A Practical Methodology for Early Buffer and Wire Resource Allocation

Charles J. Alpert, *Senior Member, IEEE*, Jiang Hu, Sachin S. Sapatnekar, *Fellow, IEEE*, and Paul G. Villarrubia

***Abstract*—As technology scales, interconnect-centric design flows become imperative for achieving timing closure. Preplanning buffers and wires in the layout is critical for such flows. Both buffers and wires must be considered simultaneously, since wire routes determine buffer requirements and buffer locations constrain the wire routes. In contrast to recently proposed buffer-block planning approaches, our novel design methodology distributes a set of *buffer sites* throughout the design. This allows one to use a tile graph to abstract the *buffer planning* problem and simultaneously address *wire planning*. We present a four-stage heuristic called resource allocation for buffer and interconnect distribution for resource allocation that includes a new, efficient technique for buffer insertion using a length-based constraint. Extensive experiments validate the effectiveness of this approach.**

***Index Terms*—Buffer insertion, deep submicron, interconnect synthesis, layout, physical design, Steiner tree.**

I. Introduction

BUFFER insertion has become a critical step in deep submicrometer design as interconnect now plays a dominating role in determining system performance. The insertion of buffers and inverters on signal nets can provide several advantages, including reducing interconnect delay, restraining noise, improving the slew rate, and fixing electrical violations [3], [13]. Current designs easily require thousands of nets to be buffered, and Cong [6] speculates that close to 800 000 buffers will be required for designs in 50-nm technology.

Achieving timing closure becomes more difficult when buffer insertion is deferred to the back end of the design process, and the buffers must be squeezed into whatever leftover space remains. The problem is particularly acute for custom designs, where large IP core macros and custom data flow structures are present, blocking out significant areas from buffering possibilities. Application-specified integrated circuit (ASIC) designs can also run into similar headaches if they are dense or have locally dense hot spots.

To manage the large number of buffers and also achieve high performance on the critical global nets, buffers must be planned for early in the design so that the rest of the design flow is aware of the required buffering resources. In addition, design routability has also become a critical problem; one must make sure that an achievable routing solution exists during the physical floorplanning stage. Thus, global wiring must be planned early to minimize routing congestion, hot spots, and crosstalk problems later on in the flow.

Manuscript received September 12, 2001; revised March 22, 2002 and Augest 5, 2002. This paper was recommended by Associate Editor C.-K. Cheng.

C. J. Alpert and P. G. Villarrubia are with the IBM Corporation, Austin, TX 78758 USA (alpert@austin.ibm.com).

J. Hu is with the Department of Electrical Engineering, Texas A&M University, College Station, TX 77843-3128 USA.

S. S. Sapatnekar is with the Department of Electrical Engineering and Computer Science, University of Minnesota, Minneapolis, MN 55455 USA.

Digital Object Identifier 10.1109/TCAD.2003.810749

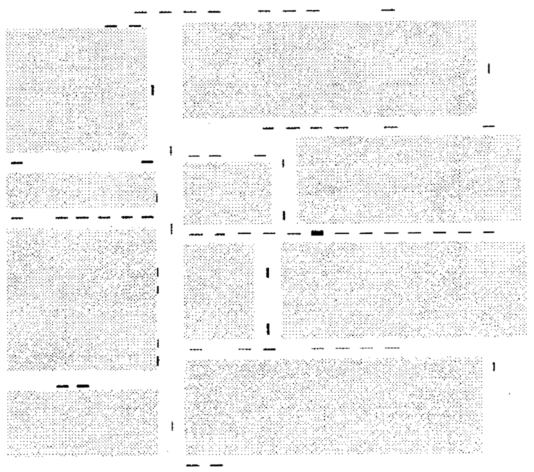


Fig. 1. Buffer-block plan on MCNC xerox circuit [9], reproduced here with permission of the authors. The ten big blocks are functional blocks and the rest are buffer blocks.

A. Buffer-Block Planning Methodology

In response to the need for an interconnect-centric design methodology, a new body of research on *buffer-block planning* has recently established itself in the literature [8], [10], [11], [16], [17]. These works focus on “physical-level interconnect planning,” as described in [7]. The works of [8], [16], and [17] all propose the creation of additional buffer blocks to be inserted into an existing floorplan. These buffer blocks are essentially top-level macro blocks containing only buffers. Cong *et al.* [8] proposed to construct these blocks using *feasible regions*. A feasible region is the largest polygon in which a buffer can be inserted for a particular net such that the net’s timing constraint is satisfied. Sarkar *et al.* [16] added a notion of independence to the feasible regions in [8] while also trying to relieve routing congestion during optimization. Tang and Wong [17] proposed an optimal buffer-block planning algorithm in terms of maximizing the number of inserted buffers (assuming that one buffer is sufficient for each net). Finally, Dragan *et al.* [10] presented a multicommodity flow-based approach to buffering two-pin nets assuming that a buffer-block plan had already been created. This approach was extended to multipin nets in [11].

Fig. 1 shows the result of buffer-block planning [8] on the Microelectronics Center of North Carolina (MCNC) circuit *xerox*, where the buffer blocks are indicated by dashes in between blocks. Observe that buffers are essentially packed between larger existing floorplanned blocks. We argue there are two fundamental problems with the buffer-block planning approach.

1) Since buffers are used to connect global wires, there will be considerable contention for routing resources in the regions between macro blocks. The design may not be routable due to heavy congestion between blocks.
2) Buffers must be placed in poor locations since better locations are blocked. Some blocks may even be so large that routing over the block is infeasible, even if buffers are inserted immediately before and after the block. For example, signal integrity could degrade beyond the point of recovery or wire delay may simply be too high. One may be able to alleviate the problem by using wider wires on thick metal, powering up to very large buffers, etc., but these solutions exacerbate the congestion problem

The flaws are not with buffer-block planning *per se*; rather, it is certainly a reasonable method for preplanning buffers within current design flows. However, buffer-block planning is really an interconnect-centric idea being applied to a device/logic-centric flow. Ultimately, this methodology will not be sustainable as design complexity continues to increase. A different methodology is required!

Ideally, buffers should be dispersed with some regularity throughout the design. Clumping buffers together, e.g., in buffer blocks, or between abutting macros invites routing headaches. A more uniform distribution of buffers will also naturally spread out global wires. *There must be a way to allow buffers to be inserted inside blocks*.[1]

B. Buffer Site Methodology

We propose an alternative methodology. Macro block designers must allow global buffer and wiring resources to be interspersed within their designs wherever possible. This resource allocation need not be uniform; a block with a lower performance requirement and complexity may be able to afford to allocate a higher percentage of its resources. A cache or blocks within a data path may not be able to allocate any resources. Ideally, as this "hole in a macro" methodology becomes widespread, even future IP blocks will have some of their area devoted to buffering resources.

To set aside a buffer resource within a block, the designer can insert what we call a *buffer site*, i.e., physical area which can denote either a buffer, inverter (with a range of power levels), or even a decoupling capacitor. When a buffer site gets assigned to a net, a logical gate from the technology is actually specified. Until this assignment takes place, buffer sites are not connected to any nets.

Allocating a percentage of a macro block for buffer sites may be viewed as wasteful; however, if the sites are not used for buffering, there are other ways to utilize them. In fact, it is desirable not to place cells in a particular block at 100% density for the following reasons.

[1]Note that we are referring to buffers for signal nets, not clock buffers. Compared to generic buffers, clock buffers occupy more area, draw more current, create hot spots, etc., all of which require different constraints and objectives than proposed in this work.

- Space needs to be allocated for spare circuits to facilitate metal-only engineering changes (ECOs) late in the design cycle.
- The design needs to be populated with decoupling capacitors to enhance local power supply and signal stability.
- Spreading of the design will help maximize the probability of successful routing by spreading out areas with significant wire congestion.

For these reasons, current IBM ASIC methodology typically uses a placement density of 70% or less, e.g., in any small region of the chip area, at least 30% of the design will be empty. Thus, often the space for buffer sites already exists; it just needs to be exploited.

Buffer sites can also be a powerful tool for semi-custom designs [19]. For example, a data flow typically routes several regular signal buses across collections of data flow elements. These routes should be implemented with straight wires wherever possible. Collectively, the wires occupy a dense portion of the routing region which severely limits the ability for either detours or other routes in the data path itself. If the data bus nets require buffering, one would like to have buffer locations available within the data path routing region to avoid detours. If the buffers lie outside the routing region, the wires may have to take large detours to reach the buffers which can significantly hurt the timing on a part of the design that can ill afford it. If buffer sites are designed into the original data path layout, it is possible to add buffers late in the design cycle while maintaining straight wiring of the data bus nets.

Buffer sites can also be used for a flat design style, e.g., a sea of buffer sites can be sprinkled throughout the placement. For hierarchical designs, one can view the buffer sites as flat to derive a similar sprinkling, but their distribution will likely be less uniform. Some macro blocks could have 5%–10% of the area devoted to buffer sites, while a similar block in a different part of the chip may require much less. To help decide the allocation of buffers sites to macros, one could assume an infinite number of available buffer sites, run a buffer allocation tool like resource allocation for buffer and interconnect distribution (RABID), and compute the number of buffers inserted in each block. Then, this number can be used to help determine the actual number of buffer sites to allocate within the block. No matter which design style is used, a resource allocation algorithm can view buffer sites as flat, which enables it to make assignments to global routes based on buffer site distribution.

Some advantages of this methodology are as follows.

- Routing congestion between blocks can easily be alleviated.
- Buffers do not have to be placed in suboptimal locations when the optimal location lies inside a block.
- One no longer has to squeeze buffers into tight spaces between blocks, thereby forcing other cells to be moved which can cause timing violations and hurt timing closure.
- Slew constraints, which previously could only be satisfied with thick metal (with wide wires) routing, can now be satisfied more easily since the space between buffers can be reduced in the presence of large blocks.

The main disadvantage of this methodology is that several tools have to be modified or built from scratch to handle buffer sites. The data model must recognize a buffer site, the router must be able to route into a buffer site, the design environment must allow the user to create and delete buffer sites, etc. Some tools may be able to piggyback onto support already built in for ECOs. For example, the placement tool needs to be able to disperse buffer sites within a design; this task is quite similar to the dispersing spare logic cells in the placement for future ECOs. Further, the methodology still must be able to handle certain custom blocks with highly structured array-type logic which cannot afford to have any buffer sites inserted. If these blocks become too large though, they must be either split or spread out in such a way as to allow buffers inside, even if they are inserted in a regular structure.

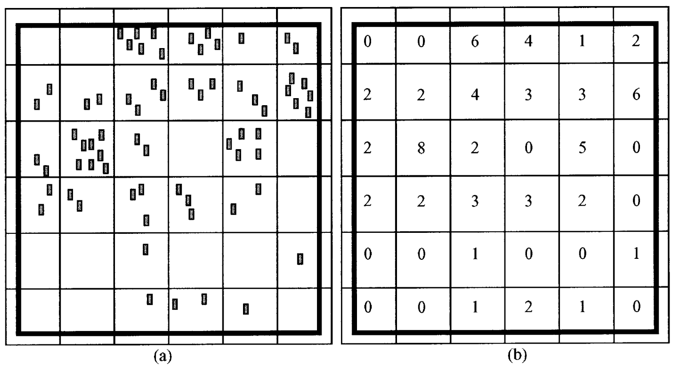


Fig. 2. (a) Set of 68 buffer site locations can be tiled and (b) abstracted to a total number of buffer sites lying within each tile.

C. Technical Contribution

We propose a new buffer and wire resource allocation formulation. Assuming that locations for buffer sites have already been chosen, the problem is to assign buffers to global nets such that each buffer corresponds to an existing buffer site. We model the problem with a tile graph to manage the complexity of thousands of buffer sites and to integrate wire congestion into the problem statement. We propose the following four-stage heuristic called RABID.

1) Construct low-cost, low-radius Steiner trees for each net.
2) Rip-up and reroute nets to reduce wire congestion.
3) Insert buffers on all nets which require them. This stage is based on a van Ginneken [18] style dynamic programming algorithm, yet we can find the optimal solution for a given net more efficiently than [18].
4) Rip-up, reroute, and reinsert buffers on nets to reduce both wire and buffer congestion.

Unlike the approaches in [8], [10], [16], and [17], our algorithm is designed to handle nets with multiple sinks (as is [11]).

The remainder of the paper is as follows. Section II describes our problem formulation and modeling assumptions, and Section III presents our four-step heuristic. Experiments are presented in Section IV and we conclude in Section V.

II. Problem Formulation

There are two fundamental characteristics of buffer and wire planning which drive our formulation.

1) Finding the absolute optimal locations for a buffer is not particularly important. Cong *et al.* [8] showed that one may be able to move a buffer a considerable distance from its ideal location while incurring a fairly small delay penalty. Their concept of feasible regions for buffer insertion is based on the principle that there is a wide range of reasonably good buffer locations.
2) At the interconnect-centric floorplanning stage, timing constraints are generally not available since macro block designs are incomplete and global routing and extraction have not been performed. Potentially crude timing analysis may not even be possible when there is too much incomplete information. Even when possible, the timing analysis results are often grossly pessimistic because interconnect synthesis has not taken place. At this stage, one needs to globally insert buffers while tracking wire congestion before the floorplan can even be evaluated. For example, in a design with a desired 5-ns clock period, say that one floorplan has a worst slack of -40 ns while a different floorplan has a worst slack of -43 ns. The designer cannot determine which floorplan is better because the slacks for both are so absurdly far from their targets. Buffer and wire planning must be efficiently performed first, then the design can be timed to provide a meaningful worst slack timing that the designer can use for evaluation. We envision performing buffer and wire planning each time the designer wants to evaluate a floorplan.

The first characteristic suggests that one does not need to worry about exactly where buffer sites are placed. The block designers should have the freedom to sprinkle buffer sites into their designs so that performance is not compromised; there just needs to be a sufficient number of buffer sites somewhere.

The optimization algorithm can view the thousands of buffer sites within a *tile graph*. Fig. 2(a) shows 68 buffer sites lying within the region of the chip. A tiling over the chip's area can be used to abstract each individual buffer site to a set of buffer sites lying at the center of each tile [Fig. 2(b)].[2] After a buffer is assigned to a particular tile, an actual buffer site can be allocated as a postprocessing step. The tile graph offers both a complexity reduction advantage (especially when there are thousands of buffer sites) and also the ability to manage routing congestion across tile boundaries. The granularity of the tiling depends on the desired accuracy/runtime tradeoff and on the current stage in the design flow.

The second characteristic suggests that timing constraints are not dependable in the early floorplanning stage. Our formulation relies on a global rule of thumb for the maximum distance between consecutive buffers. This rule of thumb was also used for buffer planning by Dragan *et al.* [10]. They note that for a high-end microprocessor design in 0.25-μm CMOS technology, repeaters are required at intervals of at most 4500 μm. Such a

[2]Note that several tiles have zero buffer sites. These might correspond to a cache, data path, or other critical part of the design for which buffer sites cannot be inserted. Having some zero buffer site tiles is not prohibitive; too many will obviously hinder solution quality.

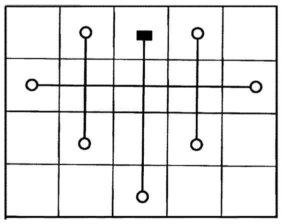


Fig. 3. Driver with seven sinks, whereby the maximum distance allowed between gates is three. With this interpretation of the distance rule, the driving gate must drive 11 units of wirelength.

rule is necessary to ensure that the slew rate is sufficiently sharp at the input to all gates.[3] Note that later in the design flow, when more accurate timing information is available, one can rip up the buffering solution for a given net and recompute a potentially better solution via a timing-driven buffering algorithm.

We represent a tiling by a graph $G(V,E)$ where V is the set of tiles, and edge $e_{u,v}$ is in E if u and v are neighboring tiles. Given a tile v, let $B(v)$ be the number of buffer sites within the tile. Let $N = \{n_1, n_2, \ldots, n_m\}$ be the set of global nets and let $W_{(e_{u,v})}$ be the maximum permissible number of wires that can cross between u and v without causing overflow. If $b(v)$ denotes the number of buffers assigned to v, the buffer congestion for v is given by $b(v)/B(v)$. Similarly, given a global routing of N, if $w(e_{u,v})$ denotes the number of wires which cross between tiles u and v, the wire congestion for edge $e_{u,v}$ is given by $w(e_{u,v})/W(e_{u,v})$.

For net n_i, let L_i be the maximum wire length (in units of tiles) that can be driven by either the driver of n_i or a buffer inserted on n_i. This interpretation of maximum distance avoids the scenario that could occur in Fig. 3. The figure shows a driver with seven sinks whereby the distance between the driver and each sink is three tile units. Using this interpretation of the distance constraint results in a legal solution where the source gate drives 11 tile units of wiring without requiring any buffers. For a slew-based distance rule, the extra interconnect (and sink load) may cause weak signals at the sinks. Thus, our rule requires the total length of interconnect that can be driven by any gate to be no more than L_i.

A. Problem Formulation

Given a tiling of the chip area $G(V,E)$, nets $N = \{n_1, n_2, \ldots, n_m\}$, the number of buffer sites $B(v)$, and tile length constraints L_i, assign buffers to nets such that the following apply.

- $b(v) \le B(v)$ for all $v \in V$, where $b(v)$ is the number of buffers assigned to tile v.
- Each net $n_i \in N$ satisfies its tile length constraint, L_i.[4]
- There exists a routing after buffering such that for all $e_{u,v} \in E$, the number of wires crossing from u to v is less than or equal to $W(e_{u,v})$.

A solution to this problem implies that constraints are satisfied, though one can also optimize secondary objectives, e.g., total wirelength, maximum and average wire congestion, maximum and average buffer congestion, and net delays. Our heuristic seeks a solution that satisfies the problem formulation while also trying to optimize these secondary objectives.

Note that the purpose of our formulation is not to find the final buffering and routing of the design. Rather, it can be used to estimate needed buffering and routing resources or as a precursor to timing analysis for more accurate floorplan evaluation. Once deeper into the physical design flow, nets which generate suboptimal performance or lie in timing-critical paths should be re-optimized using more accurate timing constraints and wire capacitances.

III. RABID Heuristic

The purpose of the RABID heuristic is to show how buffer and wire planning can be integrated into a tile-based global routing methodology. We follow a traditional rip-up and reroute type of strategy. RABID proceeds in four stages: i) initial Steiner tree construction; ii) wire congestion reduction; iii) buffer assignment; and iv) final post processing. The primary innovations are within stages 3 and 4 which handle buffer site assignment. Stages 1 and 2 deliver an initial congestion-aware global routing solution as a starting point. One could alternatively begin with the solution from any global router, e.g., the multicommodity flow-based approach of [1].

A. Stage 1: Initial Steiner Tree Construction

In this stage, we want a preliminary routing of each net so that congested regions can be evaluated and reduced in each stage. As opposed to a pure minimum length construction, the tree should be timing-driven, yet timing constraints are not necessarily available. Hence, we adopt the Prim–Dijkstra construction [4] which generates a hybrid between a minimum spanning tree and shortest path tree. The result is a spanning tree which trades off radius and wire length.[5]

Next, each spanning tree is then converted to a Steiner tree via a greedy overlap removal algorithm that iteratively searches for the two tree edges with the largest potential wire length overlap. A Steiner point is introduced to remove the overlap as shown in Fig. 4. The algorithm terminates when no further overlap removal is possible.

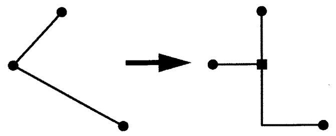


Fig. 4. Example of spanning tree edge overlap removal.

[3]A similar maximum distance rule was also used in the design of a recent high-performance IBM microprocessor with 170 million transistors [15]. A maximum distance between buffers was derived based on the desired input slew rate, and this rule was used to guide global buffer insertion. Global timing constraints were not used, nor were they even available.

[4]In general, one will use the same number of tiles L_i for each net. However, if some nets can be routed on higher metal layers while others cannot, different nets can have different L_i values depending on their layer. Also, a larger value of L_i can be used in conjunction with wider wire width assignment.

[5]The tradeoff requires a user parameter between zero and one, where a lower number emphasizes wirelength, while a higher number emphasizes path length. Our experiments use a value of 0.4.

B. Stage 2: Wire Congestion Reduction

The next step is to rip-up-and-reroute to reduce wire congestion. The tile graph $G(V, E)$ is constructed from the existing Steiner routes, and the congestion of each edge E in is computed. Instead of ripping up nets in congested regions, we rip-up and reroute every net, a technique similar in spirit to Nair's method [14]. This approach is less likely to become trapped in a local minima. The net ordering is first fixed (we sort in order of smallest to largest delays), and each net is processed in turn according to the ordering. The advantage is that even nets which do not violate congestion constraints can be improved to further reduce congestion so that other nets can be successfully rerouted in subsequent iterations. The algorithm terminates after either three complete iterations or $w(e_{u,v})/W(e_{u,v}) \leq 1$ holds for all $e_{u,v} \in E$. From experience, we observe only nominal potential improvement after the third iteration.

To reroute the net, the entire net is deleted and then rerouted using an approach similar to [5], as opposed to rerouting a single edge. The new tree is constructed on the tile graph using the same Prim–Dijkstra cost function in Stage 1, except that the cost for each edge is not its Manhattan distance. The routing occurs across the tile graph using the following congestion-based cost function:

$$\text{Cost}(e_{u,v}) = \left\{ \begin{array}{ll} \frac{w(e_{u,v})+1}{W(e_{u,v})-w(e_{u,v})}, & \text{if } \frac{w(e_{u,v})}{W(e_{u,v})} < 1 \\ \infty, & \text{otherwise} \end{array} \right\}. \quad (1)$$

The cost is the number of wires that will be crossing $e_{u,v}$ divided by the number of wires still available. The purpose of this cost is to have the penalty become increasingly high as the edge comes closer to full capacity. We perform a wavefront expansion from the tile which contains the source, updating the lowest tile cost with each expansion. When each sink in the net is reached, the algorithm terminates, and the tree subsequently recovered by tracing back the edges on the path from each sink to the source.

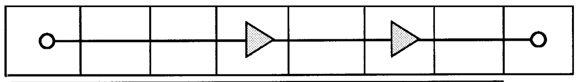
C. Stage 3: Buffer Assignment

Once a low-congestion routing exists, the next step assigns buffer sites to each net. We perform this assignment iteratively, starting with the net with highest delay. Before buffers are assigned, we first estimate the probability of a net occupying a tile. For a net n_i passing through tile v, the probability of a buffer from v being inserted onto n_i is defined as $1/L_i$. Let $p(v)$ be the sum of these probabilities for tile v over all unprocessed nets. Recall that $B(v)$ is the number of buffer sites in v and $b(v)$ is the current number of used buffer sites. We define the cost $q(v)$ for using a particular buffer site as

$$q(v) = \left\{ \begin{array}{ll} \frac{b(v)+p(v)+1}{B(v)-b(v)}, & \text{if } \frac{b(v)}{B(v)} < 1 \\ \infty, & \text{otherwise} \end{array} \right\}. \quad (2)$$

Observe the similarity between (2) and (1). Both significantly increase the penalty as resources become more contentious.

Fig. 5 shows an example of how the buffer cost is computed. Note that the $p(v)$ values do not include the currently processed net. The cost $q(v)$ is computed for each tile, and $q(v)$ is included in the cost for a net if a buffer is inserted at v. In the example, if $L_i = 3$, the minimum cost solution has buffers in the third and fifth tiles with cost $0.5 + 1.0 = 1.5$.



B(v)	8	5	12	3	5	0
b(v)	3	4	2	3	0	0
p(v)	2.5	3.6	2	0.8	4	5
q(v)	1.3	8.6	0.5	∞	1.0	∞

Fig. 5. Example of how buffer costs are computed. For a value of $L_i = 3$, the optimal solution is shown having a total cost of 1.5.

Single-Sink Buffer Insertion Algorithm

```
1. Set $C_t[j] = 0$ for $1 \leq j < L_i$ where $t$ is the single-sink. Set $v = t$.
2. while $v \neq s$ do
       for $j = 1$ to $L_i - 1$ do
           Set $C_{par(v)}[j] = C_v[j-1]$
       Set $C_{par(v)}[0] = q(par(v)) + min\{C_v[j] \| 0 \leq j < L_i\}$
       Set $v = par(v)$.
3. Let $v$ be such that $par(v) = s$. Return $min\{C_v[j] \| 0 \leq j < L_i\}$.
```

Fig. 6. Single-sink buffer insertion algorithm.

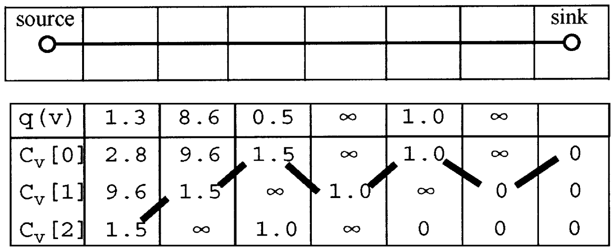
An optimal solution can be found in linear time in terms of the number of tiles spanned by the net (assuming that L_i is constant). The approach uses a van Ginneken [18] style dynamic programming algorithm, but has lower time complexity because the number of candidates for each node is, at most, L_i.

We begin with the simple case of a net n_i with a single source s and sink t. Let $par(v)$ be the parent node of tile v in the route, and assume that $q(v)$ has been computed for all nodes on the path between s and t. At each node v, the array C_v stores the cost of the solutions from v to t. The index of the array determines the distance downstream from v to the last buffer inserted. Thus, the array is indexed from 0 to $L_i - 1$, since v cannot be at distance more than L_i from the last available buffer. The full algorithm is shown in Fig. 6.

Step 1 initializes the cost array C_t to zero for the sink t. The algorithm then traverses up toward the source, iteratively setting the values for the cost array. Step 2 computes the values for $\text{par}(v)$ given the values for v. The value of $C_{\text{par(v)}}[j]$ for $j > 0$ is simply $C_v[j-1]$ since no buffer is being inserted at v for this case. If a buffer is to be inserted at $\text{par}(v)$, then the cost $C_{\text{par}(v)}[0]$ is computed by adding the current cost for insertion, $q(\text{par}(v))$, to the lowest cost seen at v. One can recover the solution by storing $\text{par}(v)$ at the index in C_v which was used to generate the solution.

Fig. 7 shows how the cost array is computed for the two-pin example in Fig. 5 (with $L_i = 3$), and the dark lines show how to trace back the solution. Observe from the table that costs are shifted down and to the left as one moves from right to left, with the exception of entries with index zero.

The algorithm is optimal since each possible solution is preserved during the execution. One can exploit the fact that the number of possible candidates at each node is no more than L_i to give a space and time complexity of $O(nL_i)$, where n is the number of tiles spanned by the net. This is a significant advantage over similar dynamic programming approaches [12], [18], [20] which have at least $O(n^2)$ time complexity. Of course,



q(v)	1.3	8.6	0.5	∞	1.0	∞	
C_v[0]	2.8	9.6	1.5	∞	1.0	∞	0
C_v[1]	9.6	1.5	∞	1.0	∞	0	0
C_v[2]	1.5	∞	1.0	∞	0	0	0

Fig. 7. Execution of the single source algorithm on the example in Fig. 5. The optimal solution has cost 1.5; the dark lines show how this cost is obtained.

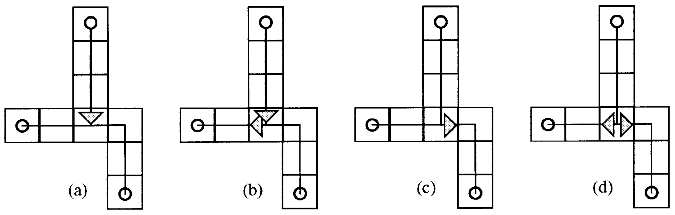


Fig. 8. For a node with two children, buffers may be used to (a, b) drive both branches, (b) decouple the left branch, (c) decouple the right branch, or (d) decouple both branches.

the reason for the reduced time complexity is not a better algorithm; all these algorithms incorporate a similar dynamic programming framework. Rather, the complexity advantage is obtained by making different assumptions, such as equally spaced tiles and a length-based objective, that simplify the problem.

Extending the algorithm to multisink nets is fairly straightforward. One still keeps a cost array at each node, but updating the cost becomes a bit trickier when a node has multiple children. Let $W(v)$ be the set of children of v. We keep an intermediate cost array K_w where $w \in W(v)$ to represent the cost of buffering the next tile, and then we join all the children cost arrays together into C_v. For example, four of the eight possible cases for buffering with two children are shown in Fig. 8. Both (a) and (b) show a buffer driving both branches, (b) shows decoupling of the left branch, (c) shows decoupling of the right branch, and (d) shows decoupling both branches. Observe that in these cases, we allow multiple buffers to be inserted in the same tile, as shown in (b) and (d).

The complete algorithm is shown in Fig. 9. The algorithm flows from the sinks to the source in a similar manner as the single-sink algorithm in Fig. 7. For readability, we have broken the pseudocode into four smaller subroutines: AdvanceTile, BufferTile, JoinChildren, and BufferMultiChildren. AdvanceTile and BufferTile perform the same concept as in the single-sink algorithm, moving up to the next parent tile and buffering on that tile. For multisink nets, this buffering is considered for the decoupling branch, hence, it is performed for each child of v. JoinChildren then merges the intermediate K_v cost arrays into a single C_v array for tile v. Finally, if there is more than one child, we need to consider additional buffering to drive multiple branches. The BufferMultiChildren routine handles the case of Fig. 8(a) and (b).[6]

Multi-Sink Buffer Insertion Algorithm

AdvanceTile(w): for $j = 1$ to $L_i - 1$ do $K_w[j] = C_w[j-1]$

BufferTile(w, v): $K_w[0] = q(v) + min\{C_w[j] \,\|\, 0 \le j < L_i\}$

JoinChildren$(v, W(v))$:
for $j = 0$ to $L_i - 1$ do $C_v[j] = min\left\{\sum_{w \in W(v)} K_w[j_w] \,\|\, \sum_{w \in W(v)} j_w = j\right\}$

BufferMultiBranches$(v, W(v))$
$C_v[0] = min\left\{C_v[0], q(v) + min\left\{\sum_{w \in W(v)} K_w[j_w] \,\|\, \sum_{w \in W(v)} j_w \le L_i\right\}\right\}$

```
1. Pick any sink $v$.
   repeat $v \ne s$ do
2.    if $v$ is a sink then
         Set $C_v[j] = 0$ for $1 \le j < L_i$.
3.    Let $W(v)$ be the children of $v$
      for each $w \in W(v)$
         AdvanceTile($w$)
         BufferTile($w$)
4.    JoinChildren($v$, $W(v)$)
5.    if there are more than one children
         BufferMultiBranches($v$, $W(v)$).
6.    mark $v$ as visited
      pick unvisited node $v$ such that $w$ is visited for all $w \in W(v)$.
   until $v = s$
7. Return $min\{C_s[j] \| 0 \le j < L_i\}$.
```

Fig. 9. Algorithm for buffer insertion for multiple sinks.

If the net has m sinks, then Step 4 will be executed exactly $m - 1$ times; thus, Step 4 adds $O(mL_i^2)$ additional time complexity above and beyond that of the single-sink algorithm's $O(nL_i)$ complexity. Consequently, the multisink variation has $O(mL_i^2 + nL_i)$ time complexity. Typically, one would expect the second term to dominate, especially when the number of sinks is small. However, for nets with lots of sinks and with a large required tile separation, the first term could potentially dominate.

D. Stage 4: Final Post Processing

The final stage of RABID attempts to reduce buffer congestion, wire congestion, and the number of nets which, up until now, have failed to meet their length constraint. Using the same flow as in stage 2, each net is ripped up and rerouted, and the buffers for the net are removed as well. However, for multipin nets, the net is ripped up one *two-path* at a time, where a two-path is a path in the tree which begins and ends at either a Steiner node, source, or sink, and contains only vertices of degree two. The two ends of the two-path are then reconnected via the path that minimizes the sum of wire and buffer congestion costs [(1) and (2)].[7]

The minimum cost two-path is computed as follows. Call the endpoint of the original two-path that is in the same sub-tree as the source the *head* and the other endpoint the *tail*. The algorithm proceeds bottom-up in a manner similar to the single-sink buffer insertion algorithm (and also the buffer insertion, maze-routing algorithms of [12] and [20]). Starting from the tail, each neighbor of the current minimum-cost tile is visited, and the costarray is updated. For each element in the cost array, a pointer

[6]Fig. 9 can handle three children, but it does not allow a buffer to drive two of the three branches. Handling this case is straightforward but tedious; we omit it to preserve the readability of the algorithm.

[7]Note that these buffer and wire congestions costs are of the same order of magnitude, so we simply add their costs. Alternatively, one could use any linear combination of the two cost functions.

TABLE I
TEST CIRCUIT STATISTICS AND PARAMETERS FOR THE FIRST SET OF EXPERIMENTS

circuit	cells	nets	pads	sinks	grid size	tile area (mm^2)	L_i	buffer sites	%chip area
apte	9	77	73	141	30x33	0.36	6	1200	0.13
xerox	10	171	2	390	30x30	0.35	5	3000	0.38
hp	11	68	45	187	30x30	0.42	6	2350	0.25
ami33	33	112	43	324	33x30	0.46	5	2750	0.24
ami49	49	368	22	493	30x30	0.67	5	11450	0.75
playout	62	1294	192	1663	33x30	0.75	6	27550	1.47
ac3	27	200	75	409	30x30	0.49	6	3550	0.32
xc5	50	975	2	2149	30x30	0.54	6	13550	1.11
hc7	77	430	51	1318	30x30	1.04	5	7780	0.33
a9c3	147	1148	22	1526	30x30	1.08	5	12780	0.52

is maintained back to the tile which was used to generate that cost. The algorithm iteratively expands the tile with lowest cost and updates the costs of neighboring tiles during wavefront expansion. The cost for the new tile also includes the wire congestion cost of crossing the tile boundary. When the head of the two-path is reached, the minimum cost solution is recovered by tracing back out the path to the tail. The buffers for the entire net are then ripped out and reinserted as in Stage 3.

IV. EXPERIMENTAL RESULTS

We implemented RABID in C++ on an RS6000/595 machine with 1 GB of memory. We tested our code on ten benchmarks which we obtained from the authors of [8]. The first six are from the collaborative benchmarking laboratory (CBL), and the other four are randomly generated.[8] We also embed the designs in the same 0.18-μm technology used in [8]. The statistics of the benchmarks are summarized in the first five columns of Table I. The nets and sinks columns present slightly smaller values than in [8] to reflect the nets that Cong *et al.* did not optimize since they did not require buffers.

A. General Performance

Our first set of experiments studies the performance of each of RABID's four stages. The grid size and number of buffer sites used are presented in Table I. We chose the grid size to have 30 tiles on the shorter side of the chip, and then derived the number of tiles for the longest side, so that each tile was roughly square. The seventh column gives the area of each tile in square millimeters. The total number of buffer sites for each circuit were also chosen arbitrarily. The number was chosen to be large enough so that buffer congestion is low, but small enough so that the percent of the total chip area occupied by buffer sites is less than 2% of the total chip area. From the last column, observe that the percentage of the chip area is less than 1% for all but two cases. Except for the last two random circuits, no tile is more than one millimeter long on a side. For each benchmark, a random nine by nine set of tiles was prohibited from having any inserted buffer sites to correspond to a large cache-like object. The buffer sites were randomly distributed among the remaining tiles.

The floorplans were generated using the buffer-block planning code supplied by Cong *et al.* [8]. We generated a floorplan by first running buffer-block planning, then removing the inserted buffer blocks. This code performs its initial floorplanning using a Monte Carlo simulated annealing technique. The results for each stage and each test case are summarized in Table II. We present only the cumulative results for the four random circuits. The statistics presented are:

- the maximum and average wire congestion over all edges in the tile graph;
- the sum of the wiring overflows, i.e., the sum over all $e_{uv} \in E$ of $w(e_{uv}) - W(e_{uv})$, for whenever $w(e_{uv}) > W(e_{uv})$;
- the maximum and average buffer density;
- the number of buffers inserted;
- the number of nets for which the tile length constraint was not satisfied;
- total wirelength in millimeters;
- maximum and average delays in picoseconds to each sink;
- CPU time in seconds.

Note, that no timing constraints are used, so we use average and maximum source-to-sink delays to give an indication for the quality of timing.

We make several observations.

- The wire congestion constraint is always satisfied. If one ignores wire congestion, as is done in Stage 1, then the maximum wire congestion is typically a factor of two to three above capacity and there are several hundred overflows.
- The algorithm never violates the buffer site constraint, but typically utilizes at least one tile to full buffer capacity. As seen from the small average buffer densities in Table II, the total number of buffer sites chosen is actually quite small relative to total area. The number of buffers, failures, and wire length all decline from Stages 3 to 4 (except for the wire length and the number of buffers for apte), which shows that our final postprocessing step is quite effective. The number of nets which fail to meet the length constraint is typically small, but not zero. These violations are caused almost exclusively by the existence of the large nine by nine tiled region with no buffer sites that was inserted into each design.
- Net delays increase significantly from Stages 1 to 2 during the congestion avoidance rerouting stage, but the insertion of buffers in Stage 3, reduces delay significantly even though the buffer insertion algorithm is "delay ignorant." The maximum and average delay is always less after Stage 4 than Stage 1, except for the maximum delay of xerox.
- The CPU time is almost exclusively dominated by the two rerouting stages: 2 and 4. Thus, our buffer insertion algorithm in Stage 3 is efficient in practice.

[8]The fifth random benchmark from [8], called pc2, has some internal problems and is no longer in use.

TABLE II
STAGE BY STAGE EXPERIMENTAL RESULTS FOR THE SIX CBL CIRCUITS. THE FINAL RESULTS ARE SHOWN FOR THE LAST FOUR RANDOM CIRCUITS

circuit	Stage	Wire Congest		Over-	Buffer Density		#bufs	#fails	wire	delay to sink		CPU
		max	avg	flows	max	avg			length	max	avg	(s)
apte	1	2.00	0.15	225	0.00	0.00	0	77	1410	5029	1700	0
	2	0.62	0.18	0	0.00	0.00	0	77	1706	5390	2156	11
	3	0.62	0.18	0	1.00	0.43	483	6	1706	1632	796	1
	4	0.75	0.18	0	1.00	0.47	529	4	1761	1688	828	37
xerox	1	2.00	0.16	466	0.00	0.00	0	171	2537	3361	1529	0
	2	0.80	0.22	0	0.00	0.00	0	171	3449	5836	2010	27
	3	0.80	0.22	0	1.00	0.41	1179	8	3449	3216	1289	2
	4	0.60	0.19	0	1.00	0.41	1119	4	3054	3622	1181	113
hp	1	3.25	0.31	368	0.00	0.00	0	68	1405	5672	1995	0
	2	1.00	0.39	0	0.00	0.00	0	68	1818	6723	2440	13
	3	1.00	0.39	0	1.00	0.22	525	7	1818	1922	767	0
	4	0.75	0.36	0	1.00	0.20	479	2	1703	1713	752	61
ami33	1	2.50	0.31	365	0.00	0.00	0	112	2471	9016	4413	0
	2	1.00	0.38	0	0.00	0.00	0	112	3028	12735	5690	27
	3	1.00	0.38	0	1.00	0.37	968	3	3028	2242	1043	1
	4	1.00	0.37	0	1.00	0.38	961	0	2958	2265	1050	81
ami49	1	2.18	0.37	887	0.00	0.00	0	368	5881	7601	1730	0
	2	1.00	0.54	0	0.00	0.00	0	368	8720	28784	3200	45
	3	1.00	0.54	0	1.00	0.20	2310	15	8720	4416	997	2
	4	1.00	0.43	0	0.86	0.16	1739	4	7019	2547	820	111
playout	1	2.38	0.22	2333	0.00	0.00	0	1294	22555	8633	1989	1
	2	0.67	0.29	0	0.00	0.00	0	1294	30177	22977	2915	221
	3	0.67	0.29	0	1.00	0.21	5711	136	30177	3682	1033	4
	4	0.70	0.25	0	0.87	0.19	5076	34	26168	3032	900	453
ac3	1-4	0.62	0.31	0	1.00	0.35	1148	15	4659	2030	828	130
xc5	1-4	0.86	0.44	0	1.00	0.28	3649	16	17078	10165	1533	542
hc7	1-4	1.00	0.52	0	1.00	0.36	2702	28	13499	4062	1002	257
a9c3	1-4	1.00	0.56	0	1.00	0.45	5601	27	28990	3077	1110	484

B. *Variations*

Our next set of experiments examines the behavior of RABID when the number of available buffer sites varies. We ran our algorithm on each of the six CBL circuits three times using small, medium, and large numbers of buffer sites that are randomly distributed (with the blocked nine by nine region). The other parameters are the same as in the previous set of experiments. Results are summarized in Table III. The three lines for each circuit correspond to small, medium, and large numbers of buffer sites. The number of buffer sites was chosen somewhat arbitrarily so that significant differences would result between each row in the table. For example, if increasing the number of buffer sites by 50% did not yield much improvement, we would subsequently increase that percentage until significant improvement resulted.

Observe that as the number of buffer sites decreases, more nets fail to meet their length constraint. Most notably, as the number of buffer sites increases, the maximum and average net delays decrease significantly. Having no more than one in every five buffer sites occupied appears necessary to obtain good solutions.

In our next set of experiments, we keep the number of buffer sites constant, but vary the size of the grid. The results are summarized in Table IV for a sampling of three of the test cases. Observe that the maximum wire congestion increases with the number of tiles. A finer-grained tiling implies tighter wire congestion, e.g., dividing a tile into four equal-sized tiles increases the number of congestion constraints by a factor of three. The increased wire congestion may cause an increase in the maximum delay because of long detours, though the average congestion can stay the same. However, a finer tiling means one can design a length constraint that is more appropriate, e.g., for a 10×11 grid one might need a length constraint of two, which is rather coarse. For a 50×55 grid, a length constraint of perhaps eight might yield results with the best delay characteristics.

The finer-grained tilings give better insight into the quality of the floorplan. A coarser tiling can indicate that the design is

TABLE III
SUMMARY OF RESULTS WHEN THE NUMBER OF AVAILABLE BUFFER SITES VARIES

circuit	Buffer sites	Wire Congest max	Wire Congest avg	Over-flows	Buffer Congest. max	Buffer Congest. avg	#bufs	#fails	wire length	delay to sink max	delay to sink avg	CPU (s)
apte	280	1.00	0.18	0	1.00	0.89	251	53	1715	2093	1007	58
	700	0.75	0.19	0	1.00	0.71	486	19	1848	3023	986	49
	3200	0.62	0.18	0	1.00	0.16	505	1	1685	1566	781	49
xerox	600	1.00	0.20	0	1.00	0.90	539	98	3138	3812	1402	97
	1300	1.00	0.21	0	1.00	0.81	1006	47	3307	2260	1018	92
	3000	0.60	0.19	0	1.00	0.41	1119	4	3054	3622	1181	99
hp	300	1.00	0.39	0	1.00	0.84	252	41	1822	2588	1110	51
	600	1.00	0.41	0	1.00	0.73	445	24	1904	2108	928	55
	2350	1.00	0.36	0	1.00	0.20	479	2	1703	1713	752	50
ami33	500	1.00	0.36	0	1.00	0.88	444	73	2954	3714	1568	83
	850	1.00	0.38	0	1.00	0.85	712	45	3071	3509	1309	82
	2750	1.00	0.37	0	1.00	0.38	961	0	2958	2265	1050	85
ami49	850	1.09	0.44	7	1.00	0.80	677	243	7067	3440	1208	157
	1650	1.00	0.45	0	1.00	0.68	1143	182	7312	2765	1037	156
	11450	1.00	0.43	0	0.86	0.16	1739	4	7019	2547	820	160
playout	3250	1.00	0.27	0	1.00	0.91	2938	776	27472	4382	1266	635
	6250	0.97	0.28	0	1.00	0.83	5089	314	29001	3575	1093	615
	27550	0.70	0.25	0	0.87	0.19	5076	34	26168	3032	900	727

TABLE IV
RABID RESULTS WITH VARYING GRID SIZES FOR THREE CBL BENCHMARKS

circuit	grid size	Wire Congest max	Wire Congest avg	Over-flows	Buffer Congest. max	Buffer Congest. avg	#bufs	#fails	wire length	delay to sink max	delay to sink avg	CPU (s)
apte	10 x 11	0.40	0.17	0	0.89	0.35	393	1	1637	1709	821	5
	20 x 22	0.46	0.17	0	1.00	0.47	510	6	1775	1987	830	21
	30 x 33	0.78	0.16	0	1.00	0.57	646	9	1754	1710	840	48
	40 x 44	0.75	0.14	0	1.00	0.53	602	17	1785	1789	808	98
	50 x 55	0.80	0.17	0	1.00	0.48	554	13	1719	1901	813	212
ami49	10 x 10	0.73	0.35	0	0.24	0.10	1082	3	6799	2919	855	16
	20 x 20	0.85	0.36	0	0.47	0.12	1341	11	6952	2868	828	70
	30 x 30	0.92	0.36	0	0.82	0.15	1700	11	6878	2449	795	157
	40 x 40	1.00	0.35	0	1.00	0.15	1643	5	6868	2870	793	311
	50 x 50	1.00	0.35	0	1.00	0.14	1633	6	6899	2331	784	568
playout	11 x 10	0.57	0.25	0	0.36	0.14	3823	17	25771	6621	1004	55
	22 x 20	0.83	0.26	0	0.86	0.17	4719	50	26502	3182	923	257
	33 x 30	0.73	0.26	0	0.87	0.23	6184	40	26669	3101	894	620
	44 x 40	0.76	0.25	0	1.00	0.21	5755	41	25922	2751	867	1268
	55 x 50	1.00	0.26	0	1.00	0.22	5847	30	26320	2667	876	2307

fairly easily routable, but a finer tiling can better highlight areas of potential congestion. Thus, if one wants to use our algorithm to evaluate the quality of a particular floorplan, a finer-grained tiling is likely more useful for wire congestion evaluation.

Finally, we observe that the CPU times roughly increase at a rate slightly higher than linear with the number of tiles.

C. Comparing RABID With Buffer-Block Planning

Our final experiments attempt to compare with previous work on buffer-block planning, yet we are not using buffer blocks. Hence, it is not feasible to simply compare with previously published results. We needed to run the code ourselves and implement routines to gather statistics from the data. For this comparison, Cong *et al.* [8] supplied us with the source code to their algorithm BBP/FR. Although other buffer-block planning results exists (e.g., the work of [16] creates more buffer blocks to reduce wire congestion), we believe this comparison is sufficient to show that our proposed methodology can deliver timing solutions that are competitive with the buffer-block planning methodology while better managing buffer and wire congestion.

As in [8], but unlike our previous experiments, we decomposed multipin net into several two-pin nets. Our results were generated using randomly distributed buffer sites that altogether occupy less than 2% of the total chip area. Cong *et al.* [8] report timing results by measuring the number of nets which fail to meet their delay constraints. The timing constraint was chosen to be between 1.05–1.20 of the optimum achievable delay. We believe this constraint generation is unrealistic since real timing constraints are path-based, and this implies that all constraints are tight, yet potentially satisfiable. In a practical situation, some of the $1.05x$–$1.20x$ timing constraints will be so tight that buffer insertion is insufficient to satisfy timing, e.g., a constraint of $0.95x$. For these cases, feasible regions are not well defined and, so, BBP does not have a mechanism to come as close as possible to satisfying the constraint (which has value to the designer). Also, other constraints may be so loose that no buffer insertion is required or many detours can be taken to still meet delay constraints. Since RABID and BBP/FR insert buffers on all nets which require them, we use maximum and average sink delay to quantify timing performance.

In addition to the previous statistics, we also measured maximum tile area percentage (MTAP). A percentage of the area of each tile is occupied by buffers; MTAP denotes the maximum such percentage over all tiles. We used the same tiling for RABID (see Table I) to measure the MTAP for BBP/FR. The minimum wire length is the sum of the minimum possible wire length routing of all the nets. This enables one to see how much additional wire length above the minimum was actually required.

Table V presents the comparisons with BBP/FR. Nevertheless, some differences between the approaches are readily seen.

- RABID always meets the congestion constraints while BBP/FR does not. The results presented here even include a postprocessing step (applied to both RABID and BBP/FR) which tries to minimize congestion for the current buffering solution without increasing wire length. Note that virtually all of the CPU time reported for BBP/FR is due to this step. The original BBP/FR code runs in 2 s or less for all the test cases.
- RABID inserts significantly more buffers, due to a tight length constraint and also due to the rerouting to avoid wire congestion.
- Because the RABID methodology invites spreading, the MTAP is significantly less. In the worst case, BBP/FR has one tile with 18.2% of its area devoted to buffers. This percentage climbs to at most 1.1% for our approach.
- The CPU time for BBP/FR is negligible. Stages 2 and 4 of RABID require much larger run times, but they are clearly not prohibitive.
- The delays for RABID are quite comparable to those of BBP/FR despite using a length-based buffer insertion algorithm and satisfying wire congestion constraints. In some cases the delays are even better. Delays can sometimes be worse because timing is not directly optimized and due to wire detours.

Thus, RABID succeeds at avoiding wire congestion and buffer density where BBP/FR cannot. This is not a shortcoming of BBP/FR per se, but rather the block-based methodology in which it is embedded.

TABLE V
COMPARISONS OF RABID TO BBP/FR [8].

circuit	Algorithm	Wire Congest		Over-flows	#bufs	MTAP	wire length	delay to sink		CPU (s)
		max	avg					max	avg	
apte	BBP/FR	2.00	0.12	23	233	2.87	1827	2026	721	14
	RABID	1.00	0.13	0	417	0.33	2010	1935	787	95
xerox	BBP/FR	1.15	0.10	14	508	5.57	4096	1486	575	29
	RABID	0.93	0.11	0	957	0.57	4541	1531	643	167
hp	BBP/FR	2.50	0.16	107	264	1.89	2194	1948	645	16
	RABID	0.83	0.17	0	450	0.28	2403	2029	707	67
ami33	BBP/FR	1.19	0.12	34	654	3.31	4923	2329	852	44
	RABID	0.69	0.12	0	1150	0.44	5232	2256	900	138
ami49	BBP/FR	4.64	0.34	1034	862	4.15	6787	2359	768	65
	RABID	0.93	0.37	0	1339	0.36	7592	2635	859	167
playout	BBP/FR	0.99	0.13	0	3422	10.34	25930	2727	880	198
	RABID	0.45	0.13	0	3840	0.64	27601	3310	947	813
ac3	BBP/FR	1.23	0.17	37	718	2.86	5586	1928	763	87
	RABID	0.58	0.18	0	1037	0.33	5954	2095	807	208
xc5	BBP/FR	4.70	0.39	3528	3186	16.40	25241	2194	655	181
	RABID	0.84	0.41	0	4410	0.81	27060	2343	700	694
hc7	BBP/FR	3.82	0.40	1363	2684	4.88	20011	2935	861	174
	RABID	0.82	0.42	0	2983	0.35	21523	3349	941	386
a9c3	BBP/FR	2.54	0.31	1329	4041	4.79	29060	2726	1093	222
	RABID	0.60	0.32	0	4225	0.44	30723	2786	1170	502

V. CONCLUSION

We have proposed an alternative methodology for buffer and wire planning that uses pre-allocated buffer sites that are distributed throughout the design. This methodology enables one to model this planning problem via a tile graph and simultaneously plan both wires and buffers. Our four-stage heuristic includes an efficient algorithm for length-based buffer insertion and also a technique for simultaneous optimization of buffer and wire congestion. Our experimental results assert that this approach can generate effective solutions in a reasonable amount of time.

Our future work seeks to integrate an industrial tile graph-based global router into Stages 1 and 2 of our heuristic. This should result in both better routing solutions and higher correlation with running the complete design flow. Ultimately, our objective is to use this tool for early and accurate floorplan evaluation, which makes strong correlation with the final routing result a necessity.

ACKNOWLEDGMENT

The authors are extremely grateful to J. Cong, T. Kong, and D. Pan for supplying the BBP/FR code and benchmarks. They also helped with running the codes, answering questions, and supplying Fig. 1.

REFERENCES

[1] C. Albrecht, “Provably good global routing by a new approximation algorithm for multicommodity flow,” in *Proc. Int. Symp. Physical Design*, 2000, pp. 19–25.
[2] C. J. Alpert and A. Devgan, “Wire segmenting for improved buffer insertion,” *Proc. 34th IEEE/ACM Design Automation Conf.*, pp. 588–593, 1997.

[3] C. J. Alpert, A. Devgan, and S. T. Quay, "Buffer insertion for noise and delay optimization," *Proc. 35th IEEE/ACM Design Automation Conf.*, pp. 362–367, 1998.

[4] C. J. Alpert, T. C. Hu, J. H. Huang, A. B. Kahng, and D. Karger, "Prim-Dijkstra tradeoffs for improved performance-driven routing tree design," *IEEE Trans. Computer-Aided Design*, vol. 14, pp. 890–896, July 1995.

[5] C. Chiang, M. Sarrafzadeh, and C. K. Wong, "A powerful global router based on steiner min-max trees," in *Proc. IEEE/ACM Int. Conf. Computer-Aided Design*, 1989, pp. 2–5.

[6] J. Cong, "Challenges and opportunities for design innovations in nanometer technologies," SRC Design Sciences Concept Paper, 1997.

[7] ——, "An interconnect-centric design flow for nanometer technologies," in *Proc. Int. Symp. VLSI Technol. Syst. Applicat.*, Taipei, Taiwan, June 1999, pp. 54–57.

[8] J. Cong, T. Kong, and D. Z. Pan, "Buffer block planning for interconnect-driven floorplanning," in *Proc. IEEE/ACM Int. Conf. Computer-Aided Design*, 1999, pp. 358–363.

[9] ——, "Buffer block planning for interconnect planning and prediction," *IEEE Trans.VLSI Syst.*, vol. 9, pp. 928–937, Dec. 2001.

[10] F. F. Dragan, A. B. Kahng, I. Mandoiu, and S. Muddu, "Provably good global buffering using an available buffer block plan," in *Proc. IEEE/ACM Int. Conf. Computer-Aided Design*, 2000, pp. 104–109.

[11] ——, "Provably good global buffering by multiterminal multicommodity flow approximation," in *Proc. Asia South Pacific Design Automation Conf.*, 2001, pp. 120–125.

[12] S.-W. Hur, A. Jagannathan, and J. Lillis, "Timing driven maze routing," in *Proc. Int. Symp. Physical Design*, 1999, pp. 208–213.

[13] J. Lillis, C.-K. Cheng, and T.-T. Y. Lin, "Optimal wire sizing and buffer insertion for low power and a generalized delay model," *IEEE J. Solid-State Circuits*, vol. 31, pp. 437–447, Mar. 1996.

[14] R. Nair, "A simple yet effective technique for global wiring," *IEEE Trans. Computer-Aided Design*, vol. CAD-6, pp. 165–172, Feb. 1987.

[15] S. Quay, private communication, 2000.

[16] P. Sarkar, V. Sundararaman, and C.-K. Koh, "Routability-driver repeater block planning for interconnect-centric floorplanning," in *Proc. Int. Symp. Physical Design*, 2000, pp. 186–191.

[17] X. Tang and D. F. Wong, "Planning buffer locations by network flows," in *Proc. Int. Symp. Physical Design*, 2000, pp. 180–185.

[18] L. P. P. P. van Ginneken, "Buffer placement in distributed RC-tree networks for minimal elmore delay," in *Proc. Int. Symp. Circuits Syst.*, 1990, pp. 865–868.

[19] P. Villarrubia, G. Nusbaum, R. Masleid, and P. T. Patel, "IBM RISC chip design methodology," in *Proc. Int. Conf. Computer Design*, 1989, pp. 143–147.

[20] H. Zhou, D. F. Wong, I. Liu, and A. Aziz, "Simultaneous routing and buffer insertion with restrictions on buffer locations," in *Proc. 36th IEEE/ACM Design Automation Conf.*, 1999, pp. 96–99.

Charles J. Alpert (S'92–M'96–SM'02) received the B.S. and B.A. degrees from Stanford University, Stanford, CA, in 1991, and the Ph.D. degree in computer science from the University of California, Los Angeles, in 1996.

He currently performs research in the physical design space as a Research Staff Member at the IBM Austin Research Laboratory, Austin, TX.

Dr. Alpert has served as Technical Program Chair for the International Symposium on Physical Design in 2003 and the Tau Workshop on Timing Issues in the Synthesis and Specification of Digital Systems. He has also served on the technical program committees for the IEEE/ACM Design Automation Conference and International Conference on Computer-Aided Design. He received a Best Paper Award at the 1994, 1995, and 2001 ACM/IEEE Design Automation Conferences and was awarded the Semiconductor Research Corporation's Mahboob Khan Mentor Award in 2001.

Jiang Hu received the B.S. degree in optical engineering from Zhejiang University, Hangzhou, China, in 1990, the M.S. degree in physics from the University of Minnesota, Duluth, in 1997, and the Ph.D. degree in electrical engineering from the University of Minnesota, Minneapolis, in 2001.

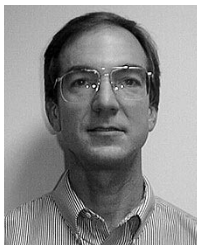
He was with IBM Electronics Design Automation, Austin, TX, from January 2001 to June 2002. Currently, he is an Assistant Professor in the Department of Electrical Engineering, Texas A&M University, College Station. His research interest is on VLSI physical design automation, especially on interconnect routing, optimization, and clock network synthesis.

Dr. Hu received a best paper award at the Design Automation Conference in 2001.

Sachin S. Sapatnekar (S'86–M'93–SM'99–F'03) received the B.Tech. degree from the Indian Institute of Technology, Bombay, India, in 1987, the M.S. degree from Syracuse University, Syracuse, NY, in 1989, and the Ph.D. degree from the University of Illinois, Urbana–Champaign, in 1992.

From 1992 to 1997, he was an Assistant Professor in the Department of Electrical and Computer Engineering, Iowa State University, Ames. He is currently a Professor in the Department of Electrical and Computer Engineering at the University of Minnesota, Minneapolis. He has coauthored two books and coedited one book in the areas of timing and layout optimization.

Prof. Sapatnekar has been an Associate Editor for the IEEE TRANSACTIONS ON VERY LARGE SCALE INTEGRATION (VLSI) SYSTEMS, IEEE TRANSACTIONS ON COMPUTER-AIDED DESIGN OF INTEGRATED CIRCUITS AND SYSTEMS, and IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS—PART II: ANALOG AND DIGITAL SIGNAL PROCESSING, and has served on the Technical Program Committee for various conferences, as Technical Program and General Chair for the Tau workshop, and the International Symposium on Physical Design. He is currently a Distinguished Visitor for the IEEE Computer Society and a Distinguished Lecturer for the IEEE Circuits and Systems Society. He is a recipient of the National Science Foundation Career Award and best paper awards at the 1997 and 2001 Design Automation Conferences and the 1998 International Conference on Computer Design.



Paul G. Villarrubia received the B.S. degree in electrical engineering from Louisiana State University, Baton Rouge, in 1981, and the M.S. degree from the University of Texas, Austin, in 1988.

He is currently a Senior Technical Staff Member with the IBM Corporation, Austin, TX, where he leads the development of placement and timing closure tools. He has also worked at IBM in the areas of physical design of microprocessors, physical design tools development, and tools development for ASIC timing closure. His interests include placement, synthesis, buffering, signal integrity, and extraction.

Mr. Villarrubia won a best paper award at the 2001 ACM/IEEE Design Automation Conference.