Interconnect-Aware and Layout-Oriented Test-Pattern Selection for Small-Delay Defects[1,2]

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

[†]Dept. Electrical and Computer Engineering
Duke University
{my, krish}@ee.duke.edu

[‡]Dept. Electrical and Computer Engineering
University of Connecticut
tehrani@engr.uconn.edu

ABSTRACT

Timing-related failures in high-performance integrated circuits are being increasingly dominated by small-delay defects (SDDs). Such delay faults are caused by process variations, crosstalk, power-supply noise, and defects such as resistive shorts and opens. Recently, the concept of output deviations has been presented as a surrogate long-path coverage metric for SDDs. However, this approach is focused only on delay variations for logic gates and it ignores chip layout, interconnect defects, and delay variations on interconnects. We present a layout-aware output deviations metric that can easily handle interconnect delay variations. Experimental results show that interconnect-delay variations can have a significant impact on the long paths that must be targeted for the detection of SDDs. For the same pattern count, the proposed pattern-grading and pattern-selection method is more effective than a commercial timing-aware ATPG tool for SDDs, and requires considerably less CPU time.

I. INTRODUCTION

Interconnect delays are a major concern for very deep submicron (VDSM) process technologies [1]. The increased susceptibility of VDSM designs to process variations make interconnect delays even more important. Higher process variation and crosstalk effects have resulted in the emergence of interconnects as the major contributor to small-delay defects (SDDs) [2]. New test-generation methods are therefore needed to target SDDs caused by process variation and crosstalk on interconnects.

In this paper, we address the problem of detecting SDDs by selecting the best test patterns from a repository test set for transition-delay faults, which is generated without considering process variations. The proposed method is based on the use of the output deviations metric for delay faults, which was recently introduced in [3]. The output deviations metric is a measure of the probability that an incorrect signal value will be captured at a scan flip-flop. As in [3], the output deviation measure is used as a coverage-metric for SDDs. We show how interconnect delay variations can be easily incorporated in the output deviations metric. We present the impact of interconnect delays and interconnect delay variations on test pattern selection, and the coverage of SDDs.

We start with a base set of n-detect transition delay-fault test patterns and apply our pattern-grading method to measure the effectiveness of each pattern. Next, we apply our pattern ordering method and select best patterns from the initial pattern set. Experimental results show that wire-delay variations must be taken into account in order to select high-quality patterns for today's high performance designs. For the same pattern count, the proposed pattern-grading and pattern-selection method is more effective than a commercial timing-aware ATPG tool for SDDs, and requires considerably less CPU time.

The remainder of the paper is organized as follows. In Section II, we describe technology trends and predictions that motivate the need to incorporate process variation on interconnects in an effective test-pattern grading and selection method. Section III presents an overview of the output deviations metric that was introduced in [3]. In Section IV, we show how variations in wire delays can be incorporated in the output deviations framework. Section V describes the proposed pattern-selection procedure. In Section VI, we present experimental results for the IWLS 2005 benchmarks [4]. Section VII concludes the paper.

II. MOTIVATION AND PRIOR WORK

Interconnects are expected to be a major performance limiter when process technologies shrink to 50nm and below [1], [5]. This is mainly because the technology scale-down in interconnects, unlike transistors, does not always lead to higher performance. ITRS predicts that the wire-delay relative to the gate-delay will increase considerably for smaller technology nodes. Inductive effects due to increasing clock frequencies are expected to push interconnect delays to even higher values, which will force industry to adopt new interconnect technologies [1].

Process variation on wires have a greater effect on circuit delay than process variation for transistors. Studies predict that process variations on interconnects will have a considerable impact on critical- and long-path delays [1], [6], [7]. The reason behind these findings is that interconnects are not only

[1]The work of M. Yilmaz and K. Chakrabarty was supported in part by SRC under Contract number 1588.

[2]The work of M. Tehranipoor was supported in part by SRC under Contract numbers 1455 and 1587.

the dominant contributor to overall path delays, they are also susceptible to large process variations due to lithography effects, etching, gradients, and various random effects [8]. For instance, as much as 40% wire-width variation is estimated for wires of 40nm width [6].

Technology shrinking is expected to have different effects on interconnects at various metal layers. For technologies below 45nm, Metal-1 and intermediate-level metal wires (local wires) are expected to have the same line-widths and thicknesses, thus they will show similar process variation effects. This is mainly because these wires usually shrink when traditional scaling is applied to transistors. The effect of process variations on local metal layers are expected to have smaller effect on overall circuit delay because of their relatively shorter lengths compared to global wires [1], [6]. Global wires, on the other hand, are expected to have increasing nominal delays and larger impact on overall circuit delay due to process variations. A recent study in [7] shows considerable increase in the variation of RC delays as the wire length increases. In addition to metal layers, vias are also affected by process variations. The resistance variation of vias is larger compared to that for metal layers because they tend to have the smallest possible dimensions [8].

Process variations are not the only parameters affecting the interconnect delay, and as a result path delays. Crosstalk effects on wires also have significant impact on a circuit's timing uncertainty [9]. The coupling-line length that can change the voltage levels by 25% of the supply voltage on a victim wire due to crosstalk is expected drop 40% by 2013 [1], [5]. This implies that for the same wire length, there will be much stronger crosstalk effects in the future. Furthermore, at multi-gigahertz clock frequencies, because of the introduction of inductive coupling, crosstalk is expected to have 60% greater impact than predicted by RC models [5].

In view of the above trends and projections, delay-test generation methods that are oblivious of interconnect and layout-related effects are not adequate for next-generation technologies. For instance, the quality of the widely-used traditional transition delay-fault ATPG [10] has often been questioned because of its tendency to excite short paths [11], [12]. As a result, a number of alternative delay-fault pattern generation techniques have been developed. Most of the proposed methods are aimed at finding the longest paths in a circuit. Gupta et al. [13] 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 ignores delay variations and suffers from the need for a complex, time-consuming search procedure and robust test-generation constraints. Qui et al. [14] 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 [13], 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. [11] 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, n-detect transition test patterns are generated. During pattern generation, constraints are applied to 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, delay variations are not considered and 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 [15]. 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 pre-processing step.

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. Such a method must also be a good measure of long-path excitation. An alternative evaluation method, referred to as the "Statistical Delay Quality Model" has been proposed by Sato et al. [16]. 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 production and it is difficult to obtain it during production test.

Due to the growing interest in SDDs, the first commercial timing-aware ATPG tools were introduced recently, e.g., new versions of Mentor Graphics FastScan, Cadence Encounter Test, and Synopsys TetraMax tools [17]–[19]. These tools attempt to make ATPG patterns more effective for SDDs by exercising longer paths. 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 take into account process variations, crosstalk, power-supply noise, or similar SDD-inducing effects on path delays. Instead, these tools rely on the assumption that the longest paths in a design are more prone to SDDs.

Another drawback of timing-aware ATPG tools is that they lead to a considerable increase in test generation time and pattern count compared to a traditional timing-unaware transition delay-fault (TDF) ATPG. We evaluated a commercial timing-aware ATPG tool using the IWLS 2005 benchmark circuits [4]. The relative run-times (ratio of run-time for timing-aware ATPG to timing-aware TDF ATPG) for the benchmarks are shown in Figure 1. For the "netcard" benchmark, which has over 1.5 M logic cells, there is a 209x increase in CPU time when timing-aware ATPG is used. For smaller benchmarks, we observed a 3-65x increase in run times. We therefore conclude that the run-time of timing-aware ATPG does not scale well with circuit

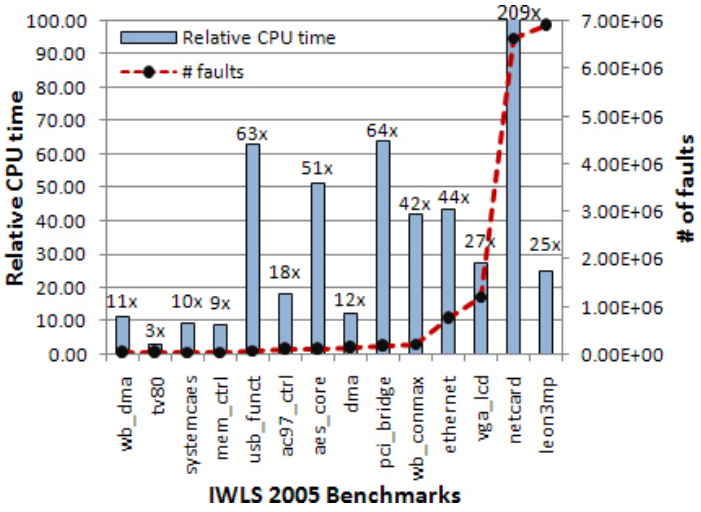


Fig. 1. CPU time of timing-aware ATPG relative to traditional TDF ATPG for the IWLS 2005 benchmark circuits.

size. Unless commercial ATPG tools are improved to deal with large designs, users will be forced to partition their designs into smaller pieces and run timing-aware ATPG on smaller circuits, with adverse impact on fault coverage.

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 [20]. The 2005 ITRS document predicted that the test data volume and test application time for integrated circuits will be as much as thirty times larger in 2010 than they were in 2005 [1]. Therefore, pattern-selection methods are required to reduce the total pattern count while effectively targeting SDDs.

Test-pattern reordering methods, which rank test patterns and place the most effective test patterns at the top of the reordered test sequence, promise reductions in both testing time and test data volume [21]–[23]. If highly effective test patterns are applied first in a reordered test set, defective chips will fail earlier, reducing test application time in an abort-at-first-fail environment. The reordered test set can be simply truncated to fit test-data-volume and meet test-time budgets.

Therefore, there is a pressing need for improved, layout-aware pattern generation methods that can handle delay variation on interconnects. These methods must be able to identify high-quality delay-test patterns with low CPU times from a large repository of candidate test patterns. Some early work has been reported in this direction [3], [24], [25], but none of these techniques handle variation on interconnects. This work is an attempt to fill this void by making the output deviations metric from [3] cognizant of interconnect and layout.

III. Overview of Output Deviations

The concept of gate-delay defect probabilities (DDPs) and signal-transition probabilities were introduced in [3]. These probabilities extend the notion of confidence levels, defined in [23] for a single pattern, to pattern-pairs; however, [3] does not consider layout information, or realistic defects such as resistive shorts, opens, crosstalk, and supply-voltage noise.

In [3], DDPs were assigned to the gates in a netlist. DDPs for a gate are provided in the form of a matrix called the Delay Defect Probability Matrix (DDPM). An example of a DDPM (with entries chosen arbitrarily) 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. Each entry denotes the probability that the corresponding $L \rightarrow H$ (rising) or $H \rightarrow L$ (falling) output transition is delayed beyond a threshold. For instance, the entry in the first row and the third column for the DDPM in Table I shows that there is 50% probability that there will be a delay defect because of the transition on `IN0` when the output makes a $H \rightarrow L$ transition starting with the initial input state of "`10`".

TABLE I
Example DDPM a 2-input OR gate

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

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

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

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

For instance, if we assume a Gaussian delay distribution for all gates (with mean μ) and set the critical delay value to $\mu + X$ ps, each DDP entry can be calculated by replacing T_{CRT} with $\mu + X$ and using a Gaussian pdf. Note that the delay for each input-to-output transition delay may have a different mean (μ) and standard deviation (σ). The delay distribution can be obtained in different ways: (i) Using the delay information provided by the Standard Delay Format (SDF) file; (ii) Using slow, nominal, and fast process corner transistor models; (iii) Simulating process variations. In the third method, employed here and in [3], 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 library gates, depending on the layout, the delay distributions for each individual gate can be updated. T_{CRT} can now be appropriately set to compute the DDPM entries.

Next we analyze the signal transitions that arise on each net of the circuit for each pattern-pair. If we assume that there

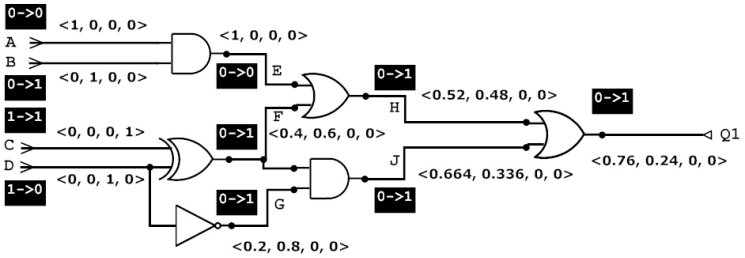


Fig. 2. Propagation of signal-transition probabilities.

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, called signal-transition probability (STP), denoted by $P_{L\rightarrow L}$, $P_{L\rightarrow H}$, $P_{H\rightarrow L}$, and $P_{H\rightarrow H}$, respectively: $< P_{L\rightarrow L}, P_{L\rightarrow H}, P_{H\rightarrow L}, P_{H\rightarrow H} >$. Note that a $L \rightarrow L$ or $H \rightarrow H$ implies that no transition occurs.

The propagation of STPs starts from the test-application points. 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.

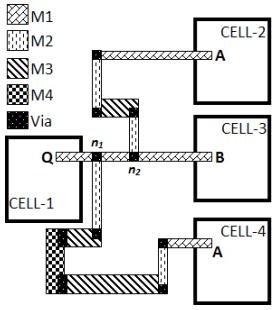
The propagation of STPs is guided by several rules as presented in [3]. These rules allow us to compute deviations for all the nets in the circuit (and finally, the observable primary outputs-either scan latches or combinational outputs) in an efficient, one-pass, feed-forward manner. The output deviation for a pattern-pair at an observable output z is given by the appropriate entry in the STP vector for net z. For example, if a rising transition is expected at z, the output deviation is simply the $P_{L\rightarrow L}$ entry in the STP vector for z. Therefore, the output deviation is a probability measure, whose value lies between `0` and `1`. A key premise of this approach is that output deviations can be used to compare path lengths, without the need for explicit path enumeration. Note that path enumeration (adopted in previous work such as [11], [16], [19], [26]) is a complex and time-consuming procedure. As in the case of path delays, the deviations for nets also increase as the signal propagates through a sensitized path [3].

Note that the pattern-selection results are sensitive to the choice of T_{CRT}. If T_{CRT} is set to a small value, many observation points will have an output deviation close to `1.0`, which will in turn make it difficult to distinguish between paths of different lengths. If T_{CRT} is set to a large value, the delay contributions of many gate instances will unfortunately be neglected during deviation computation. We therefore set T_{CRT} to be the nominal delay of the gate instance that has the shortest instance delay.

Example: Fig. 2 shows signal-transition probabilities and their propagation for a simple circuit without considering interconnect delays and their associated variations. The test stimuli and the expected fault-free transitions on each net are shown in dark boxes. The calculated signal-transition probabilities are shown in angled brackets ($\langle ... \rangle$). The DDPMs of the gates used in this circuit are given in Tables II and I. The entries in Tables II and I are chosen arbitrarily. A depth-first procedure was used to compute signal-transition probabilities for large circuits. If the number of test patterns is k and the number of nets in the circuit is N, the worst-case time-complexity of the algorithm is O(kN). However, since the calculation for each pattern is independent of other patterns, the algorithm can easily be made multi-threaded. In this case, if the number of threads is T, the complexity of the algorithm is reduced to O(kN/T). A pattern-selection method was implemented in [3] to rank test patterns based on deviation and select the best patterns for SDDs.

TABLE II
EXAMPLE DDPM FOR THE BASIC LOGIC GATES USED IN FOR FIG. 2

		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		



(a) Different wires for net ξ.

	Initial Input State	
prob	`0`	`1`
Net ξ	0.2	0.3

(b) Example DDPM for net ξ.

Fig. 3. An example of wire-delay DDPM

IV. INTERCONNECT-AWARE OUTPUT DEVIATIONS FOR MODELING WIRE-DELAY VARIATIONS

In this section, we show how to generalize the output deviations metric to include wire-delay variations.

The wire delay variations and corresponding DDPs can be taken into consideration by assigning DDPMs to all wires. In this case, all wires can have a buffer-like DDPM. Various sources of wire-delay variations can be considered using the wire DDPM.

Consider the example in Fig. 3(a), which shows different metal layers and vias. Assume that M1 and M2 are local wires, and M3 and M4 are global wires. Assume that the net connecting port Q of CELL-1 to the input ports of CELL-2, CELL-3, and CELL-4 is called net ξ. We can model the wire delays in several different ways:

1) **Lumped-delay model for a net:** In this model, we can assign a single DDPM to each net. An example is given in Fig. IV. Assume that a signal with a signal-transition

probability vector (STPV) of $\langle 0.1, 0.9, 0, 0\rangle$ is propagated through net ξ and does a $L \rightarrow H$ transition. Net ξ has a DDP of 0.2 for the corresponding signal transition, so we update the STPV as follows:
$STPV_{NEW} = 0.1 \cdot \langle 1, 0, 0, 0\rangle + 0.9 \cdot \langle 0.2, 0.8, 0, 0\rangle = \langle 0.28, 0.72, 0, 0\rangle$.
After updating the STPV, all the fanout gates use the updated probability values. This lumped model will have smaller impact on run-time compared to more detailed wire delay models. However, there is clearly a potential for incorrect output-deviation calculation. Consider the example in Fig. 3(a). CELL-1/Q to CELL-3/B has a single Metal-1 layer connection and expected to have a small delay variation in absolute terms. On the other hand, the path connecting CELL-1/Q to CELL-4/A includes global wires (M3, M4) and several vias. This path will clearly have a larger absolute delay variation compared to the path from CELL-1/Q to CELL-3/B. Thus, a more detailed delay model is required for higher accuracy.

2) **Lumped delay model for a port-to-port path:** In this model, we assign a single DDPM for each port to port path. An example is given in Table III. Similar to the previous case, assume that a signal with a STPV of $\langle 0.1, 0.9, 0, 0\rangle$ is propagated through net ξ and does a $L \rightarrow H$ transition. STPVs for each path should be updated separately using the DDP values given in Table III:
   - $STPV_{CELL-2/A} = 0.1 \cdot \langle 1, 0, 0, 0\rangle + 0.9 \cdot \langle 0.08, 0.92, 0, 0\rangle = \langle 0.172, 0.828, 0, 0\rangle$.
   - $STPV_{CELL-3/B} = 0.1 \cdot \langle 1, 0, 0, 0\rangle + 0.9 \cdot \langle 0.01, 0.99, 0, 0\rangle = \langle 0.109, 0.891, 0, 0\rangle$.
   - $STPV_{CELL-4/A} = 0.1 \cdot \langle 1, 0, 0, 0\rangle + 0.9 \cdot \langle 0.16, 0.84, 0, 0\rangle = \langle 0.244, 0.756, 0, 0\rangle$.

   As seen, the updated STPVs for the fanout ports have changed significantly for some of the ports.

3) **Distributed delay models:** In order to model the delay effects such as crosstalk on wires more accurately, we can use a distributed delay model and more than a single DDPM for each port to port path. For instance, in Fig. 3(a), assume that there is considerable delay variation between nodes n_1 and n_2 on M1, e.g., due to crosstalk. In this case, the path from CELL-1/Q to CELL-4/A will not be affected by the variation, but the other two paths will be impacted. We split the paths into smaller parts to account for this effect and create the DDPMs shown in Table IV. The new STPV values on fanout ports can be calculated as follows:

TABLE III
EXAMPLE DDPM FOR EACH PORT-TO-PORT PATH ON NET ξ OF FIG. 3(A)

	Initial Input State	
prob	`0`	`1`
`CELL-1/Q` $\rightarrow$ `CELL-2/A`	0.08	0.10
`CELL-1/Q` $\rightarrow$ `CELL-3/B`	0.01	0.01
`CELL-1/Q` $\rightarrow$ `CELL-4/A`	0.16	0.25

TABLE IV
EXAMPLE DDPM FOR DISTRIBUTED DELAY MODEL ON NET ξ OF FIG. 3(A)

	Initial Input State	
(**prob**)	`0`	`1`
`CELL-1/Q` $\rightarrow n_1$	0.01	0.01
$n_1 \rightarrow n_2$	0.12	0.05
$n_1 \rightarrow$ `CELL-4/A`	0.15	0.24
$n_2 \rightarrow$ `CELL-2/A`	0.07	0.09
$n_2 \rightarrow$ `CELL-3/B`	0.01	0.01

- $STPV_{n_1} = 0.1 \cdot \langle 1, 0, 0, 0\rangle + 0.9 \cdot \langle 0.01, 0.99, 0, 0\rangle = \langle 0.109, 0.891, 0, 0\rangle$.
- $STPV_{n_2} = 0.109 \cdot \langle 1, 0, 0, 0\rangle + 0.891 \cdot \langle 0.12, 0.88, 0, 0\rangle = \langle 0.216, 0.784, 0, 0\rangle$.
- $STPV_{CELL-2/A} = 0.216 \cdot \langle 1, 0, 0, 0\rangle + 0.784 \cdot \langle 0.07, 0.93, 0, 0\rangle = \langle 0.271, 0.729, 0, 0\rangle$.
- $STPV_{CELL-3/B} = 0.216 \cdot \langle 1, 0, 0, 0\rangle + 0.784 \cdot \langle 0.01, 0.99, 0, 0\rangle = \langle 0.224, 0.776, 0, 0\rangle$.
- $STPV_{CELL-4/A} = 0.109 \cdot \langle 1, 0, 0, 0\rangle + 0.891 \cdot \langle 0.15, 0.85, 0, 0\rangle = \langle 0.243, 0.757, 0, 0\rangle$.

As expected, the delay variation between nodes n_1 and n_2 did not change the STPV of CELL-4/A appreciably, but it changed the STPV of other ports considerably. Distributed delay models can be made even more detailed by creating a new DDPM for each piece of metal and via. However, the more details added, the longer will be the run-time. As a result, the distributed delay model is useful for SDD detection only if it makes a considerable difference to the STPV values.

To incorporate crosstalk effects in the DDPM entries, we need to consider more details, since the delay induced by crosstalk depends on the direction and timing of signal-transitions on both the aggressor and victim wires. A DDPM lists the DDPs that depend on signal transition of only the corresponding circuit element, e.g., gate or wire. In the case of crosstalk, a wire-DDPM depends to a large extent on a neighboring wire's signal-transition characteristics. This problem can be solved by adding conditional DDP entries in wire-DDPMs for each victim wire. In this paper, we focus on process variations on wires (the two lumped delay models) and leave the study of crosstalk effects (distributed delay model) as future work.

V. PATTERN SELECTION

In this section, we describe how to use output deviations to select high-quality patterns from a base (n–detect or random) transition-fault patterns. We decrease the pattern sorting run-time overhead and increase the quality of selected patterns by removing the small-output-deviation patterns from the pattern list during pattern ordering. For example, during real-time (i.e., dynamic or "on-the-fly") pattern ordering, we drop any pattern with an output deviation of less than 50% of the *instantaneous* highest output deviation (Note that we cannot know the overall maximum output deviation value before going through all patterns.) The pattern-selection procedure can be summarized as follows:

- Determine the number of patterns to be selected. This can be a user input, e.g., S. The parameter S can be set to

the number of 1-detect transition delay-fault patterns, the number of timing-aware patterns, or any value that fits the user's test-time budget.
- Update the *instantaneous* highest output deviation after checking each pattern.
- Skip any pattern with an output deviation of less than 50% of the *instantaneous* highest output deviation. Note that 50% value is selected arbitrarily and the user can adjust this value as needed.
- Until the selected pattern number reaches S, select the largest-deviation patterns for each observation point one by one in the order of scan cells.
- After selection, sort the patterns by the maximum deviation they create at an observation point (pattern reordering).
- Fault simulate the selected patterns for TDFs.
- If the fault coverage is less than the target fault coverage, use top-off ATPG.

We can further improve the quality of the selected patterns by ordering the observation points according to the maximum deviation that they have. Reordering of observation points changes the pattern-selection order. Although the quality of selected patterns may increase in this method, the overall run-time also increases due to an extra ordering step.

VI. Experimental Results

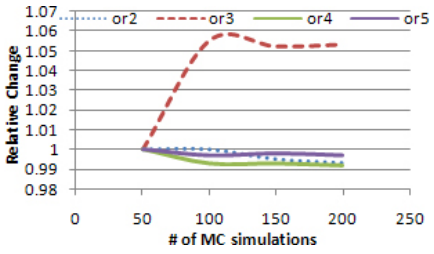
In this section, we present experimental results obtained for the IWLS 2005 benchmark circuits. We do not consider the ISCAS benchmarks because these circuits are small and it is easier for an ATPG tool to excite all long paths with a small number of patterns. We first provide details of the experimental set-up. After that, we present the simulation results.

A. Experimental Set-up

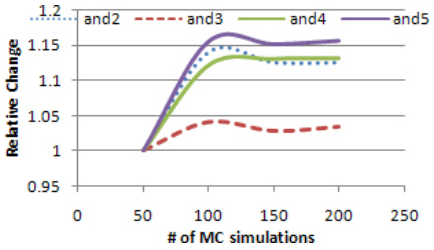
All experiments were performed on a pool of state-of-the-art servers with at least eight processors available at all times, 16GB of memory, and running Linux. The program to compute output deviations was implemented using C++. Perl scripts were used to generate the simulation input files. A commercial tool was used to perform Verilog netlist synthesis and scan insertion for the IWLS benchmark circuits, which are available in Verilog RTL format [4]. Benchmark statistics are shown in Table V. 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. Commercial tools were used for layout synthesis. Detailed wire-length data is extracted from the layout Design Exchange Format (DEF) files using an in-house Perl script. The path delays were calculated using an in-house dynamic path-timing simulator. All simulations were run in parallel on 8 processors.

TABLE V
Benchmark Statistics

	# I/O	# logic cells	# flip-flops	# transition delay-faults
tv80	45	8529	359	40022
ac97_ctrl	104	27713	2289	98702
aes_core	387	20691	554	106054
pci_bridge	367	47189	3677	168844



(a) MC results for OR gates.



(b) MC results for AND gates

Fig. 4. Change in DDPM values as a function of the number of MC simulations, relative to the base case of 50 MC simulations.

B. Generating DDPMs for Gate Instances and Interconnects

DDPM of gate instances were generated by running 200 Monte Carlo (MC) simulations on each gate type, for all possible input signal transitions, and for a range of input and output capacitances. We verified that 200 MC simulations are sufficient for generating DDPMs. Fig. 4 shows the relative change in DDPM entries for OR and AND 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 library cells.

180nm process technology parameters are used for HSpice simulations to match 180nm technology physical-design libraries. MC simulations were run using the following realistic process-variation parameters for a Gaussian distribution:

- Transistor gate length $L : 3\sigma = 10\%$
- Threshold voltage $V_{TH} : 3\sigma = 30\%$
- Gate-oxide thickness $t_{OX} : 3\sigma = 3\%$

For layout generation, we used a physical design library with 6 metal layers. Considering the library parameters, we assumed that M5 and M6 are global routing layers and all other metal layers are used for local routing. We used a delay variation of 30% on local wires and 10% on global wires, which increases with wire length as predicted in [7]. The effect of delay

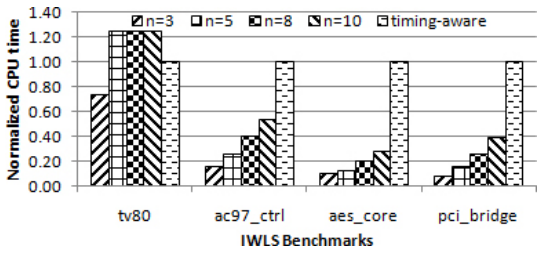


Fig. 5. The comparison of CPU run-time for the proposed method and timing-aware ATPG, using various values of n on IWLS benchmarks.

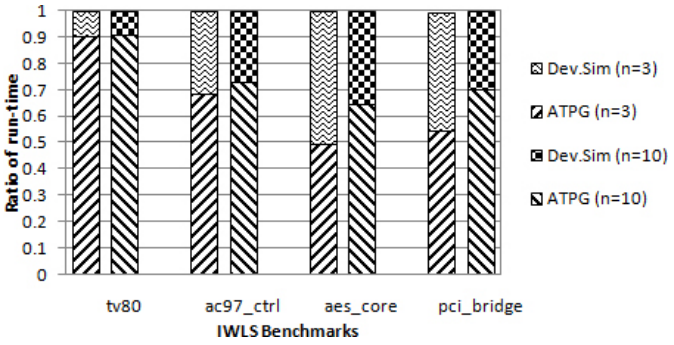


Fig. 6. The distribution of the run-time over different steps of the proposed method, for n=3 and n=10.

variations on vias is ignored and crosstalk effects are left as a future work.

C. Results

In this subsection, we present the pattern-selection results using the number of excited distinct long paths as an evaluation metric.

An initial set of n-detect transition-fault patterns and a set of timing-aware transition-fault patterns were generated. For all values of n, we implemented our method on the benchmarks to calculate output deviations. Then, we applied the pattern selection algorithm described in Section V. For each case, in order to perform a fair comparison, we set the maximum number of patterns S to be selected to the number of patterns in the timing-aware transition-fault pattern set (max S = # timing-aware ATPG patterns). Fig. 9 shows the normalized total CPU run-time for our method (sum of the CPU times needed for n-detect pattern generation, deviation simulation, and pattern selection) and the timing-aware ATPG. All CPU times are reported relative to the run-time of timing-aware ATPG.

We find that even for large values of n, the total CPU time of the proposed method is considerably less than the CPU time of timing-aware ATPG for the three biggest benchmarks. For instance, the CPU time of aes_core benchmark is 503.8 s for timing-aware ATPG , but only 142.13 s for the proposed method with n=10. For tv80, there are a large number of hard-to-control paths and a large number of fanouts. These factors reduces the efficiency of the n-detect pattern generation procedure, hence there is an increase in the overall run-time of the proposed method.

Fig. 6 shows the distribution of the run-time of the proposed method in different steps, i.e., n-detect pattern generation, deviation simulation, and pattern selection and reordering. As seen, n-detect pattern generation takes longer time compared to deviation simulation, constituting the majority of the CPU run-time. Pattern selection and reordering has negligible impact on the overall run-time; the CPU time for this step is close to zero in all case, hence we do not show it in Fig. 6. As n increases, the ATPG step takes a larger portion of the total CPU time.

Fig.7 shows the normalized number of excited distinct long paths for selected patterns and the timing-aware ATPG patterns. We assume that two paths are distinct if there is at least one non-shared gate instance. We also assume that any path with a nominal delay of larger than 70% of the clock period is a long path. We ran an in-house dynamic path-timing simulator on the benchmarks to determine the excited distinct long paths. All numbers are normalized by the number of long paths excited by timing-aware ATPG. Fig. 7 shows that the (unoptimized, academic implementation) output-deviation-based selection method finds patterns that excite a larger number of long paths compared to a commercial (and highly optimized) timing-aware ATPG tool, and with less CPU time. For the benchmark pci_bridge, the patterns selected from a 10-detect timing-unaware test set excite over 2.5x more long paths than timing-aware ATPG, where the proposed method excited 1408 long paths and the timing-aware ATPG excited only 536 long paths. For tv80, this ratio is 1.42 (921 vs. 650).

The proposed method achieves the excitation of more long paths with fewer test patterns in some cases. Table VI shows the number of selected patterns for each benchmark. Note that the number of selected patterns may be less than the pattern count of timing-aware because because we are dynamically dropping low-deviation patterns during deviation computation. Recall from Section V that we only retain the high-deviation patterns for reordering at the next step. When the pattern count is less than the expected value, we can run top-off ATPG to increase TDF fault coverage or to increase long path sensitization. Table VII shows the number of top-off ATPG patterns that need to be added to the set of selected patterns. The required number of top-off patterns is very small compared to the number of selected patterns. Furthermore, for some cases, the proposed method does not need any top-off patterns, resulting in the same TDF coverage as timing-aware ATPG, but using less number of patterns (ac97_ctrl, aes_core, and pci_bridge).

Next we explain why timing-aware ATPG missed some long paths while the same ATPG tool when used without timing information excited more long paths. There are two main reasons for this apparent discrepancy. First, unspecified bits are randomly filled by the ATPG tool for both timing-aware and n-detect timing-unaware cases. Random fill may lead to the excitation of different paths. Second, in the timing-aware case, the ATPG tool attempts to activate a fault through a single long path, whereas n-detect ATPG tries to activate the same fault multiple times, through different paths. Unless it is forced with robustness options, the timing-aware ATPG tool does not re-try activating

TABLE VI
NUMBER OF SELECTED PATTERNS S

	n=3	n=5	n=8	n=10	timing-aware
tv80	2021	2021	2021	2021	2021
ac97_ctrl	934	1284	1284	1284	1284
aes_core	2554	3657	5193	6156	7255
pci_bridge	2700	4163	4163	4163	4163

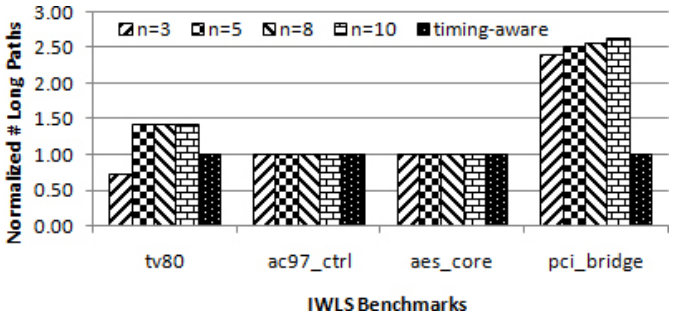


Fig. 7. The normalized number of excited distinct long paths by selected patterns using output-deviation-based method (relative to timing-aware ATPG patterns).

faults through longer paths. This implies that the ATPG tool dropped a fault as soon as it was activated through a long path, possibly not the longest path. If the robustness options are strictly used to force the tool to search for longest paths for all faults, the CPU-time of the timing-aware ATPG increases considerably.

TABLE VII
NUMBER OF TOP-OFF PATTERNS ADDED TO THE SELECTED PATTERNS TO ACHIEVE THE SAME TDF COVERAGE AS TIMING-AWARE

	n=3	n=5	n=8	n=10
tv80	169	263	263	263
ac97_ctrl	0	0	0	1
aes_core	0	0	0	0
pci_bridge	0	0	0	2

In order to further evaluate the relationship between output deviations and sensitized path lengths, we ran correlation analysis on the base n-detect patterns output deviations and their corresponding path delays. We used Matlab to compute Kendall's correlation coefficients [27] for each pattern set. Table VIII shows the average correlation coefficients for the patterns in a 10-detect test set of the IWLS'2005 benchmarks [4]. There is a strong positive correlation between output deviations and path lengths. It can be argued that 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 to account for many physical defects during pattern selection. Dynamic timing simulation can only provide variability-unaware timing information. The correlation between output deviations and path lengths is expected to be less as more physical-defect data is integrated into deviation simulations. In this case, the method of output deviations is expected to reveal unique problematic paths that may be hidden from dynamic timing analysis.

As described in Section II, the impact of local and global

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

	Average	Min	Max
tv80	0.96	0.80	0.99
ac97_ctrl	0.90	0.73	0.96
aes_core	0.97	0.94	0.99
pci_bridge	0.90	0.88	0.99

TABLE IX
DISTRIBUTION OF LOCAL AND GLOBAL WIRES ON LONG PATHS

	Local Routing	Global Routing
tv80	99.93%	0.07%
ac97_ctrl	100.00%	0.00%
aes_core	99.93%	0.07%
pci_bridge	100.00%	0.00%

wires' variations on circuit delay are different. In most designs, the local wires dominate global wires. We have determined the percentage of the local and global wires employed using our extraction tool for the above five benchmarks. Table IX shows the distribution of local and global wires on long paths for each benchmark. It is evident that local routing layers dominate the wiring of long paths. Global wires are significant on long paths only when the circuit size gets larger. For larger circuits, there are more global wires that are much longer than local wires, hence the impact of variations is expected to be higher.

We ran three different defect injection simulations to evaluate the fault-detection performance of the proposed pattern-selection method and the timing-aware ATPG. In the first defect injection simulation, in order to simulate process variation induced delay variations, we added Gaussian random delays on gates. We used a delay distribution with a standard deviation of 6% of the nominal delay of the corresponding gate. We obtained the approximate delay variation on gates from the MC simulations. In the second defect injection simulations, we also added Gaussian random delays on all nets. The additional delay on interconnects had a distribution with a standard deviation of 10% of the nominal delay of the corresponding interconnect. This delay distribution is selected using the results presented in [1], [6]. Note that majority of the added delay defects is expected to be much less than the standard deviation values since the distribution is Gaussian. Thus, no single wire or gate delay variation is likely to create a visible delay defect on observation points. However, it is expected that these small delay variations may add up and cause visible delay defects on some of the paths. We used a clock period with a 3% slack from the critical path.

For each benchmark, we created 1000 sample test cases, each with different random delay faults. We ran an abort-on-first-fail simulation on each sample using the selected patterns from various n-detect pattern sets and timing-aware ATPG patterns. Tables X(a) and X(b) show the number of failed samples for the first and second defect injection simulations using selected patterns and timing-aware ATPG patterns. For each entry, the results for the first case is followed by the results of the second case. As seen, the proposed method caught a considerably larger number of delay faults for most benchmarks in both cases.

TABLE X
NUMBER OF DELAY FAULTS DETECTED USING SELECTED PATTERNS AND TIMING-AWARE ATPG PATTERNS (A) WITH DELAY VARIATIONS ONLY ON GATES (B) WITH DELAY VARIATIONS ON GATES AND INTERCONNECTS (C) WITH RORS.

(a)

	n=3	n=5	n=8	n=10	timing-aware
tv80	814	849	849	849	5
ac97_ctrl	450	450	450	450	450
pci_bridge	1000	1000	1000	1000	1000

(b)

	n=3	n=5	n=8	n=10	timing-aware
tv80	878	922	924	924	8
ac97_ctrl	482	482	482	482	482
aes_core	1000	1000	1000	1000	1000
pci_bridge	1000	1000	1000	1000	1000

(c)

	n=3	n=5	n=8	n=10	timing-aware
tv80	26	48	48	48	36
ac97_ctrl	12	12	12	12	12
aes_core	4	4	4	4	4
pci_bridge	1000	1000	1000	1000	1000

When wire delay variations are not considered, for aes_core benchmark, the additional delay was below the 3% clock period slack, therefore there are no detectable faults. However, injecting wire delay variations caused all the samples of aes_core to fail for all pattern sets. For tv80, the proposed method provides much better fault detection capability. Another key result obtained from the above defect injection simulations is that the proposed method has a much faster ramp-up in detecting faults: For tv80, the proposed method detected 849 (924) delay faults using the first 30 patterns, whereas timing-aware ATPG detected only 5 (8) delay faults after as many as 770 patterns.

We ran a third set of defect injection simulations to evaluate these methods for resistive open and resistive short defects (RORS) on interconnects. In order to simulate the larger delay impact caused by RORS, we injected a single delay defect on a randomly selected net. The injected defects have a normal delay distribution with a standard deviation of 15% of the clock period. Note that the injected faults are not guaranteed to cause a delay fault since they can hit a short path. We created 1000 distinct test cases for each benchmark. Table X(c) shows the number of failed samples using selected patterns and timing-aware ATPG patterns. As seen, the proposed method detected the same or more number of delay faults. Furthermore, we observed a faster ramp-up in delay fault detection for the proposed method. Fig. 8 shows how fast the delay faults are detected for tv80. As seen, the proposed method detects 24 delay faults within 1000 patterns, whereas timing-aware ATPG detects 18 delay faults after 2000 patterns.

Next, we selected patterns using the wire-delay-oblivious deviation model as presented in [3]. The objective here was to determine the extent of inaccuracy that results if interconnect delay variations are ignored. Table XI shows the increase in the number of long paths excited for the lumped wire delay model

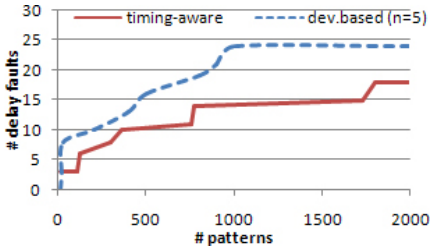


Fig. 8. The number of detected delay faults for the proposed method and timing-aware ATPG for benchmark tv80.

TABLE XI
THE PERCENTAGE INCREASE IN THE NUMBER OF LONG PATHS ACTIVATED BY THE LUMPED-DELAY MODEL RELATIVE TO THE WIRE-DELAY-OBLIVIOUS MODEL.

	n=3	n=5	n=8	n=10
tv80	37.57%	38.70%	34.65%	27.74%
ac97_ctrl	0.00%	0.00%	0.00%	0.00%
aes_core	0.00%	0.00%	0.00%	0.00%
pci_bridge	15.09%	19.15%	20.26%	16.75%

(relative to the the wire-delay-oblivious model). We find that the addition of the wire delays has a considerable impact on the activation of long paths. For tv80, lumped-delay model excited over 30% more long paths compared to the wire-delay-oblivious model. The difference is around 20% for pci_bridge.

We next ran the deviation computation and pattern selection steps using the pin-to-pin lumped delay model. Fig. 9 shows the total CPU run-time for the proposed method using the pin-to-pin delay model (sum of the CPU times needed for n-detect pattern generation, deviation simulation, and pattern selection), relative to the time needed for timing-aware ATPG. As seen, the overall run-time is still considerably less than the CPU time of timing-aware ATPG for the four biggest benchmarks. However, the need for additional computation increases the deviation calculation time. For aes_core and ac97_ctrl, the difference was negligible. However, more than 40% increase in run-time is observed for pci_bridge. This is mainly caused by the large number of fanouts on the internal nets of pci_bridge. Other benchmarks showed a 15-20% increase in run-time.

Next, we ran timing simulations on the selected patterns to determine the number of long paths excited using the pin-to-pin wire delay model. The results were very close to what we obtained for the lumped wire delay model, with a maximum

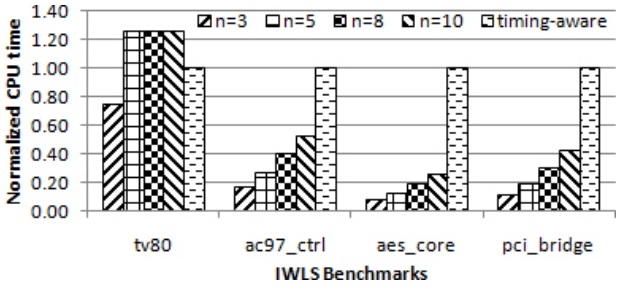


Fig. 9. The comparison of CPU run-time for the proposed method and timing-aware ATPG, using various values of n on IWLS benchmarks, when the pin-to-pin lumped wire delay model is used.

difference of 3% between the two sets of results. This shows that, for the IWLS benchmarks, the additional computational complexity of the pin-to-pin wire delay model is not accompanied by any significant benefits in long-path activation. This might be because the fanout wires on nets are balanced in the layout, resulting in very similar delays on all the fanouts of a net.

Finally, in order to evaluate impact of interconnect delay variations on new process technologies, we scaled the interconnect delays as foreseen by ITRS [1] to model a 45nm process technology. Table XII shows the percentage of patterns that are selected for the 45nm technology, but not selected for the 180nm technology. The difference in these sets of patterns is quite small in all cases. However, when we examine the ordering of patterns, we find that the ordering of the patterns (based on deviations) are very different. A quantification of this difference will be presented in the final version of this paper. These results imply that although the transition to 45nm technology did not change the long paths for a benchmark, it changed their relative importance considerably, hence criticality these paths.

TABLE XII
THE PERCENTAGE OF DIFFERENT PATTERNS SELECTED AT 180NM AND 45NM PROCESS TECHNOLOGIES.

	n=3	n=5	n=8	n=10
tv80	2.23%	4.16%	4.16%	4.16%
ac97_ctrl	0.00%	1.80%	2.96%	3.19%
aes_core	0.00%	0.00%	0.00%	0.00%
pci_bridge	0.00%	0.12%	2.57%	4.44%

VII. CONCLUSIONS

We have presented a layout-aware pattern-selection technique for screening small-delay defects (SDDs) in nanometer integrated circuits. We have defined the concept of output deviations for interconnect-delay variations and 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. Experimental results for the IWLS 2005 benchmark circuits show that the proposed method selects an effective set of patterns for SDD detection from an n-detect TDF pattern set generated using a timing-unaware ATPG tool, and it excites a larger number of long paths compared to a commercial and highly-optimized timing-aware ATPG tool. The CPU time for the proposed method is also much less than that for timing-aware ATPG. The proposed method is computationally tractable, yet more accurate than a previous deviations-based method [3] that ignores chip layout and delay variations on interconnects.

ACKNOWLEDGEMENTS

We thank Jeremy Lee of University of Connecticut for providing layout synthesis guidelines. We thank Jeff Rearick, Jeff Fitzgerald, and their colleagues at AMD for valuable discussions and for providing us access to computing resources.

REFERENCES

[1] ITRS 2005, "http://www.itrs.net/links/2005itrs/home2005.htm."
[2] E. Park, M. Mercer, and T. Williams, "Statistical delay fault coverage and defect level for delay faults," in *Proc. of IEEE Int. Test Conference*, 1988, pp. 492–499.
[3] M. Yilmaz, K. Chakrabarty, and M. Tehranipoor, "Test-pattern grading and pattern selection for small-delay defects," in *Proc. of IEEE VLSI Test Symp.*, 2008.
[4] IWLS 2005 Benchmarks, "http://iwls.org/iwls2005/benchmarks.html."
[5] J. Davis et al., "Interconnect limits on gigascale integration (GSI) in the 21st century," *Proc. of IEEE*, vol. 89.
[6] G. Lopez et al., "The impact of size effects and copper interconnect process variations on the maximum critical path delay of single and multi-core microprocessors," in *Proc. of IEEE Int. Interconnect Technology Conference*, Jun 2007, pp. 40 – 42.
[7] H. Kitada et al., "The influence of the size effect of copper interconnects on rc delay variability beyond 45nm technology," in *Proc. of IEEE Int. Interconnect Technology Conference*, Jun 2007, pp. 10 – 12.
[8] L. Scheffer, "An overview of on-chip interconnect variation," in *ACM Int. Workshop on System-Level Interconnect Prediction*, 2006, pp. 27 – 28.
[9] F. Caignet et al., "The challenge of signal integrity in deep-submicrometer CMOS technology," *Proc. of IEEE*, vol. 89, no. 4, pp. 556 – 573, Apr 2001.
[10] J. Waicukauski et al., "Transition fault simulation," *IEEE Design and Test of Computers*, pp. 32–38, 1987.
[11] N. Ahmed, M. Tehranipoor, and V. Jayaram, "Timing-based delay test for screening small delay defects," in *Proc. of IEEE Design Automation Conf.*, 2006, pp. 320–325.
[12] R. Putman and R. Gawde, "Enhanced timing-based transition delay testing for small delay defects," in *Proc. of IEEE VLSI Test Symp.*, 2006, pp. 336–342.
[13] P. Gupta and M. Hsiao, "ALAPTF: A new transition fault model and the ATPG algorithm," in *Proc. of IEEE Int. Test Conference*, 2004, pp. 1053–1060.
[14] W. Qiu et al., "K longest paths per gate (KLPG) test generation for scan-based sequential circuits," in *Proc. of IEEE Int. Test Conference*, 2004, pp. 223–231.
[15] S. Gurumurthy et al., "Automatic generation of instructions to robustly test delay defects in processors," in *Proc. of IEEE European Test Symp.*, 2007, pp. 173–178.
[16] Y. Sato et al., "Invisible delay quality - SDQM model lights up what could not be seen," in *Proc. of IEEE Int. Test Conference*, 2005, p. 9.
[17] Cadence Inc., "Encounter test - test generation and simulation reference," product Version 3.0, 2005.
[18] Mentor Graphics, "Understanding how to run timing-aware ATPG," application Note, 2006.
[19] R. Kapur, J. Zejda, and T. Williams, "Fundamentals of timing information for test: How simple can we get?" in *Proc. of IEEE Int. Test Conference*, 2007.
[20] B. Keller et al., "An economic analysis and ROI model for nanometer test," in *Proc. of IEEE Int. Test Conference*, 2004, pp. 518–524.
[21] X. Lin, J. Rajski, I. Pomeranz, and S. Reddy, "On static test compaction and test pattern ordering for scan designs," in *Proc. of IEEE Int. Test Conference*, 2001, pp. 1088–1097.
[22] Y. Tian, M. Mercer, W. Shi, and M. Grimaila, "An optimal test pattern selection method to improve the defect coverage," in *Proc. of IEEE Int. Test Conference*, 2005.
[23] Z. Wang and K. Chakrabarty, "Test-quality/cost optimization using output-deviation-based reordering of test patterns," *IEEE Tran. on CAD of Int. Cir. and Systems*, vol. 27, pp. 352–365, Feb 2008.
[24] W.-Y. Chen, S. Gupta, and M. Breuer, "Test generation for crosstalk-induced delay in integrated circuits," in *Proc. of IEEE Int. Test Conference*, Sep 1999, pp. 191 – 200.
[25] J. Lee and M. Tehranipoor, "Delay fault testing in presence of maximum crosstalk," in *IEEE North Atlantic Test Workshop*, 2007.
[26] X. Lin et al., "Timing-aware ATPG for high quality at-speed testing of small delay defects," in *Proc. of IEEE Asian Test Symp.*, 2006, pp. 139–146.
[27] B. J. Chalmers, *Understanding Statistics*. CRC Press, 1987.