Robust Testability of Primitive Faults using Test Points

Ramesh C. Tekumalla
Intel Corporation
2501 NW 229^{th} Ave.
Hillsboro, OR 97124

Prem R. Menon
Dept. of Elec. & Comp. Eng.
University of Massachusetts
Amherst, MA 01003

Abstract

Recent research has characterized a class of faults, called primitive faults, that must be tested to insure timing correctness of digital circuits. Experimental results have shown that a large fraction of primitive faults usually have only non-robust tests. In this paper, we propose a method of improving robust coverage of primitive faults using test point insertion. An important feature of the proposed method is that instead of considering primitive faults explicitly, single faults are analyzed to determine whether they can be members of robustly testable primitive faults. If not, test points are specified to make any primitive fault containing such faults robustly testable. Results of experiments to determine the effectiveness of the proposed method show significant improvements in fault coverage.

1 Introduction

Correct operation of a circuit at the intended speed requires that the delay of no path exceeds the value determined by the clock period. This is usually verified by delay testing, using the path delay fault model [1]. In this model, it is assumed that the presence of a delay fault increases the total delay along the path.

Recent research in path delay fault testing has focused on various topics like test classification (robust and non-robust), fault classification (redundant, robust-dependent, functionally sensitizable) and synthesis of delay-testable combinational circuits [2, 3, 4, 5, 6, 7, 8, 9]. A test for a path delay fault may depend on delays on other paths in the circuit under test. A test that detects a fault independent of delays in other paths is called a *robust test* [2]. A test that cannot be invalidated by any fault if certain other faults are proven to be not present is called a *validatable non-robust (VNR)* test [6]. A *hazard-free* test is one that does not produce any pulses or glitches on the tested path. In this paper, we will be concerned with hazard-free robust tests.

Robust or VNR tests may not exist for all paths in a circuit. Since the coverage of path delay faults by robust or VNR tests is usually quite low, several techniques have been proposed for improving such coverage. Synthesis techniques for complete robust testability [8, 9] and VNR testability [6] entail considerable area overhead. Test point insertion [10, 11] has been proposed as a means of improving testability, but the area overhead for making all single paths testable may still be high. It may not be necessary to test all paths in the circuit to guarantee correct operation [3, 4]. Lam et al. [4] have defined a class of faults, called *robust-dependent (RD)* faults, that need not be tested if all remaining faults are tested robustly. Chen and Cheng [3] have defined functional sensitizability to identify faults that need not be tested at all.

A necessary and sufficient set of faults that must be tested to guarantee timing correctness of any combinational circuit is defined in [12]. These faults, called *primitive faults*, may include multiple path delay faults. Several methods of identifying primitive faults and generating tests for them have been proposed [13, 14, 15, 16]. Experimental results indicate low robust test coverage of primitive faults [16], and no methods are currently available for generating VNR tests for primitive faults. Synthesis techniques for circuits in which all faults have robust or VNR tests (called delay verifiable circuits [12]) also suffer from large area overhead, and are applicable only to relatively small circuits. Design-for-Testability (DFT) techniques are therefore essential to guarantee that timing correctness of the circuit can be verified by testing. A test-point insertion technique to limit the size of primitive faults to size two, was proposed in [17]. Non-robust tests for all primitive faults are also obtained during test-point insertion, but robust tests may not exist for all primitive faults.

The goal of this paper is to develop a method of test-point insertion to improve robust testability of primitive faults. Our approach uses sensitizing cubes to identify single path delay faults that may be contained in robustly testable primitive faults. These faults need not be considered for test point insertion. Each of the remaining single faults is analyzed to determine whether its membership in a primitive fault will make the latter only non-robustly testable. This analysis also provides information for adding test points to

0-7803-5753-1/99 $10.00

make any primitive fault containing the single fault, robustly testable. The time-consuming process of primitive fault test generation is avoided in the proposed method.

The paper is organized as follows. Section 2 discusses basic definitions and the concept of sensitizability of paths. The proposed methods for identifying single faults that prevent robust testability of primitive faults and inserting test points are presented in Section 3. Experimental results and conclusions are given in Sections 4 and 5, respectively.

2 Sensitizability of paths

We shall review some basic definitions related to path delay fault testing in combinational circuits.

Definition 1: A *path* π between an input and output of a circuit consists of a sequence of gates and leads, $g_0, l_0, g_1, ..., g_i, l_i, ..., l_{n-1}, g_n$, where g_i and l_i are gates (nodes) and leads, respectively. g_0 and g_n are the source and destination of the path, respectively (usually an input and an output of the circuit). For convenience, we shall represent paths by sequences of gates only.

The *controlling value* (*cv*) of a gate determines the output value of a gate independent of the other input values. The output of a gate with the complement of the controlling value, called the non-controlling value (*ncv*), on an input will depend on other input values.

Definition 2: A *multipath* Π consists of a set of single paths $\{\pi_1, \pi_2, ...\pi_n\}$ to the same destination. Every on-path input of every gate G on $\pi_i \epsilon \Pi$ is an on-path input of Π. All other inputs of G are side-inputs of Π.

Definition 3: A *multipath delay fault (MPDF)* on Π is the situation in which the delay on every $\pi_i \in \Pi$ exceeds the clock period for the specified direction of transition (rising or falling) at the destination.

We shall use lower case Greek letters (with subscripts, when necessary) to denote single paths, and upper case Greek letters to denote multipaths. The direction of transition is always that at the destination of the path. Unless otherwise specified, path and fault will refer to multipath and MPDF respectively.

Conditions for non-robust and robust tests defined for single path delay faults [2] also apply to multiple path delay faults. A non-robust test for a rising (falling) transition on a path Π sets every side-input of Π to a final non-controlling value. In a robust test, side-inputs must be at stable non-controlling values when on-path inputs have transitions to final controlling values.

The proposed method of test point insertion uses the concept of sensitizing cubes introduced in [16]. A *cube* is defined as a subset of the set of all input literals. A cube can be represented by the values assigned to a subset of inputs, or as a product of the corresponding literals. Thus, a cube corresponds to a set of input vectors, each of which corresponds to a *vertex* of the cube.

Definition 4[3]: A single path π is functionally sensitized by a cube q if it sets every side-input of π to a non-controlling value when the on-path input has a non-controlling value. When the on-path input(s) is (are) controlling, the side inputs may be unspecified or non-controlling.

Definition 5: A path Π is static sensitized by a vector v if it produces a non-controlling value on every side-input of Π.

Definition 6[18]: An assignment of values to a minimal set of input literals to functionally sensitize a single path is called a *sensitizing cube*. A path Π is *associated* with a sensitizing cube q if (1) it sets every side-input to the non-controlling value when the on-path input is non-controlling and (2) no side input has a controlling value when the on-path inputs are controlling. A sensitizing cube that produces a 1(0) at the destination of a path is referred to as its sensitizing 1(0)-cube.

Sensitizing cubes and paths associated with them can be determined directly from their definitions. We shall now outline a more efficient method presented in [18]. Sensitizing cubes are determined by tracing back from the destination along each single path and assigning signal values to gates on the path to functionally sensitize it. If a controlling value is necessary to produce the specified gate output, the controlling value is assigned to the on-path input of the selected single path. If non-controlling values are required, all gate inputs are assigned the non-controlling value. The process is repeated until primary inputs are reached. All assigned line values are then justified, making only necessary assignments, and all implications determined. A conflict-free assignment of input variables obtained in this manner corresponds to a sensitizing cube. The path(s) associated with a sensitizing cube can be determined by tracing back from the output. This method is illustrated in Example 1.

Example 1: Fig. 1(a) shows a minimal input assignment for functionally sensitizing the single path $d35f$. This corresponds to the sensitizing cube $a\bar{d}$ which is associated with the path $a0135f$ also. Figs. 1(b), (c) and (d) show how the remaining sensitizing 1-cubes of the circuit can be derived. Note that the same sensitizing cube may be generated by functionally sensitizing different paths. □

From the definition of sensitizing cubes, it follows that a sensitizing cube of a multipath Π functionally sensitizes every single path $\pi_i \in \Pi$. It has been shown in [18] that any vertex in a sensitizing cube of a multipath Π static sensitize some multipath $\Psi \supseteq \Pi$.

Definition 7[12]: A fault on a multipath Π in a combinational circuit is primitive *iff* (1) Π is static sensitizable, and (2) no proper subset of Π is static sensitizable.

The following section discusses the method for identifying single faults that cannot be contained in any robustly testable primitive fault, and determining test points to make primitive faults containing them robustly testable.

3 Analysis of single faults

There are three main steps in the proposed method: (1) reducing the number of single faults to be considered, by identifying those that will not be contained in any primitive fault; (2) identifying faults that may be contained in robust testable primitive faults, and excluding them; (3) analyzing the remaining faults to identify test points needed.

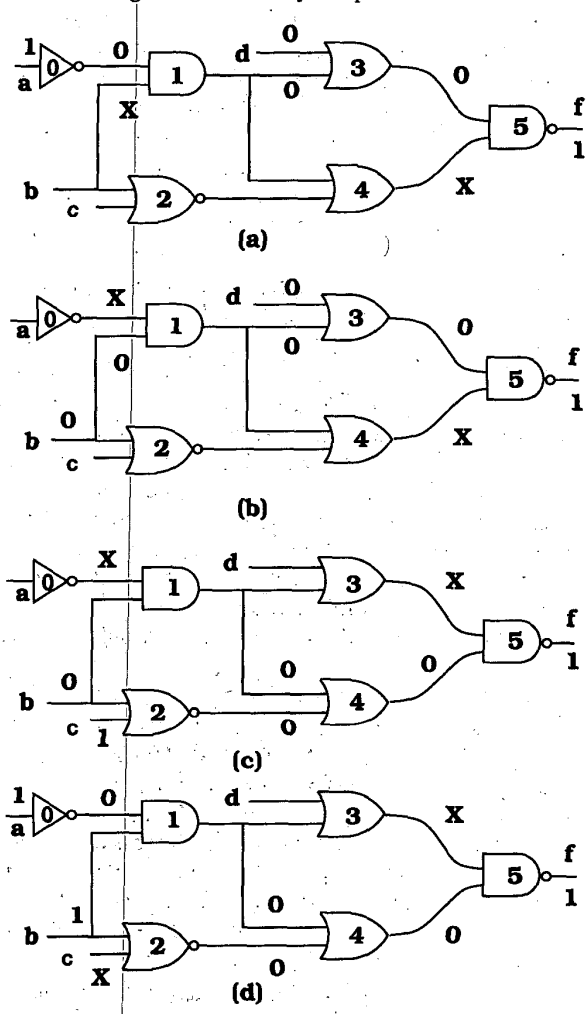

Figure 1. Sensitizing cube identification.

When there is no ambiguity, we may refer to a fault by the name of the single or multiple path that is faulty. The direction of transition and the polarity of sensitizing cubes will not be explicitly mentioned in statements that apply to both rising and falling transitions. Thus, when a rising transition (at the destination) fault is considered, sensitizing cubes will refer to sensitizing 1-cubes, unless stated otherwise. A fault π (Π) will refer to a rising transition fault on the single (multiple) path π (Π).

Lemma 1: Consider a pair of faults f_1 and f_2 on π_1 and π_2. Let Q_1 and Q_2 be the sets of sensitizing cubes that functionally sensitize π_1 and π_2 respectively. The fault f_2 will not be part of a primitive fault if
(1) $Q_1 \supseteq Q_2$;
(2) Every $q \in Q_2$ is associated with a multipath containing both π_1 and π_2; and
(3) every sensitizing cube $q \in Q_2$ associated with a multipath Π can be modified to derive an input condition that sensitizes $\Pi - \{\pi_2\}$ and produces the non-controlling value on π_2 when the on-path input on $\Pi - \{\pi_2\}$ is controlling.

Proof: If conditions (1) and (2) are satisfied, f_2 is always sensitizable with f_1 but is not known if f_1 is sensitizable either by itself or only as a multiple fault that does not include f_2. Consider a sensitizing cube q that functionally sensitizes π_1 and π_2. Let G be a gate at which π_1 and π_2 intersect. If π_1 and π_2 have non-controlling values at G, they are not sensitizable together and either may be part of a primitive fault. Let π_1 and π_2 have controlling values at G, on application of q. Let I be an assignment of values to a set of inputs that produces a non-controlling value on π_2 at G. If it does not conflict with any assignment in q, then $q \bigcap I$ is a sensitizing cube for Π - π_2. Any vertex in $q \bigcap I$ static sensitizes a multipath that does not contain π_2. If the above situation is true for every sensitizing cube of π_2, by Definition 7, Π cannot be primitive. Therefore, π_2 cannot be part of any primitive fault. □

Definition 8: The set of faults obtained after eliminating all single faults that are not part of any primitive fault, is called the set of *collapsed* faults.

If the same set of sensitizing cubes functionally sensitize faults f_1 and f_2, Lemma 1 should still be used to determine which one can be eliminated.

Lemma 2: If a fault on a multipath Π is robustly testable, then every single path $\pi_i \in \Pi$ is functionally sensitizable to both 1 and 0 at its destination.

Proof: Let $< V_1, V_2 >$ be a robust test for the rising transition fault on a multipath Π. Then, $< V_1, V_2 >$ must create an appropriate transition at the start of every $\pi \in \Pi$. Since V_2 static sensitizes Π to 1 at its destination, it must also functionally sensitize every $\pi \in \Pi$ to 1. To show that V_1 functionally sensitizes π to 0, consider gate G_1 with controlling value on π when V_2 is applied. This input must have the non-controlling value when V_1 is applied. Since the transition on π is to a final controlling value and $< V_1, V_2 >$ is a robust test, every side-input of Π at G_1 must have the non-controlling value in both V_1 and V_2. Let G_2 be a gate with the non-controlling value on π when V_2 is applied. V_1 must produce the controlling value at the same input of G_2. Since the transition on π at G_2 is to a final non-controlling value, V_1 may produce either 0 or 1 at side inputs of Π at G_2. Hence, V_1 functionally sensitizes π to 0 at its destination. □

Paths that are functionally sensitizable to both 0 and 1 must

be analyzed further to determine whether primitive faults containing them are robustly testable.

Definition 9: A single fault π is a *non-robust component* if no primitive fault containing π is robustly testable. A single fault is called a *robust component* if it is not a non-robust component.

By the above definition, a primitive fault that is only non-robustly testable must contain at least one non-robust component, while a robustly testable primitive fault cannot contain any non-robust components.

Procedure Component_Type(π)
Determine whether π is a robust or non-robust component
Q_0, Q_1: *sets of sensitizing cubes that functionally sensitize π to 0 and 1, respectively*
$< V_1, V_2 >$: *vector-pair*
Return values: robust, non-robust

$M = \{\ \}$
for every $q_1 \in Q_1$
  Set V_2 to values specified in q_1 ;
  for every $q_0 \in Q_0$
    Set V_1 to values specified in q_0 ;
    Set inputs that are not specified in V_2 but are specified in V_1, to the values in V_1 ;
    Obtain a test $< V_1, V_2 >$ for a multifault m_i ;
    if $m_j \notin M\ \ \forall m_j \subseteq m_i, M = M \bigcup m_i$;
    if $m_j \in M\ \ \forall m_j \supset m_i, M = M - m_j$;
if every $m_i \in M$ has a robust test
  return *robust*
else
  return *non-robust*

Figure 2. Determining component type.

Procedure *Component_Type*, shown in Fig. 2, determines whether a rising transition fault π is a robust or a non-robust component. It attempts to find a robust test $< V_1, V_2 >$, where V_1 and V_2 are vertices in sensitizing 0- and 1-cubes, respectively, that functionally sensitize the path. Tests (which may be robust or non-robust) are obtained for all paths, using every possible combination of sensitizing 0- and 1-cubes. The set M of testable faults is ensured to not contain faults larger than any other fault in the set. If every fault in M is robustly testable, the fault π is classified as a robust component. Otherwise, it is a non-robust component. The roles of the two sensitizing cubes is reversed for falling transition faults.

Procedure *Component_Type* may not classify all faults correctly. It assumes that all sensitizing cubes are needed for testing a circuit. Since a multipath may have more than one sensitizing cube, and not all of which may be needed to obtain a test, some robust components may be classified non-robust. A similar error may be produced when a primitive fault consists of a robust and a non-robust component. When the robust component is analyzed, the procedure may return *non-robust*, due to the non-robust component. Since all faults containing non-robust components are analyzed further for test point insertion, this inaccuracy will not affect the testability of the modified circuit, but may result in some unnecessary analysis.

Procedure *Component_Type* may some times incorrectly classify a fault as a robust component, as shown in Example 2. This may result in certain non-robust components not to be analyzed for test point insertion. Experimental results given in Section 4 show that very few non-robust components are incorrectly identified, and a large number of functionally sensitizable single faults are excluded from consideration for test point insertion. We conjecture that such errors occur only in redundant circuits.

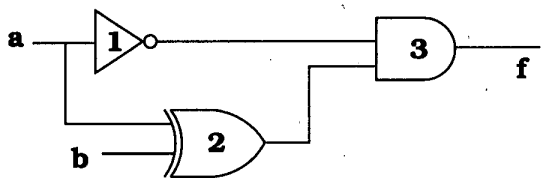

Figure 3. A circuit illustrating Example 2.

Example 2: Consider the circuit in Fig. 3. It has one sensitizing 1-cube $\bar{a}b$ and three sensitizing 0-cubes a, $a \cdot b$ and $\bar{a}\bar{b}$. The cube $\bar{a}b$ is a sensitizing 1-cube for the path $a13f$ and it has two sensitizing 0-cubes a and $a \cdot b$. Using the above sensitizing cubes in Procedure *Component_Type* for testing the falling transition fault on $a13f$, it can be seen that the multiple falling transition fault on $\{a13f, a23f\}$ is tested robustly. However, the procedure does not obtain the non-robust test $< 00, 10 >$ for the single falling transition fault on $a13f$. Hence, the falling transition fault on $a13f$ is incorrectly classified as a robust component □

Lemma 3: Let Π_0 be a primitive fault static sensitized by a common vertex of a set Q of sensitizing cubes. Π_0 is only non-robustly testable if there exists a single path π such that any $q \in Q$ (1) produces the non-controlling value on a side input l_s of a gate when the on-path input is controlling; and (2) there is a partial path π_s from the source of π to l_s that is static sensitized by q.

Proof: The final vector of any test for Π must be contained in q to static sensitize Π. Since π_s is static sensitized and $l_s = ncv$, any test will produce an $ncv \rightarrow cv$ transition on an on-path input of π and a $cv \rightarrow ncv$ transition on a side input. Therefore, the test is non-robust. □

As mentioned earlier, a multipath may have more than one sensitizing cube. As a result, the above lemma must be applied for all sets of sensitizing cubes in which a test may be contained. However, the lemma cannot be applied directly to determine whether a particular fault is a non-robust com-

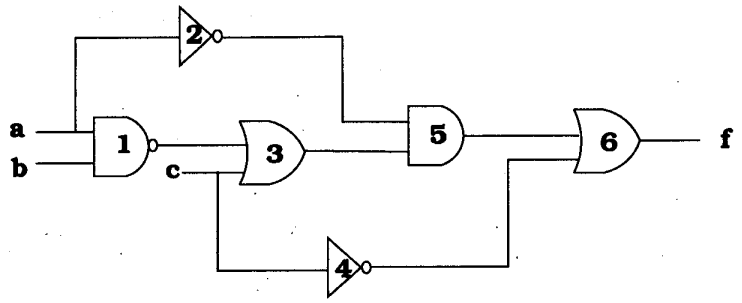

Figure 4. Identifying non-robustly testable faults and test points.

ponent, because primitive faults are not known. We therefore propose two approximate methods for the purpose.

Method 1: Let Q_i denote the set of sensitizing cubes of a multipath Π_i. Let $Q_1, Q_2, ..., Q_k$ be sensitizing cubes that functionally sensitize a single path π. A fault on π is a non-robust component if for some $Q_i, 1 \leq i \leq k$, every sensitizing cube $q \in Q_i$ produces the controlling value the on-path input of a gate G, the non-controlling value on a side-input l_s of G and sensitizes a partial path from the source of π to l_s.

This method assumes that any cube from the set of sensitizing cubes of some multipath containing π may be used in static sensitizing any primitive fault containing π. All non-robust components may not be identified because in some cases, only common vertices of specific cubes may static sensitize a primitive fault. However, we expect it to produce useful testability enhancement without considerable hardware overhead.

Method 2: Treat a fault as a non-robust component if the conditions in Lemma 3 are satisfied by any sensitizing cube that functionally sensitizes it. This method can be expected to result in better fault coverage, but using more test points than Method 1.

Using either of the above approximations, test point insertion proceeds as follows: Let a sensitizing cube q for π produce a *cv* on the on-path input of G and a *ncv* on a side input l_s of G, and let l_s be sensitive to the value at the source of π. Depending on the value at which l_s must be held constant, an AND or an OR gate G' with two inputs, is added to the original circuit. One of the inputs is l_s and the other input is the test point. The output of the new gate is connected to l_s (side-input of G) in the original circuit. If G is an AND or NAND (OR or NOR) gate, then G' is an OR (AND) gate with the test input held at a stable 1 (0) value.

Example 3: The circuit of Fig. 4 has four sensitizing 1-cubes and one sensitizing 0-cube. They are listed below, along with the paths associated with them:

Sensitizing 1-cube	Associated paths
$\bar{a}$	$a256f, a1356f$
$\bar{a}\bar{b}$	$a256f, (a1356f, b1356f)$
$\bar{a}c$	$a256f, (a1356f, c356f)$
$\bar{c}$	$c46f$

Sensitizing 0-cube	Associated paths
ac	$a256f, c46f$

Single paths and sensitizing cubes that functionally sensitize them are as follows:

Single path	Sensitizing cubes	Destination value
$a256f$	$\bar{a}, \bar{a}\bar{b}, \bar{a}c$	1
$a1356f$	$\bar{a}, \bar{a}\bar{b}$	1
$b1356f$	$\bar{a}\bar{b}$	1
$c356f$	$\bar{a}c$	1
$c46f$	$\bar{c}$	1
$a256f$	ac	0
$c46f$	ac	0

Faults on paths $a1356f$, $b1356f$ and $c356f$ each have only sensitizing 1-cubes. By Lemma 2, they are non-robust components. The remaining paths are functionally sensitizable in both directions, and must be analyzed further to determine whether they may be contained in robustly testable primitive paths. Applying Procedure *Component_Type* to path $a256f$, we see that robust tests can be obtained with each of its three sensitizing 1-cubes. For example, using the 0-cube ac and 1-cube $\bar{a}$, we obtain a test $< 1x1, 0x1 >$. Similarly, primitive faults containing path $c46f$ are robustly testable for the rising transition. Paths $a1356f$ and $b1356f$ do not satisfy the conditions of Lemma 3. Applying Lemma 3 to path $c356f$, we see that $a = 0, c = 1$ produces the controlling value on the on-path input of gate 6, and the side input is sensitive to the value at c. Therefore, any primitive fault containing this path will be only non-robustly testable for the rising transition. To make the primitive faults testable, we add a two-input AND gate G' between gates 4 and 6. The inputs of G' are the output of gate 4 and a test point, and its output is connected to gate 6.

The test point input of G' must be held at stable 0 to make the primitive fault involving $c356f$ robustly testable.

It can be verified that the rising transition faults on the single paths $a256f$ and $c46f$ are primitive. The single faults are robustly testable without the need for adding any test points to make them robustly testable. □

Procedure Test_Points()

```
Obtain functionally sensitizable paths ;
Collapse the functionally sensitizable paths ;
for p = 0, 1
  for every π with sensitizing p-cube
    if π has sensitizing a p̄-cube
      if Component_type(π) == non-robust
        R = R ⋃ π ;
    else
        R = R ⋃ π ;
  for every π ∈ R
    Assign test points using Method 1 or 2 ;
```

Figure 5. Test point identification.

The test point identification algorithm is presented in Fig. 5. Lemma 1 is used to determine the collapsed faults. The set R is determined after applying Procedure *Component_Type* on the collapsed faults. Lemma 3 is used to determine test points using the approaches discussed in Methods 1 and 2.

The above method identifies a set of test points for each non-robust component. However, only a subset of test points need be set to specific values to make a particular primitive fault robustly testable. We shall explain how test generation after test point insertion can be done using the test point site information. The set of test points are identified by checking for the conditions specified in lemma 3 for different sensitizing cubes. A set of test points identified on paths sensitized by a cube can be associated with that cube. Since primitive faults are identified by essential and certain common vertices of the sensitizing cubes [16], we can identify the set of sensitizing cubes S containing each vertex that identifies a primitive fault. The set of test points identified for each sensitizing cube in S is sufficient to make the primitive fault identified by the vertex, robustly testable. Note that if a primitive fault is robustly testable without requiring any test points, the set of test points for S will be empty, since none of the sensitizing cubes in S will satisfy the conditions in Lemma 3.

4 Experimental Results

The proposed method has been implemented and applied to combinational versions of ISCAS'89 and MCNC'91 benchmark circuits. The method presented in [18] was used to derive sensitizing cubes treating each output separately. Paths that do not have at least one sensitizing cube (either 0 or 1) are ignored because they cannot affect circuit operation.
Test points are obtained using the method shown in Fig. 5. Both Method 1 and Method 2 were used for identifying test points for non-robust components. The test points thus obtained are used to determine how many non-robust tests for primitive faults [16] become robust, if they are used.
Table 1 gives particulars of paths in the circuits used in our experiments. The number of physical paths and functionally sensitizable logical paths in each circuit in column 1 are given in columns 2 and 3 respectively. Column 4 gives the number of collapsed faults obtained using Lemma 1. The number of robust and non-robust components are given in columns 5 and 6, respectively.
Fault coverage statistics using test points identified by Procedure *Test_Points*, are given in Table 2. Column 2 gives the number of test points obtained using Lemma 3 and Method 1. The corresponding robust fault coverage of primitive faults and run time are given in columns 3 and 4 respectively. Columns 5-8 give the corresponding numbers when Method 2 was used. The last two columns of the table give the robust coverage of primitive faults in the original circuit and the run time for performing it. These were taken from [16]. The run times are on an IBM RS6000 server.
The results show that 100% robust coverage of primitive faults was obtained for most circuits by test point insertion using Method 2. The use of Method 1 resulted somewhat lower coverage, but fewer test points were used. In all cases, both methods led to some improvement in fault coverage, the best improvement being in s1423 (4.35% to 100% with Method 2). In almost all cases, the run time was considerably less than the test generation time for the original circuit. The presence of test points should reduce the amount of search involved during test generation. Therefore, the total run time for test point information and test generation should be less than the time required for the unmodified circuit.

5 Conclusion

We have presented the first method for making non-robustly testable primitive faults in a combinational circuit robustly testable. The method is based on the analysis of single faults which may make primitive faults containing it only non-robustly testable. The number of faults to be analyzed for test point insertion is reduced by first eliminating faults that cannot be contained in any primitive fault and those that do not prevent robust testability.

Circuit	# Physical Paths	# Functionally Sensitizable Faults	# Faults		
			Collapsed	Robust component	NR-component
s298	236	390	356	341	15
s344	299	440	380	245	135
s349	303	441	376	240	136
s382	400	501	476	450	26
s386	207	394	392	391	1
s400	435	507	454	425	29
s444	451	691	629	588	41
s510	371	546	495	426	69
s526	416	678	637	612	25
s641	1695	2414	1701	1483	218
s713	2615	3041	1214	855	359
s820	521	842	782	736	46
s832	533	848	770	736	34
s838	1714	2575	2545	2509	36
s1196	2243	2956	2401	2125	276
s1238	2564	3363	2538	2257	281
s1423	20324	22361	15688	10110	5578
s1488	1408	1835	1704	1326	378
s1494	1055	1463	1354	994	360
planet	2018	2299	2197	1311	886
sl	1239	1350	1317	637	680
scf	3559	4579	4364	3656	708
styr	995	1312	1203	758	445

Table 1. Test point insertion statistics for ISCAS'89 and MCNC'91 benchmark circuits.

Experimental results show improvement in fault coverage in all cases, achieving 100% coverage in most of them. Two methods of identifying test points were proposed, one that inserts fewer test points and obtains smaller improvements than the other. Thus, a trade-off between fault coverage and test points is possible. The execution time of the proposed method is considerably smaller than the time required for primitive fault identification. A test generation program for the modified circuit can be expected to be much faster than one for the original circuit, because of the availability of test point information and the reduction in backtracking resulting form higher testability.

The proposed method can be further improved in a number of ways. The procedure for determining component type was shown to incorrectly label some non-robust components as robust components and excludes them from further analysis. This results in some necessary test points not to be found. A method of identifying such cases will result in better coverage. The two methods of identifying test points are approximate methods, one selecting too few test points, and the other too many. A better method of identifying non-robust components will result in inserting closer to the optimal number of test points. A method that does optimal test point insertion within specified bounds on coverage or number of test points can be developed. Finally, test point insertion and test generation can be combined to produce an efficient system.

References

[1] G.L. Smith, "Model for delay faults based upon paths", *Proc. Intl Test Conf.*, pp. 342-349, Nov. 1985.

[2] C.J. Lin and S.M. Reddy, "On delay fault testing in logic circuits", *IEEE Trans. on CAD*, pp. 694-703, Sept. 1987.

[3] K.-T. Cheng and H.C. Chen, "Delay testing for non-robust untestable circuits", *Proc. Intl Test Conf.*, pp. 954-961, Oct. 1993.

[4] W.K. Lam, A. Saldanha, R.K. Brayton and A.L. Sangiovanni-Vincentelli, "Delay fault coverage and performance tradeoffs" *Proc. Design Automation Conf.*, pp. 446-452, June 1993.

Circuit	Method 1			Method 2			Original	
	# Test Points	Coverage	Time	# Test Points	Coverage	Time	coverage	Time
s298	9	98.87	0.68	9	98.87	0.96	97.71	16.04
s344	4	82.02	0.71	36	100	0.94	80.83	86.81
s349	4	81.59	0.66	39	100	0.96	80.54	82.01
s382	4	94.91	0.63	16	100	0.86	93.75	48.66
s386	6	93.77	0.31	9	95.16	0.46	91.01	35.61
s400	5	92.20	0.69	13	100	0.93	90.80	34.23
s444	18	93.26	0.99	18	93.26	1.44	88.50	94.42
s510	6	87.27	0.93	36	100	1.26	85.71	123.31
s526	12	100	1.45	12	100	2.10	87.35	89.46
s641	13	85.27	32.63	34	100	44.33	82.94	412.61
s713	15	59.89	37.24	32	61.37	57.09	58.59	1050.74
s820	20	100	1.45	20	100	2.86	85.94	438.16
s832	21	100	1.53	21	100	2.83	73.96	534.77
s838	7	99.45	6.09	16	100	9.02	98.98	492.28
s1196	40	78.97	24.54	91	100	31.13	65.20	1572.98
s1238	48	98.43	35.13	88	100	46.52	95.92	945.75
s1423	74	72.31	1065.72	241	100	1633.54	4.35	2012.96
s1488	68	81.18	5.83	132	100	8.39	80.07	1521.19
s1494	22	89.87	2.10	117	100	4.90	85.47	1562.80
planet	110	76.79	4.72	227	100	7.90	67.26	921.67
s1	59	69.20	1.45	187	100	2.97	59.98	682.52
scf	160	62.35	23.85	160	62.35	36.06	56.52	1683.13
styr	46	83.10	1.66	168	100	3.76	70.36	1701.68

Table 2. Fault coverage statistics for ISCAS'89 and MCNC'91 benchmark circuits.

[5] S. Devadas and K. Keutzer, "Synthesis of robust delay-fault-testable circuits:practice", *IEEE Trans. on CAD*, vol.11, no.3, pp. 277-300, Mar. 1992.

[6] S. Devadas and K. Keutzer, "Validatable nonrobust delay-fault-testable circuits via logic synthesis", *IEEE Trans. on CAD*, vol.11, pp. 1559-1573, Dec. 1992.

[7] C. Lin, S.M. Reddy and S. Patil, "An automatic test pattern generator for the detection of path delay faults", *Proc. Intl Conf. on Computer Aided Design*, pp. 284-287, Nov. 1987.

[8] A. K. Pramanick and S.M. Reddy, "On the design of path delay fault testable combinational circuits", *Proc. Intl Symposium on Fault Tolerant Computing*, pp. 374-381, June 1990.

[9] K. Roy, K. De, J.A. Abraham and S. Lusky, "Synthesis of delay fault testable combinational logic", *Proc. Intl Conf. on Computer Aided Design*, pp. 418-421, Nov. 1989.

[10] I. Pomeranz and S.M. Reddy, "Design-for-Testability for path delay faults in large combinational circuits using test-points", *Proc. Design Automation Conf.*, pp. 358-364, June 1994.

[11] P. Uppaluri, U. Sparmann and I. Pomeranz, "On minimizing the number of test points needed to achieve complete robust path delay fault testability", *Proc. VLSI Test Symposium*, pp. 288-295, May 1996.

[12] W. Ke and P.R. Menon, "Synthesis of delay-verifiable combinational circuits", *IEEE Trans. Computers.*, vol. 44, pp. 213-222, Feb. 1995.

[13] M. Sivaraman and A. J. Strojwas, "Primitive path delay fault identification", *Proc.* 10^{th} *Intl Conf. on VLSI Design*, pp. 95-100, Jan. 1997.

[14] A. Krstic, K.-T. Cheng and S.T. Chakradhar, "Primitive delay faults: Identification, testing and design for testability," *IEEE Trans. CAD*, vol. 18, pp.669-684, June 1999.

[15] A. Krstic, K.-T. Cheng and S.T. Chakradhar, "Identification and test generation for primitive faults", *Proc. Intl Test Conf.*, pp. 423-432, Oct. 1996.

[16] R. Tekumalla and P.R. Menon, "Test generation for primitive path delay faults in combinational circuits", *Proc. Intl Conf. on Computer Aided Design*, pp. 636-641, Nov. 1997.

[17] A. Krstic, S.T. Chakradhar and K.-T. Cheng, "Design for primitive delay fault testability", *Proc. Intl Test Conf.*, pp. 436-445, Nov. 1997.

[18] R. Tekumalla and P.R. Menon, "On primitive fault test generation in non-scan sequential circuits", *Proc. Intl Conf. on Computer Aided Design*, pp. 275-282, Nov. 1998.