Test Enrichment for Path Delay Faults Using Multiple Sets of Target Faults

Irith Pomeranz[1] and Sudhakar M. Reddy[2]
School of Electrical & Computer Eng. Purdue University W. Lafayette, IN 47907
Electrical & Computer Eng. Dept. University of Iowa Iowa City, IA 52242

Abstract

Test sets for path delay faults in circuits with large numbers of paths are typically generated for path delay faults associated with the longest circuit paths. We show that such test sets may not detect faults associated with the next-to-longest paths. This may lead to undetected failures since shorter paths may fail without any of the longest paths failing. In addition, paths that appear to be shorter may actually be longer than the longest paths if the procedure used for estimating path length is inaccurate. We propose a test enrichment procedure that increases significantly the number of faults associated with the next-to-longest paths that are detected by a (compact) test set. This is achieved by allowing the underlying test generation procedure the flexibility of detecting or not detecting the faults associated with the next-to-longest paths. Faults associated with next-to-longest paths are detected without increasing the number of tests beyond that required to detect the faults associated with the longest paths. The proposed procedure thus improves the quality of the test set without increasing its size.

1. Introduction

The path delay fault model [1] was proposed as a model to capture small, distributed delay defects. Ideally, all the path delay faults of a circuit should be tested. However, a circuit may have a very large number of paths [2], making it impossible to target all the path delay faults explicitly during test generation or fault simulation. The large numbers of paths in practical circuits led to the use of path selection, where only a subset of the path delay faults in a circuit are targeted for test generation. A commonly used criterion for path selection is to consider only the faults associated with the longest (critical) paths in the circuit. In [3], paths are selected such that every line in the circuit is included in at least one selected path which is one of the longest paths through the line.

Procedures to generate compact test sets for path delay faults were described in [4]-[6]. Path selection was not needed in [5] and [6], since only circuits with small numbers of paths are reported. The procedure of [4] targets the faults selected by the path selection procedure of [3]. Other test generation procedures for path delay faults also target fixed sets of faults associated with longest paths, e.g., [7].

In this work, we observe that it is possible to improve the quality of a (compact) test set for path delay faults, without increasing its size, by considering two (or more) sets of target faults. Under the approach proposed here, the set of faults P_0 is the set of faults that would be targeted by a conventional test generation procedure. These may be the faults associated with the critical paths of the circuit, or faults selected based on the criterion of [3]. In addition, we use a set of faults P_1 that contains faults which are less critical than the faults in P_0; however, detecting the faults in P_1 would improve the quality of the test set. For example, consider a circuit with paths of lengths $L_0, L_1, \cdots, L_{n-1}$ such that $L_0 > L_1 > \cdots > L_{n-1}$. Let $n_p(L_i)$ be the number of faults associated with paths of length L_i. We may include in P_0 faults associated with paths of lengths L_0 and L_1 for a total of $n_p(L_0)+n_p(L_1)$ faults. In addition, we may include in P_1 faults associated with paths of lengths L_2, L_3 and L_4 for a total of $n_p(L_2)+n_p(L_3)+n_p(L_4)$ faults. Our primary objective is to detect the faults in P_0. However, whenever possible without increasing the number of tests, we will also try to detect faults out of P_1.

Conceptually, the proposed approach is different from other approaches in that it has the flexibility of detecting or not detecting some of its target faults (the faults in P_1). The importance of detecting the faults out of P_1 results from the fact that L_i and L_{i+1} are typically very close. Thus, the paths in P_0 are not significantly longer than the paths in P_1, and small errors in the computation of the path lengths can result in a path that was placed in P_1 being longer than a path placed in P_0. While a conventional test generation procedure would not attempt to

1. Research supported in part by NSF Grant No. CCR-0098091 and in part by SRC Grant No. 2001-TJ-950.
2. Research supported in part by NSF Grant No. CCR-0097905 and in part by SRC Grant No. 2001-TJ-949.

detect faults in P_1, the proposed approach will, compensating for errors in the computation of path lengths.

We refer to the proposed approach as a *test enrichment* procedure since it increases the number of faults detected by each test while keeping the total number of tests the same. Under the proposed test enrichment procedure, the test set size is determined by the requirement to detect the faults in P_0. However, the quality of the test set is enhanced by ensuring that the tests generated for faults in P_0 detect as many faults in P_1 as possible. Thus, the detection of the faults in P_1 is "free" in terms of the number of tests. In our implementation, we ensure that the number of faults in P_1 is reasonably small so as not to increase the computational effort required for test generation beyond an acceptable level. In general, the sizes of P_0 and P_1 can be adjusted to control the test generation effort.

Experimental results presented in this work demonstrate that many faults in P_1 are not detected accidentally by tests for the faults in P_0. Thus, targeting faults out of P_1 explicitly is important in improving the quality of the test set. We also demonstrate that large numbers of faults out of P_1 can be detected without increasing the number of tests compared to the number of tests required to detect only the faults in P_0. We demonstrate this point using compact test sets for the faults in P_0 where the flexibility of detecting additional faults is low compared to non-compact test sets.

We consider only robust tests in this work. The paper is organized as follows. In Section 2 we describe the basic test generation procedure we use for a single set of target faults P. In Section 3 we describe how multiple sets of target faults are selected, and the modifications required to the basic test generation process in order to accommodate multiple sets of target faults. Experimental results are given in Section 4. Section 5 concludes the paper.

2. The basic test generation procedure

In this section, we describe the basic test generation procedure we use for a given set of target faults P. We also describe the compaction heuristics we employ to ensure that the resulting test set is as small as possible. These details will be needed later to describe the test enrichment process that targets multiple sets of faults and to demonstrate its effectiveness.

2.1 Test generation

To robustly detect a path delay fault p, it is necessary and sufficient to find a two-pattern test t that assigns to the off-path inputs of the path the values required for robust propagation of a transition along the path, and in addition assigns to the source of the path the appropriate transition ($0 \rightarrow 1$ for a slow-to-rise path delay fault, and $1 \rightarrow 0$ for a slow-to-fall path delay fault). We denote by $A(p)$ the set of values that a test for p must assign. A pair $(g, \alpha) \in A(p)$ indicates that line g must carry the values designated by α. The constant α is a triple $\alpha_1\alpha_2\alpha_3$, where α_1 is the value that must be assigned to g under the first pattern of the test, α_3 is the value that must be assigned to g under the second pattern of the test, and α_2 is the intermediate value of g (for a stable value, $\alpha_1 = \alpha_2 = \alpha_3$; for a rising transition, $\alpha_1 = 0$, $\alpha_2 = x$ and $\alpha_3 = 1$; and for a falling transition, $\alpha_1 = 1$, $\alpha_2 = x$ and $\alpha_3 = 0$). For example, we consider the combinational logic of ISCAS-89 benchmark circuit $s27$ shown in Figure 1. For the slow-to-rise fault on the path (2,9,10,15), $A(p)$ consists of the off-path values 000 on line 7 and $xx0$ on line 3, and of the source value $0x1$ on line 2.

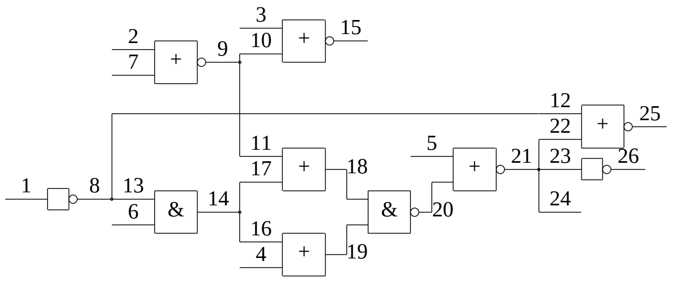

Figure 1: ISCAS-89 benchmark circuit $s27$

For a test t to detect the path delay faults $p_1, p_2, \cdots, p_m$, it is necessary and sufficient for t to assign the values included in $\bigcup_{i=1}^{m} A(p_i)$.

Test generation for a path delay fault p (or a subset of path delay faults $p_1, p_2, \cdots, p_m$) is a justification process that searches for a two-pattern test to satisfy all the values in $A(p)$ (or $\bigcup_{i=1}^{m} A(p_i)$). The justification process we use is simulation-based. Initially, every primary input b_i is assigned a triple $\beta_i = \beta_{i_1}\beta_{i_2}\beta_{i_3} = xxx$. During the justification process, we check repeatedly for *necessary* values, as follows. For every β_{i_j}, we check whether setting β_{i_j} to 0 conflicts with any value in $A(p)$, and whether setting β_{i_j} to 1 conflicts with any value in $A(p)$. If both values result in a conflict, we stop the test generation process without finding a test. If only one of the values results in a conflict, we assign the other value to β_{i_j} permanently under t. Otherwise, we leave β_{i_j} unspecified. This is repeated as long as new values are assigned to the primary inputs.

When no new necessary values can be found, we select an unspecified value β_{i_j}, where $j = 1$ or 3, and we assign a specified value to it, as follows. If there is a primary input b_i with a triple $\beta_{i_1}\beta_{i_2}\beta_{i_3}$ such that only β_{i_1} is specified, we assign the value of β_{i_1} to β_{i_2} and β_{i_3}. Similarly, if only β_{i_3} is specified, we assign the value of β_{i_3} to β_{i_1} and β_{i_2}. If no such input can be found, we randomly select an unspecified value β_{i_j} such that $j = 1$ or 3, and we specify it randomly. After a value is assigned in this way, we again assign as many necessary values as can be found. This is repeated until all the primary inputs are specified or a conflict occurs.

2.2 Test compaction

Compaction in our test generation procedure is based on the use of primary and secondary target faults to generate every test [8]. Test generation with primary and secondary target faults in [8] uses values that remain unspecified under a test t in order to detect additional faults. The process proceeds as follows. Let P be the set of target faults. To generate a new test t, a target fault $p_0 \in P$ is selected. This fault is called the primary target fault. If a test t can be generated for p_0, another target fault $p_1 \in P$ is selected, and an attempt is made to specify unspecified values in t in order to detect p_1. After p_1, another fault $p_2 \in P$ is selected, and an attempt is made to specify unspecified values in t in order to detect p_2. This is repeated until t is fully specified, or all the faults in P have been considered. The faults $p_1, p_2, \cdots$ are called secondary target faults. Before every secondary target fault is selected, specified values in t may be unspecified to increase the number of unspecified values available for the next target fault. Once the generation of t is complete, fault simulation is carried out for all the faults in P, and the detected faults are dropped from P.

Since our simulation-based justification procedure results in fully-specified tests, we cannot use unspecified values in t to detect secondary target faults. Instead, we use the following approach. We define a subset of faults $P(t)$ that initially consists of the primary target fault p_0. We attempt to add to $P(t)$ secondary target faults $p_1, p_2, \cdots, p_m$ one at a time. After p_i is added to $P(t)$, we apply the justification procedure to generate a test t that satisfies all the values in $\cup\{A(p_j) : p_j \in P(t)\}$, with p_i included in $P(t)$. The addition of p_i to $P(t)$ is accepted if the simulation-based justification procedure can generate such a test t. Otherwise, p_i is removed from $P(t)$.

Although the test generation effort is increased compared to [8] since we generate a new test after every fault is added to $P(t)$, the use of a simulation based procedure compensates for this increase. More important, the number of secondary target faults that can be detected is potentially increased, since we are not restricted by values specified under t in order to detect faults that were added to $P(t)$ earlier. New values can be specified under t after p_i is added to $P(t)$ if they are more suitable for detecting p_i.

The level of compaction achieved by a test generation procedure that uses primary and secondary target faults is influenced to a large extent by the order in which the primary and secondary target faults are selected for test generation. To obtain a smaller number of tests, the fault order should ensure that a larger number of secondary target faults would be detected by every test. We compare three heuristics, described next, in terms of their ability to minimize the number of tests.

Arbitrary order: Under this order, both the primary and the secondary target faults are selected in the order they appear in the fault list P, which is an arbitrary order.

Length-based order: Under this heuristic, the primary target fault is selected such that the path it is associated with is the longest of all the faults in P. The secondary target faults are also selected according to the same order. The longer paths tend to require more values that the test t must satisfy. Thus, the associated faults are not likely to be detected accidentally. If they are left to the end of the test generation process, it is likely that each fault would require an additional test. At the beginning of the process, we have the flexibility of detecting other faults together with them, thus minimizing the number of tests. A similar heuristic was used in [4].

Value-based order: Under this heuristic, the primary target fault p_0 is selected such that the path it is associated with is the longest of all the faults in P. This is the same criterion used to select the primary target fault under the length-based order. For the secondary target faults, we attempt to minimize the number of new values that the test t will have to satisfy. This will make it more likely that t will be able to detect the new fault in addition to all the faults already included in $P(t)$. As a result, it is more likely that t will detect a large number of faults. We use the following procedure to select the next secondary target fault. For every fault $p_i \in P - P(t)$, we find the set of values that will have to be added to $\cup\{A(p_j) : p_j \in P(t)\}$ in order to detect p_i. We denote this set by $\Delta(p_i)$. We have $\Delta(p_i) = A(p_i) - \cup\{A(p_j) : p_j \in P(t)\}$. We denote the size of $\Delta(p_i)$ by $n_\Delta(p_i)$. We select the fault p_i for which $n_\Delta(p_i)$ is minimum.

For comparison purposes, we also consider a test generation procedure that does not attempt to detect secondary target faults. Under this procedure, tests are generated for primary target faults without attempting to maximize the number of additional faults detected by each test. This procedure will allow us to demonstrate the levels of compaction achievable by the heuristics above. This is important since our goal is to show that even with a compact test set, the use of a second set of target faults can improve the quality of the test set significantly.

3. Test enrichment

In this section, we describe the derivation of multiple sets of target faults, and the proposed test enrichment process that performs test generation with multiple sets of target faults.

3.1 Sets of target faults

Sets of target faults for test generation can be defined in various ways. In our study, we include in the first set, P_0, the faults associated with the longest paths of the circuit, and we include in the second set, P_1, faults associated with the next-to-longest paths.

To define P_0 and P_1, we enumerate a set of path delay faults P that consists of the N_P faults associated with the longest paths in the circuit, where N_P is a preselected constant. We ensure that P includes *all* the faults associated with the longest paths, *all* the faults associated with the second-to-longest paths, and so on, without allowing the size of P to exceed N_P. In our implementation, $N_P = 10000$ (in general, N_P can be determined by considering the number of paths of every length and taking into account an acceptable bound on test generation effort). We eliminate from P as many undetectable faults as possible. We then select faults for P_0 as described below. The remaining faults out of P are included in P_1, i.e., we set $P_1 = P - P_0$.

To determine P_0, let the path lengths in P be $L_0, L_1, \cdots, L_{n-1}$ such that $L_0 > L_1 > \cdots > L_{n-1}$. Let $n_p(L_i)$ be the number of faults associated with paths of length L_i. In benchmark circuits, $n_p(L_0)$ is typically very small. Therefore, we include in P_0 faults associated with paths of lengths $L_0, L_1, \cdots, L_{i_0}$, where i_0 is such that the number of path delay faults in P_0 is not smaller than a constant N_{P_0}. In our implementation, $N_{P_0} = 1000$ (in general, N_{P_0} can be determined similar to N_P based on the circuit and the test generation effort).

Next, we provide a more detailed description of the derivation of P, P_0 and P_1. We start with a procedure suitable for circuits with moderate numbers of paths. We then extend this procedure to circuits with large numbers of paths.

We use an explicit enumeration procedure to find the path delay faults associated with the longest paths of the circuit. Paths are enumerated by starting from the primary inputs and adding lines that expand the paths towards the primary outputs. At an arbitrary stage of this procedure, we have a set of faults P. Some of the faults in P are associated with complete paths (paths that start at a primary input and end at a primary output), and some of the faults in P are associated with partial paths (paths that start at a primary input but end before reaching a primary output). Every time the number of faults in P reaches or exceeds a constant N_P, we remove from P the faults associated with the shortest complete paths included in P. Faults are removed until the number of faults in P is lower than N_P. We do not eliminate faults associated with the longest complete paths during this process. For circuits with moderate numbers of paths, we could always reduce the number of paths below N_P by eliminating faults associated with paths that are (significantly) shorter than the longest paths in P.

For example, we apply the enumeration process to the combinational logic of $s27$ shown in Figure 1. We assume that the delay of a path is equal to the number of lines along the path (other delay models can be accommodated by the procedure we use). For simplicity, we consider paths and not faults in this example (each path is associated with two faults). We use an upper bound of $N_P = 20$ on the number of paths included in P.

The path enumeration process starts by including each primary input in a path. We thus have P = {(1)p, (2)p, (3)p, (4)p, (5)p, (6)p, (7)p}. The p following a path indicates that this is a partial path (i.e., the path does not end at a primary output). We select the first path, (1), and extend it into the path (1,8). Next, we extend this path in all possible ways. We obtain the paths (1,8,12) and (1,8,13). Both paths are entered into P to obtain P = {(1,8,12)p, (2)p, (3)p, (4)p, (5)p, (6)p, (7)p, (1,8,13)p}. We continue with the first path, and obtain (1,8,12,25). This path now reaches a primary output, and its construction terminates. We obtain P = {(1,8,12,25)c, (2)p, (3)p, (4)p, (5)p, (6)p, (7)p, (1,8,13)p}, where c stands for a complete path (i.e., a path that ends at a primary output). Next, we extend the second path, (2). We obtain (2,9) and then the two paths (2,9,10) and (2,9,11). We now have P = {(1,8,12,25)c, (2,9,10)p, (3)p, (4)p, (5)p, (6)p, (7)p, (1,8,13)p, (2,9,11)p}. Construction of P continues until we obtain the set of paths shown in Table 1(a), that contains 20 paths. Among the complete paths, the shortest path length is two. We reduce the size of P by omitting the complete path (3,15) of length two. Since the size of P is lower than 20 after the omission, we do not remove any additional paths. Construction of P continues by extending the first partial path in Table 1(a), (1, 8, 13, 14, 16, 19, 20, 21, 22). We skip several steps, and consider the set of paths shown in Table 1(b). This set contains 21 paths, and we need to omit some of the complete paths in order to reduce its size. In this case, it is necessary to omit complete paths of lengths three and four before the size of P goes below 20. The construction of P ends with a set of 18 paths of lengths between 7 and 10. Thus, the paths of lengths 2, 3 and 4 that we eliminated are significantly shorter than the longest paths.

Table 1: Paths of $s27$

(a) Set 1	**(b) Set 2**
(1,8,12,25)c	(4,19,20,21,22,25)c
(2,9,10,15)c	(6,14,16,19,20,21,22,25)c
(3,15)c	(1,8,13,14,16,19,20,21,22,25)c
(4,19,20,21,22,25)c	(2,9,11,18,20,21,22,25)c
(5,21,22,25)c	(4,19,20,21,23,26)c
(6,14,16,19,20,21,22,25)c	(4,19,20,21,24)c
(7,9,10,15)c	(5,21,23,26)c
(1,8,13,14,16,19,20,21,22)p	(5,21,24)c
(2,9,11)p	(6,14,17,18,20,21,22,25)c
(4,19,20,21,23)p	(6,14,16,19,20,21,23,26)c
(4,19,20,21,24)p	(6,14,16,19,20,21,24)c
(5,21,23)p	(7,9,11,18,20,21,22)p
(5,21,24)p	(1,8,13,14,17)p
(6,14,17)p	(1,8,13,14,16,19,20,21,23)p
(6,14,16,19,20,21,23)p	(1,8,13,14,16,19,20,21,24)p
(6,14,16,19,20,21,24)p	(2,9,11,18,20,21,23)p
(7,9,11)p	(2,9,11,18,20,21,24)p
(1,8,13,14,17)p	(6,14,17,18,20,21,23)p
(1,8,13,14,16,19,20,21,23)p	(6,14,17,18,20,21,24)p
(1,8,13,14,16,19,20,21,24)p	(7,9,11,18,20,21,23)p
	(7,9,11,18,20,21,24)p

For circuits with large numbers of paths, the number of path delay faults in P may exceed N_P even after eliminating all the faults associated with complete paths that are not the longest paths in P. In this case, we use the extension described next. We associate with every line g in the circuit its distance from the primary outputs. In this work, the distance for g is the maximum number of lines along a path from g to the primary outputs. The distance of every line in the circuit is computed in one pass over the circuit starting from the primary outputs and progressing towards the primary inputs.

For every path delay fault in P associated with a path p, let g be the last line of p, and let $d(g)$ be the distance of g from the primary outputs. The maximum length of any path that has p as its prefix is equal to the length of p plus $d(g)$. This is illustrated in Figure 2. In the figure, $d(g)$ is the length of the longest path out of $\{p_1,p_2,p_3\}$. Let this be p_1. The longest path that includes p is the path (p,p_1). Its length is equal to the length of p plus the length of p_1, which is equal to $d(g)$. We denote the maximum length of any path that has p as its prefix by $len(p)$. We use $len(p)$ in two ways.

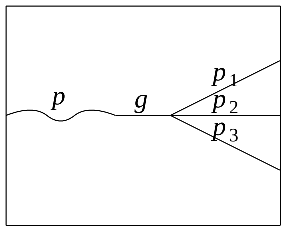

Figure 2: Distance

(1) We always extend the path delay fault in P which is associated with a path p for which $len(p)$ is maximum.

(2) When the number of faults in P reaches N_P, we remove from P faults associated with complete and partial paths p for which $len(p)$ is minimum. We continue to remove path delay faults with the minimum value of $len(p)$ until all the path delay faults have the same, maximum path length, or the number of path delay faults in P is lower than N_P.

Using the distances $d(g)$ and lengths $len(p)$ as above, we can eliminate from P path delay faults associated with partial paths that will not result in the longest paths of the circuit, as well as path delay faults associated with complete paths that are not the longest paths. Thus, more path delay faults can be eliminated than when distances are not used

Once enumeration of P is completed, we eliminate from P two types of undetectable faults. (1) If the set of values $A(p)$ of a path delay fault $p \in P$ contains conflicting values for a line g, p is undetectable and it is eliminated from P. (2) We find the implications of the values in $A(p)$ for every $p \in P$. If the implication process assigns conflicting values to a line g, p is undetectable and it is eliminated from P. It is also possible to use a method for identifying undetectable faults, such as the ones from [9]-[12], to prevent undetectable faults from being included in P.

Once P is found, we select P_0 and P_1 out of P. We use the following notation to describe the selection of P_0 and P_1.

Let $n_p(L_i)$ be the number of faults in P associated with paths of length L_i. We define $N_p(L_i) = \sum\{n_p(L_j):L_j \geq L_i\}$, i.e., $N_p(L_i)$ is the number of faults in P associated with paths of length L_i or higher. Table 2 shows the values of L_i and $N_p(L_i)$ for the 20 highest path lengths in the combinational logic of ISCAS-89 benchmark circuit $s1423$. In this circuit, the longest path is of length 96, and we have $L_0 = 96$, $L_1 = 95$, $\cdots$, $L_{19} = 77$. The number of faults in P associated with paths of length L_0 is $n_p(L_0) = 4$; the number of faults associated with paths of length L_1 is $n_p(L_1) = 8$, resulting in $N_p(L_1) = 12$

(i.e., 12 faults associated with paths of lengths $L_1 = 95$ and $L_0 = 96$); and so on.

Table 2: Numbers of faults in $s1423$

i	L_i	$N_p(L_i)$
0	96	4
1	95	12
2	94	22
3	93	36
4	92	54
5	91	84
6	90	118
7	89	160
8	88	208
9	87	256
10	86	314
11	85	378
12	84	458
13	83	556
14	82	668
15	81	799
16	80	934
17	79	1116
18	78	1314
19	77	1538

We select the first set of target faults, P_0, such that it would satisfy the following conditions. (1) P_0 includes *all* the faults associated with the longest paths in P. For this purpose, we select a length $L_i = L_{i_0}$, and include all the faults associated with paths of length L_{i_0} or higher in P_0. (2) The number of faults in P_0 would be close to a constant N_{P_0}, but not smaller than N_{P_0}. Thus, we use the highest length L_{i_0} (or the smallest value of i_0) for which $N_p(L_{i_0}) \geq N_{P_0}$. In our implementation, we use $N_{P_0} = 1000$. For $s1423$ with $N_{P_0} = 1000$, the first value of i_0 that results in P_0 of size 1000 or more is $i_0 = 17$. This value corresponds to $L_{17} = 79$, and defines a set of target faults P_0 of size 1116. Note that a lower value of i_0, $i_0 = 16$, corresponds to a longer path length, $L_{16} = 80$, and will result in fewer than 1000 path delay faults in P_0 if selected.

We include the remaining faults of P in P_1, i.e., $P_1 = P - P_0$.

It is possible to partition P into a a larger number of subsets. We consider only two subsets in this work.

3.2 Test generation

Next, we describe the test generation procedure with multiple sets of target faults.

Of the three compaction heuristics described in Section 2, we selected the value-based heuristic as the compaction heuristic for the proposed test generation procedure. Experimental results presented in Section 4 demonstrate that this heuristic is effective in minimizing the number of tests.

The test generation process with sets of target faults P_0 and P_1 proceeds as follows.

We select a primary target fault $p_0 \in P_0$. If p_0 can be detected, we include it in the set of faults $P(t)$ to be detected by the current test t. We then select secondary target faults out of P_0 one at a time. We add a secondary target fault $p_i \in P_0$ to $P(t)$ if it can be detected together with the faults already in $P(t)$. Once all the faults in P_0 have been considered, we consider secondary target faults out of P_1. We add a secondary target fault $p_i \in P_1$ to $P(t)$ if it can be detected together with the faults already in $P(t)$. The generation of t terminates after considering all the faults in P_1. Next, another primary target fault $p_0 \in P_0$ is selected, and the process is repeated until all the faults in P_0 are either detected or have been tried as primary target faults.

It is important to note that the faults in P_1 are not selected as primary target faults. In addition, a secondary target fault out of P_1 is selected only if no secondary target fault out of P_0 can be detected by the test t. As a result, faults in P_1 are detected without increasing the number of tests.

4. Experimental results

In this section, we first provide experimental results to demonstrate that compact test sets are generated by the basic test generation procedure we described, i.e., when a single set of target faults P_0 is used. We then show the importance of using a second set of target faults P_1 in order to improve the number of faults detected by the resulting test set. This improvement is achieved without increasing the number of tests.

We consider the combinational logic of ISCAS-89 benchmark circuits and ITC-99 benchmark circuits. We only consider circuits with at least 1000 paths. We use eight circuits, $s641$, $s953$, $s1196$, $s1423$ and $s1488$ from the ISCAS-89 set and $b03$, $b04$ and $b09$ from the ITC-99 set, for comparison purposes. We consider additional circuits under the proposed test generation procedure later.

In Tables 3 and 4 we show the results obtained by the basic test generation approach (where a single set of target faults P_0 is used for test generation) using the various compaction heuristics described in Section 2. In Table 3, after the circuit name, we show the value of i_0 based on which P_0 is defined (P_0 includes all the faults associated with paths of lengths $\geq L_{i_0}$, and i_0 for which we report the results is the first one for which the size of P_0 is at least 1000). We then show the total number of target faults (the number of faults in P_0). Under column *P0 detected* we show the number of faults detected using each one of the compaction heuristics of Section 2. In Table 4, under column *P0 tests* we show the number of tests obtained using each one of the compaction heuristics of Section 2. We use *uncomp* to denote the basic test generation procedure with no compaction heuristics, *arbit* for the arbitrary order heuristic, *length* for the length-based compaction heuristic, and *values* for the value-based compaction heuristic.

Table 3: Basic test generation using P_0 (detected faults)

		P0	P0 detected			
circuit	i0	flts	uncomp	arbit	length	values
s641	57	1057	915	915	915	915
s953	15	1236	1231	1231	1231	1231
s1196	13	1033	572	572	572	572
s1423	17	1116	929	931	932	924
s1488	10	1184	1148	1148	1148	1148
b03	8	1006	869	869	869	869
b04	5	1606	458	456	461	456
b09	1	1432	944	944	944	944

Table 4: Basic test generation using P_0 (numbers of tests)

		P0 tests			
circuit	i0	uncomp	arbit	length	values
s641	57	471	135	130	129
s953	15	581	308	303	312
s1196	13	329	175	172	175
s1423	17	495	332	335	324
s1488	10	464	321	321	317
b03	8	299	90	88	96
b04	5	457	301	304	302
b09	1	406	147	147	158

Table 3 shows small variations in the numbers of faults detected using each one of the heuristics. These are due to the random selection of values during test generation, and can be eliminated by using a branch-and-bound procedure instead of a simulation-based procedure for justification. From Table 4 it can be seen that all three compaction heuristics reduce the number of tests compared to the case where no compaction heuristics are used. We selected the value-based heuristic for the underlying test generation process embedded in the proposed test enrichment procedure.

For comparison with the proposed enrichment procedure, we simulated the faults in $P_0 \cup P_1$ under the test sets generated by the basic test generation procedure. This experiment will provide information about the number of faults out of P_1 that are accidentally detected when only the faults in P_0 are targeted explicitly. The numbers of faults detected are shown in Table 5. Under column *P0,P1 faults* we show the total number of faults in $P_0 \cup P_1$. Under column *P0,P1 detect* we show the numbers of faults detected out of $P_0 \cup P_1$ by the various test sets generated using the basic test generation procedure.

Table 5: Simulation of $P_0 \cup P_1$

		P0,P1	P0,P1 detect			
circuit	i0	faults	uncomp	arbit	length	values
s641	57	2127	1452	1436	1417	1420
s953	15	2312	1830	1759	1781	1778
s1196	13	4527	1414	1338	1312	1341
s1423	17	1314	1013	1019	1017	1007
s1488	10	1918	1697	1641	1651	1654
b03	8	1450	1057	1038	1035	1025
b04	5	8370	936	935	941	936
b09	1	2207	1160	1160	1160	1160

It is interesting to note, based on Table 5, that the number of faults accidentally detected out of P_1 by a non-compact test set is only slightly higher than the numbers of faults out of P_1 that are accidentally detected by compact test sets that are significantly smaller.

In Table 6 we show the results of the proposed enrichment procedure using the faults in P_0 and P_1 as target faults. After the circuit name and the value of i_0, we show the total number of faults in P_0 and the number of faults detected out of P_0. We then show the total number of faults in $P_0 \cup P_1$, and the number of faults detected out of this set. In the last column we show the number of tests. We include in Table 6 circuits considered in Tables 3, 4 and 5, as well as additional circuits. For the additional circuits, we use their more testable versions resynthesized in [13]. In this way, we obtain a larger number of testable path delay faults to consider. The following points can be seen from Tables 3, 4, 5 and 6.

Table 6: Results of test enrichment using P_0 and P_1

		P0 faults		P0,P1 faults		
circuit	i0	total	detect	total	detected	tests
s641	57	1057	915	2127	1815	127
s953	15	1236	1231	2312	2063	315
s1196	13	1033	572	4527	1932	174
s1423	17	1116	934	1314	1039	332
s1488	10	1184	1148	1918	1746	317
b03	8	1006	869	1450	1178	95
b04	5	1606	459	8370	1485	303
b09	1	1432	944	2207	1301	150
s1423*	24	1061	982	1593	1227	267
s5378*	3	1028	913	8537	5469	441
s9234*	7	1158	1158	9344	1465	824

* - resynthesized circuit from [13]

The test set sizes of the proposed procedure are very close to those produced by the basic test generation procedure with the value-based compaction heuristic. The small differences are a result of the fact that we select some primary input values randomly during the test generation procedure. The variations sometimes result in a slightly larger test set, and sometimes in a slightly smaller test set than that produced by the basic test generation procedure.

Compared to the basic test generation procedure with any one of the compaction heuristics, the proposed enrichment procedure detects a significantly larger number of faults out of P_1. This indicates that accidental detection of the faults in P_1 does not have a high likelihood of occurring, and it is important to target these faults explicitly in order to improve the quality of the test set.

Compared to the basic test generation procedure without any compaction heuristics, the proposed enrichment procedure generates significantly smaller test sets while still detecting a larger number of faults out of P_1. This indicates that compaction does not reduce the quality of the test set in this case.

Finally, we report in Table 7 the increase in run time due to the proposed enrichment procedure compared to the basic procedure. We consider the circuits of Table 3 under the value-based heuristic which is also used in the proposed procedure. We report the ratio RT_{enrich}/RT_{basic}, where RT_{enrich} is the run time of the test enrichment procedure, and RT_{basic} is the run time of the basic procedure.

Table 7: Run time ratios

circuit	i0	ratio
s641	57	1.10
s953	15	1.56
s1196	13	2.51
s1423	17	0.94
s1488	10	1.22
b03	8	1.13
b04	5	1.13
b09	1	1.60

5. Concluding remarks

We described a test enrichment approach for path delay fault test generation that uses multiple sets of target faults to improve the quality of the test set generated. The first set of target faults contains faults associated with the critical paths of the circuit. These faults must be detected by the test set (if they are detectable). The second set of target faults contains faults associated with the next-to-longest paths. The test enrichment procedure attempts to detect as many of these faults as possible, but without increasing the number of tests. In this way, the proposed procedure improves the quality of the test set without increasing its size.

We presented experimental results to show that many faults in the second set of target faults are not detected accidentally by tests for the faults in the first set. Thus, targeting faults out of the second set explicitly is important in improving the quality of the test set. We also demonstrated that large numbers of faults out of the second set can be detected without increasing the number of tests compared to the number of tests required to detect only the faults in the first set. This was demonstrated using compact test sets where the flexibility of detecting additional faults is low compared to non-compact test sets.

References

[1] G. L. Smith, "Model for Delay Faults Based Upon Paths", in Proc. 1985 Intl. Test Conf., pp. 342-349.

[2] I. Pomeranz and S. M. Reddy, "An Efficient Non-Enumerative Method to Estimate Path Delay Fault Coverage", in Proc. Intl. Conf. on Computer-Aided Design, 1992, pp. 560-567.

[3] W.-N. Li, S. M. Reddy and S. K. Sahni, "On path Selection in Combinational Logic Circuits", IEEE Trans. on Computer-Aided Design, Jan. 1989, pp. 56-63.

[4] L. N. Reddy, "Compact Test Sets for Digital Logic Circuits", Ph.D. Thesis, ECE Dept., University of Iowa, August 1992.

[5] S. Bose, P. Agrawal and V. D. Agrawal, "Generation of Compact Delay Tests by Multiple Path Activation", in Proc. 1993 Intl. Test Conf., Oct. 1993, pp. 714-723.

[6] J. Saxena and D. K. Pradhan, "A Method to Derive Compact Test Sets for Path Delay Faults in Combinational Circuits", in Proc. 1993 Intl. Test Conf., Oct. 1993, pp. 724-733.

[7] M. H. Schultz, K. Fuchs and F. Fink, "Advanced Automatic Test Pattern Generation Techniques for Path Delay Faults", in Proc. Intl. Symp. on Fault-Tolerant Computing, June 1989, pp. 44-51.

[8] P. Goel and B. C. Rosales, "Test Generation & Dynamic Compaction of Tests", in Digest of Papers 1979 Test Conf., Oct. 1979, pp. 189-192.

[9] K. Fuchs, F. Fink and M. H. Schulz, "DYNAMITE: An Efficient Automatic Test Pattern Generation for Path Delay Faults", IEEE Trans. on Computer-Aided Design, Oct. 1991, pp. 1323-1335.

[10] K.-T. Cheng and H.-C. Chen, "Delay Testing for Non-robust Untestable Circuits", in Proc. Intl. Test Conf., Oct. 1993, pp. 954-961.

[11] U. Sparmann, D. Luxenburger, K.-T. Cheng, and S. M. Reddy, "Fast Identification of Robust Dependent Path Delay Faults", in Proc. Design Autom. Conf., June 1995, pp. 119-125.

[12] S. Kajihara, K. Kinoshita, I. Pomeranz and S. M. Reddy, "A Method for Identifying Robust Dependent and Functionally Unsensitizable Paths", in Proc. 1997 VLSI Design Conf., Jan. 1997, pp. 82-87.

[13] I. Pomeranz and S. M. Reddy, "On Synthesis-for-Testability of Combinational Logic Circuits", in Proc. 32nd Design Autom. Conf., June 1995, pp. 126-132.