An Integrated Floorplanning with an Efficient Buffer Planning Algorithm

Yuchun Ma[1], Xianlong Hong[1], Sheqin Dong[1], Song Chen[1], Yici Cai[1], C.K.Cheng[2], Jun Gu[3]

[1] Department of Computer Science & Technology, Tsinghua University, Beijing, China,100088
clara99@mails.tsinghua.edu.cn; hxl-dcs@tsinghua.edu.cn

[2] Department of Computer Science and Engineering, University of California,San Diego CA 92093-0114,USA

[3] Department of Computer Science, Science & Technology University of HongKong

ABSTRACT

Previous works on buffer planning are mainly based on fixed die placement. It is necessary to reduce the complexity of computing the feasible buffer insertion sites to integrate the buffer planning with the floorplanning process. In this paper, we give an efficient buffer planning algorithm with linear complexity by computing all the feasible buffer insertion sites in a 2-step method. By partitioning all the dead spaces into blocks while doing the packing, the buffer allocation can be handled as an integral part in the floorplanning process. Our method is based on a simulated annealing approach which is divided into two phases: timing optimization phase and buffer insertion phase. Since there is more freedom for floorplan optimization, the floorplanning algorithm integrated with buffer planning can result in better time performance and chip area.

Categories and Subject Descriptors

B.7.2 [**Integrated Circuits**]: Design Aids – *Placement and Routing*

General Terms

Algorithms, Performance, Design.

Keywords

Floorplanning, Routability, Buffer Insertion.

1. INTRODUCTION

Due to the recent advances in VLSI technology, the number of transistors in a design is increasing rapidly and so are their switching speeds. This has increased the importance of the interconnect delay in the overall performance of a circuit. Many techniques are employed to reduce interconnect delay. Among them, buffer insertion has shown to be an effective approach to achieve timing closure. As transistor count and chip dimension get larger and larger, more and more buffers are expected to be needed for high performance. It was projected that over 700K buffers will be inserted on a single chip in the 70nm technology[8]. Since buffers are implemented by transistors, they cannot be placed over the existing circuit blocks. Placing a large number of buffers between circuit blocks could significantly impact the chip floorplan. Therefore, it is necessary to start buffer planning as early as possible. It is very useful that a good planning of the block positions can be obtained during the floorplanning stage so that buffers can be inserted wherever needed in the later routing stages There are several previous works addressing the interconnect issues in floorplanning design. Cong[1] define the term "feasible region" (FR) of a net, that is, the largest polygon in which a buffer can be inserted such that the timing constraint can be satisfied. Sarkar[3] added the notion of independence into feasible regions so that the feasible regions of different buffers on a net can be computed independently. These two papers give the basic idea of Feasible Region, based on which they proposed the buffer planning algorithms. But both of their methods take complex scanning to obtain the feasible buffer insertion sites. Tang and Wong[9] propose an optimal algorithm based on net flow to assign buffers to buffer blocks assuming that only one buffer is needed per net. Alpert et al.[11] make use of tile graph and dynamic programming to perform buffer block planning, while they propose that buffers should be allowed to be inserted inside macro blocks. But all of these algorithms are based on fixed die placement and it is difficult to embed those methods into the iterations of the floorplanning process because of the complexity of those algorithms. Unfortunately, the fixed placement is likely to generate some timing-constraint violations which are beyond repair unless the topological relation between blocks can be changed. Hence, to create a performance-feasible floorplan, a floorplanner that simultaneously consider area and buffer insertions is needed.

The feasible region for buffer insertion in the packing reduced by the circuits is a very complex polygon, normally concave polygon. The computation of buffer insertion sites is very difficult and time-consuming. Though the method in [9] is independent of grid size, it is still too complex of that method that it scans the packing by a sweep line and its algorithm runs in O(n^2logn) time where n is the number of the nets. Taking advantages of the dead

*This work is supported by the National Natural Science Foundation of China 60121120706 and National Natural Science Foundation of USA CCR-0096383, the National Foundation Research(973) Program of China G1998030403, the National Natural Science Foundation of China 60076016 and 863 Hi-Tech Research & Development Program of China 2002AA1Z1460

spaces blocks in the packing and the feature of the FR, we can figure out the possible insertion sites for the buffers very efficiently. Instead of computing the size of the dead space in each grid in the packing, we compute the intersection between the dead space blocks and the FRs in a 2-step method. Since our method can give the range of the possible buffer insertion sites independent of the sizes of the grids, we give a probabilistic method to budget the buffer insertion. The computation of the possible buffer insertion sites for all net can be executed in the running time of O(n) where n is the number of the nets, since we need to scan all the net only once.

Since buffers should utilize the dead space in the packing, it is necessary to obtain the information about the dead spaces very effectively. In this paper, we propose the algorithm to obtain the dead space blocks in the floorplanning while doing the packing. Based on the CBL packing algorithm, we can partition the dead spaces in the packing into no more than 2n blocks (n is the number of the circuit blocks) without overlapping between each dead space blocks. To speed up our algorithm, the simulated annealing process is divided into two phases: timing optimization phase and buffer insertion phase. The experiments prove the effectiveness of our approach.

The rest of the paper is composed as follows: Sect.2 gives the problem formulation; Sect.3 gives the computation method of the buffer insertion sites. The integrated floorplanning algorithm is described in Sect.4. The experimental results are shown in Sec.5. Finally, the conclusion is given. .

2. PROBLEM FORMULATION

In this paper, we want to involve the buffer planning issue into the floorplanning design: given the timing constraints on each net, we should give a floorplanning with buffers inserted to meet the timing constraints. Also we should find the number and locations of buffers at the same time. We consider the insertion of buffers should in the dead spaces between circuit blocks. The buffer allocation is handled as an integral part in the floorplanning process. We seek a floorplanning methodology to produce the optimal floorplan such that the floorplan area and wire length are minimized and the buffers can be inserted in the dead spaces as much as possible. Since there is more freedom for floorplan optimization, the integrated floorplanning algorithm with buffer planning can result in better time performance.

3. COMPUTATION OF POSSIBLE BUFFER INSERTION SITES

3.1 Independent Feasible Region

Cong et.al [1] introduced the concept of Feasible Region(FR) for buffer insertion, and presented analytical formula to compute FR. The feasible region for a buffer '*b*' is the maximum region where '*b*' can be located such that by inserting buffer '*b*' into any location in that region, the delay constraint can be satisfied. Sarkar et al.[3] gives the notion of independent feasible regions(IFR) and the IFRs of buffers belonging to the same net do not overlap each other. Here we use the concept of IFR so that the feasible regions of different buffers on a net can be computed independently. Each driver/buffer is modeled as a switch-level *RC* circuit [5] and the Elmore delay formula [6] is used for delay computations. The notation for the physical parameters of the interconnect and buffer we use in this paper is as follows:

r	wire resistance per unit length
c	wire capacitance pre unit length
T_b	intrinsic buffer delay
C_b	buffer input capacitance
R_b	buffer output resistance

The optimal locations of the *k* buffers for delay minimization of the net as shown in [2] are

$$x_i^* = (i-1)y_L^* + x_L^* \qquad i \in \{1,2,...k\} \tag{1}$$

Where

$$x_L^* = \frac{1}{k+1}\left(L + \frac{k(R_b - R_d)}{r} + \frac{(C_L - C_b)}{c}\right)$$

$$y_L^* = \frac{1}{k+1}\left(L - \frac{(R_b - R_d)}{r} + \frac{(C_L - C_b)}{c}\right)$$

In order to satisfy the target delay constraint, denoted by T_{tgt}^N, and $T_{opt}^N(k,l)$ is the optimal delay with k buffers inserted , for $T_{tgt}^N \geq T_{opt}^N(k,l)$, the width of the IFR for the ith buffer (i<k) of the net is

$$W_{IFR} = 2 \times \sqrt{\frac{T_{tgt}^N - T_{opt}^N(k,l)}{rc(2k-1)}} \tag{2}$$

Here we define:

$$x_{\min} = x_i^* - W_{IFR}/2\,;$$

$$x_{\max} = x_i^* + W_{IFR}/2$$

Formally, the IFR for the *i*th buffer of a net is

$$IFR_i = (x_{\min}, x_{\max}) \cap (0, L)$$

In [1], it gives the minimum number of buffers to meet the delay constraint T_{req} for an interconnect of length *l* is

$$k_{\min} = \left\lceil \frac{-K_5 - \sqrt{K_5^2 - 4K_4K_6}}{2K_4} \right\rceil \tag{3}$$

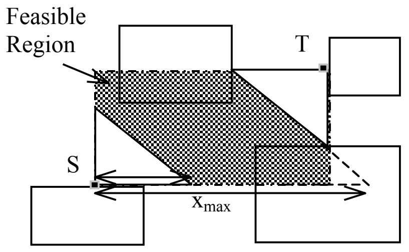
where

$$K_4 = R_bC_b + T_b$$

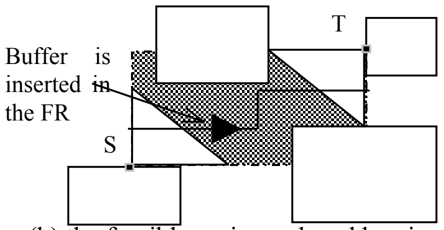
$$K_5 = (rC_b + cR_b)l + T_b + R_dC_b + R_bC_L - T_{req} - \frac{r}{2c}(C_b - C_L)2 - \frac{c}{2r}(R_b - R_d)^2$$

$$K_6 = \frac{1}{2}rcl^2 + (rC_L + cR_d)l + R_dC_L - T_{req}$$

The 2-D IFR of a buffer is defined as the union of the 1-D IFRs of that buffer on all monotonic Manhattan routes between source and sink. Therefore, 2-D IFRs, as in [2], are convex rectilinear polygons with horizontal, vertical, and ±1 slope boundaries (Fig. 1(a)). For a net with n buffers, the 2-D feasible region for the ith buffer is the region bounded by two parallel lines with Manhattan distance from the source in the range of IFRi, and by the rectangular bounding box between the source and sink. The slope of the two parallel lines is either +1 or –1, depending on the sign



(a) the 2-D feasible region



(b) the feasible region reduced by circuit blocks

Figure 1. The 2-D independent feasible region

of $K = (y_{\sin k} - y_{src})/(x_{\sin k} - x_{src})$: if K > 0, the slope is –1; if K<0, the slope is +1.Two-step method to computer the insertion sites.

3.2 Two-step method to computer the insertion sites

Since the feasible region will be reduced by the circuit blocks, the feasible region for buffer insertion in the packing is a very complex polygon, normally concave polygon, as shown in Fig.1(b). We partition the dead space into rectangular blocks to capture the constraint that the circuit blocks prevent the insertion of a buffer. Clearly, a buffer must be inserted within the intersection of its feasible region and a partitioned dead space rectangle.

Definition 1: Dead Space Block(DS block): The dead spaces between the circuit blocks can be partitioned into rectangles which are called dead space blocks and the buffer blocks should be packed within the range of dead-space blocks.

The intersection between FR of a buffer and the dead space block is a regular hexagon(including some degenerations), whose two edges parallel to the x-axis, two edges parallel to the y-axis, and other two edges have the slope of +1(45 degree) or –1(135 degree).

To facilitate the data manipulation, we partition the dead space blocks into grids and each grid provides sites for buffer insertion. Therefore, the buffers should be inserted at the grids within their FRs to meet the delay constraints. The computation of buffer insertion sites is the most difficult and time-consuming part when doing the buffer planning. A naïve solution is to examining all the grid points in the packing, thus this method has its running-time depending on the grid size and needs a lot of scanning processes.

Instead of the complex computation of the intersection between the hexagon of FR and the dead space blocks, we decompose the problem into two simple problems: the first step is to compute the intersected blocks between dead space blocks and the bounding box of the source and sink; the second step is to compute the overlapping between the result blocks in the first step and the region between two parallel lines which are the two edges of the FR. Since these two lines have the slope of +1 or –1, we order the grids in dead space blocks parallel to these two lines. Therefore by taking advantage of the information of dead space blocks, we can figure out the intersections between the feasible regions and the dead spaces for all nets very efficiently.

Step 1. Compute the intersection(intersected DS) between dead space blocks and the source-sink bounding box

Since both the dead space block and the source-sink bounding box are rectangular, the intersection between them is also rectangle. It is a basic problem on geometry to figure out the intersection between two rectangles. We solve it by the extension on the intersection of the segments.

Lemma 1: suppose that the lower left corner of the bounding box between source and the sink is (x_{ll},y_{ll}), the upper right corner is (x_{ur},y_{ur})(Similar definition in the following section). The lower left corner of the dead space block is (x_{llds},y_{llds}) and the upper right corner is (x_{urds},y_{urds}). The two rectangles intersect with each other if the following inequalities are satisfied(Fig.7):

$$\begin{cases} Xur - Xll > 0 \\ Yur - Yll > 0 \end{cases}$$

Where :

$X11 = max(x_{11},x_{11ds})$; *$Y11 = max(y_{11},y_{11ds})$*

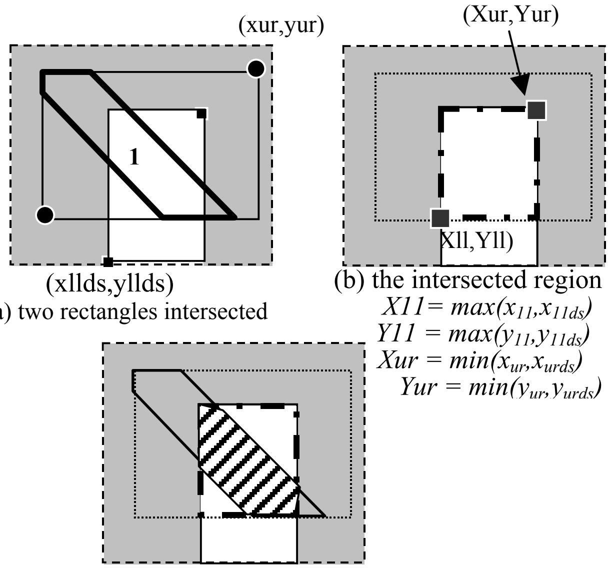
$Xur = min(x_{ur},x_{urds})$; *$Yur = min(y_{ur},y_{urds})$*

And the lower left corner of the intersection rectangle should be (Xll,Yll) ; the upper right corner is (Xur,Yur).

As shown in Fig2.(a), the dead space block 1 and the source-sink bounding box overlap each other. Following lemma 1, we can figure out the intersected rectangle (Intersected DS) as in Fig.2(b).

Step 2. Compute the intersection between intersected DS and the region between two parallel lines(Fig.2(c)).

Since the dead space blocks are divided into grids, to compute the



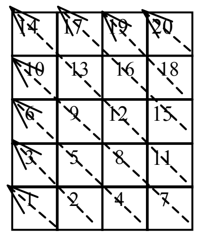
(a) two rectangles intersected

(b) the intersected region

$X11 = max(x_{11},x_{11ds})$
$Y11 = max(y_{11},y_{11ds})$
$Xur = min(x_{ur},x_{urds})$
$Yur = min(y_{ur},y_{urds})$

(c) the shadowed region is the intersection between FR and DS_block

Figure 2. the computation of the FR in the dead space block



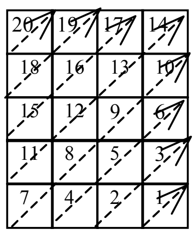


Figure 3. The Grid Ordering

intersection between two parallel lines and intersected DS, we order the grids by {G1,…Gmax} in the intersected DS by the sequence of the slope +1 and –1(Fig.3). Here we only consider the feasible edge with the slope of -1, since the other case can be handled similarly. As in Fig.3(a), we order the intersected DS from 1 to 20 by the sequence of the slope –1.

If the feasible region between the two parallel lines intersect with the intersected DS, we define the left line in the feasible region is the first line and the right line is the second line. Suppose that the first line meet the intersected DS at the bottom boundary or the right boundary at the grid G_P(if the first line does not have the intersection with the intersected DS, then $G_P = G_1$) ; the second line meet the intersected DS at the top boundary or the left boundary at the grid G_Q(if the second line does not have the intersection with the intersected DS, then G_Q =Gmax). Then the feasible buffer insertion sites should between G_P and G_Q.

In fig.4 we can see that the first line meets the bottom boundary at grid 4 and the second line meets the top boundary at grid 17. Thus, the feasible buffer insertion sites should be between the grid 4 and grid 17 in the intersected DS.

Lemma 2: The number of possible insertion sites of the ith FR (NFRi) intersected with dead space should be

$NFR_i = \{\Sigma(|\ G_P - G_Q\ |)\ |$ *for all the dead space blocks intersected with FRi* }

If $NFR_i = 0$, then the buffer will not be placed properly since there is no available insertion sites.

We assume that the probabilities of buffer insertion are equal at each possible insertion site. Suppose that grid G is a possible insertion site for ith buffer in net N. Thus the probability of *i*th buffer insertion at grid G is $P^G_{Ni} = 1/NFR_i$

The total buffer insertion probability of grid G is

P^G = {sum(P^G_{Ni}) | for each feasible region which has intersection with grid G}

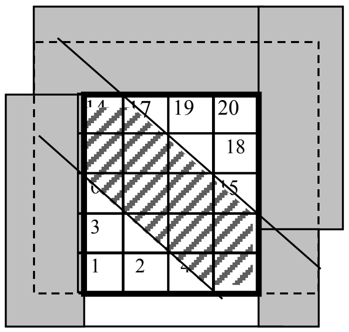
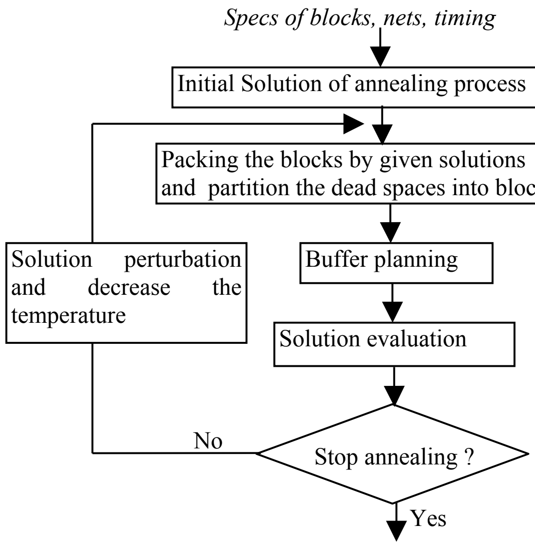


Figure 4. Possible buffer insertion sites



Specs of blocks, nets, timing
Initial Solution of annealing process
Packing the blocks by given solutions and partition the dead spaces into blocks
Buffer planning
Solution evaluation
Solution perturbation and decrease the temperature
No
Stop annealing ?
Yes

Figure 5. Overall algorithm

4. FLOORPLANNER WITH BUFFER PLANNING

Since our method can give the range of the possible buffer insertion sites independent of the sizes of the grids, we can budget the buffer insertion during the packing. To embed the buffer planning in the packing process, the overall algorithm is given as Fig.5.

4.1 CBL Representation

To do the block packing, we use the corner block list representation. CBL is derived from a simplified version of general placement called mosaic structure, in which the block is represented by the room with only topological relationship between each other. CBL represents the topological relations in mosaic structure by a triple list of (S, L, T). It divides the chip into rectangular rooms and assigns one and only one block to each room according to (S, L, T). S records the sequence of the blocks' ID, L records the orientation for each block, while T record how many blocks are covered by each block when it is packed.

The Corner Block(CB) is the block packed at the upper right corner of the floorplan. If the corner block covers other blocks from top, we define the CB to be vertical oriented, and denote it by Li = 0. Otherwise, the CB is horizontal oriented, and denote it by Li=1. The binary sub-list in T records the number of the blocks covered by this CB. The number of successive "1"s, which is ended by a "0", corresponds to the number of the blocks covered by the corner block. As in Fig.6, the corner block is block "d" and $L_d=0, T_d=\{10\}$, thus, block "d" covers two blocks "g" and "a" from top. For more detail about CBL, please infer Reference[4].

4.2 The Dead Space in CBL

To insert the buffers, we should give the partition of the dead spaces in advance. Though we can partition the dead space into blocks by many different methods, we should devise an effective method following the rules below:

(1) no overlapping between each dead space blocks;

(2) the number of the dead space blocks should be as few as possible;

Corner Block List can represent mosaic structure, which includes *n* rooms with no more than one block in each room, where *n* is the number of blocks. The dead spaces besides the blocks in their rooms compose the dead spaces in the final packing. Given a CBL list, we expand the packing from the lower left to upper right by inserting the blocks in list *S* in turn. Every insertion of the block may generate some more dead space. Thus we can figure out all the dead space while doing the packing and divide the dead spaces into rectangles.

Lemma 3: during the packing process, when the *i*th block *Mi* is packed, according to the given CBL, it will cover N_{Mi} blocks which are $\{BC_1...BC_{NMi}\}$

- If block Mi covers other blocks from top (Li = 0) then the heights of rooms containing blocks $\{BC_1...BC_{NMi}\}$ are settled.
- If block Mi covers other blocks from right(Li=1) then the widths of rooms containing $\{BC_1...BC_{NMi}\}$ are settled.

As shown in Fig.6, when block 'd' covers two blocks from the top, the height of the rooms containing blocks *g* and *a* respectively are settled.

To move the blocks in their rooms will not affect the topological relations between blocks and the final packing area. Since our partition method for dead space is based on the range of the blocks' rooms, the dead space beside the block in the room can be settled according to the strategy of how to place the blocks into their rooms. Especially, some channels are left to favor the performance of the packing in some floorplanner. In our algorithm, the circuit blocks are packed at the lower-left corner of the rooms. Therefore, we can partition the dead space in one room into no more than two dead space blocks, one is a dead space block above (DS_{above}) and the other is a dead space block at the right (DS_{right}). As shown in Fig.7(a), dead space blocks *2* and *1* are the dead space above and right to block "a" respectively(Some channels are left as the minimal distance constraints between blocks).

Thus, during the packing process, when the *i*th block *Mi* is packed, according to the given CBL, it will cover N_{Mi} blocks$\{BC_1...BC_{NMi}\}$ from top(if Li = 0) or from right (Li=1). For block M_i, we use the position of its bottom left corner (X_i,Y_i) to define its placement, the height and width for block M_i are *Wi* and *Hi* respectively. Similarly, the bottom left corner of block $BC_k(k\in[1,N_{Mi}])$ is (x^{bc}_k, y^{bc}_k) and the height and width of block BC_k are w^{bc}_k and h^{bc}_k respectively: the width and height of the dead space above block BC_k is WDS^k_a and HDS^k_a respectively, the width and height of the dead space at the right of block BC_k is WDS^k_r and HDS^k_r respectively.

The heights and the widths of the dead space blocks beside each

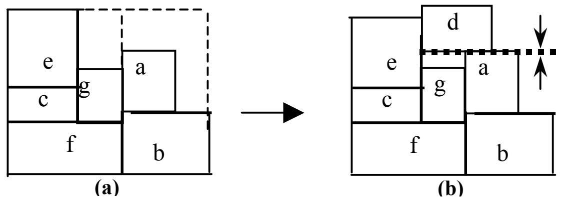


Figure 6. The CB is block 'd', L_d=0,T_d={10} and it covers 2 blocks

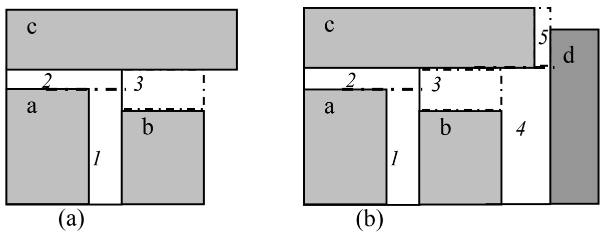


Figure 7. The example packing process

block are initialized to zero before packing,

$$HDS^k_a = WDS^k_a = HDS^k_r = WDS^k_r = 0$$

- If Li = 0 and $y^{bc}_k + h^{bc}_k < Y_i$, there should be a dead space block above block BC_k

$$HDS^k_a = Y_i - (y^{bc}_k + h^{bc}_k) \quad (4)$$

$$WDS^k_a = w^{BC}_k + WDS^k_r \quad (5)$$

- If Li = 1 and $x^{bc}_k + w^{bc}_k < X_i$, there should be a dead space block at the right of block BC_k

$$WDS^k_r = X_i - (x^{bc}_k + w^{bc}_k) \quad (6)$$

$$HDS^k_r = h^{BC}_k + HDS^k_a \quad (7)$$

Fig. 7 is the demonstration of the packing process with the dead spaces partition. In Fig.7(a), block *c* covers blocks *b* and *a* from the top, and dead space blocks 2 and 3 are generated according to formula (4) and (5). In Fig7(b), block d covers blocks *c* and *b* from the right, thus, dead space blocks 5 and 4 are generated according to formula (6) and (7).

Based on the partition strategies above, we can obtain the shapes and positions of all the dead space blocks in the floorplan while doing the packing process as described in Algorithm 1:

The dead space blocks should be non-overlapping between each other, otherwise there should be some error while computing the possible buffer insertion sites.

Algorithm 1 *Packing blocks with DS blocks partition*

```
  Initialize the heights and widths of DS blocks
  For block Mi is from M1 to Mn:
    Pack block Mi according to the given CBL;
    Figure out the DS blocks in the rooms covered by Mi;
  End for
  Figure out the DS blocks in the rooms covered by the top boundary;
  Figure out the DS blocks in the rooms covered by the right boundary;
  End.
```

Lemma 4: The partition method of algorithm 1 will not generate overlapping between dead space blocks.

4.3 Two-phase annealing process

In our algorithm, the simulated annealing process is divided into two phases: timing optimization phase and buffer insertion phase. In the timing optimization phase, we try to search for an optimal floorplanning that the timing constraints can be satisfied as much as possible.

In the beginning of floorplanning process, the buffer planning is less meaningful because the locations of the blocks are still far from their final position. But we should evaluate the floorplanning to favor the buffer planning. Here we use a probabilistic method in Sec3.2 to estimate the buffer insertion.

Suppose that grid G is a possible insertion site for ith buffer in net N. Thus the probability of *i*th buffer insertion at grid G is

$$P^{G}_{Ni} = 1/NFR_i$$

The total buffer insertion probability of grid G is

P^{G} = {*sum*(P^{G}_{Ni}) | *for each feasible region which has intersection with grid G* }

Suppose the area of the grid is A_{grid} and the area of a buffer is A_{buffer}, R = A_{grid}/A_{buffer}.

If P^{G}> R then we think the buffers inserted will be too crowded thus we should take some measure to control it.

And if NFRi = 0, then the buffer will not be inserted properly since there is no available grids for this buffer. Thus we use *B_evaluate* to budge the violations of buffer insertion:

B_evaluate = Count{(*NFRi = 0*)| *for all nets*} + *Count*{(P^{G}>*R*)|*for all grids*}

The cost function used in the phase is shown below:

*Cost = Area + p*Wire + q* Tviolations +s*B_evaluate*

Where Area is the area of the floorplan and wire is the total wirelength(p is the weight), Tviolation is the number of the net whose optimal timing with buffer inserted is larger than the given timing constraint.

Since we can obtain the feasible insertion sites for buffer very effectively, we can extend the method to insert buffers easily. In the buffer insertion phase, the buffer assignment is done by the heuristic methods similar to the method in [1] to optimize the buffer insertion. But here we do not enlarge the dead spaces when the buffers can not be placed in the dead spaces properly because of the limitation of the dead spaces. These violations will be reduced during the annealing process. The cost function used in the phase is shown below:

*Cost = Area + p*Wire + q* Tviolations + r*Bnot_inserted*

Where Area is the area of the floorplan and wire is the total wirelength(p is the weight), Tviolation is the number of the net whose optimal timing with buffer inserted is larger than the given timing constraint. Bnot_insert is the number of the buffers not inserted successfully because of the limitation of the dead spaces.

5. EXPERIMENTS

We have implemented the placement algorithm in C programming language, and all experiments are performed on a SUN SPARC III workstation. Some MCNC benchmarks are used in the experiments. The parameters(refer to Table 1) used in our experiments are based on a 0.18um technology in [10].

We have tested our algorithms on 5 MCNC benchmarks, as summarized in Table 2. In this paper, we focus on 2-pin net, so we decompose each multi-pin net into a set of source-sink 2-pin net. Because of the lack of information on signal direction in the benchmark files, we choose a pin to be the source and all the others to be sinks, and then decompose a multiple terminal net into a set of two-pin nets. We ignore all power and ground interconnects. Since the MCNC benchmarks do not come with any timing information, we generate a floorplan by running the CBL floorplanner randomly. Based on this floorplan, we assign target delays to the two-pin nets as follows: for each net, we first compute its best delay by optimal buffer insertion Topt, and assign its target delay as 1.1Topt. Notice that the sizes of the blocks are enlarged for demonstration of the effect of buffer planning.

In table 3, we report the experimental results of two floorplanners: a traditional floorplanner F1 based on simulated annealing without considering timing issue with the buffer insertion, and a timing-driven floorplanner F2 on 2-phase simulated annealing with buffer planning. In most layout tools, some channels are used between blocks to favor the buffer insertion and routing. Here we give the results with no channels and the results with channels between blocks (the channel width is settled as 2 times of the buffer width). For the result of F1, we also perform the heuristic method of buffer insertion as in F2 at the end to compare the results of F2: 1)#Meet: the number of nets for which the delay constraint is met with successful buffer insertion; 2) #Inserted B / #B: the ratio of the total number of buffers inserted successfully and the total number of the buffer needed to meet timing constraints, 3) #Violation: the number of the nets whose T_{opt}>T_{tgt} ;

Comparing F1 and F2 in table 3, the differences between F1 and F2 on area and wirelength are very small but the timing driven floorplanning algorithm with buffer planning (F2)can achieve much better timing performance than the plain floorplanning algorithm(F1). As we generate the timing on our own the test cases are different, a direct comparison between our method and [1][3] may not be fair. From the results we can see that for test case of Xerox with 455 2-pin nets, 370 nets meet their time constraints with 214 buffers inserted successfully, and the number of the nets whose T_{opt}>T_{tgt} is only 51. Therefore, there is only 34 nets violate the constraints because of the failure of buffer insertion. While the best result in [3] is that 368 nets meet their

Table 1. Parameter List

	Description	Value
R	Wire resistance per unit length(Ωμm)	0.075
C	Wire capacitance per unit length(fF/μm)	0.118
Tb	Intrinsic buffer delay(ps)	36.4
Cs/Cb	Sink/buffer capacitance(fF)	23.4
Rb/Rd	Driver/buffer output resistance(Ω)	180

Table 2. MCNC Benchmark

circuit	blocks	nets	2-pin net
apte	9	97	172
xerox	10	203	455
hp	11	83	226
Ami33	33	123	363
Ami49	49	408	545

Table 3. Comparison of floorplanning algorithm and the integrated floorplanning algorithm with buffer planning

	Area(mm^2)		Wire(mm)		#Inserted B/#B		#meet		#violation		Time(s)	
Test[1]	F1	F2	F1	F2	F1	F2	F1	F2	F1	F2	F1	F2
Xerox_1	84.57	85.41	1343	1327	189/395	214/303	193	370	124	51	12	59
Xerox_c	91.16	86.32	1424	1439	70/345	84/319	159	296	224	88	28	64
Ami33_1	30.86	31.15	431.5	461.2	117/524	192/465	101	170	124	39	26	206
Ami33_c	34.04	36.07	515.6	503.9	216/501	172/307	87	199	91	34	26	267
Ami49_1	156.67	146.6	2922	2920	315/582	244/568	234	341	227	151	55	329
Ami49_c	175.27	183.15	3471	2940	198/511	363/546	198	344	181	154	64	326
Apte_1	48.15	48.14	484.3	459.5	11/111	44/107	80	113	80	49	6.06	27
Apte_c	49.55	50.08	520.2	478.8	21/154	53/89	88	112	80	53	6.1	25
Hp_1	38.61	38.86	424.4	392.2	23/350	40/106	84	163	99	43	3.74	29
Hp_c	40.64	40.61	485.9	486.9	53/416	69/106	82	170	139	53	4.18	34
Average	+0.2%		-3.1%		--	--	+76%		-49%		+509%	

time constraints while 87 nets violates their constraint because of the failure of the buffer insertion. And the best result in [1] is that 260 nets meet their time constraints while 195 nets violates their constraint because of the failure of the buffer. The timing constraints are all satisfied in [1][3] because that those methods are based on fix-die placement, while the floorplanning should be optimized to meet the given timing constraints in our algorithm. Even though, we can see some improvement in the results because our algorithm can give a more feasible floorplan structure for buffer insertion.

Averagely, the timing constraints violation of F2 is 49% less than the results of F1. And the number of the nets which meet their constraints with buffer successful insertion increase 76% in F2 than F1. The algorithm of F2 can reduce the total wirelength, constraints violations significantly The experimental results show that the algorithm F2 can reduce the timing violations efficiently without much expense in area and wirelength.

6. CONCLUSION

In this paper, the buffer allocation is handled as an integral part in the floorplanning process. Not necessarily to scan the whole packing to find the dead spaces, we can partition the dead space into blocks while doing the packing. And the dead space blocks favor the later computation about the Feasible Region in the dead space. Instead of computing the size of the dead space in each grid, we compute the intersection between the dead space blocks and the FRs in a 2-step method. The computation of the possible buffer insertion sites for all net can be executed in the running time of O(m) where m is the number of the nets, since we can obtain the range of the possible buffer insertion independent of the grid sizes and we need to scan all the net only once. Experimental results show that our floorplanner can reduce the timing violation efficiently without much penalty in area and wirelength.

As the fundamental method of buffer planning, our method can be extended to optimize the buffer insertion in many different ways. And our algorithm can be extended to handle similar problem such as noise aware floorplanning and routability-driven floorplanning.

7. REFERENCES

[1] J. Cong, T.Kong, and D. Z. Pan, "Buffer block planning for interconnectdriven floorplanning," in *Proc. Int. Conf. Computer-Aided Design*, Nov.1999, pp. 358–363.

[2] C. J. Alpert and A. Devgan, "Wire segmenting for improved buffer insertion,"in *Proc. Design Automation Conf.*, June 1997, pp. 588–593.

[3] P. Sarkar, V. Sundararaman, and C. K. Koh. Routability-driven repeater block planning for interconnect-centric .floorplanning. *In ISPD 2000*.

[4]Hong Xianlong, Huang Gang et al. "Corner Block List: An Effective and Efficient Topological Representation of Non-slicing Floorplan" *ICCAD*'2000.pp.8-12.

[5] J. Cong, L. He, K.-Y. Khoo, C.-K. Koh, and Z. Pan, "Interconnect design for deep submicron ICs," in *Proc. Int. Conf. Computer-Aided Design*, Nov. 1997, pp. 478–485.

[6] W. C. Elmore, "The transient response of damped linear networks with particular regard to wide-band amplifiers," *J. Appl. Phys.*, vol. 19, pp.55–63, Jan. 1948.

[7] J. Cong and D. Z. Pan, "Interconnect delay estimation models for synthesis and design planning," in *Proc.ASP Design Automation Conf.*, Jan. 1999, pp. 97–100.

[8] J. Cong. " Challenges and opportunities for design innovations in nanometer technologies". In *Frontiers in Semiconductor Research: A Collection of SRC Working Papers*, 1997.

[9] X. P. Tang and D. Wong. "Planning buffer locations by network flows". *In Intl. Symp. Physical Design*, pages 186–191, 2000.

[10] Semiconductor Industry Association, *National Technology Roadmap for Semiconductors*. San Jose, CA: SIA, 1997.

[11] C. J. Alpert, J. Hu, S. S. Sapatnekar, and P. G.Villarrubia. "A practical methodology for early buffer and wire resource allocation". *In DAC*, 2001.