Generation of High Quality Non-Robust Tests for Path Delay Faults

Kwang-Ting Cheng
Department of ECE
University of California
Santa Barbara, CA 93106

Hsi-Chuan Chen
AT&T Bell Laboratories
Murray Hill, NJ 07974

Abstract - Earlier research results have shown that for many designs, a large portion of path delay faults is not robustly testable. In this paper, we investigate the test strategy for the non-robustly testable faults. We first present some experimental results to show that the quality of a non-robust test set may be very poor in detecting small delay defects caused by manufacturing process variation. We further show that a better set of non-robust tests can be obtained by including timing information in test generation. A good non-robust test can tolerate a larger timing variation on the off-inputs of the path than a poor test. An algorithm for generating such better quality non-robust tests is presented. We present experimental results to compare quality of non-robust test sets with and without using our method. We also present an algorithm, as well as experimental results, for generating validatable non-robust tests.

1. Introduction

The ultimate objective of delay testing is to find and apply a set of test vectors that can uncover all defects that affect the circuit's timing behavior. Two fault models are commonly used for timing defects: the gate delay fault model [1, 2] and the path delay fault model [3, 4, 5]. There are pros and cons for both models, which has been discussed in many articles (e.g. [6]). In this paper, we use the path delay fault model.

It requires a vector pair to test a delay fault. There are three conditions that can be imposed on the tests for path delay faults: *single path sensitization* condition, *robust* condition and *non-robust* condition. A *single-path-sensitization* test requires all off-inputs of the target path to remain in their non-controlling values such that the timing of the target path can be fully characterized by the test. A less stringent condition is the robust condition. A robust test [4, 5] guarantees the detection of the fault regardless of the delays at all other signals. Results reported in [5] and [7] show that a large number of faults do not have any robust test for many benchmark and industrial circuits. These faults are called robustly untestable faults. The least stringent condition is the non-robust condition. It only requires the vector pair creates the desired transition at the source of the path and the second vector of the pair statically sensitizes the path. A non-robust test will detect the delay fault in the target path if the arrival times of all off-inputs signals are not late (i.e. not faulty). However, if any of them is late, the test may become invalid. Faults that do not have any non-robust test are called non-robustly untestable faults.

We need some definitions for further discussion. Given a logic circuit *C*, an *instance F* of *C* is referred to an implementation of *C*. Each gate and connection of *F* is associated with a propagation delay (which can be a number or a range). Different instances have the same circuit structure (as *C*) but may have different propagation delays at signals and can be considered as models for different manufactured chips. Given a logic design, those instances that do not have any sensitizable path longer than the clock period τ are called *good instances*. The presence of different delay defects will result in different instances.

Paths that are non-robustly untestable may still affect a circuit's performance for certain instances [8]. These paths are called *functional sensitizable* paths. A sufficient condition was suggested in [8] for paths that are false (i.e. unsensitizable) for all possible instances. These paths are considered *redundant* and need not be tested. It has been shown that a large percentage of path delay faults are redundant for many benchmark circuits as well as real designs.

For a non-robust testable fault, there usually exist many non-robust tests. Our preliminary study shows that some of the non-robust tests have a higher quality than others in terms of their capability in detecting delay defects. Existing methods cannot differentiate these tests. For such a fault, we attempt to generate a non-robust test which is *closer* to a robust test. We introduce a metric, called *robustness*, for a non-robust test. The higher the robustness of a non-robust test, the lower the probability of the test becomes invalid (due to extra propagation delays at other signals). We will give the definition of *robustness* of a non-robust test in Section 3. In Section 4, we will show an algorithm for generating non-robust tests with high robustness.

A non-robust test becomes invalid if certain other paths are defective. If we can robustly test those paths that may invalidate a non-robust test for the target path, the non-robust test along with the robust tests for those associated paths form a *validatable non-robust test* for the target fault [9]. To generate high quality non-robust tests, the first attempt should be to generate validatable non-robust tests. To the authors' knowledge, no algorithm or tools have been reported for automatic generation of validatable non-robust tests. In Section 5, we present an algorithm for generating such tests.

We present experimental results in Section 6. We first show how much improvement we can make with respect to the metric we define (i.e. robustness) in generating non-robust tests. We also present results of validatable non-robust test generation for some benchmark circuits. To justify the metric we use for non-robust test generation, we designed and conducted an experiment to assess the quality of different non-robust tests with different level of robustness. We generate a large number of (normally distributed) random instances. These instances are generated in a way to mimic the impact of process variation on signal propagation delays. We then simulate different sets of tests to compare their capability of detecting faulty instances. The details of the experiment are explained in the next section and Section 6.

2. An Experiment

To give a clear motivation for this work, we designed the following experiments to compare the quality of robust and non-robust tests. For a given logic circuit, we generated a large set of instances with different propagation delays at signals to mimic the manufacturing process variation. These generated instances are used for delay testing. The way we generate the propagation delays is described in the next paragraph. An instance whose longest sensitizable path is longer than the clock period is then considered as delay-defective. The longest propagation delay of an instance can be identified by a timing analyzer. We then use a

multiple-delay logic simulator to simulate robust and non-robust tests and compare them on their capability of detecting these delay-defective instances.

In our experiments, the nominal propagation delay for each signal and each gate is given. We assume the propagation delay at each signal is a random variable of normal distribution. The mean is the given nominal propagation delay and the standard deviation is also given. We use a random number generator (for a normal distribution of given mean and standard deviation) to generate the propagation delay for each signal. We generate a large number of instances. For each generated instance, we use our timing analyzer [10] to identify the longest sensitizable path and calculate the propagation delay along that path. This number is referred as the *delay* of the instance. Curve (a) in Fig. 1 shows the distribution of the delays for 4,000 generated instances of c880. The nominal propagation delay at each signal for c880 is obtained from [11]. The nominal propagation delay of the longest sensitizable path is 46.8 nanoseconds (ns). Based on this result, we choose a clock period of 55 ns. to give a reasonable safety margin for the design. None of the 4,000 instances in Fig. 1(a) has a delay longer than this clock period. We then generate another 4,000 instances assuming a 10% shift in the nominal delay at each signal to reflect the worst case process. I.e., the mean of the propagation delay at each signal is assumed to be 1.1 times the given nominal propagation delay. Curve (b) in Fig. 1 shows the distribution of the delays for these 4,000 generated instances of c880. Our timing analyzer reports 556 instances having sensitizable paths longer than 55 ns and, thus, delay-defective. We like to see how the delay tests generated under different sensitization conditions (i.e. robust, non-robust, etc) perform in detecting these delay-defective instances. We prepared two sets of test sequences for path delay faults: (1) robust tests and (2) non-robust tests. For test generation, we only consider those paths whose nominal propagation delays are longer than 38.8 ns. There are totally 2441 paths considered for test generation. As shown in Table 1, 427 faults are not robustly testable and 292 paths are not non-robustly testable. Notice that a non-robust test has more don't cares in its two-pattern tests than a robust test, if both tests exist. In generating the non-robust tests, we leave as many don't cares as possible in the two-pattern tests and then randomly fill in 0 and 1 for those don't cares.

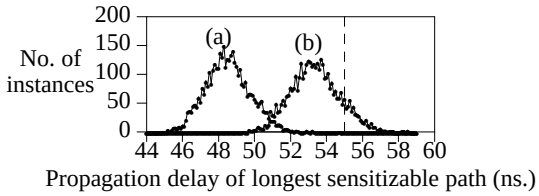


Fig. 1: Distribution of longest sensitizable path delays for 4000 generated instances of c880.

We then used AT&T's multiple-delay logic simulator MIDAS that accepts externally supplied gate/signal propagation delays to simulate both sets of tests. In simulating a two-pattern test for an instance, all signals under the first vector are stabilized before the second vector is applied. The outputs are then observed 55 ns later and the observed values are compared with the expected values to determine the detection. Table 1 shows the number of test vector pairs in each test set and the number of detected instances out of total 556 delay-defective instances.

A similar experiment is conducted for circuit *styr*, a MCNC FSM benchmark example synthesized by our synthesis system. Table 2 shows the results for circuit *styr*. Totally 210 long paths are considered for test generation. Out of the 10,000 instances that were generated assuming a 10% increase in nominal propagation delays, 5825 instances have sensitizable paths longer than the specified clock period.

Table 1 - Delay testing results of C880.

Test set	# path delay faults		No. of	# faulty instances	
	detected	untestable	vector pairs	detected	total
Robust	2014	427	2014	378	556
Non-robust	2149	292	2149	353	556

Table 2 - Delay testing results of styr.

Test set	# path delay faults		No. of	# faulty instances	
	detected	untestable	vector pairs	detected	total
Robust	145	65	145	2103	5825
Non-robust	181	29	181	139	5825

From these results, we observed that the quality of a non-robust test set is clearly inferior to a robust one. Many faulty instances detected by the robust test set were not detected by the non-robust test set. Therefore, if a fault is robustly testable, a robust test should be used. However, a certain percentage of paths are robustly untestable for most circuits [5, 7]. For those robustly untestable while non-robustly testable paths, the non-robust tests may fail to detect defects in these paths. Synthesis for robust delay testability [12, 13, 14] is a possible solution to this problem. However, synthesizing a 100% robustly testable circuit may require more logic and/or more primary inputs. Therefore, it may not be a feasible solution for certain designs. In this paper, we attempts to solve the problem by test generation. Given a set of paths which are not robustly testable, we attempt to generate a set of non-robust tests with higher test quality. For each robustly untestable but non-robustly testable fault, we attempt to generate a non-robust test which is *closer* to a robust test. We introduce a metric, called *robustness*, for a non-robust (NR) test. The higher the robustness of a non-robust test, the lower the probability of the test becomes invalid (due to extra propagation delays at other signals). We will give the definition of *robustness* of a NR test in the next section.

The results shown in Tables 1 and 2 also indicate that, for these two circuits, neither the robust test set nor the nonrobust test set detects all faulty instances. There are two reasons for this: (1) There exists some functional sensitizable paths [8] for both circuits. These path are neither robustly testable nor non-robustly testable while they may affect the circuit's performance. No tests are generated for these paths and thus timing defects on these paths might not be detected. (2) We only consider long paths for test generation. Therefore, a faulty short path may not be detected. However, for this experiment, few instances were missed due to this reason.

3. Definitions

A **path** P_x is a logical path associated with a physical path $P = (g_0, f_0, g_1, f_1, \ldots\ldots, f_{m-1}, g_m)$, which is an alternating sequence of gates and connections, and a transition $x \in$ {rising, falling} at the source of the path. Gate g_0 is a primary input and g_m is a primary output. Connections f_i, $0 \le i \le m-1$, is an **on-input** of P_x which connects gate g_i to gate g_{i+1}. A connection is called an **off-input** of f_i if it is connected to g_{i+1} but not originated from g_i. In this paper, we call a path and a path delay fault interchangeably.

Given a logic circuit, there is an instance corresponding to the desired timing specification. We call such an instance as the fault-free instance.

Let V=<v_1, v_2> be an input vector pair and v_2 is applied at time t = 0. The logic values stabilized at connection *f* and at gate *g* under v_2 are called the **stable values** at *f* and *g* under v_2 respectively. The times, when *f* and *g* become stable under v_2, are called the **stable times** at *f* and *g* under v_2 respectively. Let $ST(f, v_2)$ be

the stable time at connection f under v_2 for the fault-free instance.

A two-pattern test $<v_1, v_2>$ is a NR test for a path fault if and only if (1) it launches the transition at the root of the path and (2) v_2 causes all off-inputs to settle at their non-controlling values. A NR test may become invalid if any *non-robust off-input* of the path is late.

Definition (non-robust off-inputs): Given a path-delay fault and a NR test V (a vector pair), non-robust off-inputs are those off-inputs at which the transitions may mask the transitions at the on-inputs. Similarly, a *robust off-input* has either a stable non-controlling value or a transition that can never mask the transitions at the on-inputs.

For an AND gate g, if the on-input of g has a falling transition under V, those off-inputs of g that do not have a stable 1 under V are non-robust off-inputs. If the on-input of g has a rising transition, the off-inputs are robust off-inputs for all non-robust tests.

Definition (slack at a non-robust off-input): Given a NR test V and the fault-free instance, suppose f_s is a non-robust off-input and f_o is the corresponding on-input. The *slack at the non-robust off-input s*, denoted as $slack(f_s, V)$, is defined as: $ST(f_o, V) - ST(f_s, V)$.

Definition (slack of a non-robust test): Given a non-robust test V and the fault-free instance, suppose f_{s_i}, $i = 1,2,...,n$, are non-robust off-inputs and f_{o_i}, $i = 1,2,...,n$, are the corresponding on-inputs. The *slack of the non-robust test V*, denoted as $slack(V)$, is defined as: $MIN_{i=1,2,...,n}\, slack(f_{s_i}, V)$.

Among all non-robust tests, we attempt to find a non-robust test V that has the maximum value of $slack(V)$. The reason is the following: the larger the slack at the non-robust off-inputs, the lower the probability that the transitions at the non-robust off-inputs mask the on-input transitions. A non-robust test with a larger slack can tolerate larger timing variations at the non-robust off-inputs. For delay defects caused by process variation, the slack of a non-robust test should be closely related to the probability of fault masking at the non-robust off-inputs. Fig. 2(a) and Fig. 2(b) show two different non-robust tests for a non-robust testable while robust untestable path. Only a portion of the circuit is shown for illustration and the target path is highlighted by thicker lines. The arrival times of the transitions (for the fault-free circuit) are also shown in the figure. The propagation delay of a logic gate is assumed to be 1 ns. For the test given in Fig. 2(a), the arrival time of the on-input of the output gate is 20 ns. and the off-input arrival time is 18 ns. Therefore, it can only tolerate a 2 ns. variation at this off-input while the test in Fig. 2(b) can tolerate a 12 ns. variation.

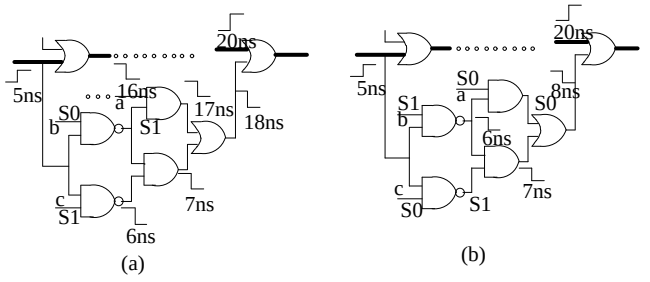


Fig. 2: (a) A non-robust test that tolerate less timing variation. (b) A non-robust test that can tolerate more timing variation.

We define the *robustness* of a non-robust test as its slack. Notice that the robustness ranges from -∞ to ∞. For a robust test, its robustness is infinity. For a vector pair which is neither a robust nor a non-robust test, the robustness is defined as -∞.

4. Generating Non-robust Tests with High Robustness

In this section, we describe an algorithm for generating non-robust tests with large slacks.

To non-robustly test a path delay fault, all off-inputs must have a non-controlling value under v_2 and a transition must be created at the source of the path under V. We first find all these *mandatory assignments* [15] and their implications. We then compute the earliest arrival time of each signal *restricted to these mandatory assignments.* Notice that given a Set of Mandatory Assignments (SMA), certain vector pairs may produce values violating the given assignments at certain signals. The earliest arrival time at a signal f under a given set of mandatory assignments *SMA* is the earliest arrival time at f among all vector pairs not violating *SMA*.

We then identify all non-robust off-inputs NI_i, $i = 1,2,...,m$. We attempt to convert the non-robust off-inputs to become robust off-inputs one at a time (by assigning proper values at these off-inputs) in a proper order. The order of the non-robust off-inputs for processing is determined in the following way. For each of NI_i, we compute the difference, denoted as D_i, between the arrival time of the corresponding on-input and the earliest arrival time of NI_i. We choose the NI_i with minimal D_i and convert it into a robust off-input by assigning a proper value to it. A non-robust off-input with a smaller D_i has a higher probability of masking its on-input transition than one with a larger D_i.

The conversion process is best illustrated by an example. Consider an AND gate in the target path. If the on-input has a falling transition and an off-input f_s has a value of X1, f_s is a non-robust off-input. If we can assign f_s to a stable 1 without causing any conflict, this off-input is converted into a robust off-input. Once we assign a stable value at NI_i, implications of the assignment are performed. If the assignment leads to a conflict, the assignment at this off-input is recovered to its original non-robust value. E.g., if assigning a stable 1 at NI_i in the above example causes a conflict in SMA, it will be backtracked and leave the value at NI_i as X1. On the other hand, if the assignment causes no conflict, the earliest arrival time at every signal is updated incrementally (the earliest arrival times for certain signals may change due to the augmented set of mandatory assignments). For either case (the conversion succeeds or fails), the next non-robust off-input with the smallest D_i is then selected for conversion. The conversion process continues until all non-robust off-inputs are processed once. This process results in a *partially assigned vector pair T.*

For a non-robust off-input NI_i that cannot be converted into a robust one, transitions at NI_i cannot be avoided under T. I.e., all possible vector pairs covered by T will create transitions at NI_i. To minimize the probability of the on-input transition being masked by the transitions at NI_i, we try to find a test that the arrival time of the transition at NI_i is the earliest possible. To achieve this, we use the calculated earliest arrival times to guide test generation. To justify a transition at the output of a gate, we choose an input, among all inputs which could have a transition under current partial test, with the earliest arrival time and assign a transition at this input and assign stable non-controlling values at all other inputs. This backward justification process continues until primary inputs are reached or a conflict is occurred. For the latter case, we backtrack to the last decision point and justify the transition at the input with the next earliest arrival time. This justification and backtracking processes are very similar to those used in a typical test generation algorithm.

In this test generation process, we gradually assign values at the internal signals and primary inputs. Therefore, those non-robust off-inputs that are processed later will have a smaller search space than those processed first. That is why we process the most critical

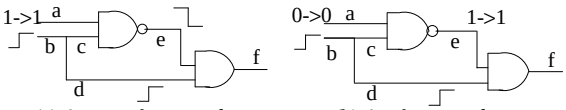
off-inputs (with large D_i's) first.

5. Generating Validatable Non-Robust Tests

In this section, we describe a method of generating tests for validatable non-robustly (VNR) testable faults.

A *set* of two-pattern tests **S** is called a *validatable non-robust test* for a path *P* if and only if no element of **S** is a robust test for *P* and if the circuit passes all tests in **S** it can be concluded that the desired transition propagates along path *P* in the time allowed [9]. To obtain a high quality test for a robustly untestable fault, the first attempt should be to generate a validatable non-robust test because the quality level of a validatable non-robust test is the same as that of a robust test.

Example (a validatable non-robust test): Consider the circuit given in Fig. 3(a). Path {bcef, rising} is robustly untestable but non-robustly testable. The test T1=<10, 11> is a NR test for path {bcef, rising}. If the partial path {bd, rising} is faulty (and, thus, path {bdf, rising} is faulty), it will invalidate the test. Test T2=<00, 01> shown in Fig. 3(b) is a robust test for path {bdf, rising}. If the circuit passes both tests, we can conclude that the path {bcef, rising} is fault-free. Therefore, these two tests form a validatable non-robust test for the path.



(a) A non-robust test for path {bcef, rising}

(b) A robust test for path {bdf, rising}

Fig. 3: A validatable non-robust test for path {bcef, rising}.

We are interested in finding such a set of two-pattern tests for a non-robustly testable fault, if exists. There may exist many validatable non-robust tests for a fault. We are interested in finding the set with minimum cardinality.

For each robustly untestable but NR testable fault, we first convert NR off-inputs to robust ones and obtain a non-robust test *T* with a minimum number of non-robust off-inputs left. This is done in a similar fashion to the procedure described in the last section. I.e., we start with a minimally specified non-robust test. We then convert the non-robust off-inputs to robust ones one at a time. If the conversion of non-robust off-input NI_i causes a conflict, NI_i will be backtracked to its value before conversion (i.e. X0 or X1). We then proceed to the next NR off-input and repeat the process.

We call the partial test obtained after processing all non-robust off-inputs (either successfully converting them into robust off-inputs or leaving them as X1 or X0) as a *semi-robust* test. The semi-robust test has don't cares at some primary inputs. We specify these don't cares in such a way that the transitions at the primary inputs are minimized. I.e., a X1 is specified as 11, X0 is specified as 00, and XX as 00 or 11. We denote the final non-robust test as *T*. The implications of *T* are then performed. We then examine the non-robust off-inputs and identify the paths that need to be robustly tested to validate *T*. Suppose the target path is $\{g_1, f_1, g_2, f_2, \ldots, g_o\}$ where g_1 is a primary input and g_o is a primary output. Suppose s_i is a non-robust off-input and its corresponding on-input is f_i. We denote the partial path from s_i to g_o as p_i. Because *T* is a non-robust test and s_i is a non-robust off-input, *T* must sensitize one or more partial paths from primary inputs to s_i. We denote these partial paths as $q_{i_1}, \ldots, q_{i_m}$. Under *T*, the arrival time of s_i is determined by the propagation delays of these paths. If we can robustly test the concatenated paths, q_{i_1}-p_i, ..., q_{i_m}-p_i, and the circuit passes these tests, it can be guaranteed that, under *T*, the transition arrived at s_i will not be late. For each non-robust off-input s_i, we identify the set of paths, S_i, that needs to be robustly tested to validate *T*. If all of them are robustly testable, test *T* along with the robust tests for these paths form a validatable non-robust test for the target path.

In developing test *T*, we minimize the number of transitions at primary inputs. Therefore, typically only a very small number of partial paths that end at a non-robust off-input are sensitized. Thus, only a small number of off paths need to be examined. This not only reduces the computational complexity but also reduces the size of the validatable non-robust test.

There is a possible extension to this algorithm to identify more VNR testable paths. For those sensitized partial paths (from primary inputs to non-robust off-inputs), instead of having them been robustly tested, we can relax the condition to have them VNR tested. However, if we adopt this extension, the following situation may occur: In generating a VNR test for path A, we require path B to be VNR testable and in generating a test for path B, we require path A to be VNR testable. For such a case, neither of these two paths is VNR testable. In our prototype program (described in the next section), we did not implement this extension.

6. Experimental Results

We have implemented a path delay fault test generator for generating (1) validatable non-robust tests and (2) non-robust tests of large slacks. We ran the program for four circuits: a two-bit adder, two ISCAS85 benchmark circuits c432 and c880 and a MCNC benchmark finite state machine *styr*. Circuit *styr* is synthesized by our FSM synthesis system and uses AT&T 0.9 μm CMOS standard cell library. These circuits are randomly picked from the example pools available to us. For *adder*, we consider all paths. For c432, c880 and *styr*, we chose 53217, 2441 and 210 longest paths (propagation delays between 45 to 49.6 ns., 38.8 to 46.8 ns., and 30.0 to 42.8 ns. respectively). Table 3 shows results for VNR test generation. The third column shows the number of robustly testable faults. The fourth column shows the number of NR testable faults among the robustly untestable faults. Among the NR testable faults, we show the number of faults identified as VNR testable. The last column shows the average number of vector-pairs required for testing such a VNR fault (i.e. the number of off paths that needs to be robustly tested plus one). For example, for c880, we identify 111 out of 135 NR testable faults are VNR testable. Each of the 111 VNR testable faults only requires exactly one other path to be robustly testable to validate its NR test. Therefore, for each of these 111 faults, its validatable non-robust test consists of exactly two two-pattern tests. Among the paths we consider for c432, none of them is robustly testable or VNR testable.

Table 3 - Test generation results for validatable non-robust tests

Ckt	# of paths consid-ered	# of robust testable	# of NR test-able	VNR testable # of paths	VNR testable ave. # of 2-pttn tests per path
adder	60	20	12	2	2.0
c432	53217	0	760	0	-
c880	2441	2014	135	111	2.0
styr	210	145	36	22	2.0

For each of the NR testable paths that are robustly untestable, we attempt to generate a NR test with a large slack. Table 4 shows the results for NR test generation. The second column gives the number of NR testable faults for this experiment. The third column shows the average slack of these NR tests using our test generation algorithm. For comparison, we generate two other sets of NR tests. One set is obtained by simply generating a NR test for each path. After a two-pattern NR test is generated, we *randomly*

fill in 0 and 1 for those unspecified bits. Instead of randomly filling in the unspecified bits, the other set is obtained by filling in the unspecified bits in a way to maximize the transitions at the primary inputs and, thus, at the internal signals for the NR test of each path. We measure the slack of each of the NR test in both sets of tests. The fourth and fifth columns show their average slacks. For adder, the average slack for randomly filled NR test is negative. This result implies that for the fault-free instance, the desired transition cannot be created at the output because the non-robust off-inputs have transitions arriving later than the on-input transitions and will mask them. For all four test cases, our test generator increases the average slacks and thus our test set will allow larger timing variations at the non-robust off-inputs.

Table 4 - Results for non-robust tests

ckt	# of NR testable	Average slack (ns.) ours	random	max. transitions
adder	12	0.16	-0.91	-1.33
c432	760	13.56	6.15	1.08
c880	135	7.99	5.81	5.51
styr	36	12.85	5.63	1.39

We then compare these three test sets on their capability of detecting delay defects. We construct 5 test sets for a set of chosen long paths. Test set 1 consists of a robust test for each of the robustly testable paths in the set of chosen paths. Test set 2 consists of a NR test for each of the robust or NR testable paths with randomly filled don't cares. Test set 3 consists of test set 1 and a NR test for each of the NR testable path with randomly filled don't cares. Test set 4 consists of test set 1 and a NR test for each of the NR testable path where the don't cares are filled to maximize the transitions at the primary inputs. Test set 5, which is our best test set, consists of test set 1, VNR tests for VNR testable paths, and a NR test with maximum slacks for each of the remaining NR testable path. We prepare these five test sets for both c880 and styr and apply them to 556 and 5825 delay-defective instances as generated by the procedure described in Section 2. These instances are verified as defective by a timing analyzer that reports the longest sensitizable path for a given instance. Table 5 shows the results. For c880, Set 1 detects 378 out of 556 defective instances. The VNR tests along with the NR tests generated by our tool intended for large slacks uncover 12 additional defective instances (390 vs. 378) while the NR tests intended for maximum transitions does not detect anyone more (378 vs. 378). If we use NR tests with randomly filled don't cares for all robustly and NR testable paths, it uncovers 25 less defective instances than set 1 (353 vs. 378). Notice that for *styr*, the quality of test set 2 is much worse than test set 1.

Table 5 - Comparison of different test sets

ckt	# of faulty instances	Set 1 (R)	Set 2 (rand. NR)	Set 3 (R + rand. NR)	Set 4 (R+ worst NR)	Set 5 (R+VNR + best NR)
c880	556	378	353	382	378	390
styr	5825	2103	139	2137	2106	2230

A large number of defective instances still cannot be detected by our tests. One of the main reasons is that our test set does not include tests for functional sensitizable paths and thus will miss some defective instances.

7. Conclusion

We conducted experiments to show that, among all possible NR tests for a NR testable path, some NR tests are better than others in defecting delay defects. A good NR test allows a larger timing variation (slack) at the off-inputs than a poor NR test.

We describe a method for generating validatable NR tests. We also present an algorithm for generating NR tests with large slacks. Our results show that different NR tests may have very different slacks. By carefully selecting those tests with large slacks, the quality of a NR test set can be improved. This claim is justified by comparing several sets of NR tests on their capability of detecting defective instances which are generated to mimic process variation. Our future work includes searching for efficient methods to test paths that are not NR testable but functional sensitizable.

REFERENCES

1. K. D. Wagner, ''The Error Latency of Delay Faults in Combinational and Sequential Circuits,'' *Proc. Int'l Test Conference*, pp. 334-341 (Nov. 1985).
2. J. A. Waicukauski, E. Lindbloom, B. Rosen, and V. Iyengar, ''Transition Fault Simulation,'' *IEEE Design and Test*, pp. 32-38 (April 1987).
3. J. P. Lesser and J. J. Shedletsky, ''An Experimental delay test generator for LSI logic,'' *IEEE Trans. on Computers*, pp. 235-248 (March 1980).
4. G. L. Smith, ''Model for Delay Faults Based upon Paths,'' *Proc. IEEE Int'l Test Conf.*, pp. 342-349 (Nov. 1985).
5. C. J. Lin and S. M. Reddy, ''On Delay Fault Testing in Logic Circuits,'' *IEEE Trans. on Computer-Aided Design*, pp. 694-703 (Sept. 1987).
6. B. Konemann et al, ''Delay Test: The Next Frontier for LSSD Test Systems,'' *Proc. Int'l Test Conf.*, pp. 578-587 (Sept. 1992).
7. K. Fuchs, F. Fink, and M. H. Schulz, ''DYNAMITE: An Efficient Automatic Test Pattern Generation System for Path Delay Faults,'' *IEEE Trans. on CAD* **CAD-10**, pp. 1323-1335 (Oct. 1991).
8. K.-T. Cheng and H. C. Chen, ''Delay Testing For Non-Robust Untestable Circuits,'' *to appear in Proc. Int'l Test Conf.* (Oct. 1993).
9. S. M. Reddy, C. J. Lin, and S. Patil, ''An Automatic Test Pattern Generator for the Detection of Path Delay Faults,'' *Proc. Int'l Conf. on CAD*, pp. 284-287 (Nov. 1987).
10. H.-C. Chen and D. H.-C. Du, ''Path Sensitization in Critical Path Problem,'' *IEEE Transactions on Computer-Aided Design* **12-2**, pp. 196-207 (Feb. 1993).
11. S. Patil and S. M. Reddy, ''A Test Generation System For Path Delay Faults,'' *Proc. Int. Conf. Computer Design (ICCD-89)* , pp. 40-43 (Oct. 1989).
12. S. Devadas and K. Keutzer, ''Synthesis of Robust Delay-Fault Testable Circuits: Theory,'' *IEEE Transactions on Computer-Aided Design* **11-1**, pp. 87-101 (Jan. 1992).
13. A. Pramanick and S. M. Reddy, ''On The Design of Path Delay Fault Testable Combinational Circuits,'' *Proc. 20th Fault Tolerant Computing Symp.*, pp. 374-381 (June 1990).
14. 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,'' *Proc. Int'l Symp. on Fault-Tolerant Computing*, pp. 280-287 (July 1992).
15. T. Kirkland and M. R. Mercer, ''A Topological Search Algorithm For ATPG,'' *Proc. 24th Design Automation Conf.*, pp. 502-508 (June 1987).