Adaptation and Evaluation of the Output-Deviations Metric to Target Small-Delay Defects in Industrial Circuits

Mahmut Yilmaz[†], Krishnendu Chakrabarty[‡], and Mohammad Tehranipoor[§]

[†]Advanced Micro Devices, Inc.
Sunnyvale, Calif., USA
mahmut.yilmaz@amd.com

[‡]Dept. Electrical and Comp. Eng.
Duke University, Durham, N.C.
krish@ee.duke.edu

[§]Dept. Electrical and Comp. Eng.
University of Connecticut, Storrs, Conn.
tehrani@engr.uconn.edu

Abstract

Timing-related defects are a major cause for test escapes and field returns for very-deep-sub-micron (VDSM) integrated circuits (ICs). Small-delay variations induced by crosstalk, process variations, power-supply noise, and resistive opens and shorts can cause timing failures in a design, thereby leading to quality and reliability concerns. We present the industrial application and case study of a previously proposed test-grading technique that uses the method of output deviations for screening small-delay defects (SDDs). The technique is shown to have significantly lower computational complexity and test pattern count, without loss of test quality, compared to a commercial timing-aware automatic test pattern generation (ATPG) tool.

I. 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 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].

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

The transition delay-fault (TDF) [2] model attempts to propagate the lumped-delay defect of a gate by logical transitions to the observation points or state elements. The TDF model is not effective for SDDs because test generation using TDFs leads to the excitation of short paths [1], [3]. Park *et al.* first proposed a statistical method for measuring the effectiveness of delay-fault automatic test-pattern generation (ATPG) [4]. The proposed technique is especially relevant today and it can handle process variations on sensitized paths; however, this work is limited in the sense that it only provides a metric for delay-test coverage, and it does not aim to generate or select effective patterns.

Due to the growing interest in SDDs, the first commercial timing-aware ATPG tools were introduced recently, e.g., new versions of Mentor Graphics FastScan, Cadence Encounter Test, and Synopsys TetraMax. These tools attempt to make ATPG patterns more effective for SDDs by exercising longer paths or applying patterns at higher-than-rated clock frequencies. However, only a limited amount of timing information is supplied to these tools, either via standard delay format (SDF) files (for FastScan and Encounter Test) or through a static timing analysis (STA) tool (for TetraMax). As a result, none of these tools can be easily extended to take into account process

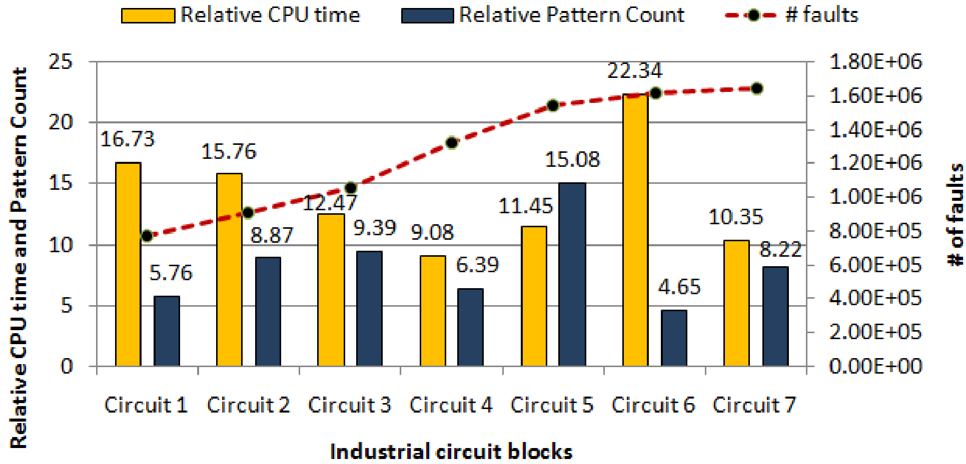


Fig. 1. CPU time and pattern count of timing-aware ATPG relative to traditional TDF ATPG for sub-blocks of an industrial microprocessor circuit.

variations, crosstalk, power-supply noise, or similar SDD-inducing effects on path delays. These tools simply rely on the assumption that the longest paths (determined using STA or SDF data) in a design are more prone to failure due to SDDs. Moreover, 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 two ATPG tools from the same EDA company: (***i***) timing-unaware ATPG, i.e., a traditional TDF pattern generator and (***ii***) timing-aware ATPG that takes timing information into account. The results are shown for some representative AMD microprocessor functional blocks. It can be seen from Fig. 1 that as much as 22 times greater CPU-time and 15 times greater pattern count are observed. The numbers are normalized such that the run-time and pattern count of timing-unaware ATPG is taken to be one unit.

Statistical static timing analysis (SSTA) can generate variability-aware delay data. Although a complete SSTA flow takes considerable computation time [5], [6], simplified SSTA-based approaches can be used for pattern selection, as shown in [7], [8]. In [7], the authors propose an SSTA-based test pattern quality metric for to detect SDDs. The computation of the metric requires multiple dynamic timing analysis runs for each test pattern using randomly sampled delay data from Gaussian pin-to-pin delay distributions. The proposed metric is also used for pattern selection. In [8], the authors focus on timing hazards and propose a timing hazard-aware SSTA-based pattern selection technique.

The complexity of today's ICs and shrinking process technologies are also leading to prohibitively high test data volumes. For example, the test data volume for TDFs is 2-5 times higher than that for stuck-at faults [9], and it has also been demonstrated that test patterns for such sequence- and timing-dependent faults are more important for newer technologies [10]. The 2007 International Technology Roadmap for Semiconductors (ITRS) predicted that the test data volume for integrated circuits will be as much as 38 times larger and the test application time will be about 17 times longer in 2015 than it was in 2007. Therefore, efficient generation, pattern grading, and selection methods are required to reduce the total pattern count while effectively targeting SDDs.

Recently, a statistical approach based on the concept of output deviations has been presented to select the most effective test patterns for SDD detection [11]. It has been shown that, for academic benchmark circuits, there is high correlation between the output deviations measure for test patterns and the sensitization of long paths under process variation. Compared to a commercial timing-aware ATPG tool, significant reductions in CPU time and pattern count and higher test quality have been reported for these benchmark circuits. However, prior work has not examined the applicability or effectiveness of this approach for realistic industrial circuits.

This paper presents the adaptation of the output-deviation metric to industrial circuits. The framework of output deviations was enhanced to make it applicable for such circuits. Experimental results show that the proposed method can effectively select the highest quality patterns from large test sets that cannot be used in their entirety

for production test environments with tight pattern-count limits. Unlike common industry ATPG practices, it also considers delay faults caused by process variations. The proposed approach is shown to incur negligible run-time penalty. For two important quality metrics, namely coverage of long paths and long-path coverage ramp-up, it is shown to outperform a commercial timing-aware ATPG tool for the same pattern count.

In remainder of this paper, Section II provides an overview of the output deviations method and describes the adaptation of the probabilistic fault model and the output deviations metric. In Section III, we evaluate the proposed method on industrial circuit blocks using n-detect TDF and timing-aware ATPG test sets. Section IV concludes the paper.

II. Probabilistic Delay-Fault Model and Output Deviations for SDDs

In this section, we first review the concept of output deviations (Section II-A). Next, we describe how the deviation-based pattern-selection method of earlier work was extended to industrial circuits (Section II-B).

A. Overview of Output Deviations

The concepts of *gate-delay defect probabilities* (DDPs) and *signal-transition probabilities* (STPs) were introduced in [11]. These probabilities introduce the notion of confidence levels for pattern pairs.

In this section, we first review the concept of gate-delay defect probabilities (DDPs) (Section II-A1) and signal-transition probabilities (Section II-A2). 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 II-A2). We describe the algorithm used for signal-probability propagation (Section II-A3). Finally, we describe how test patterns can be ranked and selected from a large repository (Section II-A4).

1) Gate-Delay Defect Probabilities: DDPs are assigned to each gate in a design. DDPs for a gate are provided in the form of a matrix called the *delay defect probability matrix* (DDPM). The DDPM for a two-input OR gate is shown in Table I. 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.

Assume that the inputs are shown in the order of `IN0, IN1`. If there is an input transition from '`10`' to '`00`', the corresponding DDPM column is '`10`'. Since the transition is caused by `IN0`, the corresponding DDPM row is `IN0`. As a result, the DDP value corresponding to this event is `0.5`, showing the probability that corresponding output transition is delayed beyond a threshold.

For initial state '`11`', both inputs should switch simultaneously to have an output transition. Corresponding DDPM entries are merged due to this requirement. The entries in Table I have been chosen arbitrarily for the sake of illustration. The real DDPM entries are much smaller than the ones shown in this example.

TABLE I
An example of DDPM for a two-input OR gate: (a) Old style; (b) New style

(a)

		Initial Input State			
	prob	**00**	**01**	**10**	**11**
Inputs	**IN0**	0.21	0	0.5	0.11
	IN1	0.12	0.20	0	

(b)

IO Path	$L \rightarrow H$		$H \rightarrow L$	
	μ	σ^2	μ	σ^2
IN0 $\rightarrow$ **Z**	100ps	112	110ps	97
IN1 $\rightarrow$ **Z**	110ps	123	120ps	117

For an N-input gate, the DDPM consists of $N \cdot 2^N$ entries, each holding one probability value. If the gate has more than one output, each output of the gate has a different DDPM. Note that the DDP is 0 if the corresponding event cannot provide an output transition. Consider DDPM(2,3) in Table I. When the initial input state is '`10`', no change in `IN1` can cause an output transition, because the OR gate output is already at high state, and even if `IN1` switches to high (`1`), this will not cause an output transition.

We next discuss how a DDPM is generated. Each entry in DDPM indicates the probability that the delay of a 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:

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

For instance, if we assume a Gaussian delay distribution for all gates (with mean μ) and set the critical delay value to $\mu + X$, 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 may have a different mean (μ) and standard deviation (σ).

The delay distribution can be obtained in different ways: (*i*) using the delay information provided by an SSTA-generated 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 (3σ) 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.

The generation of the DDPMs is not the main focus of this paper. We consider DDPMs to be analogous to timing libraries. Our goal is not to develop the most effective techniques for constructing DDPMs; rather, we are using such statistical data to compute deviations and use them for pattern grading and pattern selection. In a standard industrial flow, statistical timing data can be developed by specialized timing groups, so the generation of DDPMs is a pre-processing step and an input to the ATPG-focused test-automation flow.

We have also seen that small changes in the DDPM entries have negligible impact on the pattern-selection results. We attribute this finding to the fact that any DDPM changes affect multiple paths in the circuits, so their impact is amortized over the circuit and the test set. The absolute values of the output deviations are less important than the relative values for different test patterns. Detailed results are presented in [13].

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, in a vector form ($< ... >$): $< P_{L\to L}, P_{L\to H}, P_{H\to L}, P_{H\to H} >$. We refer to this vector as the signal-transition probability (STP) vector. 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 STPs, corresponding to the applied transition test pattern, equal to 1. All the other STPs for INs are set to 0. When signals are propagated through several levels of gates, the STPs 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 only focus on variations' impact on gate delay. The overall deviation-based framework is, however, general and it can easily accommodate interconnect delay variations if layout information is available, as reported in [14].

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 STPs:

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 if 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 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, a property that follows from the rules used to calculate STPs for a gate output. This claim is formally proven next.

Lemma 1. *For any net, let the STP vector be given by* $< P_{L\to L}, P_{L\to H}, P_{H\to L}, P_{H\to H} >$. *Among these four probabilities, i.e.,* $< P_{L\to L}, P_{L\to H}, P_{H\to L}, P_{H\to H} >$, *at least one is non-zero and at most two can be non-zero.*

Proof: If there is no signal-value change (the event $L \to L$ or $H \to H$), the expected STP is 1 and all other probabilities are 0. If there is a signal-value change, only the expected signal-transition events and the delay-fault case have non-zero probabilities associated with them. The delay-fault case for an expected signal value change of $L \to H$ is $L \to L$ (the signal value does not change because of a delay-fault). Similarly, the delay-fault case for an expected signal value change of $H \to L$ is $H \to H$. ■

Theorem 1. *The deviation on a net always increases or stays constant on a sensitized path if the signal-probability propagation rules are applied.*

Proof: Consider a gate with K inputs and one output. The signal-transition on the output net depends on one of the following cases. From Lemma 1, we note that only two cases need to be considered.

(*i*) Only one of the input-port signal-transitions is enough to create the output signal-transition.

(*ii*) Multiple input-port signal transitions are required to create the output signal-transition.

Let $P_{OUT,j}$ be the probability that the gate output makes the expected signal transition for a given pair of patterns on input j, where $1 \leq j \leq K$. Let $\Delta_{OUT,j} = 1 - P_{OUT,j}$ be the deviation for the net corresponding to the gate output.

Case (*i*): Consider a signal transition on input j. Let Q_j be the probability of occurrence of this transition. Let d_j be the entry in the gate's DDPM that corresponds to the given signal transition on j. The probability that the output makes a signal transition is given by:

$$P_{OUT,j} = Q_j(1 - d_j). \tag{2}$$

We assume here that an error at a gate input is independent from the error introduced by the gate. Note that $P_{OUT,j} \leq Q_j$ since $0 \leq d_j \leq 1$. Therefore, the probability of getting the expected signal transition decreases and the deviation $\Delta_{OUT,j} = 1 - P_{OUT,j}$ increases (or does not change) as we pass through a gate on a sensitized path. The overall output deviation Δ^*_{OUT} on the output net is calculated as:

$$\Delta^*_{OUT} = \max_{i \leq j \leq K} \{\Delta_{OUT,j}\}. \tag{3}$$

Case (*ii*): Suppose L input ports ($L > 1$), indexed 1,2,...,L, are required to make a transition for the gate output to change. Let $d^*_{max} = max_{1\leq j \leq L}\{d_j\}$. The output deviation for the gate in this case is defined as:

$$\Delta^*_{OUT} = \prod_{i=1}^{L} P_{OUT,i} \cdot (1 - d^*_{max}). \tag{4}$$

Note that $\Delta^*_{OUT} \leq P_{OUT,i}, 1 \leq i \leq L$, since $0 \leq d^*_{max} \leq 1$. Therefore, we conclude that the probability of getting the expected transition on a net either decreases or remains the same as we pass through a logic gate. In other words, the deviation is monotonically non-decreasing along a sensitized path. ■

Example: Fig. 2 shows STPs 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 STPs are shown in angled brackets ($\langle ... \rangle$). The DDPMs of the gates (OR, AND, XOR, and INV) used in this circuit are given in Tables I and II. The entries in both tables are chosen arbitrarily.

In the following example, the deviations are calculated based on the rules mentioned above for the example circuit in Fig. 2:

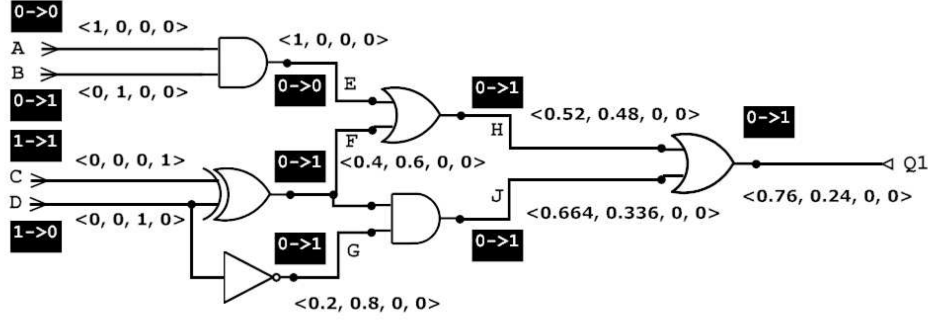


Fig. 2. An example to illustrate the propagation of STPs through the gates of a logic circuit.

TABLE II
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**		
Inputs	**IN0**	0.2	0.2		

- Net E: There is no output change, which implies that E has the STP $\langle 1, 0, 0, 0\rangle$.
- 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, i.e., the STP for net F is $\langle 0.4, 0.6, 0, 0\rangle$.
- Net G: Output changes due to `IN0` (net D) of INV, i.e., the STP for net G is $\langle 0.2, 0.8, 0, 0\rangle$.
- Net H: Output changes due to `IN1` (net F) of OR.
  - ◇ If `IN1` stays at `LOW`, output does not change. Therefore, the STP for net H is $0.4 \odot \langle 1, 0, 0, 0\rangle$, where $\odot$ denotes the dot product;
  - ◇ If `IN1` goes to `HIGH`, output changes with a DDP of 0.2, i.e., the STP for net H is $0.6 \odot \langle 0.2, 0.8, 0, 0\rangle$;
  - ◇ Combining all the above cases, the STP for net H is $\langle 0.52, 0.48.0, 0\rangle$
- Net J: Output changes due to both `IN0` (net F) and `IN1` (net G) of AND (both required).
  - ◇ If both stay at `LOW`, output does not change, which implies that J has the STP $0.4 \odot 0.2\langle 1, 0, 0, 0\rangle$;
  - ◇ If one of them stays at `LOW`, output does not change, i.e., the STP for net J is $0.4 \odot 0.8\langle 1, 0, 0, 0\rangle + 0.6 \odot 0.2\langle 1, 0, 0, 0\rangle$;
  - ◇ If both go to `HIGH`, the output changes with a DDP. Since both inputs change, we use the maximum DDP, i.e., the STP for net J is $0.6 \odot 0.8 \odot \langle 0.3, 0.7, 0, 0\rangle$;
  - ◇ Combining all the above cases, the STP for net J is $\langle 0.664, 0.336, 0, 0\rangle$.
- Net Q1: The output changes due to only one of the inputs of OR. We need to calculate the deviation for both cases and select the one that causes maximum deviation at the output (Q1).
  - ◇ For `IN0` (net H) of OR:
    - ◦ If `IN0` stays at `LOW`, the output does not change, i.e., the STP for net Q1 is $0.52 \odot \langle 1, 0, 0, 0\rangle$;
    - ◦ If `IN0` goes to `HIGH`, the output changes with a DDP, i.e., the STP for net Q1 is $0.48 \odot \langle 0.5, 0.5, 0, 0\rangle$;
    - ◦ Combining all the above cases, the STP for net Q1 is $\langle 0.76, 0.24, 0, 0\rangle$.
  - ◇ For `IN1` (net J) of OR:
    - ◦ If `IN1` stays at `LOW`, the output does not change, i.e., the STP for net Q1 is $0.664 \odot \langle 1, 0, 0, 0\rangle$;

Procedure: Propagate Probability (t_i)
1: reset all signal-transition probabilities
2: read pattern t_i
3: assign signal-transition probabilities to INs
4: reset stack
5: **for all** observation points PO_j, $j = 1, 2, ..$ **do**
6:   **if** PO_j is processed **then**
7:     go to next PO_j
8:   **end if**
9:   trace backward until a processed net is found
10:   add unprocessed gates on the traced path to the stack
11:   **for all** G =gate in stack **do**
12:     find signal-transition probabilities of the output net of G
13:     remove G from the stack
14:   **end for**
15:   find signal-transition probabilities of PO_j
16: **end for**

Fig. 3. Signal-transition probability propagation algorithm for calculating output deviations.

- ◦ If IN1 goes to HIGH, the output changes with a DDP, i.e., the STP for net Q1 is $0.336 \odot \langle 0.2, 0.8, 0, 0 \rangle$;
- ◦ Combining all the above cases, the STP for net Q1 is $\langle 0.7312, 0.2688, 0, 0 \rangle$.

⋄ Since IN0 provided the higher deviation, we finally conclude that the STP for net Q1 is $\langle 0.76, 0.24, 0, 0 \rangle$. Hence, the deviation on Q1 is 0.76.

3) Implementation of Algorithm for Propagating Signal-Transition Probabilities: A depth-first procedure is used to compute STPs 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 STPs 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 Fig. 3.

If the number of test patterns is N_s and the number of nets in the circuit is N_n, the worst-case time-complexity of the algorithm is $O(N_s \cdot N_n)$. However, since the calculation for each pattern is independent of other patterns (we assume full-scan designs in this paper), 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 $O(\frac{N_s \cdot N_n}{T})$.

4) Pattern-Selection Method: In this subsection, we describe how to use output deviations to select high-quality patterns from an n–detect transition-fault pattern set. The number of test patterns to be selected is a user input, e.g., S. The parameter S can be set to the number of 1-detect timing-unaware patterns, the number of timing-aware patterns, or any other value that fits the user's test budget.

In our pattern-selection method, we target topological coverage as well as long-path coverage. As a result, we attempt to select patterns that sensitize a wide range of distinct long paths. In this process, we also discard low-quality patterns to find a small set of high-quality patterns.

For each test observation point PO_j, we keep a list of N_p most effective patterns in EFF_j (Fig. 4, lines 1-3). The patterns in EFF_j are the best unique-pattern candidates for exciting a long path through PO_j. During deviation computation, no pattern (t_i) is added to EFF_j if the output deviation at PO_j is smaller than a limit ratio (D_{LIMIT}) of the maximum instantaneous output deviation (Fig. 4, line 10). (D_{LIMIT}) can be used to discard low-quality patterns. If the output deviation is larger than this limit, we first check whether we have added a pattern to EFF_j with the same deviation (Fig. 4, line 11). It is unlikely that two different patterns will create the same output deviation on the same output PO_j while exciting different non-redundant paths. Since we want a higher topological path-coverage, we skip these cases (Fig. 4, line 11). Although this assumption may not necessarily be true, we assume for the sake of completeness that it holds for most cases. If we observed a unique deviation on PO_j, we first check whether EFF_j is full (already includes N_p patterns); see Fig. 4, line 12. Pattern t_i is added

```
Procedure: Compute Deviations (t_0, ..., t_{N_s}, N_p)
 1: for all observation point PO_j, j = 1, 2, .. do
 2:    create list EFF_j[N_p]
 3: end for
 4: Max_Dev = 0;
 5: for all test pattern t_i, i = 1, 2, ..., N_s do
 6:    Propagate Probability(t_i);
 7:    for all observation point PO_j, j = 1, 2, .. do
 8:       Dev = deviation of PO_j;
 9:       if Dev > Max_Dev then Max_Dev = Dev;
10:       if Dev > D_LIMIT·Max_Dev then
11:          if EFF_j includes Dev then Next observation point;
12:          if EFF_j is not full then
13:             add t_i and Dev to EFF_j;
14:          else if Dev > min(EFF_j) then
15:             remove min(EFF_j);
16:             add t_i and Dev to EFF_j;
17:          end if
18:       end if
19:    end for
20: end for
```

Fig. 4. Deviation-computation algorithm for pattern selection.

```
Procedure: Select Patterns D_S(t_0, ..., t_{N_s}, S, D_LIMIT)
 1: list D[N_s];
 2: init D to all 0s;
 3: for all test pattern t_i, i = 1, 2, ..., N_s do
 4:    for all observation point PO_j, j = 1, 2, ... do
 5:       if EFF_j includes t_i and deviation of t_i > D_LIMIT·
          Max_Dev then
 6:          increment D[i];
 7:       end if
 8:    end for
 9: end for
10: sort D by values;
11: D_S = select top S patterns;
12: return D_S;
```

Fig. 5. Pattern selection algorithm.

to EFF_j along with its deviation if EFF_j is not full or if t_i has a larger deviation than the minimum deviation stored in EFF_j (Fig. 4, lines 12-17). The effectiveness of a pattern is measured by the number of occurrences of this pattern in EFF_j for all values of j. For instance, if at the end of deviation computation, pattern A was included in the EFF list for 10 observation points, and pattern B was listed in the EFF list for eight observation points, we conclude that pattern A is more effective than pattern B.

Once the deviation computation is completed, the list of pattern effectiveness is generated and the final pattern filtering and selection is carried out (Fig. 5). First, pattern effectiveness is generated (Fig. 5, lines 1-9). Since `Max_Dev` is updated on the fly, we may miss some low-quality patterns. As a result, we need to filter by `Max_Dev` (D_{LIMIT}) again to discard low-quality patterns from the final pattern list (Fig. 5, line 5). Setting D_{LIMIT} to a high value may result in discarding most of the patterns, leaving only the best few patterns. Depending on D_{LIMIT}, the number of remaining patterns can be less than S. In the next stage, the patterns are re-sorted by their effectiveness (Fig. 5, line 10). Finally, until the selected pattern number reaches S or all patterns are selected, the top patterns are selected (Fig. 5, line 11). The computational complexity of the selection algorithm is $O(N_s p)$, where N_s is the number of test patterns and p is the number of observation points. This procedure is very fast since it only includes two nested for loops and a simple list-item existence check.

B. Adaptation to Industrial Circuits

The input data required to compute output deviations has to be revisited so that appropriate information is provided and the proposed approach can be used with the data that is available in an industrial project.

The two most significant inputs required by the previously proposed output deviations method are the gate and interconnect delay variations, and T_{CRT} for gates and interconnects. Delay variations for library gates are typically computed by a timing group, based on design for manufacturability (DFM) rules. The available data is the input-to-output delay values for worst, nominal, and best conditions. Delay variations for interconnects are computed based on DFM rules defining the range of resistance and capacitance variations for different metal layers and vias.

The main challenge in practice is to find a specific T_{CRT} for gates and interconnects. Defining a T_{CRT} for individual gates and interconnects is not feasible for industrial circuits, because allowed delay ranges are defined at the circuit or subcircuit levels. Due to this limitation, it was not possible to generate DDPM tables for gates and interconnects. As a result, we redefined the manner in which output deviations are computed. We first assumed independent Gaussian delay distributions for each path segment, i.e., gates and interconnects, where nominal delay is used as the mean, and the worst-case delay is used as 3σ. Instead of using specific probability values in DDPMs, we used mean delay and variances for each gate instance and interconnect. An example of the new DDPM (with entries chosen arbitrarily) for a 2-input OR gate is shown in Table I(b). The rows in the matrix correspond to each input-to-output timing arc and the columns correspond to the mean (μ) and variance (σ^2) of the corresponding $L \rightarrow H$ (rising) or $H \rightarrow L$ (falling) output transition.

Similarly, instead of propagating the STPs, we propagated the mean delay and variance on each path using the central limit theorem (CLT) [15], similar to the method proposed by Park *et al.* [4].

Since independent Gaussian distributions are assumed, we can use Equations (5) and (6) to calculate the mean value and standard deviation of the path PDF [15].

$$\mu_c = \sum_{i=1}^{N} \mu_i \tag{5}$$

$$\sigma_c = \sqrt{\sum_{i=1}^{N} \sigma_i^2} \tag{6}$$

where μ_i and σ_i are the *pdf* mean value and standard deviation of segment i, respectively; μ_c and σ_c are the path PDF mean value and standard deviation, respectively; N is the number of path segments.

Even if Gaussian distributions are not assumed for each delay segment, as long as segment delays are independent distributions with finite variances, the CLT [15] ensures that the path delay, which is the sum of segment delays, converges to a Gaussian distribution [15].

We defined T_{CRT} for the circuit as a fraction of the functional clock period (T_{func}) of the circuit. In our experiments, we used the values $0.7 \cdot T_{func}$, $0.8 \cdot T_{func}$, $0.9 \cdot T_{func}$, and T_{func}. For each case, the output deviation is defined as the probability that the calculated delay on an observation point (scan flip-flops or primary outputs) is larger than T_{CRT}.

The pattern selection method was also adapted. We introduced a degree of path enumeration to the pattern selection procedure. This change was implemented to ensure that all patterns exciting the delay-sensitive paths are selected. We developed an in-house tool to list all the sensitized paths for a TDF pattern, in addition to each segment of the sensitized paths. This tool enumerates all paths sensitized by a given test pattern. The steps in this flow are:

- Commercial tools were used for ATPG and fault simulation.
- For each pattern, the ATPG tool reports the detected TDFs. This report includes the net name, as well as the signal transition type, i.e., falling or rising.
- Our in-house tool finds the active nets, i.e., nets that have a signal transition on them, in the circuit under test. This step has O(log N) wost-case time complexity.

- Starting from scan flip-flops and primary inputs, each net with a detected fault is traced forward. Note that, if a fault is detected, it means that this net reaches a scan flip-flop through other nets with detected faults. If the sensitized path has no branching on it, the complexity of this step is O(N). However, if there are K branches on the sensitized net, and if all these branches create a different sensitized path, the complexity of this step is O(N^K).
- Note that, although unlikely, if a test pattern can test all nets at the same time, the run-time of sensitized path-search procedure is very high. However, our simulations on academic benchmarks and industrial circuits show that, for most cases, a maximum of 5-10% of the nets can be tested for transition faults by a single test pattern, and the sensitized paths, except clock logic cones, have a small amount of branching. Our analysis showed that the number of fanout pins is three or less for 95% of all instances. Although we found some cases where the number of fanouts is high, majority of them were on clock logic cones. In our sensitized path analysis, we excluded clock cones. As a result, the run time of the sensitized-path search procedure is considerably less than the ATPG run time.

Our simulations showed that for the given AMD circuit blocks, there are no sensitized paths longer than T_{func}. Thus, setting T_{CRT} to T_{func} leads to no patterns being selected. Thus, results for $T_{CRT} = T_{func}$ are not presented here.

To minimize the total run-time, we integrated the deviation computation procedure into our sensitized-path-search tool, which is named as pathfinder. As a result, pathfinder computes the output deviations and finds all sensitized paths at the same time. In the next step, all sensitized paths are assigned a weight that is equal to the output deviation of its end point. The weight of a test pattern is defined as the sum of the weights of all the paths sensitized by this pattern.

Pattern selection is driven by the weights of test patterns. Patterns with the largest weights are selected first. However, it is possible that some of the sensitized paths of two different patterns are the same. If a path has already been detected by the selected patterns, it is not necessary to use it for evaluating the remaining patterns. The objective of this method is to minimize the number of selected patterns while still sensitizing most delay-sensitive paths.

In the proposed pattern selection procedure, the largest weighted pattern is ordered to be the first. After selecting this pattern, we re-calculate the weight of all the remaining patterns by excluding paths detected by the selected pattern. Then, the pattern with the largest weight in the remaining pattern set is selected. This procedure is repeated until some stopping criteria is met. The number of selected patterns or the minimum-allowed pattern weight can be used as a stopping criterion. Since the selected patterns are already sorted on the basis of effectiveness, there is no need to re-sort the final set of patterns.

The final stage of the proposed method is to run top-off TDF ATPG to recover TDF coverage. Since the main purpose of this paper is to show the application of pattern selection on industrial circuits, results for this step are not presented.

C. Comparison to SSTA Techniques

The proposed method can be compared to SSTA-based techniques such as [7] and [8]. Both [7] and the proposed work present a transition-test pattern quality metric for the detection of SDDs in the presence of process variations. The main focus of [8], on the other hand, is to present a timing-hazard-aware SSTA-based technique for the same target defect group. Timing hazards are not covered by [7] or the proposed work. The formulation is different in these methods. In [7], dynamic timing analysis is run multiple times, for each test pattern, to create a delay distribution. Simple operators, e.g., +/-, are used while propagating the delay values. In [8], statistical dynamic timing analysis is run once for each test pattern. Similar to [7], simple operators are used for delay propagation, but the analysis of timing hazards adds complexity to the formulation. In the proposed work, similar simple operators are used, but reconvergent fanouts and timing-hazards are ignored for a simpler formulation. Our analysis using HSpice simulation showed that this simplification in formulation introduces less than 10% error in delay variance on long paths. The run time of SSTA-based approaches [7] and [8] are expected to be a limiting factor in the

TABLE III
DESCRIPTION OF THE INDUSTRIAL CIRCUIT BLOCKS

Benchmark	Functionality
Circuit A	Cache block
Circuit B	Execution unit
Circuit C	Execution unit
Circuit D	Load-store unit

applicability to industrial-size designs. Further optimization may eliminate this shortcoming. On the other hand, the proposed method is quick and its run time increases less rapidly with increase in circuit size.

III. EXPERIMENTAL RESULTS

In this section, we present experimental results obtained for four industrial circuit blocks. We first provide details for the designs and the experimental setup (Section III-A). Next, we present the simulation results (Section III-B).

A. Experimental Setup and Benchmarks

All experiments were performed on a pool of state-of-the-art servers running Linux with at least four processors available at all times and 16 GB of memory. We used pathfinder to compute output-deviations, find sensitized paths, and select patterns were implemented using C++. We used a commercial ATPG tool to generate n-detect TDF test patterns and timing-aware TDF patterns for these circuits. The ATPG tool was forced to generate launch-on-capture (LOC) transition fault patterns. The primary input change during capture cycles and the observation of primary outputs was prevented to simulate realistic test environments. All ATPG runs and other simulations were run in parallel on four processors.

Blocks were selected from functional units of state-of-the-art AMD microprocessor designs. Each block has a different functionality. Table III shows the functionality of each block.

While generating n-detect ($n = 5$ and $n = 8$) TDF patterns, no limits were placed on the number of patterns generated, nor was any target set for fault coverage. The tool was allowed to run in automatic optimization mode. In this mode, the ATPG tool sets compaction and ATPG efforts, determines ATPG abort limits, and controls similar user-controlled options. While generating timing-aware ATPG patterns, we used two different optimization modes. In ta-1 mode, the tool was forced to sensitize the longest paths to obtain the highest-quality test patterns, at the expense of increased CPU time and pattern count. In ta-2 mode, the optimization criteria were relaxed to decrease run time and pattern-count penalty.

Timing information for gate instances was obtained from a timing library, as described in Section II-B. Both the ATPG tool and pathfinder tool use the same timing data. The ATPG tool used nominal delay values since it was not designed to use delay variations. Interconnect delays are not modeled and left as a future work. The pattern selection in pathfinder was allowed to continue until all non-zero-weight patterns were selected.

B. Simulation Results

The simulations for generating patterns can be grouped into three main categories:

- **n-detect TDF ATPG:** Patterns were generated for a range of multiple-detect values. We used $n = 1, 3, 5$, and 8. The results for $n = 5$ and $n = 8$ are shown in this paper. We used n-detect TDF pattern set because we need a pattern repository that is likely to sensitize a large number of long paths, comparable to number of long paths sensitized by timing-aware ATPG. This requirement is satisfied if we use a large number of patterns and n-detect TDF ATPG.
- **Timing-aware ATPG using different optimization modes:** Timing-aware patterns were generated for optimization modes ta-1 and ta-2, as described previously.
- **Selected patterns:** We used our in-house pathfinder tool to select high-quality patterns from both n-detect and timing-aware pattern sets.

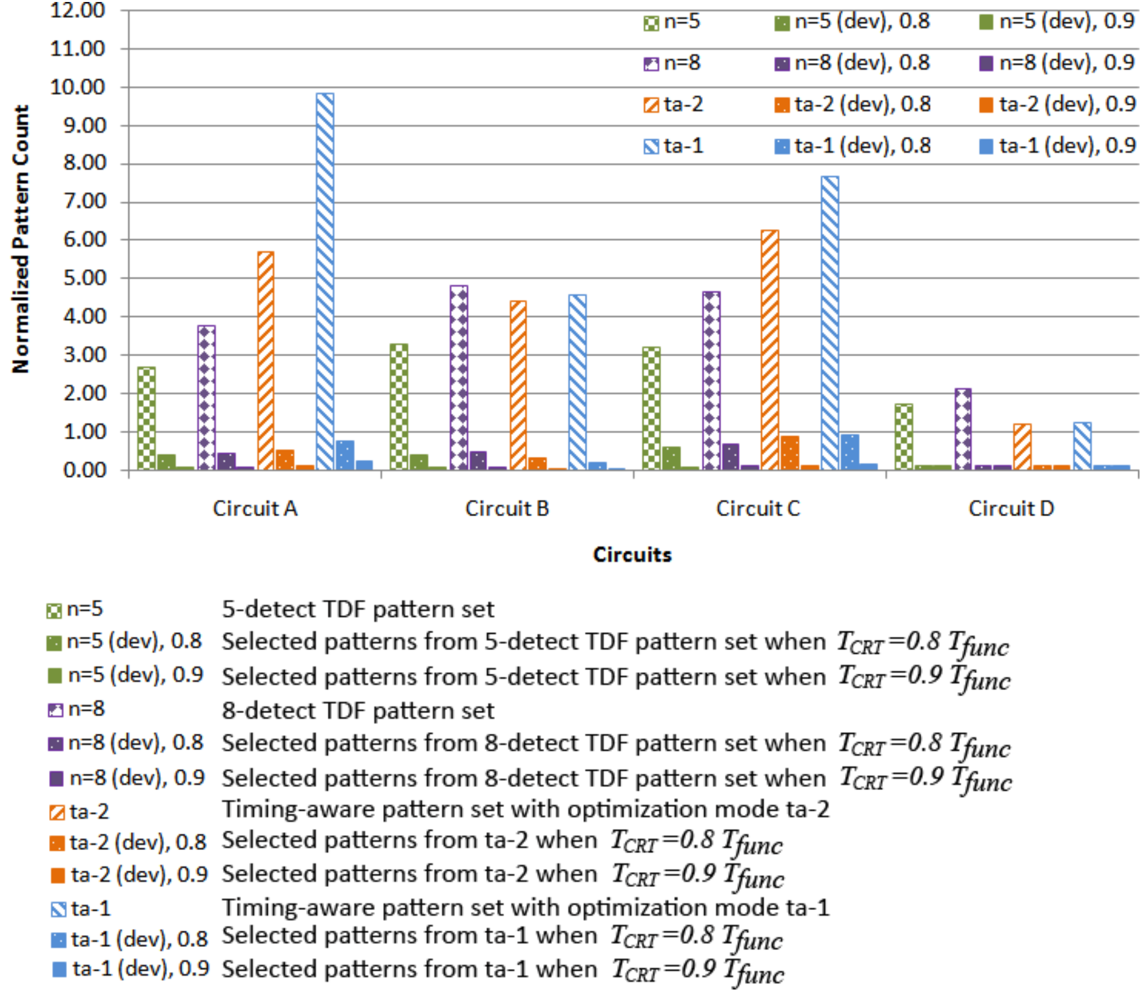


Fig. 6. Normalized number of test patterns for n-detect ATPG, timing-aware ATPG (ta-1 and ta-2), and the proposed pattern selection method (dev). T_{CRT} is set to $0.8 \cdot T_{func}$ and $0.9 \cdot T_{func}$ for the proposed method. All values are normalized by the result for $n = 1$.

We observed that the pattern count for timing-aware ATPG is much higher than the TDF ATPG pattern count. As n increases, the number of patterns in the n-detect pattern set also increases. Fig. 6 shows the number of test patterns generated by n-detect ATPG and timing-aware ATPG, and the number of patterns selected by the proposed method (while retaining the long-path coverage provided by the much larger test set). Fig. 6 shows results for $T_{CRT} = 0.8 \cdot T_{func}$ and $T_{CRT} = 0.9 \cdot T_{func}$. We find that, for all cases, the number of patterns selected by the proposed method is only a very small fraction of the overall pattern set. When T_{CRT} is set to $0.8 \cdot T_{func}$, the proposed scheme selects only 7% of the available patterns for Circuit A from the timing-aware pattern set ta-1. Similar results are obtained for other benchmarks. As expected, as T_{CRT} increases, the number of selected patterns drops as low as 3% of the original pattern set.

The results for CPU time usage is as striking as the pattern count results. Fig. 7 shows the normalized CPU time usage results for n-detect ATPG, timing-aware ATPG (ta-1 and ta-2), and the proposed pattern grading and pattern selection method (dev). As seen, the complete processing time (pattern grading and pattern selection) for the proposed scheme is only a fraction of the ATPG run time. For instance, for Circuit C, $n = 8$ ATPG run-time is 10 times longer than the pattern grading and selection time. For Circuit B, the time spent for pattern grading and selection is only 2.5% of the ta-1 timing-aware ATPG run time.

Since the proposed scheme allows us to select as many patterns as needed to cover all high-risk paths, the patterns selected by the proposed scheme sensitized all of the long paths that can be excited by the given base

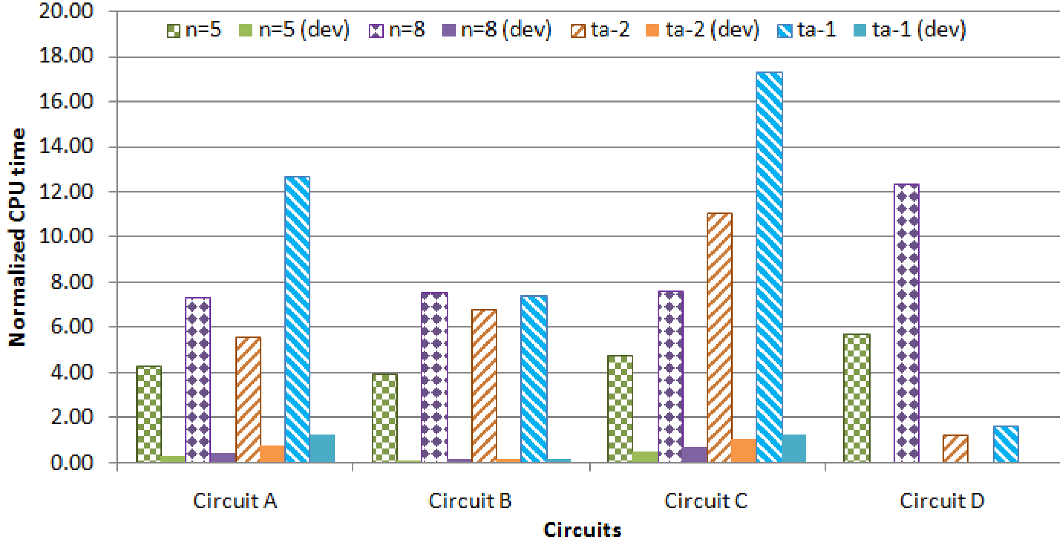


Fig. 7. Normalized CPU time usage for n-detect ATPG, timing-aware ATPG (ta-1 and ta-2), and the proposed pattern grading and pattern selection method (dev). T_{CRT} is set to $0.8 \cdot T_{func}$ for the proposed method. All values are normalized by the result for $n = 1$.

pattern set, using only a fraction of the test patterns. Note that the effectiveness of our method is bounded by the effectiveness of the base pattern set.

The long-path coverage ramp-up for the selected patterns is also significantly better than both n-detect and timing-aware ATPG patterns. Fig. 8 presents the results for the long-path coverage ramp-up with respect to the number of applied patterns. For all cases, the selected and sorted patterns cover the same number of long paths much faster, and use far fewer patterns.

C. Comparison of the Original Method to the Modified Method

In this section, we compare the original deviation-based method [11] to the new method proposed in this paper. We used three ASIC-like IWLS benchmark circuits for this comparison. Note that, although the RTL codes of these benchmarks are the same as the ones used in [11], the synthesized netlists are different due to library and optimization differences. To make a fair comparison, we re-implemented the original method [11] to use the same pattern selection method proposed in this paper. The only difference between these methods is the procedure to calculate metrics. All simulations are run on servers with similar configurations.

Fig. 9 shows a comparison of CPU time usage between the original and the modified method. As seen, for all cases, the new method has a superior CPU time usage compared to the original method. The main reason for the difference between CPU time usages is the fact that the original method consumed more time to evaluate patterns due to the larger number of selected patterns, as shown in Fig. 11. Depending on the benchmark, the impact of this effect on the overall CPU time usage can be considerable, as in the case of aes_core, or it can be negligible, as in the case of systemcaes.

Fig. 9 presents the normalized number of sensitized long paths for the original and the modified method. As seen, although the modified method consistently sensitizes more long paths compared to the original method, the difference is rather small. This can be better analyzed if we consider Fig. 11. Fig. 11 shows the normalized number of selected patterns for each method. As seen, the modified method consistently selects fewer patterns than the original method. The difference is significant for benchmarks systemcaes and tv80. Although the number of sensitized long paths is similar for these methods, the number of selected patterns is significantly different. This result shows that the modified method is more efficient than the original method in selecting high-quality patterns. The main reason for the difference in the number of selected patterns is the fact that, due to deviation saturation

on long paths, the original method is unable to distinguish between long paths and shorter paths. As a result, the original method selects more patterns to cover all of these paths.

IV. Conclusions

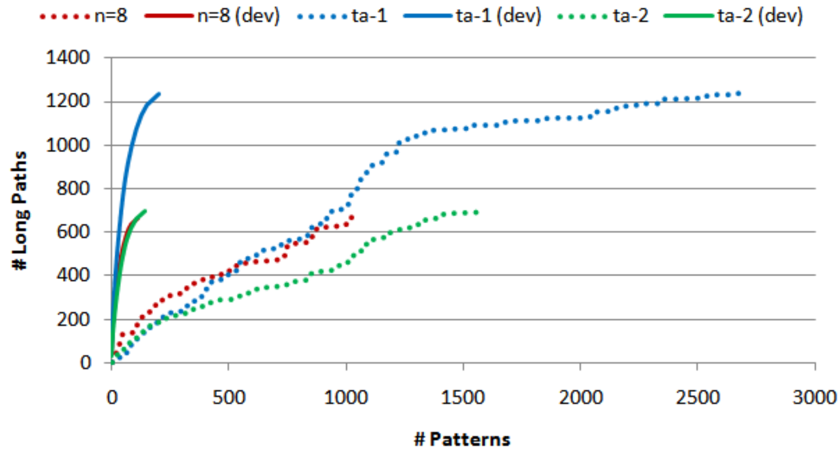
We have presented a test-grading technique based on output deviations for SDDs and applied it to industrial circuits. We have redefined the concept of output deviations so it can be applied to industrial circuit blocks, and have shown it can be used as an efficient surrogate metric to model the effectiveness of transition delay-fault (TDF) patterns for SDDs. Experimental results for the industrial circuits show that the proposed method intelligently selects the best set of patterns for SDD detection from an n-detect or timing-aware TDF pattern set, and it excites the same number of long paths compared to commercial timing-aware ATPG tool using only a fraction of the test patterns and with negligible CPU time overhead.

Acknowledgements

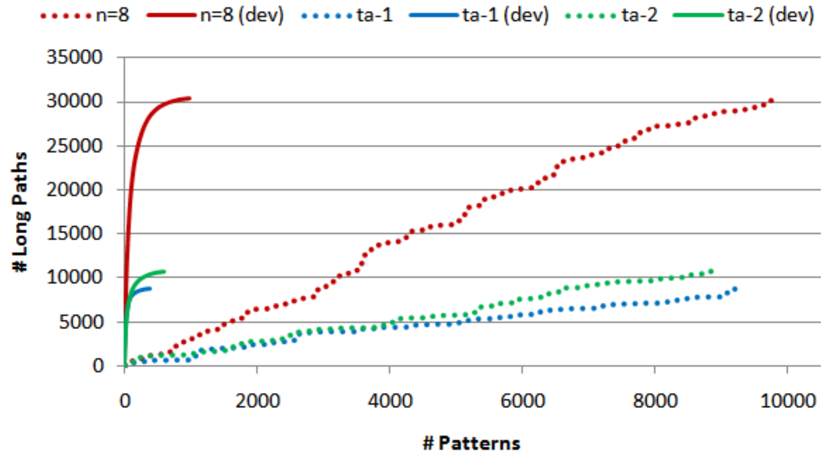
We thank Jeff Rearick, Jeff Fitzgerald, and other colleagues at AMD for valuable discussions and for providing us access to computing resources.

References

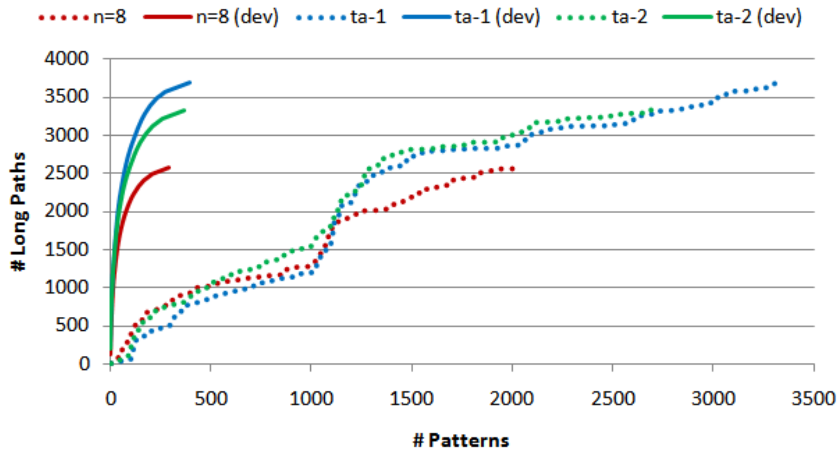
[1] N. Ahmed, M. Tehranipoor, and V. Jayaram, “Timing-based delay test for screening small delay defects,” in *Proc. IEEE Design Automation Conf.*, 2006, pp. 320–325.
[2] J. Waicukauski et al., “Transition fault simulation,” *IEEE Design and Test of Computers*, pp. 32–38, 1987.
[3] R. Putman and R. Gawde, “Enhanced timing-based transition delay testing for small delay defects,” in *Proc. IEEE VLSI Test Symp.*, 2006, pp. 336–342.
[4] E. Park, M. Mercer, and T. Williams, “A statistical model for delay-fault testing,” *IEEE Design and Test of Computers*, vol. 6, no. 1, pp. 45–55, 1989.
[5] C. Forzan and D. Pandini, “Why we need statistical static timing analysis,” in *Proc. IEEE ICCD*, 2007, pp. 91–96.
[6] I. Nitta, S. Toshiyuki, and H. Katsumi, “Statistical static timing analysis technology,” vol. 43, no. 4, pp. 516–523, Oct 2007.
[7] C.-T. M. Chao, L.-C. Wang, and K.-T. Cheng, “Pattern selection for testing of deep sub-micron timing defects,” in *Proc. IEEE DATE*, vol. 2, 2004, pp. 1060–1065.
[8] B. Lee, H. Li, L.-C. Wang, and M. Abadir, “Pattern selection for testing of deep sub-micron timing defects,” in *Proc. IEEE Int. Test Conference*, 2005.
[9] B. Keller et al., “An economic analysis and ROI model for nanometer test,” in *Proc. IEEE Int. Test Conference*, 2004, pp. 518–524.
[10] F.-F. Ferhani and E. McCluskey, “Classifying bad chips and ordering test sets,” in *Proc. IEEE Int. Test Conference*, 2006.
[11] M. Yilmaz, K. Chakrabarty, and M. Tehranipoor, “Test-pattern grading and pattern selection for small-delay defects,” in *Proc. IEEE VLSI Test Symp.*, 2008, pp. 233 – 239.
[12] Z. Wang and K. Chakrabarty, “Test-quality/cost optimization using output-deviation-based reordering of test patterns,” *IEEE Trans. CAD*, vol. 27, pp. 352–365, Feb 2008.
[13] M. Yilmaz, K. Chakrabarty, and M. Tehranipoor, “Test-pattern selection for screening small-delay defects in very-deep submicron integrated circuits,” Duke University, Tech. Rep., ECE-2009-02. [Online]. Available: http://hdl.handle.net/10161/1376
[14] ——, “Interconnect-aware and layout-oriented test-pattern selection for small-delay defects,” in *Proc. IEEE Int. Test Conference*, 2008.
[15] K. Trivedi, *Probability and Statistics with Reliability, Queuing, and Computer Science Applications (Second Edition)*. New York, NY: John Wiley and Sons, 2001.



(a)



(b)



(c)

Fig. 8. The long-path coverage ramp-up using the base 8-detect TDF ATPG ($n = 8$), timing-aware ATPG in modes ta-1 and ta-2, selected patterns from 8-detect TDF ATPG ($n = 8$ (dev)) and timing-aware ATPG (ta-1 (dev), ta-2 (dev)). T_{CRT} is set to $0.8 \cdot T_{func}$ for the proposed method: (a) Circuit A; (b) Circuit B; (c) Circuit C.

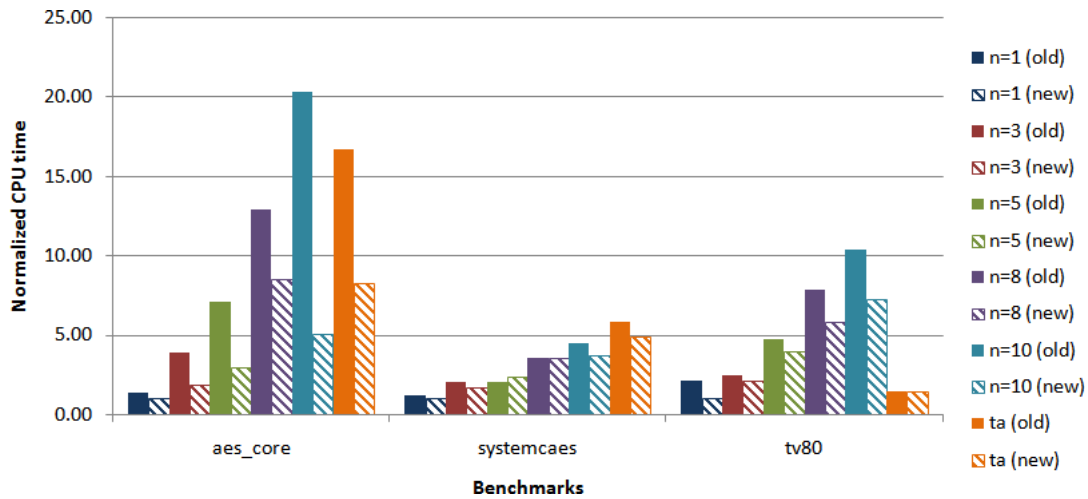


Fig. 9. Normalized CPU time usage for the selected n-detect ATPG and timing-aware ATPG patterns, for the original method (old) and the modified method (new). All values are normalized by the result for $n = 1$ (new).

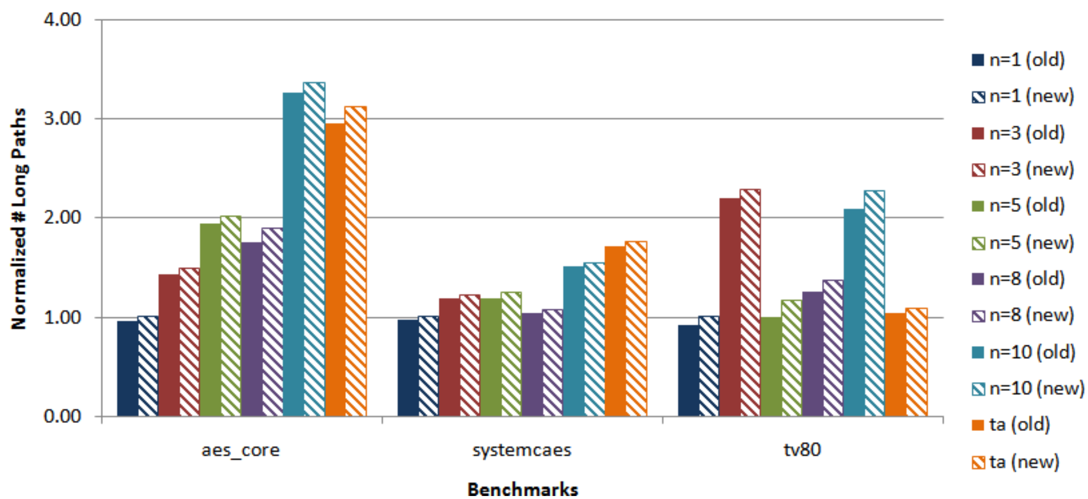


Fig. 10. Normalized number of sensitized long paths for the selected n-detect ATPG and timing-aware ATPG patterns, for the original method (old) and the modified method (new). All values are normalized by the result for $n = 1$ (new).

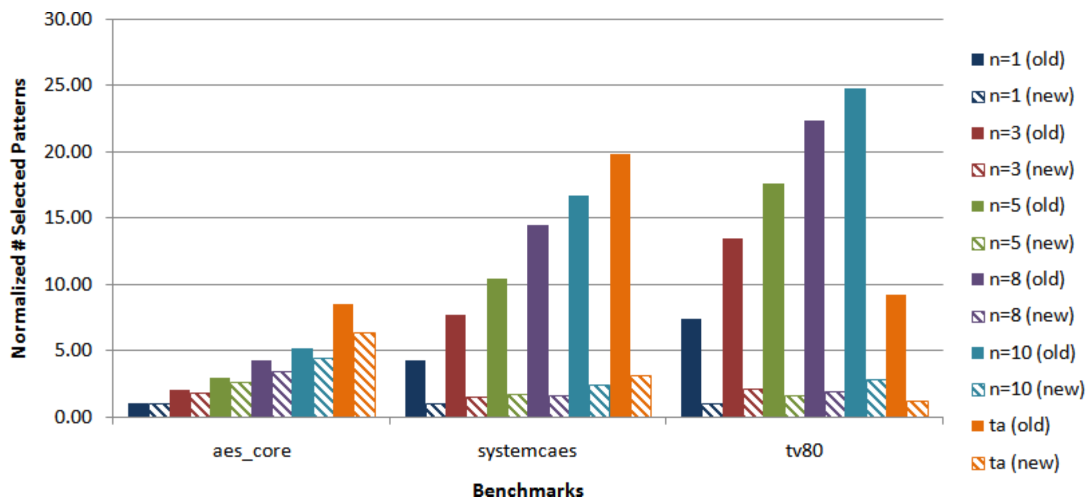


Fig. 11. Normalized number of selected patterns for the selected n-detect ATPG and timing-aware ATPG patterns, for the original method (old) and the modified method (new). All values are normalized by the result for $n = 1$ (new).