.

As we mentioned in Section 1 we will rely on this algorithm to compute the optimal test set size for fanout-free networks. However, as we had mentioned, as soon as we introduce one test point the network is no longer fanout-free. In Section 4 we show how the optimal test set size for fanout-free

networks with added test points can be computed. But first let us use the above described algorithm to place a *single* test point optimally.

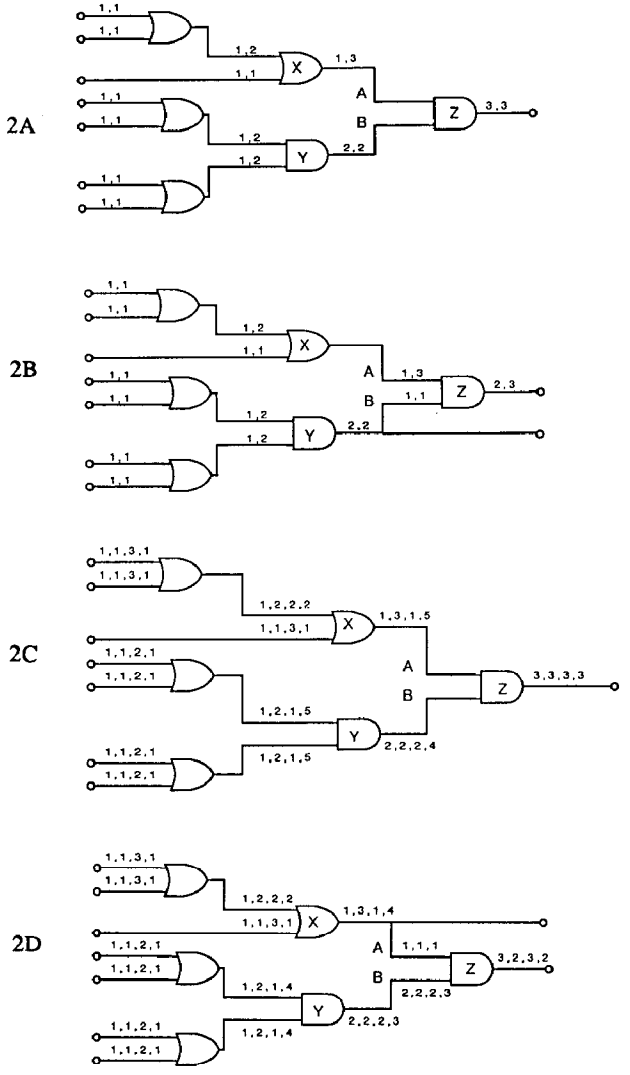


Figure 2A, 2B, 2C, 2D

### 3. Single Test Point Placement

Let us return to the network of Figure 2A. Suppose we wished to reduce the number of test vectors needed (in this case, 6) by inserting an observable test point somewhere in the network. Where would we insert it? Clearly, inserting it at the primary input leads or the primary output lead does not affect the number of test vectors needed. How about at the output of gate $Y$? The resulting network is shown in Figure 2B. Notice that this new test point can be used to observe any of the faults detected in the subnetwork commanded by gate $Y$. Thus, in Algorithm 1 we can re-initialize the sensitive test counts on lead $B$ to be 1 and continue to apply the algorithm. The resulting test counts are given in Figure 2B. This, of course, leads us to conclude that we need 5 test vectors for this modified network. Unfortunately, this is merely a lower bound and the lower bound is not optimal.

The reason is this. The output of gate $Y$ has to assume a test value of $0^+$ two times. In addition, lead $B$, (and hence, the output of gate $Y$), has to assume a logical value of 1 whenever lead $A$ assumes a sensitive test value. Since we have already concluded that lead $A$ has to be a $0^+$ once and a $1^+$ three times, it follows that lead $B$ (and hence, the output of gate $Y$) must be a logical 1 at least four times. This, in conjunction with the requirement that it be a $0^+$ twice, forces the need for at least 6 test vectors!

To account for this we introduce two new test counts, the *logic counts*, $a_0$ and $a_1$ associated with a lead $A$. They are defined as:

$a_0$ = number of test vectors in $T$ that yield a test value of 0 on $A$;

$a_1$ = number of test vectors in $T$ that yield a test value of 1 on $A$;

We now enhance Algorithm 1 to compute the logic counts.

*Algorithm 2*:

Input: A fanout-free network.

Output: The number of test vectors required to test the network for all single stuck-at faults, and a set of test counts for each lead in the network.

Procedure:

1. Perform Algorithm 1.
2. Set the logic counts of the primary output lead to be equal to the corresponding sensitive test counts.
3. Starting from the primary output and working towards the primary inputs consider each gate one at a time. Let the input leads of the gate in consideration be $A$ and $B$ and the output lead of the gate be $C$. Depending on the type of gate do:

   *AND*: Set $a_1 := b_0^+ + c_1$;
   $b_1 := a_0^+ + c_1$;
   $a_0 := a_0^+$;
   $b_0 := b_0^+$.

   *OR*: Set $a_0 := b_1^+ + c_0$;
   $b_0 := a_1^+ + c_0$.
   $a_1 := a_1^+$;
   $b_1 := b_1^+$.

4. Compute the largest value of $a_0 + a_1$ over all leads $A$ in the network. This is the required number of test vectors.

The following theorem is implicit in the work of [3] but is neither stated nor proved in that paper. We too will omit a formal proof in the interest of brevity, but will state the result:

*Theorem 2*: Let $A$ be any lead in a fanout-free network and let $a_0^+$, $a_1^+$, $a_0$ and $a_1$ be the values of the four test counts obtained using Algorithm 2. Then these values are optimal in the following sense:

1. For any complete (detects all single stuck-at faults) test set $T$ there are at least
   (i) $a_0^+$ vectors in $T$ that yield a value of $0^+$ on $A$;
   (ii) $a_1^+$ vectors in $T$ that yield a value of $1^+$ on $A$;
   (iii) $a_0$ vectors in $T$ that yield a value of 0 on $A$;
   (iv) $a_1$ vectors in $T$ that yield a value of 1 on $A$.
2. There exists a complete test set $T$ in which there are at most
   (i) $a_0^+$ vectors in $T$ that yield a value of $0^+$ on $A$;
   (ii) $a_1^+$ vectors in $T$ that yield a value of $1^+$ on $A$;
   (iii) $a_0$ vectors in $T$ that yield a value of 0 on $A$;
   (iv) $a_1$ vectors in $T$ that yield a value of 1 on $A$.

The above theorem states that the values of the test counts computed by Algorithm 2 are the best possible values. Let us now apply Algorithm 2 to the network of Figure 2A. The resulting test counts are shown in Figure 2C. The four test counts, $a_0^+$, $a_1^+$, $a_0$ and $a_1$, are shown as four element vectors in their respective order.

Let us now turn to the question of what happens when we place a test point, say at the output of gate $X$, as shown in Figure 2D. Recall from Figure 2A that since the sensitive test counts of lead $A$ are (2,2) there are four faults in the subnetwork commanded by that lead, whose tests have to pass through lead B. But now that we have a test point at the output of gate $X$ these tests do not have to be sensitized through gate $Z$ any more. We might as well use the test point to observe these tests. However, we still have to test for the stuck-at faults on the input lead $A$ of gate $Z$. To do this we set the sensitive test count values of lead $A$ to be (1,1) and proceed with Algorithm 1. On the backward pass (Step 3) of Algorithm 2 we merge the logic counts on the input lead $A$ of gate $Z$ with the output lead of gate $X$. The merging process amounts to taking the maximum of the respective logic counts. The resulting test counts are shown in Figure 2D. Notice that we now need only 5 test vectors for this network.

Of course, this only allows us to compute the required number of test vectors once we have chosen a test point. How can we efficiently choose the best place to insert a test point? Trying each possible lead in the network is surely not an efficient method. We give below an algorithm that determines the optimal place to insert a single test point in a fanout-free network. We need some additional notation. With each lead $A$ we associate two *slack values*, $\Delta_{a_0^+}$ and $\Delta_{a_1^+}$. The slack value $\Delta_{a_0^+}$ is the amount by which the sensitive test counts on the output will decrease if we insert a test point at lead $A$ and thereby decrease the sensitive test counts on $A$ to (1,1). The slack value $\Delta_{a_1^+}$ is similarly defined. Let us also define $\eta_a$ to be the number of test vectors needed if a test point were to be inserted at lead $A$. With these definitions, we are now ready to present the single test point insertion algorithm.

*Algorithm 3*:

Input: A fanout-free network.

Output: An optimal lead $A$ for inserting a single test point, together with the reduced number of test vectors needed for the new network.

Procedure:

1. Perform Algorithm 2. Let $\eta$ be the resulting number of test vectors needed for this network.
2. Set the logic counts of the primary output lead to be the corresponding sensitive test counts. Set the slack values, $\Delta_{a_0^+}$ and $\Delta_{a_1^+}$, for the output lead ($A$) to be equal to the sensitive test counts, $a_0^+$ and $a_1^+$, respectively. Set $\eta_a$ for the output lead to be $\eta$.
3. Starting from the primary output and working towards the primary inputs consider each gate one at a time. Let the input leads of the gate in consideration be $A$ and $B$ and the output lead of the gate be $C$. Depending on the type of gate do:

   *AND*: Set
   $\Delta_{a_0^+} := MIN(\Delta_{c_0^+}, a_0^+ - 1)$;
   $\Delta_{b_0^+} := MIN(\Delta_{c_0^+}, b_0^+ - 1)$;
   $\Delta_{a_1^+} := MIN(\Delta_{c_1^+}, MAX(0, a_1^+ - b_1^+))$;
   $\Delta_{b_1^+} := MIN(\Delta_{c_1^+}, MAX(0, b_1^+ - a_1^+))$;
   $\eta_a := MAX(\eta - \Delta_{a_0^+} - \Delta_{a_1^+}, a_0^+ + MAX(a_1^+, a_1 - \Delta_{a_1^+}))$
   $\eta_b := MAX(\eta - \Delta_{b_0^+} - \Delta_{b_1^+}, b_0^+ + MAX(b_1^+, b_1 - \Delta_{b_1^+}))$

   *OR*: Set
   $\Delta_{a_1^+} := MIN(\Delta_{c_1^+}, a_1^+ - 1)$;
   $\Delta_{b_1^+} := MIN(\Delta_{c_1^+}, b_1^+ - 1)$;
   $\Delta_{a_0^+} := MIN(\Delta_{c_0^+}, MAX(0, a_0^+ - b_0^+))$;
   $\Delta_{b_0^+} := MIN(\Delta_{c_1^+}, MAX(0, b_1^+ - a_1^+))$;
   $\eta_a := MAX(\eta - \Delta_{a_0^+} - \Delta_{a_1^+}, a_1^+ + MAX(a_0^+, a_0 - \Delta_{a_0^+}))$;
   $\eta_b := MAX(\eta - \Delta_{b_0^+} - \Delta_{b_1^+}, b_1^+ + MAX(b_0^+, b_0 - \Delta_{b_0^+}))$.
4. Choose the lead $A$ for which $\eta_a$ is a minimum as the optimal lead for inserting a test point. The corresponding $\eta_a$ is the required number of test vectors for the resulting network.

*Theorem 3*: Algorithm 3 finds an optimal lead for a single test point insertion in a fanout-free network.

*Sketch of Proof*: Prior to describing Algorithm 3 we showed how we might compute the number of test vectors needed in a fanout-free network with a single test point inserted. The proof of this theorem requires that we show that the computed value of $\eta_a$, for each lead $A$, by Algorithm 3 is indeed correct. We do this by induction on the number of gates between a given lead and the primary output. The induction step, of course, is equivalent to establishing the validity of the updating equations given in Step 3 of the algorithm. □

In this section we have shown that a single test point can be inserted efficiently. Recall the comments we made in Section 1 about the difficulties in inserting more than one test point. Not only is it inadequate to insert test points one at a time, but as soon as we insert the very first test point the network no longer is fanout-free. In fact, we cannot rely on Algorithm 2 to compute the required number of test vectors for such a network. In the next section we present an algorithm for comput-

ing the required number of test vectors for a fanout-free network with an arbitrary number of test points inserted.

## 4. Test Set Size for Fanout-Free Networks with Multiple Test Points

Consider the network of Figure 3A and the associated sensitive test counts obtained using Algorithm 1. By Theorem 1 the minimum test set size for this network is exactly 6. Now let us place two test points at the output of gates $X$ and $Y$. The resulting network is shown in Figure 3B. If we now apply Algorithm 2 to this network, we get the sensitive and the logic test counts shown in Figure 3B. These test counts indicate that we need at least 4 test vectors to test this network. We would like to know if 4 test vectors are sufficient to test this network, i.e., does Algorithm 2 provide the exact number of test vectors needed for such a network?

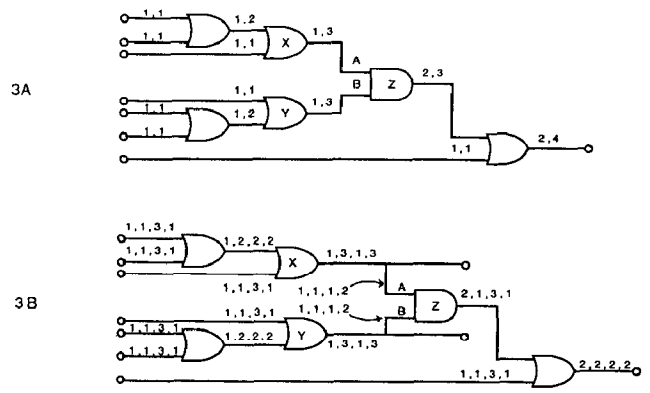


Figure 3A, 3B

Observe that even thought the logic counts on leads $A$ and $B$ indicate that they each need to be a 1 twice, they really must assume that value three times since the outputs of gates $X$ and $Y$ which feed into them must be a 1 three times. The test counts shown in Figure 3B also indicate that the output lead of gate $Z$ must be a 0 three times. Now ignoring the rest of the network and considering only gate $Z$, we can easily convince ourselves that this, or any two-input, *AND* gate cannot assume a value of 1 on each of its two inputs three times and a value of 0 on its output three times using only four sets of inputs! No matter how we design the inputs to this gate, we will require at least 5 sets of inputs. Thus, the network of Figure 3B requires at least 5 test vectors.

This is the subtle point that seems to have been overlooked in the labeling scheme of [7]. In [3] the authors have suggested the use of certain additional constraints that they call *oddball constraints* to account for this. While the oddball constraints take care of the local problem it does not seem to address the propagated effects of the problem. Let us explain this using the *AND* gate example. Once again let us consider an individual *AND* gate in isolation, and let us assume that for reasons uncovered by the test counting algorithms of this paper we have concluded that each of the two inputs of this *AND* gate must be a 1 at least five times and the output has to be a 0 at least five times. Once again, it is easy to convince ourselves that at least eight sets of inputs to the *AND* gate are needed to get the desired test counts. It has been shown in [3] that in general, if we require a 1 on each of the two inputs at least $a_1$ and $b_1$ times, respectively, and a 0 on the output at least $c_0$ times, then it will require at least $\left\lfloor \frac{a_1+b_1+c_0}{2} \right\rfloor$ pairs of inputs.

Let us pursue the case with $a_1 = b_1 = 5$ and $c_0 = 5$, for which we concluded that we require at least eight inputs. Figure 4A suggests two sets of eight input pairs that achieve this. (Recall that the test counts are only a lower bound on the actual number of times the lead has to take on a certain value.) The reader can quickly verify that no matter how we design the inputs to the *AND* gate, if we achieve the necessary test counts in eight input pairs then the output must be a 1 at least two times. On the other hand, if we allow nine input pairs then the output must be a 1 at least once. For example, Figure 4B achieves this.

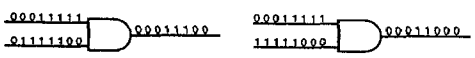

Figure 4A

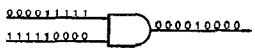

Figure 4B

In summary, given the number of times each input lead of an *AND* gate must assume each of the two logical values as well as the number of input pairs that we intend to use, we can conclude exact lower bounds for the number of times the output will assume each of the two logical values. This is given in the following Lemma.

*Lemma 1*: Let $A$ and $B$ be the two input leads of a two-input *AND* gate and $C$ be the output lead. Let $a_0$, $a_1$, $b_0$ and $b_1$ be the corresponding counts of the number of times that we require each of the input leads to assume each of the logical values. Then we will require at least $\eta$ input pairs, where $\eta \geq MAX(a_0+a_1, b_0+b_1)$. Further, if we use $n \geq \eta$ input pairs to achieve the desired counts then the output lead, $C$, must be a 0 at least $c_0$ times and a 1 at least $c_1$ times, where

$c_0 = MAX(a_0, b_0)$;

$c_1 = MAX(0, a_1+b_1-n)$.

Similar claims in which the roles of the two logical values are interchanged are true for the *OR* gate.

*Proof*: Easy. □

Notice that in the above Lemma the only values of $n$ that are interesting are those that lie between $\eta$ and $a_1+b_1$. We call this the *active range*. We are now ready to present an algo-

rithm that will compute the minimum number of test vectors needed for a fanout-free network with multiple test points inserted. The crux of the algorithm is in the application of a generalization of the previous Lemma. To explain this generalization we require additional notation. So we first develop the requisite notation and state a theorem that captures the generalization of the above Lemma. We will then state the algorithm.

With each lead, $A$, in a network we will associate the two sensitive test counts $a_0^*$ and $a_1^*$ as we have in the previous sections. We will also associate an active range, which is a closed interval on the positive integers, $[\eta_L,\eta_H]$, that represents the range of possible number of test vectors that one has to consider. For each value of $n$ in the active range we will associate two logic counts, $a_0^n$ and $a_1^n$, with that lead. These logic counts represent the minimum number of times that that lead must assume the specific logical value, given that $n$ test vectors are going to be used. $\eta_L$ is, of course, a lower bound on the number of test vectors required. For values of $n$ beyond $\eta_H$ the corresponding logic counts are the same as the logic counts for $\eta_H$.

*Theorem 4*: Let $A$ and $B$ be the input leads of a two-input *AND* gate and $C$ be the output lead of that gate. Let the associated active ranges and test counts for the two input leads be denoted by the notation described above. Then the active ranges and the test counts for the output lead are given by:

$$c_0^* = a_0^* + b_0^*$$
$$c_1^* = MAX(a_1^*, b_1^*)$$
$$\eta_L = MAX(\eta_L \text{ of } A, \eta_L \text{ of } B, c_0^* + c_1^*)$$
$$\eta_H = MAX(\eta_L, \underset{n}{MAX}(a_1^n + b_1^n))$$
$$c_0^n = MAX(a_0^n, b_0^n)$$
$$c_1^n = MAX(0, a_1^n + b_1^n - n)$$

Similar claims in which the roles of the two logical values are interchanged are true for the *OR* gate.

*Proof*: The proof of each of the above claim follows the type of reasoning used in the above examples. □

*Algorithm 4*:

Input: A fanout-free network with an arbitrary number of test points inserted.

Output: The number of test vectors needed to detect all single stuck-at faults in the network.

Procedure:

1. For each of the primary input leads assign:
   (i) the sensitive test counts to be (1,1);
   (ii) the active range to be [2,2];
   (iii) the logic counts for the one and only element in the active range to be (1,1).
2. Starting from the primary inputs and working towards the primary output (not the test outputs) consider each gate one at a time.
   (i) Apply the equations of Theorem 4 based on the gate type to compute the active range and the test counts of the output lead.
   (ii) If there is a test point at the output of the gate then (assume that all the sensitive test counts are accounted for by the test point):
      (a) Set the sensitive test count of the branch leading into the next gate to be (1,1).
      (b) Set its active range and logic counts to be the same as those of the stem.
3. The number of test vectors needed is the $\eta_L$ of the primary output lead.

*Corollary 1*: Algorithm 4 computes the optimal test set size for fanout-free networks with an arbitrary number of test points.

*Proof*: By induction on the depth of the network. The basis step consists of verifying Step 1 of the algorithm. The induction step is equivalent to Theorem 4. □

Now that we can compute the number of test vectors needed for a fanout-free network with arbitrary number of test points, can we determine where to put a specified number of test points into a given fanout-free network so that the resulting test set size computed by Algorithm 4 is minimized? This is the topic of the next section.

## 5. Multiple Test Point Insertion

Notice that in Algorithm 4 we wanted to propagate the logic counts on the inputs of a gate as logic counts on the output of the gate. But we could not make adequate claims on the output logic counts without knowing the number of test vectors that were going to be needed — the very quantity that we were computing in that algorithm! We resolved the deadlock by computing the output logic counts for each possible value for the number of test vectors that might be needed. This idea of computing a multiple set of answers to a problem and postponing until the end th decision of which answer to choose is the dynamic programming flavor in that algorithm.

In this section we consider the problem of determining the optimal leads to place $t$ test points in a fanout-free network and use a dynamic programming approach much like that of Algorithm 4. Recall a comment made in Section 1 that in placing $t$ test points we cannot place them one at a time, incrementally. Our next algorithm places the $t$ test points in one sweep.

The overall approach is the following. Let us start with a fanout-free network and a number, $t$, of test points to be inserted. Consider the last gate in the network. It has two input leads, say $A$ and $B$, each of which commands a fanout-free sub-network. What we are going to do is to determine how many of the $t$ test points must lie within the $A$ sub-network, how many of them must lie within the $B$ sub-network and whether either of the leads $A$ or $B$ should contain a test point itself. Of course, our commitments must all add up to at most $t$. In order to partition $t$ between the two sub-networks optimally, we compute for each of the two sub-networks the benefits for that sub-network of inserting $i$ test points, for each value of $i$ in the range $0 \le i \le t$. The benefits, of course are in terms of the resulting test counts on the leads $A$ and $B$ and in terms of the number of test vectors needed. Once we have this information for each of the two sub-networks and each value of

$i$, we can then determine how to partition $t$ between the two sub-networks most optimally.

Observe that once again we have used a dynamic programming approach to delay making the decision until the last moment. In fact, in order for this technique to proceed recursively within each of the sub-network, we must compute at each step the answers to the questions: What are the benefits of inserting $i$ test points, $0 \le i \le t$. Of course, a naive recursive implementation of this approach will result in an exponential algorithm. Instead, we suggest proceeding from the inputs to the output and calculating these benefits at the output of each gate.

So we must now associate with each lead $t+1$ copies of the collection of test counts and active range information that we had used in Algorithm 4. We then perform Algorithm 4 modified so that at each step we compute $t+1$ sets of data for the output lead, one for each i, $0 \le i \le t$. To compute the data on the output lead for a fixed $i$ we partition $i$ into four parts, $i_A$, $i_B$, $I_A$ and $I_B$. $i_A$ and $i_B$ are 0/1 valued and represent whether a test point is placed on the lead $A$ and $B$, respectively. $I_A$ and $I_B$ represent the number of test points allocated to the two sub-networks. Since $i_A+i_B+I_A+I_B=i$, it is easy to see that there are at most $4(i+1)$ possible ways of partitioning $i$. We compute the data for the output lead for each such partition of $i$ and choose that partition that produces the least number of test vectors needed.

Let us make one other observation before we present our last algorithm. Even though $t$ test points are to be inserted it is conceivable that the $t^{th}$ test point does not help in reducing the number of test vectors. To account for this throughout the algorithm we will associate a number, $t_a$, with each lead $A$, of the number of test points that can be fruitfully used by the network commanded by lead $A$.

*Algorithm 5*:

Input: A fanout-free network and a number, $t$, of test points to be inserted.

Output: A set of $t$ leads in the network where the test points should be inserted, so as to minimize the number of test vectors needed for testing the resulting network for all single stuck-at faults.

Procedure:

1. For each of the primary input leads assign:
   (i) Set $t_a$ to be 0.
   (ii) Initialize the test counting data indexed by $t_a=0$ using Step 1 of Algorithm 4.
2. Starting from the primary inputs and working towards the primary output consider each gate one at a time. Let the input leads to the gate be $A$ and $B$ and the output lead of the gate be $C$.
   (i) Set the value of $t_a$ for the output lead to be $MIN(t, t_a+t_b+2)$.
   (ii) For each value of $i$, $0 \le i \le t_c$ do:
      (a) Partition $i$ into four parts: $i_A$, $i_B$, $I_A$ and $I_B$, as prescribed in the preceding discussion. (There are at most $4(i+1)$) such partitions.
      (b) For each such partition compute the test counting data for the output lead $C$, indexed by $i$, using Steps 2.i and 2.ii of Algorithm 4.
      (c) Choose the partition that yields the smallest value of $\eta_L$ on $C$ and assign the test counting data of the output lead $C$ to be that obtained using this partition.
3. The test data for the output lead, indexed by $t$, will provide the number of test vectors needed in the best assignment of $t$ test points. A record of the choices made in Step 2.(ii).(a) will provide the exact test points selected.

We now state our final theorem, whose proof is immediate from the discussion preceding the above algorithm.

*Theorem 5*: Algorithm 5 yields optimal leads to insert $t$ test points in a fanout-free network.

## 6. Networks with Reconvergent Fanout

As we mentioned in the beginning of the paper, we will show how these algorithms can be lifted to the general case of networks with reconvergent fanout. Of course, we will not be able to make any claims on the optimality of the test points inserted in such networks. However, we will ensure that the complexity of the algorithms remains unchanged.

When working with networks with fanout, we must contend with multiple output networks. Further, unlike many other testing related problems we cannot solve the test point insertion problem independently for each of the cones of single output networks. We must solve the entire problem of inserting $t$ test points for the whole network.

So far we have indicated three subtle complications in inserting $t$ test points:

1. The need to insert all of the $t$ test points with global consideration, as opposed to one at a time (discussed in Section 1).
2. The need to consider oddball constraints (discussed in Section 4).
3. The need to consider multiple output networks.

A practically viable solution to the test point insertion problem could ignore some or all of these subtle complications as theoretical caveats. Indeed, while the two points above are quite appropriate when the goal is to obtain the optimal solution, they becomes less significant for general networks where any reasonable solution is all we can hope for. Thus, our approach to the general networks will consists of inserting one test point at a time ignoring the oddball effects but taking into account the multiple output network. This approach has the added advantage of being much simpler than the previous algorithms of this paper.

Unlike our previous algorithms in this paper we will not precede the next algorithm for inserting a test point within a general network with a corresponding theorem that establishes its optimality. The algorithm will consists of propagating two sets of counts from the inputs to the outputs of the network. Each set of counts will include the two sensitive counts and the two logic counts for a lead in the network. The two sets of counts will correspond to the two cases indicating whether or not a test point is inserted within the cone feeding that lead.

The propagation rules are quite simple. Through an *AND* gate the rules are similar to the ones described in Theorem 4 with the exception that we do not carry $\eta_L$ and $\eta_A$. Further, we only carry the logic counts for one value of $\eta$. The rules for the *OR* gate are analogous. However, we need to provide a new set of rules for a fanout node. These rules will be somewhat optimistic in an attempt to under estimate the number of tests needed.

The difficulty with a fanout node is that sensitive test counts entering the stem may exit through either branch, or both. So we cannot positively conclude that they must exit on any one of the fanout branches. As we said, we will be optimistic and ignore the sensitive test counts in propagating through the fanout. The interpretation here is that each branch assumes that the other branch will propagate the sensitive values. However, each branch will inherit the logic counts from the stem. Thus the sensitive test counts on the branches will be reset to (1,1) and the logic counts will be set to the corresponding logic counts of the stem.

Before we state the algorithm let us establish some notation. Since we will carry two sets of test counts with each lead, one corresponding to the case where a test point is not inserted within its cone, and one for the case where a test point is inserted, we will represent these two sets of test counts for a lead $A$ by $(a_0^+, a_1^+, a_0, a_1)$ and $(\hat{a}_0^+, \hat{a}_1^+, \hat{a}_0, \hat{a}_1)$. We will refer to the former set as the *normal test counts* and the latter set as the *used test counts*. We are now ready to state the algorithm.

*Algorithm 6*:

Input: A general combinational network within which a single test point is to be inserted.

Output: A lead in the network where the test point should be inserted, so as to minimize the number of test vectors needed for testing the resulting network for all single stuck-at faults.

Procedure:

1. For each of the primary input leads initialize two sets of test counts data, to be (1,1,1,1).
2. Starting from the primary inputs and working towards the primary output consider each gate (or fanout) one at a time.
   (i) If it is a gate (not a fanout) then:
      (a) propagate the normal test counts using the rules described above to compute the normal test counts of the output lead.
      (b) To compute the used test counts for the output lead of the gate perform the following computations and take that computation that yields the minimal number of tests required:
         (i) For each input lead use its used test count and the normal test counts of all other input leads and propagate using the rules suggested above.
         (ii) Assuming a test point at the output of this gate, assign the sensitive counts of the normal test counts computed for the output lead in Step 2.(i).(a) to be (1,1).

      In choosing one of the above alternatives break ties so as to minimize the total number of sensitive test counts.

   (ii) If it is a fanout node then set the normal and used sensitive test counts of all its branches to be (1,1) and the normal and used logic test counts of all its branches to be the corresponding values of the stem.
3. The location of the final choice for the test point can be obtained by tracing the decisions made at each gate.

We have illustrate the above algorithm in Figure 5. The reader is encouraged to apply Algorithm 6 to Figure 5 and verify the numbers. More importantly, he is encouraged to verify that by tracing the choices back the chosen point for inserting a test point is at the output of gate $A$. Before we conclude this section we must reiterate that Algorithm 6 should be applied $t$ times to insert $t$ test points in the network.

## 7. Conclusion

Let us first remark on the restriction on the gates that we have imposed throughout this paper, i.e., two input *AND* and *OR* gates only. This restriction is merely for the sake of clarity in our exposition. The reader will readily agree that Algorithms 1 through 4 can all be enhanced to accommodate other (fanout-free) gates as well as gates with an arbitrary number of inputs. The only step that might require an explanation is in Step 2.(ii).(a) of Algorithm 5, where a partition of $i$ into each of the two input leads is required. If there are more than 2 inputs we would be required to partition $i$ into an arbitrary number of parts —a task that can yield exponentially many choices. We avoid this by representing a many input gate by cascaded two input gates, with the proviso that test points cannot be inserted on the artificially created leads in such a cascading. This requires that in Step 2.(ii).(a) we sometimes will have to set $i_A$ and/or $i_B$ to be necessarily 0.

Throughout the paper we have not remarked on the time complexity of any of the algorithms. But, before we do that let us mention that in presenting the algorithm we have often separated iterated loops for the sake of clarity. In an actual implementation of these algorithms one should and can fold a

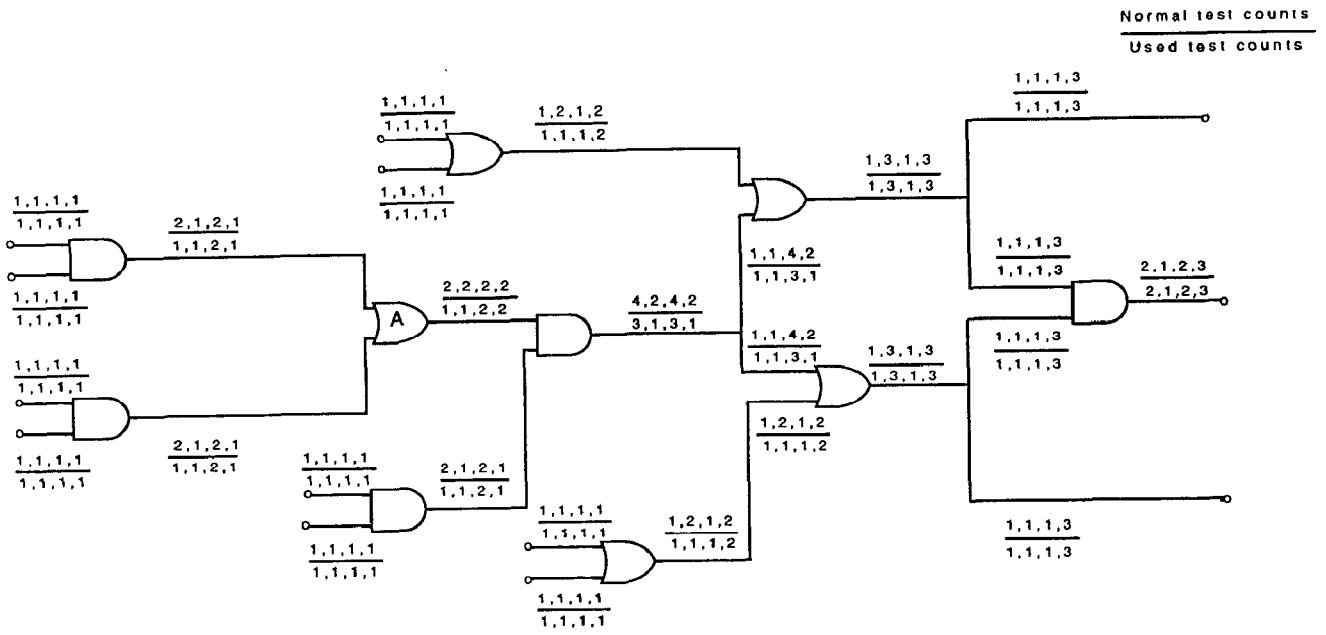


Figure 5

number of the loops into one. Similarly, we have always assumed that the test counting data associated with each lead resides in separate locations. Of course, in an implementation of these algorithms one needs to carry only those data that have not yet been used in the subsequent gate, thereby re-using the space assigned to the data for the previous leads. If we had taken this into account in our algorithms it would have made the description all the more difficult.

As for the time complexity of the algorithms, the reader will readily agree that Algorithms 1, 2 and 3 can all be implemented with a time complexity of $O(N)$, where $N$ is the size of the network —as measured by the number of leads in the network. As for Algorithm 4, the number of operations to perform Step 2.(i) is proportional to $\eta_H - \eta_L$. It would appear that this could become as high as $N$, resulting in an $N^2$ time complexity for the algorithm. It can be shown that with a suitable counting argument that the total number of steps is still $O(N)$.

Algorithm 5 seems to be the most complex to analyze. We will not perform a complete analysis in this paper. We will, however, mention that the time complexity of this algorithm can be bounded by $O(N \cdot t)$, where $t$ is the number of test points to be inserted. Finally, Algorithm 6, as indicated earlier, is comparatively simpler and can easily be shown to be linear. Nevertheless, since the insertion of $t$ test points requires the execution of Algorithm 6 $t$ times, the overall complexity is $O(N \cdot t)$.

Finally, we should mention that we have shown a provably optimal algorithm for inserting $t$ test points within a fanout-free network. However, from a practical sense fanout-free networks are of limited value. On the other hand, all complexity theoretic arguments indicate that corresponding solutions for networks with reconvergent fanout are doomed to be exponential. Our approach, as those adopted in [1, 3, 7], is to solve the problem exactly for fanout-free networks and then lift the solution to networks with fanout. This lifted solution cannot, of course, guarantee optimality. However, previous experience with similar questions, such as test set size estimation has proved that the results for more general networks are very encouraging. In our case we have tried to use the intuition about the problem obtained in solving the fanout-free case to determine the issues that are practically relevant and those that are merely pathological subtleties. By doing so, we believe, we have provided a practically viable algorithm for inserting $t$ test points within a general network with reconvergent fanout.

Of course, while we can establish the computational complexity of the resulting algorithm, experience alone will indicate the effectiveness of the algorithm. It should be pointed out that even empirical analysis is unlikely to indicate its effectiveness, particularly with respect to the decrease in the number of tests required. This is because, estimating that number is itself as difficult!

We have shown a quasi-linear time algorithm for the test point insertion problem. More importantly, we feel that the dynamic programming approaches to the class of problems addressed in this paper is a useful technique in general.

## References


1. Agarwal, V.K. and Masson, G.M., "A Functional Form Approach to Test Coverage in Tree Networks," *IEEE Trans. Comput.*, vol. C-27, pp. 50-52, 1979.
2. Agarwal, V.K. and Masson, G.M., "Generic Fault Characterizations for Table Look-Up Coverage Bounding," *IEEE Trans. Comput.*, vol. C-29, pp. 288-299, 1980.
3. Akers, S.B. and Krishnamurthy, B., "On the Application of Test Counting to VLSI Testing," *1985 Chapel Hill Conference on VLSI*, pp. 343-362.
4. Akers, S.B. and Krishnamurthy, B., *A Test Counting Technique and its Applications,* General Electric Technical Report, 1981.
5. Berger, I. and Kohavi, Z., "Fault Detection in Fanout-Free Combinational Networks," *IEEE Trans. Comput.*, vol. C-22, pp. 908-914, 1973.
6. Garey, M.R. and Johnson, D.S., *Computers and Intractability: A Guide to the Theory of NP-Completeness,* Freeman & Co., San Francisco, CA, 1979.
7. Hayes, J.P. and Friedman, A.D., "Test Point Placement to Simplify Fault Detection," *Fault Tolerant Computing Symposium*, pp. 73-78, 1973.
8. Krishnamurthy, B. and Akers, S.B., "On the Complexity of Estimating the Size of a Test Set," *IEEE Trans. Comput.*, vol. C-33, pp. 750-753, August 1984.
9. Krishnamurthy, B and Sheng, R.L., "A New Approach to the Use of Testability Analysis in Test Generation," *Proc. Intl. Test Conf.*, 1985.