Buffer Block Planning for Interconnect-Driven Floorplanning

Jason Cong, Tianming Kong and David Zhigang Pan
Department of Computer Science
University of California, Los Angeles, CA 90095
Email: {cong, kongtm, pan}@cs.ucla.edu *

Abstract

This paper studies buffer block planning for interconnect-driven floorplanning in deep submicron designs. We first introduce the concept of feasible region (FR) for buffer insertion, and derive closed-form formula for FR. We observe that the FR for a buffer is quite large in general even under fairly tight delay constraint. Therefore, FR gives us a lot of flexibility to plan for buffer locations. We then develop an effective buffer block planning (BBP) algorithm to perform buffer clustering such that the overall chip area and the buffer block number can be minimized. To the best of our knowledge, this is the first in-depth study on buffer planning for interconnect-driven floorplanning with both area and delay consideration.

1 Introduction

For deep submicron (DSM) VLSI designs, it is widely accepted that interconnect has become the dominant factor in determining the overall circuit performance and complexity. Many interconnect synthesis techniques have been proposed recently for interconnect performance optimization, such as topology construction, driver sizing, buffer insertion, wire sizing and spacing (see [1] for a tutorial). Among them, *buffer insertion* in particular, is a very effective and useful technique by inserting active devices (buffers) to break original long interconnects into shorter ones such that the overall delay can be reduced. It has been shown that without buffer insertion, the interconnect delay for a wire increases about quadratically as wire length increases, but it only increases linearly under proper buffer insertion [2, 3, 4]. As an example, [5] showed that the delay of a 2cm global interconnect can be reduced in a factor of 7x by optimal buffer insertion. As the intrinsic delay of a buffer (i.e., the cost for buffer insertion) becomes smaller and the chip dimension gets larger, it is expected that a large number of buffers shall be inserted for high-performance designs in current and future technology generations (e.g., close to 800,000 for 50nm technology as estimated in [6]). The introduction of so many buffers will significantly change a floorplan, and shall be planned as early as possible, to ensure timing closure and design convergence.

However, most existing buffer insertion algorithms (e.g., [7, 8, 9, 10]) were designed for post-layout interconnect optimization, and also for a single net. There was no global planning for tens of thousands of nets that may need buffer insertion to meet their performance requirement as in DSM designs. Meanwhile, most existing floorplanning algorithms (e.g., [11, 12, 13]) only focused on wirelength/area minimization, and did not consider buffer insertion for performance optimization. [14] considered buffer insertion during floorplanning, but it simply assumed that buffers can be inserted anywhere in an existing floorplan. However, not as wires which may have over-the-cell routing structure (given multiple metal layers), buffers must consume silicon resource and require connections to power/ground network, thus may not be placed anywhere inside an existing circuit blocks. Otherwise, it will seriously impact the hierarchical design style, make it difficult to use/reuse IP blocks. As a result, the designers often prefer to form buffer blocks between existing circuit blocks of current floorplan, which may increase the total chip area. If there is no careful planning of these large amount of buffers, one may get excessive area increase. Moreover, without careful planning, it is most likely that these buffers will be distributed rather randomly over the entire chip, which will definitely complicate global/detailed routing and power/ground distribution.

To effectively address the above issues, as part of our general effort of developing an interconnect-centric design flow [15], we study in this paper the *buffer block planning* (BBP) problem, which automatically generates buffer blocks for interconnect optimization during physical-level floorplanning. It considers buffer location constraints (e.g., hard IP blocks and pre-design layout) and provides more regular buffer structure for ease of layout design and sharing of power/ground networks. Our major contributions of this paper include:

- We first introduce the concept of feasible region (FR) for buffer insertion under certain delay constraint, and derive an analytical formula for it.
- We find that the FR for a buffer can be surprisingly large, even under tight delay constraint. This crucial observation provides us a lot of flexibility to plan a buffer's location.
- We propose to use buffer blocks to appropriately cluster individual buffers together so that the total chip area due to buffer insertion, as well as the number of buffer blocks can be minimized.
- We develop an effective algorithm for buffer block planning. It can be used as a key element for interconnect-driven floorplanning.

To the best of our knowledge, this is the first in-depth study of buffer planning for interconnect-driven floorplanning. The rest of the paper is organized as follows. Section 2 formulates the problem. Section 3 derives the feasible region for buffer insertion. Section 4 studies buffer block planning and proposes an effective algorithm for it. Experimental results are shown in Section 5, followed by the conclusion in Section 6. Due to space limitation, we omit proofs to the theorems in this paper. Interested reader may refer to [16] for details.

2 Problem Formulation

We propose to study the following *buffer block planning* (BBP) problem: given an initial floorplan and the performance constraints for each net, we want to determine the optimal locations and dimensions of buffer blocks (i.e., the extra blocks between existing circuit blocks of current floorplan) such that the overall chip area and the number of buffer blocks after buffer insertion are minimized while the performance constraint for each net is satisfied (if it is a valid timing constraint for the given floorplan that can be met by optimal buffer insertion). The output from our buffer block planning

*This research is partially sponsored by Semiconductor Research Corporation under Contract 98-DJ-605, and equipment donation from Intel.

consists of the following information: the number of buffer blocks, each buffer block's area, location, and corresponding nets that use some buffer in this buffer block to meet the delay constraints. In this study, we focus on two-pin nets and derive the closed-form formula of feasible region for buffer insertion. The concepts of feasible region and buffer block planning can be extended to multiple-pin nets as well.

We model a driver/buffer as a switch-level RC circuit [5], and use the well-known Elmore delay model for delay computation. The key parameters for interconnect and buffer in our study are listed in Table 1. The values are based on a $0.18\mu m$ technology in NTRS'97[17].

Table 1: Key parameters

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 buffer (ps)	36.4
C_b	input capacitance of buffer (fF)	23.4
R_b	output resistance of buffer (Ω)	180

3 Feasible Region for Buffer Insertion

The *feasible region* (FR) for a buffer B is defined to be 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. Figure 1 illustrates the concept of FR for inserting 1 or k buffers into a net where the source and sink of the net are connected by a given route. In the figure, the FR's are the shaded line segments.

3.1 Feasible Region for Single-Buffer Insertion

For single-buffer insertion in Figure 1(a), let us denote x to be the distance from driver to buffer. We have the following theorem for its feasible region:

Theorem 1 *For a given delay constraint T_{req}, the feasible region $[x_{min}, x_{max}]$ for inserting one buffer is*

$$x_{min} = MAX\left(0, \frac{K_2 - \sqrt{K_2^2 - 4K_1K_3}}{2K_1}\right)$$

$$x_{max} = MIN\left(l, \frac{K_2 + \sqrt{K_2^2 - 4K_1K_3}}{2K_1}\right)$$

where

$$K_1 = rc$$
$$K_2 = (R_b - R_d)c + r(C_L - C_b) + rcl$$
$$K_3 = R_dC_b + T_b + R_b(C_L + cl) + \frac{1}{2}rcl^2 + rlC_L - T_{req}.$$

□

Note that for Theorem 1 to be valid, $K_2^2 - 4K_1K_3 \geq 0$ shall hold. Otherwise, no feasible region exists and the initial floorplanning/timing budget has to be modified. Figure 2 shows the FR for inserting one buffer to an interconnect of length from 6mm to 9mm. We first compute the best delay T_{best} by inserting one buffer, then assign the delay constraint to be $(1+\delta)T_{best}$, with δ to be from 0 to 50%. The x-axis shows the δ and the y-axis shows the FR distance, i.e., $x_{max} - x_{min}$. It is interesting to see that even with fairly small amount of slack, say 10% more delay from T_{best}, the FR can be as much as 50% of the wire length. This important observation leads great flexibility for buffer planning, to be discussed later on in this paper.

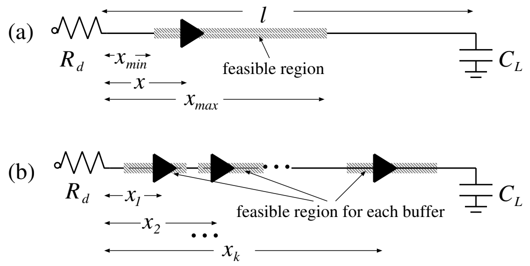

Figure 1: Feasible regions for inserting (a) one buffer; (b) k buffers.

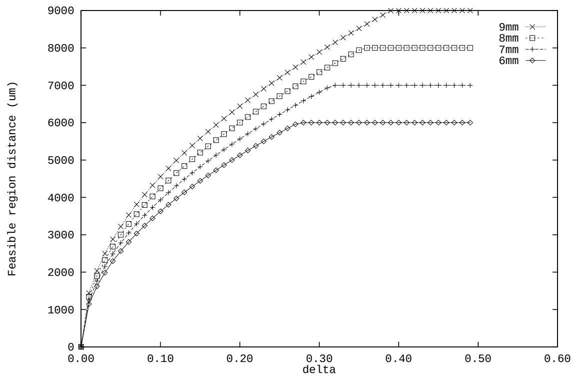

Figure 2: The distance feasible region for inserting a buffer under different delay constraint of δ.

3.2 Feasible Regions for Multiple-Buffer Insertion

For a long interconnect, more than one buffer may be needed to meet a given delay budget. For k buffers inserted, we have the following theorem to compute the feasible region for each buffer.

Theorem 2 *For a long interconnect with k buffers inserted, the feasible region for the i-th buffer ($i \leq k$) is $x_i \in [x_{min}(k,i), x_{max}(k,i)]$ with*

$$x_{min}(k,i) = MAX\left(0, \frac{K_2' - \sqrt{K_2'^2 - 4K_1'K_3'}}{2K_1'}\right)$$

$$x_{max}(k,i) = MIN\left(l, \frac{K_2' + \sqrt{K_2'^2 - 4K_1'K_3'}}{2K_1'}\right)$$

where K_1', K_2' and K_3' are functions of k and i (for simplicity of notation, we drop them in the above equations) with

$$K_1'(k,i) = \frac{(k+1)rc}{2i(k-i+1)}$$

$$K_2'(k,i) = \frac{(R_b - R_d)c}{i} + \frac{r(C_L - C_b) + rcl}{k-i+1}$$

$$K_3'(k,i) = kT_b - T_{req} + \left[R_d + (i-1)R_b + \frac{(k-i)rl}{k-i+1}\right] \cdot C_b + R_b[(k-1)C_b + C_L + cl] + \frac{rcl^2}{2(k-i+1)} + rlC_L - \frac{(i-1)c(R_b - R_d)^2}{2ir} - \frac{(k-i)r(C_b - C_L)^2}{2(k-i+1)c}.$$

□

It can be verified that Theorem 1 is a special case of Theorem 2 with $k = i = 1$. The following theorem determines the minimum number of buffers that are required to meet a given delay budget.

Theorem 3 *The minimum number of buffers to meet the delay constraint T_{req} for an interconnect of length l is*

$$k_{min} = \left\lceil \frac{K_5 - \sqrt{K_5^2 - 4K_4K_6}}{2K_4} \right\rceil \tag{1}$$

where

$$K_4 = R_bC_b + T_b \tag{2}$$

$$K_5 = T_{req} + \frac{r}{c}(C_b - C_L)^2 + \frac{c}{r}(R_b - R_d)^2 - (rC_b + cR_b)l - T_b - R_dC_b - R_bC_L \tag{3}$$

$$K_6 = \frac{1}{2}rcl^2 + (rC_L + cR_d)l - T_{req} \tag{4}$$

□

Based on these results, given a two-pin net with a delay constraint, k_{min} and the feasible region for each buffer can be computed in constant time. As a simple example, for a 1cm net with $R_d = R_b$, $C_L = C_b$ and the delay constraint $T_{req} = 1.05 \cdot T_{best}$ (T_{best} is the best delay by optimal buffer insertion, which is 464ps), we can calculate that the minimum number of buffers needed is $k_{min} = 2$, and the feasible regions for the first and second buffers are $[1.47mm, 5.20mm]$ and $[4.80mm, 8.53mm]$, respectively. Note that the FR's of adjacent buffers may overlap, as in this example. This is because FR for each buffer is computed independently, assuming all other buffers can be optimally placed to satisfy the delay constraint, i.e., our FR provides the maximum freedom for each a buffer. It shall be noticed that during the buffer planning phase (in Section 4), when a buffer is placed (i.e., "committed") to a position within its feasible region, we will need to update the FR's of all other *unplaced* buffers of the same net to safely meet its delay constraint. But since we have the analytical formula, this update can be computed in constant time.

3.3 2-Dimensional Feasible Region

So far, our discussion of FR is restricted to a one-dimensional line, i.e., we assume the route from source to sink is already specified by some global router. Thus the feasible region is also one-dimensional. In practice, however, global routing usually has not been performed prior to or during floorplanning. In this case we can compute a much larger 2-dimensional FR for each buffer. This 2-dimensional feasible region is essentially the *union* of the one-dimensional feasible regions of *all* possible routes from source to sink. Therefore, we can have much more freedom for buffer planning. Since for each net, its buffer location will then determine roughly its routing, our buffer block planning indeed determines the overall global routing structure for each net.

For a two-dimensional net, let the source location be (x_{src}, y_{src}) and the sink location be (x_{sink}, y_{sink}). We only need to consider non-degenerate two-dimensional cases here, i.e., $x_{src} \neq x_{sink}$ and $y_{src} \neq y_{sink}$. Also, we consider only the monotone (i.e., non-detour) routes from source to sink. We prove that with Manhattan monotone routing, the 2-D FR can be obtained by the following theorem.

Theorem 4 *For a net with k buffers, the 2-dimensional feasible region for the i-th buffer is the region bounded by two parallel lines with Manhattan distances from the source to be $x_{min}(k,i)$ and $x_{max}(k,i)$, respectively (the same as Theorem 2), and by the rectangular bounding box between the source and the sink. The slope of the two parallel lines is either +1 or -1, depending on the sign of $\frac{y_{sink}-y_{src}}{x_{sink}-x_{src}}$: if $\frac{y_{sink}-y_{src}}{x_{sink}-x_{src}} > 0$, the slope is -1; if $\frac{y_{sink}-y_{src}}{x_{sink}-x_{src}} < 0$, the slope is +1.* □

Note that in previous works of buffer insertion, buffers are mostly inserted in their delay-minimal positions, which we call them as *restricted* positions because they are only a small subset within our FR. The restricted positions for a 2-dimensional net can be obtained by the following corollary:

Corollary 1 *For a 2-D net with k buffers, the restricted positions of the i-th buffer for all monotone routes from the source to the sink form a restricted line within the feasible region of the i-th buffer. The line slope is again either +1 or -1, the same as that in Theorem 4.* □

Also, if there are obstacles (such as hard IP blocks), we just need to deduct them from the feasible region. An example of a two-dimensional feasible region with a restricted line and some obstacles is illustrated in Figure 3.

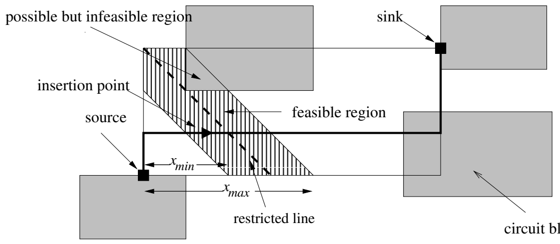

Figure 3: 2-D feasible region and a restricted line. The existing circuit blocks act as obstacles for buffer insertion.

4 Buffer Block Planning

In the previous section, we show that for a given delay constraint, a buffer may be inserted in a fairly wide feasible region. Therefore, it gives us a lot of flexibility to plan for every buffer's insertion position (within its FR) such that the overall chip area due to buffer insertion, as well as the total number of buffer blocks can be minimized. It shall be noted that such a buffer block planning also determines the overall global routing structures for long interconnects by determining their internal buffer locations. In this section, we will present an effective algorithm for buffer block planning.

The BBP problem is very difficult in the following senses: (i) Many buffer blocks might need to be optimally shaped for overall chip area minimization; (ii) To make the situation even more complicated, different buffers of the same net will not be independent of each other. For a long interconnect with more than one buffers inserted, Theorem 2 gives the maximum FR for each buffer. However, when a buffer is committed to a certain location within its FR, the FR's for other buffers in the same net will have to be updated so that the delay constraint can be safely met[1].

In the rest of this section, we will present an effective algorithm to solve the buffer block planning problem. There are several important features in our BBP algorithm: (i) It takes advantage of both the flexibility of FR and the simplicity of its analytical formulae, so

[1] Fortunately, we have the analytical formula to compute FR. Thus, the update can be done extremely fast.

Algorithm: Buffer Block Planning (BBP)
1. Build horizontal and vertical polar graph;
2. Build tile data structure;
3. For each tile, compute its area slacks;
4. **while** (there exists buffer to be inserted) {
5. tile $\leftarrow$ **Pick_A_Tile**();
6. **Insert_Buffers**(tile);
7. update W_c,H_c, FR and area slacks;
8. }

Figure 4: Overall flow of the buffer block planning algorithm.

that one may handle large circuits with tens of thousands long interconnects easily; (ii) Since in most floorplans, there are some *dead areas* that cannot be taken by any circuit module, our algorithm will use these dead areas as much as possible to save the overall chip area; (iii) Different from previous buffer insertion algorithm which only inserts one buffer for a single net, our BBP algorithm always maintains *global* buffer insertion information for *all* nets, thus it can effectively cluster individual buffers that belong to different nets into buffer blocks.

Figure 4 gives the overall flow of the buffer block planning algorithm. Lines 1 to 3 are the data preparation stages. First, we will build the horizontal and vertical polar graphs [18], for the given floorplan denoted as G_H and G_V, respectively. Let us take G_H to illustrate how to build the horizontal polar graph. G_H is a directed graph, each vertex v in it corresponds to a vertical channel, and an edge $e = (v_1, v_2)$ corresponds to a circuit module whose left and right boundaries are adjacent to channels v_1 and v_2, respectively. For each vertex v, we assign its weight $w(v)$ to be its corresponding channel width. Similarly, for each edge e, we assign its weight $w(e)$ to be its corresponding module width. Graph G_V can be built similarly. By running longest path algorithm on G_H/G_V, we can obtain the width/height of the chip (denoted by W_c/H_c). For those channels not on the critical paths in G_H/G_V, we will have some positive slacks in width/height, which lead to dead areas. It shall be noted that during buffer insertion, some circuit modules may have to shift to make room for buffer blocks (e.g., if no dead area exists). Therefore, a horizontal channel's height or a vertical channel's width may increase during BBP.

To better represent buffer block and facilitate easy data manipulation such as feasible region intersection, we divide each channel into a set of rectangular tiles. Then we compute for each tile τ, its slack with respect to the longest path in the polar graph G_H or G_V.

The **while** loop from lines 4 to 8 is the main part of our BBP algorithm. The iterative buffer insertion process will continue as long as there is still some net that needs buffer(s) to meet performance constraint. Each iteration of the **while** loop has two major steps: first, we will pick a best tile for buffer insertion (**Pick_A_Tile**); then, we will insert proper buffers into this tile (**Insert_Buffers**).

To pick the best tile in each iteration, the **Pick_A_Tile** routine works in the following two modes, depending on whether there exists some useful dead area for buffer insertion or not:

1. There exists some tile whose area slack is positive (due to dead area). In this case, buffers inserted into this tile will not increase the overall chip area as long as the total area of buffers inserted in the tile is smaller than this tile's area slack. For example of a tile τ in a vertical channel, suppose its width slack is w_τ, and its height is H_τ. Then we can insert as many as $\lfloor w_\tau H_\tau / A_b \rfloor$ buffers into tile τ without increasing the overall chip area, where A_b is the area of a buffer. The actual number of buffers that can be inserted into τ may be smaller, since only those buffers whose FR intersects with tile τ can be candidates to be inserted into τ. Therefore, the number of buffers that can be inserted into τ, without chip area increase is $n_\tau = \min(\lfloor w_\tau H_\tau / A_b \rfloor, m_\tau)$, where m_τ is the number of buffers whose FR's intersect with tile τ. Since we may have multiple tiles with positive slack (especially at the beginning of BBP), we will pick the one with largest n_τ because this greedy approach shall reduce the total number of buffer blocks, which is also our buffer block planning objective.

2. There is no tile with positive area slack. Then, any buffer insertion will increase the overall chip area. When some buffer is inserted into a tile, we have to shift some circuit modules. This shifting will make room for other tiles, so we will have some new positive-slack tiles. Our tile selection process will try to maximize such opportunity. Notice that as a buffer is inserted in τ, other tiles in the same channel with τ will have positive area and tend to have buffers inserted in the future, thus the chance of buffers clustering increases. To maximize such effect, we will pick the channel that has the maximum buffer insertion demand and choose one tile in it. Note that in this scenario, since we need to expand the channel, we only insert one buffer into it to minimize the area increase.

Our strategy for **Insert_Buffers** into the tile τ that has just been picked by **Pick_A_Tile** also works in two modes, corresponding to those two in **Pick_A_Tile**:

1. The tile τ has dead area. From case 1 in **Pick_A_Tile**, we know that n_τ buffers can be inserted into the tile. Meanwhile, there are m_τ buffer candidates whose FR's intersect with tile τ, with $m_\tau \geq n_\tau$. Then if $n_\tau = m_\tau$, we will insert all these m_τ buffers; if $m_\tau > n_\tau$, we will only insert first n_τ buffers out of these m_τ buffers, sorted according to the increasing size of their FRs. Different from previous approaches that just inserts one buffer for one net, our approach inserts as many as n_τ buffers for n_τ different nets simultaneously. Since all of them are clustered into tile τ, they form a natural buffer block.

2. The tile does not have dead area, but needs expansion to make room for any buffer insertion. In this case, we only insert one buffer, i.e., $n_\tau = 1$. Again, if there are multiple buffers that can be inserted in this tile, we insert the one with tightest FR constraint.

After deciding how many and which n_τ buffers are inserted ("committed") into tile τ, we will update the following information: (i) The feasible regions of "uncommitted" buffers in the same net for which we just inserted a buffer into τ; (ii) The corresponding vertex (i.e., channel) weights in G_H and/or G_V that are affected by the insertion of the buffer block; (iii) The new chip dimension W_c, H_c and the slacks for each channel and tile. Then we repeat the buffer insertion/clustering process until all buffers are placed. It shall be pointed that our BBP algorithm can handle both slicing and non-slicing floorplanning structures.

5 Experimental Results

We have implemented our buffer block planning algorithms using C++ on an Intel Pentium-II machine with 256M-byte main memory. This section presents the experimental results. The parameters (refer to Table 1) used in our experiments are based on a $0.18\mu m$ technology in the NTRS'97 roadmap [17].

We have tested our algorithms on 11 circuits, as summarized in Table 2. The first six circuits are from MCNC benchmark [19], and the other five are randomly generated. In this paper, we focus on 2-pin nets, so we decompose each multi-pin net into a set of

Table 2: Test circuit statistics.

circuit	#modules	# nets	# pads	#2-pin nets
apte	9	97	73	172
xerox	10	203	2	455
hp	11	83	45	226
ami33	33	123	43	363
ami49	49	408	22	545
playout	62	2506	192	2150
ac3	27	212	75	446
xc5	50	1005	2	2275
hc7	77	449	51	1450
a9c3	147	1202	22	1613
pc2	124	3126	192	4204

source-to-sink 2-pin nets[2]. We then compute the critical length for buffer insertion (defined to be the minimal interconnect length that buffer insertion is needed for delay reduction) using the analytical formula in [20]. We will use it to filter out short interconnects, i.e., if a net is shorter than the critical length, we will ignore it during buffer block planning since buffer insertion can not help reduce its delay. The initial floorplan for each circuit is generated by running the simulated tempering (an improved Monte Carlo technique of simulated annealing) algorithm as in [21]. For each net, we first compute its best delay by optimal buffer insertion T_{opt} [20], and then randomly assign its delay budget to be $1.05 \sim 1.20T_{opt}$.

We compare our BBP algorithm with a conventional buffer insertion algorithm without trying to plan buffer positions, i.e., at each iteration, a buffer is randomly picked and assigned to a feasible location, denoted as RDM algorithm. We run BBP and RDM under two different scenarios: one is RES where a buffer can only be located in its delay-minimal restricted position(s) (see Figure 3); and the other is FR where a buffer may be inserted *anywhere* in its feasible region. The results for the four different algorithmatic combinations are summarized in Table 3, where BBP/RES means BBP algorithm applied to scenario RES, RDM/FR means RDM applied to scenario FR, and so on.

In Table 3, we report for each algorithm combination: (i) the total number of buffers inserted to meet performance constraints ($buffers$), (ii) the number of buffer blocks (*#BB*), (iii) the number of 2-pin nets that can meet their delay constraints (*#meet*)[3], (iv) the chip area increase due to buffer insertion in percentage (*area*), and (v) the CPU time in second (*cpu*). It is interesting to observe from the table that

- Under the same RES scenario (i.e., only the restricted positions are allowed for buffer insertion), the RDM and BBP algorithms will have about the same number of buffers inserted and the same number of nets meeting their delay constraints. However, our BBP algorithm is able to explicitly cluster appropriate buffers together, so that it leads to significant area saving and much less number of buffer blocks than RDM algorithm. For example of circuit *pc2*, the area increase of BBP/RES is 3.21%, whereas that of RDM/RES is about 9% (2.8x larger); the #BB of BBP/RES is 542, whereas that of RDM/RES is about 1290 (2.38x larger). The same conclusion about the comparison of BBP versus RDM holds for the FR scenario. It is also interesting to observe that BBP algorithm does not indeed increase CPU time from RDM. Actually, it may use slightly less run time. This is because during BBP, one buffer block (not just one buffer) can be determined at a time.
- Under the same algorithm, e.g., BBP, the usage of FR significantly increases the number of nets that can meet their delay constraints (for example of *ac3*, from 289 to 369, a 28% increase). This is because our feasible region is usually much larger than the delay-minimal RES locations, so that one can avoid existing circuit/buffer blockages during buffer insertion. Note that as *#meet* increases, the number of buffers inserted to meet performance constraints also increases accordingly from RES to FR. However, since the FR provides much more freedom during buffer clustering, the number of buffer blocks (*#BB*) in fact reduces (for example of of *a9c3*, from 557 to 366, a 34% reduction); and the area expansion due to buffer insertion is also less by using FR with better buffer clustering. Since the FR computation/update can be computed in constant time, the run times under FR scenario only increase slightly compared to those under RES. As a result, our largest example (*pc2* with more than 13,000 buffers) only takes about 100s.

To summarize, it is obvious that the BBP/FR is the best combination among these four to meet delay targets, with very marginal area increase (less than 2.1% for all test cases), least number of buffer blocks and comparable CPU times. It shall be noticed, however, that even under this best algorithm, there may still exist quite some nets that cannot meet their delay constraints under some given floorplan and timing budget. Therefore, it is important to have an interconnect-driven floorplanning engine to work closely with our BBP/FR algorithm. We are currently working on it.

As an example, we show in Figure 5 the circuit buffer block layouts from RDM/RES and BBP/FR on circuit *xerox*.

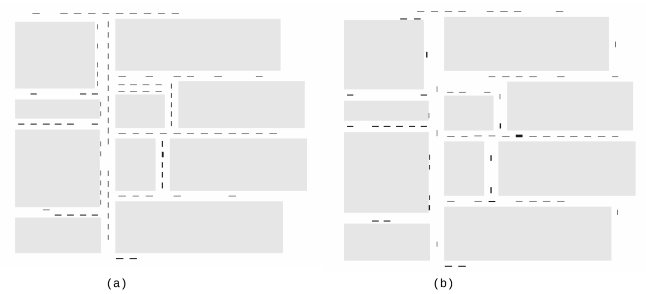

Figure 5: Floorplan and buffer block layouts of the MCNC circuit *xerox* by (a) RDM/RES; (b) BBP/FR. The ten big blocks are circuit functional modules, and the rest are buffer blocks.

6 Conclusion

In this paper, we first introduce the concept of feasible region for buffer insertion and derive the analytical formula to compute FR under given delay constraint. We then propose an effective buffer block planning (BBP) algorithm to automatically generate buffer blocks for interconnect optimization with chip area and buffer block number minimization. Experimental results show that our BBP/FR algorithm leads to significant improvement over previous buffer insertion/planning algorithms. We believe that our buffer block planning scheme will play a central role in an interconnect-driven floorplanning system.

[2] Note that the number of 2-pin nets is possibly smaller than that of original nets ($playout$) because the power/ground and single-pin nets are excluded.

[3] A net will fail to meet its delay constraint if the given delay constraint is too tight, or its buffer's feasible region is fully occupied by existing circuit blocks.

Table 3: Comparison of four different buffer insertion/planning algorithms.

	buffers	*#BB*	*#meet*	*area*	*cpu(s)*	*buffers*	*#BB*	*#meet*	*area*	*cpu(s)*
circuit	RDM/RES					RDM/FR				
apte	157	74	81	1.50%	0.14	181	93	98	1.81%	0.25
xerox	343	102	210	1.85%	0.36	403	116	262	2.11%	0.56
hp	242	119	106	2.48%	0.28	274	147	128	2.78%	0.47
ami33	625	278	271	2.91%	1.22	660	306	302	3.41%	1.78
ami49	867	376	342	3.19%	2.47	921	404	394	3.42%	3.78
playout	3997	581	1337	3.74%	13.23	4244	730	1515	4.06%	17.61
ac3	649	267	302	2.56%	1.00	736	290	377	2.90%	1.61
xc5	2860	460	1379	4.20%	7.95	3197	499	1727	4.21%	11.83
hc7	2557	909	967	6.50%	11.56	2662	941	1053	6.63%	17.89
a9c3	4035	1082	1257	4.46%	26.09	4236	1156	1416	4.66%	34.80
pc2	12237	1290	2687	8.95%	80.11	13157	1371	3339	8.99%	122.34
	BBP/RES					BBP/FR				
apte	168	48	90	0.80%	0.14	185	34	102	0.69%	0.23
xerox	330	68	195	1.06%	0.33	399	66	260	1.38%	0.53
hp	245	77	109	1.49%	0.30	280	64	131	1.24%	0.48
ami33	616	161	259	1.49%	1.06	667	125	305	1.36%	1.63
ami49	882	197	355	1.25%	2.08	946	136	412	0.78%	3.25
playout	4016	245	1350	1.13%	10.02	4263	201	1533	0.84%	13.98
ac3	639	145	289	1.28%	0.86	733	118	369	1.11%	1.39
xc5	2920	285	1431	2.58%	6.38	3210	193	1739	1.79%	10.16
hc7	2542	455	946	2.86%	9.89	2693	299	1068	1.92%	15.88
a9c3	4082	557	1293	1.83%	21.13	4265	366	1446	0.89%	29.20
pc2	12530	542	2837	3.21%	63.91	13238	416	3462	2.02%	99.84

Acknowledgments

The authors would like to thank Dr. Wilm Donath from IBM T. J. Watson Research Center, Dr. Norman Chang from HP Labs, and Dr. Lukas van Ginneken from Magma Design Automation for their helpful discussions.

References

[1] J. Cong, L. He, C.-K. Koh, and P. H. Madden, "Performance optimization of VLSI interconnect layout," *Integration, the VLSI Journal*, vol. 21, pp. 1–94, 1996.

[2] H. B. Bakoglu, *Circuits, Interconnections, and Packaging for VLSI*. Addison-Wesley, 1990.

[3] R. Otten, "Global wires harmful?," in *Proc. Int. Symp. on Physical Design*, pp. 104–109, Apr. 1998.

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

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

[6] J. Cong, "Challenges and opportunities for design innovations in nanometer technologies.," in *SRC Working Papers, http://www.src.org/prg_mgmt/frontier.dgw*, Dec. 1997.

[7] L. P. P. P. van Ginneken, "Buffer placement in distributed RC-tree networks for minimal Elmore delay," in *Proc. IEEE Int. Symp. on Circuits and Systems*, pp. 865–868, 1990.

[8] J. Lillis, C. K. Cheng, and T. T. Y. Lin, "Optimal wire sizing and buffer insertion for low power and a generalized delay model," in *Proc. Int. Conf. on Computer Aided Design*, pp. 138–143, Nov. 1995.

[9] T. Okamoto and J. Cong, "Buffered Steiner tree construction with wire sizing for interconnect layout optimization," in *Proc. Int. Conf. on Computer Aided Design*, pp. 44–49, Nov. 1996.

[10] C. C. N. Chu and D. F. Wong, "Closed form solution to simultaneous buffer insertion/sizing and wire sizing," in *Proc. Int. Symp. on Physical Design*, pp. 192–197, 1997.

[11] W. M. Dai, B. Eschermann, E. S. Kuh, and M. Pedram, "Hierachical placement and floorplanning for bear," in *IEEE Trans. on Computer-Aided Design*, pp. 1335–1349, 1989.

[12] D. Wong and C. L. Liu, "Floorplan design of VLSI circuits," in *Algorithmica*, pp. 263–291, 1989.

[13] H. Murata, K. Fujiyoshi, S. Nakatake, and Y. Kajitani, "Rectangle-packing-based module placement," in *Proc. of Int. Conf. on Computer-Aided Design*, pp. 472–479, 1995.

[14] M. Kang, W. Dai, T. Dillinger, and D. LaPotin, "Delay bounded buffered tree construction for timing driven floorplanning," in *Proc. of Int. Conf. on Computer-Aided Design*, pp. 707–712, 1997.

[15] J. Cong, "An interconnect-centric design flow for nanometer technologies," in *Proc. of Int'l Symp. on VLSI Technology, Systems, and Applications*, pp. 54–57, June, 1999.

[16] J. Cong, T. Kong, and D. Z. Pan, "Buffer block planning for interconnect-driven floorplanning," Tech. Rep. 990036, UCLA CS Dept, 1999.

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

[18] R. Otten, "Graphs in floor-plan design," *International Journal of Circuit Theory and Applications*, vol. 16, pp. 391–410, Oct. 1988.

[19] "http://www.cbl.ncsu.edu/cbl_docs/lys92.html,"

[20] C. J. Alpert and A. Devgan, "Wire segmenting for improved buffer insertion," in *Proc. Design Automation Conf*, 1997.

[21] J. Cong, T. Kong, D. Xu, F. Liang, J. S. Liu, and W. H. Wong, "Relaxed simulated tempering for VLSI floorplan design," in *Proc. Asia and South Pacific Design Automation Conf.*, pp. 13–16, Jan., 1999.