Test-Pattern Selection for Screening Small-Delay Defects in Very-Deep Submicron Integrated Circuits[†]

Mahmut Yilmaz, Krishnendu Chakrabarty, and Mohammad Tehranipoor

***Abstract*—Timing-related defects are major contributors to test escapes and in-field reliability problems for very-deep submicron integrated circuits. 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 that uses 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 a commercial timing-aware ATPG tool. Our results also show that, for the same pattern count, the selected patterns provide more effective coverage ramp-up than timing-aware ATPG and a recent pattern-selection method for random small-delay defects potentially caused by resistive shorts, resistive opens, and process variations.**

***Index Terms*—Delay test, output deviations, process variations, small-delay defects, test-pattern grading**

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 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], [2].

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 larger than the clock period, causing circuit failure or temporarily incorrect results. As a result, the detection of SDDs typically 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) [3] model attempts to propagate the lumped delay defect of a gate by logical transitions to the observation points or state elements. The effectiveness of the TDF model for SDDs has often been questioned [1], [4] due to its tendency to excite transition delay-faults through short paths [1].

Due to the growing interest in SDDs, the first commercial timing-aware automatic test-pattern generation (ATPG) tools were introduced recently, e.g., new versions of Mentor Graphics FastScan, Cadence Encounter Test, and Synopsys TetraMax [5]–[7]. 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 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 transition-delay-fault pattern generator and (***ii***) timing-aware ATPG that takes timing information into account. The results are shown for some of the IWLS'2005 benchmarks [8] and the absolute run times are shown in Section IV-F. It can be seen from Fig. 1 that, when the benchmark is large, the timing-aware ATPG takes significantly higher CPU time, e.g., as much as 209x more for the "netcard" benchmark. The CPU times in Figure 1 reflect distributed ATPG results using eight processors, and the numbers are normalized such that the run-time of timing-unaware ATPG is taken to be one unit. The absolute CPU run-time values can be found in Section IV-F.

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 transition-delay faults is 2-5 times higher than that for stuck-at faults [9], and it has been demonstrated recently that test patterns for such sequence-

---

[†]A preliminary version of this paper was published *in Proc. IEEE VLSI Test Symposium*, pp. 233-239, 2008

M. Yilmaz is with Advanced Micro Devices (AMD), Sunnyvale, CA, 94085 USA (e-mail: mahmut.yilmaz@amd.com) and with the Department of Electrical and Computer Engineering, Duke University, Durham, NC, 27708 USA (e-mail: my@ee.duke.edu).

K. Chakrabarty is with the Department of Electrical and Computer Engineering, Duke University, Durham, NC, 27708 USA (e-mail: krish@ee.duke.edu).

M. Tehranipoor is with the Department of Electrical and Computer Engineering, University of Connecticut, Storrs, CT 06269 USA (e-mail: tehrani@engr.uconn.edu).

The work of M. Yilmaz and K. Chakrabarty was supported in part by SRC under Contract no. 1588 and by NSF under Grant no. ECCS-0823835.

The work of M. Tehranipoor was supported in part by SRC under Contracts 1455 and 1587, and by NSF under Grant no. ECCS-0823992.

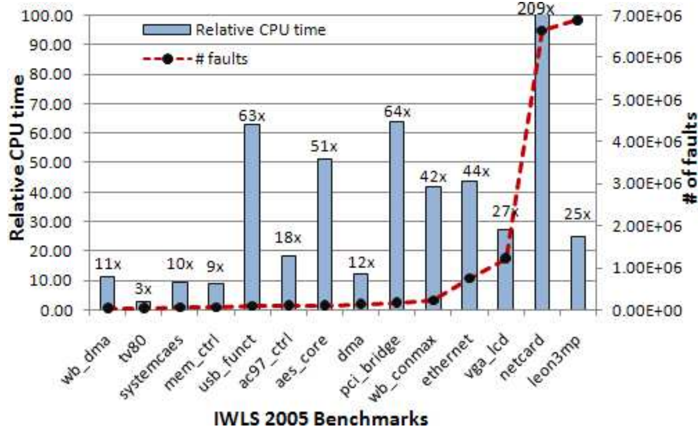


Fig. 1. Comparison of ATPG run times (CPU time): timing-aware ATPG relative to timing-unaware ATPG for IWLS 2005 benchmarks.

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 larger in 2015 than it was in 2007 [11]. Therefore, efficient pattern-selection methods are required to reduce the total pattern count while effectively targeting SDDs.

This paper presents the output deviation measure [12], [13] 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 target defects. The proposed method can be used with traditional, timing-unaware ATPG tools to generate a high-quality and compact delay-fault pattern set. It can also be used to select the most-effective patterns from large timing-aware test sets. Experimental results show that the proposed method can effectively select the highest-quality patterns from large test sets that cannot be used in entirety for production test environments with tight pattern-count limits. It also considers process-variability-induced delay variations, unlike most previous methods. The proposed approach requires significantly less CPU time than a commercial timing-aware ATPG tool under pattern-count limits. For various metrics, namely coverage of long paths, detection of injected defects, and coverage ramp-up, it is shown to outperform a commercial timing-aware ATPG tool. We also compare the proposed method with the approach proposed by Lee *et al.* [14], where path-length calculations are approximated for better run-time, and we highlight better long-path coverage, lower run times, and faster coverage ramp-up for injected faults.

The remainder of this paper is organized as follows. Section II presents related prior work in the area of SDDs. In Section III, we describe the probabilistic fault model and the output deviations metric. Section III-D presents the proposed pattern-selection procedures. In Section IV, 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 V concludes the paper.

II. Related Prior Work

SDDs were first alluded to in [15]. In recent years, high-quality delay-fault pattern generation for SDDs has received increasing attention. Most of the work is aimed at finding the longest paths in a circuit. Gupta et al. [16] have proposed the “As Late As Possible Transition (ALAPTF)” fault model, which attempts to launch one or more transitions at the fault site as late as possible, i.e., through the least-slack path using robust tests. This method suffers from the need for a complex, time-consuming search procedure part and robust test-generation constraints. Qui et al. [17] attempt to find the k-longest paths (referred to as KLPG) through the inputs and output of each gate for slow-to-rise and slow-to-fall faults. Similar to [16], a considerable amount of pre-processing is needed to search for long paths. Furthermore, a long path through a gate may be a short path in the circuit, thus not all the paths determined by the method are least-slack paths. Ahmed et al. [1] use static timing analysis tools to find long (LP), intermediate (IP), and short paths (SP) to each observation point. Using a timing-unaware ATPG tool, 15-detect transition test patterns are generated. During pattern generation, constraints are applied on IP and SP observation points to mask them. In this way, the ATPG tool is forced to generate patterns for LPs. In the post-processing phase, a pattern-selection algorithm is used to pick patterns that activate the largest number of end-points. Similar to previous methods, a time-consuming search procedure is needed for determining long paths and for path classification. A functional delay fault test generation method is proposed in [18]. This method generates sequences of instructions for testing delay faults. However, it requires a fault-free unit that can run the instructions for the test program. Similar to earlier methods, this scheme also involves a lengthy preprocessing step. In [19], delay defects within slack intervals are detected by using a clock frequency higher than the rated clock frequency. This method uses a good neighboring die to test the surrounding dies. The responses of the good die and the other dies are compared with each other to find delay defects. A new fault model, called the “Transition Path Delay Fault Model” is described in [20]. This fault model relaxes the robustness constraint required by the path-delay fault model, and aims to excite paths that are missed by the path-delay fault model. [21] presents a method to accurately determine the fault coverage of path-delay tests by analyzing path reconvergences. This method is applicable to the bounded gate-delay model.

As a result of increasing industry concern regarding SDDs, companies such as Mentor Graphics, Cadence, and Synopsys have recently released timing-aware ATPG tools [2], [5]–[7]. Lin et al. [2] use SDF files to guide ATPG to generate transition test through long paths. In a pre-processing step, the proposed method evaluates the delay for “activation” and “propagation” paths for each gate to find longest paths. Test generation is guided by the results of this pre-processing step. Although

approximation methods are used to decrease the overhead associated with delay and path-length calculations, this method still takes considerably more time compared to timing-unaware ATPG tool. Kapur et al. [7] uses STA tool generated pin slack information to guide timing-aware ATPG. Although the preprocessing step is skipped by pushing the slack data calculation to the STA tool, pattern generation takes considerably more time than timing-unaware ATPG.

Statistical static timing analysis (SSTA) can generate variability-aware delay data. However, as clearly demonstrated in recent papers [22], [23], SSTA does not find sensitized paths based on input vectors using statistical data. Furthermore, a complete SSTA flow takes considerable computation time [24], [25].

The "number of activated long paths" is a useful metric for evaluating the quality of delay-fault pattern quality, but a more computationally-tractable method is clearly needed. An alternative evaluation method, referred to as the "statistical delay quality model" (SDQM) has been proposed by Sato et al. [26]. This pattern-grading metric is based on a delay-defect distribution function, which requires delay-defect statistics for fabricated ICs. The method assigns a "statistical delay quality level (SDQL)" to each test set to evaluate its quality. A drawback of this metric is the need for delay-defect distributions for real chips. This data is not available before volume production and it is difficult and very expensive to obtain it during production test. Another shortcoming of SDQM and similar metrics is that they require knowledge of the longest sensitizable paths, which is not accurately known before production test.

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

In this section, we first introduce the concept of gate-delay defect probabilities (DDPs) (Section III-A) and signal-transition probabilities (Section III-B). 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 III-B). We describe the algorithm used for signal-probability propagation (Section III-C). Finally, we describe how test patterns can be ranked and selected from a large repository (Section III-D).

A. 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 2-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`. `0.5` shows 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
Example DDPM for a 2-input OR gate

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

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:

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

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 selection of X is described in Section IV-A.

The delay distribution can be obtained in different ways: (*i*) using the delay information provided by an SSTA generated 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 (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 patterns 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, hence 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 Section IV-F.

B. 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 $L \rightarrow L$, $L \rightarrow H$, $H \rightarrow L$, and $H \rightarrow H$. Each of these transitions has a corresponding probability, denoted by $P_{L\rightarrow L}$, $P_{L\rightarrow H}$, $P_{H\rightarrow L}$, and $P_{H\rightarrow H}$, respectively, in a vector form (< ... >): $< P_{L\rightarrow L}, P_{L\rightarrow H}, P_{H\rightarrow L}, P_{H\rightarrow H} >$. We refer to this vector as the signal-transition probability (STP) vector. Note that $L \rightarrow L$ or $H \rightarrow H$ 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. 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 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 has been reported in [27].

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, a property that follows from the rules used to calculate signal-transition probability for a gate output. This claim is formally proven next.

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

Proof: If there is no signal-value change (the event $L \rightarrow L$ or $H \rightarrow H$), the expected signal-transition probability 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 $L \rightarrow H$ is $L \rightarrow L$ (the signal value does not change because of a delay-fault). Similarly, the delay-fault case for an expected signal value change of $H \rightarrow L$ is $H \rightarrow H$. ■

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:

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

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:

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

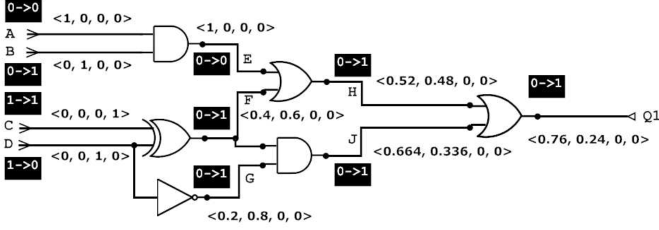


Fig. 2. An example to illustrate the propagation of signal-transition probabilities through the gates of a logic circuit.

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**		
Inpus	**IN0**	0.2	0.2		

Case (*ii*): Suppose L input ports ($L > 1$), indexed 1,2,…,L, are required to make a transition in order for the gate output to change. Let $d^*_{max} = max_{1 \le j \le L}\{d_j\}$. 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 $\Delta^*_{OUT} \le P_{OUT,i}, 1 \le i \le L$, since $0 \le d^*_{max} \le 1$. 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 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 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 as follows, based on the rules mentioned above for the example circuit in Fig. 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 delay defect probability 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 delay-defect probability. Since both inputs change, we use the maximum delay defect probability, 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 delay-defect probability, 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 $0.664 \odot \langle 1, 0, 0, 0 \rangle$;
    - ○ If `IN1` goes to `HIGH`, the output changes with a delay-defect probability, 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.

C. 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.

```
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.

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 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 $O(\frac{N_s \cdot N_n}{T})$.

D. 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 in order 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

```
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.

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 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 8 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.

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 $t_i > D_{LIMIT} \cdot$ 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.

E. An Alternative Approach

The two most significant inputs required by the proposed output deviations method are the gate and interconnect delay variations, and T_{CRT} for gates. Defining a T_{CRT} for individual gates may not be feasible for all projects. The proposed work is more directly applicable to microprocessor designs with shallow logic depth. For designs with a large number of gates, e.g., larger than 20, on a path, the output deviation metric saturates and equal output deviations (close to 1) are obtained for both long and intermediate paths. Both of these drawbacks can be overcome with slight modifications to the proposed approach as shown in [28]. Deviation-driven pattern selection can also propagate Gaussian distributions instead of defect probabilities. This adds no extra cost since propagation of Gaussians is simply the addition of means and variance (from the Central Limit Theorem of statistics) [28]. In this approach, the dependence of gate level T_{CRT} is eliminated. T_{CRT} can be defined for the circuit as a fraction of the functional clock period of the circuit. 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}.

IV. Experimental Results

In this section, we present experimental results obtained for the IWLS benchmark circuits. We first describe how we obtained DDPs for the logic gates in these benchmarks (Section IV-A). Next, we provide details for the benchmark circuits. Then, we provide details for the experimental setup (Section IV-C). Before presenting the simulation results (Sections IV-E and IV-F), we provide a summary of the dynamic-timing simulation method proposed in [14], which will be compared with our proposed deviation-based method (Section IV-D).

A. Finding Gate-Delay Defect Probabilities

In order to determine the gate-DDPs, we ran 200 HSpice Monte Carlo (MC) simulations on each gate, for all possible input signal transitions, using 45 nm process-technology BSIM4 predictive transistor models. We verified that 200 MC simulations are sufficient for generating DDPMs. Fig. 6 shows the relative change in DDPM entries for various gates as the number of MC simulations increase. The numbers are normalized by the values obtained from 50 MC simulations. As seen, there is very little change in the DDPM entries well before we reach 200 MC simulations. Therefore, we conclude that running more MC simulations will not lead to any significant difference in the DDPM entries. Similar results were obtained for other library cells.

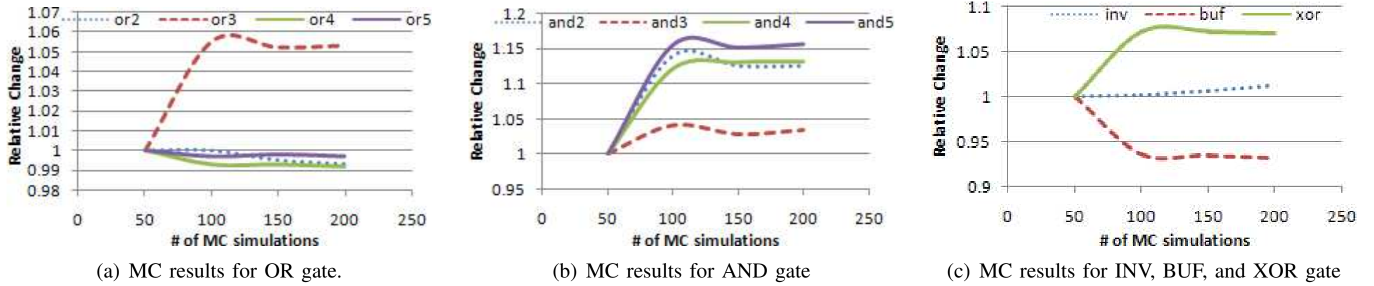
For each gate type, we simulated the schematic under various loading capacitance and input slew rate conditions to account for spatial correlations. Next, we used interpolation to find the mean and standard deviation of gate delays for individual gate instances. We have seen that interpolation is possible and it is accurate. For instance, the mean delay value approximately linearly changes with the load capacitance, and further accuracy can be achieved by using third-order polynomial curve fitting.

MC simulations were carried out using the following realistic process-variation parameters (obtained from a current VDSM technology from industry) 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, MC simulations were performed for each possible input transition. For each gate, the probability that the transition-delay value is greater than $T_{CRT} = NOM + Xs$ constitutes the DDP value for the respective input transition (where NOM refers to the nominal delay value). The parameter X is selected in such a way that all the gate instances have at least one non-zero DDPM entry. Note that X is the minimum of all MAX delays (among all gates) for each benchmark. 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 $NOM + X$. As an alternative, X can be set to the MAX delay specified by an STA tool that does not consider process variation effects.

B. Benchmarks

In this section, we present the details of the IWLS 2005 benchmark circuits. We do not consider the ISCAS benchmarks because these circuits are small and it is easier for an ATPG tool to excite all long paths with a small number of patterns.

IWLS benchmark circuit statistics are shown in Table III. As seen, IWLS benchmarks represent a wide range of application areas, including memory controllers and microprocessors. Note that IWLS benchmarks are provided in the Verilog RTL format, and the statistics given in Table III may slightly change depending on the synthesis tool and synthesis optimization options. For our experiments, we selected a subset of the IWLS benchmarks: systemcaes, usb_funct, ac97_ctrl, aes_core, pci_bridge32, wb_conmax, and ethernet. The largest benchmarks require a prohibitive amount of computing resources for the collection of simulation data for pattern quality evaluation (sensitized paths and coverage ramp-up), therefore we do not report results for them.



(a) MC results for OR gate. (b) MC results for AND gate (c) MC results for INV, BUF, and XOR gate

Fig. 6. Change in DDPM values as a function of the number of MC simulations, relative to the base case of 50 MC simulations.

TABLE III
BENCHMARK STATISTICS

Benchmark	# I/O	# gates	# flip-flops	function
wb_dma	942	7,619	563	WISHBONE DMA/Bridge IP Core
tv80	103	13,326	359	TV80 8-Bit Microprocessor Core
systemcaes	387	17,817	670	SystemC AES
mem_ctrl	208	22,015	1,083	WISHBONE Memory Controller
usb_funct	167	25,531	1,746	USB function core
ac97_ctrl	115	28,083	2,199	WISHBONE AC 97 Controller
aes_core	267	29,186	530	AES Cipher
dma	120	41,026	2,189	Direct Memory Access (DMA) Controller
pci_bridge32	361	43,907	3,359	PCI Interface
wb_conmax	388	59,484	770	WISHBONE Conmax IP Core
ethernet	45	153,948	10,544	Ethernet IP core
vga_lcd	222	252,302	17,079	WISHBONE rev.B2 compliant Enhanced VGA/LCD Controller
netcard	186	1,356,533	97,831	Network Card Controller
leon3mp	2,546	1,452,517	108,839	32-bit processor compliant with SPARC V8 architecture

C. Experimental Setup

All experiments were performed on a pool of state-of-the-art servers with at least eight processors available at all times, 16GB of memory, and running Linux. The program to compute output deviations was implemented using C++. A commercial tool was used to perform Verilog netlist synthesis and scan insertion for the IWLS benchmark circuits. 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 in order to simulate realistic test environments. The path delays were calculated using an in-house dynamic path-timing simulator. All simulations were run in parallel on eight processors.

D. Dynamic Timing Simulation

In this paper, we compare our proposed method with the one presented in [14]. Two methods for estimating the path delays for a given test vector are proposed in [14]. In particular, a *path-based* and a *cone-based* delay estimation scheme were provided. The estimated delays are called "metrics" associated with the test vectors. Two different delay models are used during the estimation process. ***(i)*** *Unit delay model*: Each gate has a unit delay, no spatial correlations are considered. ***(ii)*** *Differential delay model*: The gate type and the number of fanouts are considered.

For the path-based scheme, it is assumed that a set of critical paths is given. Non-robustly excited critical paths and their corresponding delays are found using either the unit or the differential delay model. The largest delay caused by the test pattern is assigned as the "metric" for the pattern. If there is no non-robust transition on a gate, zero delay is assumed on that gate. Because of the non-robust restriction, delay accuracy is lost. For the cone-based method, there are no non-robust restrictions, but delays are calculated for small cones of logic, and then the delays of cones are added to approximate the overall delay. Once all test vectors are associated with a "metric", the patterns are ordered by the "metric" and the top 1/3 of the patterns are selected.

In order to compare our work to the dynamic-timing simulation based method, we implemented the cone-based scheme using the differential delay model. Instead of the number of fanouts, we used capacitive loading of fanout gates to update the delay associated with each gate instance. The capacitive loading of gate ports are found by running HSpice simulations on transistor-level gate models. Note that in the absence of layouts, this data does not include layout-extracted data such as resistances and capacitances. Once the "metrics" are computed, we selected the top 1/3 of the patterns as proposed by Lee *et al.* [14].

TABLE IV
KENDALL'S COEFFICIENTS FOR EVALUATING THE CORRELATION OF PATH LENGTHS TO OUTPUT DEVIATIONS

Benchmark	(Ave,Min,Max)	Benchmark	(Ave,Min,Max)
systemcaes	(**0.96**, 0.82, 1)	pci_bridge32	(**0.93**, 0.85, 0.99)
usb_funct	(**0.97**, 0.85, 0.99)	wb_conmax	(**0.93**, 0.89, 0.98)
ac97_ctrl	(**0.98**, 0.93, 1)	ethernet	(**0.89**, 0.77, 0.95)
aes_core	(**0.97**, 0.9, 0.99)		

E. 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 calculated 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 dynamic timing simulator and determined the signal delays at the observation points of the benchmarks. Next, we used Matlab to compute the Kendall's correlation coefficients [29] for each pattern set. The Kendall's correlation coefficient is a measure between 0 and 1, where 0 indicates no correlation and 1 indicates perfect correlation. Table IV shows the average correlation coefficients for the patterns in a 1-detect test set of the IWLS'2005 benchmarks [8]. 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 IV, 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.

It can be argued that instead of output deviations, a dynamic timing simulator can be used to obtain high correlation to path lengths. However, the method based on output deviations is flexible and general, and it can be used during pattern selection to account for process variations and many physical defects. Dynamic timing simulation can only provide variability-unaware timing information. The method of output deviations is also expected to reveal unique problematic paths that may be hidden from dynamic timing analysis.

F. Pattern-Selection Results

In this section, we present the pattern-selection results for the proposed deviation-based method (`dev`) and the dynamic-timing simulation based method (`dts`) [14] for a production test environment with pattern-count limits.

We first generated 5-detect (n5), 8-detect (n8), and timing-aware (ta) transition-test patterns for each benchmark using a commercial ATPG tool. Next, `dev` and `dts` are used to select patterns from these base pattern sets. The motivation for also using timing-aware patterns as a base pattern-set lies in our observation that the pattern counts resulting from timing-aware ATPG for large industrial circuits are often prohibitively high. Test-set truncation is therefore necessary in practice. In order to obtain the same TDF coverage as 1-detect patterns, we ran top-off timing-unaware ATPG over the selected patterns.

The number of patterns generated by the commercial ATPG tool is given in Table V. When `dts` is used as the pattern

TABLE V
NUMBER OF TEST PATTERNS GENERATED BY THE COMMERCIAL ATPG TOOL FOR VARIOUS n-DETECT TDF AND TIMING-AWARE ATPG (TA)

Benchmarks	n5	n8	ta
systemcaes	1939	2821	2997
usb_funct	3802	5797	3683
ac97_ctrl	1907	2866	1643
aes_core	2734	3995	8277
pci_bridge32	4789	7383	4571
wb_conmax	21230	32189	6418
ethernet	25221	36855	13544

selection scheme, the top 1/3 patterns (ranked on the basis of the "metric") of the base patterns (n5, n8, and ta) are selected as in [14]; see Table VI. The number of patterns after top-off ATPG for TDFs is given in parentheses.

For `dev`, we used a range of D_{LIMIT} values for pattern selection. The results for $D_{LIMIT} = 0.6$ and $D_{LIMIT} = 0.8$ are shown in Table VII. We see that an increase in D_{LIMIT} resulted in a lower pattern count in the selected pattern set, but increased the number of top-off ATPG patterns. We observe that for most benchmarks, the selected pattern count is significantly smaller than the number of base timing-aware ATPG patterns, and even smaller than the number of patterns selected by `dts`. For the rest of the experimental results presented in this section for `dev`, we used the patterns selected with $D_{LIMIT} = 0.8$.

The CPU time required by timing-aware ATPG is an important concern, especially for large industrial designs under time-to-market constraints. We have seen that for large industrial circuits, timing-unaware TDF ATPG itself can take a couple of days to complete. Therefore, the CPU time for timing-aware ATPG can be prohibitive, as shown in Fig. 1 for benchmark circuits. We evaluated the total CPU time used by `dev` and `dts` and compared it to the CPU time used by the base timing-aware ATPG. The results are shown in Fig. 7 and Tables VIII-XIII. We find that, even with CPU time for top-off ATPG, for n-detect pattern sets, `dev` consistently has lower CPU time compared to both base timing-aware ATPG and `dts`. For the `ta` pattern group, the CPU time used for pattern selection and fault-grading is significantly smaller than the base timing-aware ATPG run time.

Tables VIII-X show the absolute CPU run-time in seconds, for the deviation-based pattern selection method. In each of these tables, the results are compared to base timing-aware ATPG run time. Similarly, Tables XI-XIII show the results for dynamic-timing-simulation-based pattern selection method. The CPU run-time of ATPG tool for larger benchmarks are presented in Table XIV.

In order to evaluate the effectiveness of selected patterns in terms of long-path sensitization, we ran timing simulations for the selected patterns and found the number of sensitized distinct long paths. Two paths are assumed to be distinct if there is at least one net that is not shared by them. A path is assumed to be a long path if the corresponding delay is higher than the specified long path limit (`LPL`). We ran the analysis for a range of `LPL` values, starting from 50% of the clock period to 90% of

TABLE VII
NUMBER OF TEST PATTERNS SELECTED BY THE DEVIATION-BASED SCHEME (DEV) WHEN $D_{LIMIT} = 0.6$ AND $D_{LIMIT} = 0.8$.

	$D_{LIMIT} = 0.6$*			$D_{LIMIT} = 0.8$*		
Benchmarks	**n5**	**n8**	**ta**	**n5**	**n8**	**ta**
systemcaes	974 (1054)	1135 (1210)	1245 (1417)	938 (1027)	1097 (1174)	1205 (1374)
usb_funct	1171 (1360)	1330 (1494)	1298 (1751)	659 (1110)	742 (1149)	744 (1390)
ac97_ctrl	614 (740)	699 (805)	575 (801)	281 (589)	246 (582)	63 (544)
aes_core	1104 (1214)	1232 (1309)	1379 (1479)	1060 (1182)	1179 (1277)	1294 (1413)
pci_bridge32	1069 (1525)	1287 (1683)	1432 (2005)	408 (1319)	459 (1341)	504 (1452)
wb_conmax	2434 (5588)	2572 (5608)	1949 (5648)	2091 (5575)	2275 (5535)	1525 (5651)
ethernet	11967 (13740)	13448 (14606)	8877 (11472)	11479 (13453)	12911 (14304)	8493 (11351)

*Numbers in parenthesis refer to the pattern count after top-off timing-unaware ATPG.

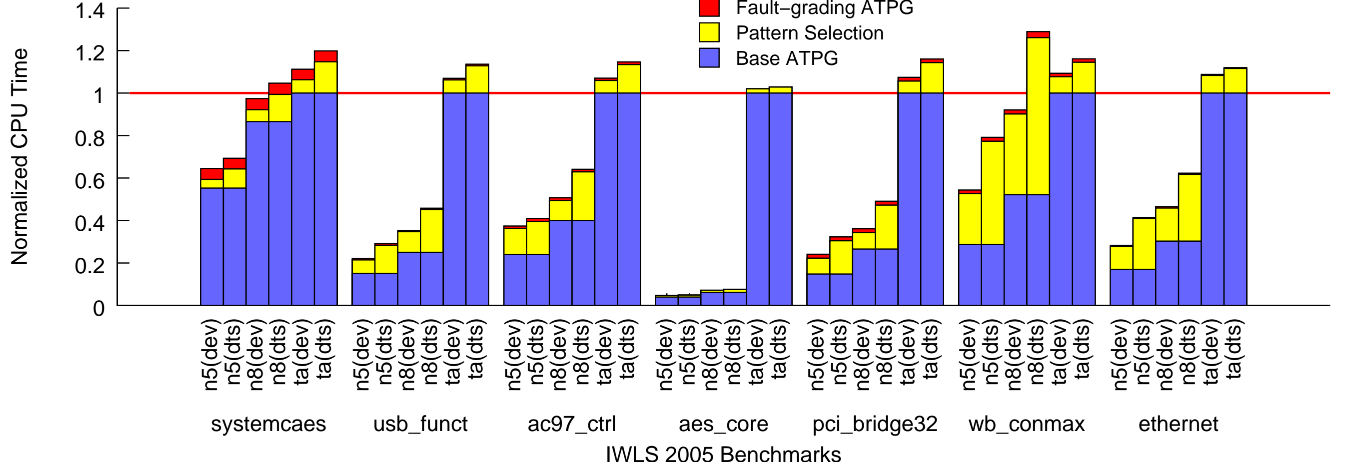


Fig. 7. Comparison of CPU time for `dev` and `dts`. The CPU times used for each step is shown in a different color. The values are normalized by the timing-aware ATPG CPU time for each benchmark. The normalization point (timing-aware ATPG CPU time) is shown as a red line.

TABLE VIII
CPU RUN-TIME RESULTS FOR DEVIATION-BASED PATTERN SELECTION METHOD. THE RESULTS ARE COMPARED TO BASE TIMING-AWARE ATPG. ALL RESULTS ARE GIVEN IN SECONDS.

Benchmarks	**n1**	**n3**	**n5**				**Base timing-aware ATPG**
	Base ATPG	**Base ATPG**	**Base ATPG**	**Pattern Selection**	**Top-off ATPG**	**Total**	
systemcaes	11.2	37.9	63.9	4.75	5.94	74.59	115.6
usb_funct	3.3	17	31.4	13.25	1.34	45.99	207.8
ac97_ctrl	6.1	14.3	26.7	13.62	1.35	41.68	111.4
aes_core	37.1	46.7	74.7	13	1.05	88.75	1892.9
pci_bridge32	7.7	34.5	73.3	36.62	9.09	119.01	493.8
wb_conmax	16.7	106.7	237.7	197.62	14.12	449.45	827
ethernet	128.3	444.3	953.6	596.88	31.71	1582.19	5589.5

the clock period. The results for 90% long-path limit is shown in Fig. 8. The results for other values of LPL follow the same trend. The contribution of the selected patterns and the top-off patterns is shown in different colors. Note that the number of patterns for the base timing-aware ATPG is much larger than any of the pattern selection methods. Therefore, we trimmed the base timing-aware ATPG patterns and selected the first P patterns (ranked by the ATPG tool) as a baseline where P is the maximum of the number of selected patterns (including top-off patterns) either by `dev` or `dts`. The results for the trimmed timing-aware ATPG patterns are shown as horizontal lines. As seen in Fig. 8, `dev` consistently excites more long paths than `dts`. We also note that the proposed method sensitizes more long paths than the timing-aware ATPG baseline with P patterns for three circuits—ac97_ctrl, pci_bridge32, and wb_conmax. It sensitizes nearly the same number of long paths for systemcaes and ethernet. Recall that in all cases, the CPU time for timing-aware ATPG is much higher (Fig. 7).

We also report the results for the full (untrimmed) ATPG pattern sets in Figs. 9-22. The number of sensitized distinct long paths for the corresponding full ATPG pattern set (n5, n8, and timing-aware) is shown as a red horizontal lines. These lines simply show the maximum achievable limit for each selected pattern set (the blue bars under them). However, with the aid of top-off ATPG patterns, it is possible to exceed these limits.

There are two sets of figures. Figs. 9-15 shows the results when a deviation limit of 0.8 is used. When the deviation limit is lowered to 0.3 (Figs. 16-22), we expect to see an increase in

TABLE IX
CPU RUN-TIME RESULTS FOR DEVIATION-BASED PATTERN SELECTION METHOD. THE RESULTS ARE COMPARED TO BASE TIMING-AWARE ATPG. ALL RESULTS ARE GIVEN IN SECONDS.

Benchmarks	n1	n3	n8				Base timing-aware ATPG
	Base ATPG	Base ATPG	Base ATPG	Pattern Selection	Top-off ATPG	Total	
systemcaes	11.2	37.9	100.1	6.38	6.15	112.62	115.6
usb_funct	3.3	17	52	20.12	1.29	73.41	207.8
ac97_ctrl	6.1	14.3	44.5	10.5	1.44	56.44	111.4
aes_core	37.1	46.7	117.5	18	0.93	136.43	1892.9
pci_bridge32	7.7	34.5	131.1	38.12	9.06	178.29	493.8
wb_conmax	16.7	106.7	431.3	314.38	15.4	761.07	827
ethernet	128.3	444.3	1695.8	867.75	33.41	2596.96	5589.5

TABLE X
CPU RUN-TIME RESULTS FOR DEVIATION-BASED PATTERN SELECTION METHOD. THE RESULTS ARE COMPARED TO BASE TIMING-AWARE ATPG. ALL RESULTS ARE GIVEN IN SECONDS.

Benchmarks	n1	n3	ta				Base timing-aware ATPG
	Base ATPG	Base ATPG	Base ATPG	Pattern Selection	Top-off ATPG	Total	
systemcaes	11.2	37.9	115.6	7.25	5.69	128.54	115.6
usb_funct	3.3	17	207.8	12.88	1.49	222.16	207.8
ac97_ctrl	6.1	14.3	111.4	6.62	1.16	119.19	111.4
aes_core	37.1	46.7	1892.9	38.25	1.06	1932.21	1892.9
pci_bridge32	7.7	34.5	493.8	27.75	8.88	530.42	493.8
wb_conmax	16.7	106.7	827	63.5	13.36	903.86	827
ethernet	128.3	444.3	5589.5	466.38	25.38	6081.25	5589.5

TABLE XI
CPU RUN-TIME RESULTS FOR DYNAMIC-TIMING-SIMULATION-BASED PATTERN SELECTION METHOD. THE RESULTS ARE COMPARED TO BASE TIMING-AWARE ATPG. ALL RESULTS ARE GIVEN IN SECONDS.

Benchmarks	n1	n3	n5				Base timing-aware ATPG
	Base ATPG	Base ATPG	Base ATPG	Pattern Selection	Top-off ATPG	Total	
systemcaes	11.2	37.9	63.9	10.43	5.78	80.1	115.6
usb_funct	3.3	17	31.4	27.65	1.51	60.56	207.8
ac97_ctrl	6.1	14.3	26.7	17.33	1.64	45.67	111.4
aes_core	37.1	46.7	74.7	17.46	1.2	93.36	1892.9
pci_bridge32	7.7	34.5	73.3	77.23	9.05	159.58	493.8
wb_conmax	16.7	106.7	237.7	402.02	15.16	654.88	827
ethernet	128.3	444.3	953.6	1335.18	27.04	2315.82	5589.5

TABLE XII
CPU RUN-TIME RESULTS FOR DYNAMIC-TIMING-SIMULATION-BASED PATTERN SELECTION METHOD. THE RESULTS ARE COMPARED TO BASE TIMING-AWARE ATPG. ALL RESULTS ARE GIVEN IN SECONDS.

Benchmarks	n1	n3	n8				Base timing-aware ATPG
	Base ATPG	Base ATPG	Base ATPG	Pattern Selection	Top-off ATPG	Total	
systemcaes	11.2	37.9	100.1	14.82	6.04	120.96	115.6
usb_funct	3.3	17	52	41.71	1.45	95.16	207.8
ac97_ctrl	6.1	14.3	44.5	25.54	1.43	71.47	111.4
aes_core	37.1	46.7	117.5	24.64	1.65	143.78	1892.9
pci_bridge32	7.7	34.5	131.1	102.35	8.86	242.31	493.8
wb_conmax	16.7	106.7	431.3	611.82	23.05	1066.17	827
ethernet	128.3	444.3	1695.8	1755.33	30.79	3481.92	5589.5

the number of selected patterns. These figures can be compared to Fig. 8 that has a single horizontal line showing the number of sensitized long paths for a trimmed timing-aware ATPG set.

A table of pattern counts for each method is shown along with the chart. As expected, the full timing-aware test sets provide higher coverage, but these test sets are much longer in length. For several benchmarks, comparable or even better coverage is achieved using the proposed method with much smaller test sets. For a range of smaller pattern counts, the proposed method clearly outperforms the full timing-aware test sets.

It is also important to cover most long paths as fast as possible to decrease the number of test patterns. We present the long-path sensitization ramp-up using the selected patterns of dev and dts, and the base timing-aware ATPG patterns in Figs. 23-25. Long paths are selected to be the paths with at least 90% of the clock period. Figs. 9-15 show the cumulative results, whereas Figs. 23-25 show the long path coverage over time. As seen, the deviation-based pattern selection method excites long paths much faster than timing-aware ATPG.

To further evaluate the long path sensitization capability of

TABLE XIII
CPU RUN-TIME RESULTS FOR DYNAMIC-TIMING-SIMULATION-BASED PATTERN SELECTION METHOD. THE RESULTS ARE COMPARED TO BASE TIMING-AWARE ATPG. ALL RESULTS ARE GIVEN IN SECONDS.

Benchmarks	n1	n3	ta				Base timing-aware ATPG
	Base ATPG	Base ATPG	Base ATPG	Pattern Selection	Top-off ATPG	Total	
systemcaes	11.2	37.9	115.6	17.01	5.91	138.53	115.6
usb_funct	3.3	17	207.8	26.72	1.52	236.04	207.8
ac97_ctrl	6.1	14.3	111.4	15.01	1.32	127.73	111.4
aes_core	37.1	46.7	1892.9	54.02	0.89	1947.81	1892.9
pci_bridge32	7.7	34.5	493.8	70.69	8.56	573.05	493.8
wb_conmax	16.7	106.7	827	119.71	13.76	960.47	827
ethernet	128.3	444.3	5589.5	647.57	22.35	6259.42	5589.5

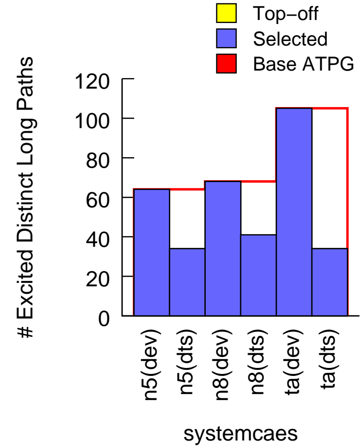
TABLE XIV
CPU RUN-TIME RESULTS FOR LARGER BENCHMARKS. ALL RESULTS ARE GIVEN IN SECONDS.

Benchmarks	# of Gates	ATPG CPU_time (seconds)					
		n1	n3	n5	n8	n10	Timing-aware ATPG
vga_lcd	252302	376	1062	2277.3	4103.3	6438.6	10332.6
netcard	1356533	9605.7	38944.9	85202.1	Not run	Not run	2008786
leon3mp	1452517	13476	44723.1	88611.2	Not run	Not run	41451.9

TABLE VI
NUMBER OF TEST PATTERNS (ONE-THIRD OF BASE SET) SELECTED BY THE DYNAMIC-TIMING SIMULATION BASED SCHEME (DTS).

Benchmarks	n5*	n8*	ta*
systemcaes	646 (784)	940 (991)	999 (1151)
usb_funct	1267 (1474)	1932 (2010)	1226 (1695)
ac97_ctrl	635 (815)	955 (1019)	551 (783)
aes_core	911 (1097)	1331 (1384)	2759 (2766)
pci_bridge32	1596 (1951)	2461 (2693)	1507 (2069)
wb_conmax	7076 (7831)	10729 (10961)	2142 (5713)
ethernet	8407 (10213)	12284 (12877)	4514 (9708)

*Numbers in parenthesis refer to the pattern count after top-off timing-unaware ATPG.



(a)

Simulation	Pattern Count			
	Selected	Top-off	Total	Base ATPG
n5(dev)	938	89	1027	1939
n5(dts)	646	138	784	
n8(dev)	1097	77	1174	2821
n8(dts)	940	51	991	
ta(dev)	1205	169	1374	2997
ta(dts)	999	152	1151	

(b)

Fig. 9. The number of sensitized distinct long paths for `dev` and `dts` (`LPL`=90%) for benchmark systemcaes.: (a) Long path coverage graph; (b) Corresponding pattern counts. The number of sensitized distinct long paths for the full ATPG patterns is shown as horizontal red lines.

the selected patterns, we determined the delay distribution of the excited distinct long paths. We set `LPL` to 80% of the rated clock period. The results are shown in Fig. 26. The results shown in this figure are for the base 8-detect pattern set (`n8(base)`), `dts`-based selected patterns and top-off patterns (`n8(dts)`), and `dev`-based selected patterns with $D_{LIMIT} = 0.8$ and top-off patterns (`n8(dev)`). To draw appropriate conclusions, it is necessary to examine the pattern counts in Table V (column **n8**), Table VI (column **n8**), and Table VII (column **n8** under $D_{LIMIT} = 0.8$). Note that the base pattern set has a much larger pattern count compared to selected pattern sets with additional top-off patterns. Furthermore, `dev`-based selection leads to lower pattern count than `dts` for all benchmarks except systemcaes and ethernet. The difference in pattern count is especially significant for pci_bridge32 and wb_conmax, where `dev`-based selection has almost half of the pattern count of `dts`. We find that `dev`-based selection clearly outperforms `dts`-based selection for most benchmarks, even with fewer patterns. For pci_bridge32 and wb_conmax, `dev`-based selection is almost as effective as `dts`-based selection even though the pattern count is only half. Another important observation is that `dev`-based selection successfully extracted almost all of the long path sensitizing patterns from the base pattern sets for most benchmarks.

To evaluate the effect of DDPM perturbations to the selected pattern quality, we ran perturbation analysis simulations. Figs 28-34 show the results for all benchmarks. For each benchmark, we injected random variations to DDPM entries using three different maximum variation values: 10% (p:0.1), 20% (p:0.2), and 30% (p:0.3). The injected variations are

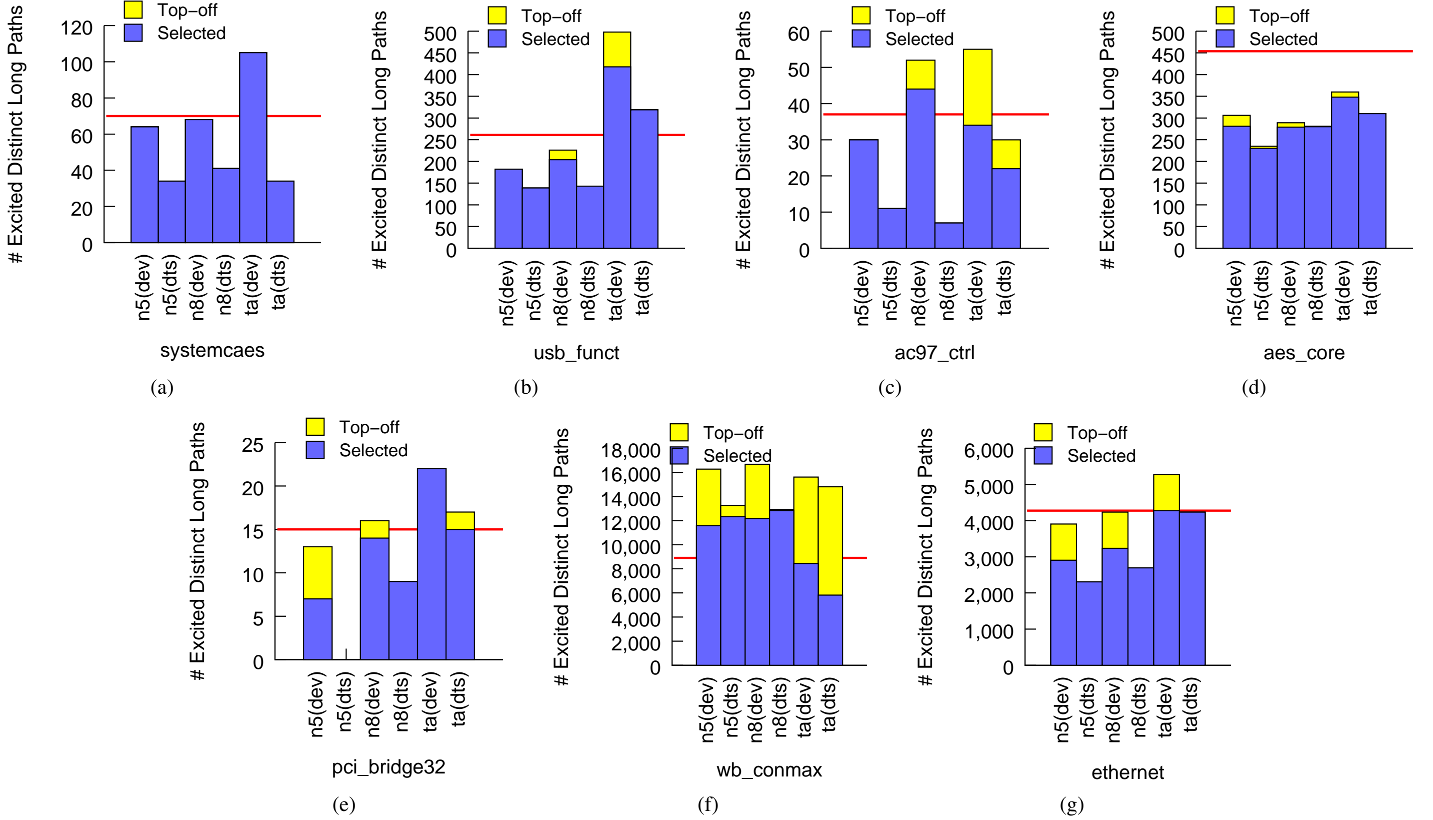


Fig. 8. The number of sensitized distinct long paths for `dev` and `dts` (`LPL`=90%).: (a) systemcaes (2997); (b) usb_funct (3683); (c) ac97_ctrl (1643); (d) aes_core (8277); (e) pci_bridge32 (4571); (f) wb_conmax (6418); (g) ethernet (13544). The number of sensitized distinct long paths for the trimmed timing-aware ATPG patterns is shown as a horizontal line. The number of sensitized distinct long paths for the full timing-aware ATPG pattern set is shown in parentheses next to benchmark names.

uniformly distributed between BASE-p and BASE+p, where BASE shows the original DDPM entry (corresponding bar is named as p:0). The number of distinct long paths excited by the selected patterns is shown as a pattern quality metric. The number of excited distinct long paths for the corresponding full ATPG pattern sets (n5, n8, and timing-aware) are shown as a red horizontal lines. These lines simply show the maximum achievable limit for each selected pattern set (the blue bars under them). For each benchmark, the results are shown for two different long path limits: 90% of the clock period and 70% of the clock period. As seen in Figs. 28-34, the pattern-selection results and test quality are relatively insensitive to small changes in the DDPM entries. We attribute this finding to the fact that any DDPM changes affect multiple paths in the circuits, hence their impact is amortized over the circuit and the test set. It is usually very difficult to estimate close-to-real timing variations. The robustness of the deviation-based pattern selection against timing estimation errors can provide great flexibility in this aspect.

To evaluate the fault coverage ramp-up provided by `dev`, `dts`, and timing-aware ATPG, we ran fault injection simulations. For each benchmark, we inserted 50000 delay defects on randomly chosen nets. We assumed that the additional delay introduced by the injected defects has a distribution of e^{-Ax} as used in [9] and [26], and we injected random delay defects using the method described below.

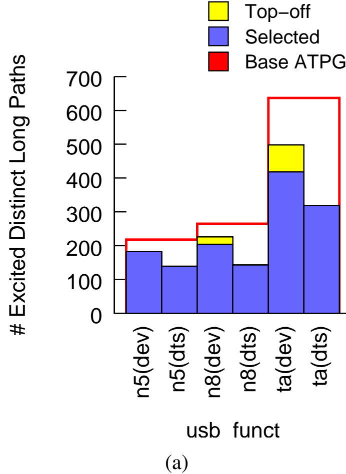
We let $A = \frac{5}{T_{CLK}}$, where T_{CLK} corresponds to the rated clock period of the circuit under test. The delay-defect distribution used in [9] and [26] is similar to the one shown in Fig. 35. However, on the y-axis, uniformly distributed random numbers are shown instead of the number of parts. We use the distribution

$$y = e^{\frac{-5x}{T_{CLK}}}, 0 < y < 1. \quad (5)$$

If Equation (5) is solved to determine x (additional delay) in terms of y (uniformly distributed random number), the following equation results:

$$x = -\ln y \cdot \frac{T_{CLK}}{5}. \quad (6)$$

The graph corresponding to Equations (5) and (6) is shown in Fig. 35 for $T_{CLK} = 100ps$. A total of 50000 uniformly distributed random numbers were generated and corresponding delay defects were injected in the circuit under test. As seen in Fig. 35, 70% of the injected delay defects are less than 20% of the clock period. Thus, our defect-injection mechanism injected more small-delay defects than gross-delay defects, as is the case for VDSM technology [9].

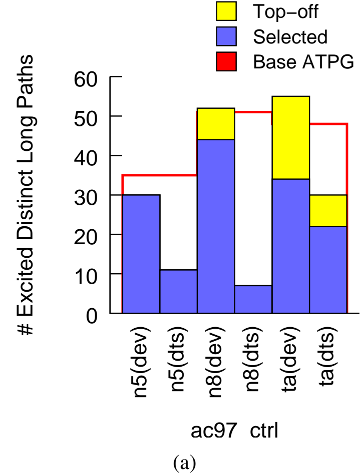


(a)

Simulation	Pattern Count			
	Selected	Top-off	Total	Base ATPG
n5(dev)	938	89	1027	1939
n5(dts)	646	138	784	
n8(dev)	1097	77	1174	2821
n8(dts)	940	51	991	
ta(dev)	1205	169	1374	2997
ta(dts)	999	152	1151	

(b)

Fig. 10. The number of sensitized distinct long paths for `dev` and `dts` (`LPL`=90%) for benchmark usb_funct.: (a) Long path coverage graph; (b) Corresponding pattern counts. The number of sensitized distinct long paths for the full ATPG patterns is shown as horizontal red lines.



(a)

Simulation	Pattern Count			
	Selected	Top-off	Total	Base ATPG
n5(dev)	281	308	589	1907
n5(dts)	635	180	815	
n8(dev)	246	336	582	2866
n8(dts)	955	64	1019	
ta(dev)	63	481	544	1643
ta(dts)	551	232	783	

(b)

Fig. 11. The number of sensitized distinct long paths for `dev` and `dts` (`LPL`=90%) for benchmark ac97_ctrl.: (a) Long path coverage graph; (b) Corresponding pattern counts. The number of sensitized distinct long paths for the full ATPG patterns is shown as horizontal red lines.

The number of detected faults for all the benchmarks is presented as Venn diagrams in Fig. 36. Although it is only expected that `dev` patterns will not catch all detectable faults, we find that, for most cases, `dev` missed less faults compared to both `dts` and timing-aware ATPG.

Early detection of defects is also important in an abort-on-first-fail methodology, and it can save considerable test time. Furthermore, if the test-time budget is limited, whereby truncation is necessary and only a small portion of the patterns can be used for production test, it is important to apply the most effective patterns before less effective ones. Fig. 37-38 show how the fault coverage increases with the number of patterns for all benchmark circuits. Each plot in these figures shows results for the base timing-aware ATPG patterns (`ta(base)`), the patterns selected or sorted by `dev` (`n8(dev)` and `ta(dev)`), and the pattern selected and sorted by `dts` (`n8(dts)` and `ta(dts)`). We find that the coverage rises more steeply for the proposed method (`dev`) compared to both `dts` and the base timing-aware ATPG patterns.

V. 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 results for the IWLS'2005 benchmark 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 a larger number of long paths compared to commercial timing-aware ATPG tool. We have also shown that, the selected patterns are considerably more effective than a recently proposed method for detecting small delay defects caused by resistive shorts, resistive opens, and process variations.

Acknowledgements

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

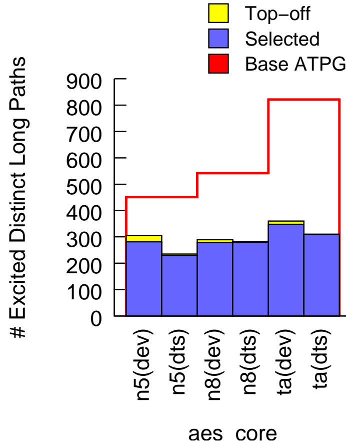
References

[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] X. Lin et al., "Timing-aware ATPG for high quality at-speed testing of small delay defects," in *Proc. IEEE Asian Test Symp.*, 2006, pp. 139–146.

[3] J. A. Waicukauski et al., "Transition fault simulation," *IEEE Design and Test of Computers*, pp. 32–38, 1987.

[4] 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.

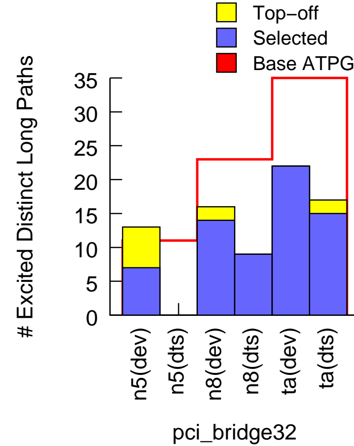


(a)

Simulation	Pattern Count			
	Selected	Top-off	Total	Base ATPG
n5(dev)	1060	122	1182	2734
n5(dts)	911	186	1097	
n8(dev)	1179	98	1277	3995
n8(dts)	1331	53	1384	
ta(dev)	1294	119	1413	8277
ta(dts)	2759	7	2766	

(b)

Fig. 12. The number of sensitized distinct long paths for `dev` and `dts` (`LPL`=90%) for benchmark aes_core.: (a) Long path coverage graph; (b) Corresponding pattern counts. The number of sensitized distinct long paths for the full ATPG patterns is shown as horizontal red lines.



(a)

Simulation	Pattern Count			
	Selected	Top-off	Total	Base ATPG
n5(dev)	408	911	1319	4789
n5(dts)	1596	355	1951	
n8(dev)	459	882	1341	7383
n8(dts)	2461	232	2693	
ta(dev)	504	948	1452	4571
ta(dts)	1507	562	2069	

(b)

Fig. 13. The number of sensitized distinct long paths for `dev` and `dts` (`LPL`=90%) for benchmark pci_bridge32.: (a) Long path coverage graph; (b) Corresponding pattern counts. The number of sensitized distinct long paths for the full ATPG patterns is shown as horizontal red lines.

[5] Cadence Inc., “Encounter test - test generation and simulation reference,” product Version 3.0, 2005.

[6] Mentor Graphics, “Understanding how to run timing-aware ATPG,” application Note, 2006.

[7] R. Kapur, J. Zejda, and T. W. Williams, “Fundamentals of timing information for test: How simple can we get?” in *Proc. IEEE Int. Test Conference*, 2007.

[8] IWLS 2005 Benchmarks, “http://iwls.org/iwls2005/benchmarks.html.”

[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. J. McCluskey, “Classifying bad chips and ordering test sets,” in *Proc. IEEE Int. Test Conference*, 2006.

[11] Semiconductor Industry Association, “(2007) international technology roadmap for semiconductors (ITRS), http://www.itrs.net/links/2007itrs/home2007.htm.”

[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] Z. Wang and K. Chakrabarty, “An efficient test pattern selection method for improving defect coverage with reduced test data volume and test application time,” in *Proc. IEEE Asian Test Symp.*, 2006, pp. 333–338.

[14] H. Lee, S. Natarajan, S. Patil, and I. Pomeranz, “Selecting high-quality delay tests for manufacturing test and debug,” in *Proc. IEEE Int. Symp. on Defect and Fault Tolerance in VLSI Systems*, 2006, pp. 59 – 70.

[15] E. S. Park, M. R. Mercer, and T. W. Williams, “Statistical delay fault coverage and defect level for delay faults,” in *Proc. IEEE Int. Test Conference*, 1988, pp. 492–499.

[16] P. Gupta and M. S. Hsiao, “ALAPTF: A new transition fault model and the ATPG algorithm,” in *Proc. IEEE Int. Test Conference*, 2004, pp. 1053–1060.

[17] W. Qiu et al., “K longest paths per gate (KLPG) test generation for scan-based sequential circuits,” in *Proc. IEEE Int. Test Conference*, 2004, pp. 223–231.

[18] S. Gurumurthy et al., “Automatic generation of instructions to robustly test delay defects in processors,” in *Proc. IEEE European Test Symp.*, 2007, pp. 173–178.

[19] H. Yan and A. D. Singh, “A new delay test based on delay defect detection within slack intervals (DDSI),” *IEEE Trans. VLSI*, vol. 14, no. 11, pp. 1216–1226, 2006.

[20] I. Pomeranz and S. M. Reddy, “Transition path delay faults: A new path delay fault model for small and large delay defects,” *IEEE Trans. VLSI*, vol. 16, no. 1, pp. 98–107, 2008.

[21] S. Bose, H. Grimes, and V. D. Agrawal, “Delay fault simulation with bounded gate delay mode,” in *Proc. IEEE Int. Test Conference*, 2007.

[22] V. Iyengar, J. Xiong, S. Venkatesan, V. Zolotov, D. Lackey, P. Habitz, and C. Visweswariah, “Variation-aware performance verification using at-speed structural test and statistical timing,” in *Proc. IEEE ICCAD*, 2007, pp. 405–412.

[23] V. Zolotov, J. Xiong, H. Fatemi, and C. Visweswariah, “Statistical path selection for at-speed test,” in *Proc. IEEE ICCAD*, 2008, pp. 624–631.

[24] C. Forzan and D. Pandini, “Why we need statistical static timing analysis,” in *Proc. IEEE ICCD*, 2007, pp. 91–96.

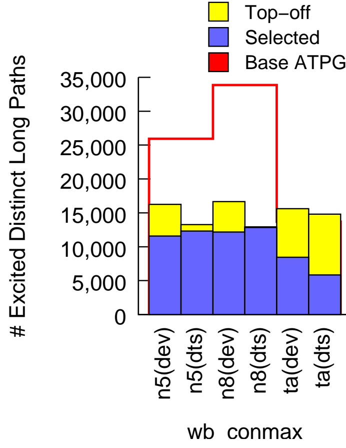
[25] I. Nitta, S. Toshiyuki, and H. Katsumi, “Statistical static timing analysis technology,” *Fujitsu Sci. Tech. Journal*, vol. 43, no. 4, pp. 516–523, Oct 2007.

[26] Y. Sato et al., “Invisible delay quality - SDQM model lights up what could not be seen,” in *Proc. IEEE Int. Test Conference*, 2005.

[27] M. Yilmaz, K. Chakrabarty, and M. Tehranipoor, “Interconnect-aware and layout-oriented test-pattern selection for small-delay defects,” in *Proc. IEEE Int. Test Conference*, 2008.

[28] M. Yilmaz, “Automated test grading and pattern selection for small-delay defects,” Ph.D. dissertation, Duke University, Apr 2009.

[29] B. J. Chalmers, *Understanding Statistics*. Monticello, NY: CRC Press, 1987.

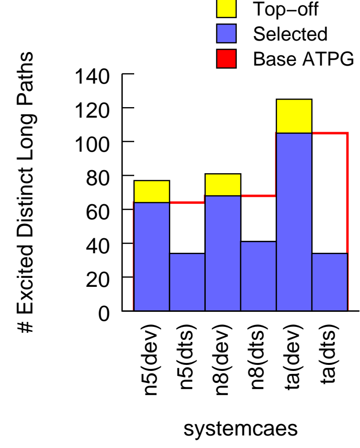


(a)

Simulation	Pattern Count			
	Selected	Top-off	Total	Base ATPG
n5(dev)	2091	3484	5575	21230
n5(dts)	7076	755	7831	
n8(dev)	2275	3260	5535	32189
n8(dts)	10729	232	10961	
ta(dev)	1525	4126	5651	6418
ta(dts)	2142	3571	5713	

(b)

Fig. 14. The number of sensitized distinct long paths for `dev` and `dts` (`LPL`=90%) for benchmark wb_conmax.: (a) Long path coverage graph; (b) Corresponding pattern counts. The number of sensitized distinct long paths for the full ATPG patterns is shown as horizontal red lines.

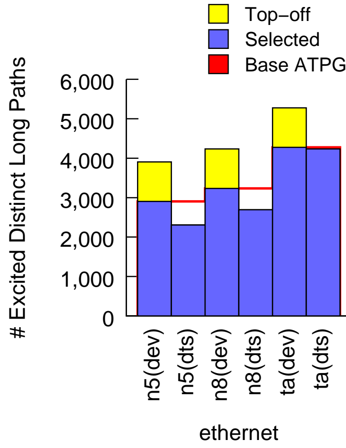


(a)

Simulation	Pattern Count			
	Selected	Top-off	Total	Base ATPG
n5(dev)	978	77	1055	1939
n5(dts)	646	138	784	
n8(dev)	1139	75	1214	2821
n8(dts)	940	51	991	
ta(dev)	1248	172	1420	2997
ta(dts)	999	152	1151	

(b)

Fig. 16. The number of sensitized distinct long paths for `dev` and `dts` (`LPL`=90%) for benchmark systemcaes.: (a) Long path coverage graph; (b) Corresponding pattern counts. The number of sensitized distinct long paths for the full ATPG patterns is shown as horizontal red lines.

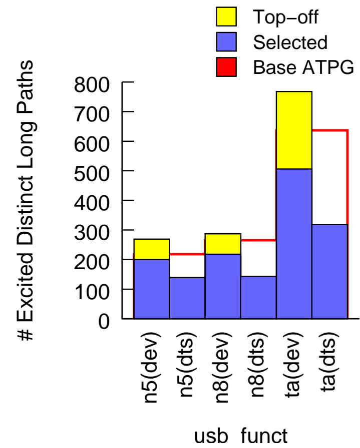


(a)

Simulation	Pattern Count			
	Selected	Top-off	Total	Base ATPG
n5(dev)	11479	1974	13453	25221
n5(dts)	8407	1806	10213	
n8(dev)	12911	1393	14304	36855
n8(dts)	12284	593	12877	
ta(dev)	8493	2858	11351	13544
ta(dts)	4514	5194	9708	

(b)

Fig. 15. The number of sensitized distinct long paths for `dev` and `dts` (`LPL`=90%) for benchmark ethernet.: (a) Long path coverage graph; (b) Corresponding pattern counts. The number of sensitized distinct long paths for the full ATPG patterns is shown as horizontal red lines.

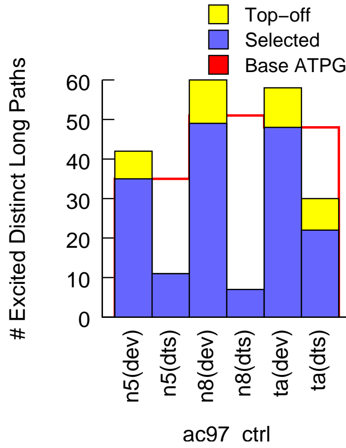


(a)

Simulation	Pattern Count			
	Selected	Top-off	Total	Base ATPG
n5(dev)	1855	52	1907	3802
n5(dts)	1267	207	1474	
n8(dev)	2177	39	2216	5797
n8(dts)	1932	78	2010	
ta(dev)	1987	304	2291	3683
ta(dts)	1226	469	1695	

(b)

Fig. 17. The number of sensitized distinct long paths for `dev` and `dts` (`LPL`=90%) for benchmark usb_funct.: (a) Long path coverage graph; (b) Corresponding pattern counts. The number of sensitized distinct long paths for the full ATPG patterns is shown as horizontal red lines.

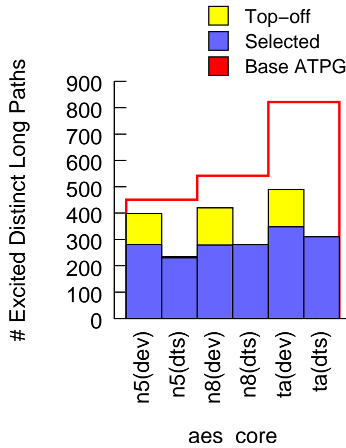


(a)

Simulation	Pattern Count			
	Selected	Top-off	Total	Base ATPG
n5(dev)	1341	2	1343	1907
n5(dts)	635	180	815	
n8(dev)	1655	3	1658	2866
n8(dts)	955	64	1019	
ta(dev)	1346	54	1400	1643
ta(dts)	551	232	783	

(b)

Fig. 18. The number of sensitized distinct long paths for `dev` and `dts` (`LPL`=90%) for benchmark ac97_ctrl.: (a) Long path coverage graph; (b) Corresponding pattern counts. The number of sensitized distinct long paths for the full ATPG patterns is shown as horizontal red lines.

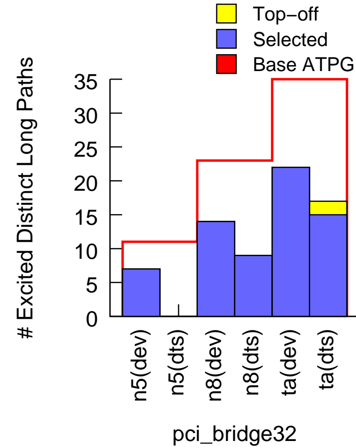


(a)

Simulation	Pattern Count			
	Selected	Top-off	Total	Base ATPG
n5(dev)	1114	105	1219	2734
n5(dts)	911	186	1097	
n8(dev)	1247	79	1326	3995
n8(dts)	1331	53	1384	
ta(dev)	1407	96	1503	8277
ta(dts)	2759	7	2766	

(b)

Fig. 19. The number of sensitized distinct long paths for `dev` and `dts` (`LPL`=90%) for benchmark aes_core.: (a) Long path coverage graph; (b) Corresponding pattern counts. The number of sensitized distinct long paths for the full ATPG patterns is shown as horizontal red lines.

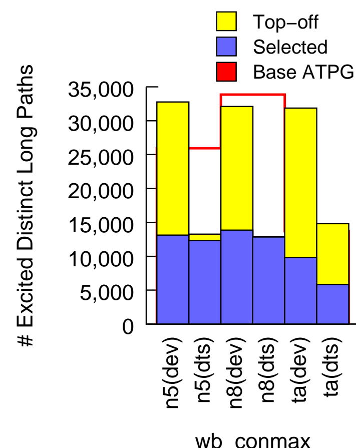


(a)

Simulation	Pattern Count			
	Selected	Top-off	Total	Base ATPG
n5(dev)	2061	145	2206	4789
n5(dts)	1596	355	1951	
n8(dev)	2476	116	2592	7383
n8(dts)	2461	232	2693	
ta(dev)	2438	352	2790	4571
ta(dts)	1507	562	2069	

(b)

Fig. 20. The number of sensitized distinct long paths for `dev` and `dts` (`LPL`=90%) for benchmark pci_bridge32.: (a) Long path coverage graph; (b) Corresponding pattern counts. The number of sensitized distinct long paths for the full ATPG patterns is shown as horizontal red lines.

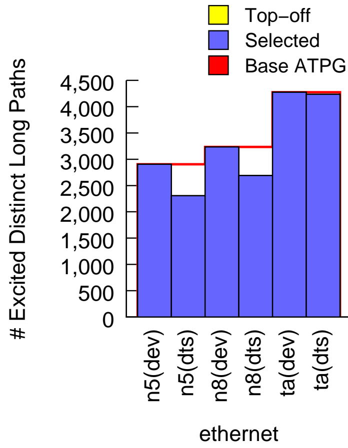


(a)

Simulation	Pattern Count			
	Selected	Top-off	Total	Base ATPG
n5(dev)	2805	2927	5732	21230
n5(dts)	7076	755	7831	
n8(dev)	2959	2806	5765	32189
n8(dts)	10729	232	10961	
ta(dev)	2305	3446	5751	6418
ta(dts)	2142	3571	5713	

(b)

Fig. 21. The number of sensitized distinct long paths for `dev` and `dts` (`LPL`=90%) for benchmark wb_conmax.: (a) Long path coverage graph; (b) Corresponding pattern counts. The number of sensitized distinct long paths for the full ATPG patterns is shown as horizontal red lines.



(a)

Simulation	Pattern Count			
	Selected	Top-off	Total	Base ATPG
n5(dev)	12219	1664	13883	25221
n5(dts)	8407	1806	10213	
n8(dev)	13544	1111	14655	36855
n8(dts)	12284	593	12877	
ta(dev)	8984	2530	11514	13544
ta(dts)	4514	5194	9708	

(b)

Fig. 22. The number of sensitized distinct long paths for `dev` and `dts` (`LPL`=90%) for benchmark ethernet.: (a) Long path coverage graph; (b) Corresponding pattern counts. The number of sensitized distinct long paths for the full ATPG patterns is shown as horizontal red lines.

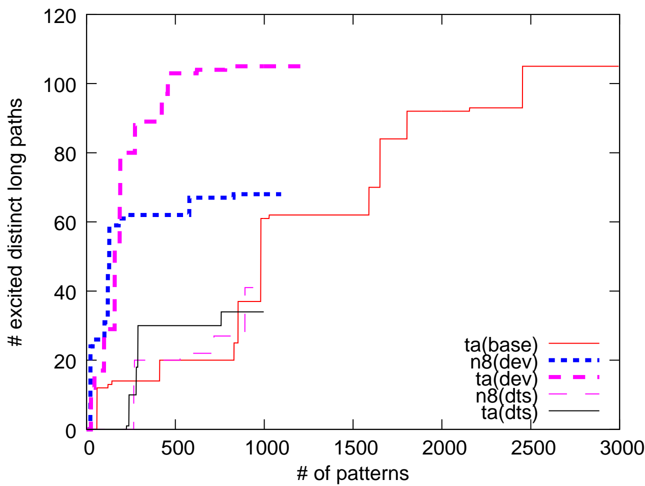
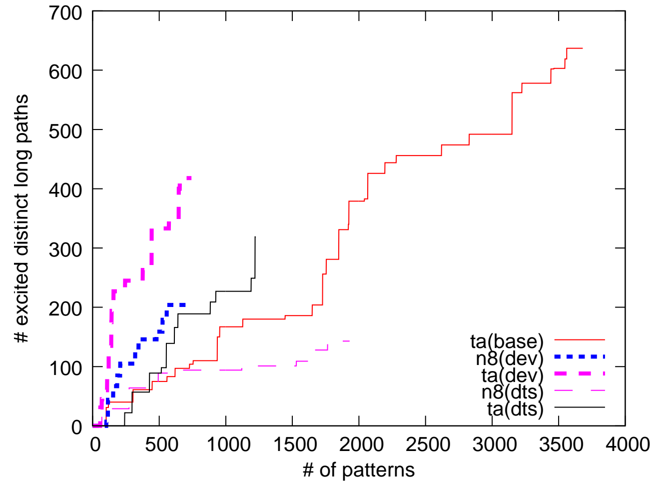
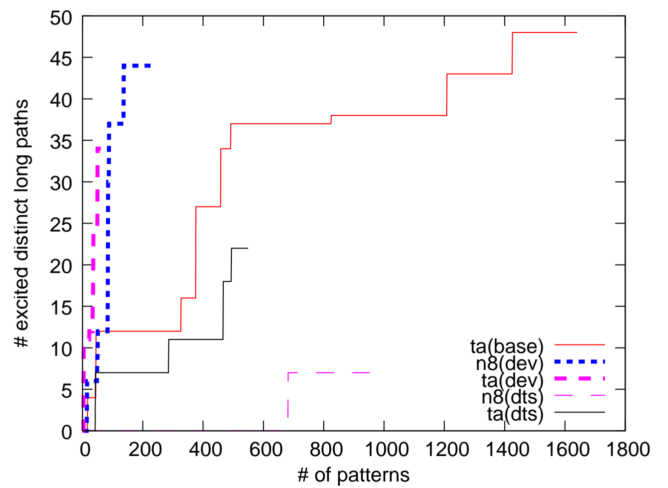


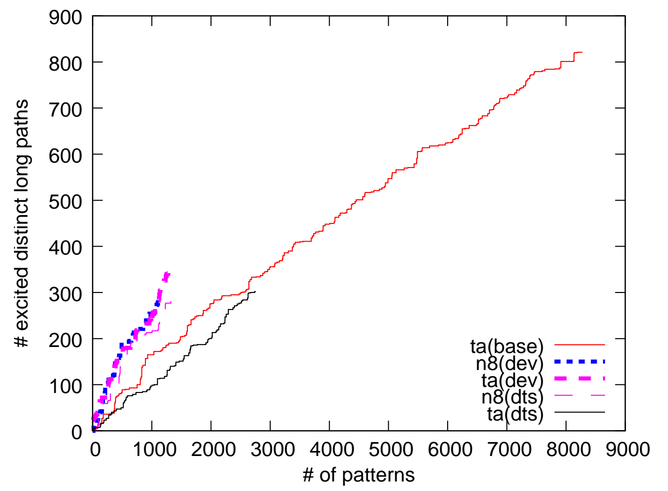
Fig. 23. The long path coverage ramp-up using the selected patterns of dev and dts, and the base timing-aware ATPG patterns: systemcaes.



(a)

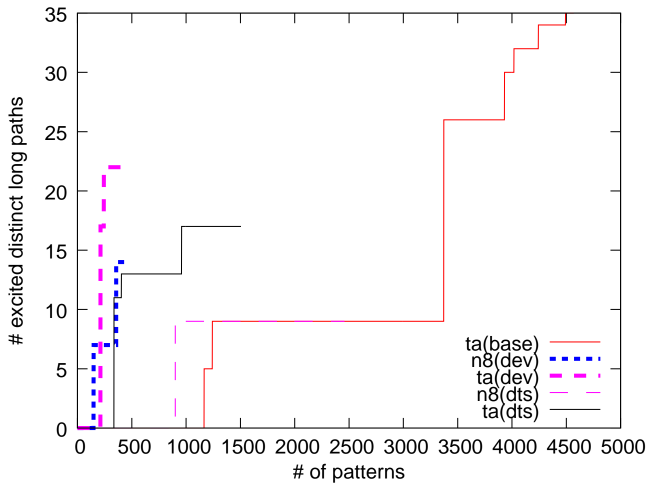


(b)

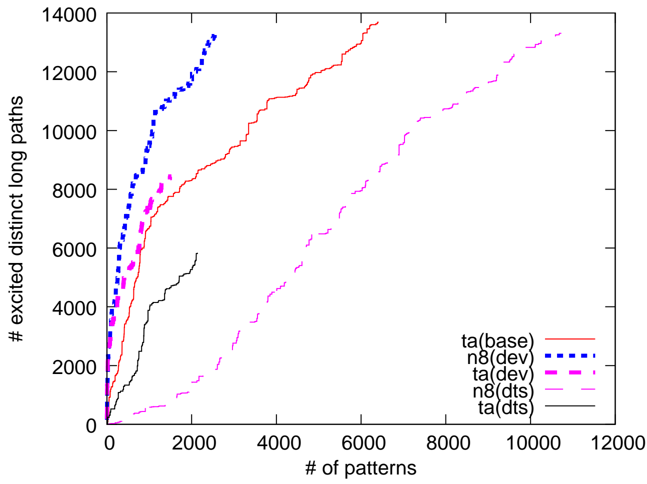


(c)

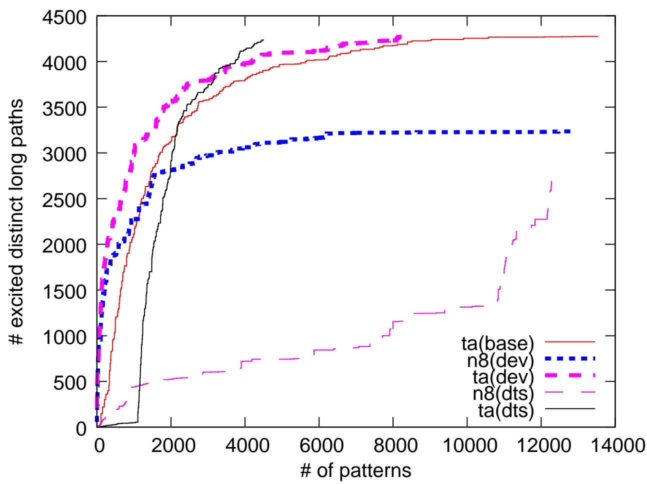
Fig. 24. The long path coverage ramp-up using the selected patterns of dev and dts, and the base timing-aware ATPG patterns: (a) usb_funct; (b) ac97_ctrl (c) aes_core.



(a)

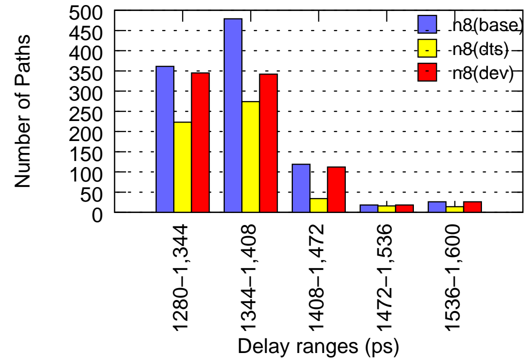


(b)

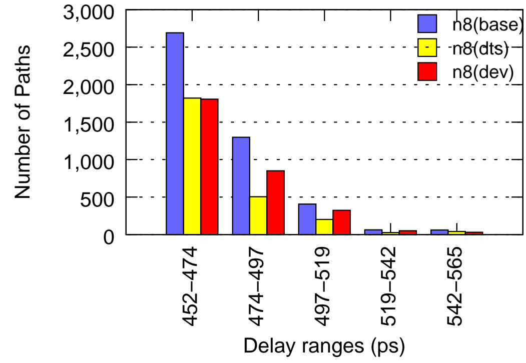


(c)

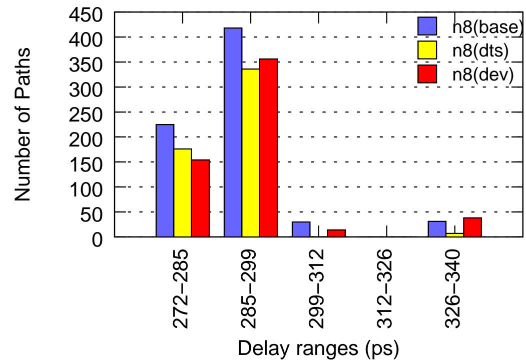
Fig. 25. The long path coverage ramp-up using the selected patterns of dev and dts, and the base timing-aware ATPG patterns: (a) pci_bridge32 (b) wb_conmax; (c) ethernet.



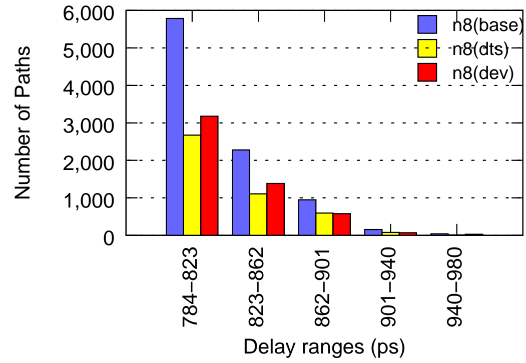
(a)



(b)

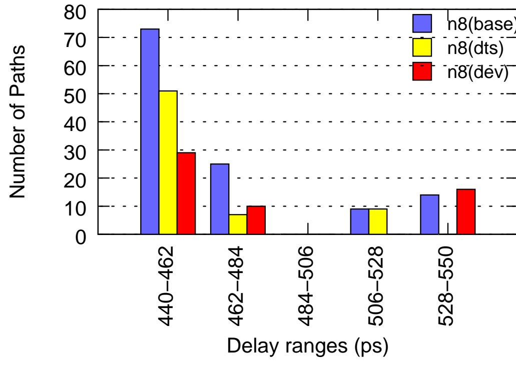


(c)

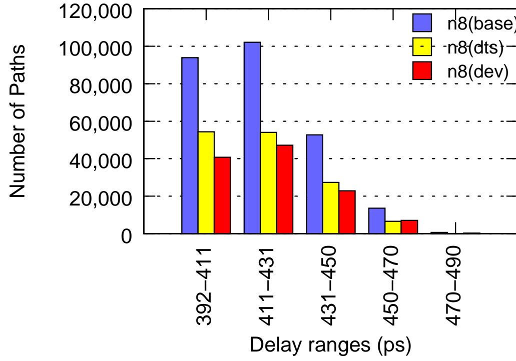


(d)

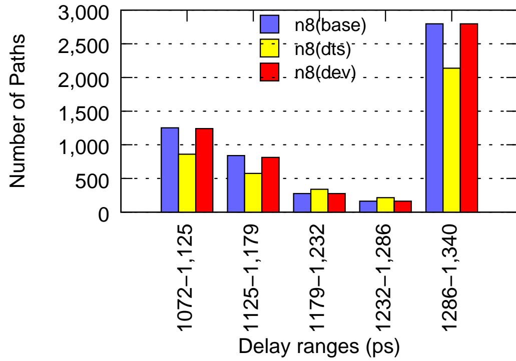
Fig. 26. The distribution of sensitized distinct paths over 80% of the clock period for base 8-detect patterns, `dts`-based selection, and `dev`-based selection: (a) systemcaes; (b) usb_funct; (c) ac97_ctrl; (d) aes_core.



(a)

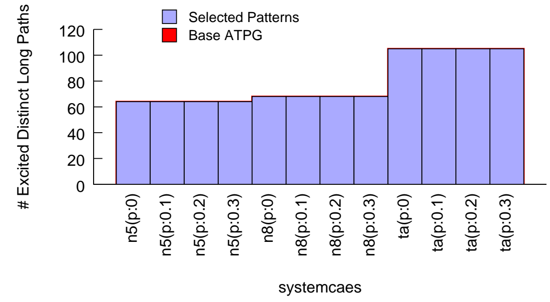


(b)

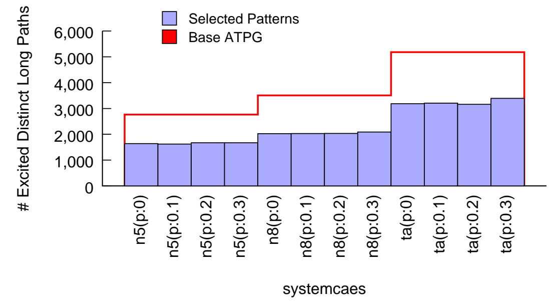


(c)

Fig. 27. The distribution of sensitized distinct paths (LPL=80%) for base 8-detect patterns, dts-based selection, and dev-based selection: a) pci_bridge32; (b) wb_conmax; (c) ethernet.

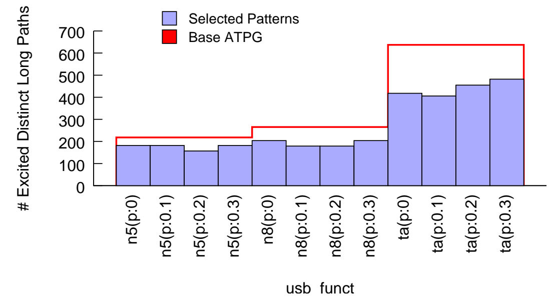


(a)

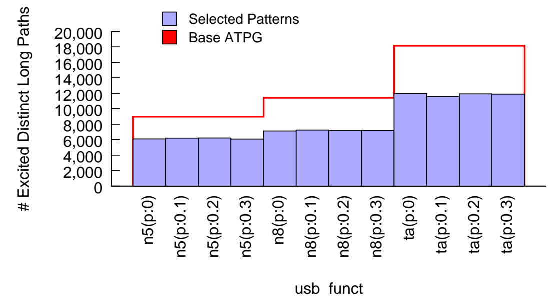


(b)

Fig. 28. The number of sensitized distinct long paths for dev for different base pattern sets and under different DDPM perturbation limits for benchmark systemcaes.: (a) LPL=90%; (b) LPL=70%. The number of sensitized distinct long paths for the full ATPG patterns is shown as horizontal red lines.

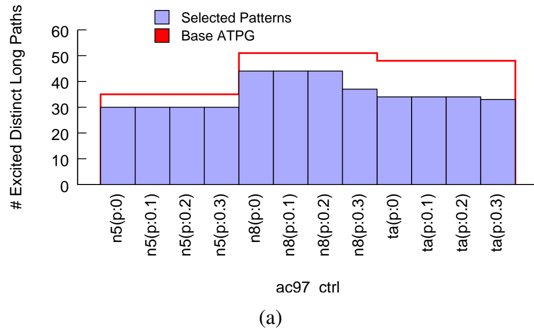


(a)

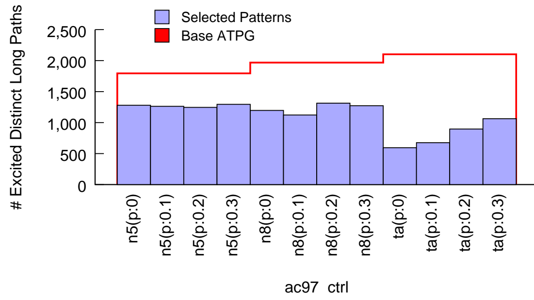


(b)

Fig. 29. The number of sensitized distinct long paths for dev for different base pattern sets and under different DDPM perturbation limits for benchmark usb_funct.: (a) LPL=90%; (b) LPL=70%. The number of sensitized distinct long paths for the full ATPG patterns is shown as horizontal red lines.

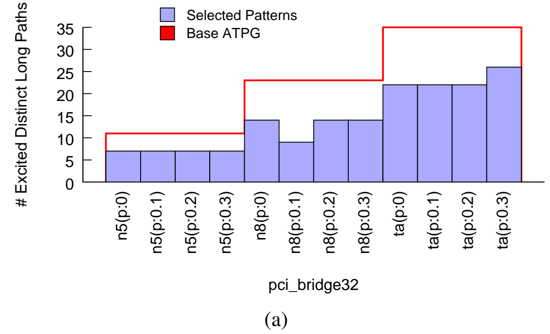


(a)

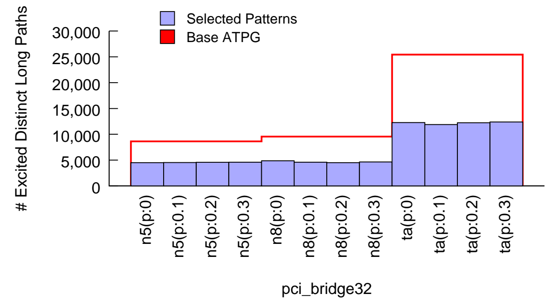


(b)

Fig. 30. The number of sensitized distinct long paths for `dev` for different base pattern sets and under different DDPM perturbation limits for benchmark ac97_ctrl.: (a) `LPL`=90%; (b) `LPL`=70%. The number of sensitized distinct long paths for the full ATPG patterns is shown as horizontal red lines.

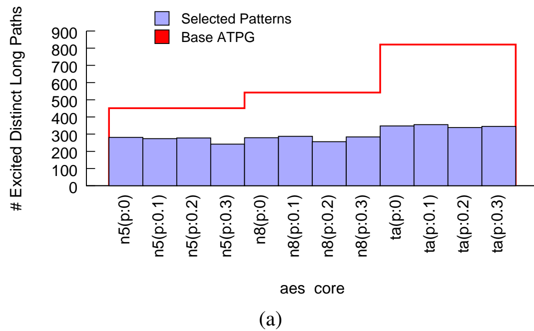


(a)

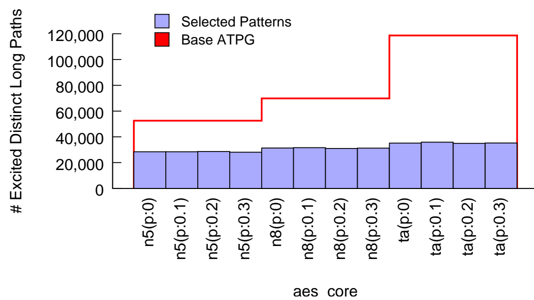


(b)

Fig. 32. The number of sensitized distinct long paths for `dev` for different base pattern sets and under different DDPM perturbation limits for benchmark pci_bridge32.: (a) `LPL`=90%; (b) `LPL`=70%. The number of sensitized distinct long paths for the full ATPG patterns is shown as horizontal red lines.

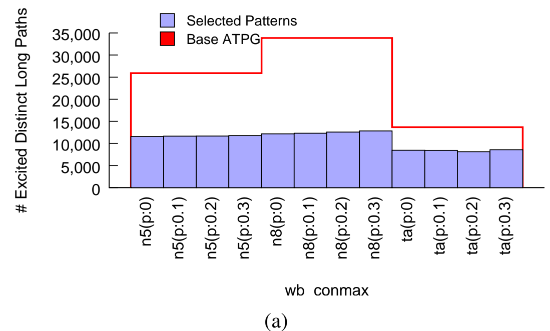


(a)

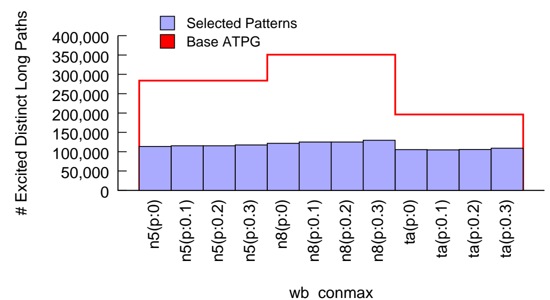


(b)

Fig. 31. The number of sensitized distinct long paths for `dev` for different base pattern sets and under different DDPM perturbation limits for benchmark aes_core.: (a) `LPL`=90%; (b) `LPL`=70%. The number of sensitized distinct long paths for the full ATPG patterns is shown as horizontal red lines.



(a)



(b)

Fig. 33. The number of sensitized distinct long paths for `dev` for different base pattern sets and under different DDPM perturbation limits for benchmark wb_conmax.: (a) `LPL`=90%; (b) `LPL`=70%. The number of sensitized distinct long paths for the full ATPG patterns is shown as horizontal red lines.

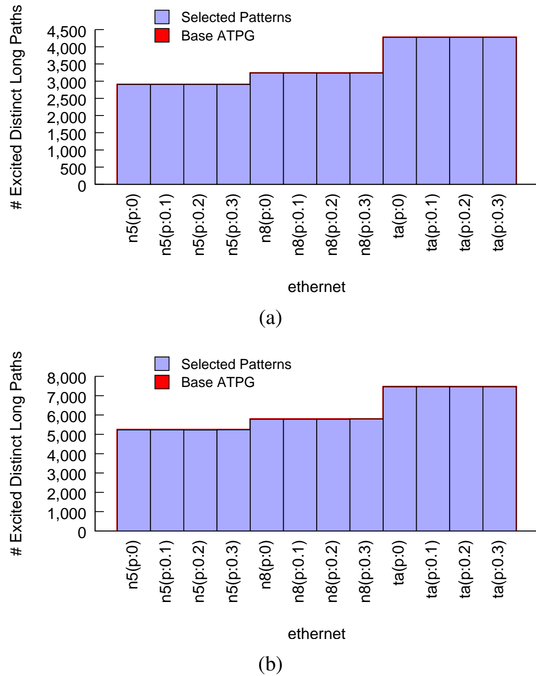


Fig. 34. The number of sensitized distinct long paths for `dev` for different base pattern sets and under different DDPM perturbation limits for benchmark ethernet.: (a) `LPL`=90%; (b) `LPL`=70%. The number of sensitized distinct long paths for the full ATPG patterns is shown as horizontal red lines.

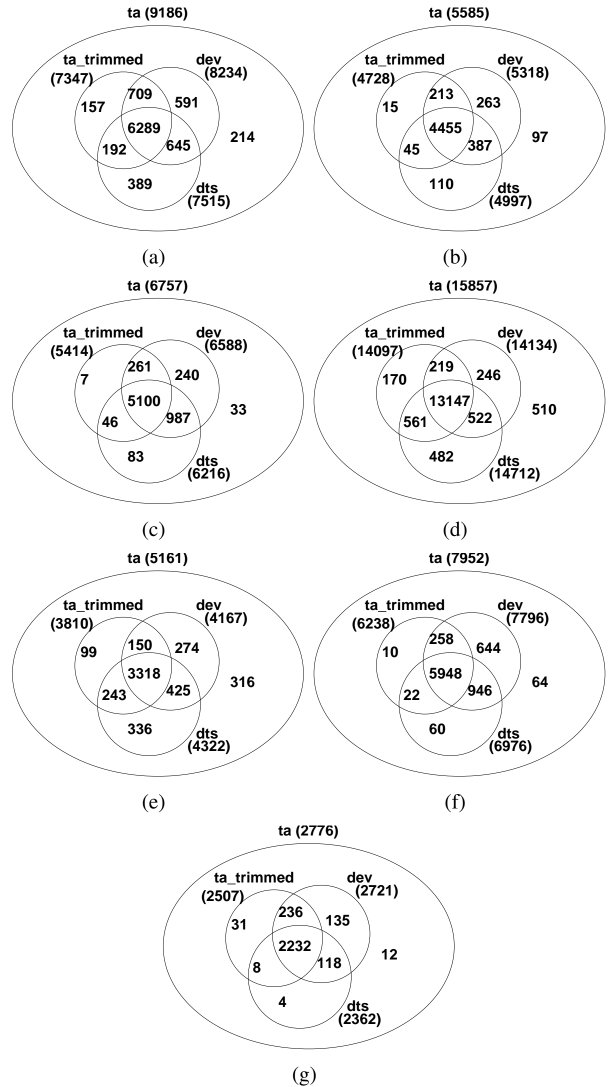


Fig. 36. The number of defects detected by the selected patterns for `dev`, `dts`, the trimmed timing-aware ATPG patterns (`ta_trimmed`), and full set of timing-aware ATPG patterns (`ta`) (`ta` is a super-set of all selected patterns): (a) systemcaes; (b) usb_funct; (c) ac97_ctrl; (d) aes_core; (e) pci_bridge32; (f) wb_conmax; (g) ethernet.

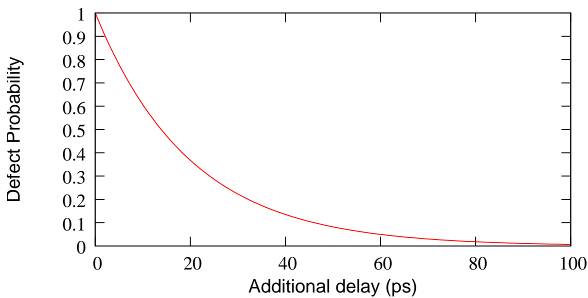


Fig. 35. The distribution of injected delay defects.

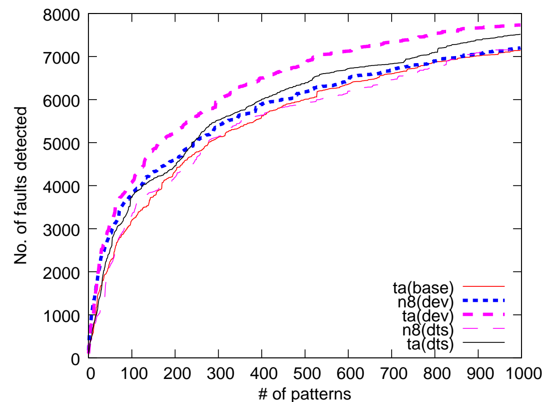


Fig. 37. The fault coverage ramp-up using the selected patterns of `dev` and `dts`, and the `base` timing-aware ATPG patterns: systemcaes.

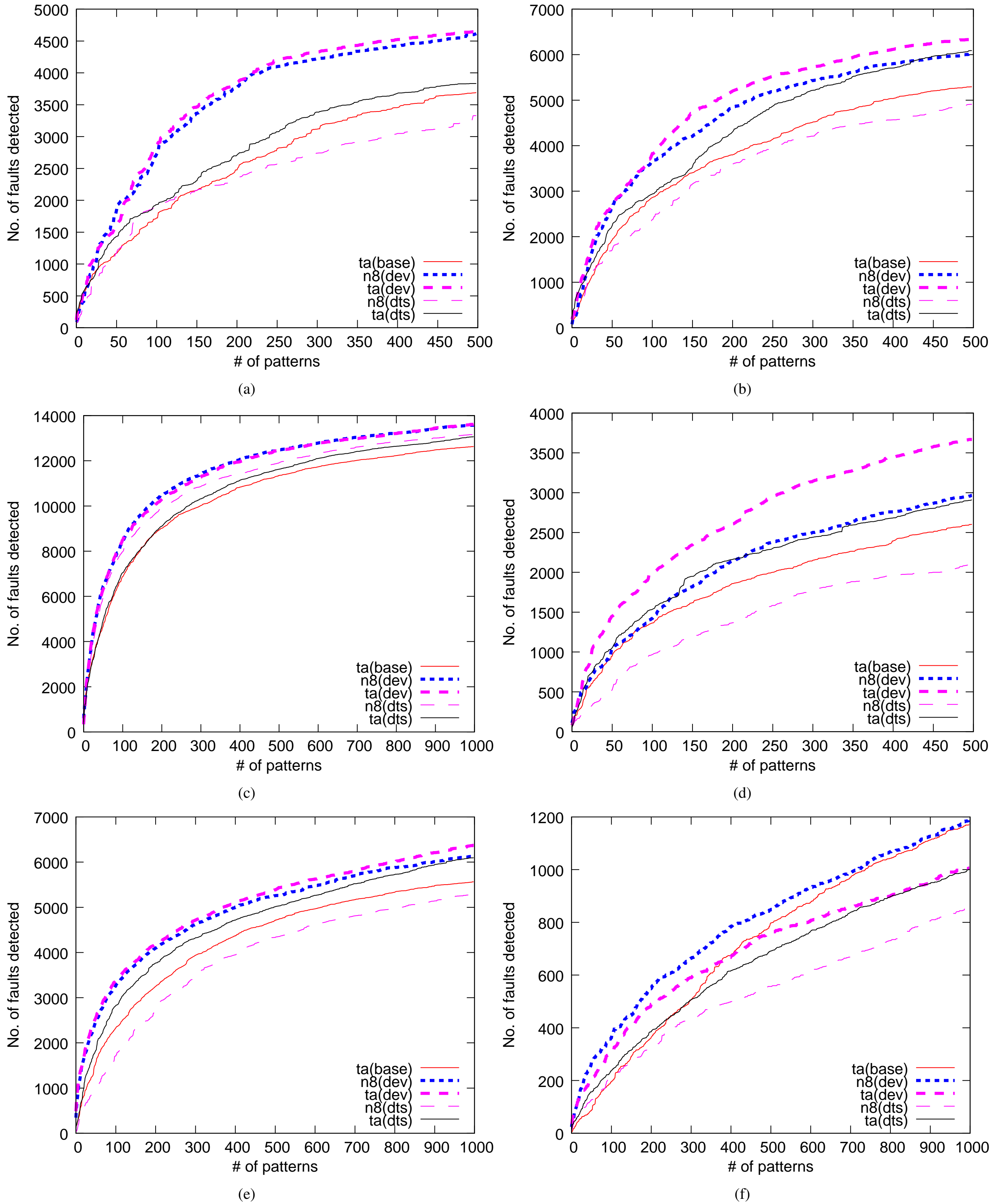


Fig. 38. The fault coverage ramp-up using the selected patterns of `dev` and `dts`, and the `base` timing-aware ATPG patterns: (a) usb_funct; (b) ac97_ctrl; (c) aes_core; (d) pci_bridge32; (e) wb_conmax; (f) ethernet.