Maximizing Crosstalk-Induced Slowdown during Path Delay Test

Dibakar Gope  
Department of Electrical and Computer Engineering  
University of Wisconsin-Madison  
Madison, Wisconsin  
gope@wisc.edu

D. M. H. Walker  
Dept. of Computer Science  
Texas A&M University  
College Station, Texas  
walker@cse.tamu.edu

Abstract—In this paper, we present a timing-driven test generator to sensitize multiple aligned aggressors coupled to a delay-sensitive victim path to detect the combination of a delay spot defect and crosstalk-induced slowdown. The framework uses parasitic capacitance information, timing windows and crosstalk-induced delay estimates to screen out unaligned or ineffective aggressors coupled to a victim path, speeding up crosstalk pattern generation. In order to induce maximum crosstalk slowdown along a path, aggressors are prioritized based on their potential delay increase and timing alignment. The test generation engine introduces the concept of alignment-driven path sensitization to generate paths from inputs to coupled aggressor nets that meet timing alignment and direction requirements. In addition, two new crosstalk-driven dynamic test compaction algorithms are developed to control the increase in test pattern count. The proposed test generation algorithm is applied to ISCAS85 and ISCAS89 benchmark circuits. SPICE simulation results demonstrate the ability of the alignment-driven test generator to increase crosstalk-induced delays along victim paths.

Index Terms—Automatic test pattern generation (ATPG), crosstalk, path delay test, aggressors.

I. INTRODUCTION

With continuous scaling of process technology in deep sub-micron (DSM) regime, capacitive coupling between adjacent interconnect wires continues to increase and now dominates total interconnect capacitance. This leads to signal crosstalk noise, causing timing violations, reduced timing margin and signal glitches. Switching activity in capacitive-coupled nets can speed up or slow down the victim path if the nets involved in coupling have simultaneous or near simultaneous transitions. In most circuits, crosstalk-induced delay leads to circuit failure [1]. The need to magnify the impact of this crosstalk delay becomes increasingly important to determine whether the circuit can still meet its timing specification.

Moreover, in modern DSM circuits, physical synthesis avoids long parallel runs of signal nets, to minimize noise from any one coupling capacitor. Because of logical constraints and different timing windows of aggressor sites, it is quite improbable for a single delay test pattern to excite a large number of aggressors on a single victim net. Significant crosstalk delay increases can only occur due to multiple aggressors coupling to multiple victim nets along a victim path.

Most of the prior work on crosstalk ATPG either did not examine the impact of multiple aligned aggressor nets along a timing-critical victim path [1][2][3][4] or they failed to consider the timing alignment of aggressor-victim nets [5][6]. As a result, the delay of the tested paths may be less than the worst case, leading to a test escape. Moreover, they [1][2][3][4] suffer from the computational complexity of their algorithm. The ATPG technique [5] has the advantage of reusing the existing transition fault infrastructure, but the disadvantage of not being able to determine timing alignment.

New crosstalk test pattern generation algorithms must focus on sensitizing a maximal subset of timing-aligned aggressors coupled to different victim nets along a victim path. The key contributions of this work are:

1. Timing-oriented test generation to target multiple aligned aggressors along a victim path to maximize crosstalk delay.
2. Alignment-driven path sensitization to generate paths from primary inputs (PIs) to coupled aggressor nets that meet the required timing alignment and direction.
3. Two crosstalk-driven dynamic compaction algorithms to limit the number of test patterns when incorporating crosstalk.

The rest of the paper is organized as follows. Section II presents the crosstalk delay model used in this work. Section III discusses the aggressor pruning methods to select the potential timing-aligned aggressors. The timing-driven crosstalk ATPG is proposed in section IV. Section V proposes two dynamic compaction algorithms. Section VI presents the experimental results on ISCAS85 and ISCAS89 circuits. Section VII demonstrates the increase in path delay by our crosstalk patterns using HSPICE simulations. Section VIII concludes this work.

II. CROSSTALK-INDUCED DELAY MODELING

Signal crosstalk between a victim net and its neighboring aggressor nets may either speed up or slow down the victim path depending on the transition direction. transition arrival time overlap and coupling capacitance between the victim and aggressor nets [7]. Transitions on aggressors change the nominal effective capacitance ($C_{eff}$) seen by the victim net’s driving gate and thus change the signal transition delay. In charge sharing based coupling models, crosstalk is modeled by scaling the physical coupling capacitance ($C_i$) with a Miller Coupling Factor ($MCF$) to obtain the $C_{eff}$. 

\[ C_{\text{eff}} = MCF \cdot C_c \]

If the aggressor transition overlaps with the victim transition and is in the same (helper) direction, then \( C_{\text{eff}} \) is reduced and the victim path speeds up. However, if the aggressor and victim nets switch in the opposite direction, then \( C_{\text{eff}} \) is increased and the victim path slows down. The following delay equation is used to estimate the potential crosstalk delay increase caused by the \( i \)-th aggressor switching at the same time, but in the opposite direction to the coupled victim net:

\[
\Delta \text{Delay}_{\text{crosstalk}} = \left( \frac{C_{\text{ci}}}{C_i + \sum_{k=1}^{n} C_{ck}} \right) \cdot \text{Delay}_{\text{driver-stage}}
\]

where \( \Delta \text{Delay}_{\text{crosstalk}} \) is the crosstalk-induced delay increase caused by the \( i \)-th aggressor; \( C_{\text{ci}} \) denotes the coupling capacitance between the \( i \)-th aggressor and the victim net, \( C_i \) is the line capacitance from the victim net to ground, \( n \) is the number of aggressors, \( \text{Delay}_{\text{driver-stage}} \) is the nominal stage delay of the victim net, assuming no transitions on the coupled nets. The denominator of the equation is the nominal value of \( C_{\text{eff}} \), assuming no transitions on the coupled nets. This equation approximates the delay increase linear in change in \( C_{\text{eff}} \) and assumes equal aggressor and victim slew rates with completely overlapping transitions, so the \( MCF \) is 2. We consider linear superposition of aggressor noise to a victim net. In practice, this is the maximum potential delay increase. Our algorithm can be extended to more accurate crosstalk model [8], but details of extension are beyond the scope of this work.

III. TEST GENERATION FOR CROSSTALK-INDUCED DELAY

Our proposed crosstalk-induced delay test generation consists of three major steps: (1) test generation for a delay-sensitive victim path; (2) pruning and sorting aggressor nets coupled to that victim path, based on logic constraints, timing alignment and potential delay increase; and (3) path generation from PIs to aggressor nets that meet timing alignment and transition direction, while also sensitizing the victim path.

In this work, KLPG test generation [9] is used to generate the longest rising and falling victim path through a target line in a circuit under robustness constraints. The victim path generated targeting a line is considered subsequently for timing-oriented crosstalk ATPG. The target line or fault site is assumed to have a spot delay defect. Paths grow from PIs to POs through the target fault site, with most false paths trimmed by direct implications and heuristics, prior to final justification. Crosstalk-induced delay increases are relatively small. They are only of concern if the spot-delay defect on the victim path under test is large enough that the path is almost failing, but not so large that the path fails regardless of crosstalk. Thus aggressors are aligned assuming that the spot-delay defect is equal to the path timing slack. This shifts the nominal transition times downstream from the defect site, as shown in Fig. 1.

A. Aggressor Pruning and Ranking

Aggressor nets that do not cause increases in victim path delay are pruned away. First, the necessary assignments (NAs) to sensitize the victim path may propagate to aggressors. These aggressors are discarded since there is no decision to be made. Second, many coupling capacitors are small and cause insignificant increases in victim path delay. Aggressors are attempted to sensitize only if their potential delay increase metric is above a specified threshold (Th):

\[
\Delta \text{Delay}_{\text{crosstalk}} \geq \text{Threshold}
\]

where \( \Delta \text{Delay}_{\text{crosstalk}} \) is the potential crosstalk delay increase caused by an aggressor, defined in Section II. The threshold is set by analyzing the potential delay increase vs. ATPG cost.

Fig. 1. (a) Path never fails, no crosstalk impact; (b) crosstalk can cause delay fault; (c) path always fails, no crosstalk impact.
IV. TIMING-ORIENTED CROSSTALK ATPG

After sorting the potential aggressors for a victim path, the aggressor with maximum coupling effectiveness is considered for sensitization in presence of victim path NAs. The goal is to find a propagation path from PIs to the aggressor that has the best timing alignment. In practice, alignment is probabilistic, depending on process variation, supply noise, and other unmodeled effects. Since we are concerned with paths that are too slow, the alignment requirement can be indirectly accounted for by using min-max gate delays in the coupling effectiveness ranking. In our work, we will use nominal circuit delays during the search for a path to the aggressor that achieves the best timing alignment, ignoring any lack of correlation due to noise or process variation. Each aggressor shifts the timing alignment of later nets on the victim path. This can be handled by updating transition times along the victim path, but the shift in alignment is small enough that this is not considered in this work.

A. Aggressor Path Store

In the aggressor path generation phase, a path store is used to store the partial paths, which are the paths originating from the PIs but have not reached the aggressor of interest. The search space for each aggressor net, as shown in Fig. 3, is the fan-in cone of the aggressor net. Paths outside the search space can provide side-input constraints for gates on the path. Fig. 4 shows an example. The partial path starts from PI, and ends at gate $g_i$. A set of partial paths are grown from the PIs towards the aggressor net, with the goal of sensitizing a path to the aggressor and achieving the best timing alignment with the victim net. Partial paths are initialized as rising and falling transitions from all the PIs of the aggressor fan-in cone that do not already have NAs from victim path sensitization.

The earliest and latest aggressor transition times are associated with each partial path. These are the sum of the length of the partial path and the min-max path delay from its last node to the target aggressor. The partial paths are sorted by their potential timing alignment to the victim net. The timing alignment metric is calculated as:

$$TimingWindow = D_{agg} \cdot (Max) - D_{agg} \cdot (Min)$$

$$LeftWindow = T_{Victim} - D_{PartialPath} - D_{agg} \cdot (Min)$$

$$RightWindow = D_{PartialPath} + D_{agg} \cdot (Max) - T_{Victim}$$

$$WindowDiff = RightWindow - LeftWindow$$

$$TimingAlign = C2 \cdot TimingWindow + C3 \cdot WindowDiff \cdot \text{1}$$

where $TimingAlign$ is the timing alignment metric. $D_{agg}$ is the path delay from the last node of the partial path to the target aggressor, $D_{PartialPath}$ is the length of the partial path. The other variables are as described earlier. In Fig. 4, the length of the partial path $g_0...g_i$ is 10, the min/max path delay from $g_i$ to the aggressor is 5/12. The victim net transition timing ($T_{Victim}$) is shown as 20. Assuming the value of $C2$ and $C3$ as 0.5 and 0.5 respectively, the timing-alignment of this partial path is 0.2.

B. Aggressor Path Generation

In each iteration of path generation, the partial path with the maximum timing alignment value is popped from the path store and extended by adding a fan-out gate that can potentially achieve the best possible aggressor alignment to the victim net transition. If the last gate of the partial path has multiple fans-out, the path will split, leaving alternate choices in the path store. For example, the partial path $g_0...g_i$ is extended by adding gate $g_j$, because extending the partial path to $g_j$ brackets the victim net transition most evenly [Fig. 5(a)] when compared to extending on fan-out gate $g_k$ [Fig. 5(b)]. After the partial path is extended $(g_0...g_j)$, the constraints to propagate the transition on the added gate $(g_j)$ are applied.

After the partial path is extended, direct implications are used to trim off false paths. If a partial path reaches the target aggressor, it becomes a complete path. It also means that the NAs from the new aggressor path sensitization are consistent with the existing victim path NAs. Then a PODEM-based
justification is performed on the combined sets of NAs to find a test pattern that simultaneously sensitizes the victim path and the target aggressor net. The aggressor path generation process is shown in Fig. 6.

V. CROSSTALK-AWARE COMPACTION

Timing-oriented crosstalk ATPG targets one aggressor at a time. However, a single aligned aggressor transition has small impact in increasing the victim path-delay. So the crosstalk ATPG should attempt to excite maximal possible number of aggressors with required timing alignment and direction along a victim path. Once aggressors are sensitized one at a time using our crosstalk ATPG, they are compacted together into the test pattern of the victim path in decreasing order of their coupling effectiveness. That way the final compacted pattern will maximize the impact of crosstalk slowdown. Maximizing the victim path-delay increase is a form of the maximum cover problem. The cost of a near-optimal solution for this problem does not make sense given our timing model approximations. We instead use a greedy algorithm [10], targeting aggressors in decreasing order of potential delay increase. So it may miss the worst possible delay increase, both due to the order dependence and stopping when the couplings are too small. However, in our experience, a small number of larger coupling capacitances dominate the delay increase. So a greedy algorithm will come close to obtain the worst delay increase, particularly when the timing alignment is uncertain due to intra-die process variation.

One approach is to first compact the maximal number of aligned aggressor paths into the test pattern of the target victim path. The NAs of the victim path and aggressors sensitized so far are used to constrain the search space for later (lower potential delay increase) aggressors, as shown in Fig. 7. We term this aggressor-first dynamic compaction, since we first compact as many aggressors as possible per victim path, then compact these group of aggressors-coupled victim paths into patterns.

The aggressor-first compaction procedure will maximize the crosstalk-induced delay increase on each victim path, but may cause a substantial increase in the number of test patterns, compared to a test set that ignores crosstalk. This pattern inflation can be avoided by compacting the victim paths first and then sensitizing aggressors, which we term pattern-first compaction, as shown in Fig. 8. The coupled aggressor nets to a victim path are sensitized in the presence of NAs from all the victim paths in a compacted pattern. The additional NAs from multiple victim paths in each compacted pattern will preclude sensitization of many aggressors. The same process will be repeated for the other compacted patterns in the set. Within a pattern, victim paths will be targeted in decreasing order of their length.

VI. EXPERIMENTAL RESULTS

The proposed path delay test generator maximizing crosstalk-induced slowdown is implemented in Visual C++ and run on a 64-bit Windows 7 PC with Intel Core 2 Duo processor (2.66GHz) and 4GB of memory. Experiments are performed on ISCAS85 and ISCAS89 benchmark circuits. For our experiments, the synthesized netlist of the benchmark circuits are layed-out using SoC Encounter in TSMC 45nm technology. During the detailed place and routing process in SoC Encounter, the NanoRoute router automatically tends to minimize crosstalk by wire spacing, net ordering, using long parallel wires and selecting routing layers for noise-sensitive nets. The NanoRoute router also performs shielded routing to protect critical wires from crosstalk. Parasitic information, such as coupling capacitance, net capacitances are extracted in the SPEF (Standard Parasitic Exchange Format) file and are
used to estimate the crosstalk delay increase in our test generator. Net-to-net nominal delays reported in the extracted Standard Delay Format (SDF) file are used for STA delay computation.

A. Aggressor Pruning

TABLE I. and TABLE II. show the results of aggressor pruning for the aggressor-first and pattern-first compaction on ISCAS85 circuits. Column 2 and 3 reports the victim path count and total number of potential aggressors coupled to those paths before any pruning respectively. Column 4 lists the aggressors with potential delay increase as more than 1% of the victim path delay under test. Existing NAs from the victim path forbid some of the aggressors to set an opposite transition. Column 5 lists aggressors after pruning for victim path NAs. Column 6 reports the number of aggressors that meet timing alignment and transition direction. For most of the ISCAS85 circuits, the 1% delay increase threshold reduces the number of aggressors by 60-70%. Of the remaining aggressors, approximately 20% have transition windows that bracket the victim transition and so are considered for alignment-driven sensitization. In pattern-first compaction, increased NAs from multiple victim paths in each compacted pattern results in less number of potential aggressors after logical pruning.

B. Timing-Oriented Crosstalk ATPG

TABLE III. compares the crosstalk test generation results between aggressor-first and pattern-first compaction with delay increase threshold set at 1%. Column 2 lists the compacted test patterns without considering crosstalk. Columns 3 and 7 compare the number of potential aggressors between aggressor-first and pattern-first compaction. These are the aggressors that meet timing alignment and transition requirements during aggressor pruning steps. Columns 4 and 8 list the number of sensitized aggressors. Columns 5 and 9 show the test pattern count. Column 3 shows that aggressor-first compaction sensitizes 60-75% of the potential aggressors. Sensitizing crosstalk prior to compaction increases pattern count by 150-200% (column 5). Although there is no increase in pattern count with pattern-first compaction, but the increased NAs from multiple victim paths in each compacted delay pattern preclude sensitization of many aggressors. This results in less number of potential aggressors left after logical pruning in column 7 when compared to column 3. The additional NAs of the multiple victim paths also reduce the search space for aggressor path generation in pattern-first. This leads to an abrupt drop in the number of sensitized aggressors in column 8.
when compared to aggressor-first compaction in column 4. The decrease in sensitized aggressors per victim path in pattern-first compaction will result in lower crosstalk delay increase. There is clearly a trade-off between pattern count and test quality.

C. ATPG Run-Time Overhead

TABLE V. lists the CPU time for each component of the crosstalk-driven pattern generation using 1% delay threshold. The number of aggressors for which a path from PIs cannot be found is shown in column 2. Column 3 lists how many complete paths failed justification. Column 4 and 5 report the CPU time required to generate the victim paths and to prune the initial aggressor candidates respectively. Column 6 lists the CPU time to generate aligned transitions for the potential aggressors and dynamically compact those to maximize the effects of crosstalk-induced delay on a victim path. As we can see, the aggressor pruning step takes little time. Most of the time is either spent in victim path generation or aggressor sensitization. The amount of time in aggressor sensitization is dominated by the number of aggressors that fail sensitization or justification in columns 2 and 3. Since justification is the most expensive step in victim path and aggressor path generation, the benchmarks with more aggressor paths failing justification spend more time in crosstalk pattern generation. So speeding up the algorithm is mostly dependent on using a faster justification procedure. TABLE VI shows the CPU time for pattern-first compaction. The increased NAs filter out more aggressors, so there are fewer justification failures and so much lower CPU time. TABLE IV shows the crosstalk pattern generation results on ISCAS89 circuits.

VII. COMPARISON AND CORRELATION

In order to verify that our alignment-driven test generator maximize crosstalk delays, we perform HPSICE circuit simulation of several delay-sensitive victim paths using our crosstalk patterns in large representative circuits (c1355, c5315, c7552). Further we compare their delay distribution against the
crosstalk patterns. The unspecified bits in a conventional PDF pattern are filled in different ways, e.g. zero-fill or random-fill. The delay increase for a path using random-filled PDF pattern and proposed crosstalk pattern is computed with respect to the path delay from the corresponding zero-filled PDF pattern ignoring crosstalk. These 10 paths have experienced increase in delay only because of crosstalk coupling. For all the victim paths selected (except one), the crosstalk patterns have longer delay than the zero-filled and random-filled patterns. Columns 4, 5 and 6 in TABLE VII. further confirm this by comparing the number of aligned aggressors sensitized by the three pattern set. Random-filling of the unspecified bits in path no. 9 have created additional fortuitous aligned aggressors coupling to the same path, which results in more delay increase than crosstalk patterns. For few paths (path no. 4 and 5) in TABLE VII., random-filling reduce path delays than zero-filled path delays ignoring crosstalk. This is due to helper transitions being accidentally generated by the random-filling of unspecified bits. Helper transitions are denoted by the negative sign for path no. 4 in column 5.

Fig. 9 further shows the delay increase (in ps) for 45 testable paths spread over the entire path delay distribution of c5135. We chose a set of 45 random paths that have a cumulative delay increase of more than 15ps from all the aligned aggressors during crosstalk ATPG. Further the number of sensitized aggressors coupled to them is at least 4. These set of 45 random paths might have affected by all kind of hazards along with crosstalk. We can see from Fig. 9, for most of the victim paths selected, the crosstalk patterns induce more delay than the zero-filled and random-filled path delay patterns. However there are quite a few cases in Fig. 9, where the zero-filled or random-filled patterns slow down the victim paths more than crosstalk patterns. The reasons behind this are:

A. Delay at the Side-Input Transition during Robust Test

The proposed crosstalk engine satisfies the conditions of robust test to generate victim paths. As the second condition of robust test allows to-non-controlling transitions at the side fans, a late transition at side fan-ins can delay the propagation of the on-path input transition, thus potentially can change the on-path timing down the victim path. A delay fault is still detected, but not on the victim path. In our work, we leverage the timing of the victim path to align the coupled.
aggressors. However, it is observed during circuit simulations that sometimes the side-input to-non-controlling transitions slow down the on-path transition. This in turn changes the victim path timing that our crosstalk test generation has utilized to align the coupled aggressors along the path. For some of the victim paths such as path 27, 33 and 42 for c5315 in Fig. 9, the crosstalk patterns cannot align the aggressor transitions to the victim path due to late side-input transitions. As a result, they do not experience any delay from coupling noise. In addition, for some of the victim paths like path 2, 3, 7, 24, 42 in Fig. 9, the zero-filled and random-filled patterns delay the side-input transitions and slow down the victim path, causing more delay increase than the crosstalk patterns.

**B. Mismatch in Cell Characterization**

In our crosstalk engine, we use SDF reported delays for timing analysis. However, the path delays obtained in HSPICE simulations are about 20-25% less than the path delays estimated using SDF data. So it is quite likely that our crosstalk patterns cannot generate aligned transitions in HSPICE simulations, due to this mismatch between HSPICE and SDF delay. In order to reduce the mismatch in cell characterization, we increase the ambient temperature to 55C during HSPICE simulations. We observe that for majority of cases the delays of the victim and aggressor paths track at elevated temperature. As a result, our crosstalk patterns are still effective in generating aligned transitions in HSPICE. Fig. 10 and Fig. 11 show delay increase plots for c7552 and c1355 respectively.

**VIII. CONCLUSIONS AND FUTURE WORK**

In this work, we proposed a novel crosstalk ATPG to maximize the impact of crosstalk slowdown. The current approach does not consider the uncertainty in alignment due to process variation and side-input transition delays. This work uses nominal delays for victim path transition. In future work, we will set a delay window around the victim path transitions and sweep aggressor alignments across the windows.

**ACKNOWLEDGMENT**

This research was funded in part by the Semiconductor Research Corporation under contract 2010-TJ-2096 and by the National Science Foundation under grant CCF-0702669. We sincerely thank Shayak Lahiri for his help during the project.

**REFERENCES**


