Capturing the Effect of Crosstalk on Delay *

Sachin S. Sapatnekar
Department of Electrical and Computer Engineering
University of Minnesota, 200 Union Street SE,
Minneapolis 55455, USA.

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: (a) introducing unwanted noise induced in a quiet line, and (b) 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, or a very high-level model that captures the spirit, if not the details, of the crosstalk phenomenon (for example, [1–3]). 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.

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 [4, 5]. 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 [6]. The approach provides exact waveforms through the use of waveform relaxation that capture the effect of coupling on delay, but has a high computational cost.

The optimization problem chosen here has the same general goal as [1], 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.

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. 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. (b) The application of this procedure to channel routing is illustrated on several examples.

2 Preliminaries and Motivation

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:

- 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.

*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.

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.

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, $C_{c,eq}$.

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

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.

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 [7] 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).

The relation between crosstalk and timing is illustrated by the simplified three-wire example in Figure 1. We assume interconnect parameters in accordance with [8], 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 [1]. 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.

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

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.

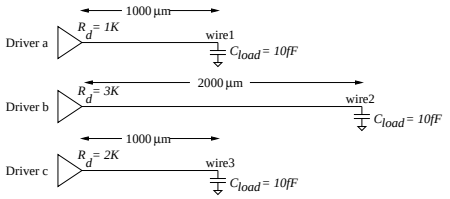

Figure 1: An example showing the effect of crosstalk on timing.

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 [0.331ns,1.081ns], [0.343ns,0.443ns], and [0.161ns,0.161ns], 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.

Consider the switching of wire 1. A switching event at any time in the interval [0.331ns,0.343ns] corresponds to a coupling capacitance of C_c, implying that incorporation of coupling capacitance effects would update these switching times to the interval [0.402ns,0.414ns]. An event in the interval [0.343ns,0.443ns] 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 [0.402ns,1.222ns], [0.554ns, 1.075ns] and [0.302ns,0.302ns], respectively.

The objective of this example was to help the reader appreciate the difficulty involved in calculating these switching intervals, and to motivate the need for a precise, efficient and systematic algorithm for the purpose.

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 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 Algorithm for Correct Delay 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. 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.

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 [9,10], 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.

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 T_start, T_end
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
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 $T_{end}(j) > T_{end}(i) > T_{start}(j)$, 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 formula $T_{end}(i) =$ 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 $2C_c$.
- If $T_{end}(i) > T_{end}(j) > T_{start}(i)$, then the latest concurrent switching 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 the update formula $T_{end}(i) = \max$[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 $T_{end}(i)$ = Update$(T_{end}(i), C_c)$ and $T_{end}(j)$ = Update$(T_{end}(j), C_c)$.

T_{start} updates

- If $T_{start}(j) > T_{start}(i)$, then update $T_{start}(i) = \min$[Update$(T_{start}(i), C_c), T_{start}(j)$].
- If $T_{start}(i) > T_{end}(j)$, then update $T_{start}(i) =$ 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: 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, 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 shown below. □

Theorem 2: The computational complexity of the algorithm is $O(nm^2 + mn^2)$, 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 $O(n^2)$.
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 forward pass, each of the n wires is updated by its immediate neighbors. This leads to a maximum of $O(m^2)$ updates per wire, implying that the cost of the first iteration is $O(nm^2)$.

From the second iteration onwards, as shown in the proof of Theorem 1, the value of T_{end} is nonincreasing. The update formula for T_{end}, listed in Section 3.1, implies that the value is determined by either

(a) a change in the value of T_{end} of a neighboring wire to alter the effective coupling capacitance from $2C_c$ to C_c, or

(b) a change in the value of T_{end} for a neighboring wire while the effective coupling capacitance is unchanged at $2C_c$.

We will now recall the statement in the proof of Theorem 1 that showed that the spans of the $[T_{start}, T_{end}]$ interval must contract from one iteration to the next; this fact will often be used in this proof.

For case (a) above, this implies that a net can be updated in this manner at most m times during the entire analysis procedure. The analysis for case (b) is more involved and is as follows.

Consider a change in the value of T_{end} that is triggered by a certain net. This may trigger a change in the value of T_{end} for one or more of its neighboring nets, which may further trigger changes in the values of T_{end} for neighbors of that net, and so on. Note that each of these changes must necessarily be reductions. By the pigeonhole principle [11], since the number of nets is n, at most n nets will be processed in this manner before the originating net is again a candidate for such a reduction.

For at least one wire in the system, the T_{end} value will not be reduced further when it becomes a candidate for the second time; if it did, it would imply that the T_{end} value for each wire could reduce indefinitely by discrete nonconverging amounts through multiple passes through this cycle, leading to a T_{end} value of $-\infty$, which is impossible since T_{end} is bounded below by a finite positive number.

Therefore, for this wire, the number of updates to its T_{end} value is no more than m, corresponding to one update of case (a) above from each of its m neighbors. If this wire has m updates to its T_{end} value, it can contribute no more than m case (b) updates to each of its m neighbors. In conjunction with the case (a) updates to these neighbors, this implies that each of the m wires would have no more than $2m$ updates in all from the originating wire. In fact, using a similar argument as before, for at least one of the m neighbors, the number of updates must be guaranteed not to exceed $2m$. Similarly, for at least one of the m neighbors of this wire, the number of updates cannot exceed $3m$, and so on. Consequently, the maximum number of T_{end} updates over all wires cannot exceed

$$m + 2m + 3m + \cdots + mn = O(mn^2)$$

In a similar fashion it can be shown that the number of T_{start} updates is $O(mn^2)$, implying that the computational complexity of the entire procedure is $O(nm^2 + mn^2)$. □

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 $O(n\log n)$.

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 2, 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.

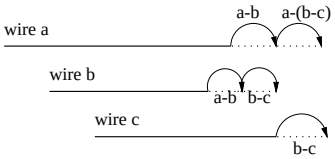

Figure 2: An example showing that the left-edge ordering is heuristic and not optimal.

4 Optimized Channel Routing

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 [9]:
(1) *horizontal constraints*: two nets whose horizontal spans overlap must not occupy the same track, and
(2) *vertical constraints*: 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.

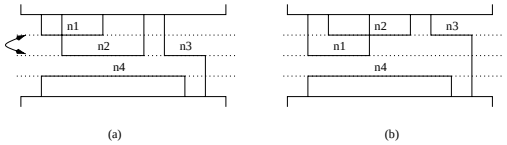

Figure 3: Two permuted channel routing solutions.

The process of exchanging tracks in a routed channel can reduce the crosstalk in a channel. In the simple example in Figure 3(a), if the first two tracks are exchanged, as shown in Figure 3(b), the crosstalk in the channel would be "reduced"; the procedure in [1] would produce such a solution. However, if the 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.

The algorithm for optimizing the channel routing solution for crosstalk effects uses a simulated annealing engine. The simulated annealing algorithm [12] 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 3(a), some possible moves are:
(a) moving net n2 to the first track and n1 to the second
(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.

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

Table 2: Results of Channel Reordering on Timing

	# nets	Improv. over init. (estimated)	CPU Time	Improv. over init. (SPICE)	Improv. 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%

Kuh channel router [9] that optimizes the height of the channel. The eight examples are taken from [9], 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 last column of Table 2. It is seen that our optimizer provides improvements in each case. Note that in the table, Deutsch1 shows larger improvements than Deutsch2 since it uses a larger number of tracks and has greater flexibility in reordering for crosstalk reduction.

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 outer loop of `Algorithm Update_Switching_Times` is never invoked more than four times 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.

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.

References

[1] 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.

[2] 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.

[3] 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.

[4] 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.

[5] 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.

[6] 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.

[7] 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.

[8] J. Cong, "Challenges and opportunities for design innovations in nanometer technologies," tech. rep., Semiconductor Research Corporation, Research Triangle Park, NC, 1997.

[9] 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.

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

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

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