Crosstalk-Aware Signal Probability-Based Dynamic Statistical Timing Analysis

Yao Chen*, Andrew B. Kahng†, Bao Liu* and Wenjun Wang*
*University of Texas at San Antonio, 1 UTSA Circle, San Antonio, TX 78249
†University of California, San Diego, CA 92093

***Abstract*— Crosstalk is an increasingly significant effect for VLSI timing performance. Traditional STA or SSTA techniques provide pessimistic crosstalk analysis based on timing window envelopes. In this paper, we present input-aware signal probability-based statistical timing analysis (SPSTA) taking crosstalk-induced delay variations into account. SPSTA achieves reduced pessimism and improved accuracy by signal propagation path/network-based timing analysis and leveraging some existing automatic test pattern generation (ATPG) techniques. Our experiments based on the 45nm Nangate open cell library show that compared with 1.35 million Monte Carlo simulation runs, while PrimeTime-SI over-estimates by 14.64%, 16.39% and 19.69%, SSTA achieves an average of 2.85%, 2.89% and 3.59% inaccuracy, and SPSTA achieves an average of 1.85%, 2.54% and 1.53% inaccuracy in crosstalk-oblivious, crosstalk-aware non-iterative and crosstalk-aware iterative analysis, respectively.**

I. Introduction

Timing analysis becomes increasingly critical in nanoscale VLSI design. As VLSI technology scaling leads to increasingly significant process and system runtime parametric variations, VLSI performance is subject to increasingly significant variability. To capture such performance variability, statistical static timing analysis (SSTA) techniques have been developed based on traditional static timing analysis (STA) techniques. STA and SSTA are input-oblivious and perform pessimistic best/worst case timing analysis, which leads to further pessimism and inaccuracy in the presence of parasitics and signal integrity effects which are increasingly significant as technology scales.

One of the most significant signal integrity effects is the crosstalk effect. VLSI technology scaling has led to increasingly significant coupling capacitance between physically adjacent interconnects. When two signals in a pair of cross-coupled interconnects take transitions at the same time, the crosstalk effect induces delay variation. Input-oblivious STA or SSTA techniques cannot accurately perform timing analysis and determine if such crosstalk-induced delay variations exist.

Our contributions in this paper are as follows.

1) We pioneer new dynamic crosstalk-aware statistical timing analysis techniques for improved accuracy and reduced pessimism in the context of cross-coupled interconnects. We apply the signal-probability-based statistical timing analysis (SPSTA) technique and take the crosstalk effect into account. By taking logic inputs into account, SPSTA performs signal propagation path/network-based statistical timing analysis, which leads to reduced pessimism and increased accuracy in crosstalk-aware timing analysis.
2) We leverage existing automatic test pattern generation (ATPG) techniques to remove false signal propagation paths/networks and false crosstalk aggressors.
3) We further develop a crosstalk-aware timing analysis flow wherein we annotate signal propagation delays based on two SDF files which are generated by Synopsys PrimeTime-SI with and without the crosstalk effect taken into account, respectively.
4) Our experiments based on the 45nm Nangate open cell library show that compared with 1.35 million Monte Carlo simulation runs, while PrimeTime-SI over-estimates by 14.64%, 16.39% and 19.69%, SSTA achieves an average of 2.85%, 2.89% and 3.59% inaccuracy, and SPSTA achieves an average of 1.85%, 2.54% and 1.53% inaccuracy in crosstalk-oblivious, crosstalk-aware non-iterative and crosstalk-aware iterative analysis, respectively.

The rest of this paper is organized as follows. We review the existing techniques in Section II. We present crosstalk-aware dynamic statistical timing analysis in Section III. We present a crosstalk-aware timing analysis methodology, our crosstalk-aware timing analysis implementation details, experimental setups and results in Section IV, then conclude in Section V.

II. Background

A. Crosstalk-aware Timing Analysis

VLSI technology scaling has led to increasingly significant interconnect crosstalk effects since the $130nm$ technology node. Between two cross-coupled interconnects, a signal transition at an aggressor interconnect induces noise in a quiet victim interconnect, a slow-down for a signal transition in the opposite direction, or a speed-up for a signal transition in the same direction as the victim interconnect.

Crosstalk-aware timing analysis is a chicken-and-egg problem: one needs to know the signal arrival times to determine if there is crosstalk-induced delay variation; however, without knowing the crosstalk-induced delay variation, one cannot know the signal arrival times. Traditional STA and SSTA techniques handle this chicken-and-egg problem by iterative pessimistic analysis. STA and SSTA calculate a *window* (i.e., of the time at which transition occurs) for each signal between

the best and worst case signal arrival times, and assume that the crosstalk effect takes place between any two signals that have overlapping timing windows. Initially, all signals have an infinite timing window and the crosstalk effect is assumed to take place between any two physically-adjacent interconnects. Subsequently, timing windows are updated based on known signal arrival times, the crosstalk effect is excluded between every pair of signals with non-overlapping timing windows, and timing analysis is iterated for progressively reduced pessimism [4]–[6], [9].

B. Statistical Timing Analysis

To capture the increasingly significant parametric variations and resultant performance variability in nanoscale VLSI systems, statistical timing analysis techniques have been developed. Statistical timing analysis calculates the arrival time distribution for each signal, and a timing yield or probability for a design to meet all timing requirements. Most of the existing statistical timing analysis techniques are *statistical static timing analysis* (SSTA) techniques which are developed on the basis of traditional static timing analysis (STA) techniques. For efficiency, STA techniques are input-oblivious and so are SSTA techniques.

There are two basic categories of SSTA techniques. *Block-based SSTA* [1], [3], [16] computes a signal arrival time distribution function for each node, e.g., in a breadth-first traversal of the netlist. *Path-based SSTA* [13], [14] computes a signal arrival time distribution function for each node in each (near-critical) path, based on the observation that signal arrival time distributions differ as the signal comes from different paths, e.g., due to path-sharing and resultant signal correlations.

As noted above, one of the most significant sources of VLSI performance variation is interconnect crosstalk effect, i.e., gate and interconnect delay variations due to simultaneous signal arrivals at cross-coupled interconnects. Significant signal coupling effects (such as interconnect crosstalk, power supply degradation, and on-chip temperature variation) imply that the performance of a VLSI system strongly depends on the statistics of its primary inputs and the states of its sequential elements, which however are not taken into account by traditional STA and SSTA techniques.

Dynamic (input-aware) statistical timing analysis techniques are developed by leveraging some of the existing VLSI power estimation techniques. VLSI power estimation and statistical timing analysis study the same stochastic signal switching activities. While VLSI power estimation focuses on average signal switching activities, statistical timing analysis studies the extremes (best and worst) of the stochastic process. Power estimation techniques are in three categories: (1) static (input-oblivious), (2) statistical (input-aware), and (3) Monte Carlo simulation/testing, given in order of increasing accuracy and decreasing efficiency. Similarly, VLSI statistical timing analysis techniques can be developed in three corresponding categories. Some of the existing techniques in power estimation [11] can be leveraged in statistical timing analysis.

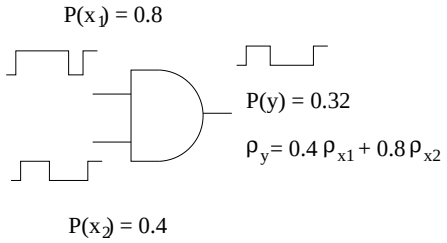

Fig. 1. Signal probability and signal toggling rate computation for an AND gate.

For example, leveraging one of the common probabilistic power estimation techniques gives Signal Probability (logic one occurrence probability) based Statistical Timing Analysis (SPSTA) [10], which improves accuracy compared with statistical static timing analysis, and improves efficiency compared with Monte Carlo (SPICE) simulation.

C. Preliminaries

SSTA calculates signal arrival time distributions by two basic operations: taking summation (SUM) and finding minimum/maximum (MIN/MAX). In best/worst case analysis, the minimum/maximum of the signal arrival times at the inputs of a gate is found. The gate output signal arrival time is subsequently calculated by summing up the minimum/maximum input signal arrival time and the gate delay. Any delays on the driven interconnect must then be added to obtain the signal arrival time at the next stage gate input. Typical SSTA techniques represent signal arrival times and gate/interconnect delays as functions of Gaussian distributions. Taking the summation of Gaussian distributions gives a Gaussian distribution, while finding the minimum/maximum of Gaussian distributions gives a distribution which can be approximated as a Gaussian distribution.

In VLSI toggling rate analysis for power estimation [11], each signal has a signal (logic one occurrence) probability, and a toggling rate or average number of togglings in a time unit such as a clock cycle. Signal probabilities can be calculated based on logic, e.g., the signal probability for the output of an AND gate is given by the product of the signal probabilities at the inputs of the AND gate (Fig. 1). Signal toggling rates can be calculated based on logic, e.g., the signal toggling rate for the output of an AND gate is given by the weighted sum of input signal toggling rates, where the weight for each input signal toggling rate is the probability for the other inputs to take logic one which is the non-controlling logic value for an AND gate. In general, the signal probability $P(y)$ for the output y of a Boolean logic function $y = f(x_1, ..., x_n)$ is given by

$$P(y) = P(x_1)P(f_{x_1}) + P(\bar{x}_1)P(f_{\bar{x}_1}) \quad (1)$$

where $f_{x_1} = f(1, x_2, ..., x_n)$ and $f_{\bar{x}_1} = f(0, x_2, ..., x_n)$ are cofactors of f with respect to x_1. The signal toggling rate $\rho(y)$ for the output y of a Boolean logic function $y = f(x_1, ..., x_n)$

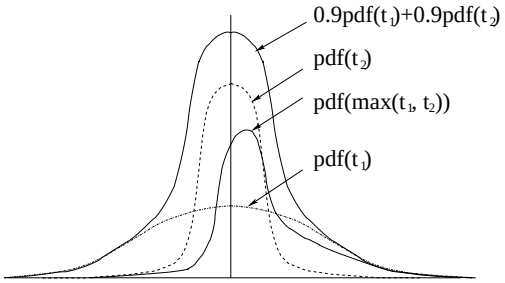

Fig. 2. The results of the MAX and the WEIGHTED SUM operations for an AND gate with two inputs both of 0.9 signal probability, and of signal arrival times t_1 and t_2 in symmetric distributions with the same mean but different deviations.

is given by

$$\rho(y) = \sum_i P(\frac{\partial y}{\partial x_i})\rho(x_i) \tag{2}$$

where the Boolean difference function captures a signal propagation path from an input x_i to the output y and is given by

$$\frac{\partial y}{\partial x_i} = y\mid_{x_i=1} \oplus y\mid_{x_i=0} \tag{3}$$

SPSTA extends the signal toggling rate in power estimation to a signal *transition occurrence probability* (*top*) function in the time domain. The area under the curve of a top function is the signal toggling rate. The extremes of a top function give the minimum and the maximum signal arrival times, respectively. A top function is computed similar to a toggling rate:

$$\phi(y) = \sum_i P(\frac{\partial y}{\partial x_i})\phi(x_i) \tag{4}$$

This is the WEIGHTED SUM operation in SPSTA, which replaces the MIN/MAX operation in SSTA in combining the input signal arrival times at each logic gate (Fig. 2). The SUM operation remains the same in SPSTA as in SSTA in accumulating the signal propagation delays along a path.

The top functions in SPSTA provide signal arrival time distributions for each signal propagation path or network (in the presence of a reconvergent fanout, a signal may propagate in a network other than a path). We develop crosstalk-aware SPSTA based on this.

III. Crosstalk-Aware SPSTA

We extend the SPSTA technique by taking crosstalk-induced delay variations into account. Compared with crosstalk-aware STA or SSTA techniques which are input-oblivious, in SPSTA we perform signal propagation path/network-based timing analysis. A signal or a group of signals may propagate simultaneously in multiple overlapping paths which form a signal propagation network. In the presence of performance variability, the worst case performance is given by a MAX operation in SSTA or SPSTA for a group of simultaneously arriving signals at the inputs of a gate (while separately, the multiple-input switching effect also increases the delay of a gate [2]). For example, Fig. 3 gives a signal propagation network. Note that not the entire fanin cone of a timing-critical logic output is a timing-critical network. For example, for a 2-input AND gate, the latest rising signal arrival at the gate output is given by two simultaneous rising signals at the gate inputs, while the latest falling signal arrival at the gate output is given by a single falling signal at a gate input with the other gate input having a non-controlling logic value which is logic 1 for an AND gate. Multiple simultaneous falling signals at the gate inputs lead to an earlier falling signal arrival at the gate output.

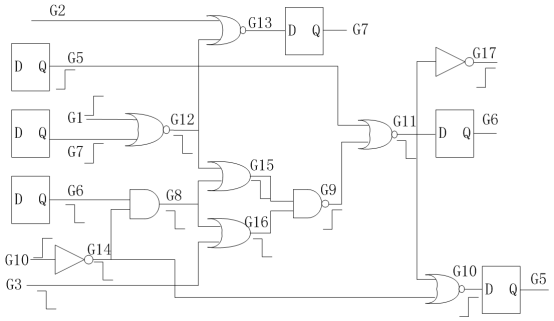

Fig. 3. A signal propagation network in an ISCAS'89 benchmark circuit.

We calculate signal arrival time distributions for each signal propagation path/network. We construct a timing window based on the $\mu + 3\sigma$ and $\mu - 3\sigma$ points for a signal arrival time distribution, and check if the timing window of a victim signal overlaps with the timing window of an aggressor signal. Compared with crosstalk-aware STA or SSTA techniques, our crosstalk-aware SPSTA calculates smaller timing windows in path/network-based analysis, which leads to an invalidation of more crosstalk aggressors, resulting in reduced pessimism and improved accuracy.

Algorithm 1: Iterative Crosstalk-Aware SPSTA

Input: netlist, cell library,
physical information including interconnect couplings,
input signal arrival time distributions,
other process/environment variations
Output: Timing report

1) Perform crosstalk-validating SPSTA assuming crosstalk effect between any two capacitively coupled interconnects
2) Construct timing windows for each signal in each path/network at each node in the netlist
3) Perform crosstalk-validating SPSTA again excluding crosstalk effects between signals of non-overlapping timing windows
4) If there exists any delay update, go to (2)
5) Else, return timing report

Similar to the existing crosstalk-aware STA or SSTA techniques, in crosstalk-aware SPSTA, we start with infinite timing windows for each signal, such that crosstalk exists between any two signals at two physically neighboring interconnects. We proceed by calculating signal arrival time distributions, updating the timing windows, and updating the criterion for the existence of crosstalk effect between any two physically adjacent interconnects. The result is progressively reduced

timing windows and reduced pessimism similar to crosstalk-aware STA or SSTA. Algorithm 1 summarizes this iterative crosstalk-aware SPSTA algorithm.

We further improve the accuracy by leveraging some of the existing ATPG techniques. Specifically, we validate each signal propagation path/network by checking the existence of an input signal transition pattern which propagates a signal transition through the path/network. Propagating a signal transition through a path/network requires that all the side inputs of the gates in the path/network take a non-controlling logic value. If any two logic assignments conflict, no signal transition can propagate through the signal propagation path/network; thus the signal propagation path/network is false.

For a crosstalk-induced delay variation to take place, the crosstalk aggressor must transition at the same time that the victim signal transitions. We further assign crosstalk aggressors to take a simultaneous signal transition, and validate such logic assignment by finding an input signal transition pattern which allows (1) the signal propagation path/network side inputs to take non-controlling logic values, and (2) the crosstalk aggressors to take simultaneous signal transitions. We achieve such validation by applying a modified PODEM algorithm [8].

In the presence of multiple crosstalk aggressors, we rank the crosstalk aggressors by their respective delay variations, and prioritize activation of the most significant crosstalk aggressors. Observing the NP-hardness of the problem, we develop a greedy algorithm for efficiency. Algorithm 2 summarizes our crosstalk-validating SPSTA algorithm.

Algorithm 2: Crosstalk-Validating SPSTA

Input: netlist, cell library,
physical information including interconnect couplings,
input signal arrival time distributions,
other process/environment variations
Output: Timing analysis including validated crosstalk effects

1) $S = \emptyset, A = \emptyset$
2) Perform crosstalk-oblivious SPSTA and find the top-most timing-critical path p_i
3) Assign respective non-controlling logic values to critical path side inputs
4) Apply PODEM for a consistent input signal assignment S
5) If S cannot be found, flag p_i as a false path and go to (1)
6) For each crosstalk aggressor a_j in descending order of its delay variation effect
7) If aggressor and victim timing window do not overlap, continue
8) Assign a signal transition to a_j
9) Apply PODEM for a consistent input signal assignment S'
10) If S' cannot be found, break
11) $S = S \cup S'$, $A = A \cup \{a_j\}$
12) Perform crosstalk-aware SPSTA based on aggressors A

IV. Experiments

A. Crosstalk-Aware Timing Analysis Methodology

For a given Verilog/VHDL design, we synthesis a gate-level netlist for a given clock cycle time based on the open source $45nm$ Nangate cell library [15] by running Synopsys Design Compiler F-2011.09. Subsequently, we achieve physical design by running Cadence SOCEncounter v10.10-s002_1, which outputs the parasitics in a SPEF file. We run PrimeTime-SI F-2011.06-SP2 based on the SPEF file, which outputs two SDF files including all the gate and interconnect delays with and without the crosstalk effect taken into account, respectively. We have implemented our crosstalk-aware SPSTA algorithm in C++. Next, we run our crosstalk-aware SPSTA program, which reads the gate-level netlist, the SPEF file for the coupling capacitors, and the two SDF files for the gate and interconnect delays with and without the crosstalk effect, respectively. As a result, we do not need to implement crosstalk-aware delay calculation in our C++ program based on this flow.

B. Crosstalk-Aware SPSTA, SSTA and Monte Carlo Simulation Implementation

The crosstalk-aware SPSTA program first assumes that the crosstalk effect exists between any two interconnects that have capacitive coupling, and performs timing analysis based on the gate and interconnect delays from the SDF file with the crosstalk effect taken into account. In subsequent iterations, the crosstalk-aware SPSTA program checks if the timing windows overlap for each pair of crosstalk aggressor and victim signals. If the timing windows do not overlap, or the crosstalk effect does not exist, the gate and interconnect delays are updated based on the other SDF file without the crosstalk effect taken into account.

For comparison, we also implement a crosstalk-aware SSTA program and a crosstalk-aware Monte Carlo simulator. The crosstalk-aware SSTA program goes through iterations in the same principle: first assume all timing windows overlap, then, based on the updated timing windows, progressively exclude some of the crosstalk-induced delay variations.

In a crosstalk-aware Monte Carlo simulation, for each set of randomly assigned input signal patterns, and randomly assigned gate and interconnect delays, we propagate logic values and calculate signal arrival times for each node in the gate-level netlist. Crosstalk-induced delay variation exists if an aggressor signal and a victim signal both take transitions. Crosstalk-induced delay variation is also in a Gaussian distribution.

Our input signal assignment scheme and parametric variation model are as follows. We randomly assign the four logic values (0, 1, rising and falling) to the logic inputs including primary inputs and flip-flop outputs, such that each logic input has 25% probability to be logic 0, 1, rising and falling, respectively. Any two logic inputs have zero correlation.

We assume that each gate or interconnect delay is in a Gaussian distribution. Based on the two SDF files, each gate or interconnect has 8 delays for rising or falling, minimum or maximum, and with or without the crosstalk effect, respectively. For each rising/falling gate/interconnect delay with or without the crosstalk effect, we take the minimum and maximum delays as the $\mu-3\sigma$ and $\mu+3\sigma$ points in a Gaussian distribution, and calculate the mean and standard deviation. The gate and interconnect delays have no correlation.

In each run of Monte Carlo simulation, we pick up a sampling point in each Gaussian distribution for the input signal arrival times, the gate delays, the interconnect delays, and the crosstalk-induced delay variations. After 150×9000 Monte Carlo simulation runs for each test case, we calculate the mean and standard deviation of the signal arrival times, and find the maximum rising and falling signal arrival times, respectively.

C. Test Cases

Our experiments are based on four modules in the AES (Advanced Encryption Standard) algorithm [12] and the integer unit of an open source SPARC V8 processor [7]. For each test case we synthesize a gate-level netlist with a 3.0ns clock cycle time based on the 45nm open source Nangate cell library [15]. Table I gives the numbers of gates, primary inputs, primary outputs, and coupled interconnect pairs (CIP) for these test cases, respectively.

TABLE I
TEST CASE CHARACTERISTICS.

	AES S-Box	AES Key-Exp	LEON IFU	LEON IDU
#Gate	572	2367	227	653
#CIP	297	2149	85	344
#PI	49	129	128	2082
#PO	37	128	144	1431

D. Experimental Results

One of the major differences between SPSTA and SSTA is that SSTA is block-based and computes a timing window for each signal at a node in the netlist, while SPSTA is signal propagation path/network-based and computes a timing window for each signal in a path/network at a node in the netlist. As a result, SPSTA timing windows are narrower than SSTA timing windows, and SPSTA tends to have more non-overlapping timing window pairs. On the other hand, SPSTA has more timing windows because at each node for each signal propagation path/network there is a timing window.

Table II gives the number of false crosstalk aggressors identified by SPSTA and SSTA for each test case, respectively. A crosstalk aggressor is false if there does not exist a timing window which overlaps with the victim net signal timing window, or the aggressor signal cannot be generated by an input signal pattern which activates the timing-critical path simultaneously. We observe that SPSTA identifies more false crosstalk aggressors than SSTA, which is consistent with our expectation. From this result, we further expect the SPSTA-calculated maximum critical path delay be smaller than the SSTA-calculated maximum critical path delay, for any given test case.

TABLE II
NUMBER OF IDENTIFIED FALSE CROSSTALK AGGRESSORS.

	AES S-Box	AES Key-Exp	LEON IFU	LEON IDU
SPSTA	293	1707	67	154
SSTA	95	959	55	69

We run SPSTA, SSTA, Monte Carlo simulation and PrimeTime-SI to find the maximum signal arrival time at a logic output for each test case. For SPSTA and SSTA, the maximum signal arrival time is given by $\mu + 3\sigma$ or $\mu + 6\sigma$ of the signal arrival time distribution at each logic output node. Table III gives the μ, σ, $\mu + 3\sigma$, $\mu + 6\sigma$ of the maximum rising/falling signal arrival time at a logic output produced by SPSTA, SSTA, Monte Carlo simulation and PrimeTime-SI in (i) crosstalk-oblivious analysis (assuming all timing windows do not overlap), (ii) crosstalk-aware non-iterative analysis (assuming all timing windows overlap), and (iii) crosstalk-aware iterative analysis (until the iteration converges, which takes place in three iterations for all test cases in our experiments).

Our observations are as follows.

Observation 1: PrimeTime-SI gives the most pessimistic logic output signal arrival time estimates.

PrimeTime-SI performs STA. The resultant logic output signal arrival times are given by the sum of worst-case gate delays or the $\mu + 3\sigma$ delays of each gate in the timing-critical path. In comparison, SSTA computes a Gaussian distribution for the critical path delay, whose variance is given by the sum of the variances of each gate delay in the critical path divided by the number of gates in the critical path, if we ignore the MAX operations. That is to say,

$$\sigma^2(D(P)) = \frac{\sum_{g_i \in P} \sigma^2(D(g_i))}{N} \tag{5}$$

where P is a critical path, $D(P)$ is the critical path delay, g_i is a critical gate in the path, $D(g_i)$ is the delay of the critical gate g_i, and N is the number of critical gates in the path P. As a result, the SSTA resultant path delay has a much smaller variance, leading to a smaller $\mu + 3\sigma$ path delay compared with PrimeTime-SI results.

Observation 2: Monte Carlo simulation results in maximum logic output signal arrival times that are smaller than the maximum logic output signal arrival times given by PrimeTime-SI, and also smaller than the $\mu + 6\sigma$ maximum logic output signal arrival times given by SSTA or SPSTA.

While PrimeTime-SI gives an upper bound, Monte Carlo simulation gives a lower bound for the maximum possible signal arrival time at a logic output. As we increase the number of runs in Monte Carlo simulation, the resultant maximum logic output signal arrival times slowly increase. Or, if we take the Monte Carlo produced maximum logic output signal arrival times as $\mu + k\sigma$, k increases as the number of Monte Carlo simulation runs increases. A critical path having a tiny activation probability may need a large number of Monte Carlo simulation runs to be activated.

Observation 3: SSTA achieves an average of 2.85%, 2.89% and 3.59% inaccuracy for maximum logic output signal arrival time based on the $\mu + 3\sigma$ metric in crosstalk-oblivious, crosstalk-aware non-iterative and crosstalk-aware iterative analysis compared with 1.35 million Monte Carlo simulation runs, respectively.

TABLE III

CHARACTERISTICS (INCLUDING μ, σ, $\mu+3\sigma$, $\mu+6\sigma$) OF MAXIMAL RISING/FALLING LOGIC OUTPUT SIGNAL ARRIVAL TIME (ns) GIVEN BY SPSTA, SSTA, MONTE CARLO SIMULATION AND PRIMETIME-SI IN CROSSTALK-OBLIVIOUS, CROSSTALK-AWARE NON-ITERATIVE, CROSSTALK-AWARE ITERATIVE ANALYSES FOR FOUR TEST CASE CIRCUITS, RESPECTIVELY.

			SPSTA				SSTA				MC Simulation			PT
			μ	σ	$\mu+3\sigma$	$\mu+6\sigma$	μ	σ	$\mu+3\sigma$	$\mu+6\sigma$	μ	σ	max	max
AES S-Box	xtalk-oblivious	r	3.358	0.037	3.469	3.580	3.358	0.037	3.469	3.580	2.020	0.429	3.473	4.110
		f	3.370	0.043	3.500	3.629	3.370	0.043	3.500	3.629	2.100	0.295	3.444	4.070
	xtalk-aware non-iterative	r	3.163	0.163	3.652	4.140	3.220	0.147	3.661	4.101	2.073	0.435	3.848	4.708
		f	3.323	0.135	3.729	4.135	3.355	0.136	3.762	4.170	2.179	0.307	3.872	4.918
	xtalk-aware iterative	r	3.360	0.037	3.470	3.581	3.228	0.093	3.506	3.784	2.184	0.299	3.479	4.708
		f	3.370	0.043	3.500	3.629	3.382	0.103	3.691	3.100	2.327	0.317	3.497	4.918
AES Key-Exp	xtalk-oblivious	r	6.107	0.075	6.330	6.554	6.288	0.071	6.502	6.716	4.843	0.490	6.231	6.650
		f	6.204	0.074	6.426	6.649	6.204	0.074	6.426	6.653	4.965	0.538	6.407	6.100
	xtalk-aware non-iterative	r	6.092	0.318	7.046	8.000	6.646	0.248	7.392	8.137	5.275	0.579	7.298	6.672
		f	5.968	0.318	6.922	7.877	6.582	0.245	7.317	8.052	5.120	0.615	7.079	6.127
	xtalk-aware iterative	r	6.286	0.075	6.510	6.733	6.277	0.186	6.834	7.392	4.942	0.502	6.488	6.672
		f	6.205	0.074	6.427	6.650	6.181	0.216	6.427	6.650	4.602	0.661	6.609	6.127
LEON IFU	xtalk-oblivious	r	4.860	0.020	4.919	4.977	4.893	0.020	4.952	5.010	3.308	0.500	4.903	5.840
		f	5.095	0.031	5.187	5.280	5.119	0.031	5.212	5.305	3.785	0.189	5.154	6.130
	xtalk-aware non-iterative	r	4.893	0.020	4.952	5.011	4.893	0.020	4.952	5.011	3.712	0.353	4.995	5.849
		f	5.119	0.031	5.212	5.305	5.119	0.031	5.212	5.305	4.255	0.357	5.295	6.141
	xtalk-aware iterative	r	4.860	0.020	4.919	4.977	4.893	0.020	4.952	5.011	3.683	0.374	4.912	5.849
		f	5.095	0.031	5.187	5.280	5.119	0.031	5.212	5.305	4.355	0.241	5.179	6.141
LEON IDU	xtalk-oblivious	r	10.596	0.109	10.921	11.247	10.961	0.060	11.142	11.323	8.575	0.385	10.181	12.020
		f	10.436	0.107	10.756	11.076	10.791	0.057	10.961	11.131	8.984	0.324	10.453	11.810
	xtalk-aware non-iterative	r	10.597	0.126	10.975	11.354	11.032	0.072	11.247	11.462	9.109	0.421	10.794	12.242
		f	10.438	0.124	10.811	11.184	10.865	0.069	11.072	11.279	9.035	0.404	10.628	12.034
	xtalk-aware iterative	r	10.596	0.109	10.921	11.247	10.971	0.060	11.152	11.332	9.007	0.357	10.335	12.242
		f	10.436	0.107	10.756	11.076	10.801	0.057	10.971	11.142	9.008	0.442	10.457	12.034

Observation 4: SPSTA achieves an average of 1.85%, 2.54% and 1.53% inaccuracy for maximum logic output signal arrival time based on the $\mu+3\sigma$ metric in crosstalk-oblivious, crosstalk-aware non-iterative and crosstalk-aware iterative analysis compared with 1.35 million Monte Carlo simulation runs, respectively.

V. CONCLUSION

We propose input-aware statistical timing analysis, notably Signal Probability-based Statistical Timing Analysis (SPSTA), for crosstalk-induced delay variation consideration. SPSTA provides less pessimism and improved accuracy by signal propagation path/network-based timing analysis and by leveraging existing automatic test pattern generation (ATPG) techniques for false path/network and false crosstalk aggressor identification. Our experiments based on the 45nm Nangate open cell library show that compared with 1.35 million Monte Carlo simulation runs, while PrimeTime-SI overestimates by 14.64%, 16.39% and 19.69%, SSTA achieves an average of 2.85%, 2.89% and 3.59% inaccuracy, and SPSTA achieves an average of 1.85%, 2.54% and 1.53% inaccuracy in crosstalk-oblivious, crosstalk-aware non-iterative and crosstalk-aware iterative analysis, respectively.

REFERENCES

[1] A. Agarwal, D. Blaauw, V. Zolotov, and S. Vrudhula. Statistical timing analysis using bounds. In *Proc. Conf. on Design Automation and Test in Europe*, pages 62–68, 2003.

[2] A. Agarwal, F. Dartu, and D. Blaauw. Statistical gate delay model considering multiple input switching. In *Proc. ACM/IEEE Design Automation Conf.*, pages 658–663, 2004.

[3] K. Agarwal, D. Sylvester, D. Blaauw, et al. Variational delay metrics for interconnect timing analysis. In *Proc. ACM/IEEE Design Automation Conf.*, pages 381–384, 2004.

[4] S. Chun, T. Kim, and S. Kang. Atpg-xp: Test generation for maximal crosstalk-induced faults. *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, 28(9):1401–1413, 2009.

[5] H. Fatemi and P. Tehrani. Crosstalk timing window overlap in statistical static timing analysis. In *Proc. Intl. Symp. on Quality Electronic Design*, pages 245 – 251, 2013.

[6] B. Franzini, C. Forzan, D. Pandini, P. Scandolara, and A. Dal Fabbro. Crosstalk aware static timing analysis: A two step approach. In *Proc. Intl. Symp. on Quality Electronic Design*, pages 499 – 503, 2000.

[7] Aeroflex Gaisler. LEON SPARC V8 Processors. `http://www.gaisler.com/`.

[8] P. Goel. Podem-x: An automatic test generation system for vlsi logic structures. In *Proc. ACM/IEEE Design Automation Conf.*, pages 260–268, 1981.

[9] I-De Huang, S.K. Gupta, and M.A. Breuer. Accurate and efficient static timing analysis with crosstalk. In *Proc. IEEE Intl. Conf. Computer Design*, pages 265 – 272, 2002.

[10] B. Liu. Signal probability based statistical timing analysis. In *Proc. Conf. on Design Automation and Test in Europe*, pages 562–567, 2008.

[11] F. N. Najm. A survey of power estimation techniques in vlsi circuits. *IEEE Trans. VLSI Systems*, 2(4):446–455, 1994.

[12] NIST. FIPS 197, Advanced Encryption Standards (AES), 2001. `http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf`.

[13] M. Orshansky and A. Bandyopadhyay. Fast statistical timing analysis handling arbitrary delay correlations. In *Proc. ACM/IEEE Design Automation Conf.*, pages 337–342, 2004.

[14] M. Orshansky and K. Keutzer. A general probabilistic framework for worst case timing analysis. In *Proc. ACM/IEEE Design Automation Conf.*, pages 556–561, 2002.

[15] Silicon Integration Initiative (SI2). Nangate Open Cell Library. `http://www.si2.org/openeda.si2.org/projects/nangatelib/`.

[16] C. Visweswariah et al. First-order incremental block-based statistical timing analysis. In *Proc. ACM/IEEE Design Automation Conf.*, pages 331–336, 2004.