A Timing Model Incorporating the Effect of Crosstalk on Delay and its Application to Optimal Channel Routing *

Sachin S. Sapatnekar
Department of Electrical and Computer Engineering
University of Minnesota, 200 Union Street SE, Minneapolis 55455, USA.
contact: sachin@ece.umn.edu

Abstract

Crosstalk is generally recognized as a major problem in IC design. This paper presents a novel approach to the efficient measurement of the effect of crosstalk on the delay of a net using an algorithm whose worst-case complexity is polynomial-time in the number of nets. The cost of the algorithm is seen to be $O(n\log n)$ in practice, where n is the number of nets, and it is amenable to being incorporated into the inner loop of a timing optimizer. To illustrate this, the method is applied to reduce the effects of crosstalk in channel routing, where it is seen to give an average improvement of 23% in the delay in a channel as compared to the worst case, as measured by SPICE.

1 Introduction

In recent years, crosstalk has become a major problem affecting the behavior of integrated circuits as device geometries have scaled down, bringing wires closer to each other, and switching frequencies have increased. Crosstalk can affect the behavior of circuits in two ways:

- introducing unwanted noise induced in a quiet line
- altering the delay of a switching transition

Each of these is a potentially serious hazard, and this has motivated work in the area of crosstalk analysis and crosstalk-tolerant design. Published techniques for crosstalk analysis typically work with either a very detailed and accurate analysis of the phenomenon (for example, [1]) or a very high-level model that captures the spirit, if not the details, of the crosstalk phenomenon (for example, [2–7]). The latter class of approaches has the advantage of speed over the former class, at the expense of accuracy, and has been therefore been used in the inner loop of optimizers. However, there is a need for greater accuracy without sacrificing the requirement of speed that is essential in the inner loop of an optimizer.

*This research was supported in part by the Semiconductor Research Corporation under contract 98-DJ-609 and by the National Science Foundation under award CCR-9800992.

Previous approaches that have been fast enough for this purpose have been simplistic in their approach. For example, they may measure crosstalk using a sum of their coupling lengths; these approaches do not adequately capture the delay reduction objective.

The goal of this work is to develop a technique that is intermediate to the two in accuracy and speed, and to show its application to optimal crosstalk-conscious channel routing. We will concentrate primarily here on the effect of crosstalk on the circuit delay; for methods for measuring the crosstalk noise, the reader is referred to [8,9]. The delay calculation procedure uses the Elmore delay model in the examples shown here, but the assumptions used herein are general enough that the only requirement of the delay model is that it should show an increased delay for an increased capacitance. Therefore, a higher order AWE-like delay model is equally applicable to this basic framework. The application of this approach to optimal channel routing is shown.

Recent research on determining the waveform for a set of wires that are subject to coupling effects was published in [10]. The approach provides exact waveforms through the use of waveform relaxation that capture the effect of coupling on delay. However, while the method is more accurate than the one we propose here, its computational cost is relatively high. Therefore, it is not appropriate for larger systems of interconnect wires or for situations where numerous repeated evaluations are desirable (for example, in the inner loop of an optimizer, where a quick estimate that captures the character of the delay variations due to coupling is useful, without necessarily calculating the exact waveforms).

We now summarize previous approaches to noise-conscious physical design. In [2, 3], methods for routing to minimize crosstalk in channels and switchboxes were presented. In [4], the spacing between tracks was altered to reduce the crosstalk, while in [5], the track assignment was performed with the goal of crosstalk minimization. Post-global crosstalk reduction algorithms were presented in [6, 7].

The optimization problem chosen here has the same general goal as [2], namely, to reduce the amount of crosstalk in a routed channel. The advantage of performing crosstalk estimation and reduction at this level is that since the details of the physical design are decided at this phase of the design cycle, the timing and neighborhood information of all nets is available, and consequently, accurate estimates of the timing and crosstalk may be made. Our method takes an initial routing solution that attempted to minimize the number of tracks, and modifies the solution to reduce the crosstalk-induced delay, while leaving the number of tracks in the channel unchanged. Our method differs from [2] in two ways: firstly, it permits the direct incorporation of the delay in the objective function, and secondly, instead of permuting full tracks, this approach (like [11]) allows segments of tracks (one or more individual nets) to be permuted, while maintaining the total number of tracks in the initial routing, thereby allowing a greater amount of flexibility. A simulated annealing approach is used to perform the permutations.

The important features of this work are as follows:

(a) It provides, for the first time, a procedure for determining the effect of crosstalk on delay that can be used in the inner loop of an optimizer. This is important since it implies that the procedure can directly flag critical paths that fail timing specifications due to coupling capacitance problems. The procedure has polynomial time complexity in the worst case and is experimentally seen never to be worse than $O(n\log n)$ where n is the number of nets. It does not attempt to consider the effect of crosstalk on noise; that topic is well covered by other published research.

(b) The application of this procedure to channel routing is illustrated on several examples, including Deutsch's difficult example. The procedure maintains vertical and horizontal constraints and reorders the nets to reduce the effect of crosstalk on delay. The number of tracks is maintained to be equal to the number of tracks for the optimal channel routing solution, unless otherwise desired. For the same number of tracks, average timing improvements of 23% over the worst case are shown.

Although only the application to channel routing is explicitly shown, this procedure can be used for global routing strategies. Current global routing procedures [6,7] have not directly considered the effect of crosstalk on delay. As a fast estimator, this method may be incorporated within those procedures to incorporate delay effects directly into the global routing optimization. However, this issue is not directly addressed in this paper.

The outline of this paper is as follows. Section 2.1 presents the models used for crosstalk here and presents an example to motivate the problem. Next, in Section 3, the algorithm for delay computation is proposed and its complexity is analyzed. Section 4 presents the formulation of the channel routing problem, and is followed by experimental results in Section 5, followed by concluding remarks in Section 6.

2 Modeling Crosstalk

2.1 Interconnect Modeling and Crosstalk Effects on Delay

This work models a wire as a succession of RC segments connected in series. We assume that the widths, w_i, of the wires are kept constant through the analysis and optimization. The resistance, R_i, and intrinsic capacitance, C_i, of the i^{th} segment are given by the formulæ $R_i = \alpha\ l_i/w_i$ and $C_i = \beta\ l_i\ w_i$, where l_i is the length of the i^{th} segment, and α and β are constants of proportionality for the resistance and intrinsic capacitance (including the fringing capacitance), respectively. The coupling capacitance, C_c, between two adjacent nets is proportional to overlap_i, the length along which the nets run next to each other, and is given by $C_c = \gamma\ \text{overlap}_i$, where γ is a constant of proportionality.

It is important to emphasize that the exact functional form that is used to estimate the capacitance and the delay are not important. As will be seen later, the only requirement that the delay model must satisfy is that an increase [decrease] in the coupling capacitance should be translated into an increase

[reduction] in the delay of a net; this is a rather simple requirement that any meaningful delay model would satisfy. In this work, we will use the Elmore delay model for simplicity, but we emphasize that the crosstalk estimation methodology is extendable to any arbitrary delay model that satisfies the above requirements.

The role of the coupling capacitances is greatly dependent on the relative switching times of the nets [12]. One of three situations is possible, as illustrated in Figure 1[1]

- If one net switches and the other remains inactive, then the equivalent coupling capacitance between the two is modeled as C_c.
- If both nets switch at the same time in opposite directions (i.e., one switches from 1 to 0, and the other from 0 to 1), then the equivalent coupling capacitance is modeled as $2C_c$.
- If both nets switch at the same time in the same direction, then the equivalent coupling capacitance is modeled as zero.

The complexity of this relationship arises from the interrelationships between the timing behavior and the coupling capacitance. The value of the equivalent coupling capacitance is affected by the switching time, which, in turn, is affected by the value of the coupling capacitance.

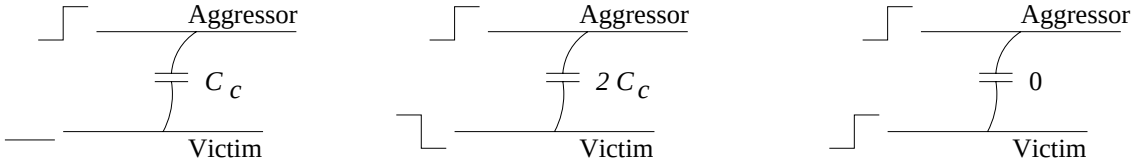

Figure 1: Effect of switching times on coupling capacitance.

To elaborate on this, consider two wires that are laid out adjacent to each other. If the input signals to driver of the two wires switch between times $[T_{min,1}, T_{max,1}]$ and $[T_{min,2}, T_{max,2}]$, respectively, and if the delays required to propagate the signal along the wires are in the range $[d_{1,min}, d_{1,max}]$ and $[d_{2,min}, d_{2,max}]$, respectively, then the intervals during which the lines switch are $[T_{min,1}+d_{1,min}, T_{max,1}+d_{1,max}]$ and $[T_{min,2}+d_{2,min}, T_{max,2}+d_{2,max}]$, respectively. Therefore, the following relationship holds between the switching times and the equivalent coupling capacitance, $C_{c,eq}$.

The value of $C_{c,eq}$ in the first line of Table 1 is chosen to be either 0 or $2C_c$, depending on whether the signals switch in the same direction, or in opposite directions. Note that it is possible for some of the above intervals to be empty when the lower bound and the upper bound of the interval coincide.

[1]We point out that the figure is meant to emphasize a point and should not be taken too literally. In particular, the capacitance of 0, C_c or $2C_c$ is, in reality, modeled as a capacitance to ground rather than a capacitance between the two lines.

Table 1: Variation of $C_{c,eq}$ with switching time.

Interval	$C_{c,eq}$
$[\max\{T_{min,1}+d_{1,min}, T_{min,2}+d_{2,min}\}, \min\{T_{max,1}+d_{1,max}, T_{max,2}+d_{2,max}\}]$	0 or $2C_c$
$[\min\{T_{min,1}+d_{1,min}, T_{min,2}+d_{2,min}\}, \max\{T_{min,1}+d_{1,min}, T_{min,2}+d_{2,min}\}]$	C_c
$[\min\{T_{max,1}+d_{1,max}, T_{max,2}+d_{2,max}\}, \max\{T_{max,1}+d_{1,max}, T_{max,2}+d_{2,max}\}]$	C_c

While the relationship shown in Table 1 looks relatively straightforward, it is considerably complicated by the fact that $d_{1,min}$, $d_{1,max}$, $d_{2,min}$ and $d_{2,max}$ are dependent on the value of $C_{c,eq}$, which is itself dependent on the values of $d_{i,min}$ and $d_{i,max}$, $i = 1, 2$. Therefore, an iterative approach is required.

It should be pointed out that the $0 - C_c - 2C_c$ model has some limitations. The work of [13] showed that a capacitance of 0 is not a strict lower bound, and likewise, $2C_c$ is not a strict upper bound on the effective capacitance. In such a case, if a lower bound and upper bound capacitance can be arrived at (including a negative lower bound) *a priori*, the techniques described here can be used to correctly determine the switching intervals (we do not provide a technique for determining these bounds *a priori* in this work).

2.2 Illustrative Example

The relation between crosstalk and timing is illustrated by the simplified three-wire example in Figure 2. The details of our calculations are described in Appendix 6, but the salient assumptions and conclusions are shown here. We assume interconnect parameters in accordance with [14], and assume that the drivers a, b and c with resistances of 2KΩ, 3KΩ and 1KΩ, respectively, and that their inputs switch at times that lie in some specified time intervals [2]. These times are assumed to be as follows:

- driver 1 switches in the interval [0.25ns,1.0ns]
- driver 2 switches in the interval [0.1ns,0.2ns]
- driver 3 switches at 0ns

For ease of description, we will assume equal rise and fall times. We point out, though, that the methods described in this paper do not require equal rise and fall times and can be extended to unequal values using standard methods in timing analysis (see, for example, [15]).

On the surface, it would appear that none of the switching time intervals overlap, and an equivalent coupling capacitance of C_c would prevail, based on Table 1. However, these switching intervals do not take the wire delay into account, and hence we will now make that correction.

[2] Variations in the switching times may occur for various reasons such as the existence of multiple paths passing through the gate with different delays.

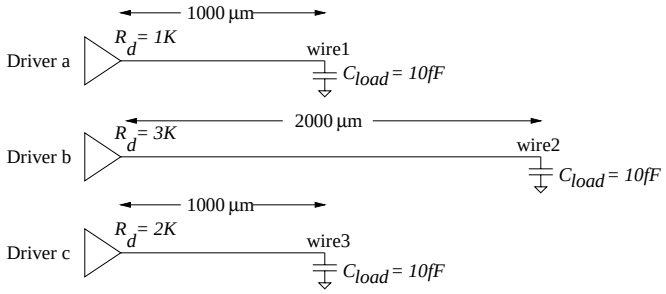

Figure 2: An example showing the effect of crosstalk on timing.

Let us, only for a moment, neglect the coupling capacitance. The switching time of wires 1, 2 and 3 considering the effects of their self-capacitance (i.e., area and fringing capacitance), and ignoring the effects of coupling capacitance entirely, may be calculated from the Elmore delay formula to be [0.3309ns,1.0809ns], [0.3432ns,0.4432ns], and [0.1609ns,0.1609ns], respectively (note that the last interval is a single point). Therefore, it is clear that the overlaps in the timing intervals at the driver inputs can be misleading and do not show the complete picture. Moreover, the effects of the coupling capacitance are yet to be incorporated, and the calculation of the switching intervals while incorporating their effects is quite involved.

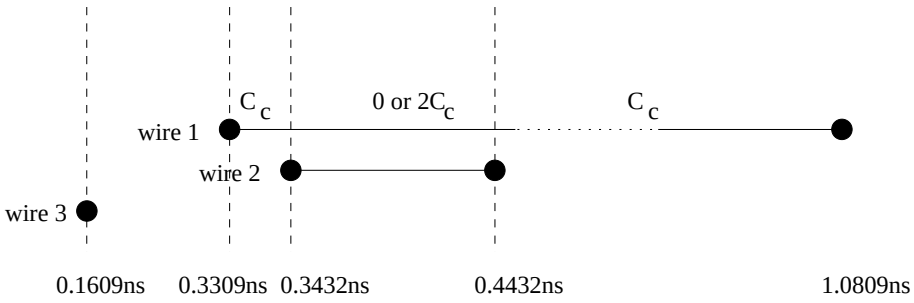

Figure 3: Illustration of the procedure for calculating switching intervals for a system of interconnects.

The intervals calculated above are illustrated in Figure 3. Consider the switching of wire 1. A switching event at any time in the interval [0.3309ns,0.3432ns] corresponds to a coupling capacitance of C_c, implying that incorporation of coupling capacitance effects would update these switching times to the interval [0.4016ns,0.4139ns]. An event in the interval [0.3432ns,0.4432ns] corresponds to a best-case coupling capacitance of 0; therefore, no correction in the earliest switching time due to the coupling capacitance is required. Consequently, the earliest switching event occurs at time 0.3432ns, assuming that the switching intervals for wire 2 have been correctly calculated. However, that is an invalid assumption, as wire 2 has a minimum coupling capacitance of C_c with wire 3, requiring its value to be corrected, leading to the calculation of a new earliest switching time for wire 1, and so on.

After several iterations, the final switching intervals for wires 1 through 3 are calculated as [0.4016ns,1.2223ns], [0.5539ns, 1.0753ns] and [0.3016ns,0.3016ns], respectively; for details, the reader is referred to Appendix 6.

The objective of this example was to help the reader appreciate the difficulty of the issue of calculating these switching intervals, and to motivate the need for a precise, efficient and systematic algorithm for the purpose, which is presented in Section 3 and proven to have polynomial time complexity. We will also point out here that the order in which the switching intervals were updated affects the number of iterations required to find these values.

This example illustrates the following points. Firstly, an iterative approach is required. Secondly, different switching times for a wire may correspond to different equivalent coupling capacitances, and a uniform value for the entire switching duration is not valid; this is illustrated by the update to wire 1 in Iteration 1 above. Thirdly, the order in which the updates are made is important for convergence. In the above example, if the updates were carried out in an order that processes wire 2 before wire 1, then the number of iterations would be brought down from two (see Appendix 6 for details) to one.

The algorithm proposed in this paper attempts to find such an order, and determines the number of computations required by the iterative procedure in the worst case. For this specific example, our algorithm completes the computation in a single iteration since the heuristic in Section 3.3 will process wire 2 before wire 1.

3 An Algorithm for Correct Crosstalk Estimation

The algorithm is described here in the context of a set of nets $N_1, \cdots, N_k$ in a channel. We will assume that the channel is positioned with its length along the x axis and its height along the y axis.

We define a *spatial adjacency graph*, G_s, whose vertices correspond to the k nets in the channel. An edge is drawn between vertices i and j if the horizontal spans of nets N_i and N_j intersect. If two nodes are connected by an edge on G_s, the corresponding nets will affect each other by means of a coupling capacitance if they are placed on adjacent tracks.

The assumption that is made in this work is that all transitions are sharp transitions that occur at a time given by the delay; we happen to use the Elmore delay model here, but the basic approach may be extended to other models.

3.1 Outline of the Algorithm

The input to the algorithm is a channel routing solution that is found without regard to crosstalk, using a standard channel router [16, 17], which provides the adjacency information required for the analysis. For each driver, a switching interval $[T_{min}, T_{max}]$ signifying the range of switching times at the input of the driver, and a source resistance, R_d, are specified. If the wire originates at a gate at the top or bottom of the channel, these quantities simply correspond to the range of switching times and the driver resistance of that gate. If the wire originates at the left or right of the channel, then R_d

corresponds to the upstream resistance. The specification of the range of switching times corresponds to the range of switching times of the driver of the net, plus the Elmore delay of the net assuming that it terminates at the left edge of the channel; this is justified by the separable structure of the Elmore delay computation[3].

The goal of the algorithm is to incorporate the information in the G_s graph and the adjacency information derived from the channel routing solution to arrive at a range $[T_{start}, T_{end}]$ for all of the wires in the channel.

We define the self-delay, d_s, of a line as its RC delay calculated by considering only the intrinsic capacitance of the line. Note that the self-delay is calculated without incorporating the effects of coupling capacitance; consideration of the coupling capacitance can only cause the delay to increase, and hence the self-delay is a lower bound on the delay of the line. The task of this algorithm is to determine whether the correction due to coupling capacitance should assume a capacitance of C_c or $2C_c$ [0 or C_c] for the maximum [minimum] switching time. Let delay(C_c) be the delay on the line due to the coupling capacitance of C_c for each neighbor of a given wire (note that the value of C_c for each wire will be different, and this is only a notational convenience). The initial switching interval is set to the value of $[T_{start}, T_{end}]$, where $T_{start} = T_{min} + d_s$ and $T_{end} = T_{max} + d_s +$ delay(C_c), both of which are clearly lower bounds on the earliest and latest switching times for the wire. The pseudocode below shows how these can be refined to arrive at the actual earliest and latest switching times.

```
ALGORITHM Update_Switching_Times
1.   For each net {
2.      calculate its d_s and delay(C_c) and update its T_start and T_end value
3.   }
        /* OUTER LOOP */
4.   Repeat {
        /* FORWARD PASS
           Update the latest switching time for each net using
           C_c,eq = C_c or 2 C_c, as appropriate
        */
5.      Repeat {
6.         For each net i {
7.            For each neighbor j of i in G_s
8.               Update T_end for i
9.            For each neighbor j of i in G_s
```

[3]Several bottom-up methods [18,19] have used the approach representing the delay as a sum of the downstream Elmore delay and the delay of the upstream net, assuming the downstream nodes to be represented by a capacitance. This method is the analog of that approach for a top-down computation. For an RC tree with an upstream resistance of $R_{upstream}$ and an upstream delay of $D_{upstream}$, the Elmore delay at a node in a downstream tree can be computed as $T_d = D_{upstream} + R_{upstream}C_{tree} + T_{d,tree}$, where C_{tree} is the total capacitance of the tree and $T_{d,tree}$ is the Elmore delay from the root of the tree to the node of interest.

```
10.              Update T_end for j
11.          }
12.      } until (no T_end changes)
         /* BACKWARD PASS
            Update the earliest switching time for each net using
            C_c,eq = 0 or C_c, as appropriate
         */
13.      Repeat {
14.          For each net i {
15.              For each neighbor j of i in G_s
16.                  Update T_start for i
17.              For each neighbor j of i in G_s
18.                  Update T_start for j
19.          }
20.      } until (no T_start changes)
21.  } until (no T_end or T_start changes)
```

In practice, the changes in the forward and backward passes are only made for neighbors of nets that were altered in the previous iteration, except in the first iteration of the outer loop, where all nets are processed.

The neighbors of a wire j above correspond to adjacent vertices in the G_s graph. The updates in lines 8, 10, 16 and 18 are performed using the scheme in Table 1, with the difference that the wire delays are calculated using the values of $C_{c,eq}$ based on the current values of T_{start} and T_{end} for the nets. The update formulæ are as follows:

T_{end} updates

- If $T_{end}(j) > T_{end}(i) > T_{start}(j)$, as in Figure 4(a), then the worst case corresponds to an equivalent coupling capacitance of $2C_c$ between wires i and j that is seen at $T_{end}(i)$, resulting in the update shown by the dotted line. Mathematically, we state this using the formula $T_{end}(i) = \text{Update}(T_{end}(i), 2C_c)$ where the right hand side implies that T_{end} is updated so that $C_{c,eq}$ between i and j is set to $2C_c$.

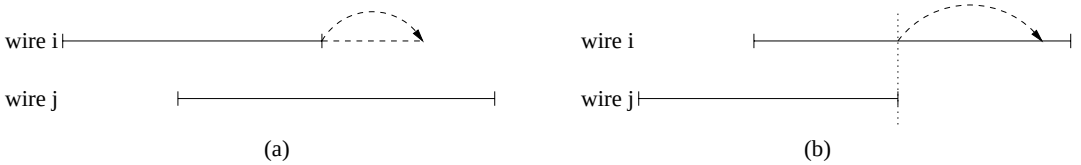

Figure 4: Updating the value of T_{end}.

- If $T_{end}(i) > T_{end}(j) > T_{start}(i)$ as in Figure 4(b), then the latest concurrent switching activity occurs at $T_{end}(j)$, where a coupling capacitance of $2C_c$ is seen by wire i between itself and

wire j. This results in a potential update shown by the dotted arrow in the figure. The value of $T_{end}(i)$ is updated only if this value exceeds the current value; in the situation shown in the figure, no update is necessary. Mathematically, we write the update formula as $T_{end}(i) = \max[\text{Update}(T_{end}(j), 2C_c), T_{end}(i)]$.

- If the two intervals do not overlap spatially, T_{end} corresponds to an effective coupling capacitance of C_c, and $T_{end}(i) = \text{Update}(T_{end}(i), C_c)$ and $T_{end}(j) = \text{Update}(T_{end}(j), C_c)$.

T_{start} updates

The updates for T_{start} may be justified similarly and are listed below.

- If $T_{start}(j) > T_{start}(i)$, then update $T_{start}(i) = \min[\text{Update}(T_{start}(i), C_c), T_{start}(j)]$.
- If $T_{start}(i) > T_{end}(j)$, then update $T_{start}(i) = \text{Update}(T_{start}(i), C_c)$.
- If $T_{end}(j) > T_{start}(i) > T_{start}(j)$, then $T_{start}(i)$ is left unchanged and corresponds to a coupling capacitance of zero.

The updates in lines 8 and 10 (and similarly, in lines 16 and 18) are performed in separate loops so that the value of T_{end} of net i in the current iteration is fully calculated before its impact on its neighbors is determined. This removes the need for unnecessary repeated applications of the update formulæ.

3.2 Theoretical Results and Complexity

Theorem 1: The iterative procedure in Algorithm Update_Switching_Times converges.

Proof: In the first iteration of the outer loop, at the end of the forward pass loop, the values of T_{end} are no smaller than they were before the pass. This is due to the fact that the coupling capacitance was taken to be C_c before beginning, and during the forward pass, some of these are updated to $2C_c$, with a consequent increase in T_{end}. Similarly, the value of T_{start} is always larger on completion of the backward pass in the first iteration of the outer loop, since some of the coupling capacitances are updated from 0 to C_c.

In the second iteration of the forward pass, the values of T_{end} are updated to reflect any altered circumstances due to overlaps that were either introduced or made absent after the preceding backward pass. Since the first backward pass kept T_{end} unaltered and only increased T_{start}, it follows that the span of each switching interval could only be diminished, and not increased during the backward pass. Therefore, it is not possible for any new overlaps to be introduced, and consequently, any updates during the second forward pass must be due to the fact that some overlaps were removed during the first backward pass. The effect of a removed overlap is that the worst-case equivalent coupling capacitance is reduced from $2C_c$ to C_c, and therefore, the updated value of T_{end} must be reduced in the second iteration. Similarly, it can be argued that since the second forward pass diminishes the overlaps, the value of T_{start} must be increased by the second forward pass.

In subsequent iterations, the T_{start} are either increased or kept constant, and the T_{end} values are either reduced or kept constant. For n nets, since the number of possible configurations is finite ($< n \cdot 3^n$, corresponding to each net having an equivalent coupling capacitance of $0, C_c$ or $2C_c$ with each other net), and since the reduction is monotone, the procedure must converge. In practice, the procedure converges much faster than $n \cdot 3^n$ steps since many of the possible configurations are eliminated by the monotone path taken by the algorithm, as illustrated in the next theorem. □

Theorem 2: The computational complexity of the algorithm is $O(mn^2)$, where n is the number of nets and $m < n$ is the maximum number of nets that are spatially adjacent to any net. Therefore, assuming that m is bounded by a constant, the complexity of the procedure is $O(n^2)$.

Comment: *In practice, this upper bound was never seen to be reached.*

Proof: We will consider the case of the forward pass in which T_{end} is updated; the argument for T_{start} is symmetric.

In the first iteration of the outer loop of Algorithm Update_Switching_Times, there are two ways in which T_{end} may be updated, corresponding to the first two bullet items under "T_{end} updates" in Section 3.1; by construction, the third is never activated in the first iteration and may only occur in subsequent iterations. We will refer to these two types of updates as "updates at T_{max}" and "updates before T_{max}," respectively.

For any given net, if the T_{max} value of a neighbor is updated, it could potentially update the T_{max} value of the given net. Once an "update at T_{max}" is made by a neighbor, no further updates that can be made by that neighbor during the current iteration, since the T_{max} values can only increase during a forward pass iteration (as shown in the proof of Theorem 1), meaning that no overlaps are removed during the execution of the loop. However, an "update before T_{max}" may result in multiple updates in the iterations of the forward pass loop, with each update corresponding to an update on the T_{max} of some neighbor.

We observe that each update to the T_{max} value of a net must be initiated by an update to the T_{max} value of some other net. Moreover, in every iteration, there must be at least one update at T_{max} since in each switching pair, there must be one net that switches first, and its effect could ripple to all of the other nets. Therefore, the forward pass loop can have no more than $O(n)$ iterations, implying that the total number of updates can be no more than $O(n^2)$ in the first iteration of the outer loop.

In subsequent iterations of the outer loop, the forward pass will only update a T_{max} value if an overlap is removed. Since each net can overlap with at most all of its m neighbors, there can be no more than $O(mn)$ overlaps that could be removed, implying that the total number of such updates can be no more than $O(mn^2)$. This implies that the overall complexity is $O(mn^2)$. □

The theorem above lists the worst-case time complexity of the procedure, corresponding to the most pathological case where every update to every net affects every other net. However, this is extremely

unlikely in practice, and with the use of heuristics (to be described in Section 3.3), the number of updates can be restricted to a complexity that is practically of the form $O(n)$. In our experiments, the number of iterations of the outer loop of Algorithm Update_Switching_Times never exceeded four and therefore, we found that the number of updates was linear in the number of nets. This ordering necessitated a sorting procedure, and therefore the complexity of the entire procedure is $O(n\log n)$.

As a parenthetical note, the character of the updates can be seen to be similar in character to those for the Bellman-Ford algorithm [21], where the neighbors of a node are first updated, followed by the neighbors of these neighbors, and so on. The difference is that the weights on the edges of the timing graph that could be drawn here are liable to change, depending on the presence or absence of overlaps, making the algorithm more complex.

3.3 Heuristics for Speeding up the Procedure

The order in which the nets are processed is important in ensuring that the switching intervals are calculated efficiently. We will illustrate this with respect to the backward pass of Algorithm Update_Switching_Times, noting that the argument is similar for the forward pass loop.

We first note that for the backward pass loop of lines 13–20, the iterations are similar to Gauss-Seidel updates, where all updates in the current iteration are taken into account while processing a net, rather than a Gauss-Jacobi iteration, where the values from the previous iteration would be frozen in place and used in the current iteration. Therefore, while processing the k^{th} net in the first forward pass, the updated T_{start} values for the first $k-1$ nets are being used.

If, in some iteration of the loop on lines 14–19, a net n_x is updated, then each neighbor of n_x is processed. The value of T_{start} of this neighbor is dependent on the values of T_{start} and T_{end} of each of its neighbors (including n_x) in the following ways:

- Due to the monotone shrinking of the switching intervals, the T_{end} value of each neighbor can affect the T_{start} of a net precisely once: when the value of T_{end} is such that a temporal overlap ceases to exist, the effective coupling capacitance for T_{start} becomes C_c instead of 0.

- A change in the T_{start} value of a net can update the T_{start} value of each neighbor according to the update formulæpreviously described. This update can occur more than once if a poor ordering is chosen, and the alignment of the timing windows for the nets (and the planets) is such that a pathological case is excited. The computation in the procedure can be reduced by heuristically choosing a good ordering.

Our heuristic updates the nets in descending order of the value of T_{start} at the beginning of the procedure. This is based on the fact that since T_{start} is guaranteed to be nondecreasing and as a result, when a net with a lower value of T_{start} is updated, it is likely not to be limited by the T_{start} values of its neighbors; if they had larger T_{start} values to begin with, they would have been updated already, and

if they had smaller T_{start} values, then their values are irrelevant as the update depends on the T_{start} value of the current net. The cost associated with performing the sorting procedure is $O(n\log n)$.

Similarly, it can be argued that for the forward pass, nets should be processed in increasing order of their T_{end} values. However, it should be noted that this is only a heuristic, and does not *guarantee* a single pass through the repeat loop; in fact, it is easy to derive examples where the application of this method would require more than one pass of the repeat loop. For instance, consider the situation in Figure 5, where the solid lines show the initial time spans, $[T_{start}, T_{end}]$, for switching events of three wires that have a spatial overlap. According to the heuristic, the value of T_{end} for wires a and b will first be updated during the forward pass, as shown by the dotted lines a-b, as the T_{end} value of wire b plus the delay due to coupling. However, when wire b is processed, it is seen that the T_{end} values of wires b and c are updated due to wire c, which necessitates another update to the T_{end} of wire a, shown by the dotted line a-(b-c), since the T_{end} value for b that was used earlier was incorrect.

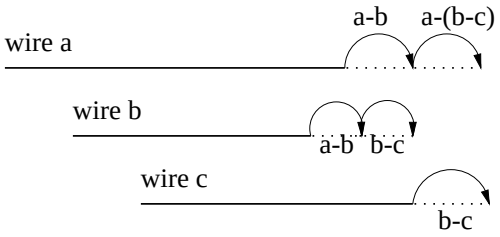

Figure 5: An example showing that the left-edge ordering is heuristic and not optimal.

4 The Channel Routing Problem

4.1 Introduction

The channel routing problem is to determine an assignments of nets to tracks in the channel with the aim of satisfying one or multiple objectives. The most commonly used objective in the past has been to minimize the number of tracks in the channel. The locations of pins on the top and bottom of the channel are fixed, and the nets are required to connect two or more pins at either end of the channel. In the final routing solution, all nets are required to satisfy two types of constraints [16]:
(1) *horizontal constraints*, which imply that two nets whose horizontal spans overlap must not occupy the same track, and
(2) *vertical constraints*, which imply that a net that is connected to a pin at the top of the channel must lie above another net that is connected to a pin at the bottom of the channel, in the same column.

The process of exchanging tracks in a routed channel can reduce the crosstalk in a channel. In the simple example in Figure 6(a), if the first two tracks are exchanged, as shown in Figure 6(b), the crosstalk in the channel would be "reduced"; the procedure in [2] would produce such a solution[4]. However, if the

[4] We point out that like [11] and unlike [2], our implementation does not restrict itself to exchanging tracks, but also

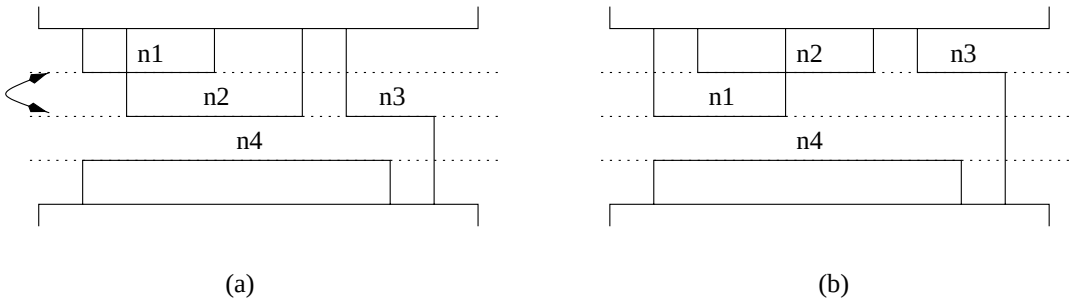

Figure 6: Two permuted channel routing solutions.

focus is on timing-critical nets, and if net n1 in the uppermost track of the initial routing is the most timing critical, it may be better to leave it in its current position, as against moving it to the second track, where it would have crosstalk interactions with a larger number of nets.

4.2 Optimized Channel Routing for Reduced Crosstalk

The algorithm for optimizing the channel routing solution for crosstalk effects uses a simulated annealing engine. The simulated annealing algorithm [22] is a well-known procedure and we will only outline the salient features of the method.

The **cost function** is chosen to be a weighted sum of the maximum delay of each net; in our implementation, all weights were chosen to be 1, but these may be adjusted appropriately to assign a larger weight for more critical nets, if desired, or any alternative cost function. The calculation of T_{end} proceeds according to the algorithm described in Section 3.

A **move** consists of an exchange of a set of nets between two tracks. These nets are chosen so that they are contiguous within the track, and the number of such contiguous nets is chosen randomly. For example, in Figure 6(a), a couple of possible moves are (see footnote 4):

(a) moving net n2 to the first track and n1 to the second track

(b) moving nets n2 and n3 to the first track and n1 to the second track

An example of an unallowable move is exchanging the positions of nets n1 and n4, since this would violate a vertical constraint. All moves are performed in such a way that the feasibility of the routing solution is maintained. In other words, no move is permitted to violate a horizontal or a vertical constraint. Moreover, the number of tracks in the routing solution is maintained. Therefore, this method may be used as a fine-tuning step after the height of the channel has been minimized.

The simulated annealing procedure proceeds according to a cooling schedule for the temperature. At each temperature, a number of moves are attempted, with cost-reducing moves being accepted and cost-increasing move being accepted probabilistically according to the Metropolis function.

exchanges subsets of the nets in a pair of tracks, if permissible under the vertical constraints

5 Experimental Results

The algorithm to minimize the objective function by reordering, subject to horizontal and vertical constraints, was implemented in C and executed on a Sparc Ultra 1/170 workstation. In our implementation, we assumed that the rise times are equal to the fall times, but this is not essential, and the procedure can be extended easily to handle rise and fall transitions separately.

A summary of the results is shown in Table 2 for 0.25μm technology parameters. The algorithm was used to reorder eight different examples, keeping the number of tracks the same as that in the original solution that was obtained from a Yoshimura and Kuh channel router [16] that optimizes the height of the channel. The eight examples are taken from [16], with the last two examples being the routing of the Deutsch difficult example without and with doglegs, respectively.

The second column of Table 2 shows the number of nets for each example. The third column shows the improvement in the objective function at the end of the simulated annealing run, as compared to the objective function value in the original channel. The CPU times for the run are shown in the next column.

The optimization was carried out on the basis of the Elmore delay model, modeling the driver as a linear resistor. Due to the well-known deficiencies of the Elmore model and the limitations of the linear resistor model for a driver, we validated the solution using SPICE, with a 0.25μm BSIM3 model for the drivers and wired appropriately modeled using coupling capacitances and capacitances to ground. The improvement provided by the final solution over the initial solution according to this model is shown in the next-to-last column of Table 2. It is seen that our optimizer provides improvements in each case, and can give improvements of over 34%. It is expected that the essence of this approach can be used to obtain even larger improvements for longer wires, by optimization over multiple channels or routing regions.

To obtain an idea of how much the optimal solution differs from the worst solution, the simulated annealing algorithm was executed again, this time with the objective of *maximizing* the objective function. At the end of this run, we have a reordered channel where the effects of crosstalk correspond to the worst possible scenario. The difference between this objective function value and the objective function value obtained earlier provides an idea of how much improvement is possible between the most optimal and the least optimal channel routing solution. Note that both of these solutions are valid solutions with the same number of tracks, and it is quite possible for a CAD tool that is not crosstalk-conscious to come up with the worst-case solution. The last column of Table 2 shows the improvement provided by the result of our technique over this worst-case solution, with the numbers corresponding to the results of SPICE simulations. These figures make the case in favor of the use of crosstalk-conscious criteria in routing.

Our claim of a linear number of updates in practice is validated by the fact that the number of times that the outer and inner loops of `Algorithm Update_Switching_Times` are invoked is bounded

Table 2: Results of Channel Reordering on Timing

	Number of nets	Improvement over initial (our metric)	CPU Time	Improvement over initial (SPICE)	Improvement over worst case (SPICE)
yk1	21	20.9%	9s	9.7%	15.6%
yk3a	45	12.2%	22s	3.9%	7.1%
yk3b	47	14.4%	68s	5.9%	14.0%
yk3c	54	12.5%	66s	4.2%	10.6%
yk4b	54	17.4%	87s	11.4%	17.3%
yk5	60	28.1%	101s	20.6%	36.6%
Deutsch1	72	35.2%	124s	34.6%	74.7%
Deutsch2	72	8.8%	201s	7.0%	9.6%

by a small constant, for all of the circuits that we tried. Since the inner loops have $O(n)$ complexity, the complexity is, in practice, dominated by the $O(n\log n)$ sorting process for the nets required by the ordering heuristic in Section 3.3. For larger systems, a more approximate sorting procedure may be used to ease this bottleneck; in this work, the run times were small enough that we did not need to resort to this.

The effect of utilizing additional tracks to reduce the crosstalk is shown in Figure 7. All numbers in this figure are calculated from the SPICE validation procedure described above. As expected, the cost function reduces with the addition of more tracks. Note that in Table 2, Deutsch1 shows larger improvements than Deutsch2 since it uses a larger number of tracks and has greater flexibility in reordering for crosstalk reduction. The graph shows that as more flexibility is permitted to Deutsch2 by increasing the number of tracks, significantly larger delay reductions are possible.

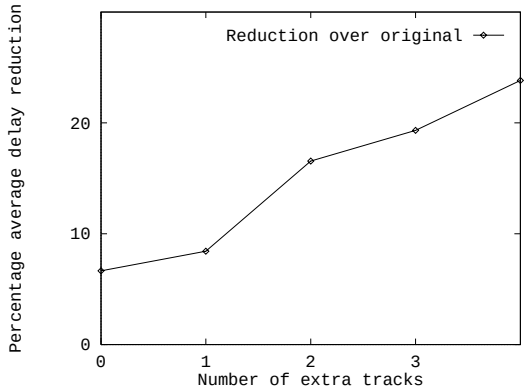

Figure 7: Effect of increasing the number of tracks on example Deutsch2.

6 Conclusion

A new provably polynomial time iterative procedure for determining the effect of crosstalk on delay has been proposed. From the proof of Theorem 1, it can be seen that it is applicable under any delay model where an increase in the effective coupling capacitance causes an increase in the delay, and vice versa. This is a property satisfied by any reasonable delay model. The method was applied to reduce crosstalk in channel routing, and the results were demonstrated to give visible improvements. It is anticipated that this method will be useful in other applications for crosstalk optimization.

With regard to future work, it may be possible to adapt this work to a full-chip noise analysis scenario, where a change in the switching time can impact the arrival time at the inputs of other gates in the circuit, and a ripple effect is possible. Starting at the inputs, the wires can be processed in a PERT-like fashion, using current values of arrival times to determine the effective coupling capacitance, continuing until convergence.

It should be pointed out that noise reduction and delay reduction are correlated objectives, in that both can be reduced by reducing the distance along which two simultaneously switching wires run adjacently. The objective of this work has been to provide a technique that directly measures the effect of crosstalk on delay. It is expected that it could be used in conjunction with noise metrics to simultaneously satisfy requirements on delay and noise.

Acknowledgements

The author would like to thank the anonymous reviewers for their suggestions.

References

[1] D. H. Xie and M. Nakhla, "Delay and crosstalk simulation of high-speed VLSI interconnects with nonlinear terminations," *IEEE Transactions on Computer-Aided Design*, vol. 12, pp. 1798–1811, Nov. 1993.

[2] T. Gao and C. L. Liu, "Minimum crosstalk channel routing," in *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, pp. 692–696, 1993.

[3] T. Gao and C. L. Liu, "Minimum crosstalk switchbox routing," in *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, pp. 610–615, 1994.

[4] K. Chaudhary, A. Onozawa, and E. S. Kuh, "A spacing algorithm for performance enhancement and cross-talk reduction," in *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, pp. 697–702, 1993.

[5] D. A. Kirkpatrick and A. L. Sangiovanni-Vincentelli, "Techniques for crosstalk avoidance in the physical design of high-performance digital systems," in *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, pp. 616–619, 1994.

[6] T. Xue, E. S. Kuh, and D. Wang, "Post global routing crosstalk risk estimation and reduction," in *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, pp. 302–309, 1996.

[7] H. Zhou and D. F. Wong, "Global routing with crosstalk constraints," in *Proceedings of the ACM/IEEE Design Automation Conference*, pp. 374–377, 1998.

[8] A. Devgan, "Efficient coupled noise estimation for on-chip interconnects," in *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, pp. 147–151, 1997.

[9] K. Shepard, V. Narayanan, P. C. Elmendorf, and G. Zheng, "GlobalHarmony: Coupled noise analysis for full-chip RC interconnect networks," in *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, pp. 139–146, 1997.

[10] P. D. Gross, R. Arunachalam, K. Rajagopal, and L. T. Pileggi, "Determination of worst-case aggressor alignment for delay calculation," in *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, pp. 212–219, 1998.

[11] K.-S. Jhang, S. Ha, and C. S. Jhon, "COP: A Crosstalk OPtimizer for gridded channel routing," *IEEE Transactions on Computer-Aided Design*, vol. 15, pp. 424–429, Apr. 1996.

[12] L. Gal, "On-chip cross talk - the new signal integrity challenge," in *Proceedings of the IEEE Custom Integrated Circuits Conference*, pp. 251–254, 1995.

[13] F. Dartu and L. T. Pileggi, "Calculating worst-case gate delays due to dominant capacitance coupling," in *Proceedings of the ACM/IEEE Design Automation Conference*, pp. 46–51, 1997.

[14] J. Cong, "Challenges and opportunities for design innovations in nanometer technologies," tech. rep., Semiconductor Research Corporation, Research Triangle Park, NC, 1997. (available at `http://www.src.org/cgi-bin/deliver.cgi/p_s98005.pdf?/pubs/opendocs/p_s98005.pdf`).

[15] S. S. Sapatnekar and S. M. Kang, *Design Automation for Timing-Driven Layout Synthesis*. Norwell, MA: Kluwer Academic Publishers, 1993.

[16] T. Yoshimura and E. S. Kuh, "Efficient algorithms for channel routing," *IEEE Transactions on Computer-Aided Design*, vol. CAD-1, pp. 25–35, Jan. 1982.

[17] N. Sherwani, *Algorithms for VLSI Physical Design Automation*. Norwell, MA: Kluwer Academic Publishers, 1995.

[18] J. Lillis, C. K. Cheng, and T.-T. Lin, "Optimal wire sizing and buffer insertion for low power and a generalized delay model," in *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, pp. 138–143, 1995.

[19] L. P. P. van Ginneken, "Buffer placement in distributed RC-tree networks for minimal Elmore delay," in *Proceedings of the IEEE International Symposium on Circuits and Systems*, pp. 865–867, 1990.

[20] A. Tucker, *Applied Combinatorics*. New York, NY: Wiley and Sons, 2nd ed., 1984.

[21] T. H. Cormen, C. E. Leiserson, and R. L. Rivest, *Introduction to Algorithms*. New York, New York: McGraw-Hill Book Company, 1990.

[22] S. Kirkpatrick, C. Gelatt, Jr., and M. Vecchi, "Optimization by simulated annealing," *Science*, vol. 220, pp. 671–680, May 1983.

Appendix A: Detailed Calculations for the Illustrative Example of Section 2.2

This section presents the detailed calculations associated with the illustrative example of the circuit in Figure 2. The drivers are modeled as linear resistors with resistances as shown in the picture, and the wires have a resistance, self-capacitance (area+fringing capacitance) and coupling capacitance of 0.02 Ω/μm, 0.07 fF/μm and 0.07 fF/μm, respectively, in accordance with the numbers in [14]. Each wire is represented by a π-model with segments of length 1000 μm. The inputs to the drivers of wire 1, wire 2 and wire 3 switch in the intervals [0.25ns,1.0ns], [0.1ns,0.2ns] and [0.0ns,0.0ns], respectively; note that the last is a single point.

If we consider only the self-capacitance of the wire, then each wire would incur an additional delay of $R_{driver} \times (C_{wire} + C_{load}) + D_{wire}$ under the Elmore model, where R_{driver} is the resistance of the driver, C_{wire} and C_{load} are the capacitances of the wire and at the load, respectively, and D_{wire} is the Elmore delay of the wire. For wire 1, this corresponds to 1K$\times$(70fF + 10fF) + 20 $\times$ (35fF + 10fF) = 0.0809ns. This updates the switching interval for wire 1 to [0.3309ns,1.0809ns]. Similarly, the effect of the self-capacitance on wires 2 and 3 would update their switching intervals to [0.3432ns,0.4432ns] and [0.1609ns,0.1609ns], respectively.

The effect of coupling capacitance must now be considered, and this is done iteratively since we do not know *a priori* whether an effective coupling capacitance of $0, C_c$ or $2C_c$ should be considered:

Iteration 1 : For wire 1, as described in Section 2.2 with the aid of Figure 3, the earliest switching time is updated to 0.3432ns. Similarly, the latest switching time corresponds to time 1.0809ns, where a coupling capacitance of C_c is seen since there is no simultaneous switching (based on the currently calculated intervals) with the neighboring wire. This updates the switching times to the interval [0.3432ns,1.2223ns].

For wire 2, the earliest switching time at 0.3432ns must be updated since there is no simultaneous switching with wire 3 (see Figure 3), and this results in effective coupling capacitances of 0 with wire 1 and C_c with wire 3. This updates the earliest switching time to 0.5539ns. Similarly, the latest switching time at 0.4432ns is updated since it experiences an effective coupling capacitance of $2C_c$ with wire 1 and C_c with wire 2, resulting in a revised switching interval of $[0.5539ns, 1.0753ns]$.

Wire 3 sees an effective coupling capacitance of C_c with wire 2, and its switching interval is updated to [0.1609ns,0.1609ns].

Iteration 2 : If the resulting intervals above were consistent with the assumptions on coupling capacitance, no further iterations would be necessary. However, we find that the updated intervals of wire 1 were dependent on the switching intervals of wire 2, which were subsequently updated. In reality, the assumption of an effective coupling capacitance of 0 at time 0.3432ns was incorrect, and therefore, an update to the earliest switching time of 0.3309ns with C_c would result in the

correct value, namely, 0.4016ns. Similarly, it can be found that the latest switching time must be updated, and the resulting switching interval for wire 1 is now [0.4016ns,1,3623ns].

The iterations stop here since the values of the switching intervals are consistent with the values of the equivalent coupling capacitances.