Post-Route Gate Sizing for Crosstalk Noise Reduction

Murat R. Becer, David Blaauw*, Ilan Algor, Rajendran Panda, Chanhee Oh, Vladimir Zolotov and Ibrahim N. Hajj†
Motorola Inc., Univ. of Michigan Ann Arbor*, Univ. of Illinois Urbana-Champaign†

ABSTRACT

Gate sizing is a practical and a feasible crosstalk noise repair 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 post-route 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.

Categories and Subject Descriptors

I.6.5 [**Simulation and Modeling**]: Model Development; G.2.2 [**Discrete Mathematics**]: Graph Theory — Path and circuit problems

General Terms

Algorithms

Keywords

crosstalk noise repair, gate sizing

1. 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.

While recent literature proposes a number of crosstalk noise analysis and avoidance methods, in this paper, we focus on correcting the identified crosstalk noise problems in the post-route design stage. Noise can be reduced through routing and interconnect optimization (wire spacing, wire widening, controlling coupling length and position) [1], buffer insertion [2] and driver sizing. In the post-route design stage, it is not desirable to use techniques that would require re-routing which can result in significant changes in net lengths and neighbors, possibly resulting in new noise failures that did not exist initially and non-converging design iterations. After routing is completed, noise failures can be 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 [3, 4]. A gate sizing method to reduce crosstalk induced delay noise is proposed in [3] and is based on a crosstalk noise aware static timing analysis. In [4], a post-route 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 post-route gate sizing algorithm for crosstalk noise reduction. Due to the non-linear dependence of crosstalk noise to the interconnect and driver parameters, the problem of post route gate sizing for crosstalk noise reduction is a non-linear optimization problem. The overwhelming system size in today's highly coupled interconnects makes it impractical to solve the non-linear 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 post-route 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 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 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. In Section 2, 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 3. Section 4 contains closing remarks.

2. PROPOSED GATE SIZING METHOD

In a post-route design stage, detailed information on the topology, neighbors and drivers of all the nets in the design is available. First, we perform an accurate post-route static noise analysis on the design, utilizing timing and logic correlation information to avoid false failures [5]. 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 'Noise Rejection Curve' method [5] where the slack is defined as ($noise\_slack = output\_noise\_threshold - V(receiver\_output)$). Note that the proposed algorithm is independent of the noise failure criterion being used.

2.1 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.

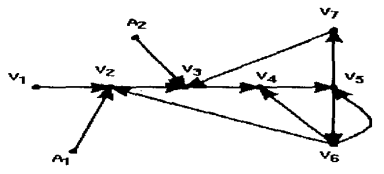

Figure 1: A sample noise graph

Figure 1 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. 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.

2.2 Sizing Algorithm

Our algorithm can be summarized as follows. After constructing the noise graph as explained in the previous subsection, 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 pseudo-code of the proposed algorithm is shown below:

```
Algorithm: Post-route 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 Size_down_type_$A$_vertices($G$)
3 Analyze_cycles($G$)
4 Break_cycles($G$)
5 $G_s$ = Topologic_sort($G$)
6 repeat until convergence
7     for each vertex $v$ in $G_s$
8         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 compexity 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. As the driver of vertex u is sized up, if the noise change on vertex v is not significant, i.e., $\Delta(noise_v)/\Delta(size_u)$ 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. 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 $O(|V| + |A| + |E|) \times sizeof(largest\ cycle)$. 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 seperate 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 Equation (1).

$$\begin{aligned} cs(C) \; = \; & \left|\frac{\delta N_2/\delta s_1}{\delta N_1/\delta s_1}\right| \times \left|\frac{\delta N_3/\delta s_2}{\delta N_2/\delta s_2}\right| \times \cdots \times \\ & \left|\frac{\delta N_n/\delta s_{n-1}}{\delta N_{n-1}/\delta s_{n-1}}\right| \times \left|\frac{\delta N_1/\delta s_n}{\delta N_n/\delta s_n}\right| \end{aligned} \quad (1)$$

where N_i is the noise on vertex i and s_i is the size of the gate at vertex i. Each term in Equation (1) represents the consequences of sizing up a gate in the cycle C. The i^{th} term, is the ratio of the sensitivity of induced noise on vertex $i+1$ to the size of 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 non-convergent 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. Although these cycles will be missing an edge, the effect of these eliminated edges are taken into account when we iterate the sizing process 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 (Break_cycles(G)) ensures that 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 $O(log|V|loglog|V|)$ [6]. 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 up-stream 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 pseudo-code 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. 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.

```
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
```

3. RESULTS

In this section, we present results of our algorithm on three large designs. The circuits used for experiments are chip_1, which has 31489 nets, chip_2, which has 39200 nets and chip_3 with 165481 nets. All three designs are actual high performance ICs in 0.18μ technology and the number of nets reflect the number of top level nets analyzed by the noise analysis tool. 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 1 shows the noise reduction results and Table 2 shows some statistical information on the runs. From Table 1, we can see that number of nets that fail the noise criterion goes down significantly, as much as by 98%. The last two

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 1: Noise reduction results

	↓ a	# ver			# cy		# edg			
		in	ac	bc	hs	ls	in	el	ac	bc
1	22	84	84	79	6	0	39	34	34	23
2	8	90	90	90	0	2	12	12	10	10
3	22	602	602	602	0	72	217	194	123	123

Table 2: Some statistics

columns in Table 1 show the maximum and average peak noise voltage reduction.

Table 2 presents the following information in column order: Number of aggressor gates that have been sized down [↓ *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" [*#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 Figures 2 and 3, 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. It can be seen from Figure 2 that, noise on most nets has been reduced and only on some nets noise has increased slightly. Figure 3 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 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. 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 2 iterations in all 3 chips and required 163, 150 and 780 seconds runtime respectively on an UltraSparc-II machine.

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.

4. CONCLUSION

In this paper, we presented a post-route 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. 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.

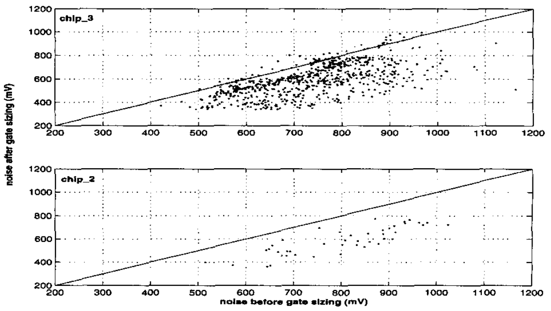

Figure 2: Noise peak changes at receiver input

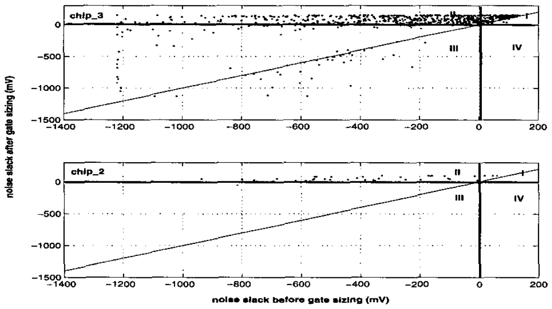

Figure 3: Noise slack changes at receiver output

5. REFERENCES

[1] P. Saxena and C. L. Liu. Crosstalk minimization using wire perturbations. In *Proceedings of Design Automation Conference DAC*, pages 100–103, 1999.

[2] C. J. Alpert, A. Devgan, and S. T. Quay. Buffer insertion for noise and delay optimization. In *Proceedings of Design Automation Conference DAC*, pages 362–367, 1998.

[3] T. Xiao and M. Marek-Sadowska. Gate sizing to eliminate crosstalk induced timing violation. In *Proceedings of ICCD*, pages 186–191, 2001.

[4] M. Hashimoto, M. Takahashi, and H. Onodera. Crosstalk noise optimization by post-layout transistor sizing. In *Proceedings of ISPD*, pages 126–130, 2002.

[5] 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 *Proceedings of Design Automation Conference DAC*, pages 233–238, June 2000.

[6] G. Even, J. Naor, B. Schieber, and M. Sudan. Approximating minimum feedback sets and multi-cuts in directed graphs. In *Int. Conf. on Integer Prog. and Combinatorial Optimization*, pages 14–28, 1995.