Simultaneous routing and buffering in SOC floorplan design

J.P. Fang, Y.-S. Tong and S.J. Chen

Abstract: An EDA tool to deal with the problems of routing and buffer-insertion in system-on-chip floorplanning simultaneously is developed. This routing and buffering tool mainly consists of a Manhattan routing (MR) algorithm and a maze-based between-buffer routing 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.

1 Introduction

With the fast scaling of modern VLSI technology the routability of a floorplan dominates the success of subsequent routing stage, also the interconnect delay plays a significant role in IC performance. Typically, congestion is estimated during the routing phase to generate a routable topology and after a routing pattern is generated, buffer insertion is frequently applied to reduce the interconnect delay.

For the congestion estimation problem a stochastic congestion estimation method [1] has been proposed and is suitable for combination with a buffer planning algorithm [2]. Concerning the buffer-insertion problems, Cong *et al.* [3] proposed a buffer block planning method to allocate buffers into feasible regions (FRs). The idea of FRs has been expanded by Sarkar *et al.* [4] into independent feasible regions (IFRs) to place more than one buffer on a single net. In the methods proposed [3, 4], the routing is sometimes restricted because the net must go through predefined buffer blocks. Besides, buffer block planning is designed to meet the timing constraint; it is hard to meet the routing congestion constraint at the same time. Dragan *et al.* [5, 6] proposed an approximation method based on multicommodity flow to solve the same routing problem. Alpert *et al.* [7] introduced a buffer site approach (BSA), where routing is guided by a Steiner tree and buffers are added at optimal locations rather than being predefined as in FR and IFR.

Instead of dividing the problem into a routing stage and a buffering stage, our method routes and places buffers at the same stage. In our algorithm we consider net congestion constraint, buffer congestion constraint, and delay constraint of each net simultaneously, which has not been considered previously.

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 postprocessing router for the nets that MR failed to route. The experimental results showed that the failure rate of our routing–buffering method is apparently lower than other existing techniques.

2 Problem formulation

Given a floorplan and the delay budget 'as defined in [3]' of each net we want to determine the routing path 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 modelling, only two-pin nets are considered because a multipin net can be divided into several two-pin nets and treated in the same way. We divide the whole routing plane into tiles, and create two matrices for recording net upper bound and buffer upper bound information.

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

Given: A tiling $G(V, E)$ of the chip, a set of nets $N = \{n1, n2, n3, \ldots\}$, a net upper-bound matrix ***NUB*** with each element denoted as $NUB(i, j)$, a buffer upper-bound matrix ***BUB*** with each element denoted as $BUB(i, j)$, 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:

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

We adopt the model presented by Sarkar *et al.* [4] to calculate delay and apply their IFR (independent feasible region) formula to calculate the regions where a buffer can be put. In the formula of IFR, the feasible region of the ith buffer is expanded from its optimal location by a distance decided by its width. The key parameters based on 0.18 μm technology in NTRS'97 [8] and used in [4] for calculating the optimal location and the width of the ith buffer are also adopted here, which include unit-length wire resistance $(0.075\,\Omega/\mu m)$, unit-length wire capacitance $(0.118\,fF/\mu m)$, input capacitance of a buffer or sink (23.4 fF), output resistance of a buffer or driver $(180\,\Omega)$, and intrinsic delay of a buffer (36.4 ps). Hereinafter, for convenience, the formula

IEE Proceedings online no. 20040072

doi: 10.1049/ip-cdt:20040072

Paper first received 20th February and in revised form 1st October 2003

The authors are with Graduate Institute of Electronics Engineering and Department of Electrical Engineering, National Taiwan University, Taipei, Taiwan, ROC
S.J. Chen is also with the IBM Thomas J. Watson Research Center on Sabbatical

for calculating IFR regions and its associated parameters are termed IFR formula and IFR parameters, respectively.

3 Models and cost functions

Applying IFR formula and IFR parameters to a routing path, the possible locations for buffer insertion are determined such that the delay constraint is not violated. This Section introduces the model and cost functions that we used in our method for satisfying the net congestion constraint and the buffer congestion constraint.

3.1 Net density estimation

Applying the stochastic model proposed by Lou *et al.* [1], we obtain the increased net density that a single net contributed to each tile in a tile model. For each net, on a tile graph, after its net-density estimation matrix is calculated we store this matrix for future procedure. Also, accumulating the net-density estimation matrix of each net gets the total net-density estimation matrix ***NEV***, in which each element of the net density estimation matrix is denoted as $NEV(i,j)$, which contributes to the calculation of net cost functions (see Section 3.3).

3.2 Buffer density estimation

Buffer-density estimation is much more complicated than net-density estimation. For each routed net to fit its delay budget the acceptable number of buffers to be added may vary in a range, and each buffer can be put in a rather wide range. For example, based on the IFR parameters, to route a net through 4800 μm Manhattan distance with a delay constraint of 1.17 in which delay constraint was defined in [3], we can use one, two, or three buffers, each with an IFR range of 3787, 1837, and 644 μm, respectively. Such a wide possibility makes buffer-density estimation difficult.

We assume all feasible buffer numbers that are equal to, or smaller than, the buffer number with which net delay can be minimised appear with same possibilities. We also assume that buffers are distributed evenly within their feasible ranges. Thus we can calculate the buffer-density estimation matrix of each single net; accumulating the buffer-density estimation matrix of each net creates a total buffer-density estimation matrix ***BEV***. Likewise, each element of the buffer density estimation matrix is denoted as $BEV(i,j)$, which contributes to the calculation of buffer cost functions (see Section 3.3).

3.3 Cost functions

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

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

$$NCF(i,j) = \infty, \quad if\ NUB(i,j) = 0. \tag{1}$$

where NCF is net cost function, NUB net upper bound (net constraint), NEV net estimation value, and $c1$ a constant. For comparable $NEV(i,j)$ and $NUB(i,j)$, $NCF(i,j) \cong c1$. On the contrary, if $NEV(i,j)$ is greater than $NUB(i,j)$, $NCF(i,j)$ becomes a big value, which will guide the routing procedure to avoid using possibly congested areas and keep these areas for later net routing.

In the same manner the buffer cost function is set to be

$$BCF(i,j) = c2 \times Exp\left(\frac{BEV(i,j) - BUB(i,j)}{BUB(i,j)}\right), \quad if\ BUB(i,j) \neq 0;$$

$$BCF(i,j) = \infty, \quad if\ BUB(i,j) = 0. \tag{2}$$

where BCF is buffer cost function, BUB buffer upper bound (buffer constraint), BEV buffer estimation value, and $c2$ a constant. Likewise, for comparable $BEV(i,j)$ and $BUB(i,j)$, $BCF(i,j) \cong c2$. On the contrary, if $BEV(i,j)$ is greater than $BUB(i,j)$, $BCF(i,j)$ becomes a large value, which facilitates avoiding buffer insertion at possibly congested areas and these areas are reserved for later buffer insertion.

Finally, for an $m \times n$ tile graph, the cost function CF for congestion evaluation is expressed as

$$CF = \sum_{j=1}^{n} \sum_{i=1}^{m} [NCF(i,j) + BCF(i,j)]. \tag{3}$$

Before illustrating our routing and buffering algorithms in the following Section, some terminologies used are described as follows.

Manhattan distance: The distance between points $(x1, y1)$ and $(x2, y2) = |x1 - x2| + |y1 - y2|$.
Manhattan box: A rectangle having source $(x1, y1)$ and destination $(x2, y2)$ as its diagonal corner points; the paths from source to destination are restricted inside the rectangle.
Buffer set: A collection of tiles that are candidates for buffer insertion.
Buffer site: A tile chosen from 'buffer set' for buffer insertion.

4 Routing and buffer-insertion algorithm

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

The net density matrix and the buffer density matrix are updated net by net. When a net is routed and its buffers are determined, the corresponding $NEV(i,j)$ and $NUB(i,j)$ are updated according to the routing path, while the corresponding $BEV(i,j)$ and $BUB(i,j)$ values are updated according to the locations of buffering, and as well, cost functions should be recalculated. If the routing of a net failed during this procedure, this net is recorded and needs further processing.

Also, the cost functions presented in Section 3.3 do not include delay constraint; this is because the IFR ensures the delay budget of a given net is not violated. Once the buffer sets are decided according to the IFR formula and IFR parameters, efforts are focused on the evaluation of net congestion and buffer congestion according to (1) and (2). From these evaluations and MR introduced in the following Section, we can separately select one buffer site from each buffer set for buffer insertion.

4.1 Manhattan router

The MR procedure is used to route a net with minimal cost such that the net congestion and buffer congestion are minimised. 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.

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 the ith buffer set and store them as $\{SB_{i,1}, SB_{i,2}, \ldots, SB_{i,mi}\}$.
3. Find the min-cost paths from T to all elements of the nth buffer set and store them as $\{TB_{n,1}, TB_{n,2}, \ldots, TB_{n,mn}\}$.
4. Combine $\{SB_{n,1}, SB_{n,2}, \ldots, SB_{n,mn}\}$ and $\{TB_{n,1}, TB_{n,2}, \ldots, TB_{n,mn}\}$ to find a min-cost path from S to T.

Fig. 1 *Algorithm MR*

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. In the figure, the given n buffer sets are calculated according to Sarkar *et al.*'s algorithm [4] and exact one element in each buffer set $\{B_{n,1}, B_{n,2}, \ldots, B_{n,mn}\}$ will be chosen as the 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 subpath from S to the ith buffer site and the subpath from the ith buffer site to T should be min-cost subpaths, which obviously have the property of optimal substructures. 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.

Take Fig. 2*a* as an example; the cost of each buffer in Fig. 2*a* is deliberately set to zero to simplify the illustration. In practice the cost of buffer congestion is included as well as the cost of net congestion. In Fig. 2*a*, two buffer sets, $\{a, b, c, d, e\}$ and $\{f, g, h, i, j\}$, calculated according to IFR formula, are 'coloured' grey; all other tiles except source S and target T are weighted by costs. To find a routing path from source S to target T we first find the min-cost paths from source S to tiles a, b, c, d, and e, which are, respectively, 10, 10, 11, 14, 15, as shown in Fig. 2*b*, and store them. Then we find the min-cost paths from source S to tiles f, g, h, i, and j, based on the previously stored data as done in step 2 of Fig. 1. The newly generated min-cost paths from source S to tiles f, g, h, i, and j are, respectively, 15, 15, 21, 21, 21, which are stored as shown in Fig. 2*b*. Finally, the min-cost paths from target T to tiles f, g, h, i, and j are found, which are, respectively, 15, 15, 18, 18, 18, as shown in Fig. 2*c*, and stored. Combining the data stored at previous steps, the min-cost from S to T is 30, as shown in Fig. 2*d*, the buffer site a or buffer site b is chosen from buffer set $\{a, b, c, d, e\}$, the buffer site f or g is chosen from buffer set $\{f, g, h, i, j\}$. Therefore the routing path is optionally S $\rightarrow a \rightarrow f \rightarrow$ T, S $\rightarrow a \rightarrow g \rightarrow$ T, S $\rightarrow b \rightarrow f \rightarrow$ T, or S $\rightarrow b \rightarrow g \rightarrow$ T, as shown in Fig. 2*d*.

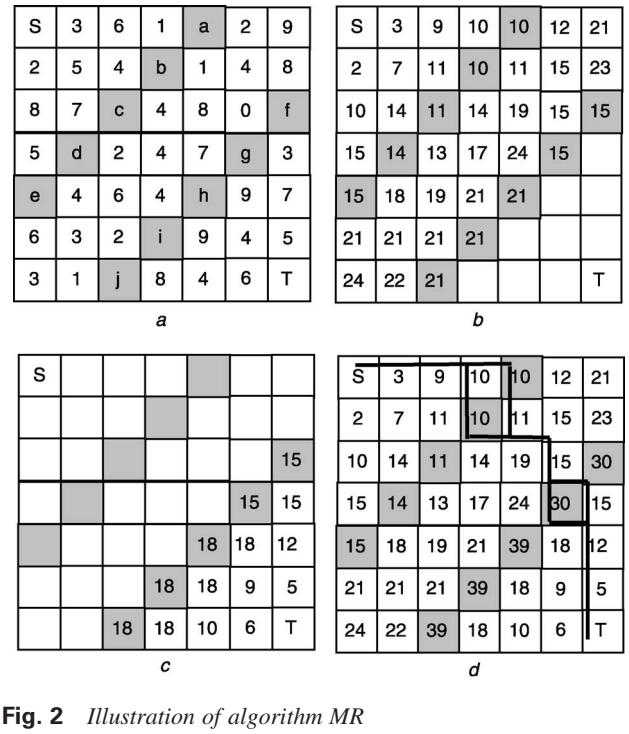

Fig. 2 *Illustration of algorithm MR*

4.2 Analysis of MR

Consider a two-pin net with end-points S and T and assume its Manhattan box has $l \times w$ tiles; if n buffers have to be added on this net, the space between any two neighbouring buffer sets $s = (l + w - 1)/(n + 1)$. The time complexity of our algorithm is described as follows.

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

Proof: The space between any two neighbouring buffer sets $= (l + w - 1)/(n + 1)$, on a tile graph, each element in one buffer set has at most $(l + w - 1)/(n + 1)$ connections with adjacent buffer sets, i.e. each element in one buffer set has at most $O(s)$ connections with adjacent buffer sets. Since each buffer set has $O(l + w)$ elements, the time complexity of finding the shortest path of an adjacent buffer site can be shown to be $O(s^2)$ and there are n buffer sets for each of which we have to select one buffer site for buffer insertion, the total time complexity becomes $O(s^3 n \times (l + w))$. Since s is treated as a constant, and n (the total number of

Given: source S and target T; a buffer set $B = \{B_1, B_2, \ldots, B_n\}$.

Goal: Choose a min-cost path from S to T, going through a subset of B, and which 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, store the cost and delay values.
2. While (not all buffer sites are chosen as a new source) do
   2.1. New source ← pick up an unchosen buffer site
   2.2. Use MR to route paths from the new source to all unchosen buffer sites and target T, update the cost and delay values.
3. Backtracking from T to S.

Fig. 3 *Algorithm MBR*

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

Not necessarily all nets can be routed successfully using MR. When a large block is located between the source and target of a net it is unavoidable to route a detour path. Therefore we proposed a Maze-based between-buffer router (MBR) to route a detour path between buffers, as shown in Fig. 3. Generally speaking, this method can find a path satisfying a delay budget if the path exists, but it needs a longer computation time in contrast to MR. Thus this method is designed mainly for postprocessing those nets that have been failed during MR.

Before using MBR we need to know the locations of buffer sites. The distance of a routing path during MBR is dependent on a routing path that is unknown before routing, while the distance of a routing path during MR is the Manhattan distance from source to tile. To decide the distances of all possible routing paths, the labelling technique of Lee's maze router [9] is adopted. After the maze-based 'wave propagation' procedure is performed, the filled labels together with IFR formula are used to derive the available locations of buffers.

In Fig. 3, at step 1, we try to find a path from S to all possible buffer sites using MR, check the delay budgets, store the costs and delay values of qualified buffer sites. At step 2, from buffer sites with delay values qualified at the previous step, one buffer site is chosen as a new source. From this new source, we use MR to route path to all unchosen buffer sites and target T, check accumulative delay values of these paths, store the costs and delay values of qualified buffer sites. Repeat this step until all buffer sites have been chosen as a source. Finally, at step 3, by treating target T as the current buffer site, starting from the target and backtracking to the buffer site next to the source, we find the preceding buffer site where the min-cost path has passed through according to the accumulative costs of all paths reaching the current buffer site.

Take Fig. 4*a* as an example, in which the blocked tiles caused by congested wires are black and the buffer locations are grey. To find a min-cost detour path from source S to target T, a min-cost table like that in Fig. 4*b* is built according to algorithm MBR, in which min-costs between S, T and buffers are listed, entries are marked × when the delay budget is violated or routing is failed. First, the entries at column S are obtained according to step 1 of Fig. 3, the min-costs from S to a, b, c, and d are, respectively, 9, 9, 6, 8, and the entries from S to e and T are both marked × because of blocked routing paths. Then the min-cost from a to b, c, d, e, T are obtained, which are, respectively, 3, 3, ×, ×, and 8. Accumulating theses values to the min-cost from S to a, we have the entries of column a, which are 12, 12, ×, ×, 17, respectively. In an iterative manner, as shown in step 2 and step 3 of Fig. 3, we have the other entries for the remaining columns. Comparing entries at row T, the min-cost entry 12 is located at column d. Inspecting entries at row d, the min-cost entry 8 is located at column S. The path from S to T is thus obtained from the backtracking procedure and is found to be $S \rightarrow d \rightarrow T$ with the min-cost $= 8 + 4 = 12$.

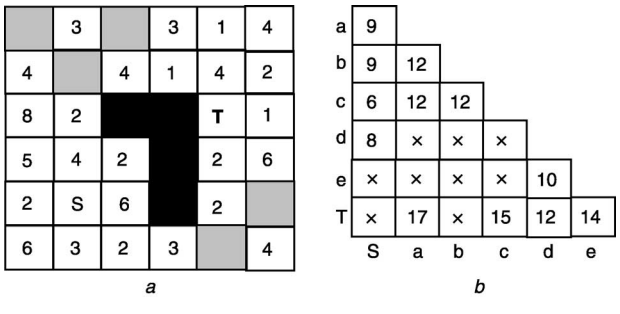

Fig. 4 *Illustration of algorithm MBR*

4.4 Procedure of floorplanning

It is easy to integrate MR and MBR into an iterative floorplanning algorithm. The procedure to integrate

```
Procedure floorplanning
floorplan = random_floorplanning();  T = initial_temperature;
while (T >= threshold) {
  count = 0;
while (not_equilibrium(count, floorplan, T)) {
 mr = mbr = FAIL;  new_floorplan = perturb(floorplan);
     ΔC = cost(new_floorplan) – cost(floorplan);
     if (MR(new_floorplan) == OK)  mr = OK;
     if (mr == FAIL) and (ΔC < 0))
        if (MBR(new_floorplan) == OK)  mbr = OK;
     if ((mr == OK) or (mbr == OK))
        if (ΔC < 0) floorplan = new_floorplan;
        else { prob = Min(1, e^(-ΔC/T)); if (random(0,1) <= prob)
               floorplan = new_floorplan; }
     count = count + 1; }
update(T); }
```

Fig. 5 *Integrating MR and MBR into SA-based floorplanning algorithm*

algorithms of MR and MBR into a SA-based floorplanning algorithm is shown in Fig. 5 as an example.

In Fig. 5, mr and mbr are two flags separately indicating the results of MR and MBR. Perturb( ) is a perturbation function which is used to generate the next floorplan from the current floorplan. Not_equilibrium( ) is a function that decides the termination condition at a given temperature, and update( ) is used to cool down the temperature. After each perturbation, cost( ) evaluates the area change between old floorplan and new floorplan, the cost change is expressed as ΔC. For each newly generated floorplan, MR is performed to qualify its wire congestion and buffer congestion. As MBR takes longer computation time than MR, MBR is performed only when MR fails and the newly generated floorplan has smaller area $(\Delta C < 0)$.

5 Experimental results and discussion

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

5.1 Routing with different MR and MBR schemes

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 is to route a single net with MR. If it succeeds, continue with the next net; otherwise route the net with MBR.

Table 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 and 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.

Table 1: Comparison of failed nets in different schemes

Circuit	Net	1st scheme 1 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

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 sorted-by-delay net ordering and with four other random net orderings. Scheme 1 was chosen as our routing procedure. The results are summarised in Table 2, where the results of *average*, *max*, *min* and *numerous* are collected after 30 different random net ordering experiments.

Table 2 showed the superiority of our net ordering. The results of our proposed sorted-by-delay net ordering are the best in each test-bench circuit compared with the other net ordering methods.

5.3 Comparison with buffer-site algorithm

Based on the same test benches and technology parameters, we compare our method with Alpert *et al.*'s buffer-site approach [7]. First, we implemented a kernel of the 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. The following reasons of failure are observed: fail to route a net, fail to place buffers within *Li*, and mostly fail to fit the delay value to our delay budget.

We use the test-bench circuits as listed in Table 1 to compare the buffer-site algorithm with ours, as shown in Table 3, in which $Li = 11$. It is surprisingly that the buffer-site algorithm has a higher rate of failed nets. Using our approach the failed net percentage decreased by 34.2 to 55.6%. This is because we tried to customise the buffer-insertion scheme for each net, while their algorithm uses the same *Li* for every net. In VLSI design, especially in system-on-a-chip design, different nets have a 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.

Table 2: Comparison of failed nets in 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

Table 3: Comparison 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 algorithm	fail net	569	790	571	692	521
	fail percentage	60.2	70.4	69.4	62.1	60.7
Our algorithm	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

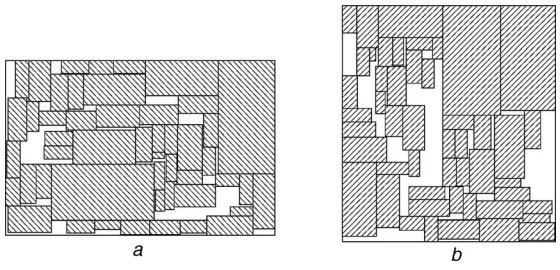

Fig. 6 *Result of integrating MR and MBR into SA-based algorithm*

a Floorplan generated by SA + MR + MBR $H = 4.97$; $W = 8.092$; dead space = 11.87%
b Floorplan generated by SA + MR $H = 6.958$; $W = 6.286$; dead space = 18.96%

5.4 Integration of MR/MBR and SA-based floorplanning algorithm

We integrated both MR and MBR into a SA-based floorplanning algorithm and used MCNC circuit, ami49, as a test bench. It is assumed that each block can supply some buffers, and the amount of buffers available in each block is dependent on its area. For each floorplan, the necessary buffers can be either inserted at dead space or supplied by blocks. When the floorplan is divided into 10 × 10 tiles, NUB and BUB are separately 45 and 20, the resultant floorplan is shown in Fig. 6*a*. As a contrast, the resultant floorplan generated by a SA-based floorplanner integrating only MR is shown in Fig. 6*b*. Obviously the ability of detour possessed by MBR facilitates obtaining more compact floorplan.

6 Conclusions

We have proposed a routing-buffering tool to perform global routing and buffer insertion simultaneously for immediate evaluating routability of a floorplan design. The algorithm includes a Manhattan routing algorithm and a maze-based between-buffer routing algorithm. Our method

- considers net congestion constraint and interconnect delay together, not easily done by Cong *et al.*'s and Sarkar *et al.*'s algorithms
- for two pin nets, instead of buffering at the shortest path as proposed by Alpert *et al.* [7], we insert buffer into a path such that the path is min-cost in terms of net congestion and buffer congestion while the delay constraint is not violated.

7 Acknowledgments

This work was supported by the National Science Council, R.O.C., under grant NSC91-2215-E002-042. The authors would like to thank Prof. Jason Cong for providing the source code for buffer-block planning [3] and benchmarks.

8 References

1 Lou, J., Thakur, S., Krishnamoorthy, S., and Sheng, H.S.: 'Estimating routing congestion using probabilistic analysis', *IEEE Trans. Comput.-Aided Des. Integr. Circuit Syst.*, 2002, **21**, (1), pp. 32–41
2 Sham, C.W., Wong, W.C., and Young, E.F.Y.: 'Congestion estimation with buffer planning in floorplan design'. Proc. Conf. on Design and Test in Europe, Paris, France, March 2002, pp. 696–701
3 Cong, J., Kong, T., and Pan, D.Z.: 'Buffer block planning for interconnect-driven floorplanning'. Proc. Int. Conf. on Computer-Aided Design (ICCAD), San Jose, CA, November 1999, pp. 358–363
4 Sarkar, P., Sandararamanm, V., and Koh, C.-K.: 'Routability-driven repeater block planning for interconnect-centric floorplanning'. Proc. Int. Symp. on Physical Design, San Diego, CA, April 2000, pp. 186–191
5 Dragan, F.F., Kahng, A.B., Mandoiu, I.I., Muddu, S., and Zelikovsky, A.: 'Provably good global buffering using an available buffer block plan'. Proc. Int. Conf. on Computer-Aided Design (ICCAD), San Jose, CA, November 2000, pp. 104–109
6 Dragan, F.F., Kahng, A.B., Mandoiu, I.I., Muddu, S., and Zelikovsky, A.: 'Provably good global buffering by multiterminal multicommodity flow approximation'. Proc. Conf. on Asia South Pacific Design Automation (ASP DAC), Yokohama, Japan, January–February 2001, pp. 120–125
7 Alpert, C., Hu, J., Sapatnekar, S., and Villarrubia, P.: 'A practical methodology for early buffer and wire resource allocation'. Proc. Design Automation Conf. (DAC), Las Vegas, NV, June 2001, pp. 189–194
8 'National technology roadmap for semiconductors', Semiconductor Industry Association, 1997
9 Lee, C.Y.: 'An algorithm for path connection and its application', *IRE Trans. Electro. Comput.*, 1961, **20**, pp. 346–365