Short Papers

Postroute Gate Sizing for Crosstalk Noise Reduction

Murat R. Becer, David Blaauw, Ilan Algor, Rajendran Panda, Chanhee Oh, Vladimir Zolotov, and Ibrahim N. Hajj

***Abstract*—Gate sizing is a practical and a feasible crosstalk noise correction technique in the post route design stage, especially for block level sea-of-gates designs. The difficulty in gate sizing for noise reduction is that, by increasing a driver size, noise at the driver output is reduced, but noise injected by that driver on other nets is increased. This can create cyclical dependencies between nets in the circuit with noise violations. In this paper, we propose a fast and effective heuristic postroute gate-sizing algorithm that uses a graph representation of the noise dependencies between nodes. Our method utilizes gate sizing in both directions and works in linear time as a function of the number of gates. The effectiveness of the algorithm is shown on several industrial high-performance designs.**

***Index Terms*—Crosstalk noise, gate sizing, noise repair, signal integrity.**

I. Introduction

Crosstalk noise is a critical design and verification issue for large, high-performance designs. This problem has become more significant due to the increased ratio of crosstalk capacitance to total capacitance of a wire and the usage of more aggressive and less noise immune circuit structures, such as dynamic logic.

In noise analysis, the nets on which crosstalk noise is injected by one or more of its neighbors are called the *victim* nets whereas the nets that inject this noise are called the *aggressor* nets. Noise can be divided into two types: *Functional noise* refers to noise that occurs on a victim net which is being held quiet by a driver. Crosstalk noise on such a victim causes a glitch which may propagate to a dynamic node or a latch, changing the circuit state and causing a functional failure [1], [2]. On the other hand, *delay noise* refers to noise that occurs when two capacitively coupled nets switch simultaneously. Depending on the direction of these transitions, the delays on both nets are affected [3], [4]. The focus of this paper is on functional noise but the presented techniques can be extended for delay noise as well.

Recent literature proposes a number of crosstalk noise analysis and noise avoidance methods. In [1] and [5], detailed noise analysis using parasitic extraction and model order reduction is proposed. In [6] and [7], the authors introduce simple noise metrics for crosstalk amplitude and pulse width in capacitively coupled interconnects. The derived expressions are also used to motivate circuit-design techniques, such as transistor sizing and layout techniques to reduce crosstalk. In other recent works, [8] proposes an improved 2π model which is extended by [9] to a 4π model. Reference [9] also uses this model to analyze the effects of several circuit parameters to noise, giving guidelines to the effectiveness of several noise reduction methods. In an effort to identify crosstalk noise problems early in the design cycle, [10] proposes an probabilistic extraction algorithm based on global route congestion and a predetailed-route crosstalk-noise analysis methodology.

In this paper, we focus on correcting the identified crosstalk-noise problems in the postroute design stage. Noise can be reduced through routing and interconnect optimization (wire spacing, wire widening, controlling coupling length and position) [11], [12], buffer insertion [13], [14] and driver sizing. In the postroute design stage, it is not desirable to use techniques such as wire perturbations and buffer insertion since they would require rerouting and, thus, increase design time. Rerouting can result in significant changes in net lengths and neighbors of nets which can cause many new noise failures that did not exist initially and may result in nonconverging design iterations. After routing is completed, noise failures are therefore more effectively corrected using driver sizing. The flexibility through scalable libraries and existing fill-space, allows one to make incremental changes to the driver sizes without affecting global routing.

Recently, transistor-sizing methods for crosstalk-noise reduction were proposed in [15]–[18]. Reference [15] uses coupling capacitance as the noise metric and optimizes noise, area, delay, and power by gate and wire sizing. A gate-sizing method to reduce crosstalk induced delay noise is proposed in [16] and is based on a crosstalk-noise aware static-timing analysis. More recently, in [18], a postroute gate-sizing algorithm for crosstalk-noise reduction is proposed. However, since this method only utilizes downsizing of aggressor drivers under delay constraints, and not increasing the size of victim drivers, it has limited effectiveness.

In this paper, we therefore propose a new postroute gate sizing algorithm for crosstalk noise reduction. Due to the nonlinear dependence of crosstalk noise to the interconnect and driver parameters [9], [17], the problem of postroute gate-sizing for crosstalk-noise reduction is a nonlinear optimization problem. The overwhelming system size (i.e., number of nets) in today's highly coupled interconnects makes it impractical to solve the nonlinear optimization problem in an exact manner. We propose a heuristic algorithm in this paper. The algorithm increases the size of victim drivers as well as reducing the size of aggressor drivers. The proposed algorithm takes into account both timing and area constraints and treats each net as both an aggressor as well as a victim. This duality is a critical factor in postroute gate sizing and must be accounted for to ensure that new noise violations are not introduced while fixing existing failures. We approach the problem by introducing a noise graph which is constructed based on the static noise analysis of the design. The noise graph represents all critical nets, their significant aggressors, and the noise dependencies (aggressor–victim relation) between them. We introduce a sensitivity measure to eliminate weak dependencies from the noise graph to reduce system complexity. The strong cyclic dependencies in the noise graph are investigated using a *cyclic sensitivity* metric. Cycles that are likely to converge are allowed to remain in the graph, however, high sensitivity cycles are eliminated from the graph by removing a minimum number of vertices. Some edges are temporarily removed from the low-sensitivity cycles to obtain an acyclical graph. The resulting acyclical graph is then sorted topologically, where the topology in the noise graph represents the noise-dependency relations. Gate sizing is then iteratively performed on the sorted noise graph under delay and area constraints. The algorithm is guaranteed to converge and has a

Manuscript received December 15, 2003; revised March 10, 2004. This paper was recommended by Associate Editor M. D. F. Wong.

M. R. Becer, I. Algor, and R. Panda are with Freescale Semiconductor, Austin, TX 78735-8598 USA (e-mail: muratbecer@motorola.com).

D Blaauw is with the Department of Electrical Engineering and Computer Science, University of Michigan, Ann Arbor, MI 48109 USA.

C. Oh is with Nascentric, Austin TX 78759 USA.

V. Zolotov is with IBM T. J. Watson Research Center, Yorktown Heights, NY 10598 USA.

I. N. Hajj is with the Department of Electrical and Computer Engineering, University of Illinois, Urbana–Champaign, IL 61801 USA.

Digital Object Identifier 10.1109/TCAD.2004.836736

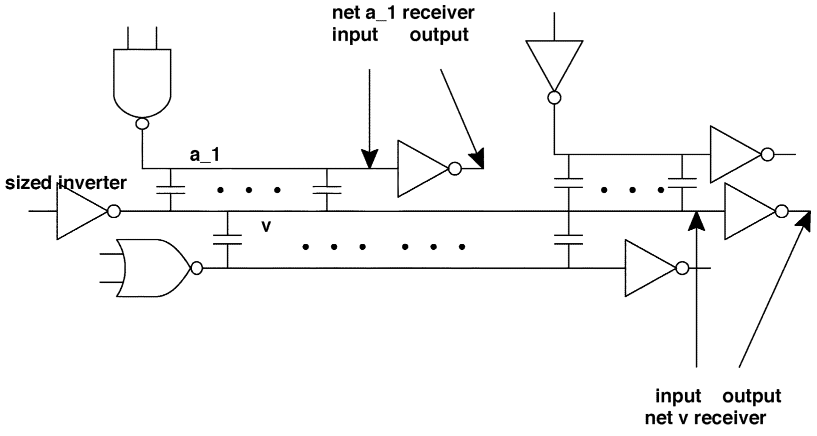

Fig. 1. Coupled net cluster.

runtime complexity that is linear with the size of the circuit. Results on three large microprocessor designs are presented to demonstrate the effectiveness of the approach.

The paper is organized as follows. Section II outlines a brief qualitative and quantitative analysis of postroute gate sizing and formally defines the postroute gate sizing problem. In Section III, we explain the noise-graph concept and our algorithm in detail, including the cycle-breaking strategy and sensitivity measures. Results on three high-performance microprocessor designs are presented in Section IV. Section V contains closing remarks.

II. Gate Sizing for Noise Reduction

If a victim driver is sized up, its effective conductance increases and more effectively holds a net at a steady voltage (v_{dd} or ground). Also, if an aggressor driver is sized down, its effective conductance decreases and as a result, noise induced by the aggressor on a victim net decreases. Fig. 1 shows a situation where several nets form a coupled cluster. We investigate the noise pulses at the receiver input and output of net v and a_1 as the driver gate of net v is sized up. Fig. 2 shows the voltage waveforms at receiver input and receiver output of net v when the driving gate of net v is changed from inv_4 to inv_12. The inverters are from a high-performance standard cell library and their transistor widths are proportional to x in inv_x. As can be seen, the noise pulse at the receiver input of net v is reduced in terms of both noise peak and width, as the driving gate is sized up. Note that, although the noise pulse peak with inv_12 is nearly 450 mV (35% of V_{dd}), the propagated noise at the receiver output is negligible. The complication of victim–aggressor duality in driver sizing emerges when we consider the voltage waveforms on net a_1. Fig. 3(a) shows the high to low transition on net v, when net v is acting as an aggressor on net a_1. As can be seen, when the driver of v is sized up from inv_4 to inv_12, the transition on net v becomes faster, making it a stronger aggressor on net a_1. This causes the noise pulse at the receiver input of a_1 to increase about 50 mV resulting in the propagated noise pulse to increase by 200 mV as seen in Fig. 3(b). As demonstrated in this example, when a net's driver gate is sized up to reduce noise on that net, it also becomes a stronger aggressor which in turn can induce more noise on other nets. Thus, a driver-sizing solution on nets with existing noise failures, may result in new failures on other nets. Postroute driver sizing, therefore, must account for this dependence between nets to ensure convergence of the algorithm.

In some cases, by sizing both victim and aggressor drivers, we can converge to a solution that fixes noise, while in other cases, the two drivers can be sized in an iterative manner and noise is not fixed on either net. The unnecessary sizing up of drivers of nets in such cyclic

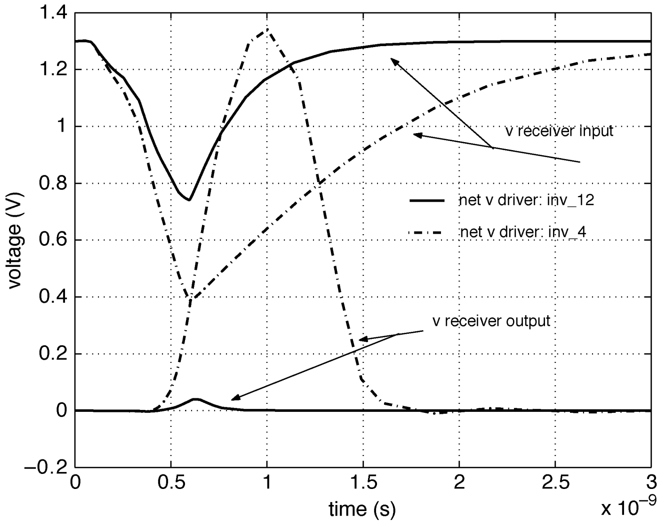

Fig. 2. Noise effects on net v due to sizing up of driver of net v.

dependencies will also result in more induced noise on other nets. Sensitivity of noise on these nets with respect to their driver sizes is a key factor in whether the sizing will converge or not. A good gate-sizing algorithm must distinguish between these two cases to eliminate those cases that will not converge. Otherwise, such nonconverging dependencies can destroy the sizing of other nets and introduce more noise in other parts of the design.

Other critical complications in the postroute driver-sizing problem can be explained as follows.

Area impact: When a driver is sized up, it requires a larger footprint and may cause the legalizor to shift around some of the neighboring instances causing some changes in the routing. Usually, the additional area required to size up a driver gate (i.e., replacing it with a bigger size from the library) is less than the area required to insert a buffer and is less likely to modify the route significantly. Also, the existing fill space around the instances can be utilized.

Timing impact: When a gate size is changed, it affects the timing of paths through this gate. Sizing up a gate speeds up the signal on the net that it drives but it also presents a higher gate capacitance to the previous net, slowing it down. Reverse effects are true if a gate is sized down. Effects on the previous net can be eliminated if the gates in the library are designed in multiple stages, keeping the first stage size the same and reflecting the size differences in the

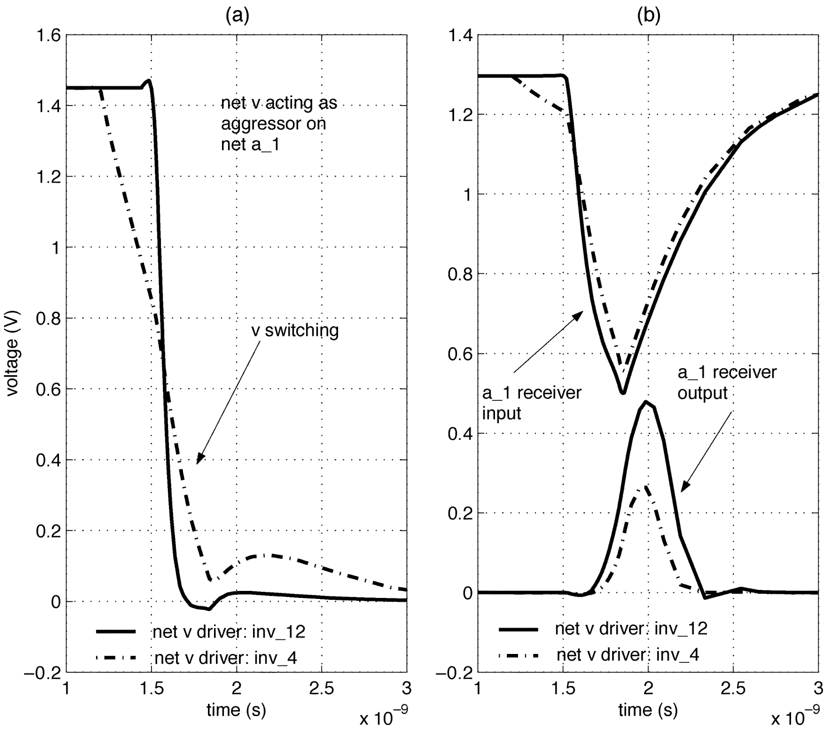

Fig. 3. Noise effects on net a_1 due to sizing up of driver of net v.

driving stage. During gate sizing, effects on timing are represented as constraints on gate sizes.

System size: The victim–aggressor duality dictates that all interacting nets and their driver-gate sizes should be taken into account in a gate-sizing algorithm. This makes the problem a constrained nonlinear multiple goal attainment problem for which the system size (>100 k nets is very common) can be prohibitive. An exact solution is therefore not practical, and we propose a heuristic solution in this paper.

A. Formal Definition of Postroute Gate Sizing Problem

Suppose there are n nets in a given design, where each net corresponds to a driver that is part of a standard cell library. Prior to postroute driver sizing, all the driver and interconnect data for all nets are available. Also a postroute noise analysis has been done on the design. Assume that, according to the results of postroute noise analysis [1], k of these nets are failing, and the remaining $n-k$ are passing the crosstalk noise failure criterion chosen in the analysis.[1]

Noise on each net can be represented as a nonlinear function of the victim net's driver size and all its aggressor nets' driver sizes

$$\text{noise}_{net_i} = f\left(s_i, s_{\text{aggressors}(\text{net}_i)}\right) \tag{1}$$

where s_x represents a gate from the standard cell library which has the same functionality of the original driving gate of net x. The goal of postroute gate sizing is to bring down this noise on the k failing nets to a nonfailing level while making sure that the $n-k$ initially nonfailing nets stay that way. Additional constraints on the driver gates are due to timing and area considerations. A net's driver has an upper and a lower limit due to area and timing constraints. So, if we put all these observations together, the postroute gate sizing problem can be formulated as follows. Solve for s_i to attain the goal of $\text{noise}_{\text{net}_i} \leq \text{threshold}_{\text{net}_i}$ for the k failing nets such that $\text{noise}_{\text{net}_i} \leq \text{threshold}_{\text{net}_i}$ remains for the $n-k$ nonfailing nets and $\min_{s_i} \leq s_i \leq \max_{s_i}$ for area and timing constrains. As can be seen, this is a discrete multiobjective nonlinear optimization problem with nonlinear and linear constraints.

[1]The interested reader is referred to [19] for a detailed discussion on functional noise failure criteria.

In practice, each net can be driven by more than one driver. This can be reduced to the above case (n nets and n drivers) by picking the worst/dominant driver for each net.

Any algorithm that attacks this problem should have an efficient yet accurate way of calculating noise on a net, given both its and its aggressors' driver gates, as well as the coupled interconnect cluster that represents the victim and the aggressor nets [i.e., a way of solving (1)]. Fast and efficient analytical models [8], [9] can be used. For more accuracy, one can use linearized drivers and PRIMA-like [20] model-order reduction methods for the interconnect. For the highest amount of accuracy, SPICE can be used at the expense of increased run time.

III. Proposed Gate-Sizing Method

In a postroute design stage, detailed information on the topology, neighbors and drivers of all the nets in the design is available. First, we perform an accurate postroute static noise analysis on the design, utilizing timing and logic correlation information to avoid false failures [1]. Noise analysis identifies the severity of noise on each net through a "slack" value. If the slack of a net is negative, it is failing the noise analysis. The failure criterion used in the paper is the so called the "noise rejection curve" method [1]. Each cell in the standard cell library is characterized with a noise rejection curve which shows the noise pulse Height/Width boundary at which the cell starts to propagate more than a predefined output noise threshold. In this context, slack is defined as the difference between the predefined output noise threshold and the propagated noise peak at the receiver output: $(noise\_slack = output\_noise\_threshold - V(receiver\_output))$. Note that the proposed algorithm is independent of the noise failure criterion being used.

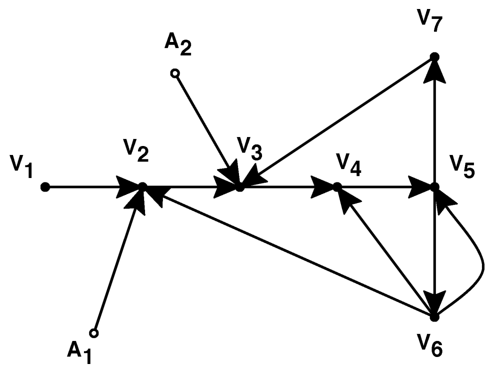

Fig. 4. Sample noise graph.

A. Noise-Graph Representation

As explained in this section, we represent the gate-sizing problem using a noise graph. A noise graph $G((V,A),E)$ consists of vertices (V,A) and edges E.

- **Vertices**: Nets are represented by vertices in the noise graph. There are two types of vertices. Type V vertices represent nets which are failing or close to failing the noise analysis. In other words, nets that have noise slack less than some predefined positive value will be of Type V vertices. Drivers of Type V vertices are candidates to be sized up. Type A vertices represent significant aggressors which have very low noise on them. A significant aggressor is an aggressor which contributes at least 20% of the total noise on a victim net. Very low noise means that net has a noise slack greater than a predefined positive slack. Drivers of Type A vertices are candidates to be sized down.
- **Edges**: A directed edge between vertex a and vertex b exists if net a is a significant aggressor of net b. Note that type A vertices always have an in-degree of 0.

Fig. 4 shows a simple noise graph. It is a directed graph which contains cycles. The noise graph contains all the failing and critical nets ($V1$-$V7$) as well as their very low-noise aggressors ($A1$-$A2$). It also contains the existing significant relations between these nets in the form of edges. In reality, victim–aggressor duality exists for each neighboring net in the design. However, in our noise graph, we incorporate only significant edges, filtering out the insignificant victim-aggressor dependencies which would otherwise increase complexity. Therefore, cycles in the noise graph represent significant victim–aggressor dependencies (in some cases in a more extended sense—V3-V4-V5-V7-V3 cycle). These cycles may lead to oscillating solutions and/or convergence problems. In the simplest case of a two-vertex cycle, made up of vertices V5 and V6, the negative slack can oscillate between the two nets as each one is sized up and neither is fixed. The noise graph also dictates an order in which Type V vertices are sized up. For example, if we first size up V2 and then V1, we might have to come back to V2 as it is affected by V1. This information is utilized to minimize the complexity of our algorithm.

B. Sizing Algorithm

Our algorithm can be summarized as follows. After constructing the noise graph as explained in the previous section, we first size down all Type A vertices. At this point, if the noise graph is acyclic, we simply size up the Type V vertices in topological order, where the topology in the noise graph represents the noise-dependency relations. However, in general, the noise graph will contain cycles which may lead to problems. We introduce *cyclic sensitivity*, a metric designed to represent the overall noise trend in a noise-dependence cycle as the drivers are sized up, to distinguish those cycles that will cause convergence problems and those that will converge. Low sensitivity cycles are stripped off some edges temporarily, to be able to obtain an acyclic graph. Problematic cycles are eliminated through the removal of some Type V vertices from the noise graph. The removed vertices are not sized and thus will not be fixed by driver-gate sizing. Our algorithm removes a minimum number of vertices to eliminate the problematic cycles. The resulting directed acyclic graph is then topologically sorted, and Type V vertices are sized up in the topological order. The problem of iterating over the low sensitivity cycles in the noise graph is solved by repeating the topological sizing procedure until convergence. The pseudocode of the proposed algorithm is shown below.

```
Algorithm: Postroute Driver Sizing
**Input**: Noise analysis results
**Output**: Instance cell replace directives
begin
1 Construct a noise graph $G = ((V,A),E)$
      based on noise analysis
2 $\mathrm{Size\_down\_type\_}A\mathrm{\_vertices}(G)$
3 $\mathrm{Analyze\_cycles}(G)$
4 $\mathrm{Break\_cycles}(G)$
5 $G_s = \mathrm{Topologic\_sort}(G)$
6 repeat until convergence
7  for each vertex $v$ in $G_s$
8    $\mathrm{Size\_up}(v)$
end
```

We now explain the algorithm stages in detail. In Step 1, we construct a noise graph based on the noise analysis. The complexity of this step is $O(|V|+|A|+|E|)$. During the construction of the noise graph, we apply a sensitivity-based pruning method to further eliminate some of the introduced edges. An edge e from vertex u to vertex v represents a significant noise contribution from the net represented by vertex u to the net represented by vertex v. We add a dynamic character to this static edge-insertion criterion by introducing a sensitivity notion. As the driver of vertex u is sized up, if the noise change on vertex v is not significant, i.e., $\Delta(\mathrm{noise}_v)/\Delta(\mathrm{size}_u)$ is very small, then we can conclude that when vertex u is sized up, this will not increase the noise on vertex v significantly. In other words, the noise dependency from u to v is weak. Edges that represent such weak dependencies are eliminated. This results in reduced complexity of the noise graph and in some cases elimination of some noise dependence cycles. Note that the effects, although small, of these eliminated edges will be caught as the topological sizing is iterated. In Step 2, we size down all Type A vertices as much as possible such that they maintain a sufficient noise slack margin and stay within the timing constraints. By sizing down the significant aggressors up front, the rest of the algorithm is simplified since from this point on only size-up operations will be performed. The constraint on the $noise\_slack$ of Type A vertices ensures that no new failures among these nets will be introduced, while trying to fix the existing failing nets.

In Step 3, we analyze the cycles in the noise graph. the complexity of this step is bounded by $O(|V|+|A|+|E|)\times$size of (largest cycle). Cycles represent significant noise dependencies in the noise graph which have a cyclic nature. These cycles may lead to oscillating solutions and thus convergence problems. We introduce a *cyclic sensitivity* metric to separate those cycles which will converge, from those which will create problems. Assume we have a cycle C in our noise graph, made up of n vertices $(V_1 \dots V_n)$ and n edges $(e_1 \dots e_n)$. Note that since we already eliminated edges that correspond to weak dependencies from the noise graph, all the edges in cycle C represent "strong" noise dependencies. Our cyclic sensitivity metric for cycle C is defined in

$$cs(C) = \left|\frac{\frac{\delta N_2}{\delta s_1}}{\frac{\delta N_1}{\delta s_1}}\right| \times \left|\frac{\frac{\delta N_3}{\delta s_2}}{\frac{\delta N_2}{\delta s_2}}\right| \times \ldots \times \left|\frac{\frac{\delta N_n}{\delta s_{n-1}}}{\frac{\delta N_{n-1}}{\delta s_{n-1}}}\right| \times \left|\frac{\frac{\delta N_1}{\delta s_n}}{\frac{\delta N_n}{\delta s_n}}\right| \quad (2)$$

where N_i is the noise on vertex i and s_i is the size of the gate at vertex i. Each term in (2) represents the consequences of sizing up a gate in the cycle C. The ith term is the ratio of the sensitivity of induced noise on vertex $i+1$ to the size of the driver of vertex i and the sensitivity of noise reduction on vertex i to the size of driver of vertex i. In other words, each term is a measure of additional noise introduced to the cycle versus the noise reduction from the cycle due to sizing up a gate in the cycle. Therefore, if $cs(C) < 1$, this means that as the gates in cycle C are sized up, overall cycle noise will tend to go down. On the other hand if $cs(C) > 1$, this means that the overall cycle noise will increase as we size up the gates, leading to a nonconvergent situation. In Step 3, we identify and analyze each cycle in the noise graph and eliminate an edge from those cycles with low *cyclic sensitivity*. The reason we eliminate an edge from these cycles is to be able to obtain an acyclic graph while keeping all the vertices of such cycles in the noise graph. Note that the apparent loss of information by eliminating an edge from low-sensitivity cycles is actually a temporary one. The practical task of iterating on these cycles is solved algorithmically by iterating on the entire noise graph in Steps 7 and 8. As noise analysis is performed on each vertex (i.e., net), all aggressors of that net is taken into account regardless of the absence or existence of an edge in the noise graph. As a result, the effect of these eliminated edges are accounted for as the sizing process is iterated in Steps 7 and 8.

After Step 3, the remaining cycles in the noise graph are those which will cause convergence problems. In Step 4, we remove such cycles in the graph by eliminating a minimum number of vertices. By removing vertices from the noise graph, we sacrifice some nets (they will not be fixed by driver sizing), but we ensure that there will not be any convergence issues. Our cycle-breaking strategy ($\mathrm{Break\_cycles}(G)$) ensures that the minimum number of Type V vertices are removed from the noise graph: Let G be a directed graph $G = (V, E)$, where V is the set of vertices and E is the set of edges. We want to find a feedback vertex set, i.e., a subset $V' \subset V$ such that V' contains at least one vertex from every directed cycle in G, while minimizing the cardinality of the feedback vertex set $|V'|$. This problem is equivalent to the known graph-theory algorithm, minimum feedback vertex set, which is shown to be approximable within $O(\log|V|\log\log|V|)$ [21]. Breaking the cycles result in a directed acyclic graph (DAG). This graph is topologically sorted in Step 5, whose complexity is $O(|V| + |A| + |E|)$.

In Steps 7 and 8, the gates are sized in topological order. This ensures that the victim-aggressor duality is taken into account. Since we are sizing in the order of noise dependence, the effects of sizing up a driver will be seen downstream, on the nets that it has an effect on. The noise graph consists of nets that are failing and that are close to failing. The topological sort approach makes sure that if any of the 'close to failing' nets start failing due to one of its upstream neighbors being sized up, this is detected and addressed. At each vertex, a proper gate size from the cell library is chosen such that the area and timing constraints are satisfied. The pseudocode for the size-up process is shown below. Elimination of some significant edges to preserve the low sensitivity cycles in Step 3, dictates that Steps 7 and 8 should be iterated until convergence. As explained above in Step 3, the effects of eliminated edges will be seen at each iteration since our noise calculation on a net takes into account all the aggressors of that net, whether or not they are represented in the noise graph. The theoretical limit on the complexity of these iterations is linear with the system size. However, in practice, the number of required iterations was found to be very small and thus can be treated as constant. Therefore, the proposed algorithm works in linear time as a function of the number of gates in the design.

TABLE I
NOISE-REDUCTION RESULTS USING THE PROPOSED ALGORITHM

Circuit	# of nets	# of failing nets		noise reduction	
		Initial	After opt.	max	avg
chip_1	31489	42	23	30%	11%
chip_2	39200	52	1	44%	23%
chip_3	165481	502	70	87%	21%

TABLE II
SOME DETAILED STATISTICS FROM THE RUNS

	↓ a	# ver			# cy		# edg			
		in	ac	bc	hs	ls	in	el	ac	bc
chip_1	22	84	84	79	6	0	39	34	34	23
chip_2	8	90	90	90	0	2	12	12	10	10
chip_3	22	602	602	602	0	72	217	194	123	123

```
Algorithm: Size_up($v$)
Input: Type V vertex $v$
Output: library cell to replace driver
    of vertex $v$
begin
1 while $area\_slack(v)$ and $time\_slack(v)$ are
    within constraints
2  replace driver of $v$ with next larger
    same functionality cell in library
3  if $noise\_slack(v) \geq 0$
4    return
5 return
end
```

IV. RESULTS

In this section, we present results of our algorithm on three large designs. The circuits used for experiments are chip_1, which has 31 489 nets, chip_2, which has 39 200 nets and chip_3 with 165 481 nets. All three designs are actual high performance ICs in 0.18-μm technology and the number of nets reflect the number of top level nets analyzed by the noise analysis tool. Coupled *RC* interconnects were extracted using a commercial extraction tool and the analysis was performed in the typical process corner. Each cell in the standard cell library used in these designs was precharacterized for holding and switching driver models and receiver noise failure criteria [1]. Initial noise analysis is performed on these three designs after they have been optimized for delay and slew constraints. During gate sizing for noise reduction, we use the timing slacks obtained from static timing analysis as timing constraints.

Table I shows the noise-reduction results obtained by applying the proposed algorithm and Table II shows some statistical information on the runs. From Table I, we can see that number of nets that fail the noise criterion goes down significantly, as much as by 98%. The last two columns in Table I show the maximum and average peak noise voltage reduction.

Table II presents the following information in column order: number of aggressor gates that have been sized down [$\downarrow a$], number of vertices in the initial graph [$\#ver/in$], number of remaining vertices after $Analyze\_cycles$ [$\#ver/ac$], number of remaining vertices after $Break\_cycles$ [$\#ver/bc$], number of "high sensitivity cycles"

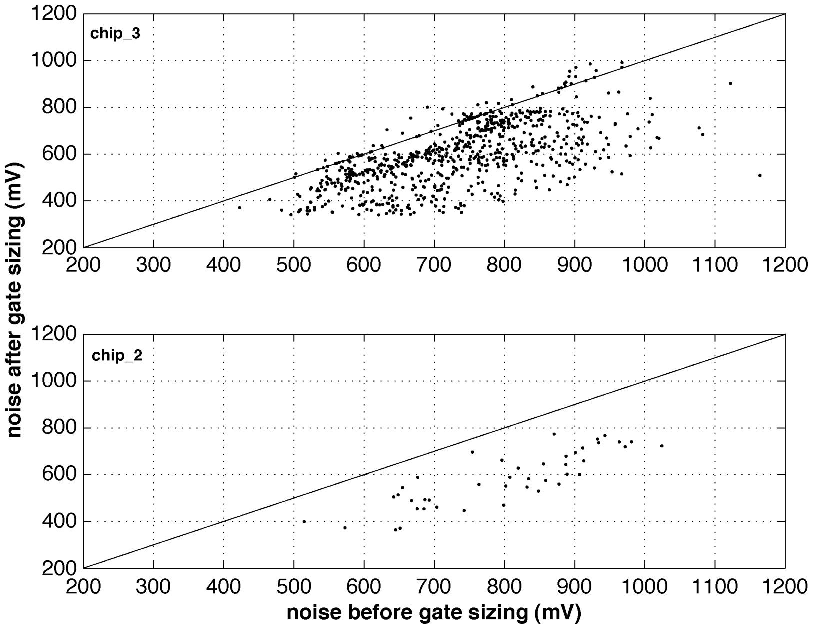

Fig. 5. Noise-peak changes at receiver input before and after employing proposed algorithm.

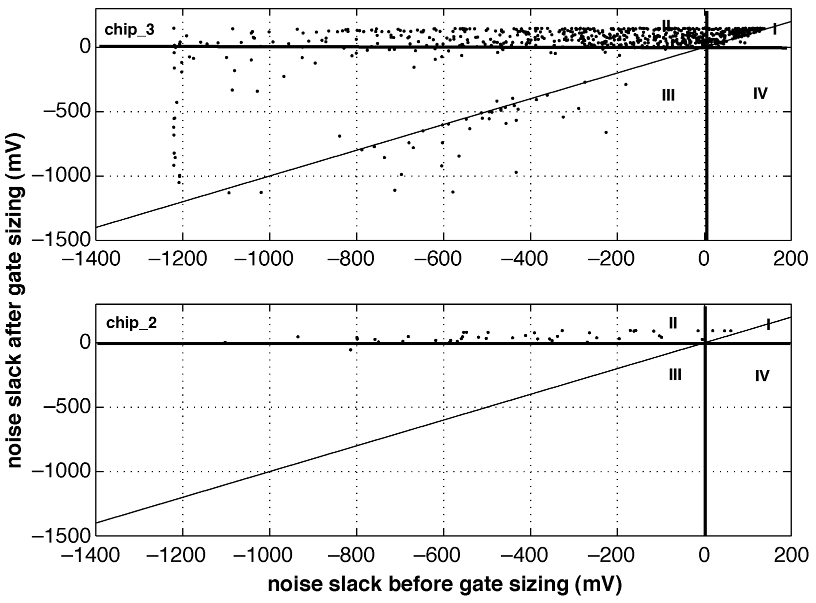

Fig. 6. Noise slack changes at receiver output before and after employing proposed algorithm.

[$\#cy/hs$], number of "low sensitivity" cycles [$\#cy/ls$], number of edges in initial graph [$\#edg/in$], number of edges remaining after sensitivity pruning [$\#edg/el$], number of edges remaining after $Analyze\_cycles$ [$\#edg/ac$], number of edges remaining after Break_cycles [$\#edg/bc$]. Note that all the cycles in chip_1 were high sensitivity cycles whereas chip_2 and chip_3 contained many low-sensitivity cycles, resulting in better noise-reduction results in chip_2 and chip_3. This shows the importance of detecting weak cycles instead of blindly breaking all cycles through vertex elimination.

In Figs. 5 and 6, we show the changes in noise peak voltages at receiver inputs and changes in noise slack values at receiver outputs. Each dot in these figures corresponds to a noise simulation. The values on the x and y axes are before and after gate sizing, respectively, in both figures. The 45° line is the $x = y$ line. The region below the line represents improvement in noise in Fig. 5 and degradation in noise in Fig. 6. Fig. 6 is additionally divided into four quadrants by the vertical and horizontal lines at $x = 0$ and $y = 0$. Each dot in these figures corresponds to a noise simulation, and each net has two noise simulations. One noise simulation is where the victim net is stable at ground and the aggressors are switching from low to high and the other is the reverse situation. It can be seen from Fig. 5 that, noise on most nets has been reduced and only on some nets noise has increased slightly. Note from Table II that chip_2 and chip_3 did not lose any vertices from the noise graph since all their significant noise dependence cycles have been eliminated using the *cyclic sensitivity* measure. Thus the few nets in chip_3, whose noise have increased after gate sizing, are those whose gates were not sized up due to other constraints such as timing and area. Note that Figs. 5 and 6 only show simulations of those nets whose peak noise exceed a certain value. This is why we do not see the noise increase on the aggressor nets whose driver gates have been sized down. Fig. 6 shows that no slack value went from positive to negative after gate sizing (i.e., quadrant IV is empty). Hence, if a net did not fail

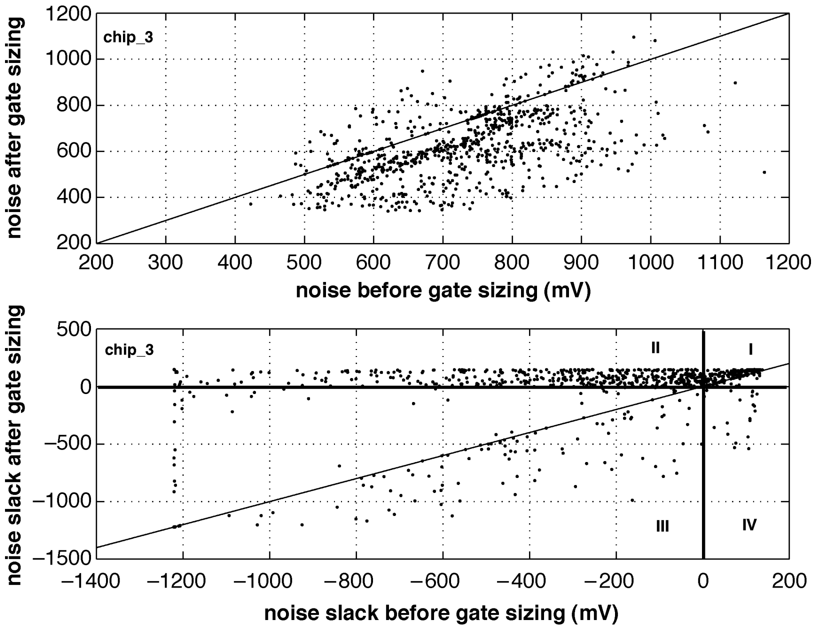

Fig. 7. Noise peak and slack changes in chip_3 when cyclic sensitivity is not used.

before gate sizing, it remained that way after gate sizing. This shows that our algorithm is successful in not introducing new noise problems while trying to fix the existing ones. This is due to the fact that our algorithm checks the noise on aggressors as they are sized down and also the aggressor–victim duality is taken into account properly through the topological sort approach. These figures also show that, noise failure on many nets has been improved although the net was not fixed. However, on the other hand, some nets that were failing initially may end up failing worse after gate sizing, especially if their corresponding vertices have been eliminated during the $Break\_cycles$ procedure. We minimize such nets by choosing the minimum number of vertices to be eliminated from the noise graph. These results show the effectiveness of the *cyclic sensitivity* measure which significantly reduces the number of eliminated vertices (sacrificed nets), improving the quality of the obtained solution. The algorithm converged after two iterations in all three chips and required 163, 150, and 780 s runtime, respectively, on an UltraSparc-II machine.

To quantify the efficiency of cyclic sensitivity, our algorithm was reapplied on chip_3, but this time skipping the $Analyze\_cycles$ step. To eliminate the 72 cycles, 53 vertices were eliminated in $Break\_cycles$ step, and the number of noise violations after gate sizing was 108. Remember that none of the 72 cycles were eliminated through vertex removal when the cyclic sensitivity is used. The change in noise and slack are shown in Fig. 7. As can be seen in Fig. 7, there are some simulations in quadrant IV. All these correspond to nets which were eliminated in $Break\_cycles$ step.

Although no new violations were created (the simulations in quadrant IV belonged to nets which were already failing due to the other type of simulation), the quality of solution is degraded, compared to the one in which cyclic sensitivity was used. Also note that 38 more nets were fixed when cyclic sensitivity was used, whereas these nets are simply ignored due to vertex elimination if cyclic sensitivity is not employed. The advantages of using cyclic sensitivity are clear when Figs. 5 and 6 are compared with Fig. 7.

We have also observed that the impact of proposed driver-sizing operations on the congestion and routability of the reported test cases has been minimal. This is partly due to the fact that the presented algorithm uses available physical information on the sizeability and legalizability of each driver by constraining the available sizes for each driver as explained in step 8. This is also due to the fact that this algorithm is employed as one of the final stages of a crosstalk noise management methodology [22]. At this stage of the design flow, several techniques (before and during routing noise prevention) have already been applied to minimize crosstalk noise problems as much as possible; thus leaving a relatively small number of failures to deal with using driver sizing.

As a result, our algorithm reduced number of failing nets significantly (45%, 98%, and 86% in three designs, respectively) while not introducing any new failures. Some controlled increase of noise on aggressor nets was allowed, making sure that they stayed within acceptable positive slack. Even with the increase of noise on eliminated vertex nets, average noise reduction was 11%, 23%, and 21%, respectively, for the three designs.

V. Conclusion

In this paper, we presented a postroute gate-sizing algorithm for crosstalk-noise reduction. The algorithm is timing and area constrained and takes into account the victim–aggressor duality through a topologically sorted noise graph. We introduced a cyclic sensitivity metric to be able to efficiently handle cyclic noise dependencies which present the most important challenge in the post route gate sizing problem. The proposed algorithm works in linear time as a function of design size, utilizes sizing in both directions and has been shown to be effective on large high performance designs.

References

[1] S. Alwar, D. Blaauw, A. Dasgupta, A. Grinshpon, R. Levy, C. Oh, B. Orshav, S. Sirichotiyakul, and V. Zolotov, "Clarinet: a noise analysis tool for deep submicron design," in *Proc. Design Automation Conf.*, 2000, pp. 233–238.

[2] K. L. Shepard and V. Narayanan, "Noise in deep submicron digital design," in *Proc. Int. Conf. Computer-Aided Design*, 1996, pp. 524–531.

[3] S. Sirichotiyakul, D. Blaauw, C. Oh, R. Levy, V. Zolotov, and J. Zuo, "Driver modeling and alignment for worst-case delay noise," in *Proc. Design Automation Conf.*, 2001, pp. 720–725.

[4] P. D. Gross, R. Arunachalam, K. Rajagopal, and L. T. Pileggi, "Determination of worst-case aggressor alignment for delay calculation," presented at the IEEE Int. Conf. Computer-Aided Design, 1998.

[5] K. L. Shepard, “Design methodologies for noise in digital integrated circuits,” in *Proc. Design Automation Conf.*, 1998, pp. 94–99.
[6] A. Vittal, L. H. Chen, M. Marek-Sadowska, K. P. Wang, and S. Yang, “Crosstalk in VLSI interconnections,” *IEEE Trans. Computer-Aided Design*, vol. 18, pp. 1817–1824, 1999.
[7] A. Vittal, L. H. Chen, M. Marek-Sadowska, K. P. Wang, and X. Yang, “Modeling crosstalk in resistive VLSI interconnections,” in *Proc. Int. Conf. VLSI Design*, 1999, pp. 470–475.
[8] J. Cong, D. Z. Pan, and P. V. Srinivas, “Improved crosstalk modeling for noise constrained interconnect optimization,” in *Proc. ASP/DAC Asia South Pacific Design Automation Conf.*, 2001, pp. 373–378.
[9] M. R. Becer, D. Blaauw, V. Zolotov, R. Panda, and I. N. Hajj, “Analysis of noise avoidance techniques in DSM interconnects, using a complete crosstalk noise model,” in *Proc. Design Automation Conf. Eur.*, 2002, pp. 456–463.
[10] M. Becer, D. Blaauw, R. Panda, and I. N. Hajj, “Early probabilistic noise estimation for capacitively coupled interconnects,” *IEEE Trans. Computer-Aided Design*, vol. 22, pp. 337–345, Mar. 2003.
[11] H. Zhou and D. F. Wong, “Global routing with crosstalk constraints,” in *Proc. Design Automation Conf.*, 1998, pp. 374–377.
[12] P. Saxena and C. L. Liu, “Crosstalk minimization using wire perturbations,” in *Proc. Design Automation Conf.*, 1999, pp. 100–103.
[13] C. P. Chen and N. Menezes, “Noise aware repeater insertion and wire sizing for on-chip interconnect using hierarchical moment matching,” in *Proc. Design Automation Conf.*, 1999, pp. 502–506.
[14] C. J. Alpert, A. Devgan, and S. T. Quay, “Buffer insertion for noise and delay optimization,” in *Proc. Design Automation Conf.*, 1998, pp. 362–367.
[15] I. H. R. Jiang, Y. W. Chang, and J. Y. Jou, “Crosstalk driven interconnect optimization by simultaneous gate and wire sizing,” *IEEE Trans. Computer-Aided Design*, vol. 19, pp. 999–1010, 2000.
[16] T. Xiao and M. Marek-Sadowska, “Gate sizing to eliminate crosstalk induced timing violation,” in *Proc. Int. Conf. Comput. Design*, 2001, pp. 186–191.
[17] ——, “Crosstalk reduction by transistor sizing,” in *Proc. Asia South Pacific Design Automation Conf.*, 1999, pp. 137–140.
[18] M. Hashimoto, M. Takahashi, and H. Onodera, “Crosstalk noise optimization by post-layout transistor sizing,” in *Proc. Int. Symp. Phys. Design*, 2002, pp. 126–130.
[19] V. Zolotov, D. Blaauw, S. Sirichotiyakul, M. Becer, C. Oh, R. Panda, A. Grinshpon, and R. Levy, “Noise propagation and failure criteria for VLSI designs,” in *Proc. Int. Conf. Computer-Aided Design*, 2002, pp. 587–594.
[20] A. Odabasioglu, M. Celik, and L. T. Pileggi, “PRIMA: passive reduced-order interconnect macromodeling algorithm,” *IEEE Trans. Computer-Aided Design*, vol. 17, pp. 645–653, Aug. 1998.
[21] G. Even, J. Naor, B. Schieber, and M. Sudan, “Approximating minimum feedback sets and multi-cuts in directed graphs,” in *Proc. Int. Conf. Integer Program. Combinat, Optim.*, 1995, pp. 14–28.
[22] M. Becer, R. Vaidyanathan, C. Oh, and R. Panda, “Signal integrity management in an soc physical design flow,” in *Proc. Int. Symp. Phys. Design*, 2003, pp. 39–46.

Statistical Timing Analysis of Coupled Interconnects Using Quadratic Delay-Change Characteristics

Tom Chen and Amjad Hajjar

***Abstract*—With continuing scaling of CMOS process, process variations in the form of die-to-die and within-die variations become significant which cause timing uncertainty. Statistical design methods have been proposed in the past to model the impact of process variations. However, all the existing methods deal almost exclusively with modeling delay variations of logical gates or physical variations of interconnect wires. This paper proposes a method of analytically analyzing statistical behavior of multiple coupled interconnects with an uncertain signal arrival time at each interconnect input (aggressors and the victim). The method utilizes delay-change characteristics due to changes in relative arrival time between an aggressor and the victim. The results show that the proposed method is able to accurately predict delay variations through a coupled interconnect.**

***Index Terms*—Coupled interconnect, crosstalk noise, delay-change curve (DCC), statistical timing analysis.**

I. Introduction

With continuing scaling of CMOS process, die-to-die and within-die variations have a significant impact on chip performance and power consumption [1]. Such variations come from process variations such as Le and Vt [2]–[4], as well as supply voltage and temperature variations. Process variations cause timing uncertainty. Current design methods for on-chip interconnects use pessimistic approaches where designs are assumed at their worst-case corners. Typically, an initial design solution is simulated. Monitoring the critical nets, an incremental technique is used with a number of iterations until the design meets its specification [5], [6]. The worst-case scenarios in measuring coupling noise are also assumed, i.e., when the aggressor noise peaks match the victim switching time in the same or opposite direction. Such an approach often leads to overdesigning circuits causing unnecessary elevation of power and other reliability problems. Statistical design methods have been proposed in the past to model the impact of process variations. However, all the existing methods deal almost exclusively with modeling delay variations of logic gates [7] or physical variations of interconnect wires, such as wire width and thickness variations [8], [9]. This paper deals with a method of analytically analyzing statistical behavior of multiple coupled interconnects with an uncertain signal arrival time at each interconnect input. The goal is to determine the statistical behavior of signal transmission from one point to another point in a circuit under the influence of uncertain multiple coupling sources.

The statistical behavior of such a delay can be obtained by Monte Carlo simulations of the circuit involved. However, Monte Carlo simulations are expensive, faster analytical methods with enough accuracy is needed to deal with complex very large scale integration (VLSI) designs. The method discussed in this paper achieves both goals of having an analytical-based faster method and high enough accuracy by calculating delay-change characteristics with respect to relative signal arrival times between the aggressors and the victim. The statistical behavior of signal delay through the coupled interconnect can then be easily obtained through analytical methods rather than Monte Carlo simulations. The remaining part of the paper is organized

Manuscript received June 25, 2003; revised October 29, 2003. This paper was recommended by Associate Editor F. N. Najm.

The authors are with the Department of Electrical and Computer Engineering, Colorado State University, Fort Collins, CO 80523 USA (e-mail: chen@engr.colostate.edu).

Digital Object Identifier 10.1109/TCAD.2004.837720