FA-STAC: A Framework for Fast and Accurate Static Timing Analysis with Coupling

Debasish Das, Ahmed Shebaita, Hai Zhou, Yehea Ismail and Kip Killpack*
EECS, Northwestern University, Evanston, IL 60208
*Strategic CAD Lab, Intel Corporation, Hillsboro, OR 97124

Abstract*— This paper presents a framework for fast and accurate static timing analysis considering coupling. With technology scaling to smaller dimensions, the impact of coupling induced delay variations can no longer be ignored. Timing analysis considering coupling is iterative, and can have considerably larger run-times than a single pass approach. We propose a novel and accurate coupling delay model, and present techniques to increase the convergence rate of timing analysis when complex coupling models are employed. Experimental results obtained for the ISCAS benchmarks show promising accuracy improvements using our coupling model while an efficient iteration scheme shows significant speedup (up to 62.1%) in comparison to traditional approaches.

I. Introduction

With the progress of deep sub-micron technologies, shrinking geometries have led to a reduction in the self-capacitance of wires. However, the aspect-ratio of modern interconnects is over 2.0 for intermediate wiring layers, and is expected to increase. This indicates that interconnect coupling capacitances have grown to dominate the total interconnect capacitance. Kumar [1] showed that in recent DSM technologies, coupling capacitances could be five times larger than the vertical capacitances. The total parasitic capacitance of an interconnect is given by

$$C_{eq} \quad = \quad C_g + \sum_j MCF_j \times C_{cj} \tag{1}$$

where, C_g is the interconnect's equivalent ground capacitance, C_{cj} is the interconnect's coupling capacitance with coupled neighbor j, and C_{eq} denotes the interconnect's equivalent total capacitance. MCF_j is some factor (often termed Miller coupling factor) that depends on the switching windows on the interconnect and j. Prior work has calculated the switching factors to be within $(0, 2)$ for step transitions [2], within $(-1, 3)$ for ramp delay models [3], [4], and within $(-1.885, 3.885)$ for exponential transitions [5].

With an increasing significance of coupling, ignoring the impact of coupling induced delay variations produces results far off from the reality. The simplistic model proposed in [2] is not accurate enough to capture the true impact of coupling on timing. It is well known that timing and coupling (or crosstalk) are mutually dependent. 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. However, the switching time on net b is not fixed; it is dependent on the switching time on net a.

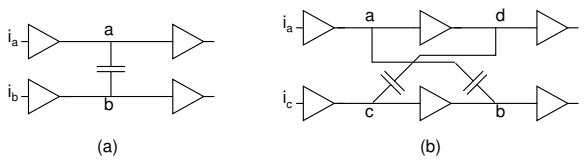

Fig. 1. Timing analysis with crosstalk is a mutual dependence problem: (a) local problem; (b) global problem

Present approaches to timing analysis considering crosstalk are iterative. Given input arrival times and starting with an initial assumption of crosstalk induced delays, the timing information on all interconnects are iteratively updated until convergence. For large designs, the run-times may be significant if the updates are not done efficiently. It is therefore critical to develop approaches to speeding up the analysis and improving the convergence rate.

Current approaches use either continuous coupling delay models [6] or discrete models (that employ Miller factors) [2], [4], [7]. In [6], three scheduling approaches, namely *dynamic event time*, *static event time*, and *smart global* are proposed, and the convergence time is compared. However, there is no thorough study on the convergence rate for different iterative schemes and the work in [6] leaves space for improvement.

Zhou [8] established a theoretical foundation for static timing analysis with crosstalk for both continuous and discrete models. A *chaotic iterative scheme* in this work provided the validity for exploring efficient iteration schedules. The approach in [6] initially assumes a situation of crosstalk delays (often the worst case situation). Subsequently, timing information is updated iteratively until convergence. This approach does not exploit the circuit and coupling structures. We propose speeding-up techniques that can greatly improve the convergence rate of a static timing analyzer such as [6] that consider realistic coupling delay models.

In this paper, we present a fast and accurate static timing analyzer *FA-STAC* that has the following novel components.

1) Waveform based accurate coupling model
2) Efficient iteration mechanism

The proposed waveform based accurate coupling model is an extension of [3] to timing analysis. Accurate coupling delay modeling is necessary to capture the impact of coupling on circuit delays correctly. The efficient iteration mechanism of *FA-STAC* uses structural information of the coupled circuit to speed up the convergence rate of iterations during timing analysis considering coupling. The ideas from the iterative procedure can be easily tuned for application to other static timing analyzers as well. The iteration mechanism first partitions all couplings into global and local groups, and then uses different orderings to iterate through them. We compare the performance of our framework with the algorithm proposed in [6].

The rest of the paper is organized as follows. In Section II, we present a waveform based accurate coupling delay model. Section III describes the proposed efficient iteration mechanism to increase the convergence rate of iterations. We present our experimental results in Section IV. Conclusions and future work are presented in Section V.

II. Accurate coupling delay calculation

Signal propagations in a circuit can be represented as a directed acyclic graph (DAG) while crosstalk couplings form bidirectional edges in the DAG. We have a general directed graph on the signals containing the circuit and the coupling structure. Formally, we define

this general directed graph as *G = (V,E)*, where, G is partitioned into two subgraphs $G^C = (V, C)$ and $G^F = (V, F)$. The edges in G^C are the bidirectional coupling edges and the edges in G^F are the fanin edges. The gates of the combinational circuit are represented by the elements of set *V*. Graph G basically represents a timing graph in presence of coupling generated out of a given combinational circuit. We present following definitions for our derivations of the coupling model.

Definition 1: Rise delay window for a gate $i : i \in G$ is defined as an ordered pair (rd_i^l, rd_i^h), where rd_i^l represents the minimum possible rise delay, and rd_i^h represents the maximum possible rise delay at the output of gate i.

Definition 2: Fall delay window for a gate i is defined as an ordered pair (fd_i^l, fd_i^h), where fd_i^l represents the minimum possible fall delay, and fd_i^h represents the maximum possible fall delay at the output of gate i.

Clearly Definition 1-2 can be interpreted as all the possible points in time where the output at gate i can switch from one logic level to the other logic level. It also gives an intuition behind the importance of using rise and fall windows in coupling analysis since it determines the MCFs as shown in Equation 1.

Definition 3: Rise slew window for a gate $i : i \in G$ is defined as an ordered pair (rs_i^l, rs_i^h), where rs_i^l represents the minimum possible rise slew, and rs_i^h represents the maximum possible rise slew at the output of gate i.

Definition 4: Fall slew window for a gate i is defined as an ordered pair (fs_i^l, fs_i^h), where fs_i^l represents the minimum possible fall slew, and fs_i^h represents the maximum possible rise slew at the output of gate i.

Based on Definitions 1-4 of interacting gates, the coupling factors can vary significantly. We next present an example for illustration.

A. Motivational example

We use an example as shown in Figure 2 to motivate the need for accurate crosstalk delay variation estimation during timing analysis. Nets $I1$ and $I2$ fanin to pin 1 and pin 2 of gates $G1$ and $G2$. Rise and fall delay windows on net $I1$ are [2,4] and [2.5,3.5]. Similarly on net $I2$ the windows are [3,5] and [3.5,4.5]. Assuming $G1$ and $G2$ are identical NAND gates, timing arcs *arc1* and *arc2* are identical. Each arc contains a 4-tuple timing information, which corresponds to rise delay, rise transition, fall delay and fall transition, respectively. Also let's assume that net $I1$ and $I2$ have similar rise and fall slew windows given by [0.2,0.6] and [0.4,0.8]. In Figure 2, we have not shown the ground capacitance. We denote the ground capacitances corresponding to a gate i as C_g^i (It is the lumped capacitance corresponding to net O^i). For the sake of simplicity, assume that we have a single rise slew $\frac{0.2+0.6}{2}$ and fall slew $\frac{0.4+0.8}{2}$. Consider *arc1* as positive unate and *arc2* as negative unate. Let the rise delay on *arc1* due to input rise slew as 0.4 and loading cap of CG^1+C_c be 0.6. Input window [2,4] transforms to [2.6,4.6]. Let the fall delay on *arc2* due to input fall slew as 0.6 and loading cap of C_g^1+C_c be 0.8. Input window [3.5,4.5] transforms to [4.3,5.3]. The final output rise window on gate $G1$ is given by [2.6,5.3]. Also the window of rise slew created on output of $G1$ due to input rise slew of 0.4 and input fall slew of 0.6 be [0.5,0.7]. Similarly the final output fall window on gate $G1$ comes from the windows [2.5+0.8,3.5+0.8] and [3+0.6,5+0.6] which is [2.5,5.6]. The fall slew window will be same as [0.5,0.7] due to symmetry. Note that $G1$ and $G2$ are similar gates. The difference lies in the fact that $G2$ has a loading capacitance of C_g^2+C_c. Suppose the difference in capacitances add 0.4 to all delay and 0.1 to all slew calculations on *arc1* and *arc2*. Thus at $G2$ we get the output rise delay window as [3.0,5.8] and fall delay window as [2.9,6.0]. Also the output rise and fall slew windows are given by [0.6,0.8]. Note that we have already simplified our calculations by considering a single slew. If I1 and I2 are nets connected to primary inputs, this assumption is valid. But for a general circuit, both the minimum and maximum slew values must be considered to get accurate output switching windows.

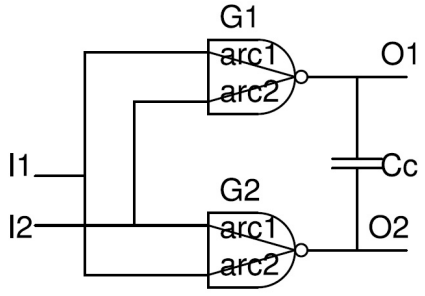

Fig. 2. Accurate analysis of coupling capacitance

For ramp based model, as shown in [3], a switching from rise/fall delay window at output of $G1$ and a switching from fall/rise delay window at output of $G2$ can give rise to coupling capacitance as large as 3×C_c. We show in this paper that on the basis of overlap ratio (Definition 5), the worst capacitance can be (1+2*(overlap ratio))×C_c.

B. Coupling factor computation

Since static timing analysis is corner case analysis, we can always take the maximum and minimum coupling factors to calculate the effect of coupling capacitances on fall and rise delay window.

$$\text{Rise-Delay-Window}^u = (rd_u^l, rd_u^h) \quad (2)$$

$$\text{Fall-Delay-Window}^u = (fd_u^l, fd_u^h) \quad (3)$$

$$\text{Rise-Delay-Window}^v = (rd_v^l, rd_v^h) \quad (4)$$

$$\text{Fall-Delay-Window}^v = (fd_v^l, fd_v^h) \quad (5)$$

$$\text{Rise-Slew-Window}^u = (sd_u^l, sd_u^h) \quad (6)$$

$$\text{Fall-Slew-Window}^u = (sd_u^l, sd_u^h) \quad (7)$$

$$\text{Rise-Slew-Window}^v = (sd_v^l, sd_u^h) \quad (8)$$

$$\text{Fall-Slew-Window}^v = (sd_v^l, sd_u^h) \quad (9)$$

Given a coupling edge $e = (u, v)$ with intrinsic coupling capacitance as C_c^e where u and v has rise and fall delay windows as shown in Equation 2-5 along with the rise and fall slew windows shown in Equation 6-9 respectively, we are seeking to find best and worst case coupling factors that might come from this configuration and thus determine the effective coupling capacitance between the nodes u and v. Note that rise and fall delay windows basically represent the maximum and minimum arrival times after which the gate output starts to change from its steady state. Therefore both the arrival times and the slews are extremely important to determine the coupling factors as it captures the shape of the waveforms accurately. According to the convention of [3] we assume one of the nodes u as *victim* while the other one v is considered as *aggressor*, though considering one particular node as the victim, while the other as an aggressor is artificial for timing analysis. The idea is that we consider one node as an victim and the rest of the nodes connected to it by coupling edges as aggressor, and the timing analysis procedure takes care of the fact that the victim node will also be considered as an

aggressor later. The worst and best coupling factors for the victim

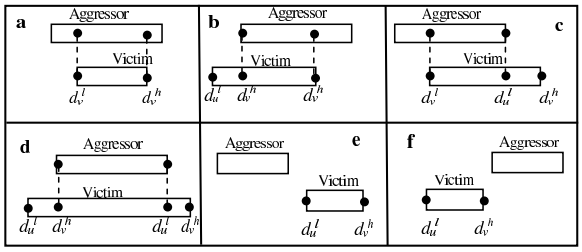

Fig. 3. Window overlap cases

u are obtained by $(1+2k)$ and $(1-2k)$ as shown in [3], and we multiply the intrinsic coupling capacitance C_c^e with it to get the effective coupling capacitance.

Definition 5: Overlap ratio k is defined as the ratio of the aggressor's output waveform that overlap with that of victim threshold voltage.

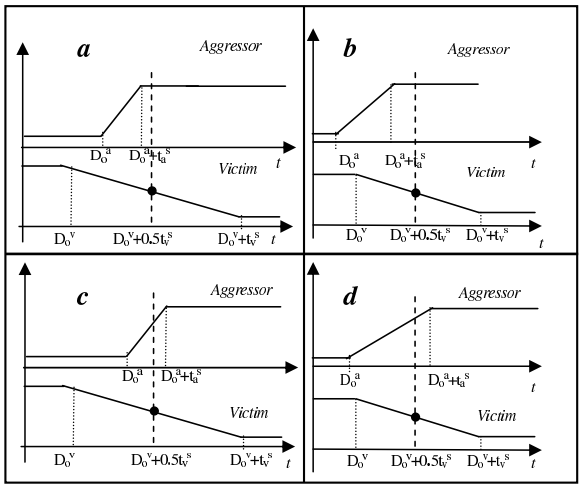

Fig. 4. Coupling factor computation

Based on the definition we have five possible Equations for k. The first four conditions are as shown in Figure 4(a-d). The last equation takes care of the case when there is no overlap between the victim and aggressor waveforms.

$$k = \begin{cases} 1 & \text{Figure 4(a)} \\ \frac{D_o^a + t_a^s - D_o^v}{t_a^s} & \text{Figure 4(b)} \\ \frac{D_o^v + 0.5t_v^s - D_o^a}{t_a^s} & \text{Figure 4(c)} \\ \frac{0.5t_v^s}{t_a^s} & \text{Figure 4(d)} \\ 0 & \text{No overlap condition} \end{cases} \quad (10)$$

Now we need to choose the points D, t^s from the delay and slew windows that gives the worst and best possible coupling factors for rise and fall delay computation. Note that worst case coupling factors come from rise and fall or fall and rise delay windows on victim and aggressor while best case coupling factors come from rise and rise or fall and fall delay windows respectively. After we choose the points D_o^a, D_o^v, t_a^s and t_v^s according to Figure 5, we consult Figure 4 for the corresponding waveforms that the chosen points from the windows will generate. Accordingly we obtain k from the equations given above. Coupling parameter computation as shown in Figure 5 uses different conditions of switching window overlap as shown in Figure 3.

Theorem 1: D and t^s points chosen for coupling factor computation as shown in Figure 4 gives the worst and best possible coupling factors for two coupled nodes at a certain time t.

Figure 3(a)
Max
$D_o^a = d_v^h, t_a^s = s_u^l$ AND $D_o^v = d_v^h, t_v^s = s_v^h$
Min
$D_o^a = d_v^l, t_a^s = s_u^l$ AND $D_o^v = d_v^l, t_v^s = s_v^h$
Figure 3(b)
Max
$D_o^a = d_v^h, t_a^s = s_u^l$ AND $D_o^v = d_v^h, t_v^s = s_v^h$
Min
$D_o^a = d_u^l, t_a^s = s_u^l$ AND $D_o^v = \text{best}(d_u^l, d_v^l), t_v^s = s_v^h$
Figure 3(c)
Max
$D_o^a = d_u^h, t_a^s = \text{worst}(s_u^l, s_u^h)$ AND $D_o^v = \text{worst}(d_u^h, d_v^h), t_v^s = s_v^h$
Min
$D_o^a = d_v^l, t_a^s = s_u^l$ AND $D_o^v = d_v^l, t_v^s = s_v^h$
Figure 3(d)
Max
$D_o^a = d_u^h, t_a^s = \text{worst}(s_u^l, s_u^h)$ AND $D_o^v = \text{worst}(d_u^h, d_v^h), t_v^s = s_v^h$
Min
$D_o^a = d_u^l, t_a^s = s_u^l$ AND $D_o^v = \text{best}(d_u^l, d_v^l), t_v^s = s_v^h$
Figure 3(e)
Max
$D_o^a = d_u^h, t_a^s = \text{worst}(s_u^l, s_u^h)$ AND $D_o^v = \text{worst}(d_v^h, d_v^l), t_v^s = s_v^h$
Min
$D_o^a = d_u^h, t_a^s = s_u^h$ AND $D_o^v = d_u^l, t_v^s = s_v^h$
Figure 3(f)
Max
$D_o^a = d_u^l, t_a^s = s_u^l$ AND $D_o^v = d_v^h, t_v^s = s_v^h$
Min
$D_o^a = d_u^l, t_a^s = s_u^l$ AND $D_o^v = \text{best}(d_v^l, d_v^h), t_v^s = s_v^h$

Fig. 5. Parameter selection for coupling factor computation

III. Efficient iteration mechanism

Static timing analysis considering coupling is an iterative approach. The efficiency of the current iterative approaches can increase greatly if the number of iterations are reduced. Commonly, timing analysis is performed in a topological order of the circuit. As shown in Figure 6, this approach updates the timing information in a way such that any update at *d* will be propagated to *e, f, g* and *h*. If the update at *d* is not permanent, those propagations will be overwritten later, and previous calculations are thus wasted.

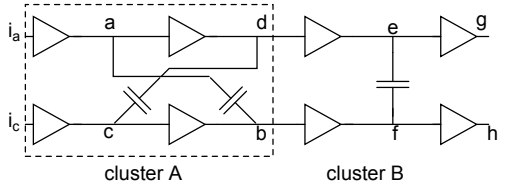

Fig. 6. Exploit circuit structure by clustering

Zhou [8] revealed that as long as we are using the same coupling delay model, the process will always converge to the same fixpoint no

matter what iterative order is adopted (termed *chaotic iteration* [9]). This result forms the basis for finding an efficient ordering scheme to do the iteration for fast convergence.

A. Clustering

Treating coupling edges as bidirectional edges we can identify strongly connected components in the graph with both circuit fanout edges and coupling edges as clusters. Processing each cluster till convergence in the topological order among clusters will trim off the wasted calculations. As shown in Figure 6 clusters A and B have different structures. B is simpler as any change in the window at e will only affect the windows at f through the coupling. We call it a *local cluster*. However cluster A has a different structure. Any change in window at a will affect the window at b due to coupling and window at d through fanout. Now since the change in window at d can affect window at c which in turn can affect a gives rise to a feedback mechanism in A. In essence cluster A includes two local clusters interacting with each other. We call such a cluster as *global cluster*. Solving the timing analysis with crosstalk is then reduced to finding a convergent solution on a set of local and global clusters.

B. Coupling edge as feedback

If we always compute local clusters together [7], [10], [11], then we need to select a set of gate inputs as feedback edges, whose removal makes the structure acyclic. We do timing analysis in the acyclic part and if it changes the values on feedback edges, we start the next iteration. Given a general graph G corresponding to a combinational circuit which includes the coupling and fanout edges, finding the feedback edges helps in reducing the number of iterations to reach the stable values of switching windows. Although there is no direct relation between the number of iterations and number of feedback edges, fewer feedback edges will give fewer possible value changes. However finding the smallest number of feedbacks is NP-hard on a general graph [12].

A good heuristic which we used to identify the feedback edges is to use the coupling edges as the feedback. This is based on the fact that if all the feedbacks are gate inputs, any 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 timing value on an input always influences timing value on an output. On the other hand, coupling relation is complex because it is bidirectional, and it is weak because timing value on one net may not always influence the value on the other (if they will not switch at the same time).

We can identify all feedback edges as the *Global* coupling edges. That is because the change of window on feedback edges will result in beginning the iteration again. Once we can identify all feedback edges, i.e the edges whose nodes have a fanin relation between them, we can delete all such edges. Global coupling edge Identification begins with considering all the coupling edges C potential candidates for global couplings. The algorithm selects each edge (u, v) out of the set C and see whether there is a path from vertex u to vertex v. A breadth-first-search [13] is started from the vertex u and we insert the edge (u, v) into the global cluster if vertex v is reached through the search. We call the coupling edges identified by the BFS based algorithm as *primary* edges of C^G. We call the set of *primary* edges as C_P^G. C_P^G represents all feedback edges as discussed earlier.

C. Coupling weight assignment

We assign an weight to each edge $e \in C - C_P^G$ where weight represents how closely the nodes forming coupling edge e are coupled with each other. To determine the weight, we calculate the initial rise and fall delay switching windows on each node of the graph G. Rise and Fall switching windows are special cases of generalized switching windows if the nature of the signals are not considered. For the initial switching windows we consider the capacitance as the sum of gate capacitance and the intrinsic coupling capacitance which is equivalent to considering MCF as 1 as evident from Equation 1. Based on the initial switching windows we calculate the overlap ratio k as described in Definition 5 for each coupling edge $e \in C - C_P^G$ and k is assigned as the weight to each edge e.

Definition 5 is an indicator of proximity between two switching windows. A high overlap ratio between nodes u, v predicts that the nodes are closely coupled with each other and change in the switching window of one node will surely lead to the change in switching window of other node. The intuition governing this observation is that if two switching windows are closely coupled with each other with MCF as 1 (considering intrinsic coupling capacitance without switching effects), they are most likely to remain closely coupled when MCF is varied due to switching effects. Overlap-ratio by definition is a probabilistic quantity which can vary between 0 and 1 and it captures the probability of two nodes to be closely coupled. Also if the overlap ratio between the nodes u, v of some coupling edge is high, it becomes a potential candidate for local coupling edges over some other coupling edge whose nodes w, x do not have a high overlap ratio.

D. Coupling partitioning

The set $C - C_P^G$ are now left with edges as shown in Section III-B. The coupling edge (e, f) has an important property. Timing information on e can be affected only because of the coupling with f. We characterize such coupling edges as *Local*. *Local Coupling Edges* C^L is a subset of edges C such that $\forall (u, v) \in C^L$ any change in rise and fall delay window at node u does not propagate to node v through the fanouts of node u. In other words, only coupling edge between u, v can change the windows on the respective gates. If we take a look at edges (a, b) and (c, d) from Figure 6, both the edges can be potential candidate for *Local Coupling Edges*. If we arbitrarily select any one of them as *Local*, it will force the participating nodes of other coupling edge related by the fanout relation. This is because once we choose one edge as *Local* we must update them simultaneously. It in-turn forces the two nodes to be considered as one super node in the graph G.

Formation of such a super node change the fanout relation of the original graph. If we choose (a, b) as local coupling edge, we need to look at them as a single node $a - b$, and therefore it will force nodes c and d to be related by fanout relation. We call (c, d) global coupling only if (a, b) is considered as local. *Global Coupling Edges* C^G is a subset of edges C such that $\forall (u, v) \in C^G$ vertex u is related to vertex v by fanin relation i.e there exists a path from u to v in G^F or $\hat{G^F}$. $\hat{G^F}$ is obtained by a transformation on graph G^F. The transformation forms a super node in the original graph by choosing some couplings as local, and thus changes the fanin relation of the whole graph.

Problem 1: Coupling Partitioning is defined as to identify feedback edges $e \in C - C_P^G$ such that the sum of coupling weights on those edges is minimized.

Due to interdependence of edges in $C - C_P^G$, identifying minimal coupling weighted feedback edges is not straight-forward. In fact we

Algorithm: Identify-NonPrimary-Global-Coupling

```
Input:Coupling Edge (u,v), G^F, G^C
Output:NonPrimary Global Coupling(NPGC)
begin
  NPGC = ∅
  Generate Ḡ^F: Complement of G^F
  Collect Ancestor(u) from Ḡ^F
  Collect Descendent(v) from G^F
  while(∃e ∈ C : e = (i,j), i ∈ Ancestor(u),
          j ∈ Descendent(v))
    NPGC←NPGC ∪ e
  Collect Descendent(u) from G^F
  Collect Ancestor(v) from Ḡ^F
  while(∃e ∈ C : e = (i,j), i ∈ Descendent(u),
          j ∈ Ancestor(v)
    NPGC←NPGC ∪ e
end
```

Fig. 7. Identify nonprimary global coupling

can give the following theorem

Theorem 2: Identification of feedback edges $e \in C - C_P^G$ such that the sum of coupling weights on the edges is minimized is NP-complete.

To solve the problem for general circuits we use the following heuristic. We sort the coupling edges $C - C_P^G$ by decreasing overlap-ratio. We select the edge with maximum overlap-ratio and insert the edge into C^L. Now as discussed above choosing the edge as a member of local coupling will change the structure of the circuit and might generate edges for C_{NP}^G , where C_{NP}^G is the set of non primary global edges. Non primary global edges are basically those feedback edges on which we are minimizing the sum of coupling weights. We identify such edges by the algorithm presented in Figure 7. Following that we remove the selected edge and the edges generated by the algorithm from coupling edges. We choose the next maximum overlap-ratio from remaining coupling edges and repeat the procedure until all the edges of $C - C_P^G$ are considered. The algorithm takes as input the coupling edge e which denotes the coupling between nodes u and v. We define relations *Ancestor* and *Descendent*.

Definition 6: Ancestor of a node u is the set of nodes, such that there exist a simple path between each node of the set to the node u traversing the fan-in edges of the graph G.

Definition 7: Descendent of a node u is the set of nodes, such that there exist a simple path from node u to each node of the set traversing the fan-in edges of the graph G.

As a designer's point of view, Ancestor and Descendent set of a node u are fanin and fanout cone of the u. The algorithm shown in Figure 7. On each node $v \in G^F$ list of all the edges which fans out from node v is kept. By Definition 7, G^F is ideal to find the descendents of a given node v by traversing the graph identifying which all nodes can be reached through the fanout edges from node v. Representation as G^F is not ideal for getting ancestors(by Definition 6. Therefore from G^F we generate the complement $\overline{G^F}$. $\overline{G^F}$ is obtained by reversing the direction of each edge of G^F and thus keeping the information on each node $v \in \overline{G^F}$ list of all edges which fan-in to node v. Beginning a traversal at node v all ancestors of the node can be obtained by the list at each node in the complement graph $\overline{G^F}$.

Using the edge list information at each nodes of G^F and $\overline{G^F}$, generating the sets Ancestor and Descendent are straightforward. Ancestor(u) and Descendent(u) respectively denotes the sets of Ancestor and Descendent corresponding to the node u. The collection of nodes is done by a BFS-based routine [13] that looks at the top of the queue element, pops it up and stores it in another list called Ancestor or Descendent depending upon the graph on which the routine is called. We are basically seeking for the elements of NPGC which is initially set as $\emptyset$. If there is an coupling edge (i, j) where i belongs to the Ancestor set of u and j belongs to the Descendent set of v, then merging the nodes i, j into $i - j$ will make the coupling edge (i, j) a member of non-primary global coupling. We identify all such edges in the *while* loop of the algorithm and store them in the set NPGC. Elements of set NPGC is added to C_{NP}^G and following that we remove the appropriate elements from E^C as discussed before. C^G is crucial for iteraation algorithm as any change in the timing window of one node related to the global coupling edge will eventually propagate to the other node and we can prune iterations based on this fact.

Algorithm: Fast-Chaotic-Iterator

```
Input:General Graph G
Output:Timing Windows with Coupling
     Capacitance Effect
begin
Procedure:Initialization
 Node-Queue← ∅
 while(∃u ∈ V : u ∉ Node-Queue)
   UpdateEffectiveCapacitance(u,1)
   ComputeWindows(u)
   Node-Queue←Node-Queue∪u
Procedure:Modified Chaotic Iterations
 while(Node-Queue≠ ∅)
   u = Pop(Node-Queue)
   while(∃v ∈ C : u → v)
     k^u ← ComputeCouplingFactors(u,v)
     UpdateEffectiveCapacitance(u,k^u)
   ComputeWindows(u)
   if(Timing Windows on u Changes)
     while(∃v ∈ F : u → v)
       Insert v to Node-Queue uniquely
     while(∃v ∈ C : u → v)
       if(v ∉ C^G)
         Insert v to Node-Queue uniquely
end
```

Fig. 8. Fast chaotic iterator

E. Iteration algorithm

The algorithm we present in Figure 8 is based on the theoretical foundation established in [8]. We call this algorithm *Fast chaotic iterator* since the iterative process the algorithm uses is known as *chaotic iteration* [9]. Our algorithm decides the timing propagation order based on coupling edge partitioning. First of all a timing iteration considering intrinsic coupling capacitance on all nodes is done. Identification of global and local couplings in Section III-D, is used to change the order in which the nodes are added to the queue. Global couplings are all those feedback edges, whose deletion will make the general graph G a DAG if the local couplings are considered as a single updating unit. We can process the graph in a topological order once the feedback edges are broken and we need to start the

timing iteration again only if the windows at the feedback edges change.

We introduce two hypothetical nodes PO and PI. In the generated timing graph, PI is connected to inputs of the circuit while all outputs are connected to PO. All the nodes are sorted by topological order. We start the iterative scheme by getting the topmost node of the queue. Now we compute the coupling factors considering that particular node as victim and the rest as aggressors. Detailed description of coupling factor computation is given in Section II. Accordingly we update the effective capacitance on the victim node which comes from the sum of the gate , interconnect and coupling capacitance multiplied by the factors computed before hand. If the timing windows on the victim node are changed, then we need to process all its local aggressors. Given vertices u, v, $u \rightarrow v$ means that a path exists from vertex u to v in graph G. Note that ComputeWindows function is the most costly operation of the algorithm as it computes the new windows from the input windows at the fanin edges to the victim node. ComputeCouplingFactor takes a victim and aggressor node, and on the basis of their rise delay, fall delay, rise slew and fall slew windows, generates the effective coupling factors. To compare the performance of our algorithm, we developed a general iterative algorithm which does not consider coupling partitioning.

IV. Experimental setup

A. Circuit modeling

We model a given circuit as a directed acyclic graph (referred to subsequently as the circuit's *timing graph*). Nodes in the timing graph represent gates in the circuit while the edges represent the corresponding interconnects. We map all nodes to logic gates from the Faraday 90nm technology library. Delay models are available from look-up tables that yield a gate's delay and slew as functions of its input slew and load (output capacitance). Extracted coupling capacitance values are used to generate a *coupling graph* that denotes the timing dependencies introduced due to coupled nets. This coupling graph is then superimposed on the timing graph. We consider that each net is coupled to at most four other nets. However, this is not a limitation in our approach.

B. Results

We present two set of results. Table I gives the accuracy of our coupling model over a discrete model based on MCFs 0,1,2 on ISCAS85 benchmarks [14]. CE denotes number of coupling edges, RT shows runtime in seconds and TA is the number of gate-delay look-up table accesses. (rd_l, rd_h) shows the rise delay window at PO. Since in our coupling model the best coupling factor can vary between -1 and 1 while the worst can vary between 1 and 3, the final windows obtained by MCFs 0,1,2 are conservative in some cases whereas non-conservative in other cases. Therefore as physically coupling capacitance can indeed be as large as $3\times C_c$ and as small as $-1\times C_c$, designing with coupling factors 0,1,2 can lead to large erros. We used an iterative algorithm [6] and used the two coupling models to obtain the results.

Performance enhancement results of our efficient iteration scheme is shown in Table II. For these experiments, we used the proposed coupling model but changed the solver algorithm. CI denotes the chaotic iterator (iterative algorithm based on [6]) and we represent our algorithm by *Fast-CI*. Number of global edges obtained by our algorithm is shown an $Global$. $P-RT$ shows the run-time of coupling edge partitioning. To demonstrate the speed-up obtained from our approach, we define the following.

$$\%Speedup = \frac{TA^{FastCI} - TA^{CI}}{TA^{CI}} \times 100.0$$

$\%Speedup$ denotes the percentage of gate-delay table-looks saved using our approach in comparison to CI. We choose this metric since the table-lookup operations are run-time expensive operations in timing analysis. From the table, we observe that our approach reduces table accesses on an average by 26.8%, and by 62.1% in the best case. The error in timing windows are negligible. We also made an interesting observation for circuit c5315. We didn't get any speed-up over the iterative approach. The reason for the same is attributed to the number of global coupling edges being much less than the number of local coupling edges. A run of coupling partitioning beforehand is necessary to realize whether is circuit is dominated by local couplings. In such a case (as in c5315) the circuit can be best solved by using the iterative approach rather than our proposed approach. But in other cases we had significant speed-up over the iterative approach. This demonstrates the significance of coupling edge partitioning in decreasing cell table accesses. Results shown are for experiments performed on a Pentium 2.4GHz processor server having 1Gb RAM and running RedHat Linux 9.0

V. Conclusions and future work

We present a framework for fast and accurate static timing analysis considering coupling. A novel coupling delay model is developed and a technique to speed up required iterations in timing analysis is proposed. Experimental results on the ISCAS benchmarks demonstrate the accuracy of our model over discrete coupling factor based model. We also demonstrate speed-ups of up to 62.1%, and average speed-ups by 26.8% using our efficient iteration scheme with negligible error in timing windows. It is thus evident that our framework improves both the accuracy and the efficiency of a timer significantly. In the future, we will study more accurate and efficient approaches for faster timing analysis considering coupling.

VI. Acknowledgment

This work is supported by a grant from Intel Corporation. The first author would also like to thank Faraday Corporation for providing their experimental 90 nm ASIC cell libraries.

References

[1] R. Kumar, "Interconnect and noise immunity design for the Pentium 4 processor," in *Proc. of the Design Automation Conf.*, 2003, pp. 938–943.

[2] 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.

[3] A. Kahng, S. Muddu, and E. Sarto, "On switch factor based analysis of coupled RC interconnects," in *Proc. of the Design Automation Conf.*, 2000, pp. 79–84.

[4] 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, Nov. 2000.

[5] M. Ghoneima and Y. Ismail, "Accurate decoupling of coupled on-chip buses," in *Proc. Intl. Symposium on Circuits and Systems*, 2005, pp. 4146–4149.

[6] 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, Nov. 2000.

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

[8] H. Zhou, "Timing analysis with crosstalk is a fixpoint on a complete lattice," in *IEEE Transactions on Computer-Aided Design, September 2003*, pp. 1261–1269.

TABLE I

MODEL ACCURACY RESULTS

Circuit	# of Nodes	# of CE	012 Model			Proposed Model		
			RT(s)	TA	(rd_l, rd_h)	RT(s)	TA	(rd_l, rd_h)
c432	198	277	0.05	1197	(0.02,24.43)	0.13	3004	(0.32,31.69)
c499	245	312	0.07	1206	(0.07,26.93)	0.28	4530	(0.02,19.15)
c880	445	620	0.22	4440	(0.16,26.98)	0.45	11595	(0.03,37.62)
c1355	589	829	0.23	6470	(0.19,31.01)	0.96	24291	(0.56,35,57)
c1908	915	1332	0.46	13009	(0.20,49.40)	1.75	45716	(0.22,53.93)
c2670	1428	1929	0.36	10094	(0.01,55.58)	1.35	34165	(0.01,50.41)
c3540	1721	2545	0.45	12822	(0.14,78.20)	2.28	60010	(0.32,71.17)
c5315	2487	3542	0.68	19467	(0.03,53.12)	3.74	67904	(0.01,60.08)
c6288	2450	3622	2.74	47135	(0.15,158.77)	15.48	246879	(0.38,167.67)
c7552	3721	5412	1.38	36887	(0.01,55.88)	6.71	127572	(0.01,64.67)

TABLE II

PERFORMANCE ENHANCEMENT RESULTS

Circuit	# of CE	CI		Fast-CI				
		TA	(rd_l, rd_h)	Global	TA	(rd_l, rd_h)	%Speedup	P-RT (s)
c432	277	3004	(0.32,31.69)	195	2831	(0.32,31.16)	5.7	0.04
c499	312	4530	(0.02,19.15)	176	3437	(0.02,19.64)	24.1	0.08
c880	620	11595	(0.03,37.62)	243	7289	(0.03,37.18)	37.1	0.13
c1355	829	24291	(0.56,35,57)	532	9207	(0.52,35.08)	62.1	0.19
c1908	1332	45716	(0.22,53.93)	679	27856	(0.20,53.30)	39.1	0.49
c2670	1929	34165	(0.01,50.41)	550	31474	(0.01,49.95)	7.8	1.21
c3540	2545	60010	(0.32,71.17)	973	44690	(0.32,69.33)	25.5	2.39
c6288	3622	246879	(0.38,167.67)	2503	163877	(0.21,165.48)	33.6	7.19
c7552	5412	127572	(0.01,64.67)	1388	119166	(0.01,63.14)	6.6	6.09

[9] 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*, Los Angeles, CA, Jan. 1977, pp. 238–252.

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

[11] 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*, San Jose, CA, Nov. 1998, pp. 212–219.

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

[13] T. H. Cormen, C. E. Leiserson, and R. H. Rivest, *Introduction to Algorithms*. MIT Press, 1989.

[14] F. Brglez and H. Fujiwara, "A neutral netlist of 10 combinatorial benchmark circuits," in *Proc. Intl. Symposium on Circuits and Systems*, 1985, pp. 695–698.