Design-for-Testability for Path Delay Faults in Large Combinational Circuits Using Test-Points

Irith Pomeranz and Sudhakar M. Reddy +
Electrical and Computer Engineering Department
University of Iowa
Iowa City, IA 52242

Abstract

We present a method for test-point insertion in large combinational circuits, to increase their path delay fault testability. Using an appropriate test application scheme with multiple clock periods, a test-point on a line *g* divides the set of paths through *g* for testing purposes into a subset of paths from the primary inputs up to *g*, and a subset of paths from *g* to the primary outputs. Each one of these subsets can be tested separately. The number of paths that need to be tested directly is thus reduced. Test-point insertion is done to reduce the number of paths, using a time-efficient procedure. Indirectly, it also reduces the number of tests and renders untestable paths testable. Experimental results are presented to demonstrate the effectiveness of the method proposed in increasing the testability of large benchmark circuits, and to demonstrate the overheads involved.

1. Introduction

The path delay fault model was proposed to model defects that change the timing behavior of a circuit [1,2]. It is the most general of all delay fault models, since it models distributed as well as localized excessive delays. However, three problems are associated with this fault model, that prevent test generation procedures from achieving complete or close-to-complete fault coverage. (1) The number of paths (and therefore the number of path delay faults) in practical circuits may be very large [3]. (2) The number of tests to detect all path delay faults may be very large [4]. (3) Many path delay faults in practical circuits are not testable [5].

The problem of handling large numbers of paths was alleviated in part by the non-enumerative methods of [3,6], however, even using these techniques, the fault coverage for large circuits is still very low. This is due in part to the large number of tests to detect all faults, and in part to the fact that many of the faults are untestable. Another method of avoiding the need to test large numbers of path delay faults was presented in [7], where it was shown that some path delay faults do not have to be tested, as correct speed of operation can be guaranteed by testing other faults. However, even when using this approach, the fault coverage obtained is sometimes low, and lower than that achieved by the method proposed here. In addition, the approach of [7] is valid only to verify the speed of operation of the circuit, and it is not capable of verifying that the manufacturing process is free of errors, due to the fact that it omits some of the paths from consideration. The method proposed here achieves this goal by testing all (or almost all) path delay faults.

Previously proposed methods of synthesis-for-testability and design-for-testability for path delay faults (e.g., [8-12]) rely on a two-level description of the circuit or on the use of Binary Decision Diagrams (BDDs), or investigate the testability of individual paths and modify the circuit to render them testable. Thus, they may not be applicable to large circuits. In addition, for large circuits with compact BDD or two-level representations, these methods may create a circuit with a large number of paths, where large numbers of tests are required, leaving the circuit practically untestable.

In this work, we present a design-for-testability method to increase the testability of a circuit to path delay faults, that is suitable for large circuits. The method achieves fault coverage figures of over 90% for large circuits when pseudo-random patterns are used, and 100% fault coverage by deterministic patterns, with overheads as explained below. It is the first method capable of giving such high fault coverages of path delay faults in large circuits. In addition, it provides a new view of path delay fault testing, which significantly simplifies the handling of this model.

The method proposed consists of placing test-points [13] in the circuit. The test-point locations are selected based on the number of paths in the circuit. Alternatively, a lower bound on the number of tests to detect all path delay faults can be used. The goal of test-point insertion is to reduce these two parameters. Indirectly, it also renders untestable faults testable. The main advantage of these parameters when used as testability measures is that they can be computed efficiently [3,4]. It is possible to use test generation to identify sites for test-point insertion, as done for stuck-at faults in [14,15]. However, we have not pursued this approach in this work, mainly due to the high computational cost of such an approach, especially for circuits with large numbers of path delay faults.

A test-point placed on a line *g* is assumed to make *g* both controllable and observable [13]. It thus divides the set of paths through *g* for testing purposes into a subset of paths from the primary inputs up to *g*, and a subset of paths from *g* to the primary outputs. Each one of these subsets can be tested separately, and subpaths common to several paths in the original circuit are tested only once. This observation is the main strength of the method, and it is explained in detail in Section 3. It results in dramatic reductions in the number of paths that need to be targeted explicitly and in the number of tests required to test them. At the same time, it also increases the testability of the original paths, by allowing the test generation objectives for different subpaths of the same original path to be satisfied separately, by different tests. Without the observation that test-points divide the circuit paths for testing purposes, even large numbers of test-points do not achieve high fault coverage in [6].

The overheads involved in the proposed method are as follows. There are area and speed overheads that result from the addition of test-points. We show that the numbers of test-points required to increase the deterministic fault coverage for ISCAS-

+ Research supported in part by NSF Grant No. MIP-9220549 and by NSF Grant No. MIP-9357581

85 benchmark circuits to 100%, or the random pattern fault coverage to above 90%, are relatively small, considering that the fault coverage when no test-points are used is very low for most circuits, and that it has to be significantly improved to achieve over 90% fault coverage. This is different from the situation for stuck-at faults, where the initial fault coverage is typically over 95% even when random patterns are used, and the number of test points required is much smaller. There is also an overhead related to test application, as follows. When test-points are inserted, some paths in the circuit are divided into subpaths, which are tested separately through the test-points. In a fault-free circuit, the delays of these subpaths add up to a normal circuit delay. Thus, to test the circuit, these subpaths have to be tested with clocks having shorter periods, that add up to a normal circuit delay. As a result, a circuit with test-points has to be tested using one *or more* clock periods. We point out that the same requirement for several clock periods exists when testing for gate delay faults, and methods to accomplish testing with multiple clock periods were described in [16,17]. Thus, this requirement is not unusual in the context of delay testing. However, it has not been used before for path delay fault testing.

Only robust tests are considered in this work. However, the test-point placements are valid for non-robust tests as well.

Several variations of the test-point placement method are possible, according to the target testing method and fault coverage, and according to the overheads allowed, as follows. In our experiments, the set of target faults includes all path delay faults in the circuit. However, it is possible to reduce the number of test-points required by targeting a subset of paths. The test-points can also be selected so as to minimize the number of different clock periods required for test application. This number can be further reduced if the set of target faults is smaller. In addition, test-point placement can target deterministic test generation, or random or weighted-random patterns. Our experiments are conducted using deterministic as well as pseudo-random patterns. The latter correspond to the simplest test application scheme, and the one that requires the highest overhead to make a circuit fully testable. Deterministic test pattern generation reduces the number of test-points required to achieve the same fault coverage, and thus reduces the overheads involved in the proposed method. We present experimental results that support this observation.

The paper is organized as follows. In Section 2, we give a more detailed description of the problems involved in path delay fault testing. In Section 3, we describe the effects of test-points on the testing of path delay faults. In Section 4, we describe the two parameters we use for guiding the placement of test-points, and we present the test-point placement procedure. In Section 5, we present experimental results for ISCAS-85 benchmark circuits. Section 6 concludes the paper.

2. Preliminaries

To demonstrate the problems arising in path delay fault testing, we give in Table 1 the number of path delay faults and a lower bound on the number of deterministic tests to detect all of them for ISCAS-85 benchmark circuits. The results are taken from [4]. For comparison, we also give the number of tests to detect all stuck-at faults from [21]. Table 1 shows that extremely large numbers of path delay faults need to be targeted, and that the number of tests to detect them, which is at least as large as the lower bound, is sometimes extremely large too. Large numbers of tests result in impractical test storage and test application times for deterministic patterns, potentially preventing thorough testing of path delay faults. In addition, test generation times can be impractically large.

Table 1: The numbers of faults and tests

circuit	path-delay faults	lower bound	stuck-at tests[21]
c880	17,284	252	30
c1355	8,346,432	2,610	86
c1908	1,458,114	2,886	115
c2670	1,359,768	3,480	67
c3540	57,353,342	76,541	115
c5315	2,682,610	5,269	56
c6288	197.886E18	892,987,520,786	16
c7552	1,452,986	4,339	87

The problem of large numbers of tests is even more acute when random pattern testing is done. The simplicity of on-chip generation of pseudo-random patterns makes it a very attractive test application scheme, however, the number of patterns required to detect all path delay faults may make it ineffective. Thus, one of the purposes of test-point placement is to allow high fault coverage to be achieved using reasonable numbers of test patterns.

Another problem involved in path delay fault testing is the large numbers of untestable faults. For example, the fault coverage achieved in [6] for ISCAS-85 benchmark circuit *c*6288 was 8.3E-19. After modifying the circuit to increase its testability, the fault coverage was 2.9E-11, still extremely low. We point out that these results are some of the best available.

3. The effects of test-points on path delay fault testing

The introduction of test-points has several effects on the testing of path delay faults. These effects are considered in this section. Unless stated otherwise, we assume that a test-point on line *g* makes *g* both controllable and observable. In other words, *g* can be regarded as a primary output driven by the logic that drives *g* in the original circuit, and as a primary input driving the logic driven by *g* in the original circuit. Practical implementations of test-points that support this view can be found in [13]. The new primary output is given a different index in the modified circuit. An example is shown in Figures 1 and 2 (based on ISCAS-85 benchmark *c*17), where a test-point is placed on line 9. The new primary output index in Figure 2 is 18, and the new primary input is given the original line index 9.

To study the effect of a test-point on the number of paths that need to be tested explicitly, consider *c*17 shown in Figure 1. There are 11 paths in this circuit. Six of these paths go through line 9, namely, the paths

3-5-9-10-12-13-16	6-9-10-12-13-16
3-5-9-10-12-14-17	6-9-10-12-14-17
3-5-9-11-15-17	6-9-11-15-17

(paths are described by the indices of the lines they go through, using the line indices given in Figure 1). Let us now place a test-point on line 9. For testing purposes, the circuit changes as shown in Figure 2. In the modified circuit, we have two paths leading to line 18, and three paths starting at line 9. The paths are

3-5-18 9-10-12-13-16

6-18 9-10-12-14-17
9-11-15-17

In the modified circuit, we have only five paths through lines 9 and 18, instead of six paths through line 9 in the original circuit.

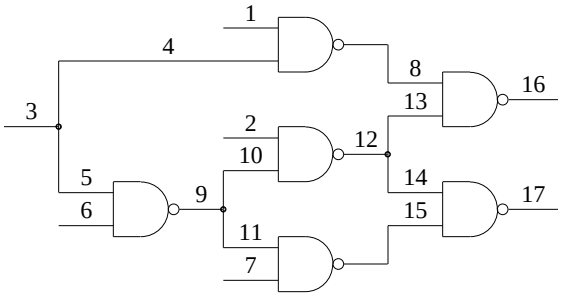

Figure 1: The circuit *c* 17

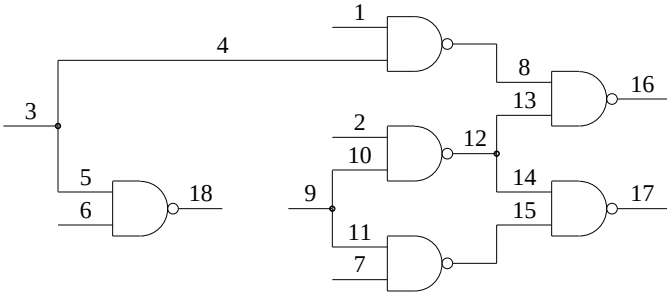

Figure 2: *c* 17 **with a test-point on line 9**

To see the effect of placing a test-point on a line *g*, let us denote the number of paths from the primary inputs to *g* by $N_{p,PI}(g)$ and let us denote the number of paths from *g* to the primary outputs by $N_{p,PO}(g)$. In the original circuit, there are $N_{p,PI}(g)\cdot N_{p,PO}(g)$ paths through line *g* (e.g., for *c* 17, $N_{p,PI}(9) = 2$, $N_{p,PO}(9) = 3$ and $N_{p,PI}(9)\cdot N_{p,PO}(9) = 6$). In the modified circuit, there are $N_{p,PI}(g)+N_{p,PO}(g)$ paths through the primary input and primary output corresponding to line *g*, as follows. There are $N_{p,PI}(g)$ paths from the primary inputs to the primary output corresponding to *g*, and there are $N_{p,PO}(g)$ paths starting from the primary input corresponding to *g* and going to the primary outputs. Since these paths are not connected any more, the total number of paths that have to be tested is simply the sum. The total number of paths in the circuit reduces by $N_{p,PI}(g)\cdot N_{p,PO}(g)-(N_{p,PI}(g)+N_{p,PO}(g))$ when a test-point is placed on *g*. Thus, the number of paths in the circuit, and hence the number of path delay faults, can be dramatically reduced by placing test-points, from products that can yield exponential numbers of paths, to sums.

When an additional test-point is inserted on a line *f*, the number of paths reduces by $N_{p,PI}(f)\cdot N_{p,PO}(f)-(N_{p,PI}(f)+N_{p,PO}(f))$. Here, the number of paths is computed not in the original circuit, but in the circuit after *g* has already been replaced by appropriate primary input and primary output. The number of paths in a circuit in the presence of any number of test-points can be computed by first replacing every test-point by a primary input and a primary output, and then counting the number of paths using the procedure from [3].

Let us now consider the test application requirements in a circuit with test-points. For purposes of illustration, we associate with every line in the circuit a unit delay and we associate with every path a delay that equals the number of lines along the path. We also assume that a test-point adds one unit of delay. We chose this delay model for its simplicity, however, any other delay model can be accommodated. Let us consider the path 3-5-9-10-12-13-16 in *c* 17 (cf. Figure 1). This is one of the longest paths in the original circuit, and therefore the original circuit has to be operated with a clock period of at least seven units (which is the number of lines along the longest paths). The presence of a test-point on line 9 adds one unit of delay to the path 3-5-9-10-12-13-16, and the circuit is now operated with a clock period of eight units. Testing is done using the modified circuit (cf. Figure 2). In Figure 2, the path we consider is divided into two paths, 3-5-9 and 9-10-12-13-16. For the original circuit to operate correctly, the delay of the path 3-5-9 must not exceed three units, and the delay of the path 9-10-12-13-16 must not exceed five units. This will ensure that in the complete circuit, the delay along the complete path does not exceed eight units. Thus, we need two additional clock periods during testing.

In general, the number of different clock periods required for testing are determined using the original circuit, where the appropriate lines are marked as test-point locations (and not in the modified circuit, where the test-points are replaced by primary inputs and primary outputs). The various clock periods are determined as follows.

(1) For every test-point *g*, we find the maximum delay from the original primary inputs to *g*, denoted by $\delta_{PI}(g)$. All the paths from the original primary inputs to the test-point *g* must have a delay which is at most equal to $\delta_{PI}(g)$. The set $\{\delta_{PI}(g)$: *g is a test-point*$\}$ is the first component of the set of clock periods required. Note that if there is a path from a primary input to *g*, which is shorter than $\delta_{PI}(g)$, then as long as its delay does not exceed $\delta_{PI}(g)$, it is not considered as a delay fault, and we do not require a special clock period for testing it. This is similar to the conventional situation where an original circuit is tested with a single clock period, equal to the delay of the longest path, although it may have paths of different lengths.

(2) For every two test-points g_1 and g_2, if there is a path from g_1 to g_2, we find the maximum delay of such a path, and denote it by $\delta(g_1,g_2)$. The set $\{\delta(g_1,g_2)$: g_1,g_2 *are test-points*$\}$ is the second component of the set of clock periods required. Again, as long as every path between g_1 and g_2 operates within the delay of the longest path, the circuit is considered free of fault.

(3) For every test-point *g*, we find the maximum delay from *g* to the original primary outputs, denoted by $\delta_{PO}(g)$. All the paths from *g* to the original primary outputs must have a delay which is at most equal to $\delta_{PO}(g)$. The set $\{\delta_{PO}(g)$: *g is a test-point*$\}$ is the third component of the set of clock periods required.

(4) In addition, the original circuit clock period is required to test the paths that go from the original primary inputs to the original primary outputs (if such paths remain).

We point out that it may sometimes be possible to allow the delay of some subpaths to exceed the delay set by their length, if they are not part of maximum delay paths, and thus reduce the number of clock periods required. We do not explore this possibility here.

The procedure for computing the required clock periods is summarized below. The procedure has two labeling phases. In the first one, the circuit is labeled starting from the primary inputs, in order to find the delays from the primary inputs to every test-point. In the second phase, the circuit is labeled starting from every test-point separately, in order to find the delay to

all other test-points and to the primary outputs. In all the procedures given in this paper, we assume that the lines in the circuit are given increasing indices from inputs to outputs, and that fanout branches are given their own indices, different from their stem index. We also assume that the first line-index is 1 and that the number of lines in the circuit (and the last line-index) is L.

Procedure 1: Finding the required clock periods

(1) Set $C = \phi$ (C is the set of required clock periods).

Phase 1:

(2) Assign to every primary input the label $\delta = 0$. Assign to every other line the label $\delta = -1$ (-1 stands for an undefined label). Set $i = 1$.

(3) If i is a gate with inputs $i_1, i_2, \cdots, i_k$, labeled $\delta(i_1), \delta(i_2), \cdots, \delta(i_k)$, respectively, then find the maximum label of any input of i which is not a test-point, i.e., compute $\delta = \max\{\delta(i_j): i_j \text{ is not a test-point}\}$. If $\delta \geq 0$, assign to line i the label δ+1 (excluding inputs i_j which are test-points ensures that labeling stops at the test-points and subpaths through multiple test-points are not considered).

(4) If i is a fanout branch with a stem g labeled $\delta(g) \geq 0$, and if g is not a test-point, label line i by $\delta(g)$+1.

(5) Set $i = i+1$. If $i \leq L$, go to Step 3.

(6) For every test-point g, if $\delta(g) > 0$, set $C = C \bigcup \{\delta(g)\}$.

(7) Find the maximum label, δ, assigned to any primary output. If $\delta > 0$, set $C = C \bigcup \{\delta\}$.

Phase 2:

(8) For every test-point g:

  (a) Assign to g the label $\delta = 0$. Assign to every other line the label $\delta = -1$. Set $i = g+1$.

  (b) Execute Steps 3 and 4.

  (c) Set $i = i+1$. If $i \leq L$, go to Step 8b.

  (d) For every test-point g, if $\delta(g) > 0$, set $C = C \bigcup \{\delta(g)\}$.

  (e) Execute Step 7.

We stress again that the use of multiple clock periods for testing of delay faults is also used for gate delay fault testing [16,17].

In addition to reducing the number of paths that need to be tested and hence the number of tests to detect all path delay faults, another advantage of test-point placement is that it allows all parts of a path to be tested, when the complete path is untestable. The circuit of Figure 3 demonstrates this point. Consider the path delay fault with the 0→1 transition at the source of the path 1-6-8-9-10. This fault is untestable, since it requires that line 3 (and consequently line 2) would have a final value of 1 in order to propagate the 0→1 transition from line 1 to line 6, and it requires that line 4 (and consequently line 2) be set to a stable 0 to propagate the 0→1 transition from line 9 to line 10. Now consider the two subpaths formed if a test-point is placed on line 8, 1-6-8 and 8-9-10. Both of these subpaths are testable. The first is testable by the test that assigns a 0→1 transition to line 1, a stable 1 to line 2 and a stable 0 to line 5. The second subpath is testable by the test that assigns a 0→1 transition to line 8, a stable 1 to line 7 and a stable 0 to line 2. Thus, the complete path can be tested by testing each one of its subpaths using an appropriate clock period.

Finally, we point out that it is not necessary to make the test-points controllable. If only observation points are added, a transition on a test-point has to be launched by the circuit driving it. However, to take advantage of the reduction in the number of faults that have to be targeted explicitly, it would be necessary in such a case to measure propagation delays from the time a transition reaches a test-point until it reaches the primary outputs. This may be difficult to accomplish in practice, and we therefore prefer to use both controllable and observable test-points.

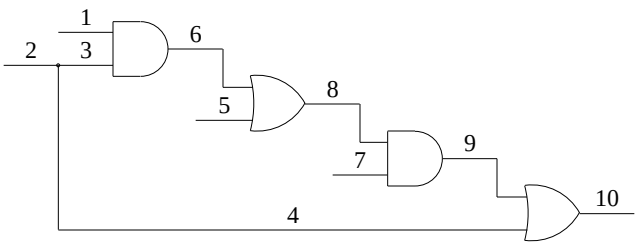

Figure 3: An example of the effect on testability

4. Test-point placement

Two heuristics for test-point placement are described in this section. Based on these heuristics, a greedy test-point placement procedure is presented. The advantage of these heuristics is that they are easy to compute, and are effective in increasing the testability of the circuit. They avoid the need for test generation as in test-point placement for stuck-at faults [14,15], which would have made the test-point placement process highly computation-intensive. Non-greedy approaches can be taken to test-point placement, however, they would be more complex, and are not investigated in this work.

The first heuristic for test-point placement is based on the number of paths in the circuit, and is aimed at reducing this number. The motivation for the use of this heuristic is that the number of paths in the modified circuit is also the number of paths that have to be explicitly targeted during test generation, in order to test all the paths in the original circuit. The smaller the number of such paths, the easier it is to test the circuit. Indirectly, paths which are untestable in the original circuit may have testable subpaths in the modified circuit (cf. Figure 3). Thus, cutting the paths by test-points may increase their testability. We showed in Section 2 that when a test-point is placed on line g, the number of paths through g is reduced from $N_{p,PI}(g) \cdot N_{p,PO}(g)$ to $N_{p,PI}(g) + N_{p,PO}(g)$, and the total number of paths is reduced accordingly. To compute $N_{p,PO}(g)$, we use the procedure from [3], that labels each line i with the number of paths from i to the primary outputs. The procedure starts by assigning to every primary output the label 1 (there is one path from a primary output to the primary outputs, that includes only the primary output itself). It then proceeds from outputs to inputs, assigning appropriate labels to all the lines. The procedure is repeated below for completeness.

Procedure 2 [3]: Computing the number of paths to the primary outputs

(1) Assign all the primary outputs the label $N_{p,PO} = 1$. Set $i = L$.

(2) If line i is the input of a gate with output g, and the label of line g is $N_{p,PO}(g)$, label line i by $N_{p,PO}(g)$.

(3) If line i is a fanout stem with branches $i_1, i_2, \cdots, i_k$, labeled $N_{p,PO}(i_1)$, $N_{p,PO}(i_2)$, $\cdots$, $N_{p,PO}(i_k)$, respectively, label line i by $\sum_{j=1} N_{p,PO}(i_j)$.

(4) Set $i = i-1$. If $i > 0$, go to Step 2.

To compute $N_{p,PI}(g)$ for every line g, we use a similar procedure, except that now, labeling is done from inputs to outputs. As a result, we obtain for every line i the number of paths from the primary inputs to i. The procedure is given next.

Procedure 3: Computing the number of paths from the primary inputs

(1) Assign all the primary inputs the label $N_{p,PI} = 1$. Set $i = 1$

(2) If line i is the output of a gate with inputs $i_1, i_2, \cdots, i_k$, labeled $N_{p,PI}(i_1)$, $N_{p,PI}(i_2)$, $\cdots$, $N_{p,PI}(i_k)$, respectively, then label line i by $\sum_{j=1} N_{p,PI}(i_j)$.

(3) If line i is a fanout branch of a stem g, labeled $N_{p,PI}(g)$, label line i by $N_{p,PI}(g)$.

(4) Set $i = i+1$. If $i \leq L$, go to Step 2.

The test-point placement procedure we implemented is based on the number of path criterion. It is a greedy procedure, i.e., it selects one test-point at a time. The line that maximally reduces the number of paths is selected at every iteration. The procedure is given next.

Procedure 4: Test-point placement

(1) Set $N_{tp} = 0$ (N_{tp} is the number of test-points).

(2) For every line g in the circuit, compute $N_{p,PO}(g)$ and $N_{p,PI}(g)$, using Procedures 2 and 3.

(3) Select the line g for which $N_{p,PI}(g) \cdot N_{p,PO}(g) - (N_{p,PI}(g) + N_{p,PO}(g))$ is maximum, and place a test-point on g.

(4) Set $N_{tp} = N_{tp}+1$. If the number of test-points does not exceed a predetermined limit, go to Step 2.

The number of test-points to be actually placed in a circuit is determined according to the fault coverage and test set size goals, as illustrated in the following section.

Another criterion for selecting the test-point locations is the number of tests to detect all path delay faults. It is based on a labeling procedure, similar to Procedure 3, that computes a lower bound on the number of tests. The procedure is given in [4], and it is omitted here for space considerations. We refer to it as Procedure 5. The basic idea behind Procedure 5 is the following. Consider an AND gate. In the best case, that results in a minimum number of tests, all the path delay faults that require $0 \rightarrow 1$ transitions on the inputs of the gate can be tested simultaneously, by the same test. Path delay faults that require $1 \rightarrow 0$ transitions on different inputs must always be tested by separate tests, since robust propagation of transitions does not allow two $1 \rightarrow 0$ transitions to be propagated simultaneously from the inputs of an AND gate to its output. Similar arguments exist for all other gate types. We use the following notation. The label $N_{1x0}(i)$ is a lower bound on the number of tests for the cone of line i, that bring the $1 \rightarrow 0$ transition to line i, and $N_{0x1}(i)$ is a lower bound on the number of tests that bring the $0 \rightarrow 1$ transition to line i. If line i is the output of a k-input AND gate with inputs $i_1, i_2, \cdots, i_k$, labeled $(N_{1x0}(i_1), N_{0x1}(i_1))$, $(N_{1x0}(i_2), N_{0x1}(i_2))$, $\cdots$, $(N_{1x0}(i_k), N_{0x1}(i_k))$, then the output i of the gate is labeled by

$$(\sum_j N_{1x0}(i_j), \max_j \{N_{0x1}(i_j)\})$$

To select test-points, we use the following heuristic to evaluate the effect on the number of tests of placing a test-point on line g. A test-point on line g changes its label from $(N_{0x1}(g), N_{1x0}(g))$ to (1,1). For this change to have an effect on the lower bound, g has to be in the cone of the output on which the highest lower bound is obtained. We select test-points according to the maximum value of $N_{0x1}(g)+N_{1x0}(g)$. However, in contrast to the number of paths criterion, in this case, it is not guaranteed that the number of tests will reduce by $N_{0x1}(g)+N_{1x0}(g)-2$. Therefore, after placing the test-point, we check whether the lower bound on the number of tests was indeed reduced. If it did not reduce, we reverse the decision and remove the test-point from g.

5. Experimental results

The circuits considered are ISCAS-85 benchmark circuits. We first eliminated untestable single stuck-at faults using the redundancy removal procedure from [19]. Then, we applied to the irredundant circuits a set of circuit modifications proposed in [4], that transform the circuit into an equivalent circuit with a reduced number of paths and a reduced lower bound on the number of tests. These modifications were applied to the circuits before test-point placement was attempted. The reason to eliminate redundant single stuck-at faults is that if a line in the circuit is redundant (has a redundant single stuck-at fault on it), path delay faults passing through it are also untestable. Thus, at least one test-point would be required to make the stuck-at fault testable, before the path delay faults become testable. To determine in a controlled way the cost of making a circuit fully testable to path delay faults, no untestable stuck-at faults should exist. The modifications of [4] to reduce the number of paths and the lower bound on the number of tests were introduced since they have no overheads, and should therefore be attempted before the overheads of test-points are incurred. They were effective for $c1355$, $c2670$, $c5315$ $c6288$ and $c7552$. They did not affect $c880$ and $c1908$. Circuits $c432$ and $c499$ were omitted from the experiment since they include XOR gates, making the results implementation-dependent, and $c17$ was omitted because of its small size.

We applied Procedure 4 to the modified circuits, to select test-point locations. The list of test-points produced by Procedure 4 was then used as an ordered list from which test-point locations were actually selected, for different fault-coverage levels. Test-point placement times for 1,000 test-points are given in Table 2. Time is measured in seconds on a SUN SPARC2 workstation.

Table 2: Test-point placement time [sec]

circuit	time	circuit	time
c880	26	c3540	123
c1355	53	c5315	200
c1908	66	c6288	259
c2670	75	c7552	265

The purpose of the first experiment we conducted was to exhibit the effect of test-points on the fault coverage by random patterns and by deterministic patterns. To check the effect on random pattern testability, we simulated fixed numbers of random tests applied to a given circuit with different numbers of test-points inserted. The number of random tests applied was either 100,000 or 200,000, according to the number of tests required before the fault coverage saturated for the circuit with no test-points. To produce K random two-pattern tests, we produced $K+1$ input patterns, and applied every consecutive pair. For example, for a three-input circuit and $K = 4$, we may produce the patterns {001, 111, 010, 011, 001}, and apply the two-pattern tests <001,111>, <111,010>, <010,011> and <011,001>.

Deterministic test generation was performed using the procedure from [18]. For a given circuit C, we used 0, 10, 20, 30, $\cdots$ test-points. Here, we report only the results (1) when no test-points are inserted; (2) for the smallest number of test-points for which the test generation procedure from [18] is applicable (the procedure from [18] is not applicable if the number of paths is large); (3) when 100% deterministic fault coverage is achieved; and (4) when 100% fault coverage by random coverage is achieved, but only if the number of test-points for this purpose does not exceed 10% of the number of lines in the original circuit. The results are reported in Table 3, as follows. There is a row for every number of test-points inserted. Each row is organized as follows. After the number of test-points, we give the number of clock periods required to test all path delay faults, computed by Procedure 1. The number of faults is given next. The following two columns give the number of detected faults and the fault coverage by random patterns. The last two columns give the number of detected faults and the fault coverage by deterministic patterns. For example, for c1355, the test generation procedure from [18] was first applicable after 4 test-points were inserted, and the fault coverage was 57.3%; c1908 (before redundancy removal and the modifications of [4]) has 1908 lines, and we inserted up to 190 test-points. The deterministic fault coverage for 70 test-points was complete. It can be seen that even small numbers of test-points give a significant increase in fault coverage. In addition, 100% fault coverage can be achieved with reasonable numbers of test-points in most cases. For c880 and c1355, we also looked for the exact number of test-points that resulted in complete fault coverage by deterministic patterns. For c880, this happened with 7 test-points, and for c1355, 10 test-points were needed.

Comparing the numbers of test-points in Table 3 to the numbers of test-points required to detect all stuck-at faults, the numbers may seem high. However, it has to be noted that the initial coverage of path delay faults is very low, compared to the initial coverage of stuck-at faults, which is typically over 95% even when random patterns are used.

In the experiment above, we ignored the need to apply each test pattern with different clock periods, and concentrated on the fault coverage achievable. In practice, when a test pattern is applied, different outputs (primary outputs or test-points) may have to be observed at different times, using different clock periods. Since the use of multiple clock periods in parallel may be too expensive, we also performed the following experiment. We applied the same number of patterns applied before, however, this time, we computed the fault coverage separately for every clock period. This experiment allows clock periods that cover a small number of faults to be discarded, if the overhead of having another clock period does not justify the additional fault coverage. We performed this experiment for several of the modified circuits above. The results are reported in Table 4, in the following way. For every clock period that is useful in detecting any faults, we report the number of faults detected and the contribution to the fault coverage. *orig* stands for the normal clock period (to test paths from the original primary inputs to the original primary outputs). It can be seen that, in most cases, there are many clock periods that detect very few faults. For example, for c1355, clock periods 2 and 3 cover only 0.6% and 0.4% of the faults, respectively. At the cost of reducing the fault coverage by 1%, these clock periods can be omitted.

6. Concluding remarks

We described a method of test-point insertion for large combinational circuits, to increase their path delay fault testability. The method was based on a test application scheme that uses multiple clock periods, to allow paths in the circuit to be tested in two or more parts, thus reducing the number of paths. Test-point insertion was directly aimed at reducing the number of paths in the circuit. As a by-product, the number of tests was reduced and the testability of the circuit increased. We showed that complete fault coverage by deterministic test generation, and above 90% fault coverage using pseudo-random patterns, are achievable by this method.

In our experiments, the set of target faults included all path delay faults in the circuit. The number of test-points required to achieve a given testability level can potentially be reduced by targeting smaller subsets of paths. Such reductions are currently under investigation.

References

[1] J. D. Lesser and J. J. Schedletsky, "An experimental delay test generator for LSI logic," IEEE Trans. Comput., vol. C-29, pp. 235-248, Mar. 1980.

[2] Y. K. Malaiya and R. Narayanaswamy, "Testing for timing faults in synchronous sequential integrated circuits," in Proc. Int. Test Conf., pp. 560-571, Oct. 1983.

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

[4] I. Pomeranz and S. M. Reddy, "On the Number of Tests to Detect All Path Delay Faults in Combinational Logic Circuits", Technical Report No. 12-1-1992, ECE Dept., U. of Iowa.

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

[6] I. Pomeranz, S. M. Reddy and P. Uppaluri, "NEST: A Non-Enumerative Test Generation Method for Path Delay Faults in Combinational Circuits", in Proc. 30th Design Autom. Conf, 1993, pp. 439-445.

[7] W. K. Lam, A. Saldanha, R. K. Brayton, A. L. Sangiovanni-Vincentelli, "Delay Fault Coverage and Performance Tradeoffs", in Proc. 30th Design Autom. Conf., 1993, pp. 446-451.

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

[9] N. K. Jha and S. Kundu, *Testing and Reliable Design of CMOS Circuits, Kluwer Academic Publishers*, Norwell, MA, 1990.

[10] A. K. Pramanick and S. M. Reddy, "On the design of path delay fault testable combinational circuits," in Proc. Int. Symp. on Fault-Tolerant Computing, Newcastle-upon-Tyne, pp. 374-381, June 1990.

[11] S. Devadas and K. Keutzer, "Synthesis and optimization procedures for robustly delay-fault testable combinational logic circuits," in Proc. Design Automation Conf., pp. 221-227, June 1990.

[12] P. Ashar, S. Devadas, and K. Keutzer, "Testability properties of multilevel logic networks derived from binary decision diagrams," in Proc. Santa Cruz Conf. on Advanced Research in VLSI, Apr. 1991.

[13] M. Abramovici, M. A. Breuer and A. D. Friedman, *Digital Systems Testing and Testable Design*, Computer Science Press, 1990.

[14] I. Pomeranz and Z. Kohavi, "Limited Exponential Algorithms for Increasing the Testability of Digital Circuits by Testing-Module Insertion", IEEE Transactions on Computer Aided Design, vol. 11, no. 2, February 1992, pp. 247-259.

[15] V. Chickermane, E. M. Rudnick, P. Banerjee and J. H. Patel, "Non-Scan Design-for-Testability Techniques for Sequential Circuits", in Proc. 30th Design Autom. Conf., 1993, pp. 236-241.

[16] W.-W. Mao and M. D. Ciletti, "A Variable Observation Time Method for Testing Delay Faults", in Proc. 27th Design Autom. Conf., 1990, pp. 728-731.

[17] V. S. Iyengar and G. Vijayan, "Optimized Test Application Timing for AC Test", IEEE Trans. on Computers, Nov. 1992, pp. 1439-1449.

[18] S. Patil and S. M. Reddy, "A Test Generation System for Path Delay Faults," Intl. Conf. on Computer Design, 1989, pp. 40-43.

[19] S. Kajihara, H. Shiba and K. Kinoshita, "Removal of Redundancy in Logic Circuits under Classification of undetectable Faults", Proc. 22nd Fault-Tolerant Computing Symp., July 1992, pp. 263-270.

[20] F. Brglez and H. Fujiwara, "A Neutral Netlist of 10 Combinational Benchmark Designs and a Special Translator in Fortran," International Symposium on Circuits and Systems, June 1985, pp. 663-698.

[21] I. Pomeranz L.N. Reddy and S.M. Reddy, "COMPACTEST: A Method to Generate Compact Test Sets for Combinational Circuits", 1991 Intl. Test Conf., Oct. 1991, pp. 194-203.

Table 3: Results for random and deterministic patterns
(a) *c* 880 **(100,000 random patterns)**

			random		deterministic	
tps	clks	faults	detect	f.c.	detect	f.c.
0	1	17284	1473	8.5%	15454	89.4%
7	17	3414	2101	61.5%	3414	100.0%
70	8	884	884	100.0%		

(b) *c* 1355 **(200,000 random patterns)**

			random		deterministic	
tps	clks	faults	detect	f.c.	detect	f.c.
0	1	644224	161	0.02%	NA	NA
4	6	154348	855	0.6%	88396	57.3%
10	5	5680	2187	38.5%	5680	100.0%
20	9	1988	1988	100.0%		

(c) *c* 1908 **(200,000 random patterns)**

			random		deterministic	
tps	clks	faults	detect	f.c.	detect	f.c.
0	1	1458050	2482	0.17%	NA	NA
5	7	120702	2800	2.3%	62329	51.6%
70	13	1962	1885	96.1%	1962	100.0%

(d) *c* 2670 **(200,000 random patterns)**

			random		deterministic	
tps	clks	faults	detect	f.c.	detect	f.c.
0	1	34358	3024	8.8%	14227	41.4%
120	13	1880	1522	81.0%	1880	100.0%

Table 3 (Cntd): Results for random and deterministic patterns
(e) *c* 3540 **(100,000 random patterns)**

			random		deterministic	
tps	clks	faults	detect	f.c.	detect	f.c.
0	1	15111450	3360	0.02%	NA	NA
20	24	96198	6240	6.5%	31635	32.9%
210	13	2874	2452	85.3%	2874	100.0%

(f) *c* 5315 **(100,000 random patterns)**

			random		deterministic	
tps	clks	faults	detect	f.c.	detect	f.c.
0	1	2506220	8172	0.33%	NA	NA
10	11	117120	7626	6.5%	59499	50.8%
320	10	4904	4867	99.3%	4904	100.0%

(g) *c* 6288 **(100,000 random patterns)**

			random		deterministic	
tps	clks	faults	detect	f.c.	detect	f.c.
0	1	9.418E15	158	0.00%	NA	NA
180	41	95386	21689	22.7%	53888	56.5%
1150	7	6618	6618	100.0%	6618	100.0%

(h) *c* 7552 **(200,000 random patterns)**

			random		deterministic	
tps	clks	faults	detect	f.c.	detect	f.c.
0	1	1310174	6987	0.53%	NA	NA
30	28	93936	9882	10.5%	39049	41.6%
650	11	6460	6451	99.9%	6460	100.0%

Table 4: Clock periods for testing
(a) *c* 880 **with 70 test-points**

period	faults	f.c
orig	188	21.3%
8	112	12.7%
6	44	5.0%
5	98	11.1%
4	100	11.3%
3	182	20.6%
2	150	17.0%
1	10	1.1%

(b) *c* 1355 **with 20 test-points**

period	faults	f.c.
orig	128	6.4%
18	896	45.1%
12	256	12.9%
11	464	23.3%
10	144	7.2%
8	32	1.6%
4	48	2.4%
3	8	0.4%
2	12	0.6%

(c) *c* 6288 **with 1150 test-points**

period	faults	f.c.
orig	4	0.06%
7	30	0.5%
6	1022	15.4%
5	492	7.4%
4	1622	24.5%
3	1128	17.0%
2	2320	35.1%