Simultaneous Shield and Buffer Insertion for Crosstalk Noise Reduction in Global Routing *

Tianpei Zhang and Sachin S. Sapatnekar
Department of Electrical and Computer Engineering, University of Minnesota
E-mail: zhangt, sachin@ece.umn.edu

Abstract

We present a method for incorporating crosstalk reduction criteria into global routing under an innovative power supply architecture, while considering the constraints imposed by limited routing and buffering resources. An iterative procedure is employed to route the signal wires, assign supply shields, and insert buffers so that both buffer/routing capacity and signal integrity goals are met. In each iteration, shield assignment and buffer insertion are considered simultaneously via a dynamic programming-like approach. Our noise calculations are based on Devgan's noise metric, and our work shows, for the first time, that this metric shows good fidelity on average. Experimental results on testcases with up to about 10,000 nets point towards an asymptotic run time that increases linearly with the number of nets. Our algorithm achieves noise reduction improvements of up to 53% and 28%, respectively, compared to methods considering only buffer insertion, or only shield insertion after buffer planning.

1 Introduction

Interconnect performance issues have become dominant in determining the performance of a circuit. In addition to considering traditional metrics, it is important to integrate the analysis and optimization of interconnect crosstalk noise into routing in order to maintain signal integrity. Functional noise is seen when a victim net changes its level due to the switching of its neighbor aggressor nets, and this could lead to circuit malfunction. Delay noise is caused when the victim and aggressor nets switch at the same time, which causes the effective coupling capacitance to become unpredictable, thus affecting the delay.

Various noise estimation and avoidance techniques have been proposed over the years [2, 4, 5, 7–9, 12, 14]. However, there are several considerations that are not fully addressed in previous work. First, although power supply wires are used as shields in [4] and [8], with the increasing number of crosstalk-affected nets, routing congestion and routability are major concerns since supply wires will also compete for the limited available routing resources. Therefore, a realistic crosstalk-conscious router must consider the trade-off between routing resource consumption and noise reduction. Second, modern designs employ a large number of buffers to achieve timing closure [17]. As a side-benefit, buffers can also effectively reduce noise by recovering the noise margin [2]. However, next generation design will see a larger number of nets requiring more buffers, and it is projected that at 32nm technology, a very large proportion of all cells will be buffers [13]. Under limited silicon area, this will produce high contention for the limited buffer resources. Hence a buffer-only noise reduction may not meet the noise requirements due to the contention. This motivates our simultaneous buffer and shield insertion scheme for functional noise reduction. With the help of shields, buffers can effectively block noise propagation.

Our work considers the problem of crosstalk noise reduction during global routing under restrictions on the availability of routing and buffer resources. We simultaneously allocate power supply wires as shields and insert buffers in the global routing phase in order to route nets under a noise budget. To utilize existing power supply wires, this method is presented under the backdrop of a power/ground (P/G) network architecture. The procedure results in a signal/power co-routing solution at the global level. While it will primarily deal with the functional noise as defined above, the insertion of supply shields with stable voltage levels between signal wires also provides the subsidiary benefit of greatly easing delay uncertainties and therefore relieves the delay noise. We have also incorporated considerations to insert a sufficient number of buffers to control the delays and slews on each signal line.

Our method works iteratively: starting with an initial global routing solution, an enumerative dynamic programming-like algorithm is used to simultaneously assign supply shields and buffers to meet the noise budget for each net, one at a time, to find a minimum cost solution for the net. Next, an iterative rip-up-and-reroute step is performed to better meet the routing and noise goal. We simultaneously take into account the limitations on routing/buffer resources and the needs for signal integrity and provide a global routing solution that is immune to capacitive coupling noise. For comparison purposes, we also implemented an intelligent greedy approach which is faster, but less effective in resource allocation.

2 Preliminaries

2.1 Global Routing and Buffer Model

As shown in Figure 1, our global routing model tessellates the entire chip into an array of grid cells, referred to as the *routing grid*. *Net N* consists of a set of electrically equivalent pins $\{s, p_1, p_2, ..., p_k\}$ distributed in different routing grid cells, that must be connected by wires, of which s is the source and $p_1, p_2, ..., p_k$ are the sinks. The dual graph of the routing grid tessellation is the *routing graph* G, which is shown in Figure 1(a) as the dashed lines. Connections among all of the pins will be routed over the routing graph G. Each edge in the routing graph corresponds to a boundary e_{ij} in the routing grid that connects grid cells i and j. The *grid length* L_e is defined as the center-to-center distance between two neighboring grid cells. Due to geometrical limitations on the boundary, we require that $W_e \leq C_e$, in which W_e is the total width (including wire spacing) used by signal and power lines passing the boundary, and C_e is the geometrical width of the boundary e, or the *boundary capacity*. Violation of this requirement results in *boundary overflow*. We follow a two-layer routing model in which horizontal and vertical lines are routed on different layers. The aim of routing is to eliminate boundary overflows while achieving other performance-related goals.

Our procedure also incorporates buffer insertion as an effective way to reduce delay and noise. We adopt the distributed buffer model proposed in [1], in which buffers are interspersed *within* the routing grids, and their exact location is undetermined until later in the design process. Figure 1(b) shows an inset view of a part of the routing grid where distributed buffers are inserted into a signal wire. For a grid cell i, the number of buffers available is denoted as B_i. If the number of utilized buffers is b_i, then $b_i \leq B_i$ must be satisfied, otherwise we have *buffer overflow*. To control the interconnect delay and slew rate, as in [1, 15], we enforce a constraint such that the maximum total in-

*This work was supported in part by the NSF under award CCR-0098117.

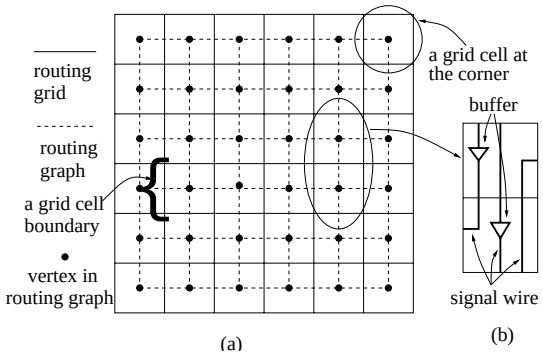

Figure 1: Routing grid and buffer insertion for signal wires.

terconnect length that can be driven by a buffer (gate) is the length of M grids.

2.2 Power Supply Architecture

A traditional power supply architecture is composed of a regular dense grid that traverses the entire layout area. However, different parts of the layout require different amounts of current, and the density of the grid does not have to be uniform. This may be exploited for routing flexibility [10].

We assume that the power grid is an array of variable density, and the integrity of the supply grid is maintained by ensuring that the average and minimum number of wires feeding every block exceeds a threshold. The layout is divided into blocks, and for each block i, we are given:

- a Minimum Average Number (MAN) of supply wires MAN_i per grid edge.
- a Minimum Number (MN) of supply wires MN_i running over each grid edge belonging to the block i.

Note that MAN_i and MN_i are both defined *per edge* and together define a basic structure of power network, thus enabling the power considerations to be incorporated even before a detailed power architecture is determined. Any extra power lines/shields beyond MN_i and MAN_i will only improve the power grid performance [10]. This model works well on intermediate metal layers like that of [11], where the variable number of power lines in adjacent blocks do not have to match exactly since they can be connected to each other through upper layers. All of the edge capacities are shared by signal wires and power supply wires. If signal wires utilize too much routing capacity at a boundary, then it is not possible to make enough room for P/G wires. We refer to the difference between the required and actual number of P/G wires in a block as the *P/G shortage*. The P/G wires in the supply grid are used not only to carry power currents, but also work as shields between aggressor and victim signal wires to reduce noise; we will use the terms *supply wire* and *shield* interchangeably. Also for the consideration of design for manufacturing, filling the remaining routing capacity left by signal routing with supply wires can improve manufacturability and performance predictability which may be deterioted by the Chemical-Mechanical Planarization (CMP) step in manufacturing [21].

2.3 Fidelity of Devgan's Noise Metric

Devgan's noise metric [5] is employed in this paper to find the capacitive coupling noise and the corresponding noise margin. This metric provides an upper bound for the crosstalk noise in an RC circuit, and its calculation is very similar to that of Elmore's delay. An example of applying Devgan's noise metric to noise estimation is shown in Section 2.4.2. However, this metric is known to potentially result in large overestimates [3]. In the context of routing multiple nets, we argue that it is the *average* error of a noise metric over many nets that is important rather than the maximum. If the average error of the noise predictor is relatively low, then the overall utilization of shielding and buffering resources will be good; for a pessimistic noise metric, a large average error will result in the over-utilization of resources.

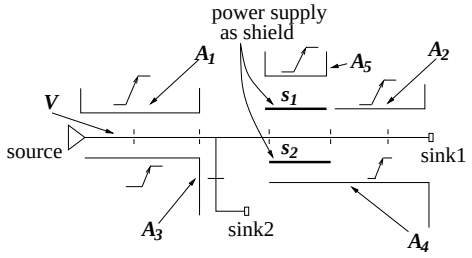

Figure 2: Switching noise on a victim net with shielding supply wires. V is the victim net, A_i, $i = 1, \ldots, 5$ are aggressor nets, and s_1, s_2 are power supply shields.

We verify the *fidelity* of Devgan's metric through a set of experiments. We randomly generate the pin locations of a circuit with several multiple-sink nets in a 6×6 grid, and then route the nets using the AHHK algorithm [6]. After that, the coupling capacitance are extracted and one net is randomly picked as the victim net while others as aggressors. With the victim net remains at a stable value, the aggressor nets switch adversely at the same time, and we simulate the coupling noise at the sinks of the victim net with both SPICE simulation and Devgan's metric. The above experiment is repeated 100 times, and we rank the 250 or so victim sinks in experiments according to their noises from SPICE simulation and Devgan's noise metric. The rank difference of each sink under the two metrics is then determined, and we take the relative error in rankings under SPICE simulation and Devgan's metric as a measure of the fidelity of Devgan's noise metric.

With different setup combinations of aggressor rise time and number of nets in circuit, we found the average error that corresponds to the distance in ranking (between Devgan's metric and under SPICE) is around 13% which suggests that Devgan's metric has acceptable fidelity in estimating and comparing crosstalk noises. On average, while comparing two structures, if one of them has lower crosstalk noise than the other under Devgan's metric, it is very likely to also demonstrate lower noise under a SPICE simulation.

These results lead to an important conclusion: *on average, Devgan's noise metric has an acceptable fidelity, and can be used as an estimation in the early phase of physical design*.

2.4 Shield Insertion and Noise Calculation

2.4.1 Arrangement of shields

The insertion of a supply wire between two signal wires will shield them from each other, so that there will be no significant capacitive coupling noise between the lines. Moreover, the insertion of the constant-voltage supply wire will reduce the delay uncertainty of adjacent signal nets, as compared to the case when that signal wire is next to a simultaneously and oppositely switching signal wire. We say that a side of a signal wire is provided *protection* if it neighbors a supply shield. Figure 2 shows five aggressor nets and a two-sink victim net with two supply wires as shields.

In the global routing phase, the exact positions of signal nets are still undetermined and hence neighborhood information is not fully available. We attempt to determine a *worst-case* scenario based on the information that is available. We assume that over a routing edge, if one side of a signal net is not placed next to a shield, it will (pessimistically) be adjacent to an aggressor net that will induce coupling noise on the net. Thus a signal wire must have supply lines placed on both sides to be fully protected. However, due to limited routing resources, a net may not be fully protected. We refer to the number of protected sides of net k at a specific routing edge e_{ij} as $P_{ij,\,k}$, and this can take a value of 0, 1, or 2.

If $P_{ij,\,k}$ is known for the signal wires across edge e_{ij}, the number of supply wires required for shielding can be found as follows. If there are S signal wires requiring protection on a single side, and D signal wires requiring protection on both sides, we must have p power supply as shields to achieve the protection, and

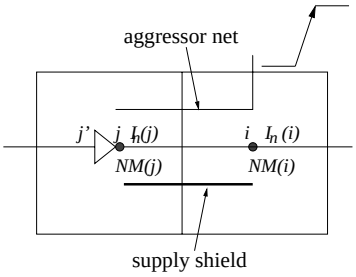

Figure 3: Calculation of noise margin by Devgan's metric.

$$p = \begin{cases} \frac{S}{2} + D & \text{if } S \text{ is even} \\ \frac{S+1}{2} + D & \text{if } S \text{ is odd} \end{cases} \quad (1)$$

It is easy to prove that an arrangement of supply and signal wires with the above numbers exists, so that the desired protection is feasible. The specific positions of the signal wires and supply wires will be handled by the detailed routing tools. Since supply wires and signal wires share the same routing resources, the capacity constraint of edge e_{ij} requires that

$$C_e \geq w_s \cdot s_e + w_p \cdot p_e \quad (2)$$

where C_e is the boundary capacity; w_s and w_p are the width (including the metal width and the spacing) of a signal wire and a supply wire respectively; s_e and p_e are the number of signal wires and supply wires passing the boundary e.

2.4.2 Noise calculation

A *noise margin*, NM_{spec}, is specified for each gate or buffer input in the circuit, and represents the largest noise voltage that will not result in a circuit malfunction. The choice of NM_{spec} is based on the fidelity of Devgan's metric, and can be selected by inflating the actually desired noise margin, so that it accounts for the overestimation in Devgan's metric. For any internal point i in an interconnect tree, the noise margin is recursively defined as:

$$NM(i) = \min_{all\ child\ node\ j}(NM(j) - V_n(i \leftrightarrow j)), \quad (3)$$

where $V_n(i \leftrightarrow j)$ is the noise voltage induced between i and j. A net is *noise free* if at both the source and any buffer output,

$$I_n \cdot R_d \leq NM \quad (4)$$

Here R_d is the gate driver resistance, and I_n is the induced noise current, calculated using Devgan's metric [5]. The noise reduction is illustrated by an example shown in Figure 3. A signal wire segment extends from the center of grid cell i to the center of its neighbor cell j with unit length coupling capacitance and resistance to be C_c and R_e respectively, and the aggressor voltage change rate is μ. Since the left side of the wire segment is shielded, only the right side aggressor will induce noise, giving us the following result according to Devgan's noise metric:

$$I_n(j) = I_n(i) + L_e \cdot C_c \cdot \mu \quad (5)$$

$$NM(j) = NM(i) - R_e L_e(\frac{1}{2} C_c L_e \mu + I_n(i)) \quad (6)$$

If the noise at j satisfies the constraint (4), the buffer will block the propagation of noise, and the noise margin at j' will be recovered to NM_{spec}, with noise current $I_n(j')$ back to 0 as well.

2.5 Problem Formulation

The formal statement of the problem is as follows. Given a tiling of a chip and the corresponding routing graph $G = (V, E)$, nets $N = \{n_1, n_2, ..., n_m\}$, the edge capacity C_{ij} for every edge $e_{ij} \in E$, and the buffer capacity B_i for each routing grid cell $i \in V$, the problem is to find a routing solution that:

1. determines the routes for each net on the routing graph,
2. satisfies the power/ground density constraints, i.e., the average and minimum density MAN_i and MN_i of each block must be met,
3. determines the grid cells in which a net is to be buffered, subject to the buffer capacity constraint,
4. finds $P_{ij,\ k}$ for each edge e_{ij} in the routing of net k, subject to the edge capacity constraint (2),
5. satisfies noise constraint (4) for all nets.
6. ensures that the total amount of interconnects that can be driven by a buffer (gate) is at most M grid units.

3 Routing and Crosstalk Reduction

Our approach to the problem is iterative and proceeds through three steps: congestion-driven global routing, a dynamic-programming-like simultaneous buffer and shield insertion procedure, and rip-up-and-reroute refinements. Steps 2 and 3 iterate until all constraints are satisfied, or no further improvement is possible. In the above buffer and shield insertion and rip-up-and-reroute, we process one net at a time and maintain a fixed order of all nets. We have experimentally found under different randomly chosen net orderings, the results change very little, as long as we maintain the same fixed net order through all of the iterations. This is due to the fact that the early iterations are seen to create good estimates of resource utilization, and this reduces the order dependence.

3.1 Step 1: Congestion Driven Routing

The signal wire routing procedure consists of two phases, similar to that in [1]. The first phase of routing constructs Steiner trees using the AHHK algorithm [6], and works as a fast estimator of the congestion map. The second phase performs a congestion-driven rip-up-and-reroute based on this initial solution, with the objective of minimizing the congestion cost over routing grid edges [1]. If there is still an overflow violation after this phase, more rip-up-and-reroute steps will be performed (with the same net order), as in [16].

We modify the congestion cost function during the congestion-driven routing to incorporate the consideration of power supply requirements. Since all of the routing capacity is shared by signal and power routing, signal routing results will determine the power supply structure, and thus must leave enough capacity for power supply to satisfy the average and minimum power supply densities MAN_i and MN_i for a block i of the routing region. The cost function for a signal wire traversing an edge e in block i is composed of two terms:

$$\text{routing cost} = cost_{traversing\ edge\ e} + cost_{passing\ block\ i} \quad (7)$$

This penalizes any violation of MN_i for edge e in block i and any violation of MAN_i of block i respectively. Both terms take the following form:

$$cost = \begin{cases} \frac{U}{R} & \text{if } R > 0 \\ 10^{-R} & \text{if } R \leq 0 \end{cases} \quad (8)$$

where R is the residual signal routing capacity on the edge [block] for the first [second] term. This value is calculated by subtracting, from the total edge [block] capacity, the power supply requirements and the capacity already used for signal routing, U. Note for the first term, the power supply requirement is MN_i; for the second term, it is MAN_i multiplied by number of edges in a block i, since MAN_i is defined *per edge*. The exponential form of the cost function after the capacity violation punishes the over-use of capacity from signal routing, and it will effectively avoid the aggregation of signal wires.

After every rerouting, each of the grid cells in the final path is added to the tree as an *Internal Node (IN)*, and a net will then be a set $\{s \cup P \cup IN \cup E\}$, where *s* is the source node, *P* is the set of sink pins, *IN* is the set of internal nodes, and *E* is the set of edges; this data structure is used for the procedure in Step 2, which follows this.

3.2 Step 2: Buffer and Shield Insertion

With a routing solution from the above step, we simultaneously allocate shield and buffer resources to each net so that the solution can meet the noise requirement while using least protection resources. Each net is processed individually, and the procedure traverses the tree

structure of the net in a bottom-up manner, starting from the sinks and moving towards the source, moving along one grid square at a time. Each tree is described in terms of nodes that correspond to the grid cells that it passes through. Assigning a direction to the tree from the source to the sinks, we refer to the grid cell that contains the immediate predecessor [successor] of a given node n in the tree as the parent [child] cell of the grid cell containing n. If a grid cell has more than two children, we can insert pseudo-nodes, so that the final tree is a binary tree for ease of later processing.

While traversing a net across a grid cell i, we have two methods for protecting it from crosstalk noise:

1. By deciding whether to insert a buffer or not at grid cell i: this corresponds to two possible buffer insertion configurations (0 or 1 buffer)
2. By protecting the net using supply shields on one side, on both sides, or choosing not to shield the net at all. If grid cell i is not the root of the tree, shield(s) may be inserted alongside the edge e_{ij} connecting current grid cell i and its parent grid cell j in the tree. This results in three possible configurations (0, 1, or 2 sides protected).

Therefore, in each bottom up step, we may have six possible configurations for the *protection structure*. However, we cannot locally determine at each grid point which scheme is globally optimal, and therefore an enumerative dynamic programming-like approach is adopted here in the same spirit as in Van Ginneken's algorithm [18].

3.2.1 Protection Cost and Solution Architecture

While traversing a net bottom-up, at grid cell i, we must find the protection cost corresponding to a protection structure, so as to measure the resource usage. For the protection cost related to buffer insertion, since the nets are processed one at a time, at any point in our insertion algorithm, the probability that an unprocessed net n_i crossing grid cell i will insert a buffer from i is $1/M$. Let p_i be the sum of these probabilities over all unprocessed nets crossing cell i, and the cost for insertion of $b_{req}(= 0 \text{ or } 1)$ buffer at a specific grid tile i is similar to that in [1]:

$$cost_{buffer\ i}(b_{req}) = \begin{cases} 0 & \text{if } b_{req} = 0 \\ \frac{b_i + p_i + 1}{B_i - b_i} & \text{if } b_{req} = 1 \\ & \text{and } b_i + b_{req} \leq B_i \\ \infty & \text{otherwise} \end{cases} \quad (9)$$

This cost function will significantly increase the cost penalty as buffer resources become contentious. For shielding cost, we can calculate the number of power supply wires required based on the number of sides to be protected N_{sh} and equation (1). In the same spirit to punish contentious resource usage, the shielding cost $cost_{shield\ ij}$ for a signal wire can be obtained from a similar form of equation as (9) above, but the predicted shield usage takes a different approach: each unprocessed signal net will probabilistically have 1 side to be protected (assuming equal probability for $N_{sh} = 0$, 1, or 2). Both the shielding cost and buffer cost are a measurement of the number of resources used, and they are approximately of the same order of magnitude. Therefore, we can combine them with a weighting factor λ (determined by resources availability) to develop a metric for resource usage, which we call the *protection cost* at grid cell i, denoted as PC_i:

$$PC_i(b_{req}, N_{sh}) = \lambda cost_{buffer\ i}(b_{req}) + cost_{shield\ ij}(N_{sh})$$

where j is the parent grid cell of grid cell i. This comprehensive cost function can be used as a metric to compare resource usages from different insertion schemes, and our goal is to find a minimum cost scheme satisfying the noise requirement, so as to resolve the contention for protection resources among nets.

At a cell i during the bottom-up traversal, the noise margin and noise current will vary according to the protection structure we choose, i.e., whether a buffer is inserted and the number of sides of the net that are shielded. If a buffer is inserted, the noise current will be "reset" to 0 and the noise margin set back to NM_{spec}. At each unbuffered location, depending on the number of sides of a signal wire that are shielded, we can have the following from equations (5) and (6):

$$\Delta I_n(N_{sh}) = (2 - N_{sh}) \cdot L_e \cdot C_c \cdot \mu \quad (10)$$

$$\Delta NM(N_{sh}) = R_e \cdot (\frac{1}{2} \cdot (2 - N_{sh}) L_e C_c \mu + I_n(i)) \quad (11)$$

where ΔI_n is the increase in the noise current due to the number of sides getting shielded $N_{sh} \in \{0, 1, 2\}$, and ΔNM is the noise margin decrease during the bottom-up step. To keep a record of the protection structure in the enumeration, we define a *protection solution* at routing cell i to be a 4-tuple $S = \{PC, NM, I_n, stru\}$, where NM is the noise margin at the end of the edge connecting grid cell i to its parent grid cell j, I_n is the noise current induced by the neighboring signal wire at the same point, and PC is the protection cost of the current solution. The last component, *stru*=$\{$*buffer*, $N_{sh}\}$, represents the protection structure of the solution, where *buffer* is a binary number representing the number of buffers utilized at grid cell i, and N_{sh} is the number of sides (0, 1 or 2) on which the wire is protected.

If a solution S_1 is provably *inferior* to another solution S_2 at the same grid cell, i.e., if $PC_1 > PC_2$, $NM_1 < NM_2$ and $I_{n1} > I_{n2}$, then it is pruned from the set of solutions. To satisfy the constraint that the total amount of interconnect can be driven by a buffer (gate) is at most M grid cells, we maintain a *solution set array* (*SSA*) of length M at each grid cell. Each element in *SSA* is a set of solutions, and the array is indexed from 0 to $M-1$. The solutions in $SSA[k], k \neq 0$ correspond to a total downstream interconnect of k grid cells to the nearest downstream buffer(s), and $SSA[0]$ stores the solutions that correspond to the insertion of a buffer at the current grid cell.

3.2.2 Protection Solution Building Algorithm

A unitary step in the enumeration algorithm is to build the solution set array at the current grid cell based on the arrays at its child cell(s). The pseudo code for the algorithm is listed in Figure 4. The main procedure in the algorithm calls function `Find_sol_set_array` at the source cell, and returns the minimum cost solution that satisfies the noise constraint. In function `Find_sol_set_array`, solutions in set $SSA[k],\ k \neq 0$ are propagated from solutions in the lower indexed sets of the children grid cells. The propagation is performed differently for one child and two children situations as shown in Figure 5 (a) and (b). During the propagation process, we update the noise margin and shielding information to form new solutions. At the same time, the least cost children solutions are selected and combined to build solutions in set $SSA[0]$ in which buffer is inserted right at current grid cell. In the above steps, functions `Propagate`, `Insert_buf` and `Merge` are called to build solutions. The solution pruning in the last steps will greatly reduce the solution set size. There are three types of pruning techniques employed here: the first discards those solutions that violate the noise constraints (4), the second discards the solutions that violate the buffer or wiring capacity, and the third removes any solution that is inferior to another solution in the same SSA.

While there is no concrete way of proving that the size of SSA will be small, the pruning technique works efficiently in practice; our experiments show that the number of solutions at each grid cell is limited between 3 and 60, and is less than 15 in most cases. We also observe that the asymptotic total run time increases linearly with the number of nets in the benchmark circuits.

3.3 Step 3: Refinement

After the simultaneous shield and buffer insertion for noise reduction, refinement steps are applied if there are still some nets cannot be protected from noise constraint violation. The procedure is similar to that used in global routing phase. We rip-up and reroute all of the nets in the same fixed order as before. After one net is ripped up, it is rerouted immediately by the rerouting algorithm described in Section 3.1. However, the cost function in rerouting is now the combination of both the wiring congestion cost and the buffer congestion cost. This will drive the net to go through regions where wiring capacity and buffers are abundant. The dynamic programming-like algorithm for simultaneous shield and buffer insertion is then applied. After all of

Algorithm: `Shield_buffer_insertion_for_noise_reduction`

Input: Net $N = \{s \cup P \cup IN \cup E\}$
Output: A protection solution with least protection cost at source *s*

1. *SSA* = `Find_sol_set_array`(*s*)
2. return protection solution $S = \{PC, NM, I_n, stru\} \in SSA$, with *PC* is minimized.

Function: `Find_sol_set_array`

Input: t is the grid cell to be processed.
Output: The *SSA* of this grid cell.

1. $SSA[i] = \Phi$ for $i = 0, 1, \ldots M-1$
2. if $t \in P$, is a leaf
   for i = 0 to $M-1$
   $SSA[i] = SSA[i] \cup \{0, NM_{spec}, 0, \Phi\}$;
3. else if *t* has one child *l*
   SSA_l = `Find_sol_set_array`(*l*);
   for i = 1 to $M-1$
   for each $S_l \in SSA_l[i-1]$
   $SSA[i] = SSA[i] \cup$ `Propagate`(S_l, t);
4. Take minimum PC solution S_m from SSA_l,
   $SSA[0] = SSA[0] \cup$ `Propagate(Insert_buf`$(S_m, t), t)$;
5. else if *t* has two children *l* and *r*
   SSA_l=`Find_sol_set_array`(*l*); SSA_r=`Find_sol_set_array`(*r*);
   for i = 2 to $M-1$
   for each $S_l \in SSA_l[j]$, $S_r \in SSA_r[k]$ and j+k=$i-2$
   $SSA[i] = SSA[i] \cup$ `Propagate(Merge`$(S_l, S_r), t)$;
6. Take min $S_{ml}.PC + S_{mr}.PC$ solutions S_{ml},S_{mr} from SSA_l, SSA_r;
   $SSA[0] = SSA[0] \cup$ `Propagate(Insert_buf(Merge`$(S_l, S_r), t), t)$;
7. Take minimum PC solution S_{ml} from SSA_l;
   for i = 1 to $M-1$
   for each $S_r \in SSA_r[i-1]$
   $SSA[i] = SSA[i] \cup$ `Propagate(Merge(Insert_buf(`S_{ml},t),S_r),t);
8. Take minimum PC solution S_{mr} from SSA_r
   for i = 1 to $M-1$
   for each $S_l \in SSA_l[i-1]$
   $SSA[i] = SSA[i] \cup$ `Propagate(Merge(`S_l,`Insert_buf(`S_{mr},t)), t);
9. Prune solution set array SSA;

Functions:

`Propagate`(S, t) /* Extend solution by one grid length upward*/
if t is source, return S;
else return $\{S.PC + PC_t(0, N_{sh}), S.NM - \Delta NM(N_{sh}), S.I_n + \Delta I_n(N_{sh}), S.stru \cup \{0, N_{sh}\}\}, N_{sh} = 0, 1, 2$;

`Insert_buf`(S, t) /* Insert a buffer to existing solution S */
return $\{S.PC + PC_t(1, 0),\ NM_{spec}, 0,\ S.stru \cup \{1, 0\}\}$;

`Merge`(S_l, S_r) /* Merge two solutions */
return $\{S_l.PC + S_r.PC, min(S_l.NM, S_r.NM), S_l.I_n + S_r.I_n, S_l.stru \cup S_r.stru\}$;

Figure 4: Algorithm for building protection solution.

the nets have been ripped up, rerouted, and then protected, the whole refinement step will be performed again if there is still some noise violation. However, our experimental results shown that there will not be much improvement after the third iteration. To provide additional protection from noise, in the last step, the unprotected nets will greedily take all of the unused wiring and buffer capacities along its path; however, this step is optional.

4 Experimental Results and Conclusion

Our algorithm is implemented in C++ on a Linux PC with a 2.8GHz CPU and 1GB memory. Out of the 12 benchmarks, the first ten benchmarks in Table 1 are obtained from the authors of [10]. The largest benchmarks *syn1* and *syn2* with over 10,000 nets are randomly generated[1]. We superimpose a grid over the the floorplan so that the geometry of each grid cell is almost a square. The number of buffers in each grid cell is generated randomly and the total number of buffers is listed in Table 1. We divide the design into several blocks, which correspond to different styles of circuits, such as control logic, data path, etc, and in our experiments, we use 7 blocks. The power supply requirements

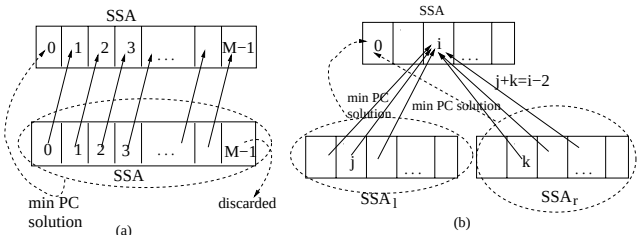

Figure 5: Updating SSA for one and two child grid cells.

MAN_i and MN_i are randomly generated but in a balanced manner across the chip (in practice, these will be dictated by the functional blocks). We also assume that a power grid wire is twice the width of a signal wire. The routing edge capacities are assigned as shown in the table, in the units of signal wire width. We assume that the grid length $L_e = 600\mu$m, that for all gates, the noise margin specification[2] $NM_{spec} = 0.4$V under a V_{dd} of 1.8V, and that the aggressor voltage change rate $\mu = 9 \times 10^9$V/s. The technology parameters used in the experiments are derived from [20] and [22] for the 0.18 μm technology: unit length coupling capacitance C_c=0.0583fF/μm, unit length resistance R_e=0.373Ω/μm, and buffer driver resistance R_d=180Ω.

We compare our results with those of two other methods. The **first method** is similar to the idea of [2], in which only buffers are inserted to reduce the noise, and the shielding effects are not considered. The buffers are also inserted by a dynamic programming-like algorithm, trying to achieve the noise constraint with the fewest number of buffers. The **second method** that we compared against is a greedy approach, in which buffers are first assigned in the same way as [1]. With the buffer positions known, we attempt to insert shield wires for each routing edge. For each net, the greedy shield insertion is composed of two steps:

1. We use a bottom-up approach, using every possible shield along the routing edges to meet the noise constraint.
2. If step 1 is successful, it may be the case that more than enough shields have been inserted. We then follow a top-down peel-off procedure to remove all of the unnecessary shields and buffers until the noise constraint or driving length constraint has been violated. The peel-off is greedy in the sense that a shield that is closer to the source of a tree will be removed greedily first. If there are multiple choices at any step in the top-down process, the branch with the higher noise margin will have its shield peeled off first.

Both of the above comparison methods employ the same routing and rerouting procedure as our approach.

The experimental results are listed in Table 1. The first eight columns show some basic properties of the circuits. Next, the results of our method which introduces buffers and shields (BS), the first method which introduces buffers only (B), and the greedy buffering and shielding method (G) are shown. Empirically, results show that the asymptotic run time of our buffer and shield insertion algorithm is linear in the number of nets, and our algorithm scales easily to cases with over 10,000 nets. The B and G columns show that these methods can result in noise protection failures on as many as 53% (circuit $apte$) and 28% (circuit $ami33$) of the total number of nets. In comparison, our simultaneous shield and buffer insertion approach has achieved the protection goals successfully without much sacrifice in speed, and in all cases, all of the nets meet the noise constraints.

The buffer-only approach shows poor performance because of the restricted number of buffers that are available. This will become more of an issue in future technologies, as projected by [13]. The greedy approach, on the other hand, performs buffer insertion and shield protection in separate steps, and no concerns of noise constraint are con-

[1]We did not use the ISPD98 placement benchmarks, because most of the nets in it are very short, which makes buffer insertion unnecessary, and cannot be used to illustrate the buffer contention problems that are projected for future technology nodes. This is consistent with the experience of the authors of [19].

[2]The noise threshold chosen here can be user-specified and is used to identify the nets with the largest noise, rather than as an exact predictor for the noise value. Since Devgan's metric has fidelity but not accuracy, the nets violating this threshold will indeed be those with the highest noise.

Circuit	# of nets	Available buffers	EC	Grid	M	Average		# noise violation nets			Run time		
						MAN	MN	BS	G	B	BS	G	B
ami33	112	3011	9	30×33	6	3.2	1.5	0	31	46	5s	4s	8s
ami49	368	6889	16	30×33	7	4.2	1.5	0	66	103	12s	9s	20s
apte	77	1811	9	30×33	6	3.4	1.7	0	6	41	5s	3s	5s
hp	68	2386	9	30×33	5	3.5	2.2	0	14	21	3s	2s	4s
playout	1294	15884	56	30×33	7	18.0	5.5	0	165	568	51s	36s	69s
a9c3	1148	11847	44	30×33	6	13.0	5.6	0	106	481	37s	29s	50s
ac3	200	4034	14	30×33	6	4.6	2.4	0	20	85	9s	6s	12s
hc7	430	7938	26	30×33	7	7.8	4.0	0	59	122	13s	10s	22s
n200b	1714	16903	65	33×33	6	19.6	9.4	0	278	850	63s	29s	48s
n300	1893	23295	68	33×33	7	19.9	8.4	0	159	879	70s	33s	54s
syn1	10086	87773	334	40×40	6	96.5	60.7	0	1057	4836	508s	331s	563s
syn2	10486	90577	348	40×40	6	102.8	61.3	0	1345	4963	637s	398s	558s

Table 1: Comparisons of routing and noise protection results. EC is the edge capacity; BS represents the simultaneous buffer and shield insertion algorithm; G represents the greedy algorithm; B represents the buffer-only algorithm.

Circuit	Overflow	
	BS	G
ami33	0	416
ami49	0	583
apte	0	22
hp	0	168
playout	0	1774
a9c3	0	981
ac3	0	228
hc7	0	1054
n200b	0	4666
n300	0	1604
syn1	0	30524
syn2	0	36634

Table 2: Overflow of routing and protection if 100% protection is achieved.

sidered in buffer insertion, resulting in an inferior performance to our integrated buffer and shield insertion solution.

An additional advantage of our approach is the adaptive P/G architecture, which enables a flexibility between the requirements of signal and power routing, so that both routing and P/G requirements are simultaneously met. For all three algorithms in Table 1, the routing overflow are almost 0 for all benchmarks, and is hence not listed. However, in cases where more nets must be protected to resolve the remaining noise violations, extra routing resources must be employed, leading to overflows. Table 2 reports the overflow results for our algorithm and the greedy algorithm if 100% protection is desired (the buffer-only algorithm does not use shield resources, and is omitted). The results show that much more routing resources have to be sacrificed to obtain a good protection for the greedy algorithm, while our algorithm can successfully achieve a good protection without extra routing overflow.

Due to the pessimistic nature of Devgan's metric, our buffer and shield insertion algorithm may over-optimize and use more than enough protection resources to accomplish full protection, or under stringent protection resources, may result in false-failures. To compensate for the pessimism of Devgan's metric, we heuristically inflate the actual specified noise margin in practice. If chosen carefully, the inflated noise margin as input to our algorithm will generate protection solutions that require fewer protection resources, but still satisfy the original noise margin requirement. For example, we have inflated the specified noise margin from the actual 0.4V to be 0.5V and 0.6V respectively. The protection solutions are simulated with SPICE, and the results are listed in Table 3. Due to long run times, we randomly selected up to 700 nets from each circuit for simulation; for smaller benchmarks such as *hp*, all nets were simulated. As can be seen, with NM_{spec} inflated to be 0.5V, almost 100% of the solutions can still satisfy the original 0.4V noise margin; while this percentage drops to about 90% when NM_{spec} is inflated to 0.6V. Practically, we may choose to inflate NM_{spec} by about 25% to acquire a good yet economic solution.

Circuit	$P_{0.5}$	$P_{0.6}$	Circuit	$P_{0.5}$	$P_{0.6}$
ami33	100%	93.0%	*ac3*	97.4%	95%
ami49	100%	90.6%	*hc7*	100%	93.8%
apte	100%	81.3%	*n200b*	100%	95.0%
hp	97.6%	91.1%	*n300*	100%	92.3%
playout	99.7%	91.6%	*syn1*	100%	94.1%
a9c3	100%	93.1%	*syn2*	100%	95.2%

Table 3: Protection rate with inflated NM_{spec}. $P_{0.5}$ and $P_{0.6}$ are the percentage of nets getting fully protected under SPICE simulation with inflated NM_{spec} at 0.5V and 0.6V respectively.

We have shown in this paper a method for simultaneously inserting supply shields and buffers during global routing to reduce crosstalk noise under a novel power supply architecture. Experimental results show that this method can route nets to meet both capacity and noise constraints. It is more effective than noise reduction using a buffer-only approach or a greedy approach.

References

[1] C. Alpert, J. Hu, S. S. Sapatnekar and P. Villarrubia, "A Practical Methodology for Early Buffer and Wire Resource Allocation," *Proc. DAC*, 2001, pp. 189-194.

[2] C. J. Alpert, A. Devgan and S. T. Quay, "Buffer Insertion for Noise and Delay Optimization," *IEEE Trans. on Comput.-Aided Design*, 18(11), November 1999, pp. 1633-1645.

[3] M. Kuhlmann and S. S. Sapatnekar, "Exact and Efficient Crosstalk Estimation," *IEEE Trans. on Comput.-Aided Design*, July 2001, pp. 858-866.

[4] S. Khatri, A. Mehrotra, R. Brayton and A. Sangiovanni-Vincentelli, "A Novel VLSI Layout Fabric for Deep Sub-Micron Applications," *Proc. DAC*, 1999, pp. 491-496.

[5] A. Devgan, "Efficient Coupled Noise Estimation for On-chip Interconnect," *Proc. ICCAD*, 1997, pp. 147-151.

[6] C.J. Alpert, T.C. Hu, J.H. Huang, A.B. Kahng and D. Karger, "Prim-Dijikstra Tradeoffs for Improved Performance-Driven Routing Tree Design," *IEEE Trans. on Comput.-Aided Design*, 14(7), July 1995, pp. 890-896.

[7] H. Zhou and D. F. Wong, "Global Routing with Crosstalk Constraints," *Proc. DAC*, 1998, pp. 374-377.

[8] T. Xue, E. S. Kuh and D. Wang, "Post Global Routing Crosstalk Risk Estimation and Reduction," *Proc. ICCAD*, 1996, pp. 302-309.

[9] M. R. Becer, D. Blaauw and I. N. Hajj, "Early Probabilistic Noise Estimation for Capacitively Coupled Interconnects," *Proc. SLIP*, 2002, pp. 77-83.

[10] H. Su, J. Hu, S. S. Sapatnekar and S. R. Nassif, "Congestion-driven Codesign of Power and Signal Networks," *Proc. DAC*, 2002, pp. 64-69.

[11] P. Saxena and S. Gupta, "Shield Count Minimization in Congested Regions," *Proc. ISPD* 2002, pp. 78-83.

[12] S. B. K. Vrudhula, D. Blaauw and S. Sirichotiyakul, "Estimation of Likelihood of Capacitive Coupling Noise," *Proc. DAC*, 2002, pp. 653-658.

[13] P. Saxena, N. Menezes, P. Cocchini and D. A. Kirkpatrick, "The Scaling Challenge: Can Correct-by-Construction Design Help," *Proc. ISPD*, 2003, pp. 51-58.

[14] J. D. Z. Ma and L. He, "Towards Global Routing with RLC Crosstalk Constraints," *Proc. DAC*, 2002, pp. 669-672.

[15] F. F. Dragan, A. B. Kahng, I. Mandoiu and S. Muddu, "Provably Good Global Buffering Using an Available Buffer Block Plan," *Proc. ICCAD*, 2000, pp. 358-363.

[16] R. Nair, "A Simple Yet Effective Technique for Global Wiring," *IEEE Trans. on Comput.-Aided Design*, 6(2), March 1987, pp. 165-172.

[17] J. Cong, "An Interconnect-Centric Design Flow for Nanometer Technologies," *Int. Symp. on VLSI Tech.*, Taipei, Taiwan, June 1999, pp. 54-57.

[18] L.P.P.P. Van Ginneken, "Buffer Placement in Distributed RC-tree Networks for Minimal Elmore Delay," *Proc. ISCAS*, 1990, pp. 865-868.

[19] J. Cong, T. Kong and D. Z. Pan, "Buffer Block Planning for Interconnect-Driven Floorplanning," *Proc. ICCAD*, 1999, pp. 358-363.

[20] J. Cong, "Challenges and Opportunities for Design Innovations in Nanometer Technologies," Invited Semiconductor Research Corporation Design Sciences Concept Paper, January 1998.

[21] P. Gupta and A. B. Kahng, "Manufacturing-Aware Physical Design," *Proc. ICCAD*, 2003, pp. 681-687.

[22] Semiconductor Industry Association, *National Technology Roadmap for Semiconductors*, 1997.