Timing Analysis with Crosstalk as Fixpoints on Complete Lattice

Hai Zhou, Narendra Shenoy, and William Nicholls
Advanced Technology Group
Synopsys, Inc.
Mountain View, 94043

ABSTRACT

Increasing delay variation due to crosstalk has a dramatic impact on deep sub-micron technologies. It is now necessary to include crosstalk in timing analysis. But timing analysis with crosstalk is a chicken-and-egg problem since crosstalk effect in turn depends on timing behavior of a circuit. In this paper, we establish a theoretical foundation for timing analysis with crosstalk. We show that solutions to the problem are fixpoints on a complete lattice. Base on that, we prove in general the convergence of any iterative approach. We also show that, starting from different initial solutions, an iterative approach will reach different fixpoints. The current prevailing practice, which starts from the worst case solution, will always reach the greatest fixpoint (which is the loosest solution). In order to reach the least fixpoint, we need to start from the best case solution. Base on chaotic iteration and heterogeneous structures of coupled circuits, we also design techniques to speed up iterations.

1. INTRODUCTION

With the progress of deep sub-micron technologies, shrinking geometries have led to a reduction in self-capacitance of wires. Meanwhile coupling capacitances have increased as wires have a larger aspect ratio and are brought closer together. For present day processes, the coupling capacitance can be as high as the sum of the area capacitance and the fringing capacitance, and trends indicate that the role of coupling capacitance will be even more dominant in the future as feature sizes shrink [2]. This makes crosstalk a major problem in IC design. Crosstalk can affect the behavior of a circuit in two ways:

- introducing noise between adjacent wires;
- altering the delay of a switching transition.

When coupling capacitance is dominant, fast switching in aggressor gates can induce a large amount of noise on a victim line. If an aggressor and a victim switch simultaneously in the same direction, the victim will speed up. Likewise, if an aggressor and a victim switch in opposite directions, the victim will slow down.

Assuming that coupling capacitances dominate all other capacitances on a wire, failure to take crosstalk effect into timing analysis may produce results far off from the reality. However, timing and crosstalk effect are mutually dependent. This makes timing analysis with crosstalk a chicken-and-egg problem. For example, consider the two coupled nets in Figure 1(a). The switching time on net a is dependent on the switching time on net b. But the switching time on net b is not fixed, it is dependent on the switching time on net a. One way to solve the mutual dependence problem is by iteration. First, a switching time on a is computed based on a fixed initial switching time on b. Then the switching time on a is fixed and used to compute a new switching time on b. This two steps are iterated until we find a converged solution. By treating nets a and b simultaneously in one system, the relative window method of Sasaki and De Micheli [13] generates the switching time on a and b directly from the switching time on i_a and i_b. But it is hard to say that their method avoids iterations since the simulation they use may implicitly use iterations. Even though we can directly generate switching time on a and b from switching time on i_a and i_b, we still cannot avoid the chicken-and-egg problem. This is because there are two kinds of chicken-and-egg problems in timing analysis with crosstalk. We call the problem in Figure 1(a), that is, the mutual dependence of a set of directly coupled nets local chicken-and-egg problem. Besides that, a circuit structure may introduce cycling dependences which form a global chicken-and-egg problem. For example, even though we can generate switching time on a and b directly from those at i_a and c, the time on c (together with the time on d) is not available until we know the time on a.

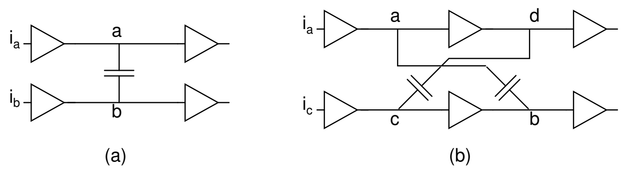

Figure 1: Timing analysis with crosstalk is a chicken-and-egg problem: (a) local problem; (b) global problem

The current practice to solve these chicken-and-egg problems relies on iterative approaches. Usually such approaches first assume a situation of crosstalk coupling (often a worst

DAC 2001, June 18-22, 2001, Las Vegas, Nevada, USA.

case situation). Then they compute the timing information and use it to modify the crosstalk coupling situation. They generally do each pass on the whole circuit and iterate till the solution converges.

In this paper, we establish a theoretical foundation for timing analysis with crosstalk. We show that solutions to the problem are fixpoints on a complete lattice. Base on that, we prove in general the convergence of any iterative approach. We also show that, starting from different initial solutions, an iterative approach will reach different fixpoints. The current prevailing practice, which starts from the worst case solution, will always reach the greatest fixpoint (which is the loosest solution). In order to reach the least fixpoint, we need to start from the best case solution. Base on chaotic iteration and heterogeneous structures of coupled circuits, we also design techniques to speed up iterations.

The rest of the paper is organized as follows. In Section 2 we formulate general timing analysis (either dynamic timing simulation or static timing analysis) as computing a fixpoint of a mathematical transformation. We then study in Section 3 Sapatnekar's approach [12] to show that multiple fixpoints exist in the system. Section 4 establishes the fact that the solution space forms a complete lattice, proves the convergence of any iterative approach, and characterizes the relation between initial solution and final fixpoint. Section 5 designs speeding up techniques based on chaotic iteration and heterogeneous circuit structures.

2. TIMING ANALYSIS AS FIXPOINTS

In this section, we will formulate general timing analysis (be it static timing analysis or dynamic timing simulation) as a fixpoint computation. Here we are given a combinational circuit which is composed of a set of gates and their interconnections. We select a set of points on the circuit as our interest points where timing information needs to be computed. They include the primary inputs, primary outputs, and the inputs and outputs of all the gates. Depending on the purpose of the timing analysis, the timing information on each of these points might be the delay, slew, switching window, the whole waveform, or any combination of them. Mathematically this means it may be a scalar value, a vector, or even a function of time (representing the whole waveform). We use a variable x_i to represent each timing information on a point i. Furthermore we use X to represent the vector $(x_1, x_2, \dots, x_n)$, that is, the timing information for the whole circuit.

Actually, the timing information on point i is only directly dependent on a subset of other points $i_1, i_2, \dots, i_k$. This means we can compute x_i by

$$x_i = t_i(x_{i_1}, \dots, x_{i_k}),$$

where t_i is any delay model used to derive the timing information on a point from that on its dependent points. Put all these local transformation together, we get a transformation for the whole system which can be written as

$$X = T(X) \tag{1}$$

A solution to the timing analysis is an X which satisfies the above equation, that is, a fixpoint of T.

When crosstalk effects are included in timing analysis, besides the fanins, the timing information on point i also depends on the timing information on coupled points. For the simplest coupling case shown in Figure 1(a), we have

$$\begin{aligned} x_a &= t_a(x_{i_a}, x_b), \\ x_b &= t_b(x_{i_b}, x_a). \end{aligned}$$

As we can see, a cycle is formed here because of the mutual dependence of x_a and x_b. Because of this, the transformation T becomes very complex in the presence of crosstalk.

For a complex transformation T, iterative method may be the only possible way to find its fixpoint. It works as follows. First an initial solution X_0 is guessed, then new solutions are iteratively computed from previous solutions $X_1 = T(X_0), X_2 = T(X_1), \dots$ until we find a fixpoint $X_n = X_{n-1}$. To the best of our acknowledgment, all previous work in the literature uses iterative methods to solve timing analysis with crosstalk. Most of them [6, 10, 13, 12, 14, 3] deal with the local chicken-and-egg problem, that is, they consider how to compute the delays of a set of coupled nets given their input time. Specifically, Dartu and Pileggi [6] propose to use an effective capacitance gate delay model to model the gates with dominant coupling capacitance. After that, a set of equations need to be solved to get the parameters, which then involve an iterative approach. Gross et al. [10] design a waveform iteration approach to explicitly solve the equations in [6]. Sasaki and co-workers [13, 14] propose a relative window method which relates the delays of two coupled nets to their input time. But their process to compute such relations by simulation may include iterations implicitly. Iterative method is also used in [3] to compute the Miller factor. Other work [15, 1, 4] analyzes the whole circuit and thus consider the global chicken-and-egg problem.

In order to use iterative methods to find a fixpoint of a transformation T, two basic questions need to be asked:

1. Is T a convergent transformation (at least on a subset A of the domain)? that is, whether there exist a finite n such that $T^{n-1}(x) = T^n(x) \ \ \forall x \in A$.
2. Does T have a unique fixpoint?

Most previous work provides convergence proofs for the approaches. Gross et al. [10] show the convergence based on the approach's similarity to waveform relaxation [11]. Both Sapatnekar [12] and Arunachalam et al. [1] base their convergence arguments on the monotonic shrinking of the switching windows. However, none of them study the uniqueness of their solutions. As we will show in the next section, uniqueness is not guaranteed by convergence and simply finding one fixpoint is not enough.

3. MULTIPLE FIXPOINTS

Since it is not possible to study an abstract transformation T, we will use Sapatnekar's model [12], which is the simplest among existing work, as our study case.

Sapatnekar [12] considered the delay computation in the presence of crosstalk for a set of wires within a routing channel. For each driver, a switching window $[T_{\min}, T_{\max}]$ signifying the range of switching time at the input of the driver, and a source resistance, R_d, are specified. The intrinsic and coupling capacitances of a wire are computed from the routing. Then coupling capacitances are modeled by effective capacitances to the ground and delays are computed by Elmore delays. The value of an effective capacitance is dependent on the switching time of the two coupling wires. Given a coupling capacitance C_c between two wires, if they switch at the same time and in the opposite direction, then an effective capacitance of $2C_c$ is used; if they switch at the same time and in the same direction, then an effective capacitance of 0 is used; if they do not switch at the same time, then an effective capacitance of C_c is used. However, in static timing analysis, a range of switching time is computed. Thus, the worst case analysis is used, which assumes that any switching within the range is possible. The algorithm to compute the wire delays works as follows. First, initialize a switching window on each wire such that the minimum and maximum time are computed by using 0 and C_c as effective capacitances, respectively. Then the maximum time for each wire is updated using effective capacitance of C_c or $2C_c$ based on whether switching windows are overlapping. Similarly, the minimum time for each wire is updated using 0 or C_c. These two updates are repeated in an alternative fashion until there is no further change.

We now use an example to show that multiple fixpoints exist for this transformation. In the example, we have only two nets a and b as shown in Figure 1(a). They couple with each other with a capacitance of 10 units. The nets are identical with the same driver of resistance of 10 units and the same load of capacitance of 1 unit. Suppose the minimum and maximum arrive time for signal i_a, that is $T_{\min}(i_a)$ and $T_{\max}(i_a)$, be 0 unit and 1 unit, respectively. Similarly, let $T_{\min}(i_b) = 10$ units and $T_{\max}(i_b) = 11$ units. Elmore delay is used to compute the delays.

According to the algorithm, the initial switching windows of wires are computed by using effective capacitance of 0 for minimum time and that of C_c for maximum time. That is, $[T_{\min}(a), T_{\max}(a)] = [10, 111]$ and $[T_{\min}(b), T_{\max}(b) = [20, 121]$. Now their switching windows overlap, thus updates are needed. We get $[T_{\min}(a), T_{\max}(a)] = [10, 211]$ and $[T_{\min}(b), T_{\max}(b) = [20, 221]$. Since there is no update needed, the approach converges to this solution.

But if we assume that there is no switching window overlap at the beginning, we can use C_c as effective capacitance both for the minimum and maximum time. In this case, we have $[T_{\min}(a), T_{\max}(a)] = [110, 111]$ and $[T_{\min}(b), T_{\max}(b)] = [120, 121]$ as initial solution. Then we find that there is no update needed, thus it is also a converged solution.

This example shows that there are more than one fixpoints in Sapatnekar's model, and starting from different initial solution, we may get different fixpoints. A similar argument can be made to prove the existence of multiple fixpoints in other models.

4. OPTIMAL FIXPOINT

As we have already seen from the previous section, there may be multiple fixpoints for a timing transformation T. A natural question from that is: which one should be used?

Since complete information is not used (functionality is not used), uncertainty is unavoidable in static timing analysis. This means that the timing information we compute on each point is a set representing the possible switchings, in stead of a single switching. The result assures the users that "the real (physical) switching is one in this set of switchings but which one is not known." A set of switchings can be represented by a switching window and a range of slew rates such that any switching falling within the window and having a slew in the range is in the set. But other representations are also possible.

Now consider the family of all sets of switchings on a point. The inclusion relation (that is $\subseteq$) forms a partial order on the family, that is, it is

- reflexive: $A \subseteq A$
- antisymmetric: $A \subseteq B \wedge B \subseteq A \rightarrow A = B$
- transitive: $A \subseteq B \wedge B \subseteq C \rightarrow A \subseteq B$

Actually, the sets of switchings on a point are subsets of the whole switching set which consists of all possible switchings. According to the lattice theory [7], a partially ordered set forms a *complete lattice* if any subset has a least upper bound and a greatest lower bound for its elements. In fact, the family of all subsets of a given set with inclusion relation forms a complete lattice. Given a set $S = \{a, b, c\}$, the partial order of inclusion on its subsets can be represented by a Hasse diagram shown in Figure 2. Here, two sets with inclusion relation are connected by an edge, and the lower set is included in the higher set. Timing information of a circuit

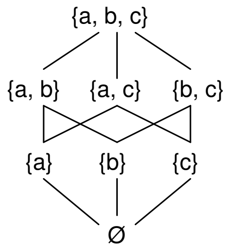

Figure 2: Subsets of a given set form a complete lattice

is a vector of timing information on all points. The partial order on each point can be extended point-wisely to get a partial order on vectors: two vectors $A = (A_1, A_2, \ldots, A_n)$ and $B = (B_1, B_2, \ldots, B_n)$ satisfy $A \subseteq B$ if and only if $A_i \subseteq B_i$ for all $1 \leq i \leq n$. It can be shown that the vectors with such a partial order also forms a complete lattice.

Now consider a transformation T. In static timing analysis, it works on a complete lattice we defined above, that is, it transforms a vector of sets of switchings to a vector of sets of switchings. We say that T is a *monotonic*(or *order-preserving*) transformation when, for any vector of subsets X and Y, if $X \subseteq Y$ then $T(X) \subseteq T(Y)$. The monotonicity of transformation T is based on the monotonicities of its member transformations $t_1, t_2, \ldots, t_n$, which must be true for any reasonable static timing analysis. Otherwise, it means that we can have fewer possible switchings at a point while there are more possible switchings at its fanins and coupling points.

Given a subset S of elements in a complete lattice L, we use $\bigvee S$ and $\bigwedge S$ to represent the least upper bound and the greatest lower bound of elements in S, respectively. The existence of a fixpoint in our system is guaranteed by the following theorem due to Knaster and Tarski [7].

Theorem 1 (Knaster-Tarski). *Let L be a complete lattice and $T : L \rightarrow L$ an order-preserving map. Then*

$$\bigvee\{x \in L | x \subseteq T(x)\} \in fix(T),$$

where $fix(T)$ is the set of fixpoints of T.

But it is not feasible to use the above theorem to compute a fixpoint since it is not feasible to compute the set $\{x \in L | x \subseteq T(x)\}$. Instead, people usually use iterative method (also called *successive approximation*) to find a fixpoint. That is, they first select an initial solution X_0 and iteratively compute $X_1 = T(X_0), X_2 = T(X_0), \ldots$ in the hope of finding an X_n such that $X_n = T(X_n)$. But the hope can not be fulfilled by starting from any initial point. Fortunately the bottom and the top elements are good candidates for that.

Following a tradition in lattice theory, we use $\perp$ and $\top$ to represent the bottom and the top elements of our complete lattice, respectively. That is, we have $\perp = \{\emptyset, \emptyset, \ldots, \emptyset\}$ and $\top = \{P_1, P_2, \ldots, P_n\}$ where P_i is the set of all possible switchings on point i. Since $T(\top) \subseteq \top$, based on the monotonicity of T, we have

$$\begin{array}{rcl} T(\top) & \subseteq & \top \\ T^2(\top) & \subseteq & T(\top) \\ T^3(\top) & \subseteq & T^2(\top) \\ & \cdots & \end{array}$$

Therefore we have a descending chain $\top \supseteq T(\top) \supseteq T^2(\top) \supseteq \cdots$. Similarly, starting with $\perp$, we have an ascending chain $\perp \subseteq T(\perp) \subseteq T^2(\perp) \subseteq \cdots$. If the chain has only finite elements, which is true on any finite solution space, the process will finally reach a fixpoint. Actually, the only properties we used about $\top$ and $\perp$ are $T(\top) \subseteq \top$ and $\perp \subseteq T(\perp)$. Therefore, any solution X_0 such that either $X_0 \subseteq T(X_0)$ or $X_0 \supseteq T(X_0)$ can be used as an initial solution to reach a fixpoint. If $X_0 \subseteq T(X_0)$, we get an ascending chain; if $X_0 \supseteq T(X_0)$, we get a descending chain.

The case with infinite chains is more complex. When chains are infinite, stronger requirements are needed on T for them to converge to fixpoints.

DEFINITION 1. *A function $T : L \to L$ is* or-continuous *if for any chain C, $T(\bigvee C) = \bigvee\{T(c) | c \in C\}$, or equivalently $T(\bigvee C) = \bigvee T(C)$. If $T(\bigwedge C) = \bigwedge T(C)$, T is called* and-continuous.

It is easy to check that an or-continuous (or and-continuous) function is order-reserving.

THEOREM 2. *If T is or-continuous, then the least fixpoint of T is $\bigvee_{n \geq 0} T^n(\perp)$; if T is and-continuous, then the greatest fixpoint is $\bigwedge_{n \geq 0} T^n(\top)$.*

When a set of switchings is represented by a switching window, the continuity on the order coincides with the traditional continuity on real functions. That is, a continuous transformation will map a small change in input windows to a small change in output windows. This is also generally true for common transformations.

The following theorem shows that if we find fixpoints by the above method, they must be the least and the greatest fixpoints.

THEOREM 3. *Let L be a complete lattice, let $T : L \to L$ be an order-preserving map and define $\alpha := \bigvee_{n \geq 0} T^n(\perp)$ and $\beta := \bigwedge_{n \geq 0} T^n(\top)$.*

1. *If $\alpha \in fix(T)$, then α is the least fixpoint;*

2. *If $\beta \in fix(T)$, then β is the greatest fixpoint.*

Furthermore, the fixpoints of T have a very good structure.

THEOREM 4. *If L is a complete lattice, and $T : L \to L$ is an order-preserving map. Then $fix(T)$ is a complete lattice.*

These theorems show that the fixpoint found by successive approximation from the top element is the union of all fixpoints and the one found from the bottom is the intersection of them. In terms of switching window, that means you will get a solution with the largest switching windows if starting from an initial assumption that all switching windows overlap with each other, and you will get a solution with the smallest windows if starting from an initial assumption that no window overlap with the other. Therefore, our conclusion is that, *in order to find a solution with the minimum uncertainty, you should start with the no-window-overlap assumption.*

In practice, people sometimes like to change timing model during the iterations: starting with a coarse estimation and gradually changing into more and more accurate models. We must be very careful with this practice, since the transformation is now changing with iterations and the convergence may not be guaranteed. Now suppose the transformation in the ith iteration is T_i. If we start with $\perp$, we have $\perp \subseteq T_1(\perp), T_2(\perp) \subseteq T_2(T_1(\perp)), \ldots$, but this only gives us a chain if we also have $T_i(X) \subseteq T_{i+1}(X)$, which means a later model should not be more accurate than a previous model. So the practice of using finer and finer model can not be used in any iterative approach with increasing windows, or you take the risk of looping infinitely. On the other hand, it can be safely used in approaches with decreasing windows since we always have a chain

$$\top \supseteq T_1(\top) \supseteq T_2(\top) \supseteq T_2(T_1(\top)) \supseteq \ldots$$

But this does not imply that an approach with decreasing windows are more efficient than that with increasing windows. Let α_1 and β_1 to be the least and greatest fixpoints of T_1 respectively. Similarly, let α_n and β_n to be the least and greatest fixpoints of T_n. For any $x \in \text{fix}(T_1)$ we have $x = T_1(x) \supseteq T_n(x)$ which means a fixpoint of T_n can be found if we start with x. Using a diamond to represent a complete lattice, the relation between $\text{fix}(T_1)$ and $\text{fix}(T_n)$ can be shown in Figure 3. Depending on whether $\alpha_1 = T_n(\alpha_1)$, we have two cases. If $\alpha_1 = T_n(\alpha_1)$ then we can prove that $\beta_n = T_1(\beta_n)$. This is shown in (a) where the fixpoint sets of T_1 and T_n overlap. In this case, using finer and finer models from $\top$ will find β_n, but keeping with the coarse model from $\perp$ will give us a better solution α_1. In the case shown in (b), α_1 is not a fixpoint of T_n, but since $\alpha_1 \supseteq T_n(\alpha_1)$, we can iterate using T_n from α_1 to get β_n. That means we can keep using the coarse model until we find a fixpoint, then change to a finer model. Furthermore, in order to find the best fixpoint α_n, the only way is to use T_n from $\perp$, which means that the solution using the coarse model has to be discarded.

5. SPEEDING UP TECHNIQUES

Strictly speaking, applying the transformation T to a solution X_i, that is, computing $X_{i+1} = T(X_i)$, uses all the previous values to compute the new values even when some of the new ones are available. This is similar to Jacobi's method in solving matrix equations [9]. In practice, people already deviate from this strict sense: they usually do the

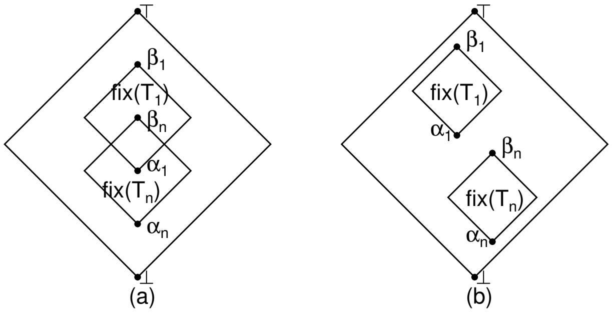

Figure 3: Structures of fixpoints

process in a topological order of a circuit and always use a new value if one is available. This is much like Seidel's method to solve matrix equations. Many useless updates are thus trimmed off. But we find that without further exploiting both circuit and coupling structures and their interaction, many updates are still wasted. For example, in Figure 4, if we process each iteration according to a topological order of the circuit, any update at d may be propagated to e, f, g and h. But if the update at d is not permanent, those propagations will be overwritten later.

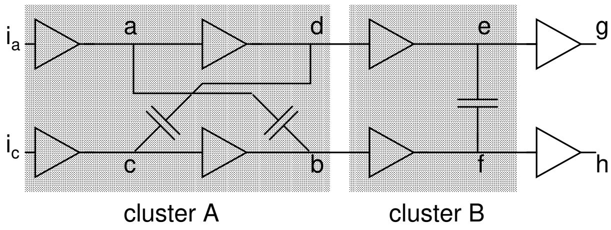

Figure 4: Exploit circuit structure by clustering

Before we design a good iterative order for updates, we need to establish its theoretical validity. That is, no matter what order is used, the process will always converge to the same fixpoint. This is called the scheme of *chaotic iteration* [5]. Here, a transformation T is composed of a set of partial transformations $t_1, t_2, \ldots, t_n$. In each step, one or more partial transformations are applied to update timing information on one or more points. All timing information on other points is kept the same. We will use T_S to represent such a partial transformation done in one step, where S represents the set of points where timing information is updated.

LEMMA 1. *Given S a subset of points, if $X \subseteq T(X)$, then $X \subseteq T_S(X) \subseteq T(X)$; if $X \supseteq T(X)$, then $X \supseteq T_S(X) \supseteq T(X)$.*

The above lemma states that no matter what evaluation order is used, the generated sequence is monotonic in the same direction and it will not over-shoot the fixpoint generated by T. Furthermore, if the evaluation order is fair, that is, a partial transformation will always be applied if its inputs and outputs are not consistent, then the chaotic iteration will always reach the same fixpoint as T.

A circuit structure forms a partial order relation on the set of points which can be represented by a DAG (Direct Acyclic Graph). Let each coupling capacitor introduce a bidirectional edge on the pair of points. Then we have a general directed graph on the nets. Identify each of the strongly connected components and call it a cluster. Then do the timing analysis on each of the clusters until it converges, in a topological order of the clusters. In Figure 4, we have two clusters A and B. Not processing cluster B until having a stable cluster A means that previous mentioned useless updates will be trimmed off. Notice that clusters A and B have different structures. Actually, cluster B is simpler and corresponding to the local chicken-and-egg problem, so we call it *local cluster*. Cluster A includes some interacted local cluster and is corresponding to the global chicken-and-egg problem, so we call it *global cluster*. Solving a problem of timing analysis with crosstalk now is reduced to solving that problem on a set of local and global clusters. Since a global cluster includes at least one local cluster, we will focus on how to solve the problem in a global cluster.

There are many different ways to arrange iterations in a global cluster. If we always compute local clusters together, as implicitly suggested by [10, 1], the structure of a global cluster can be viewed as in Figure 5, where each block represents a local cluster. To facilitate iterations, a set of gate inputs are selected as feedback edges whose removal makes the structure acyclic. Given initial values on feedback edges, timing analysis can be done in the acyclic part (perhaps with a complicated computation on each local cluster) to give new values on feedback edges. If these values become stable (i.e. new values are the same as old values), then a fixpoint is reached. Otherwise, next iteration will start with the new values. Given feedback edges, iterations in each global cluster can be processed in two ways. The first approach, called iterative approach, re-computes the whole cluster based on new values and repeats this until all values become stable. The second approach, called recursive approach, only re-computes the points on the outer cycle after all inner cycles become stable. Although there is no direct relation between the number of iterations and the number of feedback edges, fewer feedback edges may give fewer possible value changes. However, finding the smallest number of feedbacks is NP-hard on a general graph [8].

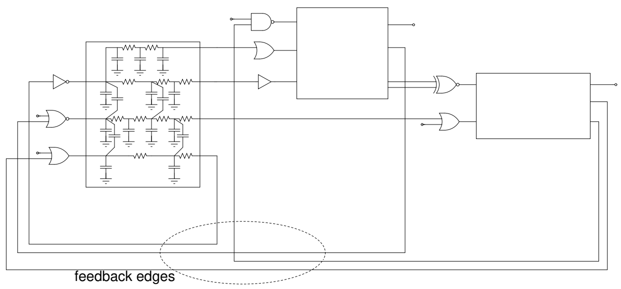

Figure 5: Use gate fan-ins as feedback edges

One drawback of the above approach is that, since all the feedbacks are gate inputs, changed values on them always need to be propagated. Studying the interactions in our system, we find that our system is a heterogeneous system. That is, there are two kinds of interaction relations: fanin relation is simple but strong; coupling relation is complex but weak. Fanin relation is simple because it is unidirectional, and it is strong because switching on input always influences switching on output. On the other hand, coupling relation is complex because it is bidirectional, and it is weak because switching on one net may not always influence the switching on the other. Based on these observations, we adopt the following principle in our iterations: always using coupling edges as feedback edges. Our procedure works as follows. Initially, assume all switching windows are empty, that is, no coupling net switches at the same time. Thus the first iteration is just a traditional timing analysis. After this iteration, a switching window is given on each point. Then for each pair of coupling points a and b, we can define their timing proximity as

$$P(a,b) = \min(T_{\max}(a), T_{\max}(b)) - \max(T_{\min}(a), T_{\min}(b)).$$

In fact, it defines the overlap length of the two switching windows; when there is no overlap, it is negative and its absolute value defines the distance between the two windows. Now we will choose a set of coupling edges with the smallest timing proximities as feedback edges whose removal makes the circuit acyclic (in the sense of treating each local cluster as a block). That is, the circuit will be viewed in a structure as shown in Figure 6. During the iterations

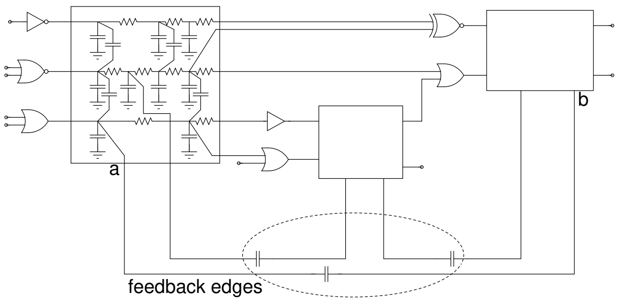

Figure 6: Use coupling edges as feedback edges

that follow, the feedback edges are not physically broken up, but the timing computation on the two points connected by each of them is separated. For example, when computing the time on point a in Figure 6, its coupling with point b is considered but time on point b is assumed to be fixed. Later, when computing time on point b, its coupling with point a is considered but time on point a is assumed to be fixed. Therefore, if the switching windows on a and b do not overlap during the iterations, time change on one point does not influence the other point. Our selection of feedback edges based on timing proximity intends to make this happen as frequently as possible.

However, it is possible for two points connected by a feedback edge to have overlapped switching windows. In this case, time change on one point will influence the time on the other point, and many iterations may be needed to bring them up to consistency. Such iterations are very expensive, since each time the circuit between the two points must be re-computed. Fortunately, approximation can be used to reduce the number of iterations. This also shows another benefit of using coupling edges as feedbacks. It works as follows. When updating the time on point a which is now influenced by time on point b, we simply assume that b has an infinite window; then later, when updating time on point b, we will use the actual window on point a. Doing this, we can guarantee to get a post fixpoint as our solution.

6. REFERENCES

[1] R. Arunachalam, K. Rajagopal, and L. T. Pilleggi. Taco: Timing analysis with coupling. In *Proc. of the Design Automation Conf.*, pages 266–269, Los Angeles, CA, June 2000.

[2] Semiconductor Industry Association. National technology roadmap for semiconductors, 1997.

[3] P. Chen, D. A. Kirkpatrick, and K. Keutzer. Miller factor for gate-level coupling delay calculation. In *Proc. Intl. Conf. on Computer-Aided Design*, San Jose, CA, November 2000.

[4] P. Chen, D. A. Kirkpatrick, and K. Keutzer. Switching window computation for static timing analysis in presence of crosstalk noise. In *Proc. Intl. Conf. on Computer-Aided Design*, San Jose, CA, November 2000.

[5] P. Cousot and R. Cousot. Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *ACM Symposium on Principles of Programming Languages*, pages 238–252, Los Angeles, CA, January 1977.

[6] F. Dartu and L. T. Pileggi. Calculating worst-case gate delays due to dominant capacitance coupling. In *Proc. of the Design Automation Conf.*, pages 46–51, Anaheim, CA, June 1997.

[7] B. A. Davey and H. A. Priestley. *Introduction to Lattices and Order*. Cambridge, 1990.

[8] M. R. Garey and D. S. Johnson. *Computers and Intractability*. W. H. Freeman and Co., 1979.

[9] G. H. Golub and C. F. van Loan. *Matrix Computations*. Johns Hopkins, 3rd edition, 1996.

[10] P. D. Gross, R. Arunachalam, K. Rajagopal, and L. T. Pileggi. Determination of worst-case aggressor alighment for delay calculation. In *Proc. Intl. Conf. on Computer-Aided Design*, pages 212–219, San Jose, CA, November 1998.

[11] E. Lelarasmee, A. E. Ruehli, and A. L. Sangiovanni-Vincentelli. The waveform relaxation method for time-domain analysis of large scale integrated circuites and systems. *IEEE Transactions on Computer Aided Design*, July 1982.

[12] S. S. Sapatnekar. A timing model incorporating the effect of crosstalk on delay and its application to optimal channel routing. *IEEE Transactions on Computer Aided Design*, 2000.

[13] Y. Sasaki and G. De Micheli. Crosstalk delay analysis using relative window method. In *ASIC/SoC Conference*, 1999.

[14] Y. Sasaki and K. Yano. Multi-aggressor relative window method for timing analysis including crosstalk delay degradation. In *Custom Integrated Circuit Conference*, pages 495–498, 2000.

[15] P. F. Tehrani, S. W. Chyou, and U. Ekambaram. Deep sub-micron static timing analysis in presence of crosstalk. In *International Symposium on Quality Electronic Design*, pages 505–512, 2000.