HIERARCHICAL PLACEMENT FOR MACROCELLS WITH SIMULTANEOUS ROUTING AREA ALLOCATION

by

Bernhard Eschermann

Memorandum No. UCB/ERL M88/49

29 July 1988

TITLE PAGE

HIERARCHICAL PLACEMENT FOR MACROCELLS WITH SIMULTANEOUS ROUTING AREA ALLOCATION

by

Bernhard Eschermann

Memorandum No. UCB/ERL M88/49

29 July 1988

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

TITLE PAGE

HIERARCHICAL PLACEMENT FOR MACROCELLS WITH SIMULTANEOUS ROUTING AREA ALLOCATION

by

Bernhard Eschermann

Memorandum No. UCB/ERL M88/49

29 July 1988

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

Hierarchical Placement for Macrocells with Simultaneous Routing Area Allocation

Bernhard Eschermann

University of California, Berkeley
Department of Electrical Engineering and Computer Sciences

ABSTRACT

This report describes the placement and routing area estimation program of the BEAR macrocell layout system. A combination of top-down and bottom-up heuristics is employed to make best use of a hierarchical description. The interdependency of placement and routing is considered explicitly. The program was implemented in C under the UNIX† operating system. Experimental results show that significant area and wire length reductions over previous approaches are possible.

† UNIX is a trademark of Bell Laboratories.

Table of Contents

Chapter 1. Introduction

1.1. Problem Description

Integrated circuit fabrication technology advanced rapidly over the last 3 decades. Three major design styles - full custom design, standard cells and gate arrays - targeted towards different applications evolved. Hierarchical decomposition is necessary in any of these approaches to deal with the complexity of VLSI circuits. At some level of abstraction the different parts of a circuit, be they developed manually, with module generators or composed of standard cells, can be viewed as rectangular or rectilinear objects with given shapes and sizes. These objects then have to be placed on a plane, oriented in one of the 8 possible ways and connected with each other at fixed terminal locations.

The objective of the placement is to provide an arrangement of blocks that - after having been routed - fits into an enclosing rectangle of minimum area with given height, width or aspect ratio. To get a high performance circuit a concurrent goal is to minimize the length of connections.

However, such a task is not well defined. For ''significant solutions'' (Fig. 1.1) wiring length generally increases when the area is decreased and vice versa.

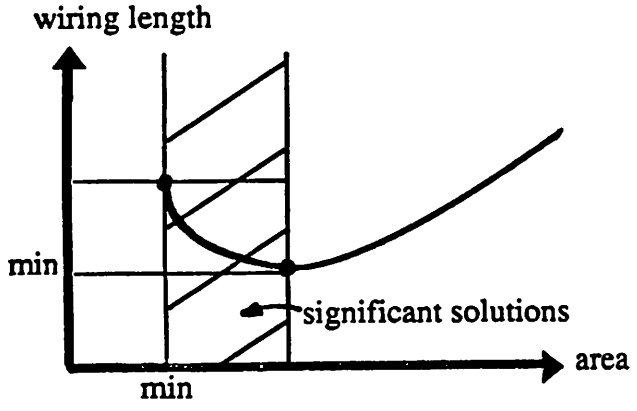


Fig. 1.1 Area / wire length trade-off

Furthermore, the impact of placement decisions on the final result cannot be predicted accurately without actually doing the detailed routing. Therefore iterations in the design process are common although they are very costly and not guaranteed to converge.

In this report a hierarchical placement algorithm for rectangular macrocells is described. The algorithm combines the goal orientation of a top-down approach with the module orientation of bottom-up techniques. The result is a "meet in the middle" strategy. It considers the mutual dependency between placement and routing explicitly by incorporating a novel method of hierarchical routing area estimation. Thus a more uniform design flow is achieved, reducing the need for iterations.

The general idea can be summarized as follows: At a given point in the design process make maximal use of all the available information. Estimate the effects of pending decisions on subsequent steps of the design. Do not optimize a local cost function that on the whole may produce a suboptimal solution unless the computational cost of a more global view is excessive.

1.2. Overview

This work is based on the general approach described in [Dai87]. The following description should give the reader a global view of the whole placement process before a more detailed description of the different parts is given.

As a first step, a hierarchical representation of the problem is generated. Macrocells that are strongly connected with each other are clustered together in clusters of some maximum size. This step is recursively repeated and thus produces the different hierarchical levels of a *clustering tree*.

In the placement step, this tree is traversed top-down. Given an overall aspect ratio goal and possibly the location of I/O terminals, at each level of the hierarchy all possible topologies for the clusters on that level of the hierarchy are examined. The objective function for choosing a particular possibility is a linear combination of a geometry cost and a connection cost. The user controls the trade-off. The chosen configuration then, in turn, sets the shape goals and the I/O goals on the next lower hierarchical level. The same decision process can be repeated.

To provide the geometry cost function on higher levels with more information, the clustering also passes shape information from the leaves towards the root of the tree. Additionally, during the top-down traversal of the tree, a lookahead is possible so that the decision on some level is not restricted by the available block shape information on the immediately following level of the hierarchy.

For each of the topological possibilities the routing space necessary to implement the connections between different clusters is estimated. This area and its location together with the cluster areas / block shapes is used to compute the value of the geometry cost function. It also helps to determine the goal shape of the clusters on the next lower level of the hierarchy. In that way space for global connections is provided on higher levels of the hierarchy, when the rough position of clusters is known. The allocation of space for local connections is deferred until later in the process when the detailed block positions in the clusters are to be generated.

1.3. Related Work

In a sense the described algorithm is a generalization of the mincut algorithm [Lau79] where the number of elements per cluster is not restricted to 2. The advantages are

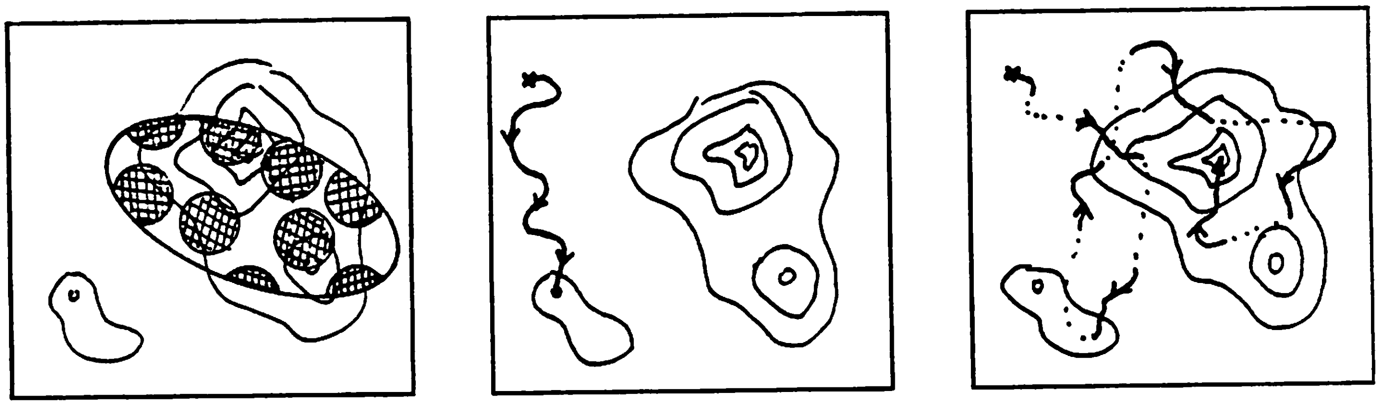
- With the larger degree of freedom to group elements together, an artificial separation of related blocks is less likely. This is enhanced by the fact that the binary mincut tree is generated by a top-down partioning, whereas in this work a data-driven clustering is used. In this way "natural clusters" of blocks are preserved as much as possible.
- A binary tree already determines the eventual placement to a great extent. Hence it is crucial to partition the blocks in such a way that the shapes of the rectangles after the last cut match the shapes of the actual building blocks as much as possible. Because this is very difficult to achieve, an iterative improvement step [Pre86] or restructuring the tree [Mue87, Koz84] is necessary.
- If the nımber of blocks in a cluster is 5 or greater, non-slicing topologies become possible. Since it is possible to effectively handle the routing of non-slicing structures by introducing L-shaped channels [Che87], there is no reason to preclude the placement from using these topologies to optimize area and / or

interconnection length for special cases [1].

The price to pay is that more time has to be spent in the enumeration and the programs become more complex.

To draw comparisons with other approaches, the notion of a search space is useful (see Fig. 1.2). The rectangle encloses the set of all possible placements. The curves represent the contour lines of some objective function.

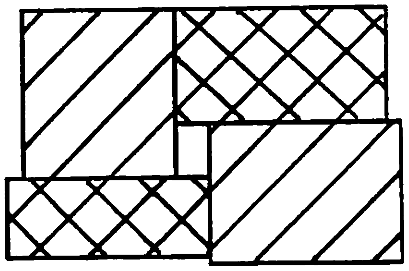
The proposed algorithm works on a subspace of the search space given by the clustering tree (Fig. 1.2.a). On the top level of the hierarchy all the possible topologies are enumerated exhaustively, however, only a subset of these possibilities is further explored on lower levels of the hierarchy.



a. cluster enumeration   b. iterative improvement   c. simulated annealing

Fig. 1.2 Exploration of the search space in different algorithms

1 The following topology seems to give a non-slicing structure for a 4-block cluster. However it does not constitute a rectangle dissection [Ott82], its polar graph [Oht70] actually represents the 5-block non-slicing case. It is obtained by "squeezing" [Lau79] a slicing structure.



Iterative improvement heuristics [Han76, Koz84, Ued85] start with some initial placement and try to monotonically improve it by performing local changes. This method might well lead to a neighboring local optimum that is far away from the global optimum (Fig. 1.2.b).

Simulated annealing programs [Sec85] avoid this problem by allowing changes that lead away from local optima on a random basis (Fig. 1.2.c). Nonlocal changes like the swapping of modules become necessary. For regular geometries like standard cells this works well, although CPU time requirements are very high. For macrocells the situation becomes more involved, because the solutions obtained after the changes are generally not feasible due to block overlaps.

Analytical techniques derive a nonlinear programming problem that can be solved with a penalty function method [Sha85]. The initial solution minimizes some wire length estimate without caring about geometrical constraints. An iteration eventually leads to an approximately feasible solution by successively increasing a penalty for block overlaps in the objective function. The solution is only approximately feasible because corner overlaps are still possible. CPU times are in the same order of magnitude as for simulated annealing. It is difficult to incorporate the allocation of routing area in the formulation.

Chapter 2. Clustering

The clustering algorithms used in the placement process have been described in [Khe87, Esc87]. In the first part of this chapter some criteria to evaluate the quality of a clustering based on the intra-cluster connectivity are introduced. In the second part the incorporation of geometrical constraints in the clustering is discussed.

2.1. Evaluation of Cluster Quality

It is difficult to compare two clustering trees obtained by one algorithm with different parameters or by two different algorithms unless some common evaluation method is available. Since the primary objective of the clustering is to combine blocks that are strongly connected, in this section criteria to examine the clustering performance with respect to that objective are proposed.

The problem is represented as a weighted graph where the nodes represent the macrocells and the number of connections between them determines the weight of the edges. n-point nets are split up into $\frac{n\,(n-1)}{2}$ 2-point nets (edges) - the clique of the n terminals of the net - with a weight of $\frac{2}{n\,(n-1)}$ each. In this way a net as a whole always has the same weight, no matter how many blocks it connects. For the clustering that is a desirable property [Esc87].

Let i and j be nodes, c_{ij} the weight of the edge between i and j. Let P(i) be the cluster to which i belongs (see Fig. 2.1).

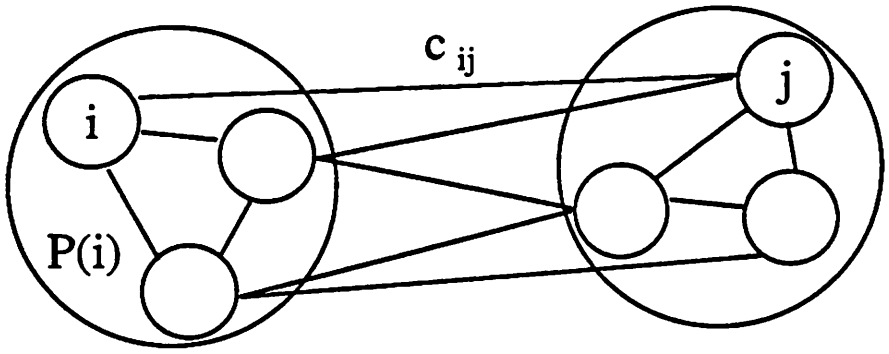


Fig. 2.1 Blocks, clusters and connections.

The number of connections within the clusters is

$$I = \sum_{P(i)=P(j)} c_{ij},$$

the number of connections between different clusters

$$E = \sum_{P(i)\neq P(j)} c_{ij}.$$

An easy way to compare two results of a one-level clustering then is

$$K_1 = \frac{I}{E+I} \times 100\%, \qquad 0\% \le K_1 \le 100\%.$$

This number tells what percentage of connections is no longer visible on the next higher hierarchical level. The criterion is very clear, but it has its shortcomings. Consider the example in Fig. 2.2. Although there is no obvious reason to prefer either of the clusterings, the evaluation criterion shows a bias towards case b.

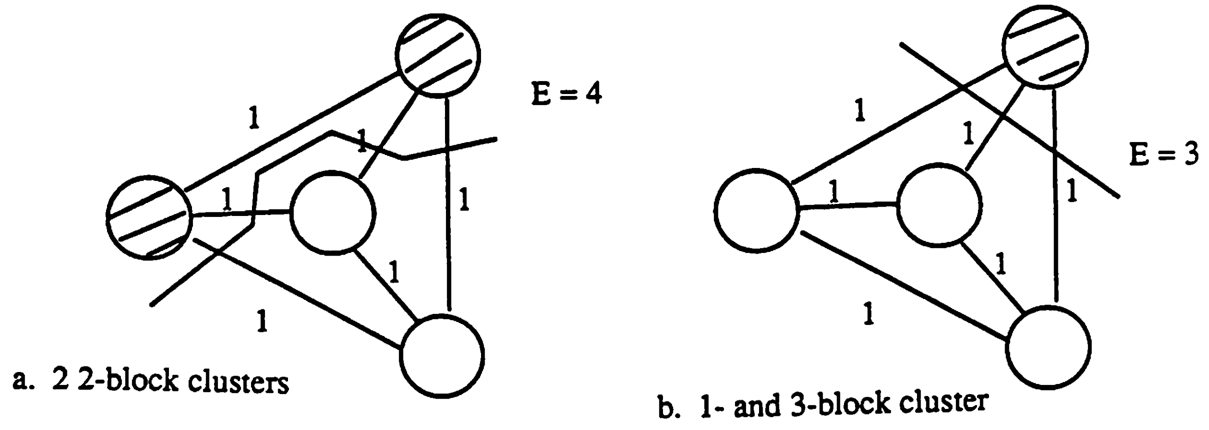


Fig. 2.2 Clusters with different sizes.

In general, clusters of different sizes are always hard to compare based on this approach. Therefore a normalization is necessary. Let n_k be the size of cluster k and m the number of clusters. Then $\sum_{k=1}^{m} n_k = n$ is the total number of nodes. The possible number of connections between nodes within the same cluster is

$$C(I) = \sum_{k=1}^{m} \frac{n_k\,(n_k-1)}{2},$$

the possible number of connections between nodes of different clusters

$$C(E) = \frac{n\,(n-1)}{2} - C(I).$$

The *possible* number of connections is taken, because non-existing edges can be interpreted as edges of weight 0. Then the improved criterion is

$$K_2 = \frac{I/C(I)}{I/C(I) + E/C(E)} \times 100\%, \qquad 0\% \le K_2 \le 100\%.$$

I/C(I) {E/C(E)} is the average weight of an edge connecting a node to another node in the same {a different} cluster.

It does not make much sense to sum up the results on different levels of the hierarchy in one number. It is better to compare the K_1 or K_2 - values of the two clustering trees level by level.

2.2. Geometrical Constraints

A clustering only based on connectivity information can result in a block shape mismatch that makes it impossible for the placement algorithm to avoid big *dead space* areas. On the other hand it is very difficult to decide for a pair of blocks whether they fit together if the other cluster elements are unknown. Even the number of other cluster elements is not yet determined. Therefore heuristics are used.

It is our conjecture that two blocks do not match if

a) their areas and

b) the lengths of their longer sides

are sufficiently different. A very simple implementation then is to prohibit the merging of block pairs whose areas or lengths differ by more than some factor. If one block is much smaller than all the other blocks, that leads to the unfavorable result that this block is left alone until all the other blocks are clustered together.

A better way is to use a threshold area. Only blocks with areas smaller than the threshold are clustering candidates. In that way clusters on each level of the hierarchy tend to become similar in their area requirements. A method to set the threshold based on the statistical distribution of block areas is described in Appendix A.1.

Whatever approach is taken, if the constraints are enforced rigorously, the cluster sizes become smaller and additional hierarchical levels may be introduced. This in turn means fewer degrees of freedom for the placement enumeration and typically leads to worse final results. Because of that it is better to implement the constraints as ''soft'' constraints. They should only be enforced as long as other clustering possibilities are open.

Chapter 3. Placement: A "Meet in the Middle" Strategy

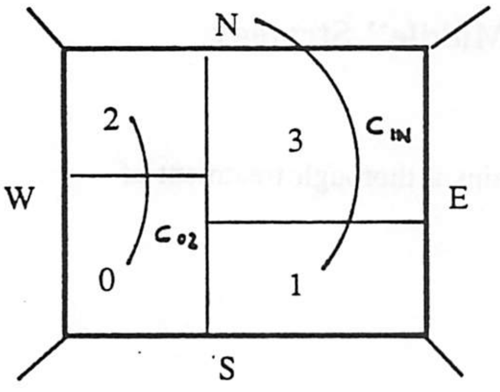
In this chapter routing area is neglected. Chapter 4 contains a thorough treatment of this subject.

3.1. Representation of Connections

To be able to evaluate many different placement possibilities efficiently, the connectivity information has to be represented in a more versatile form than as n-point nets between geometrical pin locations. The accuracy required to obtain satisfactory results is different on different levels of the hierarchy.

Independent of the hierarchical level, nets with n terminals are represented by $\frac{n\,(n-1)}{2}$ connections between every pair of terminals. A spanning tree for this n-clique has $n-1$ edges. Every edge of the n-clique has a probability of $\frac{2}{n}$ to be in the spanning tree. Therefore each of the edges is assigned a weight of $\frac{2}{n}$. That is different from the edge weight for the clustering.

On the non-leaf level, all the connections within the clusters are invisible. The connections between different clusters are assumed to connect to the center point (but see section 3.6). In that way all the connections of current interest can be summed up in one matrix attached to the parent node in the hierarchy. The element c_{ij} in row i and column j of this *connectivity matrix* contains the number of connections between cluster i and cluster j (see Fig. 3.1).



$$
\begin{array}{c|cccc|cccc}
 & 0 & 1 & 2 & 3 & N & W & S & E \\
\hline
0 & & c_{01} & \boxed{c_{02}} & c_{03} & c_{0N} & c_{0W} & c_{0S} & c_{0E} \\
1 & & & c_{12} & c_{13} & \boxed{c_{1N}} & c_{1W} & c_{1S} & c_{1E} \\
2 & & & & c_{23} & c_{2N} & c_{2W} & c_{2S} & c_{2E} \\
3 & & & & & c_{3N} & c_{3W} & c_{3S} & c_{3E}
\end{array}
$$

Fig. 3.1 Cluster of blocks with connectivity matrix.

If a block has a link to another cluster, this link is split up into two links of pertinent weight in the applicable I/O-directions as shown in Fig. 3.2.

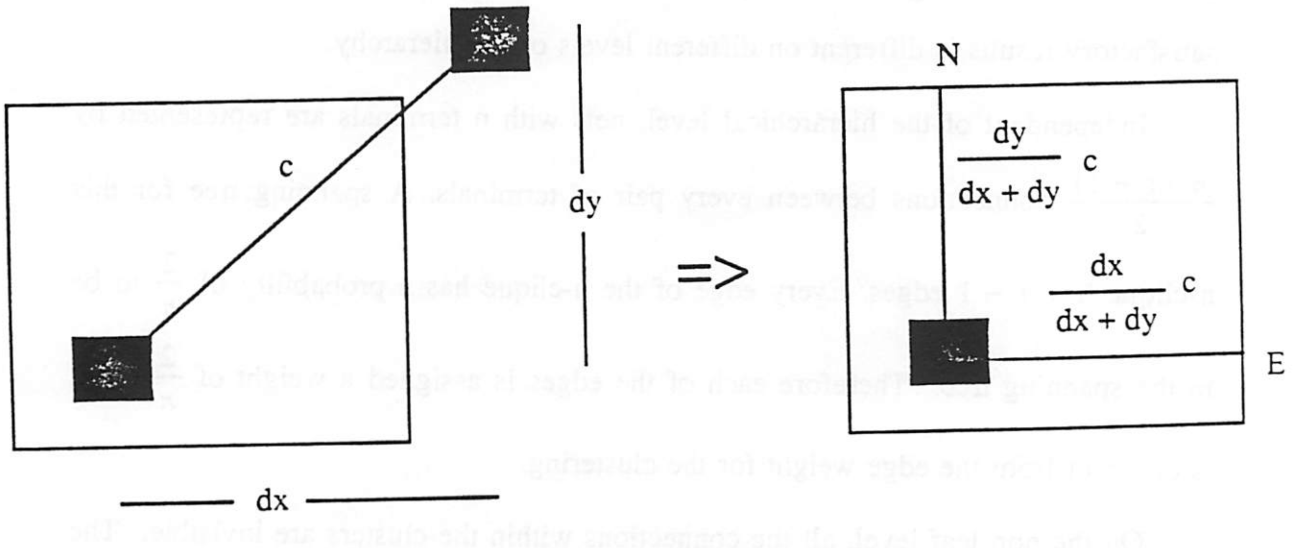


Fig. 3.2 Canonical representation of inter-cluster connections.

On the leaf level this representation is not exact enough since the optimal orientation of a block depends on the actual pin positions. However, it would be very costly to search through a list of all pin positions every time, e.g. a new block orientation is evaluated. Some accuracy has to be given up to gain efficiency. A very convenient scheme is to distinguish only between different pin directions. All the connections summarized in one number c_{ij} in previous levels of the hierarchy are now split up into a set of 4 numbers representing the connections between each of the sides of block i and block j. In this way the data structure does not have to be augmented. As shown in Fig. 3.3, the information can be held by the leaf level connectivity matrices.

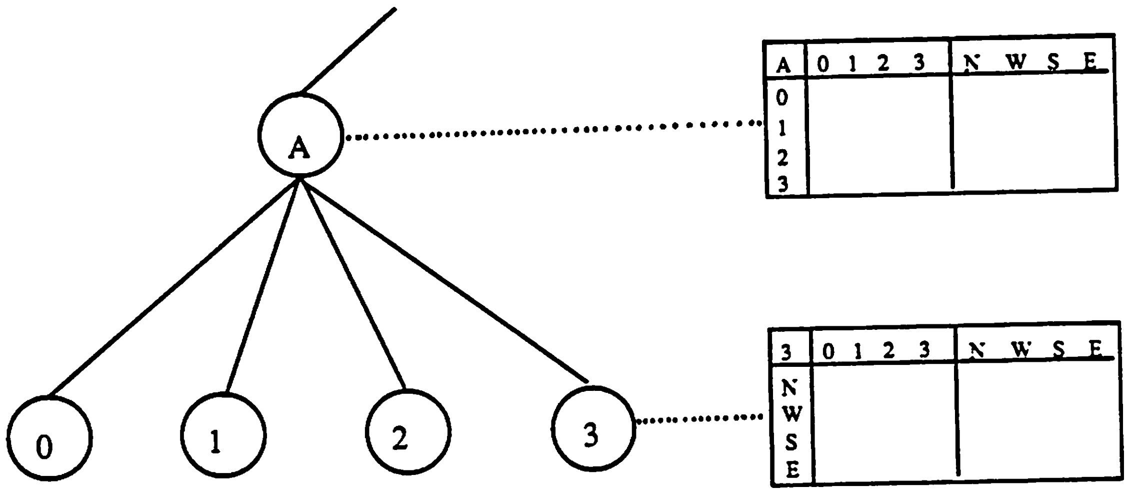


Fig. 3.3 Storage of pin connection information.

3.2. Non-Leaf Level Enumeration

The algorithm assumes that an aspect ratio goal is given or one of the sides has to be of a certain length. I/O pad directions are also assumed to be given, but this is not absolutely necessary: If an I/O connection is assigned a weight of $\frac{1}{4}$ for all four directions, the placement tries to place the connected block somewhere close to the periphery of the chip without a bias as to the direction.

The properties of each basic possible topology are stored in a set of functions, each one of them tailored to perform a specific task for that topology. Fig. 3.4 shows the possible *template* topologies for clusters with 4 elements (templates having 4 *rooms*).

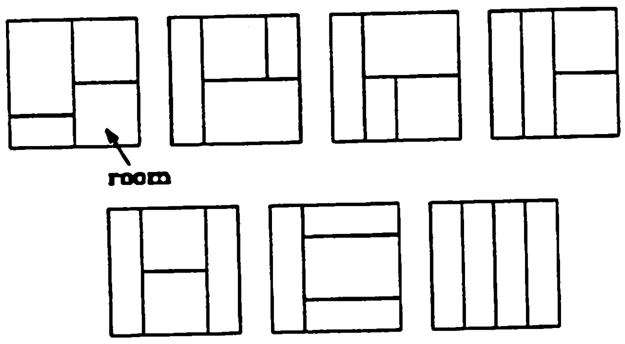


Fig. 3.4 Library of 4-room templates.

On each hierarchical level the algorithm starts with a rectangle whose aspect ratio is determined by the user or the preceding hierarchical level. The area of this rectangle is the sum of the areas of all the blocks that have to be placed in it. The rectangle then is divided into subareas corresponding to the areas of the cluster elements. A particular orientation of a template and an assignment of cluster elements to rooms in the template is considered. An objective function helps to determine a cost value representing the undesirability of the room shapes and the general configuration. This is carried out for all the possibilities (see Algorithm 3.1). The process is recursively repeated for every cluster.

```
non-leaf enumeration
input: area and shape goal, connectivity matrix, $\alpha_a$ ,$\alpha_w$
output: template, template orientation, room assignment

for all templates
    for all template orientations
        for all assignments of cluster elements to rooms
            subdivide the area in the proportion of the areas of the cluster elements
            compute a wire length cost function $f_w$
            compute an area cost function $f_a$
choose the possibility with min: $\alpha_w f_w + \alpha_a f_a$
```

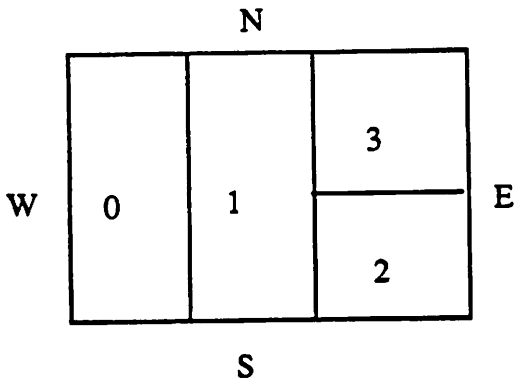
Algorithm 3.1

The number of executions f(k) of the inner loop is strongly dependent on the cluster size k. The number of possible assignments of cluster elements to rooms is k!. The number of possible topologies t(k), i. e. the number of templates and rotational variants of templates, grows exponentially with k. The resulting f(k) is shown in Table 3.1. Currently the cluster size is restricted to 4.

k	1	2	3	4	5
t(k)	1	2	6	22	92
f(k) = k! t(k)	1	4	36	528	11040

Table 3.1 Number of non-leaf topologies f(k) for diff. cluster sizes k.

The function f_w sums up the number of connections between blocks / I/O directions multiplied by their "topological distance". The topological distance is hardwired into the template functions and provides a crude estimate of the desirability of a certain room assignment for minimizing wire lengths (see Fig. 3.5). By computing the inner product with the connectivity matrix from Fig. 3.1 and summing up all the elements of the resulting matrix it is possible to produce the result for f_w very fast. This is indispensable considering the size of f(k).



	0	1	2	3	N	W	S	E
0		0	1	1	0	0	0	2
1			0	0	0	1	0	1
2				0	1	2	0	0
3					0	2	1	0

Fig. 3.5 Example of a template with topological distances.

The function f_a penalizes all the topologies in which the biggest dimension of one of the children exceeds the length of the longer side of its room. Besides that, it also discourages room shapes with aspect ratios much bigger or smaller than 1:1. In general the number of good placement alternatives on levels of the hierarchy will be smaller for extremely long goal shapes.

The overall cost function is a linear combination of the functions f_w and f_a. The weight factors for f_w and f_a are α_w and α_a respectively. The user controls the trade-off ratio $\frac{\alpha_a}{\alpha_w}$ between optimization for minimum area and optimization for minimum wire length.

3.3. Leaf Level Enumeration

On the leaf level the objective functions have to be changed. Additionally, not only the block locations but also their orientations have to be determined. An inner loop has to be added to Algorithm 3.1 in which all the possible block orientations are examined (Algorithm 3.2). Note that some cost computations are independent of the block orientations.

```
leaf enumeration

  ...
  compute a wire length cost function $f_w$
  for horizontal and vertical block orientation
    compute an area cost function $f_a'$
    for all 4 orientations achievable by mirroring
      compute a neighbor penalty function $f_n$
  ...
choose the possibility with min: $\alpha_w \ (f_w + f_n) + \alpha_a \ f_a'$
```

Algorithm 3.2

The wire length cost function stays the same, but a new function f_n that examines the pin positions for blocks with a topological distance of 0, i. e. adjacent blocks, is added.

The area cost function f_a' now has the exact dimensions of the blocks at its disposal. The step 'subdivide the area' from Algorithm 3.1 is no longer necessary in Algorithm 3.2. The shape of the enclosing rectangle obtained by packing the macrocells in the cluster as close as possible for the current topology is compared to the shape goal for the whole cluster set by the higher-level placement decisions. x- and y-dimensions are examined independently. In the following the x-dimension is considered. If the necessary width x_n exceeds the goal width x_g, $(x_n - x_g) \times Y_{chip}$ is added to the area penalty, because in the absence of other overlaps, the chip area will increase by this much (see Fig. 3.6).

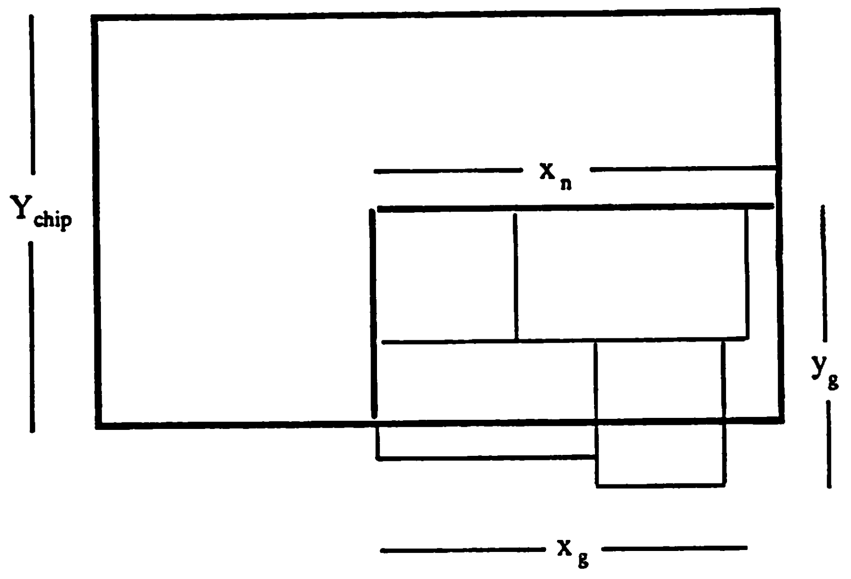


Fig. 3.6 Leaf level area cost function.

In the case that $x_n < x_g,$ $(x_g - x_n) \times y_g / f$ is subtracted from the area penalty, because an overlap in another cluster may be accommodated in this area. Since this is uncertain, the heuristic factor f decreases the effect of this subtraction.

3.4. Target Shapes and Lookahead

In many cases the heuristics for computing f_a on the non-leaf level do not lead to the best solution. It is clear that more information about the desirable shapes must be available on higher levels of the hierarchy to be able to define a better objective function.

The easiest way to do this is to find out about the optimal shape goal for the lowest level clusters and to propagate this information recursively up the clustering tree. The optimal shape goal (*target shape*) of a cluster can be derived by enumerating all the possible topologies to combine the cluster elements. The same procedure as in the top-down enumeration can be used. During the top-down enumeration the target shapes of the clusters can be used like the actual shapes of macrocells in section 3.3.

Some modifications to this scheme are necessary to make it work.

- Target shapes with aspect ratios far away from 1:1 are pretty useless, even if they represent the optimal way to place, e.g. 4 very long blocks with the same width. Strange aspect ratios have to be penalized in the same way as in section 3.2 during the enumeration process.
- If optimal goal shapes are combined, the result of this combination might be far from being optimal. The objective function has to account for this effect. The more dead space a target shape contains, the lower should be its impact on the choice of a placement topology.
- Real-world macrocell layout examples by and large consist of less than 50 blocks. Predictions for future ''superchips'' are in the range of 100 blocks. In any case, with 5 blocks per cluster, 125 blocks can be accommodated in a clustering tree with a depth of 3. The dimensions of macrocells are known, the target shapes on the hierarchical level above the macrocells are optimal. A 1-level lookahead therefore is enough to totally avoid the problem described in the last paragraph (see Fig. 3.7).

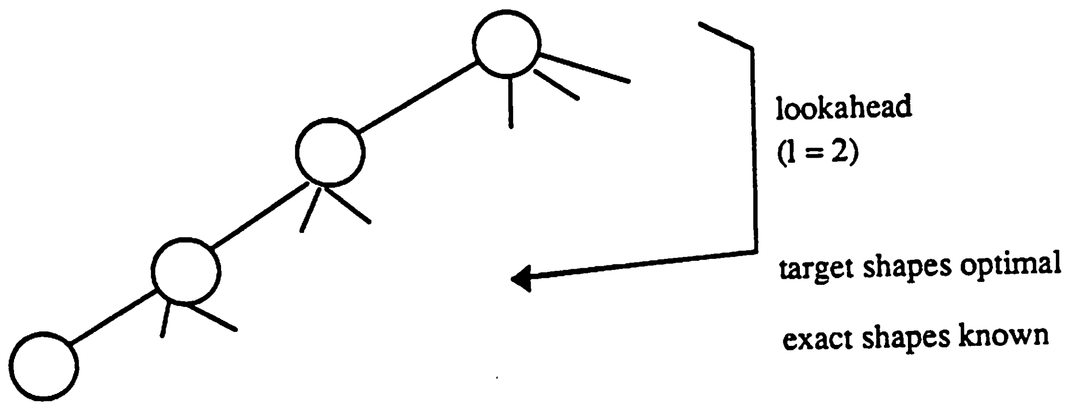


Fig. 3.7 Clustering tree and lookahead.

Although the search tree is shallow, its branching factor turns out to be very high. Under the assumption that the clustering tree is a k-tree, an l-level lookahead (see Fig. 3.7) has a complexity of

$$O\left[\frac{n-1}{k-1}\, f(k)^{l+1}\, k^{l}\right].$$

The formula is derived in Appendix A.2. Although this is still linear in the number of blocks n, f(k) is very large (see Table 3.1) and therefore makes some scheme to speed up the program necessary.

Pruning of the search tree to reduce the effective branching factor can be done with the help of the previous objective function. The disadvantage of that function was that, although it could rule out solutions that were clearly undesirable, it was not able to pick a good solution from the set of solutions not highly penalized. This decision is now no longer necessary. If only solutions within some percentage range of the "optimum" are kept as candidates that reduces the search space. Possibilities that are very unlikely to lead to desirable placements are excluded. Good topologies are not locked out arbitrarily, the number of branches to follow is never restricted. In this way it is made highly probable that the optimal solution within the region bounded by the clustering tree in Fig. 1.2.a is encountered during the search process.

By adjusting the parameter that controls the width of the search, the user can trade off the quality of the final solution against run time on the computer. This is of advantage both in the beginning of the design process, when only a fast upper bound on area and wire lengths is needed and in the end, when the best possible result is to be obtained, no matter how long the program runs, as long as the time is reasonable. In principle the program allows a complete enumeration of the whole solution space given by the clustering tree, by setting the lookahead to d-1 levels where d is the depth of the clustering tree and not specifying any pruning.

3.5. Pseudo-Leaf Level Enumeration

Pseudo-leaf level nodes are clusters with target shapes. They have to be treated differently because of the problems with target shapes described in section 3.4. Since the quality of the target shapes of different cluster elements varies, it is not appropriate to combine them like real macrocell shapes and compare the resulting shape with the goal shape of the parent cluster. Like on the non-leaf level, the goal area is subdivided first. Then the target shape of every cluster is compared to its own goal shape independent of the other cluster elements (see Algorithm 3.3).

```
pseudo-leaf enumeration

    ...
    subdivide the area in the proportion of the areas of the cluster elements
    compute a wire length cost function $f_w$
    for every cluster element i
        for horizontal and vertical target shape orientation
        compute an area cost function $f_a^i$
    ...
choose the possibility with min: $\alpha_w f_w + \alpha_a \sum_i f_a^i$
```

Algorithm 3.3

Because the cluster elements are examined individually, only $2k$ orientation possibilities instead of all 2^k have to be evaluated.

The area cost function f_a^i consists of two parts

$$f_a^i = \beta f_a + (1 - \beta) f_a', \quad 0 \leq \beta \leq 1.$$

As indicated by the indices, f_a is the same as in section 3.2, f_a' the same as in section 3.3. The weighting factor β is a function of the wasted area in the target shape of the cluster under consideration: If a lot of area was wasted in the target shape, it is as good as having no information about it and β is close to 1. If the target shape does contain information that is valuable in guiding the placement into a direction that makes lower level decisions better, β is close to 0.

Besides that the penalty contributed by f_a is much lower than that contributed by f_a' for an equal geometrical constellation. In that way the placement is biased to provide adequate goal shapes for clusters of which good target shapes are known; clusters for which nothing better is known must take whatever shape the context assigns to them.

Because the clusters behave like "soft blocks", it is not indispensable to match all the target shapes on the first level. To mediate between the aspect ratio provided by the user at the root of the tree and the dimensions of leaf blocks, the target dimensions only have to latch on to desirable room dimensions on some intermediate level of the hierarchy.

3.6. Intra-Level Dependencies

In the way the algorithm was described up to now, after the placement for one level of the hierarchy was done, all the clusters on the next level of the hierarchy are processed independently in an arbitrary sequence. This can lead to undesirable results for the wire length as indicated in Fig. 3.8, because the center points of the parent clusters are taken as reference point for all the connections between the clusters.

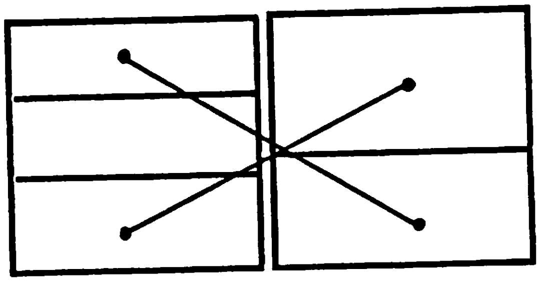


Fig. 3.8 Mutual dependency of room assignments.

It was not attempted to find an optimal solution for this case, but one that was practical from a computational point of view and would work well for most of the cases. First of all it is clear, that once one cluster was placed, the center points of the cluster *elements*, which now have a fixed location, should be used to compute the edge weight of I/O connections of other clusters (Fig. 3.2 and 3.3). This is easily achieved by attaching a 'placed'-bit to every cluster. When the I/O-goals for a cluster are computed, all the connections to other clusters are iterated. If the connected cluster is already placed, its center point is taken as reference point. If a cluster is not yet placed, the center point of the parent is examined. During the lookahead it is possible that the examination of parent clusters is repeated recursively until an element is found of which the position is already known.

This procedure introduces an ordering dependency. The later a cluster is placed, the stronger is the influence of already completed partial placements. Therefore the cluster with the largest area is placed first. Alternatively the cluster with the longest common boundary with other clusters could be taken first.

All the clusters, of which the parents are placed, are kept in a queue data structure. So this is simply a matter of sorting the queue whenever a new hierarchical level is started.

3.7. Analysis

It is somewhat artificial to evaluate a placement without the routing. Nonetheless some measure of performance must be defined to be able to show the effect of the various procedures introduced in this chapter. Two figures of merit are needed to compare the effects with respect to area and wire length minimization.

The figure of merit used for the first goal is *area utilization*. It is defined as the proportion of the sum of all block areas to the area of the enclosing rectangle for the placement result. The estimate for achievements with respect to the second goal is the *sum of net half perimeters* (although without routing area no net could actually be implemented). The *maximal net half perimeter* is not explicitly optimized in the implemented algorithm and only shown for comparison purposes.

	original version			with target shapes		
α_a / α_w (normal.)	area util.	sum of net half perims.	max. net half perim.	area util.	sum of net half perims.	max. net half perim.
0.1	76.02%	10497	354	77.98%	10412	350
1	77.58%	10188	350	84.12%	10795	335
10	79.63%	10719	346	87.23%	11441	328

Table 3.2 Experim. results (33-block example) with diff. cost function parameters.

Table 3.2 supports two assertions:

- The behavior claimed in Fig. 1.1 really shows up. As soon as the weight of the wire length cost function goes above a certain threshold, the dimensions of the circuit due to additional dead space have grown so much, that the gain in relative positions of blocks is overcompensated by the additional wire length needed to bridge the gaps between blocks.
- Target shapes help the objective function to find out rectangle dissections that lead to a better match between goal shapes and block shapes on the lowest level of the hierarchy. Up to five blocks per cluster should offer so many degrees of freedom for combinations that the objective function usually finds useful target shapes.

Even though the target shapes are better than the original heuristic, they may be misleading since in the bottom-up shape enumeration the eventual context, in which the cluster will be placed, is unknown. In these cases and of course in those cases where target shapes include too much dead space, a lookahead can help to avoid unfavorable template choices. Table 3.3 gives some experimental results for a 1-level lookahead employed in the placement of a 33-block example. Originally a complete search was done, the result appears in the last line of Table 3.3. Then only branches within some percentage of the best cost value on the current level were considered on the next level of the hierarchy. This percentage, the *pruning threshold*, was decreased until only the best solution on the current level was left. In this case the result is the same as if no lookahead were specified.

pruning threshold	area utilization	sum of net half perims.	max. net half perim.	elapsed time (normalized)
0%	87.23%	11441	328	1
50%	88.59%	10937	324	2.07
200%	90.09%	10607	339	4.67
∞	90.09%	10607	339	75.56

Table 3.3 Experimental results (33-block example) for 1-level lookahead.

After restricting the search space to approximately 6% of its full size by cutting away all solutions whose cost function on the current level is more than 200% larger than the local optimum, the globally optimal solution is still found. After decreasing the search space further, at least a solution that is better than the original one was detected.

Chapter 4. Routing Area Estimation

4.1. Motivation

If the technology provided a spare couple of interconnection layers above the macrocells, placement and wiring would be independent of each other, the block packing results as shown in the last section would be satisfactory. Since typically this is not the case, routing area has to be interspersed between the blocks.

Even when the spare interconnection layers are available, it might be necessary to provide space between the blocks. After the first metallization layer is deposited on the chip, the chip surface is quite ragged. To avoid problems of electromigration, the second layer of metal typically has to follow other design rules with wider wires and more space between the wires. Contacts between one of the active layers and metal2 also need more space. As a result, there might not be enough space to complete the routing on the layers above the functional blocks.

The area needed for a chip basically consists of the 3 components block area (BA), routing area (RA) and dead area (DA) [Ued85]

$$A = BA + RA + DA.$$

The classic objective for the placement is to minimize the total wire length (RA). An algorithm that tries to minimize the area (A) should only increase RA if this can be compensated by a larger decrease in DA [1].

1 It is assumed that the block area (BA) is constant.

If the placement does not include an explicit estimation of the routing area, its results will suffer from the fact that it cannot distinguish between dead space that later can be used for routing and dead space in DA that can lead to an increase in routing area because the blocks are further apart from each other.

A second reason why routing area estimation *during* the placement is necessary rather than including the routing area *after* the placement, is shown in Fig. 4.1. A placement that was optimized to have a rectangular shape of a given aspect ratio is probably neither rectangular any more, nor does it conform to the desired aspect ratio after routing area was added.

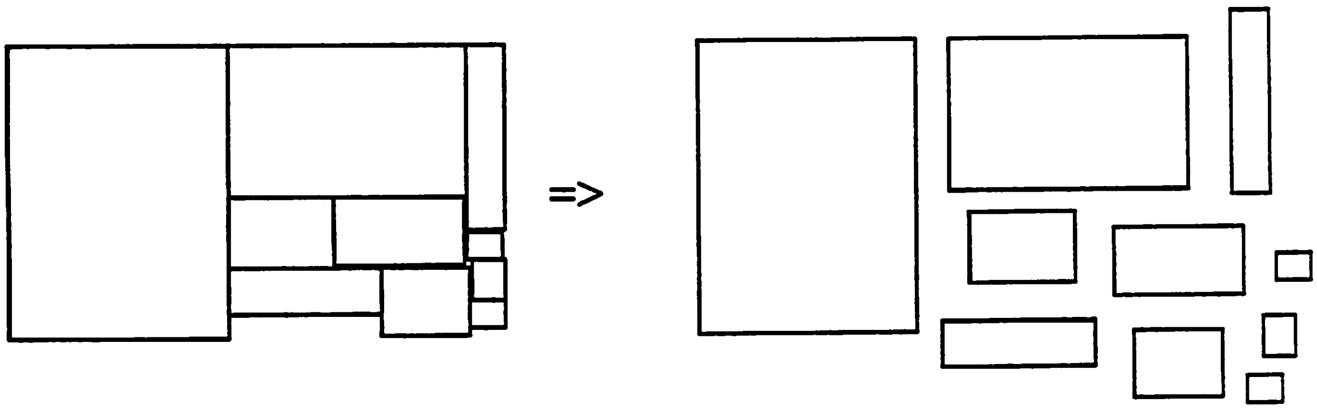


Fig. 4.1 Scenario for a placement before and after routing area estimation.

4.2. Previous Approaches

It was seen earlier that the placement has to take care of the routing area as well in addition to block areas and the dead space between them. However, most placement programs do not deal with this problem. There are several reasons:

- The problem of placing arbitrarily sized rectangles in a minimal enclosing rectangle in an optimal way is already difficult enough; it is NP-complete [Sha85].
- It is not enough to just estimate the routing *area*. For the estimation of dead space and to get a rectangular shape of the final layout, the location of the routing space also has to be determined.
- A good estimate for the space needed for routing has to have some information about the route a particular wire takes. The exact information is not available before the detailed routing is done, an approximate information that would be good enough is produced by the global routing. To compute an objective function for a placement in a program then would require global routing every time the objective function has to be evaluated. This is clearly not feasible because of its computational cost.
- If a program has a way to estimate the location of interconnection area and the estimate is not exact enough, that might mislead the objective function so much that the result becomes worse than without routing area estimation. This was true for an early version of the program described here, that introduced a lot of unnecessary dead space.

There are, however, some approaches that - with different degrees of sophistication - tackle the problem. In [Sec85] and [Wip85] heuristics are applied to each block before the placement to derive a hypothetical block shape including some routing area based on the number of pins of the block. In [Che83] a pseudo-routing of pairs of modules is performed to derive the necessary distance between the two modules in a row-based placement. Both solutions fail to account for connections outside of the immediate neighborhood of their terminals.

For gate arrays, Burstein et. al. introduced an algorithm that merges placement and routing in a hierarchical fashion [Bur83]. Because of the simple array structure, this grid-based approach is feasible. In [Sze86] a simultaneous placement and global routing of restricted slicing structures was proposed. A method to generate the global routing at the same time as the placement, suited to the more general topologies used in this work, was described in [Dai87]. In both cases no attempt was made to estimate the routing area based on the global routing information. Because of their representation of global routing paths as links between the center points of adjacent blocks, that would not be straightforward.

More literature exists on the topic of routing area estimation after the placement is finished. Statistical approaches [Hel78, ElG81] use a Poisson model for the generation of wires along block edges and assume an exponential distribution of wire lengths. They do not take actual pin positions or detailed connectivity information into account. Therefore their usefulness for the problem described in section 4.1 is questionable. Most programs perform a global routing after the placement is finished and then estimate the necessary routing area [Lau79, Fow85, LaP85].

4.3. Top-Down Space Allocation

Besides using the hierarchical decomposition of the problem, the basic idea is to avoid a dynamical shortest path or Steiner tree determination by precomputing the paths for the finite number of templates and storing the information in the library of floorplan templates.

For every template and for each connection between blocks / clusters / I/O-goals all the channels on the shortest topological paths are marked with a probability. This probability represents the likelihood that the connection will really pass through that channel (see Fig. 4.2). p_{ij}^{kl} is the probability that a connection between i and j will pass through the channel between k and l.

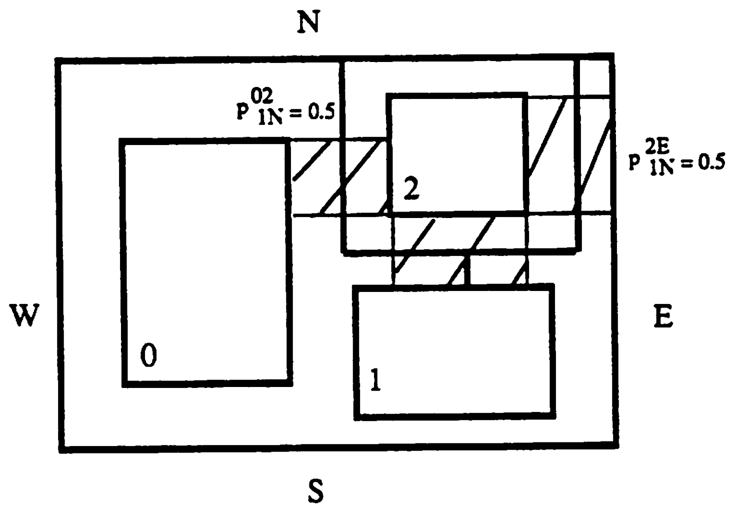


Fig. 4.2 Hierarchical routing area allocation.

Then the required normalized "channel" width $\frac{s_{kl}}{w}$ ("channels" on higher levels consist of many real channels; w is the design-rule dependent track width) can be estimated as

$$\frac{s_{kl}}{w} = t_{kl} \sum_i \sum_j p_{ij}^{kl} c_{ij},$$

where c_{ij} is the pertinent element of the connectivity matrix and t_{kl} a heuristic factor that accounts for track sharing. This requires exactly the same operations as the computation of the topological wire length cost function f_w from section 3.2, although it has to be done for all the channels (*bottlenecks* [Che83]) in a template.

The storage requirements for the library of parameters are acceptable (see Appendix A.3.). Thus it can be done very fast in the inner loop of Algorithm 3.1, 3.2 or 3.3 before the area penalty $f_a \,/\, f_a'$ is evaluated. In this fashion routing area is treated equivalently to block area. It not only influences the choice of templates on the current level of the hierarchy, but also the shape goal for the next level.

The estimation takes advantage of all the information gathered about the position and connectivity of clusters of blocks up to that level. Earlier in the process space for global connections between different clusters is provided, later on more of the internal connections within the clusters become visible.

The allocation of space along the shortest path makes the job of a global router easier, but does not constrain it in doing whatever is recognized as optimal after the complete topological information produced by the placement is available. Besides that, this approach is very flexible; for "over-the-block wirable" cells [Ued85] the probabilities can be easily adjusted.

Before refining the basic idea, it seems appropriate to describe how the numbers p_{ij}^{kl} and t_{kl} are derived. In general the numbers to be assigned to p_{ij}^{kl} are pretty obvious as in Fig. 4.2, since most of the time only two distinct shortest paths exist. For fine-tuning, statistics can be compiled that characterize the behavior of the global and detail routers used. It is likely that different routing algorithms will yield slightly different results for the parameters. It is one of the strengths of this approach that without changes in the program it can be applied to different design technologies and methodologies.

For the non-leaf levels it is assumed that the connections leave the clusters on the side that is closest to the end point of the connection. In Fig. 4.2 that means that a connection between blocks 0 and 2 would leave the right side of block 0 and enter the left side of block 2. It is the task of the next lower level to provide the space to get to this side. On the leaf level, this is no longer possible. In this case the pin position information already needed for the determination of block orientations comes in handy. For a given block orientation, if needed, additional space has to be provided along the sides of the block to bring the wires around the block to the location that is closest to the end point of the connection.

4.4. Bottom-Up Estimation

Due to the nature of the objective function used, area taken away from the cluster area to account for the space needed by the wiring must have been added beforehand. Instead of just assigning the sum of the areas of the children to a parent node in the clustering tree, a routing area estimate has to be included as well. Note that at this stage it is not necessary to know the routes that connections take (that would be impossible anyway because the placement is not done yet), only the approximate *area* needed for connections has to be estimated.

The task of predicting the space needed for routing before invoking a floorplanner or placer has gained some attention recently [Kur86, Che88, Zim88]. Unfortunately the reported results applicable to the macrocell layout style are not very encouraging: Errors of 20% (of the whole layout area, much more if the known block area is excluded) seem to be the current state of the art ([Che88], [Zim88]). This would be unacceptable to the area penalty functions described in sections 3.2 and 3.3 and would lead to very strange results.

Fortunately a bit more information to estimate the routing area is available here. After having done the clustering on one level of the hierarchy, we know all the connections between the cluster elements and from cluster to cluster. On the lowest level of the hierarchy the number of pins along the 4 sides of a block is known as well. When the optimal target shape is chosen, a reasonable way of arranging the blocks topologically is generated as a byproduct. All this information can be used to produce a routing area estimate that exactly mirrors the more sophisticated top-down routing area estimate with respect to its hierarchical decomposition.

Since the exact constellation that tries to minimize the number of connections between non-adjacent blocks / clusters is not known, a worst case approach is taken by building a hypothetical connectivity matrix $\bar{C}$ with identical connection strengths $\bar{c}_i$ and $\bar{c}_e$ for all intra-cluster and inter-cluster connections respectively. That means that all $\bar{c}_{ij}$ are set to $\bar{c}_i$ for $j \in \{0,...,k\text{-}1\}$ and to $\bar{c}_e$ for $j \in \{N,W,S,E\}$.

$$\bar{c}_i = \frac{2}{k\,(k-1)} \sum_i \sum_{j \in \{0,\ldots,k-1\}} c_{ij}$$

$$\bar{c}_e = \frac{1}{4\,k} \sum_i \sum_{j \in \{N,W,S,E\}} c_{ij}$$

With this connectivity matrix and the target shape topology the top-down wiring space allocation function is called. The exact distribution of the wiring space generated by that function is of no interest here, the only thing that is extracted is the total area of the space needed for routing. It is not necessary that in the top-down placement process the same topology is chosen for the routing area estimate to be reasonably close, although that helps of course and really happens on lower levels of the hierarchy if the wasted area in the target shape is small. The routing *area*, contrary to its exact allocation, is very similar for different good topologies, i. e.

topologies without too much dead space.

On the leaf level the block areas are inflated in both dimensions based on the number of pins of the module in each direction.

In the final result connections between non-adjacent blocks tend to be weaker than connections between adjacent blocks. Therefore the initial routing area estimate is reduced by some heuristic factor. If with the default value the routing area is consistently over- or underestimated, it can be adjusted by the user.

Table 4.1 provides some evidence that the method described yields a satisfactory match between the routing area estimated in the bottom-up phase of the program and the routing space allocated top-down. In none of the cases the default value referred to in the last paragraph had to be adjusted.

			aspect ratio 1 : 1		aspect ratio 1 : 1.5	
example	level	# clusters	area ratio bu/td	std. dev.	area ratio bu/td	std. dev.
primBBL1	0	1	1.000		1.005	
	1	3	1.045	0.028	0.998	0.062
primBBL2	0	1	1.000		0.986	
	1	3	0.994	0.022	0.990	0.014
	2	9	1.008	0.120	1.014	0.104

Table 4.1 Comparison between bottom-up (bu) and top-down (td) estimation.

4.5. "Fuzzy" Rectangle Dissection

The function 'subdivide the area' in Algorithm 3.1 becomes somewhat more involved now. Its task is, given X, Y and the area of the cluster elements a_i (see Fig. 4.3), to generate the shape goals x_i and y_i for the cluster elements. If $X \times Y$ is equal to $\sum_i a_i$ plus the routing area given by the s_{kl}, that is easy because then a greedy algorithm is successful. Starting with $y_1 = Y$ in Fig. 4.3, all the unknowns can successively be computed.

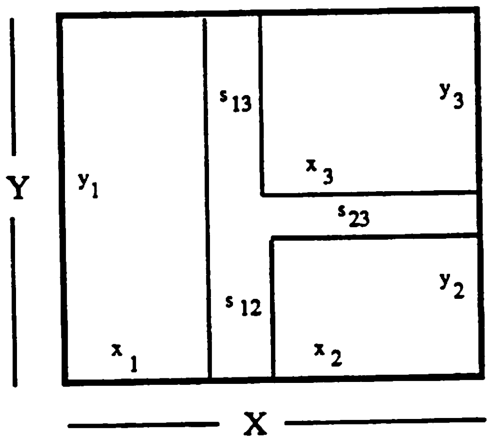


Fig. 4.3 Example problem for rectangle dissection with routing area.

If the bottom-up routing area estimation is not 100% exact, the last cluster element gets all the space that is left, even if that space is negative. For the area cost computation to work right it would be better to reduce or increase each side of all the cluster elements by the same factor r. Under the assumption that the area provided is too small, r has to be smaller than 1. Then the rectangle dissection problem for the template of Fig. 4.3 can be represented in the following form:

MAXIMIZE r

$$X = r\,x_1 + \max\,(s_{12} + r\,x_2, s_{13} + r\,x_3)$$

$$Y = \max\,(r\,y_1, r\,y_2 + s_{23} + r\,y_3)$$

$$x_i\,y_i = a_i, \quad i = 1, 2, 3$$

Since there is no closed solution even for the simple case shown, an iterative method is employed. First the greedy algorithm is applied ($r^0 = 1$); a_3 therefore does not fit into the space left for the last room. If in the j-th iteration the area difference between the last cluster and the last room is $\Delta\,a^j$, r^{j+1} is set to

$$r^{j+1} = (1 - \frac{\Delta a^j}{X\,Y})\,r^j.$$

In the next call of the greedy algorithm, the cluster element areas are set to

$$a_i^{j+1} = (r^{j+1})^2\,a_i.$$

The stopping criterion for the iteration is

$$\Delta a^j \leq 0.02\,a_3.$$

It can be proven that this kind of iteration converges for all the templates and in fact, it converges quite fast because the initial guess $r^0 = 1$ is very good and the stopping criterion is not very demanding.

4.6. Advantages and Shortcomings

To analyze the behavior of the proposed routing area allocation scheme, consider the example in Fig. 4.4 and specifically the routing paths provided for the connection from 1 to 2.

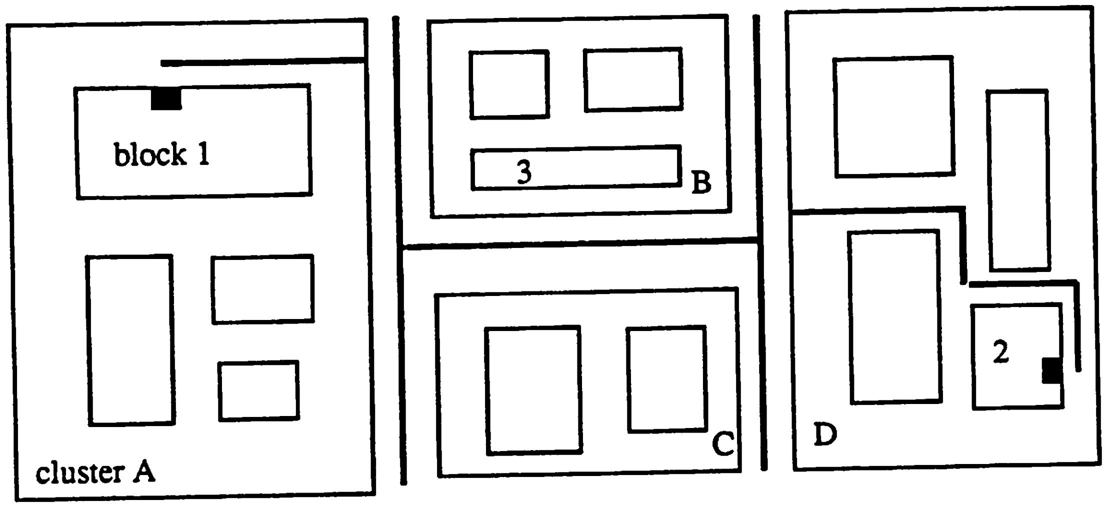


Fig. 4.4 Placement example with routing paths.

The picture only gives a conceptual overview, lower probability paths are not included and the paths shown differ with respect to their probability. That means that for a given number of connections between block 1 and block 2, different numbers of tracks are assigned to different "channels" on different paths. In spite of this, several properties of the algorithm become evident:

- Because the internal structure of clusters is unknown on higher levels of the hierarchy, space for global connections is not provided in "cluster feedthroughs" (like in cluster B). This disadvantage is not too serious, because if the global router decides to use the feedthrough in cluster 3 for the connection shown, that merely makes a local move of block 3 towards cluster C necessary. The main property is that the space for the horizontal connection between clusters A and D *was* taken into account and influenced the choice of the templates so as to accommodate the connection without making block moves outside of the enclosing rectangle necessary (like in

Fig. 4.1) or having to change the relative placement.

- Because the space generated between cluster B and D is eventually added to all the channels along this boundary, non-leaf level estimates tend to be too high. The algorithm accounts for this fact by normalizing the spacing requirements on different hierarchical levels with different constants.
- Another disadvantage that cannot be seen in Fig. 4.4 is that blocks that fit together by abutment cannot be distinguished from other blocks having many connections between adjacent sides. That is a general limitation of the simplified representation of connections introduced in chapter 3.1. Since abutment is very special and given the savings in CPU-time with the simplified representation of terminal locations, it is excusable that too much routing space is provided for that case.

Chapter 5. Sensitivity

5.1. Introduction

Since CAD programs do not produce a global optimum solution for most problems, incremental changes in the problem description should not yield solutions that are radically different from the original ones. Small changes in the input occur quite frequently. To overcome functional problems, connections are changed, added or removed; to optimize the timing of the circuit or to lower heat dissipation in critical areas, sizes of transistors are changed. If the CAD program is very sensitive to these changes and produces a totally different layout, this might defeat the purpose for the input modifications, possibly causing further changes in the input, requiring many iterations until the solution is stable again (see Fig. 5.1). On the other hand the algorithm should be able to react to significant changes in the input I, otherwise it could not find at least near-optimal solutions. Furthermore, it should be sensitive to the control input C, otherwise a designer could not influence the program to consider his particular requirements.

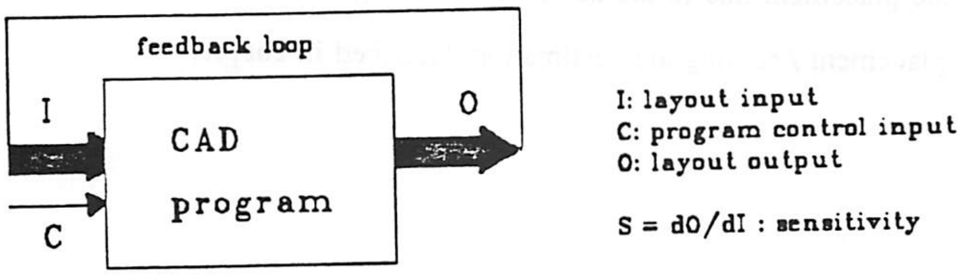


Fig. 5.1 Symbolic representation of automatic layout processes.

The approach taken in the presented placer can in its major parts be followed by a floorplanner as well. If floorplanning and placement were treated independently and the result of the placement was significantly different from the floorplanning result, the shape optimization for soft blocks would have been done for the wrong topology and it might become necessary to repeat the block generation or design process.

Maintaining the global layout without sacrificing the ability to optimize it, is possible by taking the original clustering tree again for the changed input data. The clustering tree still allows the placement to adapt to changes in the design in various ways, whereas a binary slicing tree is already quite restrictive. The sensitivity can be controlled in still another way: When looking for the minimum of the objective function, a better solution is only accepted if its cost is *significantly* lower than that of the best solution up to date. What is significant, is determined by the user, according to how important this issue is for him. This introduces a bias towards the solutions produced earlier, so the templates are enumerated in a sequence corresponding to their general "desirability".

Changes in the placement due to the addition of routing area are minimized by the simultaneous placement / routing area estimation described in chapter 4.

5.2. Analysis

To be able to evaluate the effectiveness of various measures to decrease the sensitivity S, the formula for S has to be changed to $S = \frac{\Delta O}{\Delta I}$ and input and output changes have to be clearly defined.

ΔI^C can be defined as the proportion of nets that are changed, added or deleted to the total number of nets[1]. Similarly ΔI^B can be defined as the proportion of area changes to the sum of all block areas[1]. To avoid that positive and negative area changes cancel each other out, they both have to be added to ΔI^B with a positive sign.

Since the program under consideration is a placement program, ΔO is measured in terms of deviations from the original block positions: For the origin always lying in the lower left hand corner of the chip, ΔO can be defined as

$$\Delta O = \sum_i ((x_i - x_i')^2 + (y_i - y_i')^2)$$

where (x_i, y_i) is the original position of block i and (x_i', y_i') is the position after the changes. The argument to use the squares of a'l distances is to make a few large movements count more than many small ones.

1 If the changes are small, the denominator can be treated as being constant.

It is practical to split up the sensitivity into two parts: S^C $(= \frac{\partial O}{\partial I^C})$, the sensitivity for changes in the connectivity input and S^B $(= \frac{\partial O}{\partial I^B})$, the sensitivity for changes in the block area input[2]. Symbolically this can be written as

$$d\,O \;=\; S^C\,d\,I^C + S^B\,d\,I^B,$$

where $d\,I^C$ and $d\,I^B$ represent the respective input changes.

The following Table 5.1 shows the effect of some sensitivity reduction mechanisms of section 5.1. The results in the first row were obtained by changing two connections ($\Delta\ I^C$), the results in the second row by changing two block shapes ($\Delta\,I^B$).

clustering tree	no constraint	original tree		
significance threshold for changes	none (0.00)		0.01	0.05
$\Delta\ O$ (1 net deleted, 1 changed)	596	0	0	0
$\Delta\ O$ (2 blocks resized)	2988	2988	2154	504

Table 5.1 Effect of some sensitivity reduction mechanisms (primBBL2).

2 Other sensitivities - like the sensitivity for block shape changes - could be readily incorporated as well, but are not considered in this report.

Chapter 6. Examples and Results

The program has been tested with several examples. The placement results are compared with those obtained by the BBL program [Che83] and MOSAICO [Cas86] for the two benchmark examples for macrocell layout used at the 1988 MCNC International Workshop on Placement & Routing.

Table 6.1 shows the results for primBBL1, Table 6.2 the results for primBBL2. The numbers are normalized with respect to the BEAR placement, e. g. the column 'area' is obtained by computing the ratio of the total layout area for the program in question and the area obtained with the BEAR placement. Actual layout plots can be found in Appendix B.

The time needed to obtain the placements by the BEAR placement program was 131 seconds for primBBL1 and 762 seconds for primBBL2 (elapsed time on a VAX 8800 with a workload between 10.5 and 12.5).

example primBBL1 after routing	area	sum of wire lengths	number of vias
BEAR	1.000	1.000	1.000
BBL	1.077	1.179	n/a
MOSAICO	1.019	1.026	1.308

Table 6.1 Comparison of results for the 10-block example primBBL1.

example primBBL2 after routing	area	sum of wire lengths	number of vias
BEAR	1.000	1.000	1.000
BBL	1.047	1.211	n/a
MOSAICO	1.117	1.157	1.019

Table 6.2 Comparison of results for the 33-block example primBBL2.

Table 6.3 helps to determine how well the routing area estimation predicts the area actually needed by the routing. The areas in the first column are given by the output of the placement program, the results in the second column reflect the final areas after the global and detailed routing were completed. The routing area needed by the ring router is not included because it is not taken into account in the routing area estimation during the placement either.

example	area after placement		area after routing (core)		Δ %
primBBL1	2384	(40.4×59.0)	2470	(41.3×59.8)	+3.6
primBBL2	235.3	(15.9×14.8)	246.1	(16.3×15.1)	+4.6
apte	5060	(74.9×67.6)	5560	(73.9×68.4)	-1.2

Table 6.3 Mismatch between estimated and actual layout areas.

Chapter 7. Possible Extensions

All the results described were obtained with clusters of a maximal size of 4. It is not advisable to run the program with only a very small subset of 5-room templates, because as soon as 5 elements per cluster are allowed, the clustering makes extensive use of this opportunity. It is a time-consuming but otherwise straightforward process to add the 5-room templates to all the sets of template specific functions and data structures.

The addition of 5-room templates would have two positive consequences:

- New topological possibilities not achievable with the current set of structures would be made possible by the use of non-slicing templates.
- For certain numbers of blocks the number of hierarchical levels could be decreased, making the information available at the higher decision levels more accurate.

However, there would also be a major disadvantage: As can be seen in Table 3.1, the time needed to decide about the placement of a 5-block cluster is about 20 times higher than the time for a 4-block cluster.

A possibility to improve the block shape information that is propagated up is to replace the single target shape with a set of target shapes, i. e. a shape function [Lau88]. In that case no more lookahead step would be necessary. The top-level goal shape could be chosen from a finite set. Shape functions can be combined efficiently for slicing structures [Ott83]. For non-slicing structures however, it seems that there is no way to find the composite shape function other than enumerating all the possibilities[1]. If 5 clusters with 10 possible shapes each had to be combined in a non-slicing structure, the number of possibilities would be huge ($10^5 \times 11040 > 1$ billion). The problem could be avoided by restricting the non-slicing structures to the leaf level of the hierarchy.

Even though as indicated in section 3.6 there are some dependencies within a hierarchical level, the algorithm lends itself to an implementation on a multiprocessor. At least the lookahead could be done in parallel for the different cluster elements without many modifications. A substantial increase in speed could also be achieved for the leaf level block orientation determination.

1 In [Sto83] it was proven that finding the optimal orientations of blocks for a given floorplan is NP-complete for non-slicing structures. Because the problem of finding the shape function - lower area bound of all possible rectangles - of a cluster involves finding the optimal orientations of its children, that problem is clearly NP-complete as well.

Appendix A. Supplementary Material

A.1. Threshold Value for Clustering Constraints

The distribution of block areas may be arbitrary with a minimum block area $a_{\min}$ and an average block area μ_a. The threshold value a_{th} is set to $a_{th} = a_{\min} + \Delta a$. The Markov inequality states that for arbitrary distributions u(X) > 0

$$P[u(X) \geq c] \leq \frac{E[u(X)]}{c}, \qquad c > 0.$$

The random variable X is identified with the block area a, $u(X) = a - a_{\min}$, $c = \Delta a$. Hence

$$P[a - a_{\min} \geq \Delta a] \leq \frac{\mu_a - a_{\min}}{\Delta a}.$$

For a given distribution, $c_1 \leq 1$ can be introduced so that

$$(1) \quad P[a - a_{\min} \geq \Delta a] = c_1 \frac{\mu_a - a_{\min}}{\Delta a}.$$

In any clustering step, at least k blocks should be available for clustering (k = maximal cluster size, n = total number of blocks)

$$(2) \quad P[a - a_{\min} < \Delta a] = 1 - P[a - a_{\min} \geq \Delta a] \;\; \geq \frac{k}{n}.$$

With the introduction of $c_2 \leq 1$ representing the relative importance of similar block areas in a cluster, (1) and (2) can be converted to

$$1 - \frac{k}{c_2 n} = c_1 \frac{\mu_a - a_{\min}}{\Delta a}$$

$$\Delta a = c_1 \frac{\mu_a - a_{\min}}{1 - \dfrac{k}{c_2 n}}.$$

c_1 and c_2 can be user inputs to control the clustering process.

Example: n = 13 blocks, k = 5 blocks / cluster

minimum area $a_{\min} = 20$, average area $\mu_a = 100$

$c_1 = c_2 = 0.5 \quad \rightarrow \quad \Delta a \approx 170, a_{th} \approx 190.$

Because the matching of block areas is most important on lower levels of the hierarchy, c_2 can be changed on every level i to yield a weaker Δa-restriction, e. g. $c_2^i = r \times c_2^{i-1}$.

A.2. Complexity of Lookahead Search

Let n be the number of blocks, k the number of elements per cluster, f(k) the number of topological possibilities as in Table 3.1. For a node in a k-tree f(k) possibilities have to be enumerated (lookahead l = 0). For each of the possibilities, each cluster element itself has f(k) possibilities to be placed. Therefore

$$k\ f(k)$$

possibilities have to be examined for a given goal shape. If the effect of the leaf level, where the lookahead cannot go on, is neglected, for a lookahead depth of l levels,

$$f(k)\ (k\ f(k))^l$$

computations of the objective function are necessary.

A k-tree of depth d contains

$$n = k^d$$

leaf nodes and

$$1 + k + \cdots + k^{d-1} = \frac{k^d - 1}{k - 1}$$

non-leaf nodes (clusters). Hence the complexity is bounded by

$$\frac{k^d - 1}{k - 1} f(k)(k\, f(k))^l = \frac{n - 1}{k - 1} f(k)^{l+1} k^l .$$

A.3. Storage Space for Routing Area Estimation Parameters

Every bottleneck kl in a template with r rooms requires the storage of

$$(1)\quad \frac{r\,(r-1)}{2} + 4\,r + 1$$

numbers for the probabilities p_{ij}^{kl} and for t_{kl}. The following two theorems help to determine the number of bottlenecks in a template:

Theorem: In every floorplan graph[1] with r rooms there are $v = 2r + 2$ vertices.

Proof: Every room is a rectangle bounded by 4 corners. Each corner is the site of a node of the floorplan graph. Seeing that all the nodes[2] (except for the 4 nodes at the corners of the floorplan) are of degree 3, every node is in the corner of exactly 2 rooms (see Fig. A.3.1). The 4 corner nodes are only in the corner of one room. Hence

$$v = \frac{4\,r - 4}{2} + 4 = 2\,r + 2.$$

1 Floorplan graphs are defined in [Dai87].

2 A '+'-intersection is interpreted as a degenerate case of two arbitrarily close 'T'-intersections.

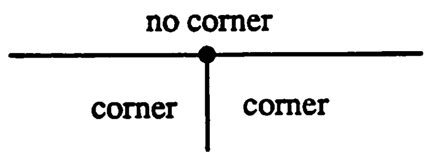


Fig. A.3.1 'T'-intersection at a node of the floorplan graph.

Theorem: Every floorplan graph with v vertices has $e = \frac{3}{2} v - 2$ edges.

Proof: Nodes only exist at the 'T'-intersections in the floorplan graph[2] and at the 4 corners of the floorplan. Therefore 4 nodes are of degree 2, $v - 4$ nodes are of degree 3. Every edge is incident to two nodes. Hence

$$e = ((v - 4) \times 3 + 4 \times 2) \frac{1}{2} = \frac{3}{2} v - 2.$$

All the templates in the library produce rectangle dissections, none of them introduces empty rooms. For that case every edge of the floorplan graph corresponds to a bottleneck in the layout. The number of bottlenecks is therefore

$$(2) \quad e = \frac{3}{2} (2r + 2) - 2 = 3r + 1.$$

(1) and (2) combined result in

$$\frac{3}{2} r^3 + 11 r^2 + \frac{13}{2} r + 1$$

numbers to be stored with each template. In the case of $r = 5$ rooms under the assumption that each number is stored as a 2-byte short integer (e. g. the probabilities are normalized to a number between 0 and 100) that is approximately 1 kByte per template.

Appendix B. Layout Examples

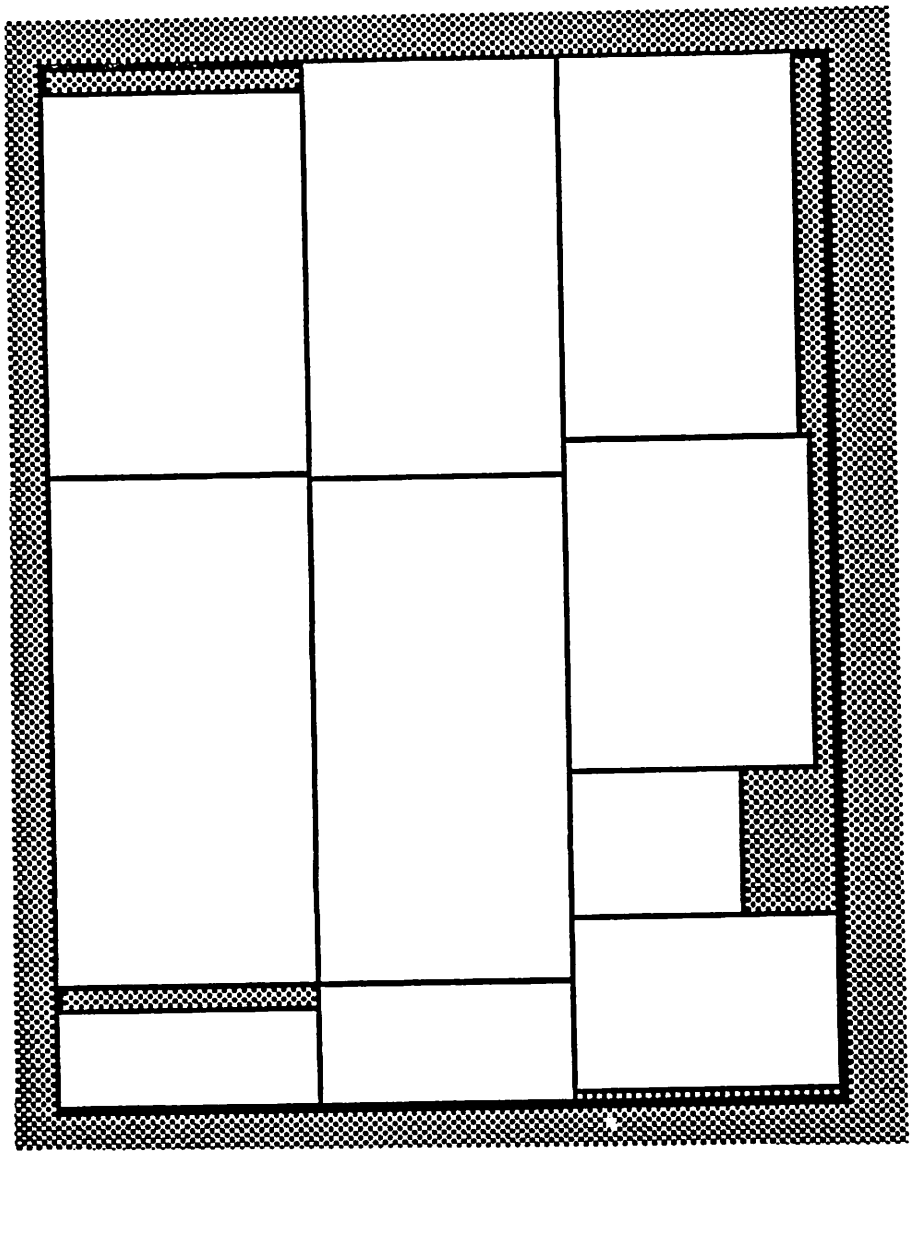


Fig. B.1.1 Block packing of example primBBL1.

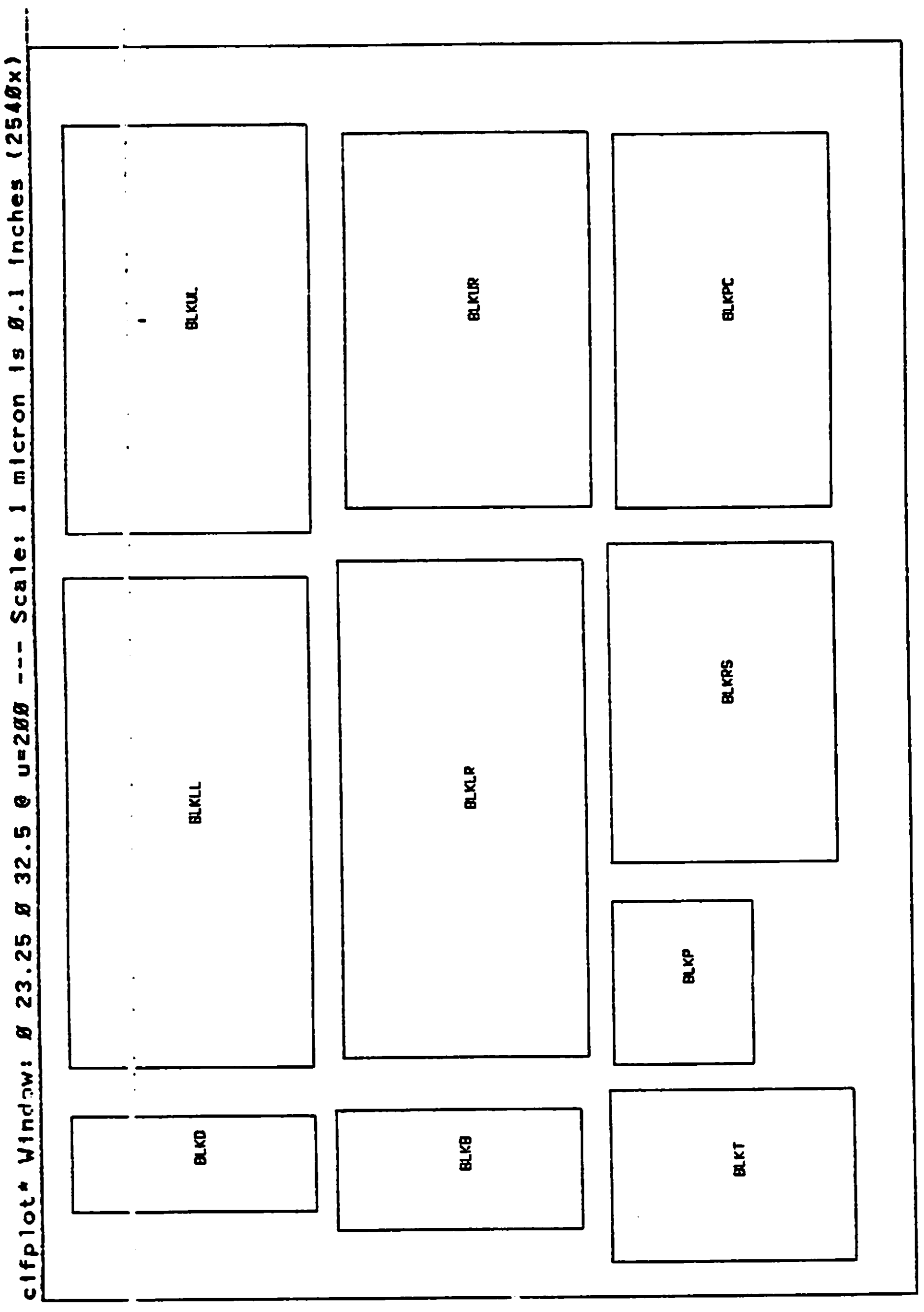

clfplot* Window: Ø 23.25 Ø 32.5 @ u=2ØØ --- Scale: 1 micron is Ø.1 inches (254Øx)
BLKUL
BLKUR
BLKPC
BLKLL
BLKLR
BLKRS
BLKP
BLKD
BLKB
BLKT

Fig. B.1.2 Placement of example primBBL1.

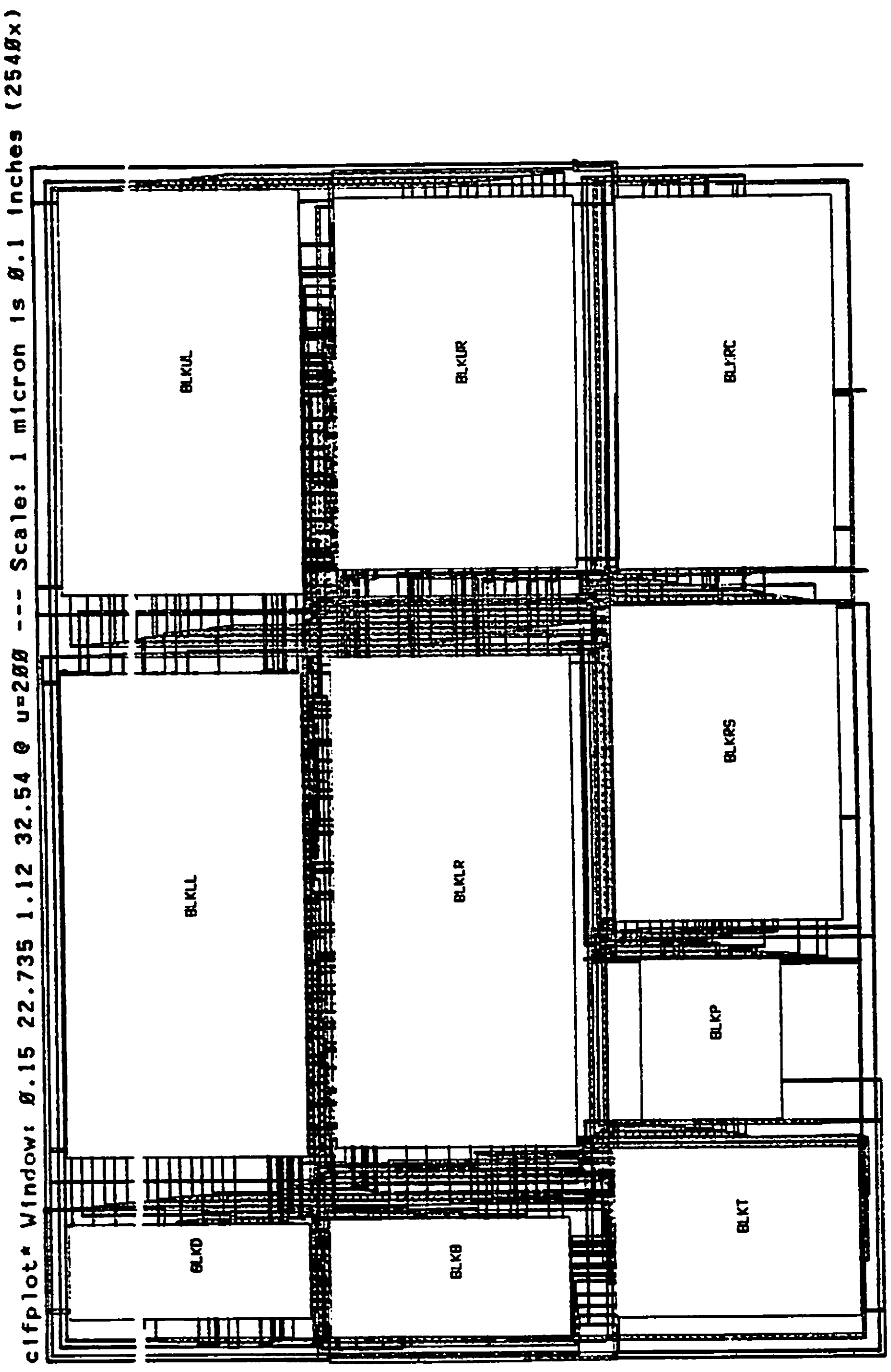

clfplot* Window: Ø.15 22.735 1.12 32.54 @ u=2ØØ --- Scale: 1 micron is Ø.1 inches (254Øx)
BLKUL
BLKUR
BLKRC
BLKLL
BLKLR
BLKRS
BLKP
BLKD
BLKB
BLKT

Fig. B.1.3 Layout of example primBBL1 after routing.

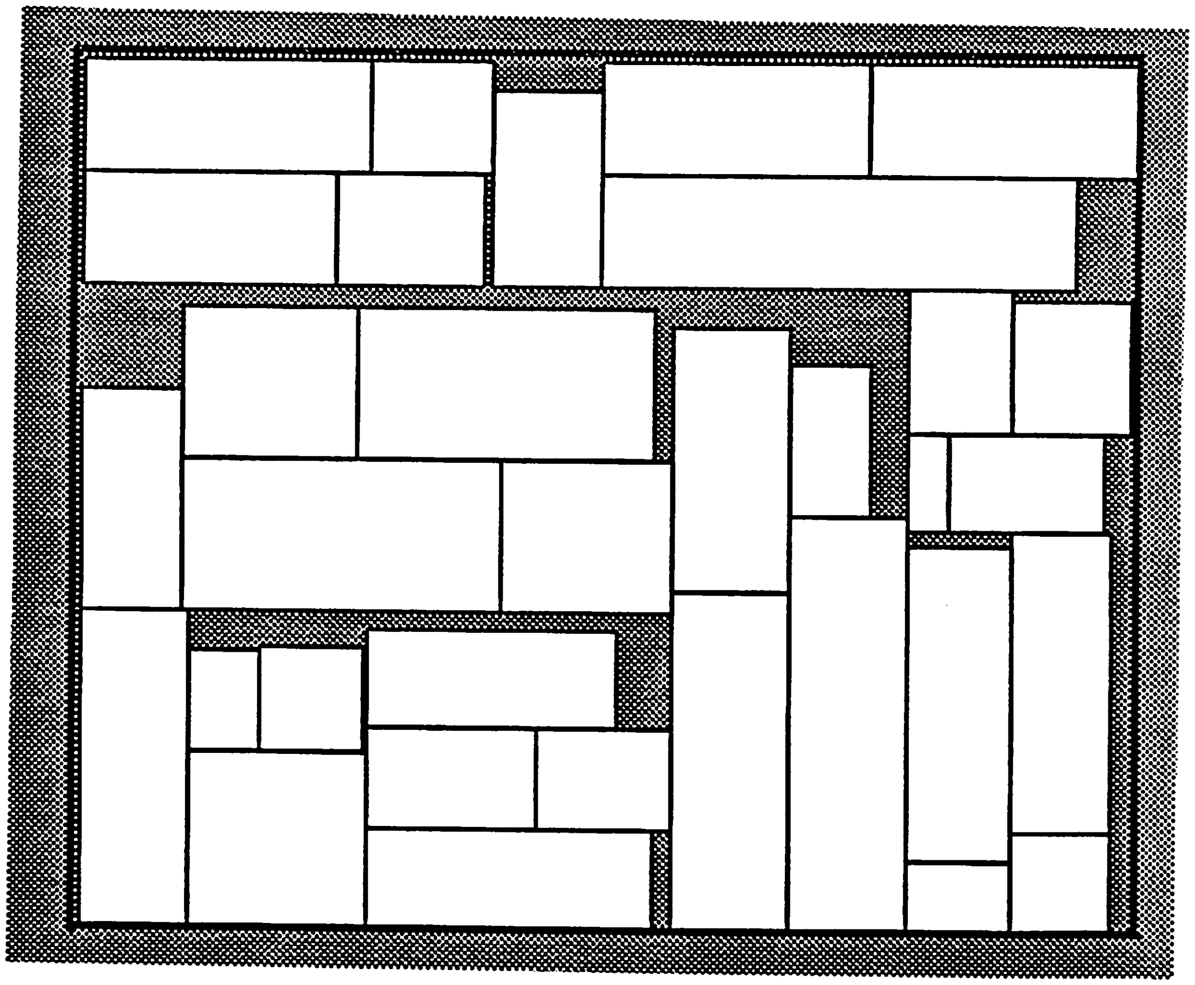


Fig. B.2.1 Block packing of example primBBL2.

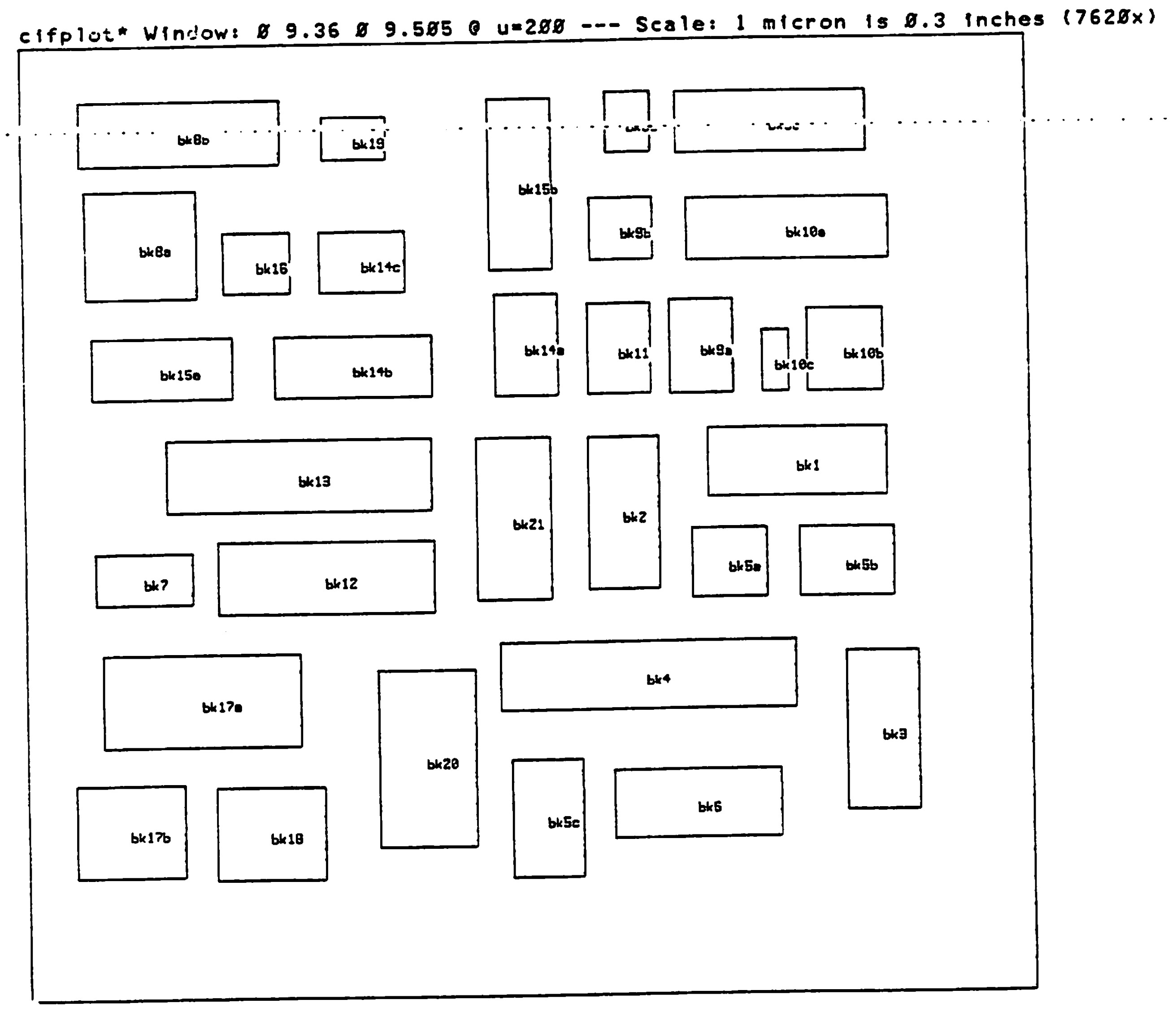


Fig. B.2.2 Placement of example primBBL2.

cifplot* Window: Ø 8.89 -Ø.Ø35 9.115 @ u=2ØØ --- Scale: 1 micron is Ø.3 inches (762Øx)

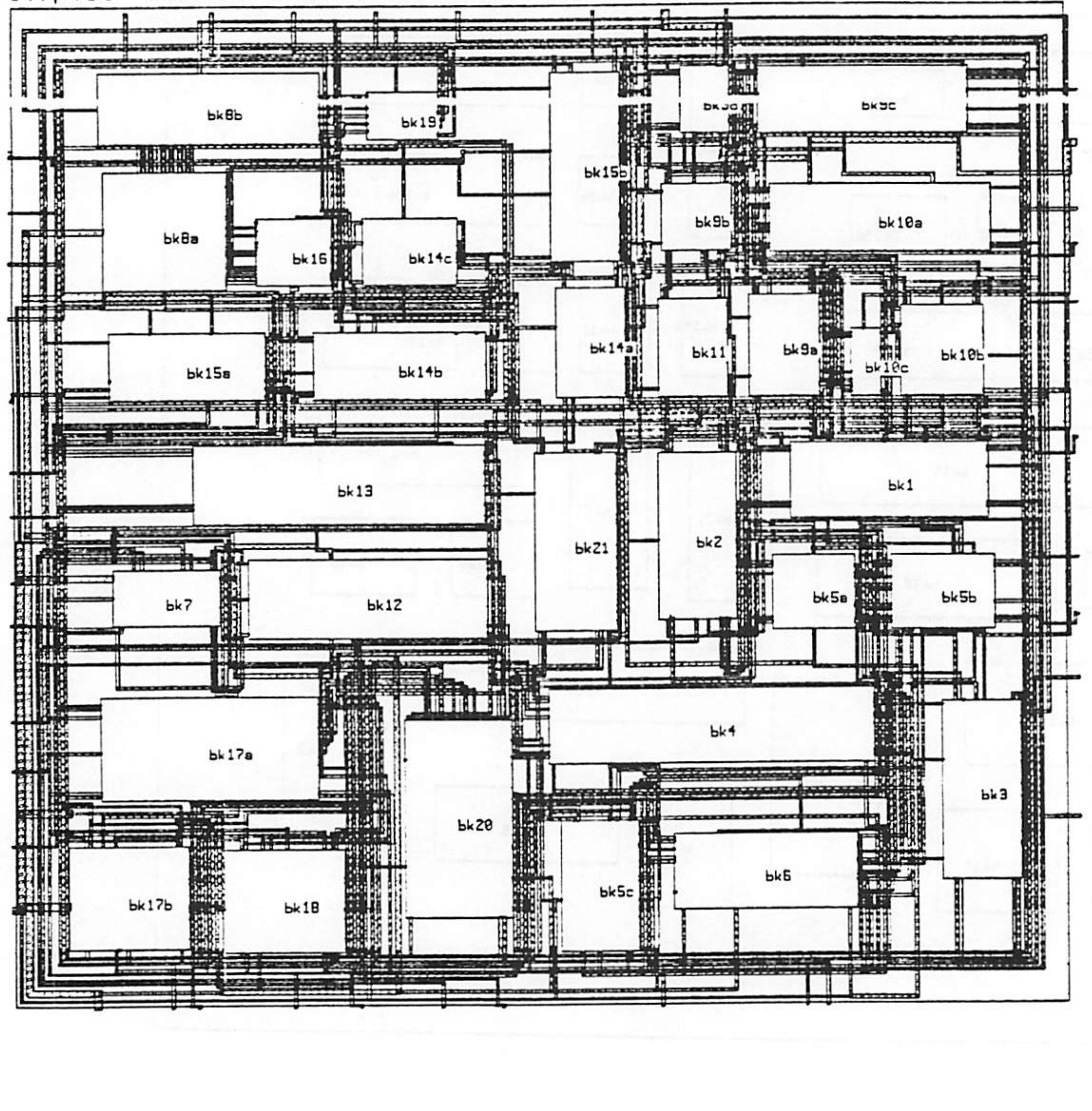


Fig. B.2.3 Layout of example primBBL2 after routing.

Appendix C. User's Manual

C.1. User Interface

The following description of the user interface of the BEAR placement program assumes that a chip window has been opened and the clustering is done. The placer then can be invoked by typing *pl* while the curser is in the chip window. The reader should be familiar with the general approach taken in the placement program before continuing to read here.

To start the placer, 3 types of input parameters are requested from the user. They will be described in the sequence the program prompts for them.

- Determination of the goal shape.

  The desired shape of the final layout can be specified in either of two ways: As goal aspect ratio (ratio of width to height) or as a fixed width or height of one dimension of the layout. The default shape in either case is a square. Although it cannot be guaranteed that the specified goal can be achieved exactly, the results are never far away from the desired value.

  Because the goal shape plays an important role in the computation of the objective function, it is often helpful to play around with this number to get the best result (a change from 1.1 to 1.15 may have a big impact). This problem is alleviated if a 1-level lookahead (see below) with reasonably large search region is specified.

- Trade-off between area and sum of wire lengths.

  On a scale from 0.0 to 1.0 the relative weight of the objective functions for area minimization and for wire length minimization can be influenced. 0.0 means that area minimization is most important, 1.0 emphasizes wire lengths.

  It is very difficult to know exactly what the optimal weighting for minimal wire lengths is, because the dead space that is introduced by moving strongly connected blocks closer to each other may have a detrimental effect. The internal weight factors are adjusted so as to make it probable that a value of 1.0 gives the optimal solution, for some examples however, it might be better to choose 0.8 or even 1.2.

- Lookahead and pruning of the search tree.

  To improve the placement results, the breadth-first traversal of the hierarchy can be complemented by a depth-first lookahead to improve the reliability of the objective function. The user is free to specify different lookahead depths from different levels of the hierarchy and to narrow down the search space more or less drastically.

  Because of the additional computational complexity of the lookahead, most of the time only 0 and 1 will be considered as relevant. If a lookahead (> 0) is selected, the user is prompted for a constant that determines the width of the search. On every level the objective function of that level is computed. Only those possibilities that lie between its minimal value and (1 + pruning constant) x minimal value are further explored.

  For a lookahead of 0 (no lookahead) most time is spent in the last level to determine the orientations of the macrocells. To make the time spent on higher levels comparable to that time, a pruning constant of 0.5 is OK (but that depends very much on the example). A constant of 1.0 most of the time gives near-optimal results; only in few cases can the result be improved above values of 2.0.

After that the program actually begins to run. At the end of each hierarchical level, information concerning the match between bottom-up and top-down routing area estimation is printed out. If it turns out that the routing area allocated does not match with the area needed, the routing area estimate can be increased with the *routing area adjustment factor* (value > 1.0) or decreased (value < 1.0) before a new clustering. If the crude bottom-up estimation is wrong, it can be adjusted with the *bottom-up adjustment factor* in the same way. Although a mismatch does not cause any errors, it may produce strange results (especially if too much area is made available by the bottom-up estimation).

At the end of the program the dimensions of the produced layout are printed out. To transfer the placement from the internal data structure of the placement to the data base (and the chip window) an additional operation is necessary.

C.2. File Interface for Clustering

To make it possible to keep the clustering tree constant after input changes, a clustering tree can be stored to and retrieved from a file. That file also offers the opportunity to edit the clustering tree. An example shows best how the file is organized:

```
9
1 18 23 24
3 4 5 30
6 7 20 27
2 31 32 33
11 12 0 0
8 9 10 0
13 17 28 29
14 15 16 19
21 22 25 26

3
1 9 0 0
2 3 4 0
5 6 7 8

1
1 2 3 0
```

A blank line starts a new clustering level, the lowest level appearing first. The second line gives the number of clusters on that level. Every cluster occupies one line with as many numbers as elements are allowed per cluster. If there are less elements than the maximal cluster size, zeros are used to pad the input lines to the standard length. The numbers on the first clustering level correspond to the sequence in which the blocks are stored in the input file. On the following clustering levels the numbers indicate the position in the preceeding clustering level. For example on the second clustering level the first cluster consists of clusters 1 (blocks 1, 18, 23, 24) and 9 (blocks 21, 22, 25, 26) of the preceeding level.

References

[Bur83] M. Burstein, S. J. Hong, R. Pelavin, "Hierarchical VLSI Layout: Simultaneous Placement and Wiring of Gate Arrays", *Proc. VLSI '83*, pp. 45-60, 1983.

[Cas86] A. Casotto, F. Romeo, A. Sangiovanni-Vincentelli, "A Parallel Simulated Annealing Algorithm for the Placement of Macro-Cells", *Dig. Tech. Papers, IEEE Int. Conf. on CAD*, pp. 30-33, 1986.

[Che83] N. P. Chen, C. P. Hsu, E. S. Kuh, C. C. Chen, M. Takahashi, "BBL: A Building Block Layout System for Custom Chip Design", *Dig. Tech. Papers, IEEE Int. Conf. on CAD*, pp. 40-41, 1983.

[Che87] H. H. Chen, "Routing L-Shaped Channels in Nonslicing-Structure Placement", *Proc. 24th Design Autom. Conf.*, pp. 152-158, 1987.

[Che88] X. Chen, M. L. Bushnell, "A Module Area Estimator for VLSI Layout", *Proc. 25th Design Autom. Conf.*, pp. 54-59, 1988.

[Dai87] W. Dai, E. S. Kuh, "Simultaneous Floor Planning and Global Routing for Hierarchical Building Block Layout", *IEEE Trans. on CAD*, vol. CAD-6, pp. 828-837, 1987.

[Esc87] B. Eschermann, "A Clustering Algorithm for Hierarchical Floorplanning", *Project Report EECS 244, Univ. of California, Berkeley*, Fall 1987.

[ElG81] A. A. ElGamal, Z. A. Syed, "A Stochastic Model for Interconnections in Custom Integrated Circuits", *IEEE Trans. on Circuits and Systems*, vol. CAS-28, pp. 883-893, 1981.

[Fow85] C. Fowler, G. D. Hachtel, L. Roybal, "New Algorithms for Hierarchical Place and Route of Custom VLSI", *Dig. Tech. Papers, Int. Conf. on CAD*, pp. 273-275, 1985.

[Han76] M. Hanan, P. K. Wolff, B. J. Agule, "A Study of Placement Techniques", *Design Automation and Fault-Tolerant Computing*, pp. 28-61, 1976.

[Hel78] W. R. Heller, W. F. Mikhail, W. E. Donath, "Prediction of Wiring Space Requirements for LSI", *Design Automation and Fault-Tolerant Computing*, pp. 117-144, 1978.

[Khe87] M. Khellaf, "On the Partitioning of Graphs and Hypergraphs", *Ph.D. Diss., Dept. IEOR, Univ. of California, Berkeley*, 1987.

[Koz84] T. Kozawa, C. Miura, H. Terai, "Combine and Top Down Block Placement Algorithm for Hierarchical Logic VLSI Layout", *Proc. 21st Design Autom. Conf.*, pp. 667-669, 1984.

[Kur86] F. J. Kurdahi, A. C. Parker, "PLEST: A Program for Area Estimation of VLSI Integrated Circuits", *Proc. 23rd Design Autom. Conf.*, pp. 467-473, 1986.

[LaP85] D. P. LaPotin, S. W. Director, "Mason: A Global Floor-Planning Tool", *Dig. Tech. Papers, Int. Conf. on CAD*, pp. 143-145, 1985.

[Lau79] U. Lauther, "A Min-Cut Placement Algorithm for General Cell Assemblies Based on a Graph Representation", *Proc. 16th Design Autom. Conf.*, pp. 1-10, 1979.

[Lau88] U. Lauther, *private communication*, June 1988.

[Mue87] T. R. Mueller, D. F. Wong, C. L. Liu, "An Enhanced Bottom-up Algorithm for Floorplan Design", *Dig. Tech. Papers, Int. Conf. on CAD*, pp. 524-527, 1987.

[Oht70] T. Ohtsuki, N. Sugiyama, H. Kawanishi, "An Optimization Technique for Integrated Circuit Layout Design", *Proc. ICCST*, pp. 67-68, 1970.

[Ott82] R. H. J. M. Otten, "Layout Structures", *Proc. IEEE Large Scale Systems Symp.*, 1982.

[Ott83] R. H. J. M. Otten, "Efficient Floorplan Optimization", *Proc. Int. Conf. on Computer Design*, pp. 499-502, 1983.

[Pre86] B. T. Preas, P. G. Karger, "Automatic Placement - A Review of Current Techniques", *Proc. 23rd Design Autom. Conf.*, pp. 622-629, 1986.

[Sec85] C. Sechen, A. Sangiovanni-Vincentelli, "The TimberWolf Placement and Routing Package", *IEEE J. Solid-State Circuits*, vol. SC-20, pp. 510-522, 1985.

[Sha85] L. Sha, R. W. Dutton, "An Analytical Algorithm for Placement of Arbitrarily Sized Rectangular Blocks", *Proc. 22nd Design Autom. Conf.*, pp. 602-607, 1985.

[Sto83] L. Stockmeyer, "Optimal Orientation of Cells in Slicing Floorplan Designs", *Information and Control*, vol. 57, pp. 91-101, 1983.

[Sze86] A. A. Szepieniec, "Integrated placement/routing in sliced layouts", *Proc. 23rd Design Autom. Conf.*, pp. 300-307, 1986.

[Ued85] K. Ueda, H. Kitazawa, I. Harada, "CHAMP: Chip Floor Plan for Hierarchical VLSI Layout Design", *IEEE Trans. on CAD*, vol. CAD-4, pp. 12-22, 1985.

[Wip85] G. J. Wipfler, D. A. Mlynski, H. Hillner, "An Automatic Placement Procedure for Integrated Circuits", *Proc. Int. Symp. Circuits and Systems*, pp. 13-16, 1985.

[Zim88] G. Zimmermann, "A New Area and Shape Function Estimation Technique for VLSI-Layout", *Proc. 25th Design Autom. Conf.*, pp. 60-65, 1988.