Simultaneous Routing and Buffering in Floorplan Design

Jyh Perng Fang, Yang-Shan Tong, and *Sao Jie Chen*

Graduate Institute of Electronics Engineering and
Department of Electrical Engineering
National Taiwan University
Taipei, Taiwan, ROC

E-mail: csj@cc.ee.ntu.edu.tw

Abstract

To deal with the floorplan design in a System-on-a-Chip (SOC), we have developed an EDA tool that simultaneuosly considers the problems of routing and buffer-insertion in floorplanning. This routing and buffering tool mainly contains a Manhattan routing (*MR*) algorithm and a maze-based between-buffer routing (*MBR*) algorithm. Since the processing speed of its *MR* is very fast, this tool can be integrated into an iterative floorplanning algorithm to promote the routability of a floorplan solution.

Keywords: Floorplanning, buffer insertion, global routing

1. Introduction

With the fast scaling of modern VLSI technology, interconnect delay plays a significant role in IC performance. During the routing phase, typically, congestion is estimated to generate a routable topology, and after a routing pattern is generated, buffer insertion is frequently used to reduce the interconnection delay.

On the congestion estimation problem, stochastic congestion estimation model [1] is suitable to be combined with another design algorithm [2]. On the other hand, among the literature concerning the buffer-insertion problems, Cong *et al.* [3] proposed a buffer block planning method, which allocate buffers into a feasible region, while delay constraint can be satisfied. In this method, the routing sometimes is restricted because net must go through predefined buffer blocks. Besides, buffer block planning is designed to meet timing constraint, it is hard to meet routing congestion constraint on the same time. Dragan *et al.* [4, 5] proposed an approximation method based on multi-commodity-flow to solve the routing problem given predefined buffer blocks. Alpert *et al.* [6] introduced a buffer site approach, where routing is guided by a Steiner tree and buffers are added at optimal location. However, the shortest length of the Steiner tree does not guarantee the lowest cost routing path.

Instead of dividing the problem into stages, our method tries to route and place buffers simultaneously. We consider net congestion constraint, buffer congestion constraint, and delay target of each net as our algorithm constraints, which were not considered simultaneously in previous works.

Two algorithms are combined in our method to route and place buffers: Manhattan Routing (*MR*) and Maze-based Between-buffer Routing (*MBR*). *MBR* is used mainly as a post-processing router for the nets that MR failed to route. The experimental results showed that the failure rate of using our routing-buffering method is apparently lower than that of using the existing techniques.

This work was supported by the National Science Council, R.O.C., under Grant NSC91-2215-E002-042.

This paper is organized as follows: Section 2 formulates the problems. Section 3 depicts the net and buffer density estimation model used, from which the corresponding cost functions are derived. Section 4 discusses our proposed routing and buffer-insertion method, which consists of a Manhattan Router and a Maze-based Between-buffer Router. The experimental results are presented in Section 5. The last section draws a conclusion.

2. Problem Formulation

Given a floorplan and the delay budget of each net, we want to determine the routing of each net and location of buffer insertion, such that the delay budget of each net is met and the routing/buffer congestion constraints of the whole floorplan are satisfied.

In our problem modeling, only two-pin nets are considered. We assume that multi-pin nets can be divided into several two-pin nets and treated in the same way.

After dividing the whole routing plane into tiles. we can create two matrices; one representing net upper bound, the other representing buffer upper bound.

The routing and buffer-insertion problem is thus formulated as follows:

Given: A tiling $G(V, E)$ of the chip, a set of nets $N = \{n1, n2, n3, \ldots\}$, a buffer upper-bound matrix $BUB(i, j)$, a net upper-bound matrix $NUB(i, j)$, locations of two end-points, and a delay budget for each net.

Goal: Route each net and insert buffers such that its delay budget is not violated and the following congestion constraints are satisfied.

Constraint: $n(i, j) \leq NUB(i, j)$, where $n(i, j)$ is the number of nets running through tile (i, j); $b(i, j) \leq BUB(i, j)$, where $b(i, j)$ is the number of buffers assigned in tile (i, j).

We adopt Sarkar *et al.*'s delay model [7] to calculate delay. We also adopt their IFR (independent feasible region) width and location formulas to calculate the locations where a buffer can be put. As in [3], [6], [7], key parameters for calculation are listed in Table 2.1. The values are based on a 0.18μm technology in NTRS'97 [9].

Some terminologies are listed in the following:

Manhattan Distance (*MD*): The MD between points $(x1, y1)$ and $(x2, y2) = |x1 - x2| + |y1 - y2|$.

Manhattan Box (*MB*): The MB is a rectangle having $(x1, y1)$, $(x2, y2)$ as its diagonal corner points.

Table 2.1 Parameter Values for Interconnect, Buffer, Driver, and Sink

r	unit length wire resistance ($\Omega/\mu m$)	0.075
c	unit length wire capacitance ($fF/\mu m$)	0.118
T_b	intrinsic delay for buffers (ps)	36.4
C_b	input capacitance of buffer (fF)	23.4
R_b	output resistance of buffer (Ω)	180
C_s	input capacitance of sink(fF)	23.4
R_d	output resistance of driver (Ω)	180

3. Models and Cost Functions

3.1 Net Density Estimation

Applying Lou *et al.*'s method [1], we can obtain the estimation of a single net quickly. A table-look-up method can be applied to speed up this estimation procedure [8]. For each net, after its net estimation matrix is calculated, we can store this matrix for future procedure. Also, adding up the net estimation matrix of each net gets the total net estimation matrix, which is needed for calculating net cost functions.

3.2 Buffer Density Estimation

We assume all feasible buffer numbers that are equal to or smaller than the buffer number with which net delay can be minimized appear with same possibilities. We also assume that a buffer is distributed evenly within its feasible range. Thus, we can calculate the buffer density estimation matrix of each single net; adding up the buffer density estimation matrix of each net creates a total buffer density estimation matrix.

3.3 Cost Functions

In our algorithm, net and buffer cost functions are defined in each tile. For a single tile (i, j), its net cost function is set to be:

$$NCF(i,j) = Exp\left(c1 \times \frac{NEV(i,j) - NUB(i,j)}{NUB(i,j)}\right), \quad if\, NUB(i,j) \neq 0;$$

$$NCF(i,j) = \infty, \quad if\, NUB(i,j) = 0.$$

where NCF: Net Cost Function,
NUB: Net Upper Bound (net constraint),
NEV: Net Estimation Value, and
$C1$: a constant.

And its buffer cost function is set to be:

$$BCF(i,j) = Exp\left(c1 \times \frac{BEV(i,j) - BUB(i,j)}{BUB(i,j)}\right), \quad if\, BUB(i,j) \neq 0;$$

$$BCF(i,j) = \infty, \quad if\, BUB(i,j) = 0.$$

where BCF: Buffer Cost Function,
BUB: Buffer Upper Bound (buffer constraint),
BEV: Buffer Estimation Value, and
$C2$: a constant.

4. Routing and Buffer-Insertion Algorithm

In our routing-buffer-insertion algorithm, the plane is divided into tiles. For a single tile, its BUB and NUB values are determined by occupied area of the tile and fabrication parameters. Nets are sorted by delay budgets. Nets with the tightest delay budgets (closest to 1) are routed first.

It should be noted that each net has its own NEV and BEV matrices and should be stored for later updating procedure. When a net is routed and its buffers are determined, total NEV and BEV matrices and the NUB and BUB values are updated and cost functions should be re-calculated.

MR procedure is carried out first. If the routing of a net failed during this procedure, this net is recorded for later processing.

4.1 Manhattan Router

The Manhattan Router (MR) procedure is used to route a net with minimal resources consumed. The path must reside in a Manhattan Box and no detour is permitted. If there are numerous paths satisfying this criterion, all we have to do is to choose a path with a minimal accumulative cost value and assign this path as the final routing path.

The min-cost path with buffers inserted has an optimal substructure, which makes dynamic programming possible. We formulate our algorithm as shown in Fig.1.

Given: two end points S and T; n buffer sets $\{B_{1,1}, B_{1,2}, \ldots, B_{1,m1}\}$, $\{B_{2,1}, B_{2,2}, \ldots, B_{2,m2}\}$, ..., $\{B_{n,1}, B_{n,2}, \ldots, B_{n,mn}\}$.
Goal: choose a min-cost path from S to T, which resides within the Manhattan Box with no detour. The path will intersect each buffer set with one element, choose this element as the buffer site.

Algorithm MR

```
1. Find min-cost paths from $S$ to all elements of the 1st
   buffer set $\{B_{1,1}, B_{1,2}, \ldots, B_{1,m1}\}$, store them as $\{SB_{1,1}, SB_{1,2}, \ldots, SB_{1,m1}\}$.
2. For ($i$ = 2 to $n$) do
       Use a previously stored min-cost path to find the
       min-cost paths from $S$ to all elements of $i$th buffer
       set and store them as $\{SB_{i,1}, SB_{i,2}, \ldots, SB_{i,mi}\}$.
3. Find the min-cost paths from all elements of the $n$th
   buffer set to $T$ and store them as $\{B_{n,1}T, B_{n,2}T, \ldots, B_{n,mn}T\}$.
4. Combine $\{SB_{n,1}, SB_{n,2}, \ldots, SB_{n,mn}\}$ and $\{B_{n,1}T,
   B_{n,2}T, \ldots, B_{n,mn}T\}$ to find a min-cost path from $S$ to $T$.
```

Fig. 1 Algorithm MR

In Fig. 1, the given n buffer sets are calculated according to Sarkar et al.'s algorithm [7] and exact one element in each buffer set $\{B_{n,1}, B_{n,2}, \ldots, B_{n,mn}\}$ will be chosen as a buffer site on the min-cost path from S to T. Since the chosen path from S to T has a min-cost, both the sub-path from S to a buffer site n and the sub-path from buffer site n to T should be min-cost sub-paths, which obviously have the property of optimal substructure. Besides, at each iteration of Step 2 in Fig. 1, the calculations of min-cost paths from S to each elements of the

(i+1)th buffer set are all based on the min-cost paths found at previous iteration $\{SB_{i,1i}, SB_{i,2i}, \ldots, SB_{i,mi}\}$. For the problem of the (i+1)th iteration, there apparently exists a property of overlapping subproblem. These two properties imply the correctness of the applicability of dynamic programming.

4.2 Analysis of *MR*

Consider a two-pin net with end points S and T and assume its Manhattan Box to be $l \times w$ tiles; also assume n buffers must be added in this net, with inter-buffer space $s = (l+w-1)/(n+1)$. The time complexity of our algorithm is described as follows:

Theorem 4.1: The time complexity of *MR* for a single net described above is $O(s^3 n \times (l+w))$, where s is determined mainly by fabrication specifications, actual tile size, and delay budget and can be treated as a constant. Thus, the original complexity is reduced to $O((l+w)^2)$.

Proof: Since each buffer set has $O(l+w)$ elements, each element of this buffer set has at most $2^{0.5} \times s$ connections with next buffer set, the time complexity of the shortest path of an adjacent buffer site can be shown to be $O(s^2)$. With n buffer sets to be connected with the adjacent buffer set, the total time complexity is $O(s^3 n \times (l+w))$. Since s is treated as a constant, and n (the total number of buffers to be added in this net) is roughly proportional to $l+w$, the complexity is reduced to $O((l+w)^2)$.

4.3 Maze-based Between-buffer Router

We proposed a Maze-based Between-buffer Router (*MBR*) to route a path between buffers, as shown in Fig. 2. Generally speaking, this method can find a path satisfying delay budget if it exists, but it needs longer computation time in contrast to *MR*. Thus, this method is designed mainly for post-processing those nets that have been failed during *MR*.

Given: Two end points S and T; a buffer set $B = \{B_1, B_2, \ldots, B_n\}$.
Goal: Choose a min-cost path from S to T, which going through a subset of B, and of which the total delay value does not violate any delay budget.

Algorithm MBR

```
1. Use MR to route a path from $S$ to all possible buffer
   sites, then store the route and delay value.
2. While (not all buffer sites are chosen as a new source)
   do
    2.1. new source $\leftarrow$ pick-up an unchosen min- delay
         buffer site
    2.2. Use MR to route paths from the new source to all
         un-chosen buffer sites, then update the minimum
         delay value if necessary.
3. Using MR to route paths from each buffer site to $T$,
   choose a path with a minimum delay value fitting the
   delay budget as the final routing path.
```

Fig 2 Algorithm MBR

To route nets between buffers, we try to find a path from S to all possible buffer sites using a maze router. Then we check the delay budget and choose a buffer site with the minimal path delay as a new source. From this new source, we route a path to all buffer sites except S, check accumulative delay of these paths, if the new delay is smaller than the previous one, update with this route and delay. Repeat this step until all buffer sites have been chosen as a source.

5 Experiment Results and Discussions

We take several random tile patterns as testbenches for the following two experiments, each pattern has about a thousand nets, each net has a delay budget from 1.00001 to 1.2.

5. 1 Routing with Different Schemes of *MR* and *MBR*

There are two schemes to apply our routing algorithms. One scheme is to route all nets with *MR* and record the failed nets; then route these failed nets (if any) using *MBR*. The other scheme is to route a single net with *MR*. If it succeeds, continue with the next net; otherwise route the net with *MBR*.

Table 5.1 Failed nets of Two Schemes

Circuit#	Net#	1st Scheme1 after MR	2nd Scheme 1 after MBR	Scheme 2 (MR+MBR)
1	944	77	71	70
2	1122	297	286	288
3	823	298	289	290
4	1114	180	172	170
5	858	47	44	44

Table 5.1 shows the comparisons of routing results with the above two schemes on a 20×20 grid. The table lists the number of failed-to-be-routed nets after running MR, after running MBR with Scheme 1, and after running MR+MBR with Scheme 2, respectively, for five random-generated testbench circuits. From these results, we observed that these two schemes differ little, which can be explained by an evident fact: over 95% nets are routed successfully using *MR*. This evidence implicates that most of the routing results are known after running a fast *MR*, and this information can be fed back to the floorplanning stage and used by the floorplan tool. *MBR* can then be regarded as a post processing router, which will be performed only if the *MR* result is not acceptable.

5.2 Routing with Different Net Orderings

We also want to know the relationship between net order and the routing result. To show it, we perform the same routing algorithm with our proposed ordering by sorted-delay and with other four random net orderings. Scheme 1 was chosen as our routing procedure. The results are summarized in Table 5.2, where the results of *Average*, *Max*, *Min* and *Numerous* are collected after 30 different random net ordering experiments.

Table 5.2 showed the superiority of our net ordering. The results of our proposed sorted-by-delay net ordering proves to be the best in each testbench circuit compared with the other net ordering methods.

Table 5.2 Failed nets of Different Net Orderings

Net Orderings	Circuit 1 (944)		Circuit 2 (1122)		Circuit 3 (823)		Circuit 4 (1114)	
	1st	2nd	1st	2nd	1st	2nd	1st	2nd
Delay-Sorted	77	71	297	286	298	289	180	172
Random order *Average*	79	78	307	304	304	300	185	180
Random order *Max*	87	86	318	316	312	308	211	193
Random order *Min*	72	72	299	296	293	290	178	173
Random order *Numerous*	77	75	309	305	308	304	186	179

5.3 Comparison with Buffer Site Algorithm

Based on the same benchmarks and technology parameters, we compare our method with Alpert *et al.*'s Buffer Site Approach [6]. First, we implemented the kernel of Buffer Site Algorithm, in which we consider 2-pin nets only. Then, we examined the impact of different *Li* (the number of tiles that a buffer can drive) to the routing, the evaluation is described by the number of failed nets. It is observed that when *Li >10,* there is no difference in the number of failed nets. This can be explained by the fact that when *Li* reached a specific value, almost all nets can put buffers within its path as long as the net is routed successfully. Besides, the following reasons of failure are observed: (1) fail to route a net, (2) fail to place buffers within *Li*, and mostly (3) fail to fit the delay value to our delay budget.

Table 5.3 Comparisons of Alpert's and Our Algorithm

Benchmark Circuits		Circuit 1	Circuit 2	Circuit 3	Circuit 4	Circuit 5
Total net #		944	1122	823	1114	858
Alpert's Algorith m	Fail net #	569	790	571	692	521
	Fail percentage	60.2%	70.4%	69.4%	62.1%	60.7%
Our Algorith m	Fail net #	70	288	290	170	44
	Fail percentage	7.4%	25.7%	35.2%	15.2%	5.1%
Improvement percentage		52.8%	44.7%	34.2%	46.9%	55.6%

We use the benchmark circuits as listed in Table 5.1 to compare Buffer Site Algorithm with ours, as shown in Table 5.3, in which *Li* = 11. It is surprisingly that the Buffer Site Algorithm have high rate of failed nets. Using our approach, the failed net percentage decreased by 34.2% to 55.6%. This is because we tried to customize the buffer-insertion scheme for each net, while their algorithm uses the same *Li* for every net. In VLSI design, especially in SoC design, different nets have different delay budget and their values can be roughly determined only after the logic design stage is completed. Our algorithm provides a good method for routing with a detailed delay budget specified for each net.

6. Conclusions

We proposed a routing-buffering tool to perform global routing and buffer insertion simultaneously for floorplan design. The algorithm includes a Manhattan routing (*MR*) algorithm and a maze-based between-buffer routing (*MBR*) algorithm. Our method considers net congestion constraint, which are not easily considered in Cong *et al.*'s and Sarkar *et al.*'s algorithms.

Besides, as the processing speed of *MR* is quite fast, we can easily integrate our *MR* into any iterative floorplanning algorithm to facilitate the routability of a floorplan solution.

References

[1] J. Lou, S. Thakur, S. Krishnamoorthy, and H. S. Sheng, "Estimating routing congestion using probabilistic analysis," *IEEE Trans. CAD*, vol. 21, No.1, pp. 32-41, Jan. 2002.

[2] C. W. Sham, W. C. Wong and Evangeline F. Y. Young, "Congestion estimation with buffer planning in floorplan design," *Proc. Intl. Symp. Physical Design*, 2002.

[3] J. Cong, T. Kong, and D. Z. Pan, "Buffer block planning for interconnect-driven floorplanning," *Proc. ICCAD*, pp. 385-363, 1999.

[4] F. F. Dragan, A. B. Kahng, I. I. Mandoiu, S. Muddu, and A. Zelikovsky, "Provably good global buffering using an available buffer block plan," *Proc. ICCAD*, pp. 104-109, 2000.

[5] F. F. Dragan, A. B. Kahng, I. I. Mandoiu, S. Muddu, and A. Zelikovsky, "Provably good global buffering by multiterminal multicommodity flow approximation," *Proc. ASP-DAC*, pp. 120-125, 2001..

[6] C. Alpert, J. Hu, S. Sapatnekar, and P. Villarrubia, "A practical methodology for early buffer and wire resource allocation," *Proc. DAC*, pp. 189-194, 2001.

[7] P. Sarkar, V. Sundararaman, and C.-K. Koh, "Routability-driven repeater block planning for interconnect-centric floorplanning," *Proc. Intl. Symp. Physical Design*, pp. 186-191, 2000.

[8] X. Young, R. Kastner and M. Sarrafzadeh, "Congestion estimation during top-down placement," *Proc. Intl. Symp. Physical Design*, 2001.

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