On Silicon-Based Speed Path Identification

Leonard Lee, Li-C. Wang
Department of ECE, UC-Santa Barbara
lylee, licwang@ece.ucsb.edu

Praveen Parvathala, T. M. Mak
Intel Corporation
praveen.k.parvathala, t.m.mak@intel.com

Abstract

Speed path identification is an indispensable step for pushing the design timing wall and for developing the final speed binning strategy in production test. For complex high-performance designs, pre-silicon timing tools have so far not been able to deliver satisfactory results in predicting the actual speed limiting paths on the silicon. The actual speed paths are mostly uncovered through test and silicon debug, where tremendous manual effort is involved. This paper presents a novel approach as the first step for automating the speed path identification process. Our approach is silicon-based, meaning that timing information is extracted through testing of silicon sample chips. We call this step silicon learning. Based on silicon learning, we present an iterative flow for speed path identification. Experimental results are presented to explain the new methodologies and to demonstrate the effectiveness of our techniques.

1. Introduction

Speed paths are those paths that limit the performance of a chip. There are at least two purposes for finding the speed paths. First, speed paths may be the places where potential design fixes can be applied. By applying these fixes, we can further improve the performance and push the design timing wall. Second, speed paths may indicate places where potential holes in the design methodologies exist. These paths require further attention and study. Usually, additional tests are produced for each path in order to exercise the worst delays on the path. These tests can be used as part of the test suite for speed binning in mass production.

Speed path identification (SPI) is a crucial step in the post-silicon stage for speed-sensitive products. For complex high-performance designs, it is commonly recognized that the critical paths reported from the pre-silicon timing analysis tools rarely correlate well to the actual speed paths for two reasons: (1) Any pre-silicon analysis tool is only as accurate as the model and the algorithms it uses. Obtaining 100% accurate process models for nanometer processes is difficult. Moreover, the analysis algorithms often try to compute approximate results because of the complexity. (2) In the pre-silicon phase, factors that affect circuit timing are analyzed separately. While timing behavior on the silicon is a result of all factors mingled together, in timing analysis, it may not be possible to mingle the analysis of all these factors together because of its high complexity. Therefore, SPI remains a crucial problem to be resolved in the post-silicon stage. A silicon-based SPI methodology needs to resolve two important issues:

- To identify the speed paths, we need an effective diagnosis framework.
- To ensure that the results are not biased by a pattern set, we need a meaningful methodology to add patterns.

Suppose that from the first silicon, we obtain M sample chips $\{c_1, \ldots, c_M\}$. An initial set of patterns are applied on these chips and N patterns $\{P_1, \ldots, P_N\}$ are identified to be the speed-limiting patterns. For each pattern, logical diagnosis decides a set of paths that potentially affect the delay of the pattern. Suppose that the union of these path sets results in a total of L paths. In other words, the potential speed paths after logical diagnosis of the N patterns are $\{Path_1, \ldots, Path_L\}$. Essentially, the inputs to the SPI problem are two matrices, R and S:

$$R = \begin{vmatrix} d_{11} & d_{12} & \cdots & d_{1M} \\ d_{21} & d_{22} & \cdots & d_{2M} \\ \cdots & \cdots & \cdots & \cdots \\ d_{N1} & d_{N2} & \cdots & d_{NM} \end{vmatrix} \qquad S = \begin{vmatrix} s_{11} & s_{12} & \cdots & s_{1L} \\ s_{21} & s_{22} & \cdots & s_{2L} \\ \cdots & \cdots & \cdots & \cdots \\ s_{N1} & s_{N2} & \cdots & s_{NL} \end{vmatrix}$$

where d_{ij} is the delay observed on sample chip c_j by pattern P_i and s_{ij} indicates if path $Path_j$ can affect the delay of pattern P_i, i.e. if $s_{ij} = 1$, then $Path_j$ can affect the delay of pattern P_i; otherwise $s_{ij} = 0$ and it does not affect the delay of the pattern. We note that given a pattern P_i, $d_{i1} \neq d_{i2} \neq \cdots \neq d_{iM}$ because of process variations. We call R the delay matrix and S the sensitization matrix. Given R and S, the SPI problem is to deduce the top k speed-limiting paths.

Logical diagnosis relies on logic simulation and utilizes no timing information. One can imagine that logical diagnosis involves tracing back from the failure point(s) (a scannable flipflop or a register) to extract the potential paths that may contribute to the pattern delay. The effectiveness of logical diagnosis is obviously crucial to the success of SPI (to decide S); however, the diagnosis problem to be solved here is statistical in nature because $d_{i1} \neq d_{i2} \neq \cdots \neq d_{iM}$ for each pattern P_i.

The notion of statistical diagnosis was first proposed in [1]. However, the methodology proposed in that work is not applicable in the SPI application just described because it relies on the assumption of an underlying fault (error) model in order to restrict the search space.

The results of logic diagnosis (matrix S) depend on the initial patterns applied to observe the silicon behavior. Even with a good statistical diagnosis framework, the top k speed paths resulting from one pattern set can be quite different from another.

In this work, our objective is to propose new methodologies aiming to overcome the two challenges mentioned above. For statistical diagnosis, what we need is a new approach that does not rely on an assumed fault model. We call this type of statistical diagnosis *silicon learning* in order to differentiate it from the statistical diagnosis problem defined in [1]. For the issue related to the pattern set, we need a silicon-based test flow where extracted silicon information can be used to guide additional test generation.

1.1. The meaning of top k speed paths

Given M sample chips, the top k speed paths can be defined in several ways. In this paper, we take an "average" point of view. We rank paths based on patterns' average delays across the M sample chips. Alternatively, one can rank paths based on patterns' maximum delays (or 3σ delays) obtained from the sample chips.

By simplifying the delay matrix to their average values (to a delay vector), one may wonder why statistical diagnosis is still required in SPI. For example, suppose that by using a very good timing analysis tool, we can decide L potential speed limiting paths $Path_1, \ldots, Path_L$. The problem is that we do not know which k of them are the top k speed paths, where $k \ll L$. Suppose that we can compose L patterns $P_1, \ldots, P_L$ such that P_i sensitizes only $Path_i$. By taking the delay of P_i averaged over the M samples, we know the average path delay for $Path_i$. Then, these average path delays can be used to rank paths and decide the top k paths.

This simple methodology is not feasible due to two issues (even though we have such a timing analysis tool):

- Given a path $Path_i$, there may not exist a pattern that sensitizes only this path. It may be the case that sensitizing $Path_i$ always involves the sensitization of other paths.
- The path delay of a path $Path_i$ may be affected by the sensitization of other paths. Therefore, we cannot rely on single-path-sensitization patterns to decide the speed-limiting paths. In other words, suppose that pattern P_1 sensitizes only path $Path_1$ and P_2 sensitizes only path $Path_2$. Suppose that another pattern P_3 sensitizes paths P_1 and P_2 together and does not sensitize any other paths. It is not true that $Delay(P_3) \leq \max[Delay(P_1), Delay(P_2)]$. It is possible that sensitizing P_1 and P_2 together may cause excessive delay due to their interaction. Such an interaction can be caused by Multiple Input Transition [2].

Because of these two issues, the SPI problem cannot be solved by assuming that we are given a sensitization matrix where one pattern sensitizes only one path. Moreover, we cannot assume that the initial pattern set is sufficient to expose the worst scenarios of path delays. This means that we need two methodologies: one to draw statistical inference about path delays from (average) pattern delays and the other to allow additional patterns to be added.

The rest of the paper is organized as the following. In Section 2 we propose silicon learning and the silicon-based test flow as the two methodologies to overcome the issues discussed. Section 3 and Section 4 describe the implementations of the silicon learning methodology and the test flow, respectively. Section 5 presents initial experimental data. Section 6 analyzes the experimental results in detail. Section 7 concludes the paper and discusses future research.

2. Silicon learning and test flow

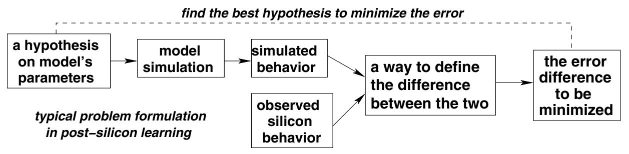
find the best hypothesis to minimize the error
a hypothesis on model's parameters
model simulation
simulated behavior
a way to define the difference between the two
the error difference to be minimized
typical problem formulation in post-silicon learning
observed silicon behavior

Figure 1. Problem formulation - silicon learning.

Figure 1 presents the general problem formulation for *silicon learning*. In silicon learning, the objective is to derive values for the parameters of interest based on observed silicon behavior. These parameters are associated with a design model, not with an error model. In silicon learning, we need to define two important components: (1) model simulation, and (2) error difference function.

Model simulation contains a special simulator to simulate the logic netlist of the circuit with an assignment of parameter values. The simulated results are compared with the observed results. This comparison is based on an *error difference function*. The error function determines the "difference" between the simulated results and the observed results. The formulation defines a minimization problem: finding the best hypothesized parameter values in order to minimize the difference.

Figure 2 shows the silicon learning of path delays. The model parameters are based on a given set of paths, and the parameter values are the path delays. The model simulation takes the hypothesized path delays and estimates pattern delays. The simulator can be implemented based on logic path

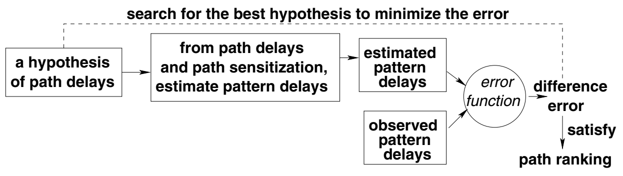
search for the best hypothesis to minimize the error
a hypothesis of path delays
from path delays and path sensitization, estimate pattern delays
estimated pattern delays
observed pattern delays
error function
difference error
satisfy
path ranking

Figure 2. Apply silicon learning to derive path ranking.

sensitization. With the hypothesized delays on the paths, we can estimate a pattern delay based on those paths sensitized by the pattern. The objective is then to find the best path delay assignments in order to minimize the difference between estimated pattern delays and observed pattern delays. The final answer is given as a *path ranking* based on the final path delay values.

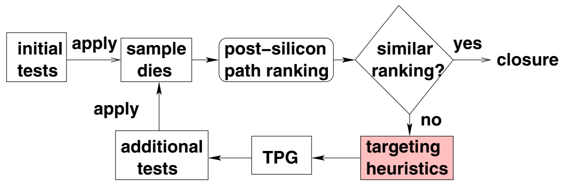

Figure 3. SPI test flow using silicon learning.

Figure 3 shows how the results of this silicon learning can be incorporated into a silicon-based test flow for SPI. The SPI flow is iterative, where the silicon-learned path rankings in two consecutive iterations are compared to decide the stopping criteria.

One key component in this test flow is the *targeting heuristic* that decides which paths should be targeted for producing additional tests. The targeting heuristic is based on the estimated path ranking. A simple heuristic can be to target the top k paths in the current ranking.

The test flow provides two nice properties: (1) additional TPG effort is guided through learned silicon information, which in this case is represented as a path ranking. (2) The flow has a clean way to reach closure, which is determined by the change of path ranking between two consecutive iterations. A closure point indicates that investing more effort in silicon learning and/or TPG will not help more in SPI.

In this work, we assume that a TPG tool is available. TPG in the functional domain remains to be an open problem. For speed binning, efforts have been made to replace functional tests with structural tests [3]. Here, we do not intend to debate whether functional tests or structural tests should be used for speed binning. For the rest of the paper, our goal is to develop the methods for realizing the path-based silicon learning methodology and the proposed test flow, and to devise experiments for studying their feasibility and effectiveness. We note that the proposed silicon learning methodology and SPI test flow are independent of the type of patterns in use.

In this work, silicon samples are produced by simulation that is based on models for process variations. Because our chip samples are simulated samples, we know what the top speed paths are. Hence, the effectiveness of the methodologies in Figure 2 and Figure 3 can be quantified by comparing the ”silicon-extracted” speed paths to the true speed paths from the simulation.

3. Implementing silicon learning

The path-based silicon learning in Figure 2 derives path ranking. We call it a *ranking optimization* problem in [4] and propose an iterative algorithm for it. Here, we briefly discuss the key points in the algorithm. The inputs to the problem contain a set of N patterns $\{P_1,\ldots,P_N\}$ and a set of M silicon sample chips. By applying these patterns at various frequencies on the samples, we discover the first failing frequency of each pattern on each sample. This value is treated as a pattern delay. For each pattern, we utilize its average delay d_i over all samples.

Given P_i, logic sensitization analysis gives us a set set_i of paths that have a chance to decide the pattern's delay. Our work does not try to optimize logical diagnosis. Hence, for each pattern we simply extract all possible paths that may potentially affect its delay [4]. For simplicity, we use structural test patterns for the experiments. Our objective is to demonstrate the effectiveness of silicon learning and the proposed test flow methodology. Therefore, we began with a simple experimental setup to avoid the complexity of involving functional patterns. However, we note that our proposed methodologies are general and not limited to the application with structural test patterns only.

The ranking optimization consists of:

1. Inputs are the pattern set $P = \{P_1,\ldots,P_N\}$, average-delay set $D = \{d_1,\ldots,d_N\}$, and set of sensitized path subsets $Set = \{set_1,\ldots,set_N\}$ (the sensitization matrix S).
2. From (P,D,Set), calculate a path delay for path p as $D_p = \frac{w_1d_1+\cdots+w_kd_k}{(w_1+\cdots+w_k)}$, where the pattern delay d_i appears in D_p iff the path $p \in set_i$. All the assigned weights w_i begin at 1.
3. By using fixed-delay simulation with the calculated path delays, we obtain the calculated pattern delays.
4. Calculate the *total error value of difference* between the calculated pattern delays and the observed pattern delays. The calculation of this error is defined in [4].
5. Apply a simple greedy heuristic to adjust the weight assignment for the pattern whose error value is among the largest. Repeat steps 2 through 4 until the total error is brought down to a desired level.
6. Construct a path ranking based on their delay values calculated in the last iteration.

The idea of the greedy heuristic is simple: In each iteration, we will increase the weight assignment of a pattern whose error value is among the largest. This will reduce its error value in the next iteration. By changing a pattern's weight w_i, we will change the calculated path delays for all paths contained in set_i in the next iteration, which consequently alters the path ranking.

4. Implementing the test flow

To obtain the initial set of patterns in Figure 3, we first use a timing analysis tool to extract a set of paths for the circuit. Given these paths, we use a SAT-based path oriented ATPG tool [7] to produce robust patterns for the paths, one pattern for each path. The don't-care primary inputs are filled randomly with 0 or 1s. Usually, this will ensure that each pattern's delay can be affected by many paths, not just the target path. We use the same sensitization criteria as that in [4] to decide if a path has a chance to affect a pattern's delay. This sensitization is different from path delay testing, where a path is sensitized and considered to be covered if it is certain that the path delay fault can be detected by the pattern. Since an extracted path can be a false path, the ATPG may fail to produce a pattern for these paths; however they would still be kept in the initial path set.

This work utilizes a statistical timing simulator (STS) developed in the past. The prototype simulator was used for the work in [4][5][6]. Our statistical timing model in the simulator assumes pin-to-pin delay random variables. The timing model is cell-based, and interconnects' delays are included. For experimental purposes, the random delay distributions were obtained using a Monte-Carlo-based SPICE simulator (ELDO) [8]. We extracted the cells' pin-to-pin delay distributions from a 0.25μm, 2.5V CMOS technology.

To obtain silicon-based timing behavior in our experiments, we use STS to produce a circuit sample by assigning a random value based on the statistical timing model to each pin-to-pin delay. After generating M circuit samples, STS applies patterns onto these samples. The resulting pattern delay information from STS is our silicon-based timing behavior.

4.1. SPI test iteration

The first iteration of the SPI test flow consists of:

- Use STS to apply the initial patterns onto a number of sample chips, collecting delays for these patterns.
- Apply the path-based silicon learning described in Section 3 to derive a path ranking.
- As this is the first iteration, there is no previous path ranking to compare with, and so closure cannot be determined.

Subsequently, every iteration consists of:

1. Use the top G ranked paths from the last iteration's ranking as the *targeting* paths.
2. Use the ATPG to generate patterns for the top G ranked paths, n patterns for each path. These new patterns are added into a cumulative pattern set, which includes the initial patterns.
3. Use STS to apply the new patterns onto the same sample chips. We add the new pattern delays into a cumulative set, which includes the initial pattern delays from iteration 0.
4. Apply silicon learning to derive a path ranking, using the cumulative pattern set and cumulative delay set.
5. Compare the top C ($C < G$) ranked paths from the current iteration to the top C ranked paths from the previous iteration. Determine if: (a) Closure, as defined in Section 4.2, has been reached, then the top C ranked paths are identified as the speed paths. (b) Closure has not been reached, then continue to the next iteration.

4.2. Closure in the SPI test flow

To terminate the SPI test flow, we introduce two different strategies to check if the speed path ranking from one iteration is similar enough to that from the previous iteration, as shown in Figure 3. We can claim that the SPI test flow has reached closure once the criteria defined here has been reached. Both strategies compare the top C ranked paths from the current iteration to the top C ranked paths from the previous iteration.

- *Set closure* is a simple set equivalence check, ignoring path ranking. For example, closure can be reached if the top C ranked paths are the same in two consecutive iterations.
- *Ranking closure* takes into account the ordering of the paths. We assign each of the top C ranked paths a rank number, starting with 0 for the path with the largest delay and $C-1$ for the path with the shortest delay. Let $r_p(i)$ be the rank number for a path p in iteration i. We define the **ranking error** of iteration i as $\mathbf{R_e(i)} = \Sigma_{p=1}^{C}|r_p(i) - r_p(i-1)|$. If set closure has not been reached, then we let $R_e(i) = \infty$. Otherwise, ranking closure is reached when $R_e(i) \leq T_e$, where T_e is a threshold value. For the strongest ranking closure (call it **exact ranking closure**), we have $T_e = 0$.

5. Experiments and results

In this section, we present experimental results to demonstrate the effectiveness of path-based silicon learning and the SPI test flow. We compare the *silicon-based path ranking* obtained through silicon learning to the *true path ranking* calculated from the simulation. Recall that our silicon experiments are done through STS simulation and hence, we know the true path ranking for each case. In reality, the true path ranking is unknown.

In STS, the number of simulated samples is 100. As mentioned earlier, in the path ranking, a path with the small-

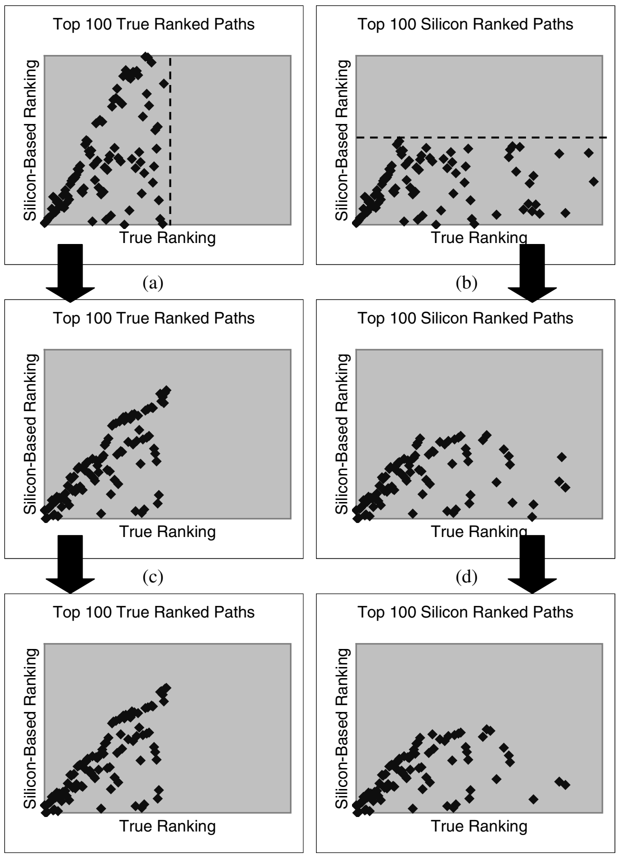

Figure 4. Results upon reaching closure on c880 by various criteria. (a) and (b) show the rankings at iteration 0. (c) and (d) show the improved rankings upon reaching set closure (with $R_e = 30$). (e) and (f) show the rankings after reaching ranking closure (with $R_e = 16$).

est rank value has the largest learned delay. In these experiments for the SPI test flow, $G = 300$ and $C = 100$. In each iteration, 1 additional pattern is generated for each path in the top $G = 300$ paths in the current path ranking. We allow the experiments to run indefinitely, only terminating when an exact ranking closure has been reached. A set closure is always reached before or concurrently with an exact ranking closure.

In Figure 4, we present experimental results based on the ISCAS benchmark circuit c880 by selecting an initial path set with 1271 paths. Then, we produce a pattern for each path to create the initial pattern set. In these plots, we evaluate the effectiveness of the proposed methodology against the true path ranking. All plots have x-axis and y-axis scales from 0 to 200.

In plots (a), (c), and (e), we plot the top 100 paths from the true path ranking. Because of this, no points fall beyond the line $x = 100$ (shown in (a)). In contrast, plots (b), (d), and (f) are for the top 100 paths from the silicon-based path ranking. Hence, in these plots, no points fall above the line $y = 100$ (shown in (b)).

Plots (a) and (b) are results from iteration 0 in the SPI test flow. They demonstrate the effectiveness of the path-based silicon learning working on the initial pattern set. Plots (c) and (d) are results based on reaching a *set closure*. Plots (e) and (f) are results based on reaching a *ranking closure*.

The nice correlations between the silicon-based ranking and the true ranking demonstrate the effectiveness of the proposed methodology. These plots show that for many paths, their true ranks can actually be uncovered through silicon learning, and they can be improved through iterations of the SPI test flow (from plots (a), (b) to plots (c), (d) and then, to plots (e), (f)).

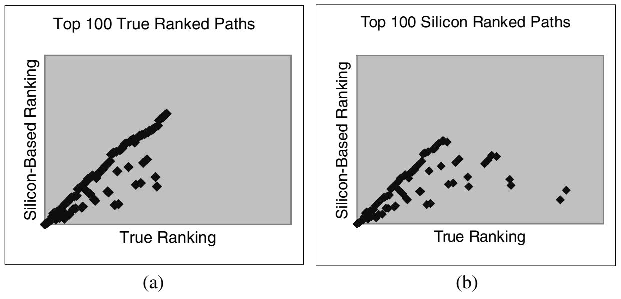

Figure 5. Results when reaching exact ranking closure (with $R_e = 0$).

Figure 5 shows the results upon reaching an exact ranking closure (a ranking closure with error $R_e = 0$). As can be seen, by enforcing a more stringent closure criterion in the SPI test flow, the silicon-vs-true ranking correlation improves significantly.

Figure 6 shows similar results for s1488, and Figure 7 shows the results for s5378. Again, we show the ranking comparisons at iteration 0 (plots (a) and (b)), set closure (plots (c) and (d)), and exact ranking closure (plots (e) and (f)). A similar improvement can be observed upon reaching different closure criteria.

6. Analysis of results

In the previous section, we compare silicon-based path rankings to their corresponding true path rankings. In practice, the top $C = 100$ silicon-based ranked paths are those used for further analysis. Instead of the paths' actual rankings, it is sometimes more important to know how many among the top 100 paths extracted from the SPI test flow are among the true top 100 paths. We present these results below.

Let S_{true} be the true top 100 speed paths. Let S_{si} be the top 100 speed paths extracted from the SPI test flow. We want to know the percentage of coverage $Cov = \frac{|S_{true} \cap S_{si}|}{|S_{true}|}$.

It is also important to study the number of iterations needed to reach closure in each SPI test flow experiment.

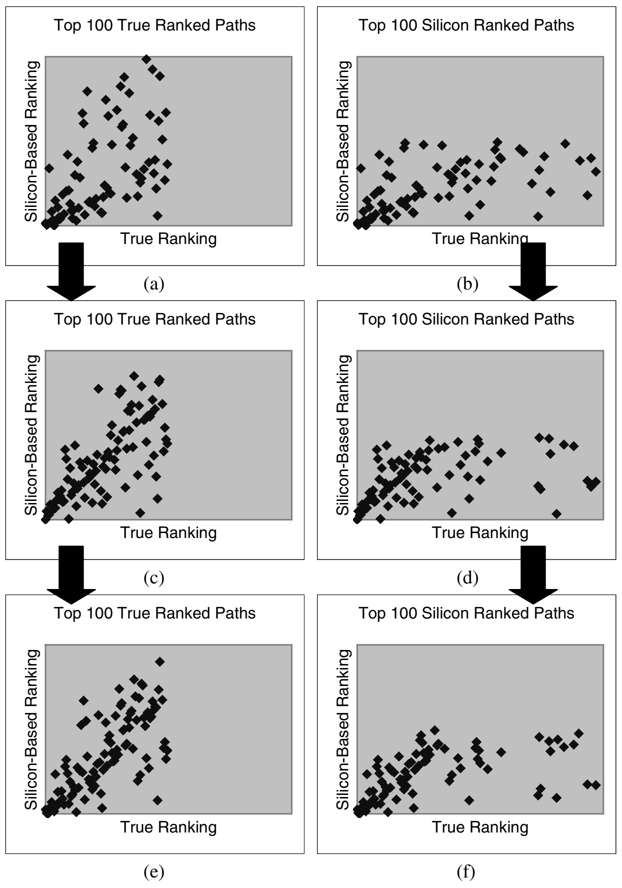

Figure 6. Results for s1488. (a) and (b) show the rankings at iteration 0. (c) and (d) show the rankings upon reaching set closure, with an error of 62. (e) and (f) show the rankings at exact ranking closure.

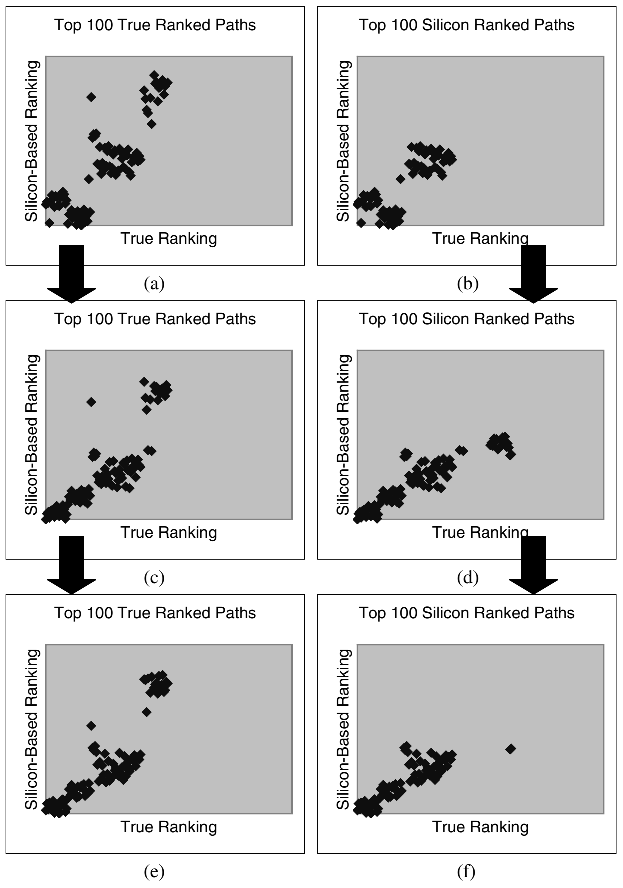

Figure 7. Results for s5378. (a) and (b) show the rankings at iteration 0. (c) and (d) show the rankings upon reaching set closure, with an error of 132. (e) and (f) show the rankings at exact ranking closure.

Table 1 shows the results of *Cov* and the numbers of iterations needed. There are several interesting points to observe from these data.

- From the path coverage point of view, set closure is good enough. Although in Figure 5, exact ranking closure can help to improve the correlation between silicon ranking and true ranking, it does not improve path coverage by much.
- The "best *Cov*" data are the highest coverages found in all SPI test iterations. One may wonder why we do not stop the iteration at these points. This is because when applying the SPI test flow in reality, we are not supposed to know the true answer S_{true}. The results of "best *Cov*" are shown here so that they can be compared to the results established at set and exact ranking closures. Because the closures are based on the comparison of the top 100 paths found in two consecutive iterations in the SPI test flow, these results demonstrate that our closure criteria are valid. The answers found by our closure criteria are close to the best answer found during the iterations of our methodology.
- The number of iterations required to reach a closure is a big concern because each SPI test iteration involves test generation as well as test application. Unless we have an integrated and automatic test generation/application framework, it is important to minimize the required number of iterations to reach closure as much as possible.

Table 1. Coverages at key iterations.

Circuit	Iteration 0	Set Closure		Exact $R_e = 0$		Best *Cov*	
	Cov	Iteration	*Cov*	Iteration	*Cov*	Iteration	*Cov*
c880	66%	45	75%	68	76%	20	78%
s5378	67%	10	79%	92	78%	8	79%
s1488	59%	27	64%	69	68%	58	69%

6.1. Reduce the number of iterations in SPI test flow by adding more patterns in each iteration

Recall that in the above experiments, in each SPI test iteration, 1 pattern is produced for each of top 300 paths. This is actually a very ineffective approach in practice. Because the TPG targets on only the top 300 silicon-based ranked paths, in each iteration we can produce as many patterns as the tester memory allows. Generating as many patterns per path as possible in each SPI test iteration allows silicon learning to be based on as complete information as possible. It is interesting to see if this change can reduce the required number of iterations to reach closure.

Table 2. Coverages at key iterations for c880 and s5378.

Circuit	Iteration 0	Set Closure		Exact $R_e = 0$		Best *Cov*	
	Cov	Iteration	*Cov*	Iteration	*Cov*	Iteration	*Cov*
c880	66%	**7**	77%	18	81%	14	81%
s5378	67%	**3**	78%	24	78%	2	78%

Table 2 shows the results based on generating 100 patterns per path in each SPI test iteration. Two interesting points can be observed: (1) The numbers of required iterations are dramatically reduced from those in Table 1 and the *Cov* results are improved. Figure 8 shows the ranking correlation plots for c880 and s5378, when set closures are reached. The results are better than before.

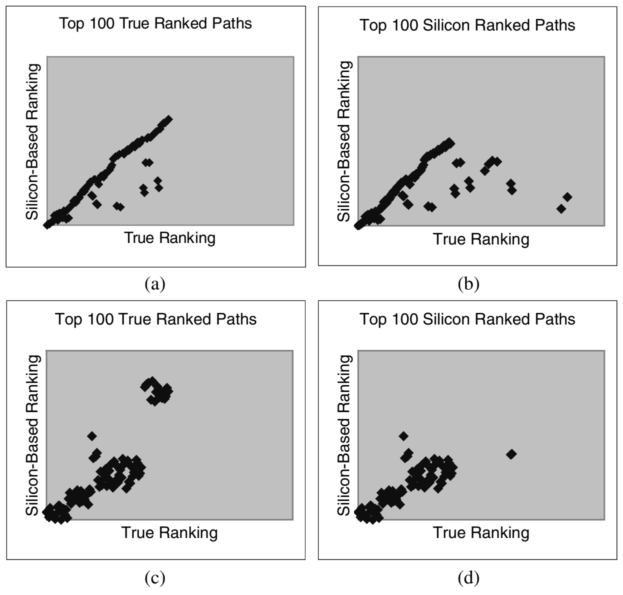

Figure 8. Results upon reaching set closure when generating 100 patterns per path per iteration instead of 1. (a) and (b) show the rankings for c880, and (c) and (d) show the rankings for s5378.

7. Conclusion

In this paper, we propose a novel framework for silicon-based speed path identification. The SPI framework consists of two methodologies: silicon learning and SPI test flow. The objective of silicon learning is to perform statistical diagnosis to extract silicon-based information, and that of the SPI test flow is to utilize the extracted information to guide further TPG in order to reach a test closure.

We develop the experiments to study the effectiveness of the proposed methodologies. Among the top 100 speed paths identified through our framework, roughly 65-80% are the actual top 100 speed paths. For these paths, their silicon-based rankings usually correlate well to their true rankings. These results show the effectiveness of our approach.

Depending on the application, *set closure* usually is good enough, which allows the SPI test flow to stop at an earlier iteration. To minimize the number of required SPI test iterations, as many patterns as possible should be generated for each selected path in each iteration. This strategy has a significant impact on the number of required iterations to reach SPI test closure. Moreover, this strategy can deliver better results in terms of path coverage *Cov* and path ranking.

In our experiments, set closure is reached if 100% of the top *C* silicon-based ranked paths are the same in two consecutive iterations. If the number of required iterations to reach this set closure is of concern, further reduction in the number of iterations can be achieved by relaxing the 100% constraint.

The current experiments over-simplify the role of logical diagnosis by using ISCAS benchmark circuits with structural tests. In practice, logical diagnosis will be important if the proposed framework is to be applied in the functional domain. In that case, we need to develop an effective logical diagnosis methodology to be used with the proposed SPI test flow. Moreover, we will need an effective functional TPG approach to minimize the overhead of producing additional patterns in each iteration of the SPI flow.

References

[1] Angela Krstic, et. al. Delay Defect Diagnosis Based Upon Statistical Timing Models – The First Step. in *Proc. DATE*, 2003, pp. 328-323

[2] Agarwal, A.; Dartu, F.; Blaauw, D. Statistical gate delay model considering multiple input switching in *Proc. DAC*, 2004 pp. 658-663

[3] Cory, B.D.; Kapur, R.; Underwood, B. Speed binning with path delay test in 150-nm technology. *IEEE D&T*, Vol 20, No 5, 2003, pp. 41- 45

[4] Leonard Lee, Li-C. Wang, T.M. Mak, and Kwang-Ting Cheng. A Path-Based Methodology for Post-Silicon Timing Validation. in *Proc. ICCAD*, 2004.

[5] Li-C. Wang. Regression Simulation: applying path-based learning in delay test and post-silicon validation in *Proc. DATE*, March 2004, pp. 692-693.

[6] Li-C. Wang, et. al. On Path-based Learning and Its Applications in Delay Test and Diagnosis in *Proc. ACM/IEEE DAC*, June 2004,

[7] Kai Yang, et. al. TranGen: A SAT-Based ATPG for Path-Oriented Transition Faults. in *Proc. ASP-DAC*, Jan 2004.

[8] Anacad. Eldo v4.4.x User's Manual. 1996.