On Minimizing the Number of Test Points Needed to Achieve Complete Robust Path Delay Fault Testability

Prasanti Uppaluri*
Avant! Corporation
P.O.Box 13665
Research Triangle Park
NC 27709, U.S.A

Uwe Sparmann†
Computer Science Dept.
University of Saarland
66041 Saarbruecken
Germany

Irith Pomeranz‡
ECE Dept.
University of Iowa
Iowa City
IA 52242, U.S.A

Abstract

Recently, Pomeranz and Reddy [7], presented a test point insertion method to improve path delay fault testability in large combinational circuits. A test application scheme was developed that allows test points to be utilized as primary inputs and primary outputs during testing. The placement of test points was guided by the number of paths and was aimed at reducing this number. Indirectly, this approach achieved complete robust path delay fault testability in very low computation times. In this paper, we use their test application scheme, however, we use more exact measures for guiding test point insertion like test generation and *RD* fault identification. Thus, we reduce the number of test points needed to achieve complete testability by ensuring that test points are inserted only on paths associated with path delay faults that are necessary to be tested and that are not robustly testable. Experimental results show that an average reduction of about 70% in the number of test points over the approach of [7] can be obtained.

1 Introduction

Correct operation of logic circuits requires proper logic and timing behavior. Manufacturing defects and random variations in process parameters may cause propagation delays in the circuit to exceed their specification. Such defects are modeled as delay faults. Two delay fault models have been proposed in the literature, namely, the gate delay fault model [1] and the path delay fault model [2]. Gate delay faults model defects that cause gate delays to be outside their specified range. Path delay faults model defects that cause cumulative propagation delays along circuit paths to exceed their specification. The path delay fault model is the more general fault model. It models defects that affect circuit performance better by incorporating the additive nature of delays along the gates in the path. Two types of tests have been proposed: robust and non-robust [3-6].

The major disadvantages of the path delay fault model are the number of faults that need to be targeted and the total number of tests required to test all the faults. These are often very large and in the worst case can be exponential in the size of the circuit. Also, the testability of the circuit under this fault model is sometimes very poor.

The works in [8-20] have targeted one or more of these disadvantages. [8-9] tackled the problem of the prohibitively large size of the fault universe by proposing non-enumerative techniques for considering faults. However, the fault coverages for large circuits are still very low, mainly due to the poor testability of the circuits. [10,11] showed that it is not necessary to test all the path delay faults in the circuit to verify the timing behavior of the circuit. The faults that need not be tested are referred to in [10,12] as *Robust-Dependent Faults* (*RD* faults). However, even after the identification of *RD* faults, in large circuits, the subset of faults which needs to be tested to verify the speed of operation of the circuit (*non-RD* faults) may be very large and some of these faults may not be testable.

Synthesis-for-testability and design-for-testability (DFT) techniques for the path delay fault model were described in [13-20]. The former start with the circuit function and synthesize a testable circuit. The latter modify a given circuit implementation so that it becomes testable. However, under these techniques, a testable circuit may result which has a large fault universe and test set size and, hence, testing of the circuit remains difficult.

Recently, in [7], test point insertion was introduced as a DFT technique for path delay faults. It tackled all the disadvantages of the path delay fault model simultaneously. A test application scheme was presented wherein every path through a test point is partitioned into two parts, that can be tested separately. It thus reduces the number of paths that need to be tested and increases the testability of these paths. A review of the test application scheme is given in Section 2. In [7], complete robust path delay fault testability was achieved with low computation times.

The heuristic for test point insertion in [7] is based on reducing the number of paths and on reducing a

*Work was done while visiting the University of Saarland and partly while at the University of Iowa. Research supported in part by DFG, SFB 124 'VLSI Entwurfsmethoden und Parallelitaet' and in part by NSF Grant No. MIP-9220549.

†Research supported by DFG, SFB 124 'VLSI Entwurfsmethoden und Parallelitaet'.

‡Research supported by NSF Grant No. MIP-9220549.

lower bound on the test set size introduced in [21]. These were used as testability measures. It has been noted in [7] that the number of test points needed may be reduced by targeting a subset of all the path delay faults such as the *non-RD* faults and/or by considering more exact measures of testability such as the number of testable faults by a given test generation procedure. In this work, we explore such a higher computational cost approach based on test generation to guide the insertion of test points. Experimental results indicate that an average reduction of about 70% in the number of test points can be achieved.

The goal of the algorithm presented here is to minimize the number of test points placed to achieve complete robust path delay fault testability. *RD* fault identification is performed as in [12] to determine the subset of faults which need not be tested to check the temporal correctness of the circuit. Among the necessary-to-test faults (*non-RD* faults), we determine the faults that are not robustly testable using test generation. Only these faults are considered for testability modifications. Thus, test points are placed only on paths associated with path delay faults that are not robustly testable and which are necessary to test in order to ascertain the timing of the circuit.

As part of the test point insertion procedure, we have to perform test generation and *RD* fault identification. Test generation for path delay faults is a difficult and computationally intensive task mainly because of the total number of faults that need to be targeted. Even after *RD* fault identification, the subset of faults that needs to be tested is often very large. Also, the *RD* fault identification methods of [10-12] and fast test generators for path delay faults [22,24] cannot handle in a reasonable time circuits with extremely large numbers of paths like c6288 of the ISCAS85 benchmarks which has over 10^{20} paths. In this work, these problems are overcome by a bottom up approach where gradually increasing portions of the circuit are analyzed, as outlined next.

We partition the circuit into subcircuits. Initially, only the paths in every subcircuit are considered. We identify *RD* path delay faults and use test generation to determine the untestable *non-RD* path delay faults local to the subcircuit. There may be more than one way to select a *non-RD* set, however, all these sets have in common the subset of non-robustly testable faults [12]. Test point insertion is performed to make these faults testable. This procedure is iterated starting with small subcircuits. As test points are inserted, the number of path delay faults in the circuit that need to be considered for test generation is reduced [7], its testability is improved and larger subcircuits can be considered. Finally, the analysis can be performed over the entire circuit. In a postprocessing step, we find a *non-RD* set for the complete circuit and, if necessary, insert additional test points to make its paths testable.

The paper is organized as follows. Section 2 gives some definitions and reviews the basic concepts of test point insertion. The bottom up circuit analysis approach to find local untestable path delay faults is described in Section 3. Test point insertion to make the untestable path delay faults testable is explained in Section 4. Section 5 presents the complete algorithm. Experimental results are given in Section 6. Section 7 concludes the paper.

2 Definitions

In this section, we start with several definitions followed by a review of the test point insertion framework from [7].

A directed graph $G = (V, E)$ is used to represent the circuit. The vertices V are the gates and the fanout points in the circuit and the edges E are the interconnections between them. A vertex $v \in V$ can be one of the following types: *nand, nor, and, or, not*, primary input, primary output or fanout point.

A *path, p*, is defined as a sequence of edges $e_1 - e_2 - ... - e_n$ where each e_i connects vertices v_{i-1} and v_i. (Note that v_i is not necessarily incident only to e_i and e_{i+1}.) If v_0 is a primary input and v_n is a primary output, the path is said to be a *complete path.* If v_0 is not a primary input or v_n is not a primary output, the path is said to be a *partial path.*

Consider any edge e_i on a path p, which connects gates v_{i-1} and v_i. All edges, other than e_i, which feed v_i are called *off-path inputs* of p. e_i is called an *on-path input* of p.

There are two delay faults associated with each path p: the rising transition fault and the falling transition fault. The type of the fault is determined by the transition at e_1. In general, two-pattern tests are necessary to detect a delay fault on p. The first pattern (T_1) serves as an initialization pattern. After the circuit has stabilized, the second pattern (T_2) is applied which launches a transition at the primary input of p and propagates it to the primary output of p.

A two-pattern test $< T_1, T_2 >$ for a delay fault is said to be *robust* [5] if and only if the test is valid independent of the delays in the rest of the circuit. A circuit has 100% *robust path delay fault testability* if and only if every path delay fault has a robust test.

Various definitions for a non-robust test exist in the literature. We will use the one from [6]. A two-pattern test $< T_1, T_2 >$ is said to be a non-robust test for a path p if it launches a transition at the primary input of p and all the off-path inputs of p have non-controlling values under T_2. (The *non-controlling* value for an *and, nand (or, nor)* gate is 1(0).)

A test point can be used as a primary input to enhance the controllability and/or as a primary output to improve the observability of embedded parts of a circuit [25]. As in [7], we assume that test points are both controllable and observable. The test application scheme of [7] allows test points to reduce the number of paths that need to be tested as follows. Let P be a set of paths going through an edge e_i. Suppose that a test point is placed on e_i. Then every path $p = e_0 - ... - e_i - ... - e_n$ in P is divided by the test point into two parts, $p_1 = e_0 - ... - e_i$ and $p_2 = e_i - ... - e_n$. p_1 can be tested by propagating a transition from e_0 to e_i along the path and observing the transition on e_i after the propagation delay corresponding to p_1. We denote this delay by δ_1. p_2 can be similarly tested by launching a transition on e_i, propagating it to e_n along the path and observing it on the primary output e_n after the propagation delay corresponding to p_2. We denote this delay by δ_2. If the propagation delay of p is δ, then $\delta_1 + \delta_2 = \delta$. By ensuring that the propagation delay of p_1 is at most δ_1 and that the propagation delay of p_2 is at most δ_2, we can ensure that the propagation delay of p is at most δ. Thus, p does not have to be tested directly, if p_1 and p_2 are tested. This test application scheme involves multiple clock periods (e.g., δ_1 and δ_2 in the above example). Its advantage is the following. Let the number of paths from the primary inputs to e_i be N_1. Let the number of paths from e_i to the primary outputs be N_2. Then the total number of paths through e_i is $N_1 \times N_2$. After placing a test point on e_i, only $N_1 + N_2$ paths need to be tested. If N_1 and N_2 are large, this is a substantial reduction in the number of paths that need to be tested.

The testability of the circuit also improves due to the test points inserted. This is because in the presence of test points, partial paths (e.g., p_1 and p_2 in the above example) have to be tested as opposed to complete paths (e.g., p). For partial paths there are fewer test generation constraints and tests may exist even if the complete path is untestable.

The above concepts are clarified through an example. Consider the circuit given in Fig. 1. The line numbers are given in the brackets. This circuit has 11 paths. 8 of these paths go through line 10. A test point placed on line 10 cuts it into two parts 10^o and 10^i. 10^o is a primary output for the logic feeding line 10 and 10^i is a primary input for the logic driven by line 10. The modified circuit is given in Fig. 2. The modified circuit now has only 9 paths. Thus, the total number of paths to be tested has reduced.

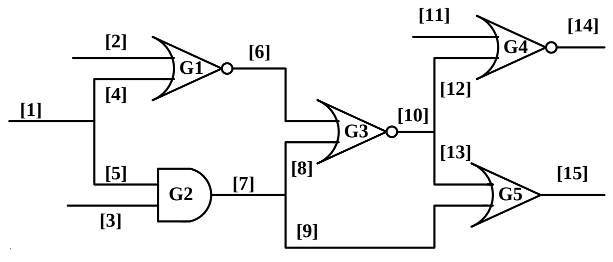

Figure 1: Circuit before test point insertion at line 10.

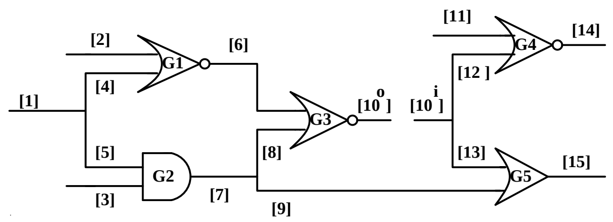

Figure 2: Circuit after test point insertion at line 10.

The details of the test application scheme can be found in [7]. The test point inserted at line 10 also improves the testability of the circuit and makes it completely testable.

3 Bottom Up Approach to Determine Untestable Path Delay Faults

The goal of this paper is to minimize the number of test points inserted to achieve complete testability. Our algorithm is based on making testability modifications only to a set of delay faults that need to be tested (*non-RD* faults) and that are not robustly testable. Computation of this set of faults can not be performed for large circuits in reasonable time. Also, the size of this set of faults can be extremely large. We handle these problems by using a bottom up approach which is described next.

In the bottom up approach, we partition the circuit into subcircuits. The subcircuits may not be disjoint. We start with small subcircuits and solve the testability problems associated with them by inserting test points. Due to the test points inserted, the overall number of faults in the circuit that need to be tested decreases and untestable faults become testable. This enables us to handle larger subcircuits and more global problems.

After choosing a set of subcircuits $\{C_i | i = 1, ..., n\}$, we determine for each C_i a set of path delay faults P_i in C_i that need to be made robustly testable. Test points are then inserted to make the faults from $P = \cup_{i=1,...,n} P_i$ fully testable. In the bottom up approach used, the size of the subcircuits considered in every iteration is gradually increased. In this section we discuss the subcircuit selection and the determination of P. A procedure to insert test points to make P testable is presented in the next section.

3.1 Selection of Subcircuits

We choose the subcircuits such that it would be possible to perform fast identification of local testability problems. Since a reconvergence is usually the cause for untestable faults, the subcircuits for analysis are chosen according to the reconvergence structure of the circuit. The subcircuits are referred to as *reconvergence slices*.

Definition 1: A *reconvergence* is said to occur at a line l due to a line f if there exist at least two disjoint paths from f to l. (Two paths p_1 and p_2 from f to l are said to be disjoint if and only if $p_1 \neq p_2$ and the only edges common to both paths are f and l.)

Definition 2: Let there exist a reconvergence at line l due to line f. Then the *reconvergence slice of f and l* contains all the gates driven by f and driving l and all the partial paths from f to l. The *size* of the reconvergence slice is equal to the number of partial paths from f to l.

For example, consider the circuit in Fig. 1. It has two disjoint paths from line 7 to line 15, viz., 7-8-10-13-15 and 7-9-15. Hence, there is a reconvergence at line 15 due to line 7 where $l = 15$ and $f = 7$. The slice is made up of the gates G3 and G5. It contains two partial paths: 7-8-10-13-15 and 7-9-15 and, hence, its size is 2. A concept similar to reconvergence slices was used to synthesize delay fault testable circuits in [13].

A reconvergence slice is selected for every line l in the circuit for which it is possible to find a line f such that a reconvergence occurs at l due to f. At each line l in the circuit more than one reconvergence may end. In each iteration, we will consider one of them. If there exists more than one reconvergence slice for l, we select that slice from the set of previously unconsidered slices which has the shortest maximal path. We refer to this as the *nearest* reconvergence slice. We run the algorithm in an iterative manner by starting the analysis for reconvergence slices with a small size and increasing the size of the slices analyzed in every iteration. Procedure 1 summarizes the algorithm.

Procedure 1: *Bottom up approach to test point insertion*

1. *path_limit* = 100.
   Repeat until all reconvergence slices with size not exceeding *MAXPATH* have been analyzed:
   2. For every line l in the circuit, choose a *nearest* reconvergence slice C_l that was not analyzed in previous iterations and with size not exceeding *path_limit*.
   3. Determine P_l, the set of faults that need to be made robustly testable in C_l, for all l. (The derivation of this set is explained in Section 3.2). Set $P = \cup P_l$, $\forall l$.
   4. Insert test points to make the faults from P robustly testable. (A procedure for test point insertion is given in Section 4.)
   5. If (*path_limit* < *MAXPATH*),
      path_limit = *path_limit* ×10.

MAXPATH is used to limit the amount of computation time spent in this procedure.

To avoid analyzing the same reconvergence slice at a line l in each iteration (Step 2, Procedure 1), we maintain a list of the reconvergence slices analyzed for l. We do so by keeping track of the lines f of the previously picked reconvergence slices for l. This enables the next nearest reconvergence slice to be considered in the following iteration.

3.2 Determination of P_l

We now discuss the selection of P_l, the set of faults that need to be made robustly testable in any reconvergence slice C_l. Test points are placed on the basis of this set. Hence, we want to include in P_l only those faults that are *necessary* to be made robustly testable for the entire circuit to become robustly testable.

Before going into the details of P_l selection, we formally define the notion of testability of a partial path in a circuit. This is necessary since the paths of C_l are often only partial paths of the overall circuit C.

Definition 3: A path delay fault associated with a *partial path* p in circuit C is said to be non-robustly testable if and only if there exists a two pattern sequence $< T_1, T_2 >$ of input vectors to C such that:

1. The fault transition is launched at the primary input of p (which need not be a primary input of C since p is a only a partial path) and
2. All the off-path inputs of p have non-controlling values under T_2.

The definition of robust testability for a fault associated with a *partial path* is done analogously.

Some basic facts from [10-12] are now reviewed. This will lead to the method used to determine P_l. It was shown in [10-12] that it is sufficient to test robustly a subset of all the path delay faults to check the temporal correctness of the circuit C. This subset of faults that is sufficient to be tested is referred to as a *non-RD* set. The *non-RD* set for a circuit is not unique and there is flexibility in choosing it. Let S_i, $i = 1, 2, ..., n$, denote all the possible *non-RD* sets for a given circuit. Let *RT* (*NRT*) denote the set of robustly (non-robustly but not robustly) testable path delay faults. In [12], it has been proven that the sets *RT* and *NRT* must be part of any S_i, i.e, $RT \cup NRT \subseteq S_i$ for all i.

During the local analysis, we insert test points only to make the *NRT* set of faults robustly testable. Thus, we choose P_l such that it consists of all faults in C_l which are non-robustly testable but not robustly testable in C (cf. Definition 3). A particular *non-RD* set S for the complete circuit is only selected in the final stage. A way to choose this set is given in Section 5. The rationale behind considering only the *NRT* set during the local analysis is explained intuitively next.

Consider a circuit with two *non-RD* sets, S_1 and S_2. Assume that $S_1 = RT \cup NRT \cup \{p_1\}$ and $S_2 = RT \cup NRT \cup \{p_2\}$. When test points are inserted on the basis of the *NRT* set, assume p_2 also becomes robustly testable and p_1 remains robustly untestable. Then, choosing S_2 instead of S_1 as the *non-RD* set of the circuit results in fewer test points. (If S_1 is chosen as the *non-RD* set for the circuit, we have to insert an additional test point to make p_1 robustly testable.)

The heuristic for choosing S aims at minimizing the number of untestable faults in S. This may reduce the total number of test points needed to achieve complete testability.

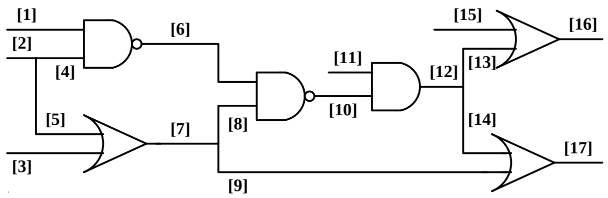

Figure 3: Example for determining P

We now determine P for the circuit in Fig. 3. The circuit has two reconvergence slices C_1 with $l = 10$ and $f = 2$ and C_2 with $l = 17$ and $f = 7$. The size of both the reconvergence slices is 2. The two partial paths in C_1 are 2-4-6-10 and 2-5-7-8-10. The rising and falling transition faults along these partial paths are robustly testable. Hence, $P_1 = \emptyset$ (empty set).

The two partial paths in C_2 are 7-9-17 and 7-8-10-12-14-17. The rising and falling transition faults along 7-9-17 are robustly testable. The falling transition fault along 7-8-10-12-14-17 (transition specified at line

7) is only non-robustly testable. The rising transition fault along 7-8-10-12-14-17 is not even non-robustly testable. Hence, P_2 only includes the non-robustly testable falling transition fault along 7-8-10-12-14-17. Thus, $P = P_1 \cup P_2 = P_2 = \{$*falling transition fault along path 7-8-10-12-14-17* $\}$.

Note that the analysis is performed only on the partial paths in a slice. This is possible since, if the delay fault along a path in a slice is non-robustly testable, it implies that the delay fault along every complete path in the circuit containing this partial path is at most non-robustly testable. In a very rare situation, the delay fault along every complete path containing this partial path may be unnecessary to test and in that case this partial path need not be considered for testability modifications. However, this is an unlikely possibility and we approximate by assuming that there exists a delay fault along at least one complete path containing this partial path which is necessary to test. Hence, we will consider it for test point insertion.

4 Test Point Insertion

In this section we describe the test point insertion algorithm. The aim of the algorithm is to make a given set of faults P robustly testable with the addition of a minimum number of test points. The problem of determining an optimum set of test points is computationally difficult. Hence, a greedy approach of selecting one test point at a time is adopted here.

A weighting procedure is used to determine the order in which edges are selected on which test points are inserted to make P robustly testable. The weight assigned to each edge is chosen to reflect its "goodness" as a test point. Every $p \in P$ contributes to the edge weights. The weighting is such that an edge with a higher weight makes more faults from P testable than an edge with a lower weight. The edge weights due to each $p \in P$ are determined as follows.

For every edge on p, we determine if a single test point placed on that edge makes p robustly testable. If it does, we increase the weight of the edge by 1. If no single edge can make p testable, (i.e., more than one test point is required) we increase the weight of every edge on p by 1.

In the weighting procedure explained above, we only considered the edges on paths from P for weight assignment and, thus, as candidates for test point placement. This is a restriction since, in some cases, a partial path can also be made testable by placing a test point on an edge not included in the path. We reduce our search space and computation time significantly by only considering a subset of all the edges as test point candidates for each path in P. This is sufficient since complete robust path delay fault testability can be achieved by placing test points only on paths along which the delay faults are not robustly testable. Also, in most cases, we observed that this was the best way to improve testability.

The weighting procedure is given in Procedure 2. The weight of an edge e_i is denoted by $\mathrm{W}(e_i)$.

Procedure 2: *Weight of an edge as a test point*
1. Initialize $\mathrm{W}(e_i) = 0$, $\forall i$.
2. For every $p \in P$, $p = e_1 - e_2 - ... - e_n$, do :
   3. For $i = 1$ to n do :
      If a single test point placed at e_i makes p robustly testable, then : $\mathrm{W}(e_i) = \mathrm{W}(e_i) + 1$.
   4. If more than one test point is required to make p robust testable, then :
      For $i = 1$ to n do : $\mathrm{W}(e_i) = \mathrm{W}(e_i) + 1$.

The selection of the first test point now involves choosing the edge with the maximum weight. After a test point is inserted, we re-evaluate the testability of all paths in P. The weights are then updated for all the edges in the circuit. Selection of a test point and updating of weights is done iteratively until P is empty. Procedure 3 summarizes this process.

Procedure 3: *Test point placement for P*
1. Compute weights using Procedure 2.
2. Repeat until P is empty :
   3. Place a test point on the edge with the maximum (non-zero) weight.
   4. Remove all the faults from P that become robustly testable due to this test point. (Test generation to determine robustly testable faults is done using the procedure from [23]).
   5. Update the edge weights.

During the selection of a test point in Step 3 of Procedure 3, if there is more than one edge with the maximum weight, we place the test point on that edge which results in a larger reduction in the total number of paths.

5 Complete Algorithm

The complete algorithm is presented in this section. The algorithm is divided into two phases. The first phase is the bottom up phase which starts by analyzing small regions. Test points are inserted to make these regions testable. This results in an improvement in the testability and a reduction in the number of paths in the circuit. Thus, larger portions of the circuit can be handled. This phase, summarized in Procedure 1, considers regions of size up to *MAXPATH*.

The first phase of the algorithm is not sufficient to make the circuit completely testable. This is because the size of the reconvergence slices analyzed is bounded by *MAXPATH*. Also, in each reconvergence slice, test points were inserted only on the basis of the non-robustly testable faults. These faults along with the robustly testable faults are contained in every *non-RD* set S_i (cf. Section 3.2). However, these faults may only form a subset of every S_i. Hence, the second phase analyzes the entire circuit by choosing a particular *non-RD* set S. S is chosen according to Heuristic 1 in [12]. This heuristic computes a near optimal solution to the problem of minimizing the size of S. Note that the computation of S may not have been feasible at the beginning of phase one for circuits like c6288 of the ISCAS85 benchmarks which have a large number of paths. However, S can be determined in the second phase of the proposed algorithm even for large circuits because the number of paths is much reduced after phase one. Procedure 4 presents the complete algorithm.

Procedure 4: *Complete algorithm*
Phase 1:
  Perform Procedure 1 with $MAXPATH = 10{,}000$, using Procedure 3 for test point insertion.
Phase 2:
  Pick a *non-RD* set S based on Heuristic 1 in [12].
  Set P equal to all the faults from S that are not robustly testable.
  Insert test points using Procedure 3.

Faults which are aborted by the test generator are not considered for test point insertion, i.e., they are not included in P until the end of Phase 2. If some of the aborted faults remain robustly untestable even after Phase 2, additional test points are added using Procedure 3 to make these aborted faults robustly testable. The resulting circuit is completely testable for delay faults, i.e., for every path delay fault of the picked *non-RD* set S, there exists a robust test.

6 Experimental Results

The complete test point insertion algorithm was implemented in the *C* programming language. The underlying test generator of the proposed algorithm is the PODEM-based delay test generator in [23]. A backtrack limit of 100 was used for the test generator. The program was run on the modified ISCAS85 benchmark circuits used in [7]. Table 1 gives the results for these circuits. Under the "Faults" column, the total number of path delay faults in the circuit is given. The next two columns give the total number of test points inserted by the proposed algorithm and the algorithm in [7] respectively. The fifth and sixth columns give the time taken by the proposed algorithm in seconds on a SUN SPARC20 workstation. Column "Phase 1" lists the time taken for the first phase of the algorithm. Under "Total", the total time taken by the program is given. The percentage reduction in the number of test points under the proposed algorithm from [7] is listed in the last column.

At the beginning of Phase 2, if the total number of path delay faults in the circuit was more than 120,000, then test points were added to reduce the number of path delay faults maximally as in [7] (cf. Section 2) to below 120,000. This was required only for c3540. Of the 83 test points added in c3540, 3 of them were added for this purpose.

The proposed heuristics can be added to any test generator for path delay faults. A more sophisticated test generator which quickly recognizes conflicting assignments will improve the run times. Also, it may result in a slight reduction in the number of test points due to fewer aborted faults.

Table 2 gives the number of test points inserted for the combinational parts of the ISCAS89 benchmarks. These circuits have been modified by the same transformations that were applied to the ISCAS85 benchmarks in [7]. Since the robust path delay fault testability of the ISCAS89 benchmarks is better than that for the ISCAS85 benchmarks [22], the number of test points which had to be added to make these circuits completely testable is much less.

There is a trade-off between fault coverage and hardware overhead (i.e. number of test points). The hardware overhead can be significantly reduced if robust fault coverage of less than 100% is acceptable. In Figs. 4 and 5 we illustrate this trade-off by showing how the robust fault coverage improves as the number of test points added to the circuit is increased. Figs. 4 and 5 give the variation for c1908 and c5315 respectively. The curves show that for c1908 (c5315) a fault coverage of about 90% (85%) can be achieved with only half the number of test points needed to achieve complete testability.

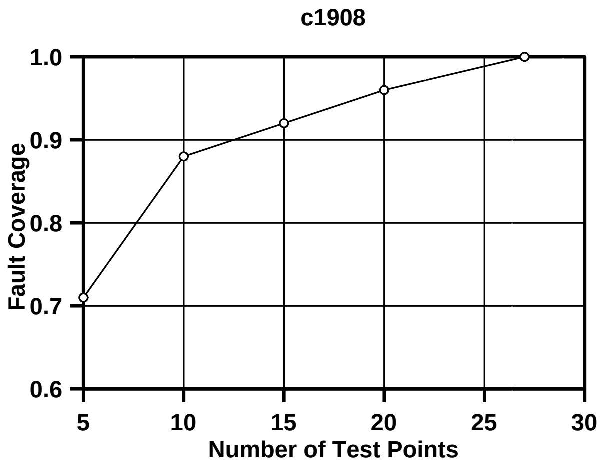

Figure 4: Increase in coverage for c1908 with the addition of test points

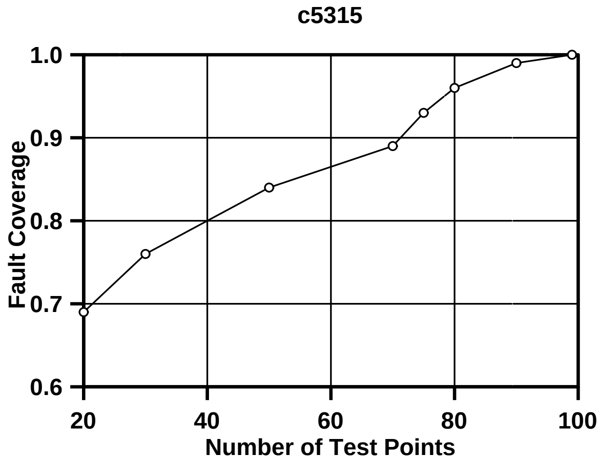

Figure 5: Increase in coverage for c5315 with the addition of test points

It is also important to note that test points can be used to significantly reduce the fault universe and, consequently, to reduce dramatically the test generation and test application time. Most design-for-

Circuit	Faults	Proposed	[7]	Time		%Reduction
				Phase 1	Total	
c880	17,284	4	7	25	268	42.86
c1355	644,224	10	10	6354	6503	0
c1908	1,458,050	27	70	3274	8989	61.43
c2670	34,358	32	120	739	2375	73.33
c3540	15,111,450	83	210	28861	44982	60.48
c5315	2,506,220	99	320	1829	23753	69.06
c6288	9.418×10^{15}	322	1150	12146	25478	72.00
c7552	1,310,174	215	650	11042	87325	66.92
		792	2537			68.78

Table 1: Total test points required to achieve complete testability

testability and synthesis-for-testability techniques do not have this advantage and often result in testable circuits with large fault universes. Consider the two modified circuits c3540 and c6288 which have 15,111,450 and 9.418×10^{15} faults. After the addition of test points to make the circuits completely testable, the total number of faults in the two circuits is only 11,766 and 31,192 respectively.

7 Conclusions

We considered the problem of design-for-testability for path delay faults in large combinational circuits using test points. We presented an algorithm to significantly reduce over an existing technique the number of test points required to obtain completely robust path delay fault testable circuits.

The number of test points required to achieve complete testability is still too high for circuits like c6288 with large numbers of paths and poor path delay fault testability. For such circuits with large path numbers, an approach often used in practice is to only test those paths with an expected delay greater than a given threshold. An interesting subject for future research would be to modify our approach so that test point insertion is targeted at only making these path delay faults robustly testable in order to reduce the number of test points needed.

In our experiments we found many situations where a single test point improved the testability of the circuit significantly. If we had to modify the circuit to achieve the same level of testability (e.g. by Shannon's expansion [20]), the overhead required would have been much higher. However, there were also cases where local circuit modifications such as [19] could have solved the problem at a lower cost. We conjecture that test point insertion should be combined with other design-for-testability and synthesis-for-testability techniques to obtain testable designs cost-effectively. Future work will investigate such possibilities.

Acknowledgements

The authors would like to thank Prof. G. Hotz of the University of Saarland and Prof. S.M. Reddy of the University of Iowa for their support and for many valuable discussions on this subject.

References

[1] J.L. Carter, V.S. Iyengar, and B.K. Rosen, "Efficient Test Coverage Determination for Delay Faults," in Proc. Intl. Test Conf., pp. 418-427, 1987.

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

[3] E.S. Park and M.R. Mercer, "Robust and Nonrobust Tests for Path Delay Faults in Combinational Circuits," in Proc. Intl. Test Conf., pp. 1027-1034, 1987.

[4] S.M. Reddy, M.K. Reddy, and V.D. Agrawal, "Robust Tests for Stuck-Open Faults in CMOS Combinational Logic Circuits," in Intl. Symp. on Fault-Tolerant Computing, pp. 44-49, 1984.

[5] C.J. Lin and S.M. Reddy, "On Delay Fault Testing in Logic Circuits," in IEEE trans. on Computer-Aided Design, Vol. 6, pp. 694-703, Sept. 1987.

[6] M.H. Schulz, F. Fink, and K. Fuchs, "Parallel Pattern Fault Simulation of Path Delay Faults," in ACM/IEEE Design Automation Conf., pp. 357-363, 1989.

[7] I. Pomeranz and S.M. Reddy, "Design-for-Testability for Path Delay Faults in Large Combinational Circuits Using Test-Points," in Proc. Design Automation Conf., pp. 358-364, 1994.

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

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

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

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

Circuit	Faults	Proposed
s298	462	2
s344	710	4
s349	710	4
s382	800	4
s386	414	1
s400	800	4
s444	800	4
s510	738	4
s526	816	3
s526n	816	3
s641	3,488	5
s713	3,284	5
s820	984	2
s832	984	2
s953	2,312	2
s1196	6,196	14
s1238	6,216	13
s1423	84,178	23
s1488	1,924	13
s1494	1,924	12
s5378	21,952	37
s9234	66,086	97
s13207	170,756	50
s15850	8,018,412	114

Table 2: Results for modified ISCAS89 benchmarks

[12] U. Sparmann, D. Luxenburger, K.T. Cheng, and S.M. Reddy, "Fast Identification of Robust Dependent Path Delay Faults," in Proc. 32nd Design Automation Conf., pp. 119-125, 1995.

[13] K. Roy, J.A. Abraham, K. De, and S. Lusky, "Synthesis of Delay Fault Testable Combinational Logic," in IEEE Intl. Conf. on Computer-Aided Design, pp. 418-421, 1989.

[14] A.K. Pramanick and S.M. Reddy, "On the Design of Path Delay Fault Testable Combinational Circuits," in IEEE Intl. Fault-Tolerant Computing Symp., pp. 374-381, 1990.

[15] S. Devadas and K. Keutzer, "Synthesis and Optimization Procedures for Robustly Delay-Fault Testable Combinational Logic Circuits," in ACM/IEEE Design Automation Conf., pp. 221-227, 1990.

[16] 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.

[17] N.K. Jha, I. Pomeranz, S.M. Reddy, and R.J. Miller, "Synthesis of Multi-Level Combinational Circuits for Complete Robust Path Delay Fault Testability," in Fault-Tolerant Computing Symp., pp. 280-287, 1992.

[18] A. El-Maleh and J. Rajski, "Delay-Fault Testability Preservation of the Concurrent Decomposition and Factorization Transformations," in the 12th IEEE VLSI Test Symp., 1994.

[19] H. Hengster, R. Drechsler, and B. Becker, "Testability Properties of Local Circuit Transformations with Respect to the Robust Path-Delay-Fault Model," in VLSI Design Conf., pp. 123-126, 1994.

[20] S. Kundu and S.M. Reddy, "On the Design of Robust Testable CMOS Combinational Logic Circuits," in Fault-Tolerant Computing Symp., pp. 220-225, 1988.

[21] 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., Univ. of Iowa.

[22] K. Fuchs, M. Pabst and T. Roessel, "RESIST: A Recursive Test Pattern Generation Algorithm for Path Delay Faults Considering Various Test Classes," in IEEE trans. on Computer-Aided Design, Vol. 13, No. 12, pp. 1550-1562, Dec. 1994.

[23] S.M. Reddy, C.J. Lin, and S. Patil, "An Automatic Test Pattern Generator for the Detection of Path Delay Faults," in IEEE Intl. Conf. on Computer-Aided Design, pp. 284-287, 1987.

[24] M. Henftling and H. Wittmann, "Bit Parallel Test Pattern Generation for Path Delay Faults," in European Design and Test Conf., pp. 521-525, 1995.

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