# Test-Pattern Grading and Pattern Selection for Small-Delay Defects[1,2]

Mahmut Yilmaz, Krishnendu Chakrabarty
Duke University
Dept. of Electrical and Computer Engineering
{my, krish}@ee.duke.edu

Mohammad Tehranipoor
University of Connecticut
Dept. of Electrical and Computer Engineering
tehrani@engr.uconn.edu

## Abstract

*Timing-related defects are becoming increasingly important in nanometer technology designs. Small delay variations induced by crosstalk, process variations, power-supply noise, as well as resistive opens and shorts can potentially cause timing failures in a design, thereby leading to quality and reliability concerns. We present a test-grading technique to leverage the method of output deviations for screening small-delay defects (SDDs). A new gate-delay defect probability measure is defined to model delay variations for nanometer technologies. The proposed technique intelligently selects the best set of patterns for SDD detection from an n-detect pattern set generated using timing-unaware automatic test-pattern generation (ATPG). It offers significantly lower computational complexity and it excites a larger number of long paths compared to previously proposed timing-aware ATPG methods. We show that, for the same pattern count, the selected patterns are more effective than timing-aware ATPG for detecting small delay defects caused by resistive shorts, resistive opens, and process variations.*

## 1 Introduction

Very deep sub-micron (VDSM) process technologies are leading to increasing densities and higher clock frequencies for integrated circuits (ICs). However, VDSM technologies are especially susceptible to process variations, crosstalk noise, power-supply noise, and defects such as resistive shorts and opens, which induce small delay variations in the circuit components. Such delay variations are referred to as small-delay defects (SDDs) in the literature [1, 13].

Although the delay introduced by each SDD is small, the overall impact can be significant if that path is critical, has low slack, or includes many SDDs. The overall delay of the path may become larger than the clock period, causing circuit failure or temporarily incorrect results. As a result, the detection of SDDs typically requires fault excitation through shortest-slack paths. The longest paths in the circuit, except false paths and multi-cycle paths, are referred to as the least-slack paths.

As a result of growing industry concerns regarding SDDs, commercial timing-aware ATPG tools have become available, e.g., new versions of Mentor Graphics FastScan, Cadence Encounter Test, and Synopsys TetraMax tools [2, 6, 9]. However, the test generation time increases considerably when these tools are run in timing-aware mode. Fig. 1 shows a comparison of the run times of traditional transition delay-fault ATPG and timing-aware ATPG (using the commercial FastScan tool) for some of the IWLS'2005 benchmarks [5]. As seen, when the benchmark is large, the timing-aware ATPG takes significantly longer CPU time, e.g., as much as 19x more for the "wb_conmax" benchmark. The CPU times in Figure 1 are normalized such that the run-time of traditional transition delay-fault ATPG is taken to be one unit.

Delay defects are commonly targeted using the transition delay-fault (TDF) model [11]. The effectiveness of the TDF model for SDDs has often been questioned [1, 10]. Classical ATPG tools tend to excite transition delay-faults through short paths [1]. This results in test sets whose coverage is inadequate for SDDs. If we can create a list of all the long paths in a design and generate path delay-fault patterns for all these paths, the pattern-set quality will be high in terms of SDD coverage. However, the number of paths in a circuit increases exponentially with circuit size [7], and it is not practical to identify all the long paths in a large circuit using static timing analysis (STA) tools. Even with a list of long paths, it is unlikely that an ATPG tool can find patterns for all paths, mainly due to robust and non-robust pattern-generation constraints.

The length of a sensitized path is a more effective measure of the quality of delay-fault patterns. However, "long paths" alone do not completely address the effects of SDDs if only nominal delay values are considered. As discussed above, SDDs are often caused by process variations and crosstalk, which introduce delay variations.

[1]The work of M. Yilmaz and K. Chakrabarty was supported in part by SRC under Contract no. 1588 and by an equipment grant from Intel Corporation.
[2]The work of M. Tehranipoor was supported in part by SRC under Contracts no. 1455 and 1587.

1093-0167/08 $25.00 © 2008 IEEE
DOI 10.1109/VTS.2008.32

IEEE computer society

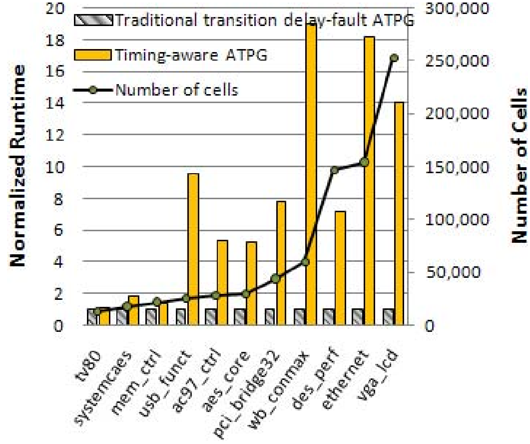


**Figure 1. Comparison of FastScan run times (CPU time) for traditional TDF ATPG and timing-aware ATPG on IWLS 2005 benchmarks.**

The complexity of today's ICs and shrinking process technologies are also leading to prohibitively high test data volumes. The 2005 ITRS document predicts that the test data volume and test application time for integrated circuits will be as much as thirty times larger in 2010 than they are today [4]. Therefore, novel TDF pattern selection methods are required to reduce the total pattern count while effectively targeting SDDs.

Test-pattern reordering methods, which rank test patterns and place the most effective test patterns at the top of the reordered test sequence, promise reductions in both testing time and test data volume [8, 12]. Applying the most effective patterns first during volume ramp-up increases the likelihood of detecting manufacturing defects in less time. If highly effective test patterns are applied first in a reordered test set, defective chips will fail earlier, reducing test application time in an abort-at-first-fail environment. Test pattern reordering is also important for time-constrained and wafer-sort environments. The reordered test set can be simply truncated to fit test-data-volume and test-time budgets.

This paper uses the output deviation measure [12] as a surrogate coverage-metric for SDDs and a test-pattern grading method to select the best patterns for SDD detection from a large repository test set. A flexible, but general, probabilistic fault model is used to generate a probability map for the circuit, which is subsequently used for pattern reordering. The proposed method can be used with traditional, "no-timing" ATPG tools to generate a high-quality delay-fault pattern set. We start with an initial set of patterns (without loss of generality, we consider $n$-detection transition-fault test patterns) and apply our pattern-grading method to calculate output deviations for each pattern. We then sort the patterns according to their ability to detect SDDs effectively. The number of patterns in the final pattern set is determined by the user depending on test-time budget and target fault coverage. Experimental results show that the proposed method can effectively select the highest-quality patterns. It also considers delay variations, unlike most previously proposed methods.

The remainder of this paper is organized as follows. In Section 2, we describe the probabilistic fault model and the output deviations metric. Section 2.4 presents the proposed pattern-selection procedure. In Section 3, we evaluate the proposed method for benchmark circuits and $n$-detection TDF test sets. We also conduct simulated defect injection experiments to evaluate the effectiveness of the selected patterns for detecting small delays caused by resistive shorts and opens. Section 4 concludes the paper.

## 2 Probabilistic Delay-Fault Model and Output Deviations for SDDs

In this section, we first introduce the concept of gate-delay defect probabilities (DDPs) (Section 2.1) and signal-transition probabilities (Section 2.2). These probabilities extend the notion of confidence levels, defined in [12] for a single pattern, to pattern-pairs. Next, we show how to use these probability values to propagate the effects of a test pattern to the test observation points (scan flip-flops/primary outputs) (Section 2.2). We describe the algorithm used for signal-probability propagation (Section 2.3). Finally, we describe how test patterns can be ranked and selected from a large repository (Section 2.4).

### 2.1 Gate-Delay Defect Probabilities

Each gate in a design is assigned DDPs based on the schematic/layout information. DDPs for a gate are provided in the form of a matrix called the *Delay Defect Probability Matrix* (DDPM). The DDPM for a 2-input OR gate is shown in Table 1. The rows in the matrix correspond to each input port of the gate and the columns correspond to the initial input state during a transition. Each entry in the matrix denotes the probability that corresponding output transition is delayed beyond a threshold. For instance, the entry in the first row and the third column for the DDPM in Table 1 shows that there is 50% probability that there will be a delay defect because of the transition on `IN0` when the output makes a ($H \rightarrow L$) transition starting with the initial input state of '`10`'. For initial state '`11`", we consider simultaneous transition on `IN0` and `IN1`. The entries in Table 1 have been chosen arbitrarily for the sake of illustration.

**Table 1. Example DDPM for a 2-input OR gate**

| | | Initial Input State | | | |
|---|---|---|---|---|---|
| | **prob** | **00** | **01** | **10** | **11** |
| Inputs | **IN0** | 0.2 | 0 | 0.5 | 0.1 |
| | **IN1** | 0.1 | 0.2 | 0 | |

For an $N$-input gate, the DDPM consists of $N \cdot 2^N$ entries, each holding one non-zero probability value. If the gate has more than one output, each output of the gate has a different DDPM, which depends on the inputs affecting the output. Note that the DDP is 0 if the corresponding final input state cannot provide the expected output transition.

We next discuss how a DDPM is generated. Each entry in DDPM indicates the probability that the delay of the gate is more than a predetermined value, i.e., the *critical delay value* ($T_{CRT}$). Given the probability density function (pdf) of a delay distribution, the DDP is calculated as:

$$DDP = Prob(x > T_{CRT}) = \int_{T_{CRT}}^{\infty} pdf(x)\, dx \qquad (1)$$

For instance, if we assume a Gaussian delay distribution for all gates (with mean $\mu$) and set the critical delay value to $\mu + X$ ps, each DDP entry can be calculated by replacing $T_{CRT}$ with $\mu + X$ and using a Gaussian pdf. Note that the delay for each input-to-output transition delay may have a different mean ($\mu$) and standard deviation ($\sigma$).

The delay distribution can be obtained in different ways: (*i*) using the delay information provided by the Standard Delay Format (SDF) file; (*ii*) using slow, nominal, and fast process corner transistor models; (*iii*) simulating process variations. In the third method, which is employed in this paper, transistor parameters affecting the process variation and the limits of the process variation ($3\sigma$) are first determined. Monte Carlo simulations are next run for each library gate under different capacitive loading and input slew rate conditions. Once the distributions are found for the library gates, depending on the layout, the delay distributions for each individual gate can be updated. Once the distributions are obtained, $T_{CRT}$ can be appropriately set to compute the DDPM entries. The effects of crosstalk can be simulated separately and the delay distributions of individual gates/wires can be updated accordingly.

### 2.2 Propagation of Signal-Transition Probabilities

Since pattern pairs are required to detect TDFs, there can be a transition on each net of the circuit for every pattern-pair. If we assume that there are only two possible logic values for a net, i.e., `LOW` (L) and `HIGH` (H), the possible signal transitions are $L \to L$, $L \to H$, $H \to L$, and $H \to H$. Each of these transitions has a corresponding probability, denoted by $P_{L\to L}$, $P_{L\to H}$, $P_{H\to L}$, and $P_{H\to H}$, respectively: $< P_{L\to L}, P_{L\to H}, P_{H\to L}, P_{H\to H} >$. Note that $L \to L$ or $H \to H$ implies that the net keeps its value, i.e., no transition occurs.

The nets that are directly connected to the test-application points are called *initialization nets* (INs). These nets have one of the signal-transition probabilities, corresponding to the applied transition test pattern, equal to 1.

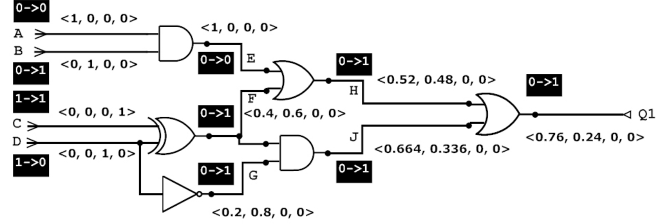


**Figure 2. An example to illustrate the propagation of signal-transition probabilities through the gates of a logic circuit.**

All the other signal-transition probabilities for INs are set to 0. When signals are propagated through several levels of gates, the signal-transition probabilities can be computed using the DDPM of the gates. Note that interconnects can also have DDPMs to account for crosstalk. In this paper, due to the lack of layout information, we assumed that there are no delay variations on interconnects. The overall deviation-based framework is however general and it can easily accommodate interconnect delay variations if layout information is available.

**Definition 1.** Let $P_E$ be the probability that a net has the expected signal-transition. The deviation on that net is defined by $\Delta = 1 - P_E$. The following rules are applied during the propagation of signal-transition probabilities:

1. If there is no output-signal transition (output keeps its logic value), then the deviation on the output net is 0.
2. If there are multiple inputs that can cause the expected signal-transition at the output of a gate, only the input-to-output path that causes the highest deviation at the output net is considered. The other inputs are treated as they have no effect on the deviation calculation (i.e., they are held at the non-controlling value).
3. When multiple inputs are required to change at the same time in order to provide the expected output transition, all required input-to-output paths of the gate are considered. Only the unnecessary (redundant) paths are discarded.

A key premise of this paper is that output deviations can be used to compare path lengths. As in the case of path delays, the net deviations also increase as the signal propagates through a sensitized path. This claim can be proven on the basis of the rules used to propagate the signal-transition probabilities.

**Example:** Fig. 2 shows signal-transition probabilities and their propagation for a simple circuit. The test stimuli and the expected fault-free transitions on each net are shown in dark boxes. The calculated signal-transition probabilities are shown in angled brackets ($\langle ... \rangle$). The DDPMs of the

**Table 2. Example DDPM for AND, XOR, INV**

| | | Initial Input State | | | |
|---|---|---|---|---|---|
| AND | **prob** | **00** | **01** | **10** | **11** |
| Inputs | **IN0** | 0.2 | 0.3 | 0 | 0.2 |
| | **IN1** | | 0 | 0.2 | 0.3 |
| XOR | **prob** | **00** | **01** | **10** | **11** |
| Inputs | **IN0** | 0.3 | 0.4 | 0.1 | 0.2 |
| | **IN1** | 0.3 | 0.4 | 0.2 | 0.4 |
| INV | **prob** | **0** | **1** | | |
| Inpus | **IN0** | 0.2 | 0.2 | | |

gates used in this circuit are given in Table 2. The entries in Table 2 are chosen arbitrarily.

The deviations are calculated based on the rules mentioned above for the example circuit in Fig. 2. For instance, there is no output change for net $E$. Thus for $E$, we have $\langle 1, 0, 0, 0\rangle$. On the other hand, on net $F$, the output changes due to `IN1` (net D) of XOR. There is a delay-defect probability of 0.4. It implies that with a probability of 0.4, the output will stay at `LOW` value. Therefore, for $F$, we have $\langle 0.4, 0.6, 0, 0\rangle$. When the calculations are continued until the primary output net Q1, one can find the deviation on Q1 as 0.76.

### 2.3 Implementation of Algorithm for Propagating Signal-Transition Probabilities

A depth-first procedure is used to compute signal-transition probabilities for large circuits. When we use a depth-first algorithm, only the nets that are required to find the output deviation on a specific observation point are processed. In this way, a smaller number of gate pointer stacking is required compared to the alternative of simulating the deviations starting from INs and tracing forward.

We first assign signal-transition probabilities to all INs. Then, we start from the observation points (outputs) and backtrace until we find a *Processed Net* (PN). A PN has all the signal transition probabilities assigned. The pseudocode for the algorithm is given in Algorithm 1.

If the number of test patterns is $k$ and the number of nets in the circuit is $N$, the worst-case time-complexity of the algorithm is $O(kN)$. However, since the calculation for each pattern is independent of other patterns, the algorithm can easily be made multi-threaded. In this case, if the number of threads is $T$, the complexity of the algorithm is reduced to $O(\frac{kN}{T})$.

### 2.4 Pattern-Selection Method

In this subsection, we describe how to use output deviations to select high-quality patterns from $n$–detect transition-fault patterns. The pattern-selection procedure is as follows:

- Determine the number of patterns to be selected. This can be a user input, e.g., $S$. The parameter $S$ can be set to number of 1-detect patterns, the number of timing-aware patterns, or any value that fits the user's test budget.
- Until the selected pattern number reaches $S$, select the largest-deviation patterns for each observation point one by one. If a pattern is previously selected, do not select it again.
- After selection, sort the patterns by the maximum deviation they create at an observation point (This step called pattern re-ordering).
- Run top-off transition delay-fault ATPG to increase the fault coverage if necessary.

**Algorithm 1** Propagate_Signal_Transition_Probability

1: Input: $Tran_Fault_Patterns\{P_0, ..., P_k\}$
2: Output: $Output_Deviations$
3: **for** $i = 0$ to $k$ **do**
4: reset all signal-transition probabilities
5: read pattern $P_i$
6: assign signal-transition probabilities to $INs$
7: reset stack
8: **for all** $OP$ =Observation Point **do**
9: **if** $OP$ is processed **then**
10: go to next $OP$
11: **end if**
12: trace backward until a processed net is found
13: add unprocessed gates on the traced path to the stack
14: trace forward
15: **for all** $G$ =gate in stack **do**
16: find signal-transition probabilities of the output net of $G$
17: remove $G$ from the stack
18: **end for**
19: find signal-transition probabilities of $OP$
20: **end for**
21: **end for**

## 3 Experimental Results

In this section, we present experimental results obtained for the IWLS and ISCAS89 benchmark circuits. We first describe how we obtained DDPs for the logic gates in these benchmarks (Section 3.1). Next, we provide details of the experimental setup (Section 3.2). Finally, we present the simulation results (Sections 3.3 and 3.4).

### 3.1 Finding Gate-Delay Defect Probabilities

In order to determine the gate-DDPs, we ran HSpice Monte Carlo (MC) simulations using 45 $nm$ process-technology BSIM4 predictive transistor models. For each gate type, we simulated the schematic under various loading capacitance and input slew rate conditions to account for spatial correlations.

MC simulations were done using the following realistic process-variation parameters (obtained from a current technology) for a Gaussian distribution: Transistor gate length $L : 3\sigma = 10\%$, threshold voltage $V_{TH} : 3\sigma = 30\%$, and gate-oxide thickness $t_{OX} : 3\sigma = 3\%$. For each configuration, 50 MC simulations were performed for each possible input transition. For each gate, a transition-delay value greater than $T_{CRT} = NOM + 2ps$ constitutes a DDP value for the respective input transition. The parameter $X = 2ps$ is selected in such a way that all the gate instances have at least one non-zero DDPM entry. $X$ is the minimum of all `MAX` delays (among all gates). This corresponds to the maximum delay of an inverter driving a single inverter load. Note that selecting too large a value for $X$ may cause many DDPMs to have all-zero entries, simply because the gate would never have a delay larger than $NOM + X$. Ideally, $X$ can be set to the $MAX$ delay specified by a Static Timing Analysis tool, which does not consider process variation effects.

## 3.2 Experimental Setup

All experiments were performed on a state-of-the-art server with 8 processors running Linux at 3 GHz frequency and 12GB memory. The program to compute output deviations was implemented using C++. Perl scripts were used to generate the simulation input files. Mentor Graphics FastScan [9] was used to generate $n$-detect and the timing-aware transition delay-fault test patterns. Path delays are calculated using an in-house dynamic path-timing simulator. All the flip-flops in the benchmarks were replaced with muxed-DFF scan flops. FastScan was forced to generate Launch-on-capture (LOC) transition fault patterns. The PI change during capture cycles and the observation of POs is prevented in order to simulate realistic test environments. For timing-aware ATPG, the SDF file was generated using the results of the MC simulations. At this point, all wire delays were ignored. For all simulations and during pattern generation, only one CPU was used, i.e., multi-threading was not enabled.

## 3.3 Correlation Between Output Deviations and Path Lengths

We ran correlation analysis to determine the relationship between output deviations and sensitized path lengths. For each benchmark, we found output deviations and path lengths for $n$-detect transition-fault test-patterns ($n = 1, 3, 5, 8, 10$). We simulated these patterns using the in-house path-length calculator and determined the signal delays at the observation points of the benchmarks. Next, we used Matlab to compute the Kendall's correlation coefficients [3] for each pattern set. Table 3 shows the average correlation coefficients for the patterns in a 1-detect test set of the ISCAS'89 and IWLS'2005 benchmarks [5].

**Table 3. Kendall's coefficients for evaluating the correlation of path lengths to output deviations**

| Benchmark | (Ave,Min,Max) | Benchmark | (Ave,Min,Max) |
|---|---|---|---|
| s9234 | (**0.97**, 0.87, 1) | des_perf | (**0.98**, 0.97, 0.99) |
| s13207 | (**0.96**, 0.92, 0.99) | mem_ctrl | (**0.96**, 0.91, 0.99) |
| s15850 | (**0.97**, 0.91, 0.99) | pci_bridge32 | (**0.93**, 0.85, 0.99) |
| s35932 | (**0.90**, 0.72, 0.94) | systemcaes | (**0.96**, 0.82, 1) |
| s38417 | (**0.98**, 0.96, 0.99) | tv80 | (**0.95**, 0.81, 1) |
| s38584 | (**0.96**, 0.86, 0.99) | usb_funct | (**0.97**, 0.85, 0.99) |
| ac97_ctrl | (**0.98**, 0.93, 1) | aes_core | (**0.97**, 0.9, 0.99) |

The results for other values of $n$ are similar. The minimum and maximum values of the correlation coefficients are also given. As seen in Table 3, there is a strong positive correlation (close to the perfect correlation measure of 1) between output deviations and path lengths. Thus, the method of output deviations is a promising metric for evaluating the capability of transition delay-test patterns to sensitize long paths.

## 3.4 Pattern Selection Results

In this section, we present the pattern-selection results using the number of excited long paths as an evaluation metric. For each benchmark, we found the longest path delay and ranked any path with a delay of at least 70% of the clock period as a long path. Fig. 3 shows the normalized number of excited long paths for the ISCAS'89 and IWLS'2005 benchmarks for a range of $n$-detect and the timing-aware ATPG patterns. It can be easily seen that, as $n$ increases, a larger number of long paths are excited. Timing-aware ATPG pattern-set does only marginally better than the 1-detect pattern-set, and for some benchmarks (s13207, s38417), even worse than 1-detect pattern set. This is not completely unexpected because the pattern counts for large values of $n$ are higher than that for timing-aware ATPG.

We applied the pattern selection algorithm described in Section 2.4 and selected $S$ patterns from each $n$-detect pattern-set, where $S$ is equal to the number of timing-aware patterns. The results are shown in Fig. 4. For many benchmarks, the output-deviation-based selection method finds patterns that excite a larger number of long paths compared to a proprietary timing-aware ATPG method. For the benchmark systemcaes, the patterns selected from a 10-detect timing-unaware test set excite 1.6x more long paths than timing-aware ATPG. For s15850, this ratio is 1.4. Note that the pattern count is the same in each case. As an alternative selection method, we used randomly selected test patterns from 10-detect TDF pattern sets. The results for the random selection are also shown in Fig. 4. Deviation-based pattern selection clearly outperforms random selection. For most circuits, patterns selected on the basis of output deviations excite more long paths than timing-aware ATPG

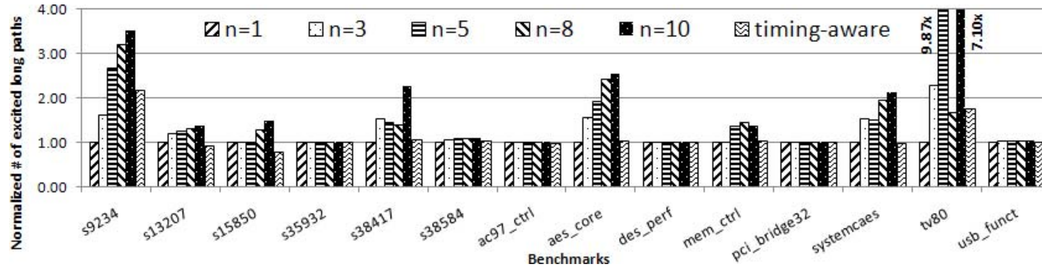


**Figure 3. Normalized number of excited long paths for each benchmark for the original pattern-sets.**

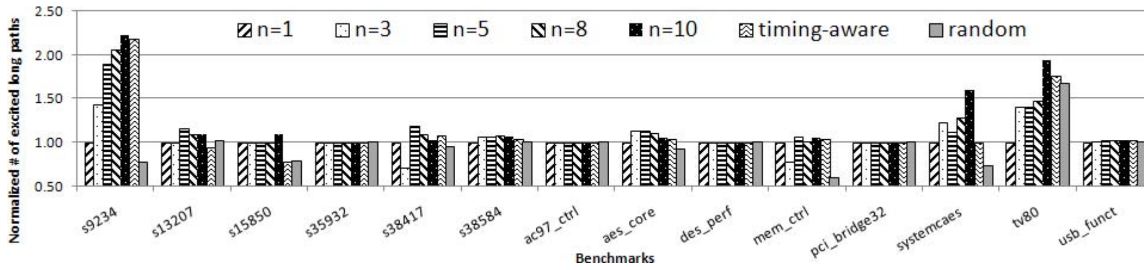


**Figure 4. Normalized number of excited long paths for each benchmark for the pattern-sets selected using the algorithm described in Section 2.4.**

patterns. For example, for s13207, timing-aware ATPG patterns excite 58 out of the 86 long paths, whereas the the deviation-based patterns excite 70 long paths. For the larger tv80 benchmark, timing-aware ATPG patterns excite only 1482 out of the 5965 long paths; the proposed method does significantly better, exciting 1632 long paths, but obviously more patterns are needed to detect SDDs on more long paths. The proposed method, which offers the flexibility of pattern selection from larger repositories, can therefore be used to increase the "long path" coverage.

In order to evaluate the fault-detection performance of our selection method and the timing-aware ATPG, we ran fault injection simulations. For each benchmark, we inserted 1000 delay faults on randomly chosen long paths to model resistive shorts and opens. The additional delay caused by each fault is slightly more than the slack for the corresponding path. The percentage of detected faults is given in Fig. 5 for the selected patterns from various $n$-detect pattern sets and timing-aware ATPG. Fig. 6 shows how the fault coverage increases with the number of patterns for three benchmark circuits. Similar results are obtained for the other benchmarks, but they are not included here because of lack of space. For the deviation-based method, the patterns are ordered on the basis of the deviation metric, while for timing-aware ATPG, they are considered in the same order as reported by the ATPG tool. We find that the coverage rises more steeply for the proposed method; moreover, higher cumulative fault coverage is obtained over all the patterns (for the same pattern count). Fig. 5 also shows that the coverage provided by timing-aware ATPG is very low for circuits such as pci_bridge32 and systemcaes. For example, it is less than 20% for pci_bridge32.

The total CPU time for timing-unaware $n$-detect TDF ATPG, deviation computation, and deviation-based pattern-selection is expected to be much less for large circuits compared to timing-aware TDF ATPG. As shown in Fig. 1, the ATPG time is much higher for timing-aware ATPG, and the procedures for deviation computation and pattern selection are polynomial in circuit size. For the much smaller benchmark circuits considered in this paper, the CPU times for the two approaches are comparable, but we are comparing an unoptimized academic implementation to a highly optimized commercial tool. For similar optimized implementations, the method will need less CPU time because, unlike timing-aware ATPG, it does not perform path enumeration.

## 4 Conclusions

We have presented a test-grading technique, based on output deviations, for screening small-delay defects (SDDs). We have defined the concept of output deviations for pattern-pairs and shown that it can be used as an efficient surrogate metric to model the effectiveness of transition delay-fault (TDF) patterns for SDDs. We have introduced a gate-delay defect probability measure to model delay variations for nanometer technologies. Experimental

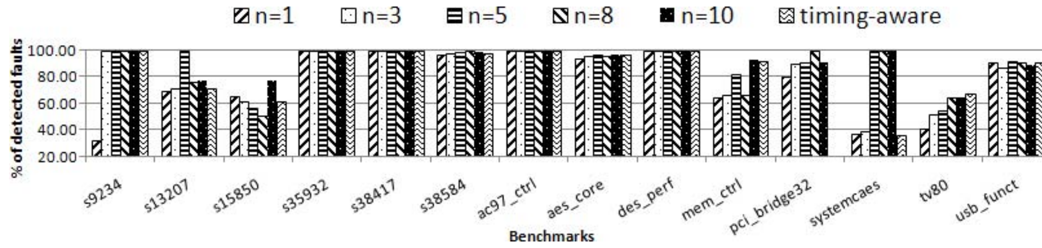


**Figure 5. The percentage of faults that are detected using the selected patterns and timing-aware ATPG patterns.**

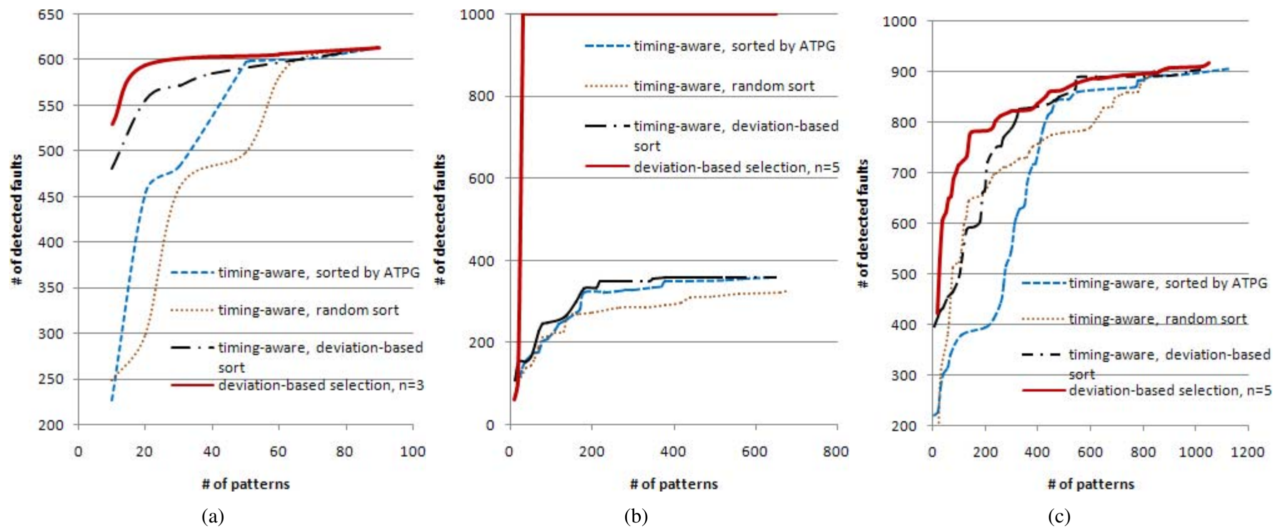


**Figure 6. Fault detection results for: (a) s15850; (b) systemcaes; (c) usb_funct.**

results for the ISCAS'89 and the IWLS'2005 benchmark circuits show that the proposed method intelligently selects the best set of patterns for SDD detection from an n-detect TDF pattern set generated using a timing-unaware ATPG tool, and it excites a larger number of long paths compared to previously proposed timing-aware ATPG methods. We have also shown that, for the same pattern count, the selected patterns are more effective than timing-aware ATPG for detecting small delay defects caused by resistive shorts, resistive opens, and process variations.

## References

[1] N. Ahmed, M. Tehranipoor, and V. Jayaram. Timing-based delay test for screening small delay defects. In *Proc. of IEEE Design Automation Conf.*, pages 320–325, 2006.

[2] Cadence Inc. Encounter test - test generation and simulation reference. Product Version 3.0, 2005.

[3] B. J. Chalmers. *Understanding Statistics*. CRC Press, 1987.

[4] ITRS. http://www.itrs.net/reports.html.

[5] IWLS 2005 Benchmarks. http://iwls.org/iwls2005/benchmarks.html.

[6] R. Kapur, J. Zejda, and T. Williams. Fundamentals of timing information for test: How simple can we get? In *Proc. of IEEE ITC*, 2007.

[7] H. Li, Z. Li, and Y. Min. Reduction of number of paths to be tested in delay testing. *JETTA*, 16(5):477–485, 2000.

[8] X. Lin, J. Rajski, I. Pomeranz, and S. Reddy. On static test compaction and test pattern ordering for scan designs. In *Proc. of IEEE ITC*, pages 1088–1097, 2001.

[9] Mentor Graphics. Understanding how to run timing-aware ATPG. Application Note, 2006.

[10] R. Putman and R. Gawde. Enhanced timing-based transition delay testing for small delay defects. In *Proc. of IEEE VTS*, pages 336–342, 2006.

[11] J. Waicukauski, E. Lindloom, B. Rosen, and V. Iyengar. Transition fault simulation. *IEEE Design and Test of Computers*, pages 32–38, 1987.

[12] Z. Wang and K. Chakrabarty. Test-quality/cost optimization using output-deviation-based reordering of test patterns. *IEEE Tran. on CAD*, 27:352–365, Feb 2008.

[13] X. Lin *et al*. Timing-aware ATPG for high quality at-speed testing of small delay defects. In *Proc. of IEEE ATS*, pages 139–146, 2006.