# XTalkDelay: A Crosstalk-aware Timing Analysis Tool for Chip-level Designs

Yinghua Li[*]
UC Berkeley
Berkeley, CA
yinghua@eecs.berkeley.edu

Rajeev Murgai  Takashi Miyoshi
Fujitsu Laboratories of America, Inc.
Sunnyvale, CA
{murgai,miyoshi}@fla.fujitsu.com

Ashwini Verma[*]
Amdocs
San Jose, CA
ashwini.verma@amdocs.com

## Abstract

*This paper describes* XTalkDelay, *an industrial-strength methodology and tool for measuring the impact of crosstalk on delays of paths in a design. The main cornerstone of* XTalkDelay *methodology, vis-a-vis other approaches, is its high delay computation accuracy. It deliberately avoids the use of approximate models for cells and nets and interconnect reductions.* XTalkDelay *employs a path-based approach; uses detailed and accurate distributed RC parasitics for critical nets and their aggressors; uses BSIM3-accurate gate models; and invokes HSPICE for delay computation using only the minimum required set of input patterns.* XTalkDelay *has been successfully applied on two industrial designs.*

## 1 Introduction

In deep sub-micron circuit designs, the coupling capacitance between adjacent interconnects has become significant as the wires become taller and narrower while the distance between them decreases. Due to these changes, crosstalk noise between physically adjacent nets has become an important concern [3]. The affected net is known as the *victim* $v$ and the neighboring switching net(s) $a$ causing the noise is (are) called *aggressor(s)*. As shown in Figure 1, crosstalk can cause the arrival time of the victim to increase (decrease) when aggressors switch in a direction opposite to (same as) the victim. The distributed coupling capacitance between $v$ and $a$ is shown as $C_ci$. Crosstalk can also lead to logic hazards and circuit malfunction [2]. For instance, if i) the delay change is large enough to render a critical path slower than the clock cycle and thus cause a timing violation, or ii) the large spike generated on the victim due to capacitive coupling with a switching aggressor is close enough to the clock edge so as to latch a wrong value at the target flip-flop, the circuit can malfunction.

Accurate computation of the delay change due to crosstalk then becomes very important in the circuit design process. The paper addresses this problem in the context of a chip-level design and computes accurate path delays in the presence of crosstalk coupling. The paper organization is as follows. Previous work on crosstalk and its impact on delay is summarized in Section 2. We present the proposed methodology and tool XTalkDelay in Section 3, including its I/O and flow details. Section 4 presents and analyzes results obtained by applying XTalkDelay on two industrial designs. In Section 5, we critique our approach and conclude with directions for future work.

[*] This work was done when these authors were at Fujitsu Laboratories of America as summer intern and software developer respectively.

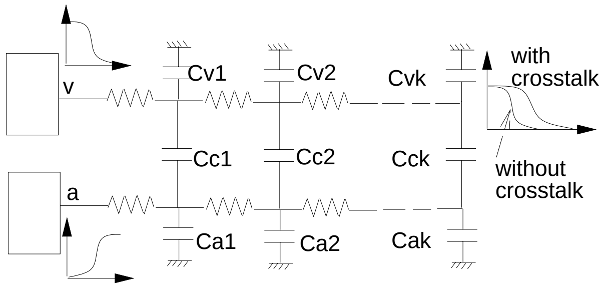

Figure 1: Crosstalk & its impact on delay of victim $v$

## 2 Previous Work

The relevant literature can be classified broadly as addressing two problems: 1) the problem of crosstalk-aware delay computation for a single net and 2) the problem of crosstalk-aware timing analysis of a circuit. A solution to the second problem requires applying the solution to the first problem to at least a subset of nets in the circuit.

An accurate solution to the first problem (i.e., crosstalk-aware delay computation for a single net) can be provided by circuit or timing simulation techniques such as SPICE. However, such techniques are inherently slow and cannot be used to analyze large systems. When the system can be modeled as a linear circuit, linear model reduction techniques such as [13, 12] can improve the speed and hence help handle larger systems. However, the cost is still too much for the complex interconnects. Examples of applying reduction techniques to the crosstalk domain include [4, 6, 8, 11]. [4] derives closed form equations for slow-down and speed-up effects of crosstalk. However, the interconnect model used is a single self capacitance for the victim and for the aggressor and a single coupling capacitance between victim and aggressor. This is simplistic, since i) it uses a lumped model and ii) it ignores the interconnect resistance, thus ignoring resistive shielding and attenuation of signal strength. [6] and [8] improve the circuit template used to model the interconnect. [11] derives an expression of the victim voltage assuming a linear ramp for the input waveform (or at most a combination of three linear ramps) and a two-pole approximation for the victim waveform. The accuracy is shown to be within 12% of SPICE.

Techniques have also been proposed to estimate the maximum crosstalk noise [7, 18]. Their main strength is that they are fast and can be used in the inner loop of physical design to quickly calculate an upper bound on the noise. However, due to the unrealistic assumptions such as the aggressor input being an infinite ramp, the accuracy of [7] is quite low and its applicability is limited to short aggressor wires with large slews. [18] derives an upper bound on the *effective peak noise*. However, the bound can be optimistic and the authors do not provide the maximum resulting error. Moreover, these two papers do not address the problem of delay change due to crosstalk.

We believe that due to simplifying assumptions and reductions used, most of the aforementioned techniques such as [4, 6, 7, 11, 18] are neither accurate nor general enough to be included in an accurate crosstalk-aware timing analysis methodology for a circuit.

A separate body of research work focuses on the second problem, i.e., that of crosstalk-aware timing analysis of a circuit [15, 9, 1, 16]. [15] describes a static timing analysis (STA) tool to calculate the longest path in the design taking into account the impact of crosstalk on gate delays.[1] This work, however, uses a simplistic net-based analysis and ignores changes in net delays due to crosstalk. [9] proposes a novel two-step approach in the timing analysis tool CASTA: initially use a simple yet pessimistic interconnect macromodel and obtain a quick net ranking according to the crosstalk effects, and in the second step, progressively reduce the pessimism by applying more accurate macromodels. To cut down the run-time, it derives Thevenin equivalent model for the driver and uses interconnect reduction techniques. But no accuracy results vis-a-vis an accurate simulator like SPICE are presented to justify the driver models and reductions. Another problem with this approach is that it is net-based. Two paths passing through a victim net $v$ can have different switching times at $v$ and depending on the aggressors timing windows, may be affected non-identically by different aggressors at those times. A net-based technique does not distinguish between the two paths and uses the worst-case scenario for $v$. Subsequently, the delay of a path is computed by adding the worst-case net (and cell) delays. The path pessimism in the net-based approach is thus at least the sum of the net pessimisms. Like [9], [1] also proposes a two-step STA methodology, albeit slightly different. In the first step, a switch factor (also called coupling compensation) approach [10] is used to calculate the worst-case equivalent capacitance-to-ground for each net, which is then used to derive a superset of violating paths. In the second step, these paths are analyzed more carefully. Although this approach seems to finally avoid the source of pessimism inherent in the net-based analysis, the details of the all-important second step are sketchy and no results on the accuracy of the technique are provided. [16] presents a method where degradation tables are built which capture the delay effects due to crosstalk for different values of relative signal arrival time (difference between aggressor and victim signal arrival time). These degradation tables can be used during timing analysis.

[1] In this paper, we will use gate & cell interchangeably. Same holds for arrival & switching times, and slew & transition time.

## 3 XTalkDelay

We propose a highly accurate analysis methodology and tool called XTalkDelay to measure the effects of crosstalk on path and circuit delay. The main highlights of our approach are as follows.

1. Our approach is path-based and does not suffer from the pessimism inherent in the net-based analysis described above. For each path $p$ under analysis, for a given victim net, the true aggressors and their switching times are computed based on the switching time of the victim net with respect to $p$.

2. For a given path, the delays through gates and nets (lying on the path) in the presence of crosstalk are computed very accurately using HSPICE. Our approach models nets as distributed RC network. We apply no macromodel reduction techniques.

3. As we will show, crosstalk has a significant impact on gate delays (in addition to the net delays). Hence it is very important to use accurate gate delay models. Our approach uses BSIM3 gate models, which is in contrast to the simple resistive models used in the previous work.

4. To compute gate delays, static timing analysis tools typically replace the interconnect parasitics at the output net by a single effective capacitance $C_{eff}$. Computation of $C_{eff}$ is approximate and is an attempt to fit the output-load based cell delay model used in STA tools. Instead, our methodology uses HSPICE and the complete RC network at the output net to compute the pin-to-pin delay through the gate and is very accurate.

5. For cell delay recomputation, we present a method which generates the minimum number of patterns that

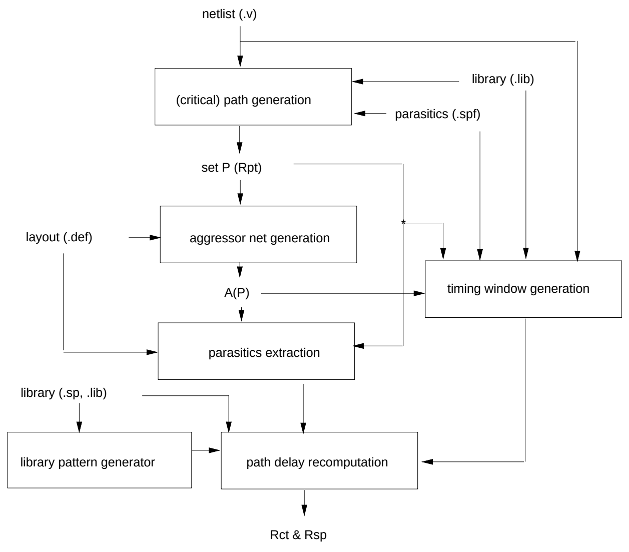

Figure 2: XTalkDelay flow

should be simulated to derive the worst case pin-to-pin delay through a cell on the critical path for a given input-output pin pair and transition directions.

6. XTalkDelay generates SPICE-accurate delay reports for the critical paths for two scenarios: one, in the presence of switching aggressors and coupling capacitances, and the other, in their absence. This allows the designers to see the impact of crosstalk easily.

7. We have successfully applied our methodology on two industrial designs and present the results in this paper.

The overall methodology of XTalkDelay is as follows. It assumes that a mapped, placed and routed design is available. XTalkDelay recomputes the delay of a set of (critical) paths in the presence of neighboring aggressor nets. First, XTalkDelay identifies potential aggressor nets for each net $v$ of a (critical) path $p$. Then it extracts parasitics for $v$ in the presence of the aggressors. The parasitics include distributed coupling capacitances, self capacitances and resistances. Finally, XTalkDelay recomputes $p$'s delay by traversing $p$ from the start-point, recomputing the delay and slew through each cell on $p$ & the associated output net $v$ in the presence of the coupling capacitances and aggressor transitions.

## 3.1 I/O

XTalkDelay reads a mapped and post-routed design, including a gate-level hierarchical netlist and placement and routing data. The designer can also provide an optional list $P$ of paths which he/she wishes to analyze for delay in the presence of the capacitive coupling. If such a list is provided, it is expected to have the actual arrival times and transition times for all points (pads or pins) on each path of $P$. If the path list $P$ is not provided (default mode), XTalkDelay will automatically generate an intermediate timing report $R_{pt}$ which contains the list $P$ of critical and near-critical paths in the design. XTalkDelay also requires the cell library and SPICE model files.

Currently, XTalkDelay invokes CAD vendor tools for the following tasks: PrimeTime for STA, StarXtract for parasitic extraction, and HSPICE for circuit analysis and delay computation.[2]

The output of XTalkDelay are two timing reports $R_{ct}$ and $R_{sp}$. The timing report $R_{ct}$ contains timing information for each path $p$ in $P$ in the presence of crosstalk. Actual arrival times and slews at all points and delays through cells and nets on $p$ are reported. The second report $R_{sp}$ contains the same timing information, but in the absence of crosstalk from switching aggressors. The difference between $R_{sp}$ and the timing report $R_{pt}$ generated by PrimeTime is that $R_{sp}$ is generated using HSPICE. Since PrimeTime is usually pessimistic as compared to HSPICE (i.e., PrimeTime reports higher delay numbers as compared to HSPICE), it is better to compare $R_{ct}$ with $R_{sp}$: both are generated using HSPICE and are more accurate than PrimeTime.

## 3.2 Flow

The main steps in the XTalkDelay flow are shown in Figure 2 and are as follows.

1. **(Critical) Path Generation**: If not provided, generate the set of critical paths $P$. All nets on $P$ along with their directions (i.e., rise or fall) and arrival times constitute the set $V$ of victim nets. For our designs, a PrimeTime script is used to generate this set. This script also reads in the net parasitics to model the interconnect. Let $t_{PT}$ be the path delay computed by PrimeTime.

2. **Aggressor Net Generation**: For each net $n$ in $V$, compute the set of potential aggressor nets, $A(n)$. These are the nets that are *physically close to* $n$. Given $n$, the following procedure computes $A(n)$. First, from the layout, all the net segments $NS(n)$ of $n$ and their end point coordinates are determined. Next, those segments in the entire design which are within some user-defined maximum distance (in terms of grids) from some segment in $NS(n)$ are extracted. The owning nets of these segments determine the set of possible aggressor nets $A(n)$ for $n$. The user can also specify the minimum length for which a net segment must run in parallel with a segment in $NS(n)$ to qualify as an aggressor. Let $n \cup A(n) = S(n)$, also called the *victim-aggressor set*.

3. **Parasitics Extraction**: For each $n \in V$, a parasitic extraction tool (StarXtract) is used to generate the parasitics for the victim-aggressor set $S(n)$. The parasitics form an

[2]All the three tools are from Synopsys.

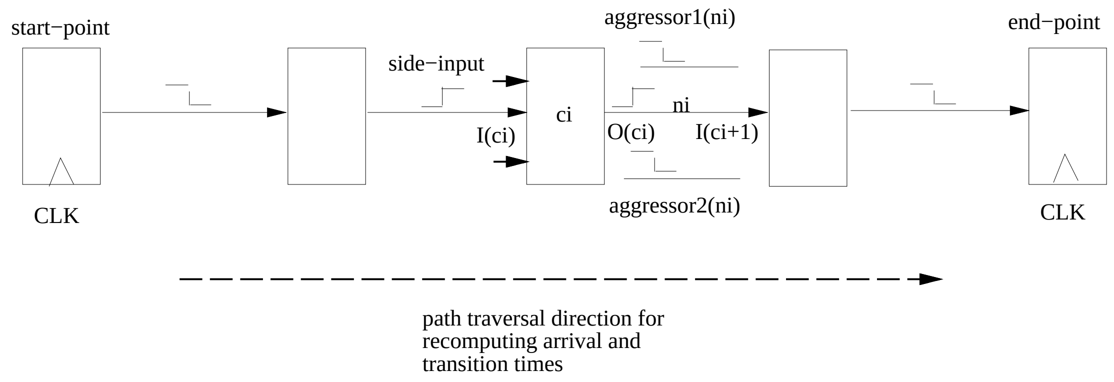

Figure 3: Path delay recomputation

RC network, which includes distributed net resistances, capacitances to ground and coupling capacitances between the nets in $S(n)$. An example of such a network is shown in Figure 1 for the victim $v$ and a single aggressor $a$.

4. **Timing Window Generation**: Compute timing window for each net $m$ in $\cup_{n \in V} S(n)$. Ideally, in order to reduce pessimism and false timing violations, we would like the timing window to explicitly list delays of all sub-paths that pass through $m$, i.e., delays from their respective start-points up to $m$. However, we do not have access to any tool that generates such information. We were forced to use PrimeTime, which only computes the shortest and the longest paths. Then, in our case, the timing window consists of min/max & rise/fall arrival times and transition times. These timing windows are used to generate aggressor nets' waveforms, which will be used in the subsequent SPICE simulations.

5. **Path Delay Recomputation**: The delay of each critical path $p \in P$ is recomputed taking into account coupling and aggressor information. As shown in Figure 3, a path $p$ is an alternation of cells and nets, along with the transition direction (rise or fall) at each point. $p$ is traversed from the start-point and delay of each cell $c_i$ from the input pin $I(c_i)$ to the output pin $O(c_i)$ and net $n_i$ rooted at $O(c_i)$ are recomputed. The arrival time and transition time at the sink $I(c_{i+1})$ of $n_i$ are updated. They will be used to compute the delay of the stage $i+1$. The process is continued till the new arrival time at the end-point of $p$ is known.

The delays of the cell $c_i$ and net $n_i$ are computed in the presence of the entire parasitic RC network (including coupling capacitances in the nets $S(n_i)$) and transitions on the aggressor nets $A(n_i)$. To compute the maximum impact of the aggressor $\alpha \in A(n_i)$ on the victim delay, $\alpha$ should make a transition in a direction opposite to that of the victim $n_i$, if the timing window of $\alpha$ contains the actual arrival time of $n_i$ on the path $p$. The arrival time of $\alpha$ is identical to that of $n_i$, and its transition time or slew is the minimum slew in the appropriate direction. This timing information is obtained from static timing analysis. Note that, in general, the minimum slew on the aggressor will result in the maximum delay increase. When the aggressor $\alpha$'s timing window does not contain the arrival time of $n_i$, $\alpha$ is kept static at VDD (if $n_i$ is falling) or GND (if $n_i$ is rising). The arrival time, transition time and direction (rising, falling, or constant VDD/GND) together constitute the aggressor waveform.

The parasitics extracted for $n_i$ and $A(n_i)$ nets are combined with the aggressor waveforms into a SPICE deck. The only missing information in the deck is the values on the side inputs of the cell $c_i$. The side inputs of $c_i$ are all inputs of $c_i$ except $I(c_i)$. Note that the waveform on $I(c_i)$ is already known: it is based on the new arrival time and the transition time computed from the previous stage $i-1$. To measure the worst delay through $c_i$, all possible values need to be specified at the side inputs of $c_i$. The delay is measured with HSPICE for each case, and the maximum of all these delay values yields the worst pin-to-pin delay through $c_i$. This is the *naive approach* for delay computation. We can speed up the delay characterization process significantly by utilizing the information about sensitivity and input/output transitions at $c_i$. For instance, if $c_i$ is a three-input AND gate, with critical input $I(c_i) = x_1$ rising and the output $O(c_i)$ also rising as a result, the naive approach requires 4 SPICE simulations, corresponding to the 4 vectors 00, 01, 10, and 11 at $x_2$ and $x_3$. However, the input transition at $x_1$ can propagate to the output only if the side inputs $x_2$ and $x_3$ are both 1. So only one input vector needs to be applied and simulated. We call this the *smart approach*. In general, assume we are given that the output $O(c_i)$ of the cell $c_i$ implements logic function $f(x_1, x_2, \ldots, x_m)$, where $I(c_i) = x_1$ is on the critical path. Without loss of generality assume $x_1$ makes a rising transition and $f$ makes a falling transition. We are interested in computing the minimum set of patterns that need to be simulated to compute the worst case delay from $x_1$ to the output $f$ of the cell $c_i$ for the given pair of transitions on $x_1$

and $f$. Before $x_1$ rises, $x_1 = 0$ and $f = 1$. This corresponds to the condition

$$g(x_2, x_3, \ldots, x_m) = f_{x_1'}, \quad (1)$$

where $f_{x_1'}$ is the cofactor of $f$ with respect to $x_1 = 0$ and represents exactly those input combinations at $x_2$ through $x_m$ for which $f = 1$ when $x_1$ is set to 0. After $x_1$ rises, $f$ falls. The other inputs of $f$ do not change. The final state is $x_1 = 1$ and $f = 0$. This corresponds to the condition

$$h(x_2, x_3, \ldots, x_m) = f'_{x_1}, \quad (2)$$

where $f'_{x_1}$ represents exactly those input combinations for which $f = 0$ when $x_1$ is set to 1. Since the only input that has changed is $x_1$, the function

$$gh(x_2, x_3, \ldots, x_m) = f_{x_1'} f'_{x_1} \quad (3)$$

represents exactly all the combinations at $x_2$ through $x_m$ which are possible both before and after $x_1$ and $f$ make the transitions in the specified directions. It is easy to see that $f_{x_1'} f'_{x_1}$ also represents the set of combinations for the case when $x_1$ falls and $f$ rises. When both $x_1$ and $f$ rise (or fall), the desired function is

$$f_{x_1} f'_{x_1'} \quad (4)$$

**Example 3.1** *Consider the above three-input AND gate example, where $x_1$ rises and $f$ rises as a result. $f(x_1, x_2, x_3) = x_1 x_2 x_3$. $f'(x_1, x_2, x_3) = x_1' + x_2' + x_3'$. From (4), the function $f_{x_1} f'_{x_1'} = (x_2 x_3)(1) = x_2 x_3$ yields all the $x_2$ and $x_3$ combinations under which rising $x_1$ can result in rising $f$. This implies $x_2 = x_3 = 1$.*

We have developed a SIS-based [17] library pre-processor, which applies the above analysis for each input-output pin pair (and their transition directions) of all library cells, computes the above functions, and generates the minimum set of patterns that need to be simulated. XTalkDelay incorporates these patterns and computes the worst delay through $c_i$ from the input pin $I(c_i)$ to the output pin $O(c_i)$. From the arrival time at $I(c_i)$ and the cell delay, arrival time at $O(c_i)$ is computed. Corresponding to this worst case, we also measure using HSPICE the new net delay from $O(c_i)$ to $I(c_{i+1})$ (which in turn determines the new arrival time at $I(c_{i+1})$) and the transition time at $I(c_{i+1})$. This completes the delay recomputation through the cell $c_i$ and net $n_i$.

Repeating this for all the stages of $p$, we compute $p$'s new delay, $t_{CT}$, which we call the **crosstalk-aware delay**.

6. Since PrimeTime and HSPICE can yield quite different delay values, for an accurate computation of delay change due to crosstalk, we recompute the path delay of $p$ by repeating the above delay computation process, but without using any aggressor switchings: in other words, all aggressors are assumed to be either at VDD or GND. This effectively replaces the coupling capacitances in $S(n_i)$ with capacitances to ground. The path delay thus obtained is called the **SPICE delay** ($t_{SP}$).

| design | #cells | #nets |
|---|---|---|
| D1 | 165K | 167K |
| D2 | 454K | 460K |

1K = 1000.
Both benchmarks are in 0.11-$\mu$ technology.

Table 1: Benchmark statistics

## 4 Experimental Results

We have applied XTalkDelay on two industrial designs: $D_1$ and $D_2$. Both use 0.11-$\mu$ technology and $V_{DD}$ of 1.2V. Table 1 shows the numbers of cells and nets in these two designs. We analyzed the designs after they had been successfully placed and detail-routed. We also extracted the layout parasitics and used them in the static timing analysis tool PrimeTime (version 2002.03-SP1).

In the next subsections, we present results on the impact of crosstalk on path delays and gate delays for $D_1$ and $D_2$, accuracy of our methodology, and the comparison between naive and smart approaches for cell delay characterization.

### 4.1 Impact of Crosstalk on Path Delays

First, we report the results on $D_1$. Initially, PrimeTime reports 65 critical or near-critical paths in $D_1$. Out of these, only 36 are unique: 29 were found to be duplicates and removed. This simple utility reduced the run-time of XTalkDelay by a factor of almost 2, since the run-time is roughly linear in the number of paths analyzed. The total number of critical or victim nets on these 36 paths was 130. The total number of aggressor nets was 309. On average, there were about 2.4 aggressor nets per victim net. It turned out that 68 victim nets had no neighboring aggressor nets. We applied XTalkDelay on each of the 36 paths to compute $t_{SP}$ (HSPICE delay without crosstalk) and $t_{CT}$ (HSPICE delay in the presence of crosstalk). It turns out that only 11 of these paths had a delay change of more than 10ps, i.e., $\Delta t = t_{CT} - t_{SP} \geq 10ps$. Table 2 provides delay information for each of the 11 paths. Paths 7 and 10 have maximum $\Delta t$: more than 350ps. This prompted us to investigate them further. We discovered that on path 7, there was a net $n_1$ that had four aggressors. $n_1$ had an overlap length of 950$\mu$ with two of them and 180-255$\mu$ with the other two. On path 10, there were two nets with significant overlaps: 400-650$\mu$. These paths, their $t_{SP}$ and $t_{CT}$ delays, and the overlap lengths with aggressors were reported to the designers. They verified that coupling did cause these paths to become longer and moved the relevant victim and aggressor nets away from each other to reduce the delay increase.

For the second design $D_2$, PrimeTime reports 60 (unique) critical paths. In all, there are 450 (victim) nets on these paths. They had a total of 247 aggressor nets. It turned out that 336 critical nets did not have any aggressors. XTalkDe-

| **path** | $t_{PT}$ | $t_{SP}$ | $t_{CT}$ | $\Delta t = t_{CT} - t_{SP}$ | $\Delta g$ | $\frac{\Delta g}{\Delta t}(\%)$ |
|---|---|---|---|---|---|---|
| 1 | 3754.5 | 3641.83 | 3682.85 | 41.02 | 34.29 | 83.6 |
| 2 | 3769.3 | 3663.32 | 3703.21 | 39.89 | 33.22 | 83.3 |
| 3 | 3769.1 | 3663.13 | 3703.02 | 39.89 | 33.22 | 83.3 |
| 4 | 3768.73 | 3668.61 | 3707.47 | 38.86 | 32.19 | 82.8 |
| 5 | 3767.42 | 3661.60 | 3701.49 | 39.89 | 33.22 | 83.3 |
| 6 | 1672.37 | 1641.73 | 1657.86 | 16.13 | 9.54 | 59.2 |
| 7 | 4057.18 | 4027.49 | 4379.20 | 351.71 | 350.35 | 99.6 |
| 8 | 1653.58 | 1626.83 | 1637.37 | 10.54 | 10.54 | 100.0 |
| 9 | 1404.57 | 1335.93 | 1354.03 | 18.10 | 14.97 | 82.7 |
| 10 | 4089.96 | 4015.67 | 4384.56 | 368.89 | 347.88 | 94.3 |
| 11 | 1348.9 | 1294.35 | 1332.07 | 37.72 | 37.72 | 100.0 |

All delays are in ps.

Table 2: Critical path delays by PrimeTime, HSPICE, & crosstalk-aware analysis for $D_1$

| **path** | $t_{PT}$ | $t_{SP}$ | $t_{CT}$ | $\Delta t = t_{CT} - t_{SP}$ |
|---|---|---|---|---|
| 1 | 1878.27 | 1821.26 | 1833.89 | 12.63 |
| 2 | 1791.09 | 1758.72 | 1776.79 | 18.07 |
| 3 | 1947.89 | 1893.51 | 1911.73 | 18.22 |
| 4 | 1834.71 | 1798.49 | 1816.67 | 18.18 |

All delays are in ps.

Table 3: Critical path delays by PrimeTime, HSPICE, & crosstalk-aware analysis for $D_2$

lay found only four paths to have a delay increase of more than 10ps. They are listed in Table 3. The main reason why $D_2$ had smaller impact from crosstalk is that the average number of aggressors per victim net was 0.55 as compared to 2.4 for $D_1$. In turn, this was because $D_2$ had already been optimized by the designers for crosstalk prevention. This version of the design was obtained after increasing the spacing between the net segments that had significant coupling.

During our experiments, we also found that any overlap of smaller than $20\mu$ between two net segments did not result in significant coupling capacitance.

## 4.2 Impact of Crosstalk on Gate Delays

We performed an experiment to study the impact of crosstalk on gate delays and the relative contribution of gate delay changes to the path delay degradation $\Delta t$. For the chip $D_1$, for each path reported in Table 2, we computed the sum of the gate delay changes due to crosstalk. This is listed under the column $\Delta g$ in Table 2. The percentage fraction is shown in the column $\frac{\Delta g}{\Delta t}(\%)$. For instance, for path 1, the crosstalk resulted in a delay increase of 41.02ps, out of which 34.29ps was contributed by the gate delay increase. Only 6.73ps increase was due to interconnect. For almost all paths, the contribution of gate delay change to $\Delta t$ was found to be over 83%. This points to a very high impact of crosstalk on gate delays. Therefore, it is important to accurately model and compute not only interconnect delays but also gate delays.[3]

## 4.3 Accuracy of XTalkDelay

We would like to make a few observations on the accuracy of XTalkDelay methodology.

First, note from Table 2 that the PrimeTime delay $t_{PT}$ for a path $p$ is different from $t_{SP}$ on average by 72.5ps (for all these paths, the PrimeTime delay values are greater). This significant difference, we believe, is due to three factors: 1) PrimeTime reduces the interconnect at an output pin to a single $C_{eff}$ in order to compute the cell delay, 2) PrimeTime uses a look-up-table based scheme to compute the cell delay, and 3) PrimeTime does not compute the delay through interconnect as accurately as HSPICE. We noticed several cases where the interconnect delays computed by PrimeTime differed by more than 10% from those computed by HSPICE. Usually the PrimeTime-computed interconnect delays are smaller. This justifies the use of HSPICE in XTalkDelay.

Next, we present data to highlight the inaccuracy of net-based analysis as compared to path-based analysis for crosstalk. In net-based analysis, the maximum arrival time of a net is used to derive aggressors' waveforms. In de-

[3]We did not conduct a similar study for $D_2$, since the path delay degradations $\Delta t$ were not significant for $D_2$.

sign $D_1$, we found a critical net $n$ whose maximum arrival time was $t_m$ = 4694ps. The net-based analysis will result in aggressors switching at $t_m$. In this case, only one aggressor's timing window contained $t_m$. However, $n$ was on two critical paths, and on one of these paths (path 7 in Table 2), the arrival time of $n$ was $t$ = 3520ps. Path 7 will not be analyzed correctly using net-based analysis, since the aggressor switching time is forced to $t_m$, very different from the correct value $t$. In fact, the net-based analysis computed that the delay of path 7 changed by less than 15ps over $t_{SP}$. However, in the path-based analysis of XTalkDelay, the switching times of aggressors are set at $t$ instead of $t_m$. The net $n$ had two aggressors whose timing windows contained $t$. By setting the switching times of these two aggressors to $t$ and carrying out the analysis, the delay of path 7 was found to increase by more than 350ps over $t_{SP}$, as shown in Table 2! Another similar case was discovered on path 10. This example illustrates the inaccuracy inherent in the net-based crosstalk delay analysis (in terms of the aggressors that should switch and their switching times to model the worst-case scenario and its inability to distinguish different signal arrival times at a single net) and strengthens the case for a path-based analysis.

### 4.4 Naive vs. Smart Cell Delay Characterization

We compared the naive and smart approaches for cell delay characterization in the presence of crosstalk. Recall from Section 3 that the naive approach applies all possible input transitions to the side inputs of a cell, whereas the smart approach only applies the minimum set of vectors needed. On $D_1$, using the smart technique, the total number of HSPICE simulations (for 36 paths) was reduced from 484 (for the naive method) to 327: a reduction of 32%. The total runtime for characterization went down from 173 minutes to 109 minutes, a speed-up of 1.59. This data underscores the effectiveness of the smart approach for delay computation.

## 5 Discussion

We have developed an industrial-strength analysis tool XTalkDelay for measuring the impact of crosstalk on delays of (critical) paths in a design. The crosstalk-aware delay information is used by the designers to modify the design and prevent crosstalk. The main cornerstone of our approach, vis-a-vis other approaches, is its high delay computation accuracy. We deliberately avoided the use of approximate models for cells and nets and interconnect reductions. XTalkDelay employs a path-based approach, uses detailed and accurate distributed RC parasitics for critical nets and their aggressors, uses BSIM3-accurate gate models, and invokes HSPICE for delay computation using only the minimum complete set of input patterns. We have applied XTalkDelay to two real designs. We found that the crosstalk impact was much greater on one design $D_1$, since a significant number of critical net segments in the other design ($D_2$) had no neighboring nets (owing to previous crosstalk optimization). We also demonstrated the severe impact of crosstalk on gate delays, which underscores the need to model gate delays very accurately.

As pointed out in Section 1, most of the earlier crosstalk estimation work is pessimistic and does not meet the stringent accuracy requirements. On the other hand, an exhaustive path-based approach, though accurate, is impractical due to exponential number of paths. We believe that a hybrid two-step methodology similar to the one proposed in [1] and this paper is a viable way to solve the problem. The first step prunes the number of paths that will be passed to the second step. In the first step, either a pessimistic net-based crosstalk analysis can be used to report a superset of actual paths that may violate timing requirements, or as in XTalkDelay, simply the most critical or near-critical paths can be chosen. The second step then accurately analyzes for crosstalk effects each of the paths selected in the first step and determines the true violations. It may use an approach similar to the one proposed in this paper.

Finally, we describe the limitations of XTalkDelay and directions for fixing them in the near future.

1. XTalkDelay makes heavy use of the extraction tool StarXtract (which is invoked once for each victim net) and HSPICE for cell delay characterization. Although smart pattern generation speeds up delay characterization, delay computation and extraction are the bottlenecks in our flow. The current version of our tool is useful for analyzing up to about 150 paths. Beyond that, the run-time may become very large (depending upon the total number of nets on the selected paths). We are currently exploring faster extraction and circuit simulation techniques, for instance [14, 19]. Another solution is parallel computing. Parallelization can be carried out at various levels. Different paths can be analyzed in parallel. Or, extraction for each victim net and its associated aggressors can be done in parallel. Finally, during delay recomputation, multiple HSPICE invocations for a single stage can be done in parallel.

2. To capture the maximum impact on the victim delay, XTalkDelay assumes that if the timing window of the aggressor contains the victim arrival time, the aggressor arrival time can be made to coincide with the victim arrival time. However, this may not be possible, since the timing window computed by PrimeTime contains information only about the minimum and maximum arrival times at a gate. Storing more detailed timing information can help alleviate this problem.

3. For HSPICE simulation, the aggressor arrival time is derived from that of the victim net (for the path under consideration) as reported by PrimeTime. This is because the true victim arrival time in the presence of aggressors is not known (that is what XTalkDelay will compute). From

Table 2, we can see discrepancies between PrimeTime and HSPICE numbers. One way to fix this is as follows. If the victim arrival time as reported by PrimeTime is different from that computed by HSPICE in the presence of coupling, say by more than 5ps, the new arrival time is used to generate the aggressor waveform and the delay characterization is repeated. This fix can be expensive if the convergence is slow, in which case a limit on the maximum number of iterations may be required.

4. XTalkDelay does not check if there exists a pair of input vectors that will cause aggressors to make transitions in a direction opposite to that of the victim at a certain time. In other words, we assume that such a pair exists. Such a check can be done using ATPG (or SAT), but the signal arrival times, transition times and gate delays need to be incorporated [5].

5. Any change in the timing window of an aggressor due to coupling at its transitive fanin nets is ignored.

## References

[1] N.V. Arvind, K. A. Rajagopal, H. S. Ajith, and Suparna Das. Path Based Approach for Crosstalk Delay Analysis. In *Int. Conf. on VLSI Design*, pages 727–730, January 2004.

[2] N.V. Arvind, P.R. Suresh, V. Sivakumar, C. Pal, and D. Das. Integrated Crosstalk and Oxide Integrity Analysis in DSM Designs. In *Int. Conf. on VLSI Design*, pages 518–523, January 2001.

[3] H. B. Bakoglu. Circuits, Interconnections and Packaging. In *Addison Wesley*.

[4] W. Chen, S. Gupta, and M. Breuer. Analytic Models for Crosstalk Delay and Pulse Analysis Under Non-Ideal Inputs. In *ITC*, 1997.

[5] W. Chen, S. Gupta, and M. Breuer. Test Generation for Cross-talk Induced Faults: Framework and Computational Results. In *Asian Test Symposium*, 2000.

[6] J. Cong, D. Z. Pan, and P. V. Srinivas. Improved Crosstalk Modeling for Noise Constrained Interconnect Optimization. In *ASP-DAC*, pages 373–378, 2001.

[7] A. Devgan. Efficient Coupled Noise Estimation for On-Chip Interconnects. In *ICCAD*, November 1997.

[8] L. Ding, D. Blaauw, and P. Mazumder. Efficient Crosstalk Noise Modeling Using Aggressor and Tree Reductions. In *ICCAD*, pages 595–600, November 2002.

[9] B. Franzini, C. Forzan, D. Pandini, P. Scandolara, and A. Dal Fabbro. Crosstalk aware Static Timing Analysis: A Two-step Approach. In *Int. Symposium on Quality Electronic Design*, pages 499–503, 2000.

[10] A. B. Kahng, S. Muddu, and E. Sarto. On Switch Factor Based Analysis of Coupled RC Interconnects. In *DAC*, pages 79–84, June 2000.

[11] M. Kuhlmann, S. Sapatnekar, and K. Parhi. Efficient Crosstalk Estimation. In *ICCD*, 1999.

[12] A. Odabasioglu, M. Celik, and L. T. Pillegi. PRIMA: Passive Reduced-order Interconnect Macromodeling Algorithm. In *IEEE Transactions on Computer-Aided Design*, pages 645–654, August 1998.

[13] L. T. Pillage and R. A. Rohrer. Asymptotic Waveform Evaluation for Timing Analysis. In *IEEE Transactions on Computer-Aided Design*, pages 352–366, April 1990.

[14] V. Rajappan and S. Sapatnekar. An Efficient Algorithm for Calculating the Worst-case Delay due to Crosstalk. In *ICCD*, pages 76–81, October 2003.

[15] M. Ringe, T. Lindenkreuz, and E. Barke. Static Timing Analysis Taking Crosstalk into Account. In *DATE*, pages 639–649, May 1995.

[16] Y. Sasaki and K. Yano. Building a Crosstalk Library for Relative Window Methods - Timing Analysis that Includes Crosstalk Delay Degradation. In *Proc. of the Asia-Pacific Conference on ASIC*, August 2000.

[17] E. M. Sentovich, K. J. Singh, L. Lavagno, C. Moon, R. Murgai, A. Saldanha, H. Savoj, P. R. Stephan, R. K. Brayton, and A. Sangiovanni-Vincentelli. *SIS: A System for Sequential Circuit Synthesis*. Memorandum No. UCB/ERL M92/41, Electronics Research Laboratory, College of Engineering, University of California, Berkeley, CA 94720, May 1992.

[18] A. Vittal, L. H. Chen, M. Marek-Sadowska, K-P. Wang, and S. Yang. Crosstalk in VLSI Interconnections. In *IEEE Transactions on Computer-Aided Design*, December 1999.

[19] T. Xiao and M. Marek-Sadowska. Worst Delay Estimation in Crosstalk Aware Static Timing Analysis. In *ICCD*, October 2000.