Switching Window Computation for Static Timing Analysis in Presence of Crosstalk Noise

Pinhong Chen
Dept. of EECS
U. C. Berkeley
Berkeley, CA 94720, USA
pinhong@eecs.berkeley.edu

Desmond A. Kirkpatrick
Intel Corp.
Microprocessor Products Group
Hillsboro, OR 97124, USA
desmond.a.kirkpatrick@intel.com

Kurt Keutzer
Dept. of EECS
U. C. Berkeley
Berkeley, CA 94720, USA
keutzer@eecs.berkeley.edu

Abstract

Crosstalk effect is crucial for timing analysis in very deep submicron design. In this paper, we present and compare multiple scheduling algorithms to compute switching windows for static timing analysis in presence of crosstalk noise. We also introduce an efficient technique to evaluate the worst case alignment of multiple aggressors.

1 Introduction

As chip design enters the very deep submicron(**VDSM**) realm, decreasing feature sizes increases the significance of coupling effects. In this realm, these subtle effects may be no longer ignored as they may affect the timing and signal integrity of the design. In the worst case, a design may be obviated by very subtle coupling problems.

Crosstalk noise affects design integrity in two ways: one is timing deterioration and the other is signal integrity. In this paper, we concentrate on how coupling noise affects timing by studying how to capture coupling delay effects in static timing analysis.

Traditional static timing analysis(**STA**) only considers cell and interconnect delay, searching for the longest or the shortest path to assess the most critical timing. Capacitive load is simply considered nominally: to a given node, each adjacent node is considered quiescent, making no transition when the node is transitioning. However, active coupling to a switching node may result in additional delay or reduced delay on that node depending on the direction of adjacent switching[1, 2].

If both nodes are switching in the same direction, the delay on both nodes is reduced, whereas if they switch in the opposite directions, the delay is increased.

It is possible to make the worst case assumption that the twice the coupling capacitance is used to capture this opposite-direction coupling effect, forming a decoupled version of the circuit for each node, where capacitances are replaced by their Miller equivalent. However, this approach can limit the design space dramatically and lead to a very pessimistic, impractical design.

Static timing analysis has been studied for more than a decade[3, 4, 5]. However, no crosstalk coupling is involved in those analyses. Recently, in [6], the authors provide a design methodology to avoid coupling noise and address static analysis of noise on the transistor level. In [7], the author analyzes the functional aspect of how signals can couple together. In [8], a formulation is proposed to calculate the maximum noise, but it can only apply to small circuits due to its complexity. In [2], an algorithm to calculate the worst case aggressor alignment due to coupling is proposed. [1] shows the Miller factor can be more than 2X for the upper bound of the maximum coupling delay. Furthermore, in [9, 10], the authors show that the bound is 3X instead of 2X. For a design in VDSM, the functional aspect of crosstalk coupling is almost impractical to analyze. STA with crosstalk coupling effects serves as a very practical and efficient way to verify a circuit design not to violate any timing constraints[11, 12, 13]. However, neither scheduling techniques nor convergence issues have been reported in [11, 12].

In this paper, we address the problem of static timing analysis considering coupling effects. Unlike traditional STA, the critical path delay cannot be obtained simply by topological traversal. **Switching windows**, within which a node makes transitions, are the key to determine whether the coupling noise can affect timing. Only when two coupling nodes have overlapping switching windows can their timing may change due to their coupling. However, switching windows depend on the signal timing itself, so we have a circular problem to resolve. We propose an event-driven calculation algorithm to solve this mutual dependency problem, resolving cycles through causality. We assume a single worst case driver resistance and apply superposition of waveforms extensively. This is a conservative assumption and the result is an upper bound of the actual switching window.

The rest of this paper is organized as follows: we first review necessary background and definitions in Section 2. In Section 3, we discuss the alignment of multiple aggressors for worst-case delay. Section 4 presents an event-driven algorithm in detail, including proof, event scheduling techniques, complexity analysis, and efficiency issues. Experimental results are shown in Section 5.

2 Background and Definitions

For a pair of coupling nodes, the node which suffers from the coupling noise is called a **victim node**, and the other node that contributes the noise is called an **aggressor node**. They can change their roles depending on the context of which is calculated for timing analysis. The **worst case delay** of a node, shown in Fig.1, is the minimum or maximum delay considering all topological minimum or maximum delay dependency and the worst case crosstalk coupling. For example, the worst case timing may be computed using zero coupling capacitance for min delay and 3X coupling capacitance for max delay, respectively. The **nominal delay** of a node is defined as the delay calculated when each aggressor is quiet, that is, using 1X coupling capacitance for delay calculation. The worst case switching window thus forms the outer bound of the actual switching window and the nominal case switching window forms the inner bound.

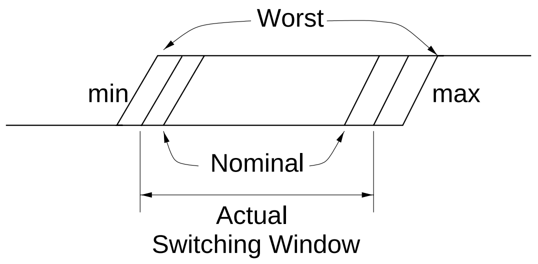

Figure 1: Min/Max Timing

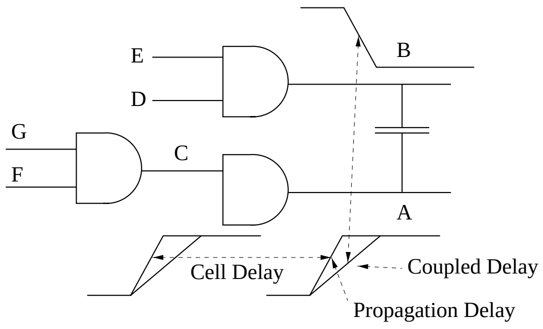

Figure 2: Propagation/Coupled Delay

The **coupled delay** of a node, shown in Fig.2, is the delay due to the aggressors' coupling of the node. It is computed from the aggressor's maximum coupling noise to achieve the min or max delay. The **propagation delay** of a node(Fig.2) is the delay due to the previous stage delay. It is computed from the previous stage coupled delay plus the cell delay.

Given a node i, the min propagation delay is

$$t_i^{ppg,min} = \min_{j \in FI(i)} \{d_{j,i}^{min} + t_j^{cpl,min}\} \quad (1)$$

and the max propagation delay is

$$t_i^{ppg,max} = \max_{j \in FI(i)} \{d_{j,i}^{max} + t_j^{cpl,max}\} \quad (2)$$

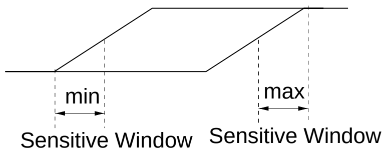

Figure 3: Sensitive Min/Max Windows

,where $t_i^{ppg,min}(t_i^{ppg,max})$ denotes the min(max) propagation delay, $FI(i)$ is the set of fanin of node i, $d_{j,i}^{min}(d_{j,i}^{max})$ is the min(max) node-to-node delay, and $t_j^{cpl,min}(t_j^{cpl,max})$ denotes the min(max) coupled delay of node j.

Suppose the driving resistance of the aggressor is linear. Coupled delay can be calculated using superposition of the victim waveform and the coupling noise waveforms from the aggressor nodes.

The **min sensitive window**, shown in Fig.3, of a node for the min delay is the duration starting from the rising point to the switching threshold point of the transition. In this duration, the coupling noise may speed up the transition and the delay may be reduced. This duration is used to determine the min delay variation because the possible range for a signal to be sped up is just within this window. Similarly, the **max sensitive window** of a node for the max delay is from the switching threshold point to the end point of the transition. In this duration, the coupling noise may slow down the transition and the delay may be increased.

2.1 Piecewise Linear Waveform

For ease of calculation, we assume piecewise linear waveforms. The number of linear segments can be used to trade accuracy and run time for modeling waveforms from circuit level characterization. Moreover, they can be easily manipulated to do waveform superposition and to compute an envelope waveform, which will be described in Section 4. All of these computations are proportional to the number of linear segments in the waveforms.

3 Multiple Aggressor Alignment Problem

In this section, we will discuss how to determine the worst case alignment given multiple aggressor waveforms and a victim waveform. The problem is to align aggressor waveforms so as to achieve the maximum or the minimum delay on the victim node. Because each node has a switching window, in which a node can possibly make transitions, these switching windows restrict the range where the worst case alignment can occur. In [2], the authors prove that the worst case delay for a pair of aggressor and victim nodes occurs when the peak noise aligns up to the switching threshold of the victim. Based on their result, we address the case when multiple aggressors are aligned, which is common for any STA scenario. We propose an envelope waveform to perform this computation, something relatively easy to compute, where the complexity is simply proportional to the number of linear segments in all the waveforms.

In physical layout, there may be several aggressors coupling to a victim node. Each of them is constrained by some switching window.

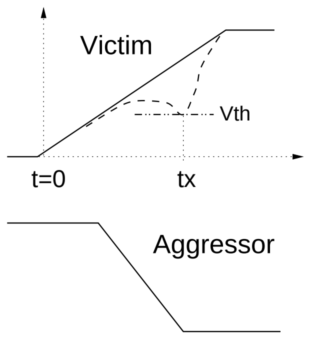

Figure 4: Maximum Delay under Coupling Effect

Consider the waveforms of 2 nodes switching at the opposite directions shown in Fig. 4. The problem of finding the delay due to noise effect is equivalent to sliding or convolving the aggressor waveform subject to the switching window constraint to achieve the maximum delay on the victim node. Specifically, we have to find the scenario which maximizes t_x point in Fig.4[2]. At that point, the waveform of victim touches the switching threshold point of the next stage logic gate, making a sharp transition. If the coupling noise waveform slides continuously from the left bound of the aggressor switching window to the right bound, which is shown in Fig.5, this waveform envelope forms a range and magnitude of noise that could possibly affect the victim waveform. After superposition of this envelope and the victim waveform, the resulting waveform is the worst case waveform envelope of the victim node. The worst case delay can be found on the last point crossing the switching threshold (usually 0.5 Vdd). The bold lines in Fig.5 show the noise peak and the corresponding aggressor transition to create this worst case timing.

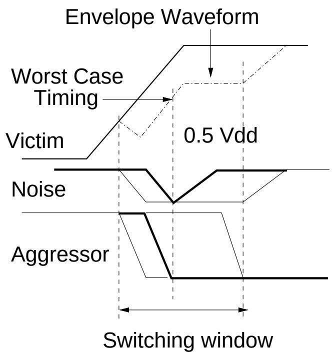

Figure 5: Sliding Noise and Envelope Waveform

Theorem 1 *The technique described above can find the worst case alignment which creates the worst case delay on a victim node, given the switching window constraints of multiple aggressors.*

(Proof:) The victim envelope waveform actually depicts the minimum voltage values that the victim waveform can possibly reach over time. Due to the superposition assumption, we can superpose each aggressor envelope waveform on the victim envelope waveform one by one. The resulting waveform envelope is the final worst case voltage it can reach over time. By tracing this envelope waveform, the worst case delay can be obtained. □

4 Coupling Delay Computation in Presence of Crosstalk Noise

4.1 Algorithm

In today's technology, RC delay calculation consumes a major portion of the total computation time for delay calculation. Typical RC delay calculation algorithm involves effective capacitance computation[14] and model order reduction of the RC interconnect[15]. Cell delay computation is a relatively simple computation often via a table lookup. Finally, waveform superposition is another complexity that adds to the whole coupling computation. Therefore, our algorithm is optimized toward reducing the number of coupling computations.

There are two types of events in our event-driven algorithm. A **coupling event** is the event triggering calculation of the coupling waveform envelope based on the victim and the aggressors' waveforms to derive the coupled delay. A **driving event** is the event triggering calculation of the propagation delay based on the previous stages' coupled delay.

Given a circuit with the coupling noise for each pair of victim and aggressor, and the waveform that has been characterized, we propose the following event-driven algorithm to compute effective circuit delay:

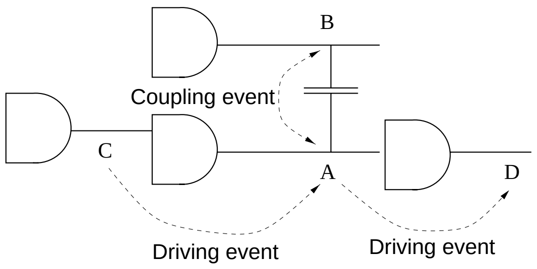

Figure 6: Coupling/Driving Events

1. Schedule a coupling event for each node.
2. Pop an earliest event until the queue is empty according to the current status of the circuit.
   (a) If it is a coupling event, for example, a coupling event from node B to node A in Fig.6, compute the superposition coupling waveform and get the coupled delay of node A. Schedule the next stage driving event for node D.
   (b) If it is a driving event from node C to node A, update the propagation delay of the current node A,

schedule a driving event from node A to node D, force a coupling event on node A to recompute the coupling effect, and find if node A newly attacks the adjacent nodes. Schedule the coupling events for newly attacked adjacent nodes, that is to say, for example, a coupling event from node A to node B.

Our algorithm keeps track of all old delay values. If a coupling or a driving condition does not change, it won't be necessary to recompute or schedule an event.

Note that not only the victim can have delay variation due to noise effect, but also the aggressor has delay variation. The event-driven algorithm proposed above will check each node to compute the coupled delay based on the propagation delay and coupling effect, and propagate extra delay forward the stages. That is to say, depending on the context, each node will be considered as a victim to update the delay value.

4.2 Convergence of Our Algorithm

Intuitively, our algorithm tries to maintain a consistent system that each node has its propagation min and propagation max delays as defined by Eq.(1) and (2) and the coupled delay conforms to the worst case alignment of its aggressor waveforms as described in Section 4. Our algorithm corrects the local inconsistency of delays and issues the related delay perturbation event to the next stages or the adjacent coupling delays.

Theorem 2 *Given an accuracy requirement, the algorithm described above converges to a consistent value for each delay in a circuit using finite steps.*

(Proof:) If there is a coupled delay inconsistency or a driving delay inconsistency on a node, our algorithm recomputes it according to the coupling nodes or the incoming driving delays by the algorithm described in Section 4 and Eq.(1) or (2), to update its coupled or propagation delay, and issue events for updating related coupling nodes and the next stage nodes. This maintains local consistency. Note that we assume each isolated sub-circuit group has at least one input to initiate the event-driven process. Initially, we assume that the propagation delay for each primary input is fixed.

We now prove the convergence of our algorithm. Supposing there is no coupling, we can compute the propagation delays in a topological order in one single pass. However, due to crosstalk coupling, a victim node may have coupling from its transitive fanouts whose switching windows cannot be finalized at the time when we calculate the propagation delay of the victim. Suppose one of the aggressor nodes j is a transitive fanout of a victim node i, and their switching windows overlap each other. Fig.7 shows these waveforms. We will prove this converges to a single point. As aggressor j's transition moves from left to right, we can plot $t_i^{cpl,min}$ as a function of $t_j^{ppg,min}$, which is shown in Fig.8, where T is a shorthand for $t_i^{ppg,min}$, h_j is the normalized peak noise coupling from aggressor j, $d^*_{i,j}$

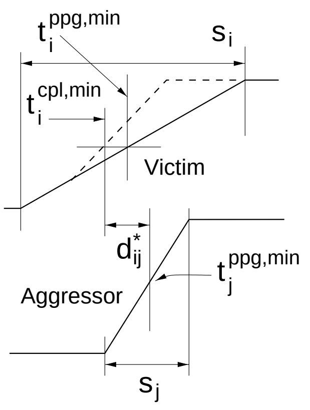

Figure 7: Transitive Fanout as an Aggressor

is a transitive delay from node i to node j, and the propagation delay of node j is equal to

$$t_j^{ppg,min} = t_i^{cpl,min} + d^*_{i,j}. \tag{3}$$

The function means that it can be sped up to the lower bound of $T - s_i h_j$ and it has an upper bound of T, which is no speed-up. In addition, Eq. (3) has a lower bound when $d^*_{i,j}$ equals to zero. The convergence process is shown in Fig.9. If the first $t_i^{cpl,min}$ is at point a, the value of $t_j^{ppg,min}$(point b) can be obtained by Eq. (3). The value of $t_i^{cpl,min}$ is thus obtained by the $t_i^{cpl,min}$ function shown in Fig.8, which is point c. It will continue this process to point d, e, until it converges to the crossing point z, at which it meets the accuracy requirement.

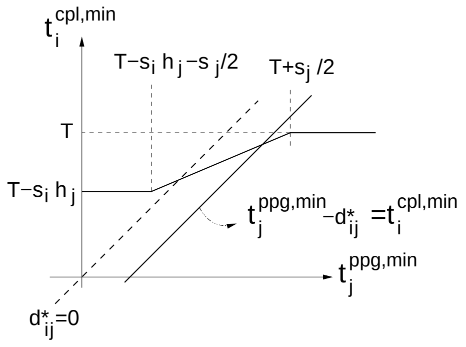

Figure 8: $t_i^{cpl,min}$ Function of $t_j^{ppg,min}$ and convergence

Moreover, the slope of middle segment of $t_i^{cpl,min}$ function can be shown to be greater than 1, which means we can have only one crossing point since Eq. (3) is also linear. As a result, the iteration process must be able to improve towards convergence, which means that we can reach the accuracy requirement on finite steps. □

Note that the iteration occurs when the aggressor's sensitive window overlaps with the victim's switching threshold point, and the aggressor is one of the transitive fanouts of the victim. As the transitive delay is shorter and the aggressor's slew time

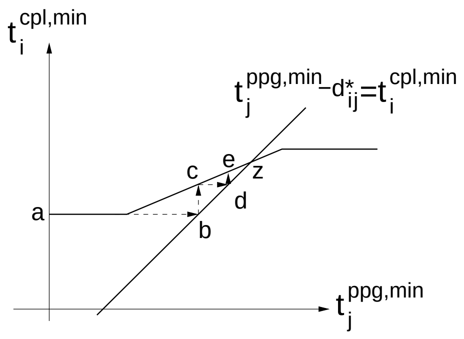

Figure 9: Convergence of $t_i^{cpl,min}$ and $t_j^{ppg,min}$

is longer, it is likely to overlap and increase the computation time.

For the max delay, because of the conservative assumption that the aggressor may switch at any time point within the switching window to create the worst case coupling, the result of event-driven algorithm always takes the worst case timing, which is conservative and no iteration needed. The $t_i^{cpl,max}$ function has two types shown in Fig.4.2.

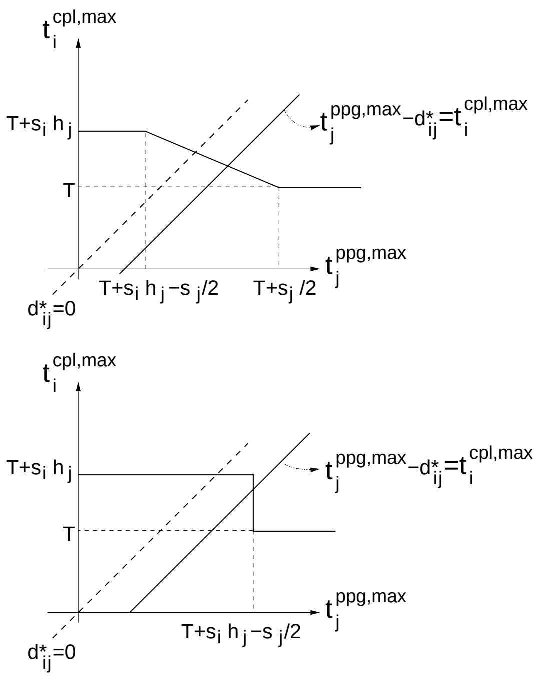

Figure 10: $t_i^{cpl,max}$ Function of $t_j^{ppg,max}$

4.3 Properties of Our Algorithm

It is interesting to note that our algorithm can reach the same result even if the initial values, propagation and coupled delays, are given totally different. That is to say, our algorithm results in a very robust calculation. Different initial value settings only affect the number of events and calculation time. Typically if the initial values are closer to the final result, the computation time is reduced. This principle generally applies to all the iterative algorithms to converge in fewer steps if the initial value are closer to the final convergence value.

4.4 Event Pruning

Any event may cause a series of computations to update the whole system. However, coupling and driving events may not be necessary if they will not change any delay value of a circuit. Therefore, it is desirable to reduce the number of events issued to speed up the computation.

Coupling may be considered harmless if two signals switch with non-overlapping timing windows. That is to say, due to the temporal isolation, the two nodes that are physically coupled result in no crosstalk noise effect. We can also compute the lower bound of min delay and upper bound of max delay by 0 or 3X coupling capacitance. It can be done before running the event-driven algorithm, and provides a valuable information for pruning events.

When a node changes its propagation delay, our algorithm issues events based on the facts:

1. if the coupling computation in some previous event may still keep the same condition for coupling, it does not need to schedule this event, since the same coupling condition results in the same amount of coupling noise.

2. when the coupling condition changes, a coupling event has to be issued to update the corresponding coupling nodes, and the propagation delay of next stage may change accordingly, so a driving event is issued.

The event-driven type of calculation makes the computation very robust and efficient without least redundant recomputation.

4.5 Scheduling Technique

Scheduling is a key for the efficiency of convergence. We can reduce complexity by an order of magnitude with careful arrangement of events. Among all the scheduling techniques, including topological order of the nodes, the event timing sequence, static or dynamic updating, and updating frequency, we identify several significant scheduling approaches as follows:

Dynamic Event Time We schedule events based on the right hand bound of a sensitive window defined in Section 2, and dynamically sort the events according to the current circuit status(delay value). Intuitively, it forms a sweep timing line across the circuit. If any event occurs earlier in the event time, our event algorithm schedules it first and keep iterating on its related events until it converges.

Static Event Time We schedule events based on the right hand bound of a sensitive window. No dynamic sorting of events is performed. When two events have the same time, we schedule events based on the topological order of the nodes.

Ckt.	#Nodes	#Edges	#Cpl. Comp.	Re-Comp.%	Run Time
C17	11	12	23	4.5%	0.005s
C432	196	336	424	8.2%	0.083s
C499	243	408	538	10.7%	0.099s
C880	443	729	1042	17.6%	0.200s
C1355	587	1064	1361	15.9%	0.276s
C1908	913	1498	2041	11.8%	0.441s
C2670	1350	2076	3082	14.1%	0.638s
C3540	1719	2939	3924	14.1%	0.880s
C5315	2485	4386	5477	10.2%	1.225s
C6288	2448	4800	7281	48.7%	1.350s
C7552	3718	6144	9220	24.0%	1.896s

Table 1: Result for ISCAS85 Combinational Circuits

Smart Global For each updating pass, we maintain a flag on each node to identify if the node needs to be updated in the next pass. At each pass, each node is examined and processed if necessary. The updated delay will propagate to its coupling nodes and next stages. The event pruning technique is also used to reduce the number of updates. If no update is needed through out a pass, it is converged.

The number of events strongly depends on the number of coupling edges and the number of propagation edges. It implies some preprocessing to prune to loosely coupling edges can be very effective.

5 Experimental Results

We demonstrate our algorithm on a 233MHz PC with 64M bytes memory based on a Linux OS. We benchmark our algorithm on the ISCAS85 combinational circuits. For each circuit, each node is presumed to have four random coupling nodes. The coupling noise between each pair of aggressor and victim and the slew on each node are pre-characterized or estimated. We also vary these parameters with different scheduling approaches to test the efficiency of our algorithm.

The total run time for all the ISCAS85 eleven combinational circuits is just 7.09 seconds using dynamic event time scheduling technique with a convergence accuracy of 10^{-8}ns. It is observed that on average, 21.9% of nodes are recomputed for coupling calculation, which means only 21.9% of the nodes have to be calculated twice for the coupling to obtain to the final delay value. Table 1 shows the result, where the first column is the name of circuits, the second column is the number of nodes, the third column is the total number of fanouts, which is equal to the number of driving edges, the fourth column is the number of coupling computation, the fifth column is the percentage of re-computation of coupling, and the last column is the run time.

With different initial values, the number of coupling computations can have 22% difference such as shown in Table 2, where the first column shows W factor, which is the factor how the initial value is close to the worst case value: 0.0 represents using the nominal delay value, and 1.0 represents using the worst case value for initial values. The second column is the total number of coupling computations of all eleven combinational circuits from ISCAS85. The third column is the percentage of re-computations, and the last column is the run time.

W factor	#Coupling Comp.	ReComp.%	Run Time
1.0	40528	43.6%	7.55s
0.8	37192	31.8%	7.31s
0.6	34467	22.1%	7.17s
0.5	34413	21.9%	7.09s
0.4	35080	24.3%	7.17s
0.2	37121	31.5%	7.21s
0.0	41828	48.2%	7.39s

Table 2: Initial values affects the number of coupling computations

Scheduling factor Method	#Coupling Comp.	Run Time
Smart Global	240292	96.8s
Dynamic Event Time	308048	113.2s
Static Event Time	286718	108.2s

Table 3: Performance for Different Scheduling Approaches

We also implement the ISCAS combinational circuits in a $0.25\mu m$ technology. The result is shown in Table 4, where the first column shows the circuit name, the second column is the number of nodes in the circuit, the third column is the number of propagation edges, the fourth column is the number of the coupling edges, the fifth column is the number of coupling computation, and the last column is the run time. In Table 3, we compare different scheduling approaches in terms of total run time for all these circuits. In gerenal, Smart Global scheduling approach is a winner among all the scheduling approaches.

6 Conclusion

We propose a robust and efficient algorithm to compute the coupling delay effect on static timing analysis using a flexible and practical waveform model. The convergence property has been shown as well as different scheduling techniques to reduce the run time. We expect this approach can be directly implemented in a very practical industrial tool for advanced static timing analysis targeting for very deep submicron designs.

References

[1] G. Yee, R. Chandra, V. Ganesan, and C. Sechen. “Wire Delay in the Presence of Crosstalk”. In *IEEE/ACM International Workshop on Timing Issues in the Specification and Synthesis of Digital Systems*, pages 170–175, 1997.

Ckt.	#Nodes	#Edges	#Cpl. Edges	#Cpl. Comp.	Run Time
c17	22	20	48	184	0.01s
c432	709	794	3049	4769	1.31s
c499	1239	1372	9757	5319	1.59s
c880	2382	2568	21026	12002	3.59s
c1355	1655	1796	13110	7204	2.15s
c1908	1353	1473	11113	6177	1.81s
c2670	2760	2870	30223	12719	5.58s
c3540	4650	5077	52341	22376	10.10s
c5315	6093	6647	63232	35871	12.74s
c6288	21153	24375	245889	109308	45.21s
c7552	8630	9445	83962	43846	20.54s

Table 4: Results for 0.25μm process implementation of ISCAS85 combinational circuits

[2] P. D. Gross, R. Arunachalam K. Rajagopal, and L. T. Pileggi. “Determination of Worst-Case Aggressor Alignment for Delay Calculation”. In *Proc. of International Conferences on Computer Aided Design*, pages 212–219, Nov. 1998.

[3] T. Sasao(ed). *“Logic Synthesis and Optimization, Ch.8: Delay Models and Exact Timing Analysis”*. Kluwer Academic Publishers, 1993.

[4] S. Devadas, K. Keutzer, S. Malik, and A. Wang. “Certified Timing Verification and the Transition Delay of a Logic Circuit”. *IEEE Trans. on Very Large Scale Integration Systems*, 2:333–342, Sep. 1994.

[5] S. Devadas, K. Keutzer, and S. Malik. “Computation of floating mode delay in combinational networks: Theory and algorithms”. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, 12:1913–1923, Dec. 1993.

[6] K. L. Shepard. “Design Methodologies for Noise in Digital Integrated Circuits”. In *Design Automation Conference*, pages 94–99, 1998.

[7] D. A. Kirkpatrick. *“The Implication of Deep Sub-micron Technology on the Design of High Performance Digital VLSI System”*. PhD thesis, CAD Group Ph.D. Dissertation, U.C. Berkeley, 1997.

[8] P. Chen and K. Keutzer. “Towards True Crosstalk Noise Analysis”. In *Proc. of International Conferences on Computer Aided Design*, pages 132–137, Nov. 1999.

[9] A. B. Kahng, S. Muddu, and E. Sarto. “On Switch Factor Based Analysis of Coupled RC Interconnects”. In *Design Automation Conference*, pages 79–84, 2000.

[10] P. Chen, D. A. Kirkpatrick, and K. Keutzer. “Miller Factor for Gate-Level Coupling Delay Calculation”. In *Proc. of International Conferences on Computer Aided Design*, 2000.

[11] B. Franzini, C. Forzan, D. Pandini, P. Scandolara, and A. D. Fabbro. “Crosstalk Aware Static Timing Analysis:a Two Step Approach”. In *IEEE of 1st International Symposium on Quality Electronic Design*, pages 499–503, Mar. 2000.

[12] P. F. Tehrani, S. W. Chyou, and U. Ekambaram. “Deep Sub-Micron Static Timing Analysis in Presence of Crosstalk”. In *IEEE of 1st International Symposium on Quality Electronic Design*, pages 505–512, Mar. 2000.

[13] R. Arunachalam, K. Rajagopal, and L. T. Pileggi. “TACO: Timing Analysis With COpling”. In *Design Automation Conference*, pages 266–269, 2000.

[14] J. Qian, S. Pullela, and L. T. Pillage. “Modeling the “Effective Capacitance” for the RC Interconnect of CMOS gates”. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, 13:1526–1535, Dec. 1994.

[15] K. L. Shepard, V. Narayanan, P.C. Elmendor, and Gutuan Zheng. “Global Harmony: Coupled Noise Analysis for Full-Chip RC Interconnect Network”. In *Proc. of International Conference on Computer Aided Design*, pages 139–146, 1997.