BEAR-FP: A Robust Framework for Floorplanning

Massoud Pedram Ernest S. Kuh

Abstract

This paper presents a hierarchical floorplanning approach for macrocell layouts which is based on the bottom-up clustering, shape function computation, and top down floorplan optimization with integrated global routing and pin assignment. This approach provides means for specifying and techniques for satisfying a wide range of constraints (physical, topological, timing) and is, therefore, able to generate floorplans for a number of different layout styles. A systematic and efficient optimization procedure during the selection of suitable floorplan patterns that integrates floorplanning, global routing and pin assignment, a new pin assignment technique based on linear assignment and driven by the global routing solution and floorplan topology, and an effective timing-driven floorplanning scheme are among the other novel features of the floorplanner. These techniques have been incorporated in BEAR-FP, a macro-cell layout system developed at the University of California, Berkeley. Results on various placement and floorplanning benchmarks are quite good.

[1]This research was sponsored by the National Science Foundation, under Grant No. MIP 88-03711 and by the Semiconductor Research Corporation, under Grant No. 91-DC-008.

Massoud Pedram is with the Department of Electrical Engineering – Systems, University of Southern California, Los Angeles, CA 90089.

Ernest S. Kuh is with the Department of Electrical Engineering and Computer Sciences, University of California, Berkeley, CA 94720.

1 Introduction

Macro-cell layout style is often used when a complex circuit must have minimum area and high performance. This layout style has advantages in terms of both area and performance over other cell-based layout styles with regular structures (e. g., standard cell and gate array/sea-of-gates). The price to pay is higher design cost and longer design time. The macro-cell design aid tools are not as mature as standard cell or gate array tools, and hence, macro-cell layout requires more time and effort on part of the chip designers. The macro-cell layout, however, uses more flexible topologies and cells, hence allowing very good area and performance optimizations. In addition, this layout style is more suited to a hierarchical design environment since it relies mainly on interface description of the cells, and therefore, permits higher degrees of design modularity and concurrent group work. The cells to be laid out in a macro-cell assembly can each be optimized independently based on the functionality and requirements of the cell. Individual cells can be implemented by the most appropriate layout style.

During the early stages in the design of electronic systems, decisions are made which have a dramatic effect on the quality (performance, density or area) of the resulting design. Choices must be made in partitioning functions into macro cells and in choosing interface characteristics of the cells such as size, shape, and pin positions. These choices are difficult because their effects on the circuit area and speed may not become known until much later. Floorplanning helps solve these problems.

The objective of floorplanning is to trade off cell interface characteristics (size, shape, and pin position) and cell locations to optimize the layout. Floorplanning is the first step in the physical design which determines the spatial and interface characteristics of given cells such that the desired physical and electrical constraints are satisfied. The floorplanner must generate a chip plan that can be implemented by cell generators. Such a floorplanner would allow better automation of the physical design process.

Floorplanning is useful in yet another way. One must know something about how a design is to be laid out in order to get an accurate estimate of its area and speed. The commonly used method for evaluating a register-transfer level design, namely the number of functional blocks and registers and the number of control steps, is no longer adequate. That is why VLSI designers generally work from a floorplan, even when they are developing the basic system architecture.

Most ASIC vendors provide pre-layout interconnect delay estimations based on fanout and gate count, using statistical data from previous layouts. Since the lengths of wires are unknown, these pre-layout delay estimates are not accurate. One can rely on floorplanning tools to attain more accurate estimates of interconnect delay. These tools allow designers to place large macros, manipulate their size, aspect ratio, and pin positions and receive feedback on routing density and wire lengths. Based on this data, delay

estimators can produce an estimated distributed RC delay. In addition to helping designers predict interconnect delays, floorplanning can provide valuable information for layout designs to reduce the number of placement and routing iterations needed.

Floorplanning tools can be used in the feedback loop of a high-level synthesis system to improve scheduling and allocation. First, a schedule and register-transfer data path are constructed. Then, floorplanning and global routing are performed which produce information about critical paths. This information is back-annotated into the circuit and high-level synthesis is repeated.

Floorplanning algorithms should model the cells' interface flexibility and any constraints on that flexibility. Three classes of cells are used in floorplanning: 1) Some cells are already laid out and are stored in a library. All the interface characteristics of these cells are known and fixed. To provide some flexibility, several versions of cells with different characteristics may be stored in a library. 2) The designs of some cells are known, but their layouts are flexible and can be influenced by the results of floorplanning. For example, standard or general cell layout methods can produce a wide range of shapes for a given design. PLAs and memory cells can be distorted through folding or layout design. 3) Cells of the third class are flexible because their designs (and perhaps even the design methods) are not known or are uncertain. In this case it is difficult for the designers or algorithms to even specify nominal interface characteristics or constraints thereon.

An important aspect of cell modeling is estimation of area and shape. This task often requires appropriate modeling of corresponding cell generators so that cells' shape functions can be accurately and quickly estimated. The floorplanner must exploit these functions in order to trade off the sizes and shapes of the leaf cells against each other to optimize the layout.

We have developed a floorplanning procedure which is based on the bottom-up clustering, shape function computation, and top down floorplan optimization with integrated global routing and pin assignment. An important feature of our floorplanner is its ability to accept various constraints and design requirements. This is in contrast with most existing floorplanners. Since in the past, specialized conventional floorplanners have been required to handle data-path dominated assemblies, mixed macrocell and standard-cell circuits, and macrocell designs. The main difference among these layout styles is the type of constraints that must be satisfied and optimization procedures that must be applied. Our floorplanner, however, provides means for specifying and techniques for satisfying a wide range of constraints (physical, topological, timing) and is, therefore, able to handle all these layout styles.

In addition to above, the floorplanner introduces the following new components:

- Accurate and dynamic routing area estimation during floorplan computation in

order to avoid an increase in the chip area after global routing.

- A systematic optimization procedure during the selection of suitable floorplan patterns that integrates floorplanning, global routing and pin assignment.
- Extensions to incorporate timing issues by a novel timing-driven clustering technique, followed by top-down floorplan optimization.
- A new pin assignment technique based on linear assignment and driven by the global routing solution and floorplan topology.
- Tight coupling between floorplanner and module generators through modeling important classes of generators, i.e., standard cell and data path assemblies.
- Accommodation of channel-free style layout which is especially useful when floorplanning large channelless gate arrays or mixed macrocell and standard-cell assemblies.

2 Floorplanning

2.1 Prior Work

There are two major thrusts in floorplanning research. The first thrust uses floorplanning in the initial stages of design to develop constraints that can be passed to succeeding synthesis steps. The second thrust relies on the existence of powerful cell generators that can implement cells according to specifications.

Lauther's min-cut placement [20] is a good example of the first thrust. He uses a top-down method that divides cells into two partitions (blocks). The process of dividing blocks into smaller blocks continues until the number of cells per block is small. This approach considers higher level of abstraction before it considers more detailed levels and is *goal-oriented*. Placement decisions made at higher levels of the hierarchy, however, may suffer from lack of detailed information. La Potin [31] improved the min-cut method by considering the pad positions. Dai [11] extended La Potin's work to general non-slicing structures and multi-way cluster trees and combined floorplanning with hierarchical global routing.

Otten's shape propagation placement technique [26] is an example of the second thrust. His algorithm initially derives a physical hierarchy in the form of a binary slicing tree. Next, the slicing tree is traversed bottom-up. At each internal node of the tree, a composite shape function is calculated from the shape functions of its children (child nodes) and directions of the cuts. These shape functions are combined and propagated recursively up the tree until the shape function for the root of the tree (root node) is

obtained. An (x, y) pair on the shape function for the root node which satisfies the user specified aspect ratio is chosen and - using pointers saved during the bottom-up process - a complete floorplan is generated. Otten's approach uses a binary slicing tree which is very restrictive and uses simplistic shape models for the leaf cells. This technique has difficulty incorporating I/O pin locations in the optimization process. Another shortcoming is that the bottom-up floorplan sizing does not consider the global connection costs. In general, this consideration is necessary because the choice of a suitable floorplan pattern for a cluster must be made on the basis of the connection cost as well.

Zimmerman [41] improved Otten's technique to optimize direction of the cuts during the shape propagation phase. He estimated the wiring area required for each node of the binary slicing tree and shifted nodes' shape functions to account for the wiring areas. Herrigel [16] and Lengauer [21] proposed a top-down, follow-up phase to minimize the total interconnection length by switching cells across cuts. This method has limited effect since the layout area minimization and interconnection length minimization are separated and since the cell switching - which is performed such that the floorplan area does not change - represents small perturbation to the solution found during the bottom-up phase.

Dai et al. [10] attempted to bridge the gap between the two thrusts mentioned above by computing the best shape (i.e., a target shape) for each node of the cluster tree bottom-up and using these target shapes during the top-down floorplanning phase to evaluate the relative merits of various floorplan patterns and labelings for nodes of the tree. Single target shapes, however, do not carry enough information about the leaf cells to allow reliable decision making at higher levels of the tree hierarchy.

We extend [10] by computing the shape functions for the cluster nodes bottom-up, incorporating constraints on the variability in cell sizes, shapes and pin positions, coupling pin assignment to global routing and finally honoring various physical, topological, and timing constraints.

2.2 Terminology

A k-room *floorplan pattern* is a floorplan structure with exactly k rooms. An *orientation* of a pattern is a clockwise rotation of the pattern with respect to the boundary pin locations. A *labeling* of a pattern corresponds to the assignment of nodes of the cluster tree to individual rooms. A *topological possibility* - denoted by TP - refers to a particular choice of floorplan pattern, pattern labeling and pattern orientation. (See Figure 1.)

The *shape function* for a cluster node gives the lower bound on height of the node as a function of its width. The *interconnection length function* for a cluster node gives the lower bound on area of the node as a function of the estimated length of interconnections

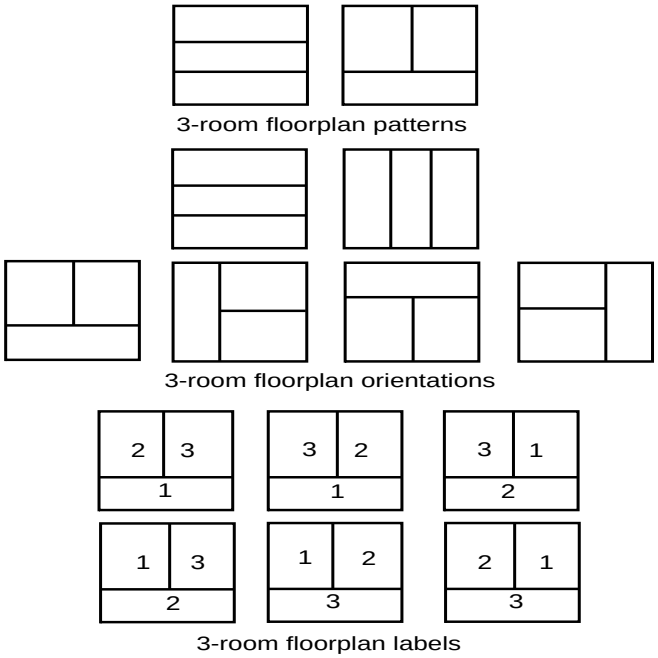

Figure 1: 3-room floorplan patterns, orientations of these patterns, and some of the possible labelings

within the node. This length accounts for all wires which are required to complete connections among children of the node and from children to the external I/O pins. The combined shape function for a particular cluster node - denoted by SF_{TP} - is calculated using the shape functions for its child nodes and the TP assigned to the node. Similarly, the combined interconnection length function for a particular cluster node - denoted by LF_{TP} - is calculated using the interconnection length functions for its child nodes and the TP assigned to the node. (See Subsection 2.4.)

Inputs to the floorplanner are a collection of variable shape cells, shape and orientation constraints on cells, pin position constraints, locations of the chip I/O pads, a net list specifying connections among various cells and a target aspect ratio for the chip. Outputs of the floorplanner are locations, shapes and pin positions for the cells such that all constraints are satisfied and a combination of layout area, total interconnection length and aspect ratio mismatch is minimized.

2.3 Cluster Tree Generation

Hierarchy is the typical way to deal with the complexity of large designs. A hierarchical approach appropriately prunes the solution space and reduces the design objects to

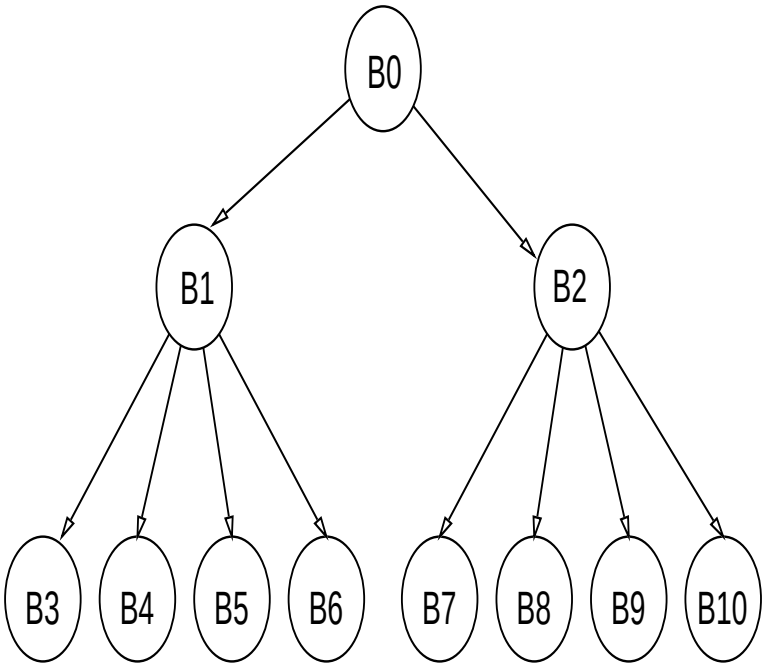

Figure 2: A multi-way cluster tree with 8 cells and 3 levels

manageable sizes. In the context of floorplanning, the hierarchy usually takes the form of a tree where highly connected cells are grouped together. This hierarchical representation of cells greatly simplifies the floorplanning problem since floorplanning algorithms can recursively operate on one hierarchical cell at a time. The tree itself could have a restricted structure (e.g., binary slicing with fixed cut directions and cluster-to-room labelings) or could have a flexible structure (e.g., a multi-way tree). A more flexible structure allows for a higher degree of floorplanning optimization.

A multi-way cluster tree is, therefore, generated (Figure 2). This tree is obtained by minimizing connections among various cells. However, to avoid a cell shape mismatch in the clusters that makes it difficult to find a good placement for cells, shapes of the cells are also considered. The shape mismatch penalty is given more weight at the lower levels of the cluster tree, whereas at the higher levels of the cluster tree, the governing cost measure is that of the connections among various clusters. The maximum branching factor in the tree is restricted to a small value (e.g., four). This allows us to take more floorplan topologies into account that is possible with binary cut trees. At the same time, non-slicing floorplan topologies are avoided. (Our approach can handle non-slicing topologies as well, however, we decided to keep away from them for reasons of efficiency.) Furthermore, if the branching factor is too large, the problem of finding a floorplan solution for a node in the tree becomes as complex as the general floorplanning problem.

To generate one level of the tree, the matching algorithm for simple graphs is used (as in [18]). Without loss of generality, we assume that the number of vertices is a multiple of 4 ($|V| = 4 \times p$). Otherwise, we add up to 3 isolated vertices. The clustering algorithm finds a maximum weight perfect matching[1] M_1 of G and constructs G' by contracting all edges of M_1 using the updated weights. Then, it finds a maximum weight perfect matching M_2 of G' and forms p disjoint clusters of 4 vertices each by grouping end vertices of edges of M_2. The complexity of the algorithm which is dominated by the matching problem is $O(min(|V|^3, |V|\ |E|\ log|V|))$ [35].

The above operations are repeated recursively until a rooted, 4-ary cluster tree is formed. The cell shape mismatch is used as a "soft" constraint, that is, only macros whose area and shape satisfy shape matching constraints become candidates for merging into the same cluster. (A soft constraint is honored as long as there is at least one feasible solution satisfying that constraint. It is ignored otherwise.)

The cluster generation procedure is driven by connectivity and shape matching measures. It is, however, possible to derive the cluster tree directly from the architectural or logical considerations. For example, a designer may decide to put a set of macrocells in a cluster for timing or power distribution reasons. One advantage of my floorplanning procedure is that it will place and size the macros *preserving* the clusters.

2.4 Shape Function Computation

Each general cell has a shape function that defines its height as a function of its width. Assuming a binary tree, [34, 26] showed how the combined shape function for each internal node is calculated. Briefly, the shape function for a node n_0 is found by taking the shape functions for its two children n_1 and n_2 and considering all possible combinations of an element from shape function for n_1 and one from shape function for n_2. Some of these combinations are inferior to other combinations and hence can be discarded. The way in which the dimensions from the two child nodes are combined depends on the direction of the cut line. If the cut line is horizontal, the x-dimension of n_0 is the maximum of the x -dimensions of n_1 and n_2 and the y-dimension of n_0 is the sum of the y-dimensions of n_1 and n_2. Conversely, if the cut is vertical, the x-dimension is the sum and the y-dimension is the maximum. As the shape function for n_0 is constructed, adequate routing space is added to accommodate the connections. (See Figure 3.)

Zimmerman [41] extended the shape computation procedure to a binary node (with unspecified cut orientation). Briefly, to obtain the combined shape function for an unoriented binary node, the lower bound of the shape function corresponding to a hori-

[1] Let $G = [V, E]$ be an undirected graph each of whose edges has a real-valued *weight*. A *matching* M on G is a set of edges no two of which have a common vertex. The *weight* of M is the sum of edge weights. The *maximum weight matching problem* is that of finding a matching of maximum weight [35].

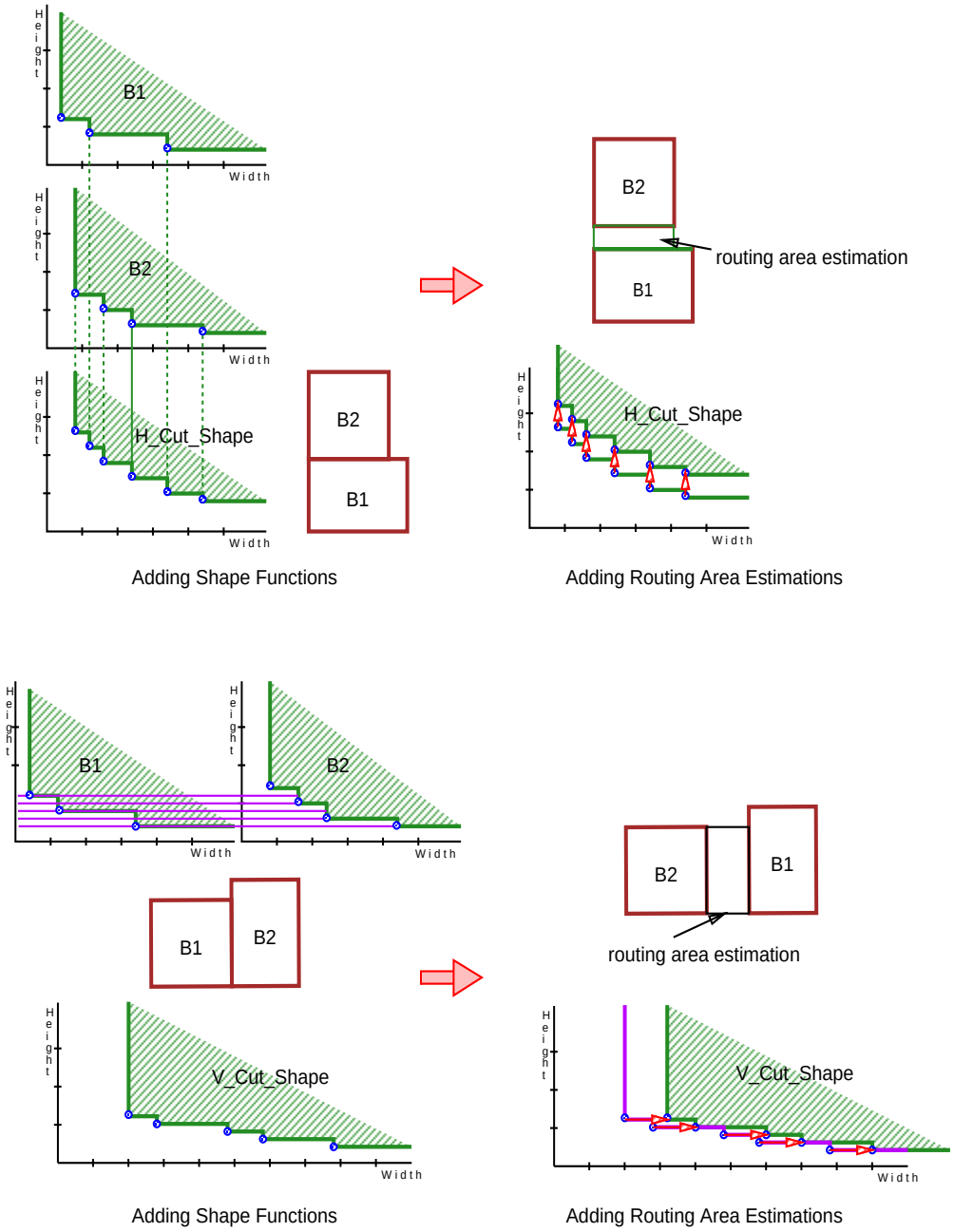

Figure 3: Adding shape functions for horizontal and vertical cuts

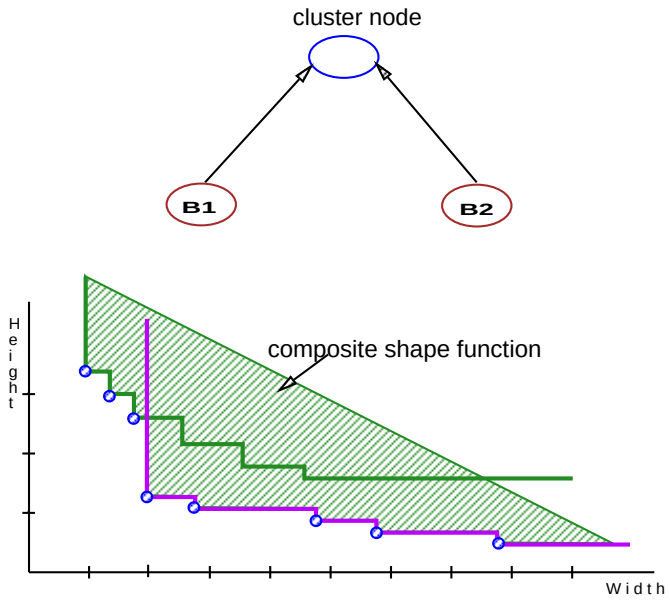

Figure 4: Lower bound merging of the shape functions corresponding to horizontal and vertical cuts

zontal cut and that with a vertical cut is calculated. The routing area is estimated and added to the shape function. (See Figure 4.)

This procedure may be extended to unoriented multi-way cluster trees as follows. Consider a cluster node with k children to which a TP has been assigned. Assume that this TP corresponds to a slicing structure[2], thus, there is a unique binary decomposition tree which represents it. The leaves of this binary tree are children of the cluster node and the internal nodes of the binary tree define directions of the cuts in the TP. The combined shape function for the root of this binary subtree is calculated as described above.

Clustering phase, however, generates a multi-way cluster tree without assigning TPs to cluster nodes. The shape function for an unoriented multi-way cluster node is thus computed as follows. All TPs with k rooms are examined, and their corresponding shape functions are computed. For each TP, the routing areas around the child nodes are estimated. Each (x, y) pair on the SF_{TP} is thus shifted up and to the right to account for the wiring area required. (This is in contrast with [41] which shifts the whole shape function for the node in order to account for the routing area.) Next, the

[2]For a non-slicing TP, the problem of calculating the combined shape function is NP-complete [14]. Either an approximate [38] or a branch-and-bound [37] technique can be used to do this calculation.

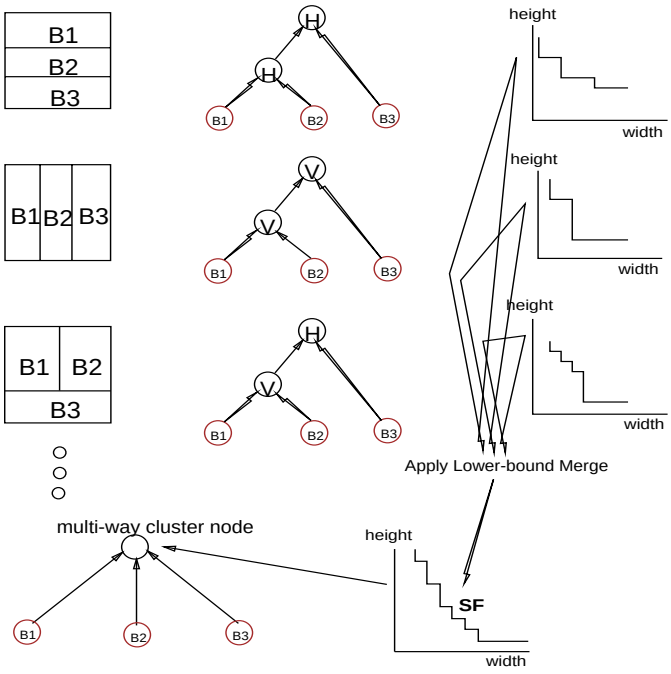

Figure 5: Calculation of the combined shape function for a cluster node with 3 children

lower bound of all SF_{TP}s is taken to obtain the shape function for the multi-way cluster node. (See Figure 5.) (x, y) pairs on the shape function are marked to represent the chosen TP.

This procedure is recursively applied up the tree until the composite shape function for the root node is calculated. The bottom-up shape function and a user specified aspect ratio can be used to generate a minimum area floorplan by propagating the associated geometries of the floorplan solution to the leaf nodes. The above procedure minimizes the layout area - to the extent that the bottom-up wiring area estimation is accurate - but does not optimize the total interconnection length or positions of the signal pins on flexible cells. In Subsection 2.5, we will describe a top-down floorplan optimization procedure which uses the bottom-up shape functions for its cost evaluation, and at the same time, searches for a floorplan solution minimizing interconnection length and satisfying timing and topological constraints. However, we shall first explain how the interconnection length functions for internal nodes of the cluster tree are computed.

The interconnection length function for each cluster node is computed as described below. For each (x, y) pair on the SF_{TP}, an $(l, a = x \times y)$ pair is calculated as follows. The list of nets *intersecting* (i.e., having at least one pin inside) the cluster node is known. For each such net, a minimum rectangle which touches all the flexible blocks and goes through the centers of the fixed blocks is constructed. (Here, flexible blocks

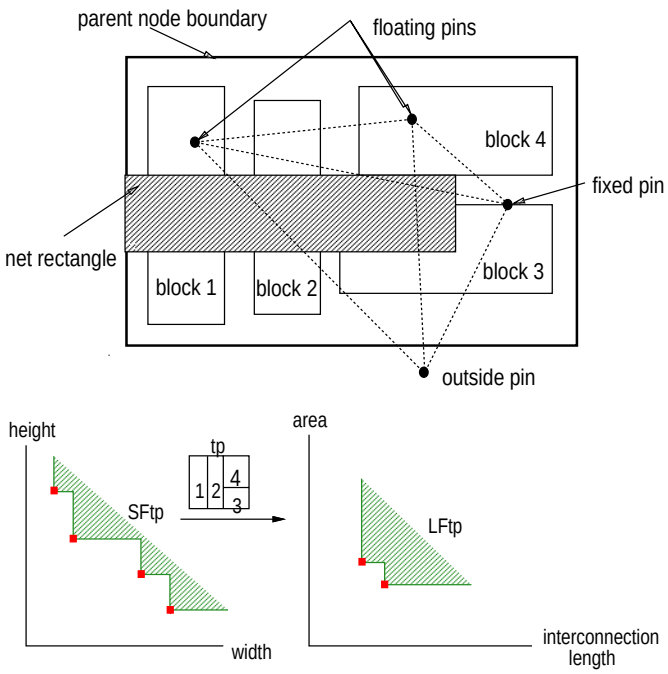

Figure 6: Calculation of the interconnection length function for a cluster node

refer to sub-clusters or variable shape leaf cells within the node, and fixed blocks refer to fixed shape leaf cells.) The idea is that since the pin positions on flexible blocks can be optimized, the minimum rectangle must only touch the blocks. However, since the cell orientations are not known during the bottom-up calculation, the center points of the fixed cells are used. If the net has pins outside the node boundary, the minimum rectangle is extended to touch the closest side on the node boundary. The half perimeter length of this new rectangle gives an estimate on the interconnection length required to complete the routing of the net in question within the node. These half perimeter lengths are summed over all intersecting nets to get l. Next, for each (x, y) pair on the SF_{TP}, an (l, a) is calculated, the inferior (l, a) pairs are dropped and the LF_{TP} is stored at the node. (See Figure 6.)

Again, lower bound merge operation is used to compose the successive LF_{TP}s into the composite interconnection length function for the node. This step is repeated recursively until the composite interconnection length function for the root node is calculated.

2.5 Floorplan Optimization

Each cluster node is floorplanned in a breadth-first manner starting from the root node. When floorplanning a node, the node aspect ratio and the external I/O pin positions

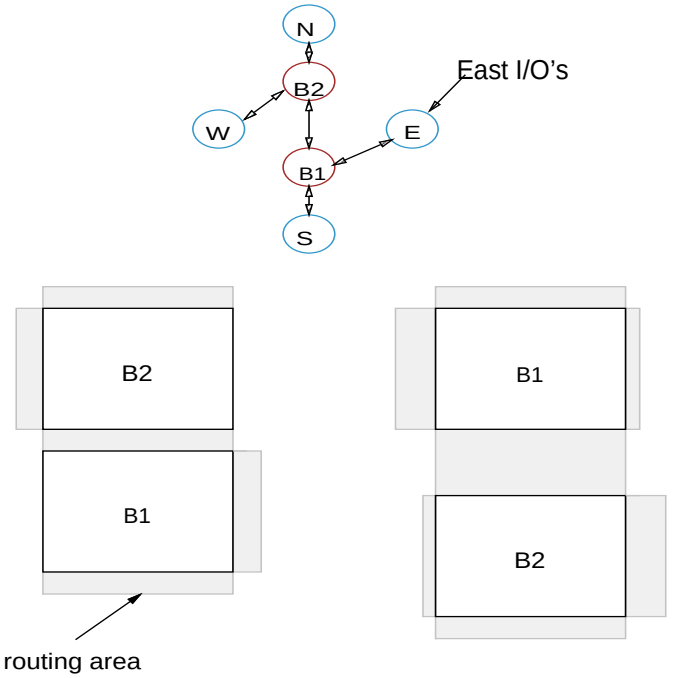

Figure 7: Effect of I/O connections on the routing area estimation during top-down floorplanning

are known. (These are the user specified aspect ratio and the chip I/O pads for the root node. For the non-root nodes, this information has been passed down the tree.) All *TP*s for the node are enumerated, and a floorplanning solution which has the minimum objective function value is selected. The objective function is a combination of area cost, interconnection cost and aspect ratio mismatch cost. Given a cluster node and a particular *TP*, the area and interconnection length costs are calculated by summing the area and interconnection length estimates for its child nodes plus the area and interconnection length required to combine the child nodes into the enumerated *TP*. (The shape and the interconnection length functions of the child nodes act as estimates of the expected layout cost from the partial layout solution where the child nodes are not yet floorplanned to the complete layout.) In Figure 7, the two *TP*s shown are equivalent in terms of area and wire length during the bottom-up cost calculation. However, after considering the global connections, i.e., during the top-down search, these *TP*s have different areas and wire length costs. This is exactly why a purely bottom-up floorplanning procedure is incapable of producing high quality floorplan solutions.

As a result of floorplanning a cluster node, shapes, locations and orientations of its child nodes will become known. Positions of the boundary pins on the node, the estimated routing area around the child cells and the local net list are also known. However, I/O pins for the child nodes must be assigned positions. This I/O pin computation

is necessary in order to influence the floorplanning of the child nodes by the outside connections as well as the internal connections. The task is to assign pins of the nets intersecting the cluster node to the appropriate locations on the boundaries of the child nodes such that the total interconnection length within the cluster node is minimized.

In a manner to be described in Section 3, the I/O pins are propagated to the boundaries of the child nodes. This procedure, therefore, sets the external I/O pins for next lower level of tree hierarchy and directly influences the choice of floorplan topology for the child nodes. At the leaf level, this I/O pin computation coincides with the pin assignment for general cells.

It is worth noting that a designer may partially or fully specify the hierarchical structure. In the latter case, he or she may not only specify the cluster tree but also assign a topological possibility to each internal node of the tree. The floorplanning problem is then to place the macros and assign shapes and pin positions to the flexible ones. The floorplanner handles such input specification. This is useful since in some cases, due to timing constraints or sensitive design, the designer may want to ensure a particular topological relationship between blocks.

2.6 Area Estimation

It is important to include routing area *during* rather than *after* placement. A placement which is optimized to have a rectangular shape of a given aspect ratio is probably neither rectangular nor does it conform to the desired aspect ratio after the routing area is added. In our floorplanning procedure, accurate area estimation during the bottom-up shape computation step and later during the top-down floorplan optimization step is necessary. Besides using the hierarchical decomposition of the problem, the basic idea is to avoid a dynamic shortest path or Steiner tree determination by precomputing the paths for the finite number of floorplan patterns and storing the information in the library of patterns (or *templates*). [10] describes our area estimation procedures in more detail.

2.7 Complexity Analysis

Under the assumptions that the clustering tree is a tree with n leaf nodes, that each leaf node has a shape function with at most m shapes, and that every internal node has k children, the complexity of the floorplan optimization algorithm is given by:

$$O\left(d\ n\ m\ f(k)\right)$$

where $f(k) = k! \times t(k)$ is the number of enumerated TPs and is given in Table 2.7. ($t(k)$ is the number of non-isomorphic oriented floorplan patterns.) The derivation of

k	1	2	3	4
$f(k) = k! \times t(k)$	1	4	36	528

Table 1: Number of topological possibilities $f(k)$ for non-leaf clusters

complexity is as follows. Assume that root of the cluster tree is at level 0 and leaf nodes are at level d. A k-tree of depth d contains $n = k^d$ leaf nodes and $(n-1)/(k-1)$ internal nodes. During floorplanning, the combined shape function for each internal node of the tree is computed. In particular, for a node at level $d-1$, $f(k)$ TPs are enumerated where each TP requires $k-1$ shape function add and $k-2$ shape function merge operations. Each operation takes time proportional to the number of shapes in the shape functions. At level d, each shape function has m points, therefore, $O((k-1)\ m\ f(k))$ is required to compute the combined shape function for a node at level $d-1$. There are k^{d-1} nodes on level $d-1$, therefore, the processing time for nodes on this level is $O(k^{d-1}\ (k-1)\ m\ f(k) \sim n\ m\ f(k))$. The number of shapes on a shape function at level $d-1$ is at most $k\ m$. For nodes at level $d-2$, the processing time is $O(k^{d-2}\ (k-1)\ k\ m\ f(k) \sim n\ m\ f(k))$. The shape functions for a node at level $d-2$ will have at most $k^2\ m$ points. By induction, it can be shown that the processing time for every level of the cluster tree is $O(n\ m\ f(k))$ and since the tree has d levels (excluding the leaf level), the desired result is derived. Note that if bucket sorting is used, that is if an upper bound, say $M \geq m$, is set on the number of shapes in the shape function for each internal node, then the algorithm runs in $O(n\ M\ f(k))$. The run time can be significantly reduced if symmetries of k-room TPs are exploited and / or sub-templates are shared.

3 Pin Assignment with Global Routing

3.1 Prior Work

Previous works on pin assignment assume that shapes and positions of cells are given as input data. These algorithms can be classified into three categories:

1 Those which assign pins on a cell by cell basis [19, 1];

2 Those which assign pins on a net by net basis [40, 39];

3 Those which sequentially process edges of a supergraph containing the global route solutions for all nets, finding a coarse pin assignment and global routing solution followed by a local pin assignment optimization for that global routing [7].

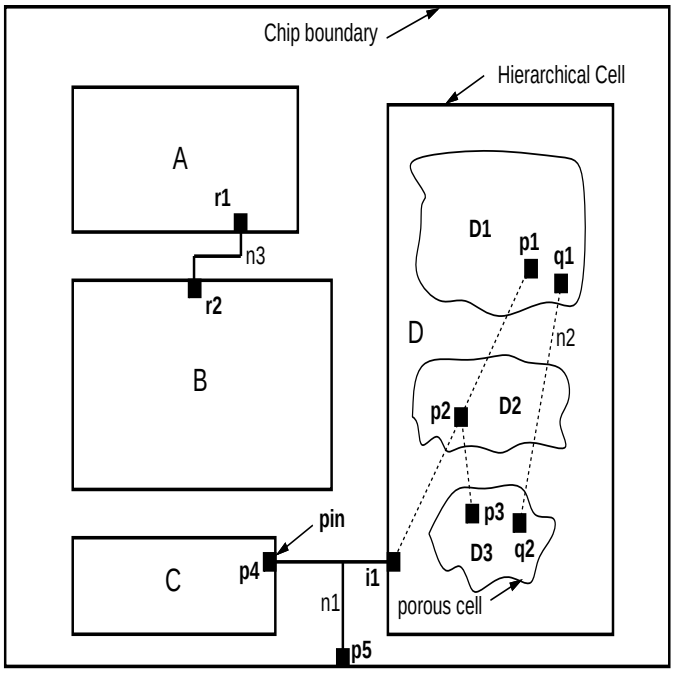

Figure 8: Partial floorplan solution prior to floorplanning D.

Among these approaches, only [7] correlates pin assignment with global routing. However, the quality of the global routing with coarse pin assignment depends on the ordering in which the 'non-essential' edges are eliminated and there is no way to determine a 'good' ordering for edge deletion. It is not clear to us how any of these approaches could be extended to include the floorplanning task.

The technique proposed here solves the pin assignment and global routing problems simultaneously. This technique avoids difficulties associated with the cell or net ordering during pin assignment. Floorplanning determines positions and shapes of hierarchical cells, sets the channel topology and assigns capacities to the routing regions. Pin assignment and global routing operate on the hierarchical floorplanning solution and are weaved in order to produce assignments for floating pins which minimize the layout area as well as the total interconnection length. The initial pin assignment sets the stage for the global routing step by assigning positions to the floating pins. Global routing, then, determines connection patterns and defines channel densities. This information is subsequently used to adjust the pin positions. Global spacing is also performed in order to guarantee routing success.

3.2 The Procedure

Suppose that the root of the cluster tree has been floorplanned. In the process, children of the root have been assigned shapes, positions and pin locations. Next, these child nodes are floorplanned in the order of decreasing area. However, prior to floorplanning a child node, the global net list is updated to include cells and connections inside the node. Updating the cell list means that the node is deleted from the list and its children are inserted into the list. Updating the pin lists consists of deleting pins on the node boundary and adding pins on the cells inside the node. Referring to Figure 8, D is about to be floorplanned. The current net list consists of cells A, B, C, D and nets $n_1 = (p_4, p_5, i_1)$ and $n_3 = (r_1, r_2)$. The global net list is updated to include cells A, B, C, $D1$, $D2$ and $D3$ and new nets $n_1 = (p_1, p_2, p_3, p_4, p_5)$, $n_2 = (q_1, q_2)$ and $n_3 = (r_1, r_2)$. This updating is beneficial since it provides a global view of the layout plane and the connections during floorplanning and pin assignment of D.

After a node (e.g., D) has been floorplanned, shapes and positions of its child nodes ($D1$, $D2$ and $D3$), shapes, positions and pin locations for other nodes (A, B and C), the estimated routing area around the cells (channel capacities) and the global net list are known (Figure 9). The goal is, then, to assign locations to the I/O pins on the child cells such that the channel capacity constraints are satisfied while the total interconnection length and the critical net length violations are minimized .

In order to avoid necessity for the sequential processing of cells or nets, the pin assignment problem is transformed into a linear sum assignment problem as follows. A cost matrix whose rows correspond to the floating pins on the child cells and its columns correspond to the *pin slots* (feasible pin locations) on the child cells is constructed. By solving the linear assignment problem, locations for the floating pins are determined. For that assignment, global routing is performed and channel densities are calculated. If some channels are over-subscribed, the pin assignment procedure is repeated. The initial and final assignments differ only in the way that the linear assignment cost matrix is set-up and filled in.

Routing area may be very irregular. Therefore, in order to store the routing information, the general approach of [27, 12] is used. The entire area of a layout is covered with rectangles referred to as *tiles*. There are two kinds of tiles: *solid* tiles which represent cells and *space* tiles which represent empty space for routing between the cells. Given a placement of rectangular shaped general cells, two tile planes are defined: the horizontal tile plane where all space tiles are maximal horizontal strips and the vertical tile plane where all space tiles are maximal vertical strips. In the tile plane, each space tile has four edges: two of them are called *spans* of the tile (which are completely covered by the solid tiles); the other two form *sides* of the tile. A space tile is a *bottleneck* tile if its sides are covered by the sides of adjacent space tiles. These are areas where wire congestion is most likely to occur. A *junction region* is the maximal empty space which is completely surrounded by the solid tiles, bottleneck tiles or the plane boundaries. Each side of a solid cell is divided into a set of *segments*. The *bottleneck* segments are those

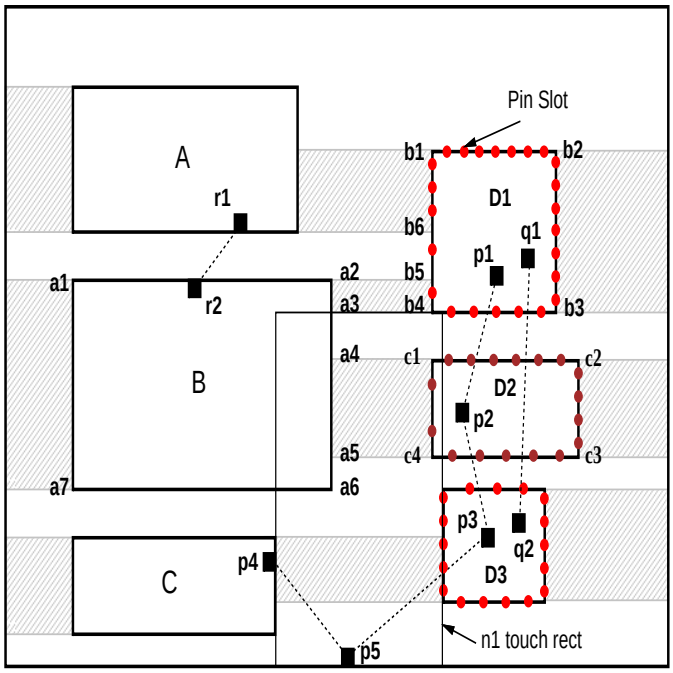

Figure 9: Partial floorplan solution after shape and position calculation.

maximal intervals of sides of cells which are fully covered by the adjacent bottlenecks. The *junction* segments are the remaining maximal segments.

Tile planes are shown in Figure 9. $a_2a_3b_4b_5$ and $a_4a_5c_4c_1$ are bottleneck tiles while $a_3a_4c_1b_4$ is a junction region. (a_2, a_3), (a_4, a_5) and (a_7, a_1) are the vertical bottleneck segments of the cell A. (a_3, a_4) and (a_5, a_6) are vertical junction segments of cell A. Similarly, (b_2, b_3), (b_4, b_5) and (b_6, b_1) are the vertical bottleneck segments of B and (b_5, b_6) is a vertical junction segment of B.

For each segment of each cell, the number of feasible pin slots is calculated. First, the procedure for calculating the number of pin slots on the bottleneck segments is described. Suppose that at most t parallel wires can pass through a bottleneck. Presume that $\alpha \times t$ pins can be placed on each segment covered by that bottleneck. $1/\alpha$ is the track utilization factor which is about 0.65 using a standard channel router. The pin slots are uniformly distributed along the bottleneck segment. (The length of a bottleneck segment and the number of pin slots in it determines the minimum pin-to-pin spacing which must be at least as large as that dictated by the design rules.) Next, the number of pin slots on the adjacent bottleneck segments are summed over to give the number of pin slots on each cell. If a cell has less slots than it has floating pins, new pin slots are added on the junction segments. For each junction segment, the more pessimistic spacing of the adjacent bottleneck segments is used. For example, referring to Figure 9,

spacing of pin slots in segment (b_4, b_5) is bigger than the spacing in segment (b_6, b_1), therefore, for junction segment (b_5, b_6), the same spacing as that in segment (b_4, b_5) is picked. If after adding pin slots to the junction segments, there are not enough slots on some cells, the node may be decompacted so that the number of feasible slots on each child cell may be increased to be equal to or bigger than the number of floating pins there.

For the initial pin assignment, the cost matrix is denoted by $[C]$ and its entries are determined as follows. For each net having floating pins on the child cells, a minimum rectangle which touches the child cells and the external I/O pins connected by the net is constructed. (Figure 9 shows the touch rectangle for net n_1.) All the slots that fall within this *touch rectangle* and lie on the child cells connected by the net are assigned a zero cost. Other slots have positive costs proportional to their Manhattan distances from the touch rectangle. Pin slots on the cells which are not connected by the net are assigned infinite costs. Next, a linear assignment algorithm [2] is run on the matrix $[C]$. Since rows in the cost matrix $[C]$ correspond to floating pins and columns correspond to the pin slots, the linear assignment determines pin assignment with the minimum cost.

During the initial pin assignment, the bottleneck congestions are only implicitly considered (by controlling the number of available slots per segment of each child node). However, chip area and total wire length can be accurately estimated only after global routing. It is, therefore, necessary to combine global routing with pin assignment as is described below. After initial pin assignment, global routing on the partial floorplan produces the shortest connection paths for all nets. This routing scheme may result in over-congested channels. In that case, a final pin assignment which repositions the floating pins on the child cells in order to reduce congestions in the over-subscribed channels is performed. First, the number of pin slots on each segment of each child cell is re-calculated based on the bottleneck congestions after global routing. In particular, the number of pin slots in the over-subscribed bottlenecks (density $>$ capacity) is decreased and this number in the under-subscribed bottlenecks (density $<$ capacity) is increased. Next, the new cost matrix $[D]$ is calculated. Its structure is similar to that of the matrix $[C]$, that is, $[D]$ has the same number of rows as $[C]$ but may have different number of columns.

For each net, the connection tree produced by the global router is examined. For this tree, the list of junction regions that the net goes through are identified. All the slots that fall within these junction regions and are on cells connected by the net have cost zero. All other slots on the connected cells have a cost proportional to their minimum Manhattan distances from the nearest junction region. Slots on cells which are not connected by the net have infinite cost.

To motivate the above slot cost calculation, consider Figure 10. This figure shows the partial floorplan solution after the initial pin assignment and global routing on Figure 9. The global routing for n_1 goes through junction regions j_1, j_2 and j_3 and for n_2 goes

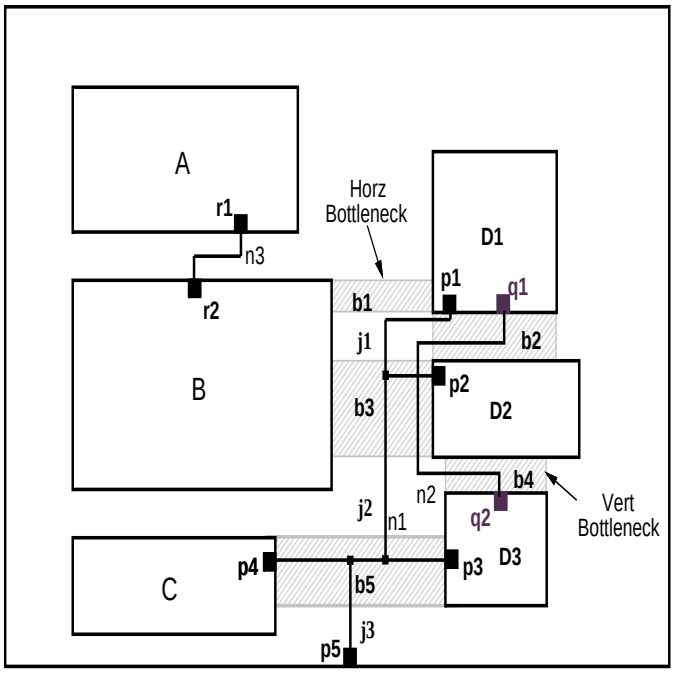

Figure 10: Partial floorplan solution after initial pin assignment.

through j_1 and j_2. Without loss of generality, assume that bottleneck region b_1 is under-subscribed and b_2 is over-subscribed. The goal of the second pin assignment is to alleviate routing congestion in b_2 by moving pins out of that region. To achieve this goal, the number of pin slots in b_2 is reduced, and the number of pin slots in b_1 is increased. Therefore, there will be more competition (among pins of competing nets n_1 and n_2) for available pin slots in b_2 and less competition for those in b_1. Since there are not enough pin slots in b_2 to accommodate all the pins, pins of some nets have to be shifted out. There are pin slots in b_1 and b_2 which have the same Manhattan distances from the junction region j_1. Therefore, either p_1 or q_1 can be moved into b_1 without increasing the sum cost. No pins in b_1 has to move out since the number of slots in b_1 have been increased. Thus, the linear assignment solver will move either p_1 or q_1 out of b_2 into b_1 (Figure 11.) In general, this cost calculation procedure tends to reduce the channel congestions with a minimal increase in the total interconnection length.

It is worthwhile noting that the above procedure based on linear sum assignment does not find the optimal pin locations within each routing channel, and therefore, must be followed by a channel pin arrangement procedure as in Section 5. For example, consider net n_3 in Figure 11. This net does not pass through any junction region. It is advantageous to minimize the half perimeter length of the box enclosing its pins. However, this task cannot be accomplished using linear assignment since the cost of assigning a pin to a slot is dependent on the position of the other pin.

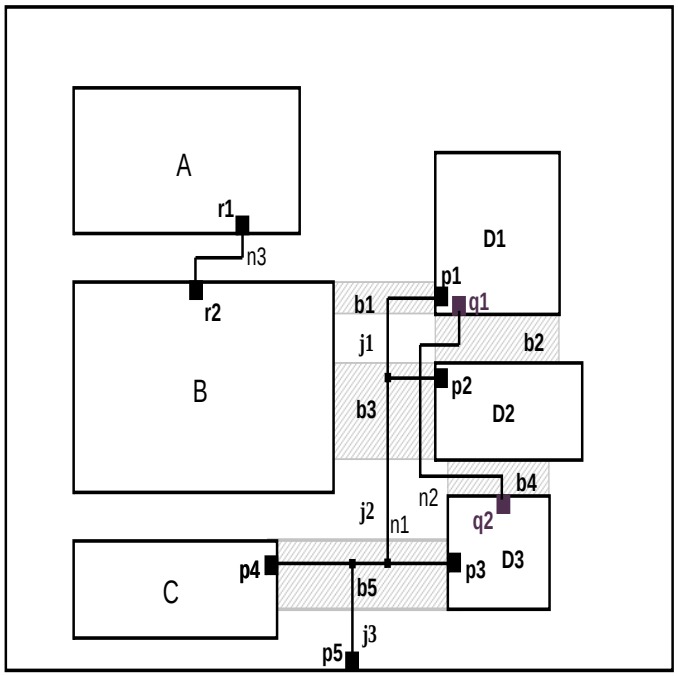

Figure 11: Partial floorplan solution after final pin assignment.

This method also handles nets whose pins were preassigned at the expense of more work during the floorplanning phase and slightly more complex processing during the slot generation and positioning phase. In particular, during the floorplanning step, orientations of the cells must be optimized based on the locations of their fixed pins, and when calculating the number and the distribution of pin slots within bottleneck boundaries, the presence of fixed pins is taken into account. (A list of free boundary regions for each child node is maintained.) One may wish to have a special pin assignment for power and ground nets to satisfy planar routing topology for these nets. In one such scheme, all *Vdd* pins are placed on pin slots located on the top and left cell boundaries and all *Gnd* pins are placed on bottom and right boundaries by giving infinite cost to undesirable pin slots for each *Vdd* or *Gnd* pin. If there are some critical nets, pin slots which are located outside the zero-cost regions for the nets are assigned very high costs, hence, ensuring minimum interconnection lengths for the critical nets. Of course, this may lead to increased wire length for non-critical nets and increased total wire length.

Feedthroughs can be inserted on the non-leaf nodes. After constructing the minimum touch rectangle for a given net, if the rectangle is completely 'blocked' by a child node, two feedthrough pins are inserted on the child node. Next, the net is decomposed into two spanning subtrees as follows. A minimum spanning tree connecting all pins of the net (which must include the feedthrough pin(s) and the edge that goes through the child node) is constructed. The feedthrough edge is subsequently removed and two connected

subtrees are left. The pins in each subtree define a subnet which is then passed to the pin assignment and the global routing steps.

4 Shape Optimization

The initial floorplanning is followed by the global routing process which defines densities of the routing areas. After global routing, the global spacing procedure assures that all bottleneck tiles have capacities equal to or exceeding their corresponding densities. A global shape optimization phase follows next. This is useful because after global routing and spacing, the channel densities are likely to change, and hence, it is possible to decrease the chip area by reshaping some of the flexible blocks. The basic idea is to compute through longest paths through the chip and iteratively compute new dimensions for flexible blocks in those paths in order to reduce the chip extents. The global routing information is incrementally updated and channel densities recomputed from one iteration to the next, therefore, longest paths through the layout surface remain accurate and representative of the final chip dimensions. [10] describes our shape optimization procedure in more detail.

5 Channel Pin Arrangement

The unconstrained channel pin arrangement problem is to determine the pin positions on the bottom and top edges of a routing channel such that the resulting channel density is minimum. In [8] a near-optimal solution for the unconstrained problem is presented. The solution technique, however, places pins of the same net which are on the same side of the channel next to each other. Although the channel density is then nearly minimized, the produced pin arrangement solution is far from desirable. In fact, in many cases, it is desirable to have pins of the same net which are on the same side of a macro-cell be placed far from one another, because the signal must be sent to different regions on that macro-cell.

The generalized channel pin arrangement problem imposes position and partial order constraints on the pins. In [4], it is shown that the generalized channel pin arrangement problem is NP-complete and a polynomial time algorithm is presented for the case of a linear ordering with no position constraints.

Currently, we use a heuristic procedure for assigning pins on the sides of the channel which is similar to that in [8]. This procedure is effective since in most of the examples, nets have at most 2 pins (on opposite sides) and 2 exits in any routing channel. (See Figure 12.)

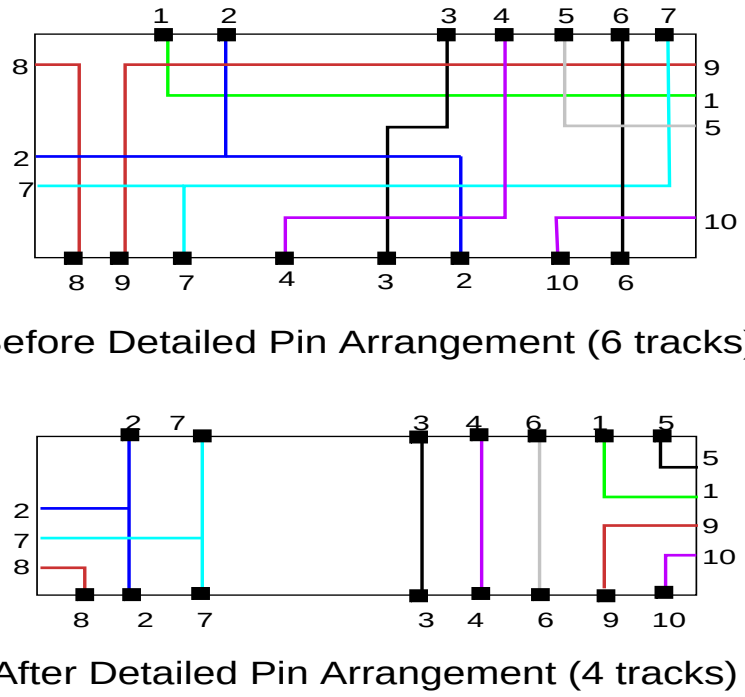

Figure 12: Effect of channel pin arrangement procedure.

6 Performance Oriented Floorplanning

6.1 Introduction

As IC fabrication technology improves and performance requirements on designs increase, it is becoming essential to explicitly optimize the chip performance. Furthermore, since the contribution of interconnection length to the overall chip delay is increasing, automatic control of interconnection length is indispensable. To meet the needs of an expanding electronic industry, high-performance chips must be designed in a short period. Accordingly, a straight forward design flow which incorporates timing analysis and verification into the the physical design process is desirable. This fact motivates the development of layout tools which optimize layout area and chip performance simultaneously. This chapter focuses on the floorplanning step since it often plays a bigger role in overall circuit performance compared to the subsequent global and detailed routing phases.

6.2 Prior Work

Many researchers have addressed the timing-driven placement. These research efforts are mainly tailored to layout styles which have regular structure such as gate array and standard cell design styles. [32] is the only work which presents a technique for the timing-driven placement of the general cells. [17] has addressed the timing-driven global routing for the general cell layouts.

One common approach for solving the timing-driven placement problem is to transform timing constraints into net weights and to use these weights to guide the placement process [13, 3, 24]. This step is repeated and weights are dynamically adjusted until all timing constraints are met. Another approach transforms timing constraints into maximum interconnection lengths [23, 25] and then places the cells based on these net length constraints.

Timing requirements are often represented as path-delay constraints from the primary inputs or the outputs of the sequential logic blocks to the primary outputs or the inputs of other sequential blocks. There are two approaches to fill the gap between a net-based timing model and a path-delay input specification format. The first approach relies on a static timing analyzer which generates timing constraints based on the required arrival times at the primary inputs and outputs prior to the placement step [23]. The second approach uses a timing simulator incorporated into the placement process which dynamically adjusts timing constraints for each net [13, 24, 36].

Prasitjutrakul et al. [32] presented a mathematical programming approach for the timing-driven initial placement of macro-cells. The authors used a source-sink connection structure to model the interconnections among various cells. They assumed that the delays of wires connected to the same signal net are independent. However, delays through wires connected to the same net are closely related and an independent assignment of delays to these wires does not produce a valid timing model. They adopted the total normalized interconnection delay for signal paths as their objective function. This function does not, however, consider the minimization of total interconnection length. A timing-driven placement algorithm which minimizes both of these metrics while satisfying the timing and geometric constraints is needed. Furthermore, the number of timing constraints is proportional to the number of signal paths which could be exponential and the resulting non-linear programming problem is non-convex and can be solved only by breaking it into separate steps. An approach that gives rise to fewer number of timing constraints is more desirable.

[28] presents a path-based timing-driven floorplanning approach similar to that of [33] which addresses these difficulties. In the remainder of this section, we will describe a net-based timing-driven floorplanning approach.

6.3 The Procedure

A path is expressed as a sequence of nets between any source-sink pair in the circuit. Timing requirements for layout can be written as

$$\sum_{i=1}^{k} t(n_i) \leq S_p$$

where n_i is a net in the path, S_p is the slack of the path and $t(n_i)$ is the delay allowed to be used on net n_i. Slack for a path may be positive (indicating early arriving signal at the path end point) or negative (indicating late arriving signal). It is calculated as

$$S_p = T_{eff} - T_p^{max}$$
$$T_{eff} = T_{period} - T_{skew}^{max} - T_{setup} - T_{clk \rightarrow Q}$$
$$T_p^{max} = \sum_{n \in p} (\tau_n + R_{out,n}\ C_{fanout,n})$$

where τ_n is the intrinsic delay through node n, $R_{out,n}$ is the output resistance of n, $C_{fanout,n}$ is the fanout load seen by n, T_{period} is the clock cycle time, T_{skew}^{max} is the maximum clock skew, T_{setup} and $T_{clk \rightarrow Q}$ are the setup time and the internal clock to output delay for the synchronizing elements respectively.

The slack must be distributed to nets on the path [15]. Normalized slack value is defined as $\hat{S}_p = S_p/k$ for a path with slack S_p and k nets. Let $C_{nominal}$ denote the nominal capacitance for all nets. Then, the capacitance constraint for net n_i is given by

$$C_{bound} = C_{nominal} + \frac{Min_{p \in paths(i)} \hat{S}_p}{R_{out,i}}$$

where $paths(i)$ denotes the signal paths which go through source i.

The capacitance constraint on delay given for each net is converted to a wire length constraint for the net. This is because the net length constraint is more easily handled during floorplanning. This conversion is based on a simple delay model which assumes a lumped capacitance at each input pin and a linear on-resistance for each driving pin. The upper bound on the net length, $length_{bound}$ is calculated as follows

$$length_{bound} = \frac{C_{bound}}{C_{unit}}$$

where C_{unit} is the wiring capacitance per unit length.

To make the delay calculation procedure less pessimistic, one must differentiate delays between rising and falling signals. Distinct values are provided for each signal type, and unateness information about each output pin of each macro is used during the path tracing, so that the proper values of the intrinsic and extrinsic delay are selected.

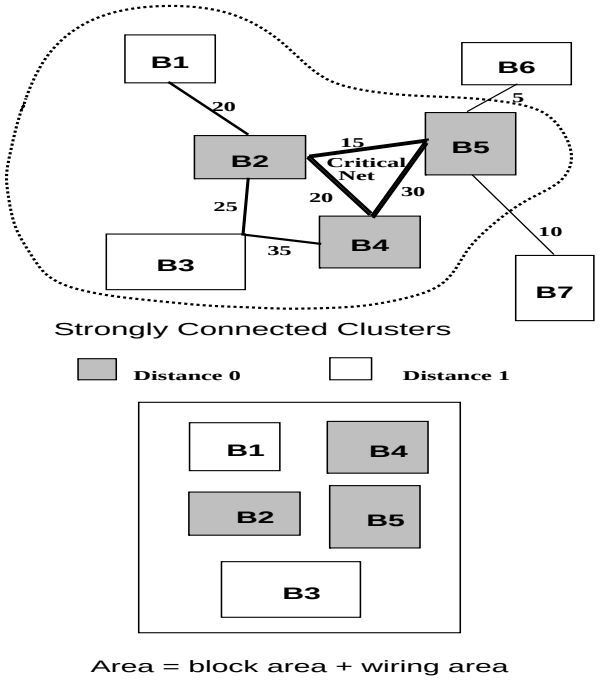

Figure 13: Net neighborhood population for a macrocells

It is also possible that the designer has specified the delay constraints as net delays in which case the net wire length constraints are easily calculated using the output resistance of the driving cell and C_{unit}.

The multi-way cluster tree is generated based on block connectivities and net length constraints. Assume that root of the cluster tree is at level 0 and the leaves are at level d. A net is *l-critical* if it is a net with length constraint which must be handled at level $\geq l$ of the cluster tree. Otherwise, with high probability the net length constraint associated with this net can not be satisfied in the subsequent steps.

Given the net list, a complete graph $G(V, E)$ is built where each vertex represents a leaf cell and each edge corresponds to connections between pairs of cells. Initially, edges are assigned weights according to the number of connections between end points of the edge (i.e., their "natural connectivities"). A set of *d-critical* nets are identified and graph edge weights are updated to force merging of leaf nodes which are connected by these nets. Then, the matching algorithm described in Subsection 2.3 is used to form the clusters. This process is recursively applied until a rooted tree is constructed.

A key question is how to find a *minimal set of level critical nets* (MSLN) at level l of the tree. At this time, levels d up to $l + 1$ of the tree have been generated, and hence, the interconnection length which is necessary to build the partially completed connection tree for each critical net can be easily calculated. This length is denoted by $length_{bu}$ of

the net. The expected length of a critical net on p pins $length_{cur}$ at level l of the tree is estimated as (Figure 13)

$$length_{cur} = \rho(p) \times 2 \times \sqrt{A_a + A_w}$$

where $\rho(p)$ is ratio of the maximum length of a minimal Steiner tree (MST) connecting p pins on the net to the half perimeter length of the bounding box enclosing the net pins [6], A_a and A_w represent sum of the active and the routing areas of *strongly connected* clusters at level l. The notion of strongly connected clusters is related to that of the net neighborhoods which is defined in [30]. These are clusters which are at *distance* zero or one from the net and are connected to one another by edges with weights greater than a threshold. This threshold is the minimum weight on all the edges for the net in question.

An *l-critical* net is one which satisfies

$$length_{bound} - length_{bu} - length_{cur} \leq T_\epsilon$$

where $T_\epsilon > 0$ is the *criticality threshold.* After computing MSLN, the graph edge weights are updated to force merging of the tree nodes connected by MSLN. The updated weight for an edge belonging to MSLN, w_{new} is given by

$$length_{tot} = length_{bu} + length_{cur}$$
$$w_{new} = bias + w_{old} + \sum (\frac{length_{bound} - length_{tot}}{length_{bound}})^2$$

where w_{old} is the weight before updating and $bias$ is the maximum value of the old connectivities. The $bias$ is set to some large value in order to force merging of the nodes.

Reasons for choosing a *minimal* set of level critical nets are the following: Satisfying net length constraints may increase the overall cost of the cluster tree (in terms of the total interconnection length and layout area). In general, the earlier a commitment is made toward satisfying a particular net length constraint, the larger this increase is. Hence, the clustering decisions based on critical net length requirements are postponed as much as possible so that the restrictive effects of premature decisions are avoided.

After generating level l of the cluster tree, *interconnection length function* s for the critical nets are computed as described in Subsection 2.4. These functions are used during the top-down phase to guide the search for a good placement solution while satisfying timing constraints.

Top-down floorplanning as in Subsection 2.5 is the next step. However, a new cost term is added to the floorplanning objective function. The new term penalizes net length constraint violations and is given by

$$criticalNetCost = \frac{\sum_{nets} boundViolation_{net}}{numCriticalNets}$$

where the summation is over all the critical nets which have pins within the node. *numCriticalNets* is the number of such nets. The $boundViolation_{net}$ is calculated as

$$boundViolation_{net} = \begin{cases} (\frac{length_{tot}}{length_{bound}})^2 & \text{if } length_{tot} > length_{bound} \\ \frac{length_{tot}}{length_{bound}} & \text{otherwise} \end{cases}$$

$$length_{tot} = length_{intra} + length_{inter} + length_{td}$$

where $length_{intra}$ is the length of wires used within subclusters of the cluster node in question, $length_{inter}$ is the length of wires required to complete connections among subclusters and to the current I/O pins for the cluster node and $length_{td}$ is the length of wires *used-up* in order to produce the partial placement solution starting from the root. Expressions for $length_{intra}$ and $length_{inter}$ were given previously; $length_{td}$ is easily calculated by adding the $length_{inter}$ of the already placed cluster nodes.

7 Experimental Results

These ideas have been incorporated into BEAR-FP [29] is a macro-cell based layout system which builds on its predecessor, the BEAR system [9]. It inherits BEAR's dynamic and efficient data representation, which unifies topological and geometrical information, and BEAR's routing system. BEAR-FP, however, has a new floorplanning procedure with integrated global routing and hierarchical pin assignment, a new Steiner-tree global router, a detailed channel pin arrangement procedure, and a timing-driven clustering and placement capability. BEAR-FP supports traditional macro-cell layout with routing channels as well as channel-free layout style. BEAR-FP runs on workstations which support the X-window (currently X11, release 4). Users may customize the colors, font styles, and other parameters by setting the *X-defaults* file [29]. BEAR-FP has been integrated into the OCT framework [5].

Due to lack of published results on floorplanning and pin assignment, comparative results for BEAR-FP floorplanner could not be presented. Therefore, we ran BEAR-FP on *Xerox*, *Ami33* and *Ami49* general cell benchmarks and recorded layout area and interconnection length after routing (for chip aspect ratios 1.0, 2.0, and 4.0). Next, we ran BEAR-FP on the variable-shape version of these benchmarks. For *Xerox-F*, we used the shape list specified in [22]; for *Ami33-F* and *Ami49-F*, we generated 7 shapes per macrocell (spanning aspect ratios 1.0 to 3.0 with constant area). Pins on the fixed-shape macrocells were fixed, and pins on the variable-shape macrocells were completely floating (no side, position or ordering constraints). Cell rotation was permitted. Table 2 summarizes characteristics of these benchmark circuits.

Table 3 presents the results. The chip area and total wire length for the variable-shape benchmarks are about 10% and 15% less than their fixed-shape counterparts respectively.

example	numCells	numNets	numIOs	shapeType	pinType
Xerox	10	203	2	F	F
Ami33	33	121	38	F	F
Ami49	49	408	22	F	F
Xerox-F	10	203	2	V	L
Ami33-F	33	121	38	V	L
Ami49-F	49	408	22	V	L

Table 2: Description of benchmark circuits (F = Fixed, V = Variable, L = fLoating)

The run time (on DEC3100) for the *Xerox* and *Xerox-F* circuits are 17 and 43 seconds. Most of the time is spent in the hierarchical pin assignment and global routing steps.

	aspect ratio = 1		aspect ratio = 2	
example	chip area	wire length	chip area	wire length
Xerox	27.2	626.1	25.6	613.8
Ami33	2.65	131.5	2.69	132.3
Ami49	50.6	983.3	52.3	994.1
Xerox-F	26.1	540.4	25.1	535.3
Ami33-F	2.34	109.7	2.45	110.6
Ami49-F	45.2	713.4	48.0	721.6

Table 3: Chip area (mm^2) and total wire length (mm) for the benchmark circuits

Figures 14 and 15 show the pre- and post-routing results for *Ami33* benchmark. Figures 16 and 17 show the results for *Ami33-F* benchmark. There is good match between the top-down routing area estimations and the final routing areas.

The net-based approach has been implemented and incorporated into BEAR-FP. We carried out experiments on the *MCNC* benchmarks in order to evaluate the performance of the net-based floorplanning procedure. Tables 4 and 5 depict wire lengths for half of the critical nets in *Xerox* and *Ami33*. (Chip aspect ratios are set to 2.0.) The remaining critical nets show similar improvements and are skipped in order to save space. The first column gives the net name, the second column specifies the upper bound lengths, the third and fourth columns correspond to the estimated interconnection lengths after routing with conventional placement and with timing-driven placement. The upper bound length constraints for *Xerox* have been specified as part of the *MCNC* benchmark set. For *Ami33*, we assigned tight upper bound constraints to eight of the nets.

Timing-driven placement of *Xerox* produces a layout with 6.5% increase in chip area and 3.1% increase in total interconnection length (compared to placement without length constraints). Timing-driven placement of *Ami33* produces a layout with 2.1% increase in chip area and 1.7% increase in total interconnection length. (See Table 6.) This is to be expected since satisfying the critical net length constraints often leads to an

Figure 14: Placement result for *Ami33* benchmark

Figure 15: Routing result for *Ami33* benchmark

Figure 16: Placement result for *Ami33-F* benchmark

Figure 17: Routing result for *Ami33-F* benchmark

net name	upper bound	conventional	timing-driven
A1PC	850	281	220
NRLFTA	600	5432	560
TXREQ	600	906	526
PAORB	1200	671	873
A1A	2000	5648	1294
LK	850	684	731
ID	2200	3453	1720
RXOPDA	850	254	206

Table 4: Critical net length results for *Xerox* circuit (all values are in μ meters)

net name	upper bound	conventional	timing-driven
7	600	1859	299
99	600	488	338
222	700	2787	77
257	700	2215	645

Table 5: Critical net length results for *Ami33* circuit (all values are in μ meters)

increase in chip area and/or total interconnection length. There are no upper bound length violations for the benchmark examples.

example	placement	chip area	total wire length
Xerox	conventional	25.6	613.8
	timing-driven	27.3	632.3
Ami33	conventional	2.69	132.3
	timing-driven	2.75	134.5

Table 6: Chip area (mm^2) and total interconnection length (mm) results for *Xerox* and *Ami33* circuits

8 Concluding Remarks

BEAR-FP provides a complete pipeline from a net list specification of a circuit to a finished layout. The user specifies various physical, timing and topological constraints, controls the order that various optimization tools are invoked and sets parameters to control the outcomes of these procedures. BEAR-FP addresses the issue of performance optimization by including a scheme for timing-driven layout that spans the entire layout process, i.e., the timing constraints influence the clustering, floorplanning, pin assignment and global routing steps.

The BEAR-FP floorplanner minimizes chip area and total interconnection length subject to various topological, geometrical and timing constraints. Novel features of the floorplanner may be stated as: extension of shape computation method to multi-way unoriented clusters; a systematic top-down floorplan optimization which minimizes wire length as well as chip area; efficient and accurate wiring area estimation methods during the shape computation and floorplan optimization; integration of pin assignment and global routing procedures; introduction of a novel hierarchical, performance-oriented floorplanning technique; and accommodation of analog device constraints. The floorplanner can be used either in the feedback loop of a high-level synthesis system or as a stand-alone system for final layout generation in macro cell design.

Floorplanning for large channelless gate arrays can also be addressed by BEAR-FP. At the beginning of the design cycle, the gates are partitioned into a set of frames based on the timing requirements and functional hierarchy. During the physical implementation, placement improvement algorithms move cells within (but not outside of) frames with specified timing for critical paths influencing cell movement.

Today's VLSI chips often contain whole systems including the interfaces with analog inputs or outputs. The design time and cost associated with the dedicated analog interface modules often constitute a bottleneck in semicustom design of mixed analog / digital systems. Layout of an analog circuit strongly influences its performance. Performance constraints imposed on an analog circuit are often converted into physical constraints such as placement symmetries, matching orientations, equal distances among pairs of modules, and net length bounds. An analog module has a number of different implementations for its shape; its pin positions are, however, fixed for each implementation. These physical constraints can be easily incorporated into BEAR-FP by suitable modifications to cluster tree generation procedure followed by appropriate pruning of the search space during the top-down floorplan optimization phase.

BEAR-FP can be used for early planning during the conceptual design phase as well. In this mode, it does a quick floorplanning whose results can be used to perform trade-off analysis when selecting appropriate interconnect technology and layout methodology for subsystem design, and guide the synthesis tools by providing information about the interface characteristics of the modules and how the physical design process impacts the timing on certain critical paths of the circuit.

Acknowledgements

The authors would like to thank Professor W. W. Dai of the University of California, Santa Cruz, Professor M. Marek-Sadowska of the University of California, Santa Barbara, Dr. B. T. Preas of Xerox, PARC and Y. Ogawa of Hitachi Research Center for their valuable discussions.

References

[1] H. N. Brady. An approach to topological pin assignment. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, CAD-3:250–255, July 1987.

[2] R. E. Burkhard and U. Derigs. *Assignment and matching problems: solution methods with Fortran programs.* Springer Verlag, 1980.

[3] M. Burstein and M. N. Youssef. Timing-influenced layout design. In *Proceedings of the 22nd Design Automation Conference*, pages 124–130, June 1985.

[4] Y. Cai and D. F. Wong. An optimal channel pin assignment algorithm. In *Proceedings of the IEEE International Conference on Computer Aided Design*, pages 10–13, November 1990.

[5] A. Casotto, editor. *Oct Tools Distribution 5.0.* Electronics Research Laboratory, University of California, Berkeley, March 1991.

[6] F. R. K. Chung and F. K. Hwang. The largest minimal rectilinear steiner trees for a set of n points enclosed in a rectangle with given perimeter. *Networks*, 9:19–36, 1979.

[7] J. Cong. Pin assignment with global routing. In *Proceedings of the IEEE International Conference on Computer Aided Design*, pages 302–305, November 1989.

[8] J. Cong. A provable near-optimal algorithm for the channel pin assignment problem. In *Proceedings of the International Conference on Computer Desgin*, pages 319–322, October 1991.

[9] W. M. Dai, H. H. Chen, R. Dutta, M. Jackson, E. S. Kuh, M. Marek-Sadowska, M. Sato, D. Wang, and X. M. Xiong. BEAR: A new building-block layout system. In *Proceedings of the IEEE International Conference on Computer Aided Design*, pages 34–37, November 1987.

[10] W. M. Dai, B. Eschermann, E. S. Kuh, and M. Pedram. Hierarchical placement and floorplanning in BEAR. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, CAD-8:1335–1349, December 1989.

[11] W. M. Dai and E. S. Kuh. Simultaneous floor planning and global routing for hierarchical building-block layout. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, CAD-6:828–837, 1987.

[12] W. M. Dai, M. Sato, and E. S. Kuh. A dynamic and efficient representation of building-block layout. In *Proceedings of the 24th Design Automation Conference*, pages 376–384, June 1987.

[13] A. E. Dunlop, V. D. Agrawal, D. N. Deutsch, M. F. Jukl, P. Kozak, and M. Wiesel. Chip layout optimization using critical path weighting. In *Proceedings of the 21st Design Automation Conference*, pages 133–136, June 1984.

[14] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-completeness*. W. H. Freeman and Company, 1979.

[15] P. S. Hauge, R. Nair, and E. J. Yoffa. Circuit placement for predictable performance. In *Proceedings of the IEEE International Conference on Computer Aided Design*, pages 88–91, November 1987.

[16] A. Herrigel, M. Glaser, and W. Fichtner. A global floorplanning technique for VLSI layout. In *Proceedings of the International Conference on Computer Desgin*, pages 92–95, October 1989.

[17] M. A. B. Jackson, E. S. Kuh, and M. Marek-Sadowska. Timing-driven routing for building block layout. In *Proceedings of the International Symposium on Circuits and Systems*, pages 518–519, 1987.

[18] M. Khellaf. *On the Partitioning of Graphs and Hypergraphs*. PhD thesis, University of California, Berkeley, 1987.

[19] N. L. Koren. Pin assignment in automated printed circuit board. In *Proceedings of the 9th Design Automation Conference*, pages 72–79, June 1972.

[20] U. P. Lauther. A min-cut placement algorithm for general cell assemblies based on a graph representation. In *Proceedings of the 16th Design Automation Conference*, pages 1–10, June 1979.

[21] T. Lengauer. *Combinatorial algorithms for integrated circuit layout*. Computer Science. Wiley-Teubner, 1990.

[22] Microelectronics Research Center of North Carolina, Research Triangle Park, NC. *Floorplanning benchmarks*, 1990.

[23] R. Nair, C. L. Berman, P. S. Hauge, and E. J. Yoffa. Generation of performance constraints for layout. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, CAD-8(8):1989, August 1989.

[24] Y. Ogawa, T. Ishii, Y. Shiraishi, H. Terai, and T. Kozawa. Efficient placement algorithms optimizing delay for high-speed ECL masterslic LSI's. In *Proceedings of the 23rd Design Automation Conference*, pages 404–410, June 1986.

[25] Y. Ogawa, M. Pedram, and E. S. Kuh. Timing-driven placement for general cell layouts. In *Proceedings of the International Symposium on Circuits and Systems*, pages 872–875, May 1990.

[26] R. H. J. M. Otten. Efficient floorplan optimization. In *Proceedings of the International Conference on Computer Desgin*, pages 499–502, October 1983.

[27] J. K. Ousterhout. Corner stitching: A data structuring technique for VLSI layout tool. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, CAD-1, 1984.

[28] M. Pedram. *An integrated approach to logic synthesis and physical design.* PhD thesis, University of California, Berkeley, August 1991. Technical Report UCB/ERL M91/69.

[29] M. Pedram, W. M. Dai, M. Marek-Sadowska, G. Carvalho, D. Wang, and B. Chen. BEAR-FP manual: Distribution 1.0. Technical Report UCB/ERL M90/118, University of California, Berkeley, 1989.

[30] M. Pedram and B. T. Preas. Interconnection length estimation for optimized standard cell layouts. In *Proceedings of the International Conference on Computer Desgin*, pages 390–393, October 1989.

[31] D. P. La Potin and S. W. Director. MASON: A global floorplanning approach for VLSI design. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, CAD-5(4):828–837, October 1986.

[32] S. Prasitjutrakul and W. J. Kubitz. Path-delay constrained floorplanning: A mathematical programming approach for initial placement. In *Proceedings of the 26th Design Automation Conference*, pages 364–369, June 1989.

[33] A. Srinivasan, K. Chaudhary, and E. S. Kuh. RITUAL: An algorithm for performance-driven placement of cell-based IC's. In *Proceedings of the Third Physical Design Workshop*, May 1991.

[34] L. Stockmeyer. Optimal orientations of cells in slicing floorplan designs. *Information and Control*, 57:91–101, 1983.

[35] R. E. Tarjan. *Data Structures and Network Algorithms.* Society for Industrial and Applied Mathematics, 1983.

[36] S. Teig, R. L. Smith, and J. Seaton. Timing-driven layout of cel-based IC's. In *IFIP International Conference on Very Large Scale Integration*, pages 63–73, 1986.

[37] S. Wimer, I. Koren, and I. Cederbaum. Optimal aspect ratios of building blocks in VLSI. In *Proceedings of the 25th Design Automation Conference*, pages 66–72, June 1988.

[38] D. F. Wong and P. Sakhamuri. Efficient floorplan area optimization. In *Proceedings of the 26th Design Automation Conference*, pages 586–589, June 1989.

[39] X. Yao and C. L. Liu. Pin position assignment for movable pins in macro-cells. *Int'l Journal of Computer Aided VLSI Design*, 1990. To appear.

[40] X. Yao, M. Yamada, and C. L. Liu. A new approach to the pin assignment problem. In *Proceedings of the 25th Design Automation Conference*, pages 566–572, June 1988.

[41] G. Zimmermann. A new area and shape function estimation technique for VLSI layout. In *Proceedings of the 26th Design Automation Conference*, pages 60–65, June 1988.