Chapter 6

Floorplanning and Pin Assignment

After the circuit partitioning phase, the area occupied by each block (sub-circuit) can be estimated, possible shapes of the blocks can be ascertained and the number of terminals (pins) required by each block is known. In addition, the netlist specifying the connections between the blocks is also available. In order to complete the layout, we need to assign a specific shape to a block and arrange the blocks on the layout surface and interconnect their pins according to the netlist. The arrangement of blocks is done in two phases; Floorplanning phase, which consists of planning and sizing of blocks and interconnect and the Placement phase, which assign a specific location to blocks. The interconnection is completed in the routing phase. In the placement phase, blocks are positioned on a layout surface, in a such a fashion that no two blocks are overlapping and enough space is left on the layout surface to complete the interconnections. The blocks are positioned so as to minimize the total area of the layout. In addition, the locations of pins on each block are also determined.

The input to the Floorplanning phase is a set of blocks, the area of each block, possible shapes of each block and the number of terminals for each block and the netlist. If the layout of the circuit within a block has been completed then the dimensions (shape) of the block are also known. The blocks for which the dimensions are known are called *fixed* blocks and the blocks for which dimensions are yet to be determined are called *flexible* blocks. Thus we need to determine an appropriate shape for each block (if shape is not known), location of each block on the layout surface, and determine the locations of pins on the boundary of the blocks. The problem of assigning locations to fixed blocks on a layout surface is called the *Placement* problem. If some or all of the blocks are flexible then the problem is called the *Floorplanning* problem. Hence, the placement problem is a restricted version of the floorplanning problem. If one asks for planning of the interconnect in addition to floorplanning, then it is referred to as the chip planning problem . Thus floorplanning is a restricted version of chip planning problem. The terminology is slightly confusing as

floorplanning problems are placement problems as well but these terminologies have been widely used and accepted. It is desirable that the pin locations are identified at the same time when the block locations are fixed. However, due to the complexity of the placement problem, the problem of identifying the pin locations for the blocks is solved after the locations of all the blocks are known. This process of identifying pin locations is called *pin assignment.*

Chip planning, Floorplanning and Placement phases are very crucial in overall physical design cycle. It is due to the fact, that an ill-floorplanned layout cannot be improved by high quality routing. In other words, the overall quality of the layout, in terms of area and performance is mainly determined in the chip planning, floorplanning and placement phases. In this chapter we will review Floorplanning and pin assignment algorithms. Algorithms for placement will be discussed in the subsequent chapter.

There are several factors that are considered by the chip planning, floorplanning, pin assignment and placement algorithms. These factors are discussed below:

1. **Shape of the blocks:** In order to simplify the problem, the blocks are assumed to be rectangular. The shapes resulting from the floorplanning algorithms are mostly rectangular for the same reason. The floorplanning algorithms use *aspect ratios* for determining the shape of a block. The aspect ratio of a block is the ratio between its height and its width. Usually there is an upper and a lower bound on the aspect ratios, restricting the dimensions that the block can have. More recently, other shapes such as L-shapes have been considered, however dealing with such shapes is computationally intensive.

2. **Routing considerations:** In chip planning, it is required that routing is considered as an integral part of the problem. In placement and floorplanning algorithms it maybe sufficient to estimate the area required for routing. The blocks are placed in a manner such that there is sufficient routing area between the blocks, so that routing algorithms can complete the task of routing of nets between the blocks. If complete routing is not possible, placement phase has to be repeated.

3. **Floorplanning and Placement for high performance circuits: For** high performance circuits the blocks are to be placed such that all critical nets can be routed within their timing budgets. In other words, the length of critical paths must be minimized. The floorplanning(placement) process for high performance circuits is also called as *performance driven* floorplanning(placement).

4. **Packaging considerations:** All of these blocks generate heat when the circuit is operational. The heat dissipated should be uniform over the entire surface of the group of blocks placed by the placement algorithms. Hence, the chip planning, floorplanning and placement algorithms must place the blocks, which generate a large amount of heat, further apart

from each other. This might conflict with the objective for high performance circuits and some trade off has to be made.

5. **Pre-placed blocks:** In some cases, the locations of some of the blocks may be fixed, or a region may be specified for their placement. For example, in high performance chips, the clock buffer may have to be located in the center of the chip. This is done with the intention to reduce the time difference between arrival time of the clock signal at different blocks. In some cases, a designer may specify a region for a block, within which the block must be placed.

In this chapter, we will discuss floorplanning and pin assignment problems in different design styles. Section 6.1 discusses the Floorplanning problem and algorithms for the floorplanning problems. Section 6.2 presents a brief introduction to chip planning, while pin assignment is discussed in Section 6.3. In Section 6.4, we discuss integrated approach to these problems.

6.1 Floorplanning

As stated earlier, Floorplanning is the placement of flexible blocks, that is, blocks with fixed area but unknown dimensions. It is a much more difficult problem as compared to the placement problem (discussed in Chapter 7). In floorplanning, several layout alternatives for each block are considered. Usually, the blocks are assumed to be rectangular and the lengths and widths of these blocks are determined in addition to their locations. The blocks are assigned dimensions by making use of the aspect ratios. The aspect ratio of a block is the ratio of the width of the block to its height. Usually, there is an upper and a lower bound on the aspect ratio a block can have as the blocks cannot take shapes which are too long and very thin. Initial estimate on the set of feasible alternatives for a block can be made by statistical means, i.e., by estimating the expected area requirement of the block. Many techniques of general block placement have been adapted to floorplanning. The only difference between floorplanning and general block placement is the freedom of cells' interface characteristic. Like placement, inaccurate data partly affects floorplanning. In addition to the inaccuracy of the cost function that we optimize, the area requirements for the blocks may be inaccurate.

Floorplanning algorithms are typically used in hierarchical design. This is due to the fact that, although the dimensions of each leaf of the hierarchical tree may be known, the blocks at the node level in the tree are *flexible,* i.e., they can take any dimension. Hence, the floorplanning algorithms are used at each of the nodes in the tree so that the area of the layout is minimum and the position of all the blocks are identified.

6.1.1 Problem Formulation

The input consists of $B_1, B_2, \ldots, B_n$ circuit blocks, with area $a_1, a_2, \ldots, a_n$ respectively. Associated with each block are two aspect ratios A_i^l and A_i^h, which

give the lower and the upper bound on the aspect ratio for that block. The floorplanning algorithm has to determine the width w_i and height, h_i of each block B_i such that $A_i^l \leq \frac{h_i}{w_i} \leq A_i^h$. In addition to finding the shapes of the blocks, the floorplanning algorithm has to generate a valid placement such that the area of the layout is minimized.

A *slicing floorplan* is a floorplan which can be obtained by recursively partitioning a rectangle into two parts either by a vertical line or a horizontal line. The cut tree obtained by min-cut algorithm is known as *slicing tree.* A slicing tree is a binary tree in which each leaf represents a partition and each internal node represents a cut. Consider the floorplan as shown in Figure 6.1. Partitions are labeled with letters and cutlines are labeled with numbers. Figure 6.1(b) shows the slicing tree for the floorplan in Figure 6.1 (a). Figure 6.1(c) is the slicing tree indicating the cut direction. Figure 6.1(d) shows a floorplan for which there is no valid slicing tree.

A floorplan is said to be *hierarchical* of *order* k, if it can be obtained by recursively partitioning a rectangle into at most k parts. The hierarchy of a hierarchical floorplan can be represented by a *floorplan tree.* Figure 6.2 shows a hierarchical floorplan of order 5 and its floorplan tree. Each leaf in the tree corresponds to a basic rectangle and each internal node corresponds to a composite rectangle in the floorplan. An important class of hierarchical floorplans is the set of all slicing floorplans.

6.1.1.1 Design Style Specific Floorplanning Problems

Floorplanning is not carried out for some design styles. This is due to the fixed dimensions of blocks in some design styles.

1. **Full custom design style:** Floorplanning for general cells is the same as discussed above.
2. **Standard cell design style:** In standard cell design style, the dimensions of cells are fixed, and floorplanning problem is simply the placement problem. For large standard cell design, circuit is partitioned into several regions, which are floorplanned, before cells are placed in regions.
3. **Gate array design style:** Like standard cells, the floorplanning problem is same as placement problem.

6.1.2 Classification of Floorplanning Algorithms

Floorplanning methods can be classified as follows:

1. Constraint based methods.
2. Integer programming based methods.
3. Rectangular dualization based methods.
4. Hierarchical tree based methods.

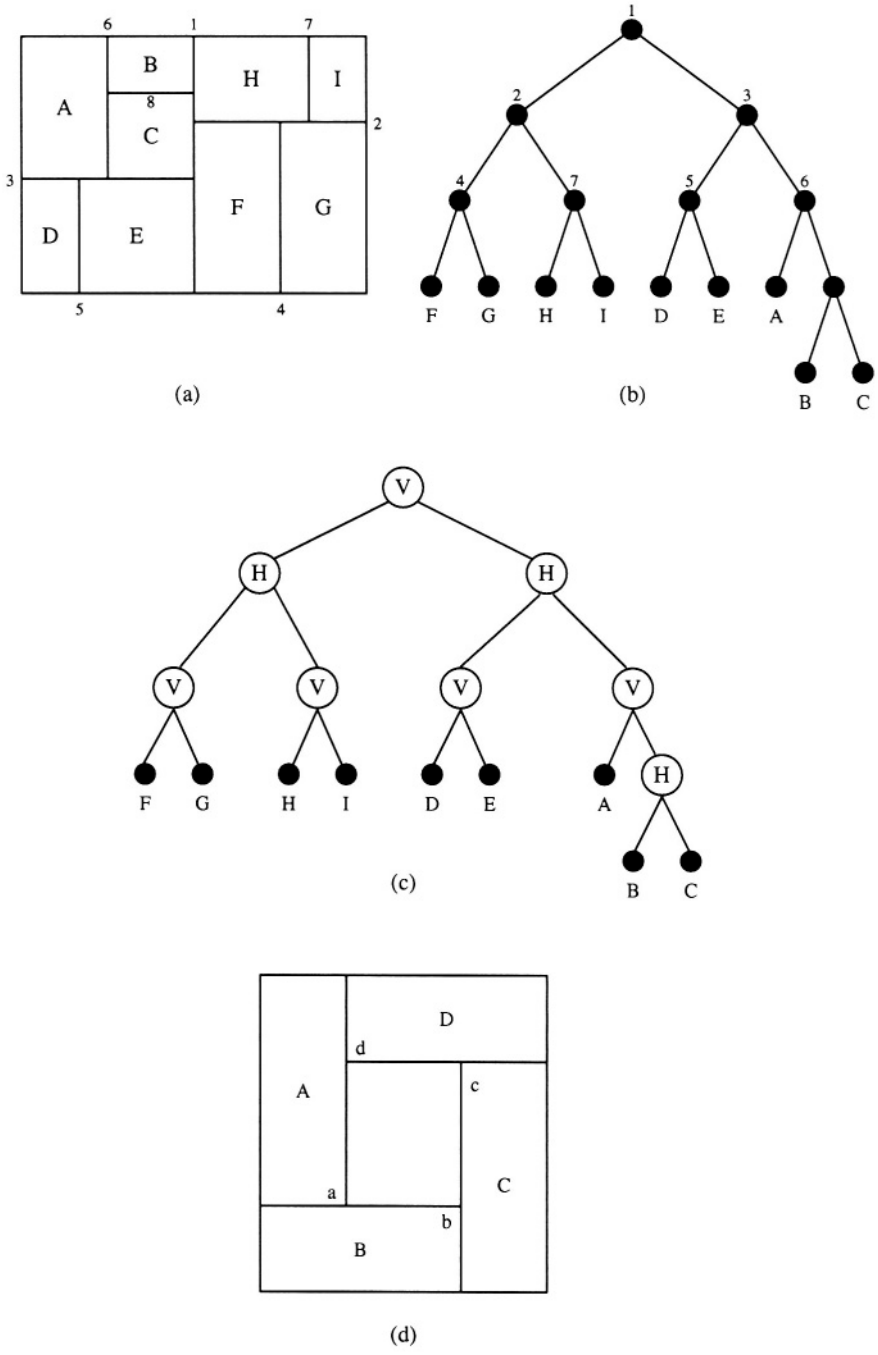

Figure 6.1: A floorplan with slicing tree and a non-slicing floorplan.

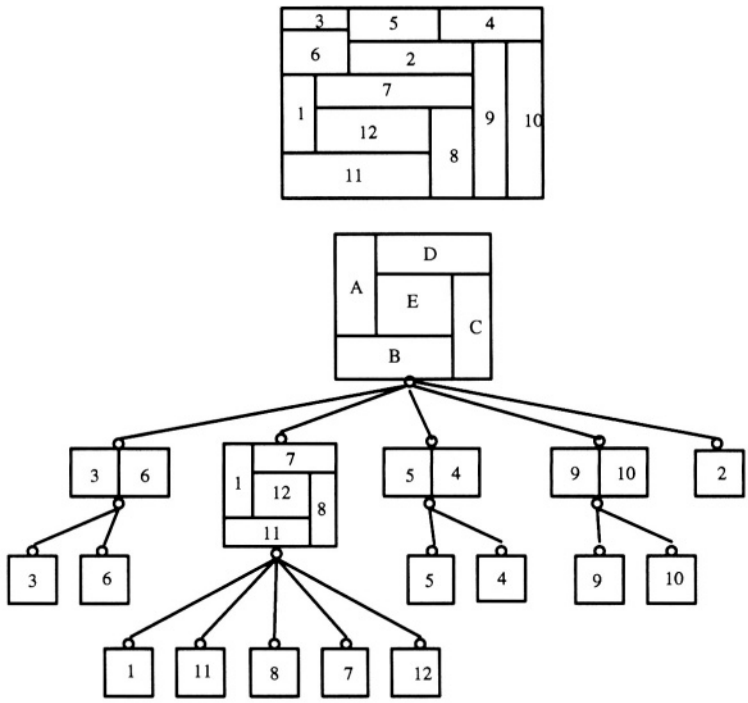

Figure 6.2: Hierarchical Floorplan.

5. Simulated Evolution algorithms

6. Timing Driven Floorplanning Algorithms

These methods will be discussed in the following subsections.
Beside the methods stated above, several simple methods may be used, such as min-cut method. The process of min-cut can be used to construct a sized floorplan. The first phase of min-cut method i.e., bipartition of a weighted graph, helps in constructing the floorplan. The weight of the vertex roughly estimates the area taken up by the block. This weight may represent the area of the corresponding ceil in general-cell placement. The initial sized floorplan represents an empty base rectangle whose area is the total of all weights of the vertices of the weighted graph and each node in the tree represents a rectangular room in the layout area. All the floorplans that can be generated with min-cut bipartitioning are slicing floorplans.

6.1.3 Constraint Based Floorplanning

This method, proposed by Vijayan and Tsay [VT91], constructs a floorplan of optimal area that satisfies (respects) a given set of constraints. A set of

horizontal and vertical topological (i.e., ordering) constraints is derived from the relative placement of blocks. Given a constraint set, it is usually the case that there is no reason to satisfy all the constraints in the set. This is especially true when a majority of the blocks have flexible shapes. A floorplan is said to respect a constraint, if for each pair of blocks, the floorplan satisfies at least one constraint (horizontal or vertical). A constraint set is said to be *overconstmined* if it has many redundant constraints. It is desirable to derive a complete constraint set from the input relative placement and then to remove those redundant constraints that result in reduction of floorplan area.

A topological constraint set of a set of blocks is given by two directed acyclic graphs (G_H, G_V): G_H is the horizontal constraint graph and G_V is the vertical constraint graph. In order to reduce the floorplan area, the heuristic iteratively removes a redundant constraint from the critical path of either G_H or G_V and also iteratively reshapes the blocks on the critical paths of the two graphs. Critical path is the longest path in G_H or G_V. The *input* to the algorithm is a constraint set (G_H, G_V) of the set of blocks. To minimize the floorplan area, repeat steps 1 and 2 until there is no improvement in the floorplan area.

1. **Step 1:** Repeat the following steps until no strongly redundant edges on P_H or P_V exist. G_H and G_V are topologically sorted and swept. Either P_H or P_V, whichever is more critical is selected, where P_H and P_V are the critical paths of G_H and G_V respectively. The strongly redundant edge on the selected critical path is eliminated.

2. **Step 2:** The current shapes of the blocks are stored and a path, either P_V or P_H is selected depending on which of the two is more critical. All the flexible blocks on the selected path are reshaped. G_H and G_V are scanned again to construct the new floorplan. If the newly generated floorplan is better than the previous one the stored block shapes are updated. All the steps described above are repeated, a specified number of times.

Each pass of the algorithm constitutes one execution of two steps. Constraint reduction takes place in step 1 and step 2 does the reshaping of the blocks. If the chip dimensions are fixed, the passes are repeated until the target dimensions are reached. Otherwise the passes can be repeated until there is improvement in the floorplan area. Typically three or four passes are required.

The purpose of removing a redundant edge on the critical path is to break the path into two smaller paths. A good choice for such a redundant edge is the one which is nearest to the center point of the path. The above heuristic removes only one redundant constraint from a critical path at each iteration, and thus seeks to minimize the number of constraints removed. An edge can be checked for strong redundancy in constant time if we maintain the adjacency matrix of G_H and G_V. It takes $O(n^2)$ time to set up adjacency matrices. A topological sort of a directed acyclic graph with n nodes and m edges takes $O(m + n)$ time. The number of topological sorts executed depends on the number of redundant edges removed, the user-specified value for the number of reshaping iterations, and the number of passes.

6.1.4 Integer Programming Based Floorplanning

In this section an integer programming formulation for generating the floorplan developed by Sutanthavibul, Shragowitz and Rosen [SSR91] is presented. The floorplanning problem is modeled as a set of linear equations using 0/1 integer variables. Two types of constraints are considered: the overlap constraints and the routability constraints. The overlap constraints prevent any two blocks from overlapping whereas the routability constraints estimate the routing area required between the blocks. For the critical nets, net lengths are specified which should not be exceeded. The length of the net depends on the timing budget of that net. The critical net constraints ensure that the length of the critical nets does not exceed this specified value. We now describe how the constraints can be developed.

1. **Block overlap constraints for fixed blocks:** Given two fixed (rigid) blocks, B_{ri} and B_{rj} which should not overlap, we have four possible ways to position the two blocks so as to avoid overlap. Let $\{x_i, y_i, w_i, h_i\}$ and $\{x_j, y_j, w_j, h_j\}$ be the 4-tuples associated with blocks B_{ri} and B_{rj} respectively, where (x_i, y_i) gives the location of the block, w_i is the width of the block and h_i is the height of the block. The block B_{rj} can be positioned to the right, left, above or below block B_{ri}. These conditions transformed into equations given below:

$$\begin{array}{lll} x_i + w_i \leq x_j & (B_{rj} \text{ is to the right of } B_{ri}), & \text{or} \\ x_i - w_j \geq x_j & (B_{rj} \text{ is to the left of } B_{ri}), & \text{or} \\ y_i + h_i \leq y_j & (B_{rj} \text{ is to the above of } B_{ri}), & \text{or} \\ y_i - h_j \geq y_j & (B_{rj} \text{ is to the below of } B_{ri}) & \end{array} \quad (1)$$

To satisfy one of these equations, two 0-1 integer variables x_{ij} and y_{ij} are used for each pair of blocks. Two bounding functions W and H are defined such that, $|x_i - x_j| \leq W$ and $|y_i - y_j| \leq H$. W can be equal to $W_{\max}$ which is the maximal allowed width of the chip or $W = \sum_{i=1}^{p+q+r} w_i$. Similarly, $H = H_{\max}$, the maximal allowed height of the chip or $H = \sum_{i=1}^{p+q+r} h_i$. Equation set (1) can be rewritten with the introduction of the integer variables to generate the 'or' condition as,

$$\begin{aligned} x_i + w_i &\leq x_j + W(x_{ij} + y_{ij}) \\ x_i - w_j &\geq x_j + W(1 - x_{ij} + y_{ij}) \\ y_i + h_i &\leq y_j + W(1 + x_{ij} - y_{ij}) \\ y_i - h_j &\geq y_j + W(2 - x_{ij} - y_{ij}) \end{aligned} \quad (2)$$

As the integer variables x_{ij} and y_{ij} can take either 0 or 1 values, only one of the above equations in (2) will be active and other equations will be true depending on the value of x_{ij} and y_{ij}. For example, when $x_{ij} = y_{ij} = 1$, the first equation in (2) becomes active and all other equations are true.

$$\begin{aligned} & x_i \geq 0, \qquad y_i \geq 0 \\ & x_i + w_i \leq W, \\ & y^* \geq y_i + h_i \end{aligned} \quad (3)$$

where y^* is the height to be minimized. To allow rotation of the blocks so as to optimize the solution, another integer variable z_i is used for each block. z_i is 0 when the block is in its initial orientation and 1 when the block is rotated by 90°. The constraints for the fixed blocks can be rewritten as:

$$\begin{aligned} &x_i + z_i h_i + (1 - z_i) w_i \leq x_j + M(x_{ij} + y_{ij}) \\ &x_i - z_i h_j - (1 - z_j) w_j \geq x_j + M(1 - x_{ij} + y_{ij}) \\ &y_i + z_i w_i + (1 - z_i) h_i \leq y_j + M(1 + x_{ij} - y_{ij}) \\ &y_i - z_j w_j - (1 - z_j) h_j \geq y_j + M(2 - x_{ij} - y_{ij}) \end{aligned} \tag{4}$$

where, $M = \max(W, H)$. Constraints (3) are rewritten as:

$$\begin{aligned} &x_i \geq 0, \quad y_i \geq 0, \\ &x_i + (1 - z_i) w_i + z_i h_i \leq W, \\ &y^* \geq y_i + (1 - z_i) w_i + z_i h_i \end{aligned} \tag{5}$$

where y^* is the height to be minimized. The floorplanning problem, for fixed blocks without taking into consideration either routing areas or critical nets can be solved by finding the minimum y^* subject to constraints (4) and (5).

2. **Block overlap constraints for flexible blocks:** So far we discussed about fixed blocks. We can now see how constraints for flexible blocks can be developed. The flexible blocks can take rectangular shapes within a limited aspect ratio range i.e. its width and height can be varied keeping the area fixed. The non-linear area relation is linearized about the point of maximum allowable width by applying the first two members of the Taylor series giving,

$$h_i = h_{i0} + \Delta w_i \lambda_i$$

where,

$$\begin{aligned} h_{i0} &= \frac{A_i}{w_{imax}}, \\ \lambda_i &= \frac{A_i}{w_{imax}^2}, \\ \Delta w_i &= w_{imax} - w_i \end{aligned}$$

where Δw_i is a continuous variable for block B_{fi}. The overlap constraints for a flexible block B_{fi} and a fixed block B_{rj} can be written as:

$$\begin{aligned} &x_i + w_{imax} - \Delta w_i \leq x_j, && (B_{fj} \text{ is to the right of } B_{ri}), && \text{or} \\ &y_i + h_{i0} + \Delta w_i \lambda_i \leq y_j, && (B_{fj} \text{ is above } B_{ri}), && \text{or} \\ &x_i - w_j \geq x_j, && (B_{fj} \text{ is to the left of } B_{ri}), && \text{or} \\ &y_i - h_j \geq y_j, && (B_{fj} \text{ is below } B_{ri}) \end{aligned} \tag{6}$$

Using two integer variables x_{ij} and y_{ij} per block pair as was done for fixed blocks, the 'or' condition between the equations can be satisfied.

The same set of equations can be extended to get overlap constraints between two flexible blocks. Using the same technique, the interconnection length constraints and routing area constraints can be developed. This set of equations are the input to any standard linear programming software package such as LINDO. The locations of the blocks and their dimensions are variables, the values of which are calculated by the software depending on the constraints and the objective function.

In [CF98], authors present a new convex programming formulation of the area minimization with a lesser numbers of variables and constraints than previous papers.

6.1.5 Rectangular Dualization

The partitioning process generates a group of subcircuits and their interconnections. This output from a partitioning algorithm can be represented as a graph $G = (V, E)$ where the vertices of the graph correspond to the subcircuits and the edges represent the interconnections between the subcircuit. The floorplan can be obtained by converting this graph into its *rectangular dual* and this approach to floorplanning is called *rectangular dualization.* A rectangular dual of graph $G = (V, E)$ consists of non-overlapping rectangles which satisfy the following properties:

1. Each vertex $v_i \in V$ corresponds to a distinct rectangle R_i, $1 \leq i \leq |V|$.
2. For every edge $(v_i, v_j) \in E$, the corresponding rectangles R_i and R_j are adjacent in the rectangular dual.

When this method is directly applied to the graph generated by partitioning, it may not be possible to satisfy the second property for generating the rectangular dual.

The problem of finding a suitable rectangular dual is a hard problem. In addition, there are many graphs which do not have rectangular duals. A further complication arises due to areas and aspect ratios of the blocks. In rectangular dualization, areas and aspect ratios are ignored to simplify the problem. As a result, the output cannot be directly used for floorplanning. Kozminski and Kinnen [KK84] have presented an algorithm for finding a rectangular dual of a planar triangulated graph. Usually, the graph is processed only if a rectangular dual for the graph exists. Bhasker and Sahni [BS86] have extended the approach in [KK84] to present a linear time algorithm for finding a rectangular dual of a planar triangulated graph.

A planar triangular graph (PTG) G is a connected planar graph that satisfies the following properties:

1. every face (except the exterior) is a triangle.
2. all the internal vertices have a degree ≥ 4.
3. all cycles that are not faces have length ≥ 4.

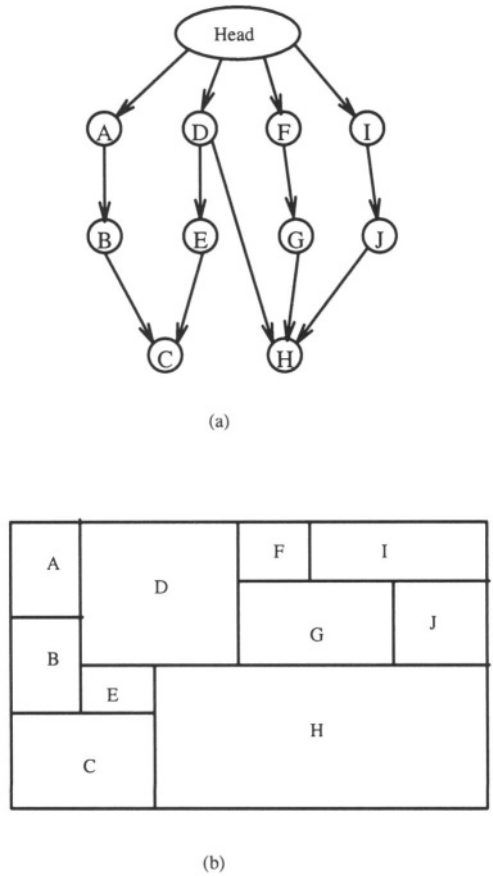

Figure 6.3: Conversion of planar digraph to a floorplan.

Given a PTG, a planar digraph is constructed which is a directed graph. Once a planar digraph is constructed, it can be converted into a floorplan as shown in Figure 6.3. Lokanathan and Kinnen [LK89] presented a procedure for floorplanning that minimizes routing parasitics using rectangular dualization. The use of rectangular dualization maximizes adjacency of blocks that are heavily connected or connected by critical nets.

6.1.6 Hierarchical Tree Based Methods

Hierarchical tree based methods represent a floorplan as a tree. Each leaf in the tree corresponds to a block and each internal node corresponds to a composite block in the floorplan. A floorplan is said to be *hierarchical* of *order* k, if it can be obtained by recursively partitioning a rectangle into at most k parts. Physical hierarchy can be generated in two ways: top-down partitioning or

bottom-up clustering. Partitioning assumes that the relative areas (or number of nodes) Within partitions, at a given level of hierarchy, may be fixed during a top down construction of a decomposition tree (or partitioning tree). There is no justification, except convenience, for this assumption. The optimal choice of relative areas varies from problem instance to problem instance, but there is no way to determine a desirable ratio, in top-down construction. Placements performed by min-cut method, a popular partitioning algorithm, often creates lot of vacant space or white space. Clustering on the other hand is a bottom-up algorithm for constructing a decomposition tree (or cluster tree).

In [DEKP89] a hierarchical floorplanner for arbitrary size rectangular blocks using the clustering approach has been proposed. At each level of the hierarchy, highly connected blocks (or clusters of blocks) are grouped together into larger clusters. At each level, the number of blocks is limited to five so that simple pattern enumeration and exhaustive search algorithms can be used later. Blocks (or block clusters) which are connected by edges of greater than average edge weight are grouped into a single cluster, if the resulting cluster has less than five members.

After forming the hierarchical clustering tree, a floorplanner and a global router together perform a top-down traversal of the hierarchy. Given an overall aspect ratio goal and I/O pin goal, at each level of the hierarchy, the floorplanner searches a simple library of floorplan templates and considers all possible room assignments which meet the combined goals of aspect ratios and I/O pins. At each level, the global routing problem is formulated as a series of minimum steiner tree problems in partial 3-trees. The global routing solution at the current level is used as the I/O pin goal for the floorplan evaluation, and as base for the global routing refinement at the next level. This floorplanning and global routing create constraints on the aspect ratio of the rooms, and gives assignments of I/O pins on the walls of the rooms, which are recursively transmitted downward as sub-goals to the floorplanner and global router. While evaluating the cost of a given floorplan template and room assignment, both chip area and net path length are considered. When undesirable block shapes and pin positions are detected, alternate floorplan templates and room assignments are tried by backtracking and using automatic module generators. This algorithm performs better than other well-known deterministic algorithms and generates solutions comparable to random-based algorithms.

Ting-Chi Wang and D. F. Wong [WW90] have presented an optimal algorithm for a special class of floorplans called *hierarchical floorplans of order 5.* Two types of blocks have been considered; L-shaped and rectangular. The algorithm takes a set of implementations for each block as input and identifies the best implementation for each block so that the resulting floorplan has minimum area.

6.1.7 Floorplanning Algorithms for Mixed Block and Cell Designs

All the algorithms discussed in the previous section can be used for floorplanning of Mixed Block and Cell (MBC) designs. These designs can be viewed as a set of blocks in a sea of cells. This is a popular ASIC layout design style. However, these algorithms were not implemented as a part of any tool which can generate floorplans for MBC designs. In this section, we describe some of the algorithms that were developed as a part of a tool specifically designed for floorplanning of MBC designs.

In [AK90], a heuristic algorithm has been developed for MBC designs. The algorithm employs a combined floorplanning, partitioning and global routing strategy. The main focus of the algorithm is in reducing the white space costs and the wiring cost. In [Sec88], the simulated annealing approach is used to solve the floorplanning problem for MBC designs. In [UKH85], "CHAMP", a floorplanning tool for MBC designs using the hierarchical approach has been presented. In [USS90, PSS+88, cL93], the floorplanning problem for MBC designs has been considered. All existing floorplanning algorithms, except [cL93], for the MBC designs restrict the block shapes to rectangular in order to simplify the problem at hand. Even in [cL93], only the pre-designed block shapes are considered to be rectilinear and the shapes generated for soft modules are always rectangular with varying aspect ratios. None of the existing floorplanning algorithms for MBC designs take advantage of the flexibility of the standard cell regions.

6.1.8 Simulated Evolution Algorithms

[RR96] describes a Simulated Evolution (Genetic) Algorithm for the Floorplan Area Optimization problem. The algorithm is based on suitable techniques for solution encoding and evaluation function definition, effective crossover and mutation operators, and heuristic operators which further improve the method's effectiveness. An adaptive approach automatically provides the optimal values for the activation probabilities of the operators. Experimental results show that the proposed method is competitive with the most effective ones as far as the CPU time requirements and the result accuracy is considered, but it also presents some advantages. It requires a limited amount of memory, it is not sensible to special structures which are critical for other methods, and has a complexity which grows linearly with the number of implementations. Finally, it is demonstrated that the method is able to handle floorplans much larger (in terms of number of basic rectangles) than any benchmark previously considered in the literature.

In [FSZ+97] a Multi-Selection-Multi-Evolution (MSME) scheme for parallelizing a genetic algorithm for floorplan optimization is presented and its implementation with MPI and its experimental results are discussed. The experimental results on a 16 node IBM SP2 scalable parallel computer have shown that the scheme is effective in improving performance of floorplanning over that

of a sequential implementation. The parallel version could obtain better results with more than 90parallel program could reduce both chip area and maximum path delay by more than 8also speed up the evolution process so that there could be higher probability of obtaining a better solution within a given time interval.

The genetic algorithm (GA) paradigm is a search procedure for combinatorial optimization problems. Unlike most of other optimization techniques, GA searches the solution space using a population of solutions. Although GA has an excellent global search ability, it is not effective for searching the solution space locally due to crossover-based search, and the diversity of the population sometimes decreases rapidly. In order to overcome these drawbacks, the paper [TKH96] proposes a new algorithm called immunity based GA (IGA) combining features of the immune system (IS) with GA. The proposed method is expected to have local search ability and prevent premature convergence. IGA is applied to the floorplan design problem of VLSI layout. Experimental results show that IGA performs better than GA.

6.1.9 Timing Driven Floorplanning

With increasing chip complexities and the requirement to reduce design time, early analysis is becoming increasingly important in the design of performance critical CMOS chips. As clock rates increase rapidly, interconnect delay consumes an appreciable portion of the chip cycle time, and the floorplan of the chip significantly affects its performance.

[SYTB95] presents a timing-influenced floorplanner for general cell IC design. The floorplanner works in two phases. In the first phase the modules are restricted to be rigid and the floorplan to be slicing. The second phase of floorplanner allows modification to the aspect ratios of individual modules to further reduce the area of the overall bounding box. The first phase is implemented using genetic algorithm while in the second phase, a constraint graph based approach is adopted.

In [YSAF95] a timing driven floorplanning program for general cell layouts is presented. The approach used combines quality of force directed approach with that of constraint graph approach. A floorplan solution is produced in two steps. First a timing and connectivity driven topological arrangement is obtained using a force directed approach. In the second step, the topological arrangement is transformed into a legal floorplan. The objective of the second step is to minimize the overall area of the floorplan. The floorplanner is validated with circuits of sizes varying from 7 to 125 blocks.

[NLGV95] describes a system for early floorplan analysis of large designs. The floorplanner is designed to be used in the early stages of system design, to optimize performance, area and wireability targets before detailed implementation decisions are made. Unlike most floorplanners which optimize timing by considering only a subset of paths this floorplanner performs static timing analysis during the floorplan optimization process, instead of working on a subset of the paths. The floorplanner incorporates various interactive and automatic

floorplanning capabilities.

6.1.10 Theoretical advancements in Floorplanning

In [PL95] P. Pan and C. L. Liu propose two area minimization methods for general floorplans with respect to the floorplan sizing problem.

The traditional algorithm for area minimization of slicing floorplans due to Stockmeyer has time and space complexity $O(n^2)$ in the worst case. For more than a decade, it has been considered the best possible. [Shi] presents a new algorithm of worst-case time and space complexity $O(n \log n)$, where n is the total number of realizations for the basic blocks, regardless whether the slicing is balanced or not. It has also been shown that $\theta(n \log n)$ is the lower bound on the time complexity of any area minimization algorithm. Therefore, the new algorithm not only finds the optimal realization, but also has the optimal running time.

In [PSL96], the complexity of the area minimization problem for hierarchical floorplans has been shown to be NP-complete (even for balanced hierarchical floorplans). A new algorithm has been presented for determining the nonredundant realizations of a wheel. The algorithm has time cost $O(k^2 \log k)$ and space cost $O(k^2)$ if each block in a wheel has at most k realizations. Based on the new algorithm for wheel, the authors have designed a new pseudo-polynomial area minimization algorithm for hierarchical floorplans of order-5. The time and space costs of the algorithm are $O((nM)^2 \log(nM))$ and $O(n^2 M)$, respectively, where n is the number of basic blocks and M is an upper-bound on the dimensions of the realizations of the basic blocks. The area minimization algorithm was implemented. Experimental results show that it is very fast.

In [CT], the authors have found an $\Omega(k^2)$ lower bound for area optimization of spiral floorplans. Let F be a spiral floorplan where each of its five basic rectangles has k implementations. It has been shown that there can be as many as $\Omega(k^2)$ useful implementations generated for F, in the worst case. This implies that the previously known $O(k^{\frac{2}{\log k}})$-time algorithm is almost optimal.

In [PL], the authors have presented two area minimization methods for general floorplans, which can be viewed as generalizations of the classical algorithm for slicing floorplans of Otten (1982) and Stockmeyer (1983) in the sense that they reduce naturally to their algorithm for slicing floorplans. Compared with the branch-and-bound algorithm of Wimer et al (1989), which does not have a nontrivial performance bound, these methods are provably better than an exhaustive method examined for many other examples.

[HL97] presents a formal algebraic specification (in SETS notation) that is appropriate for VLSI physical design layout and capable of representing both the floorplan topology and the modules' dimensions. The specification proposed allows a concise and rigorous representation of arbitrarily complex composite floorplans. This algebraic description unifies-under a rotation-invariant single-expression formalism-slicing and non-slicing generalized wheels floorplans. As needed by specific floorplan algorithms, it supports either a topology-dimensionless description or the introduction of module dimensions. Finally, it

allows an eightfold reduction-over previous representations-of the total number of floorplan solutions considered in floorplanning problem algorithms.

In [KD97] a new method of non-slicing floorplanning is proposed, which is based on the new representation for non-slicing floorplans, called bounded slicing grid (BSG) structure. The authors have developed a new greedy algorithm based on the BSG structure, running in linear time, to select the alternative shape for each soft block so as to minimize the overall area for general floorplan, including non-slicing structures. A new stochastic optimization method, named genetic simulated annealing (GSA) for general floorplanning is proposed. Based on BSG structure, SA-based local search and GA-based global crossover is extended to L-shaped, T-shaped blocks and high density packing of rectilinear blocks is obtained.

In [DSKB95], it is shown that for any rectangularly dualizable graph, a feasible topology can be obtained by using only either straight or Z-cutlines recursively within a bounding rectangle. Given an adjacency graph, a potential topology, which may be nonslicible and is likely to yield an optimally sized floorplan, is produced first in a top-dozen fashion using heuristic search in AND-OR graphs. The advantage of this technique is four-fold: (i) accelerates top-down search phase, (ii) generates a floorplan with minimal number of nonslice cores, (iii) ensures safe routing order without addition of pseudo-modules, and (iv) solves the bottom-up algorithm efficiently for optimal sizing of general floorplans in the second phase.

[TY95] addresses the problem of minimizing wiring space in an existing slicing floorplan. Wiring space is measured in terms of net density, and the existing floorplan is adjusted only by interchanging sibling rectangles and by mirroring circuit modules. An exact branch and bound algorithm and a heuristic are given for this problem. Experiments show that both algorithms are effective in reducing wiring space in routed layouts.

6.1.11 Recent Trends

Several new trends are emerging in floorplanning, we discuss a few of them. Interactive floorplanning can improve productivity, improve performance and reduce die size. In [EK96a] an interactive floorplanner based on the genetic algorithm is presented. Layout area, aspect ratio, routing congestion and maximum path delay are optimized simultaneously. The design requirements are refined interactively as knowledge of the obtainable cost tradeoffs is gained and a set of feasible solutions representing alternative and good tradeoffs is generated. Experimental results illustrate the special features of the approach.

In [YTK95], a hybrid floorplanning methodology is proposed. Two hierarchical strategies for avoiding local optima during iterative improvement are proposed: (1) Partial Clustering, and (2) Module Restructuring. These strategies work for localizing nets connecting small modules in small regions, and conceal such small modules and their nets during the iterative improvement phase. This method is successful in reducing both area and wire length in addition to reducing the computational time required for optimization. Al-

though the method only searches slicing floorplans, the results are superior to the results obtained even with non-slicing floorplans.

In [WC95], a new approach to solve a general floorplan area optimization problem is proposed. By using the analogy between a floorplan and a resistive network, it has been shown that a class of zero wasted area floorplan can be achieved under the shape constraint of continuous aspect ratio. However, in many practical designs, each module may have constraints on its dimensions such as minimum length or width. In this paper, the authors have defined the floorplan area minimization problem under the constrained aspect ratio and give necessary conditions for the realization of zero wasted area floorplan under the shape constraints. A set of optimization methods is developed to minimize the wasted area if no zero wasted area floorplan is achievable.

6.2 Chip planning

Both floorplanning and placement problems either ignore the interconnect or consider it as a secondary objective. Chip planning is an attempt to integrate floorplanning and interconnect planning. The basic idea is to comprehend impact of interconnect as early as possible.

6.2.1 Problem Formulation

The input consists of $B_1, B_2, \ldots, B_n$ circuit blocks, with area $a_1, a_2, \ldots, a_n$ respectively. Associated with each block are two aspect ratios A_i^l and A_i^h, which give the lower and the upper bound on the aspect ratio for that block. In addition, we have $S_1, S_2, \ldots, S_m$ signals. For each signal we have criticality, width, source and sink. The chip planning algorithm has to determine the width w_i and height, h_i of each block B_i and layout of each signal such that $A_i^l \leq \frac{h_i}{w_i} \leq A_i^h$. In addition to finding the shapes of the blocks, the chip planning algorithm has to generate a valid placement for blocks and interconnect such that the area of the layout is minimized.

6.3 Pin Assignment

The purpose of pin assignment is to define the signal that each pin will receive. Pin assignment may be done during floorplanning, placement or after placement is fixed. If the blocks are not designed then good assignment of nets to pins can improve the placement. If the blocks are already designed, it may be possible to exchange a few pins. This is because some pins are *functionally equivalent* and some are *equipotential* Two pins are called functionally equivalent, if exchanging the signals does not effect the circuit. For example, exchanging two inputs of a gate does not effect the output of the gate. Two pins are equipotential if both are internally connected and hence represent the same net. The output of the gate may be available on both sides, so the out-

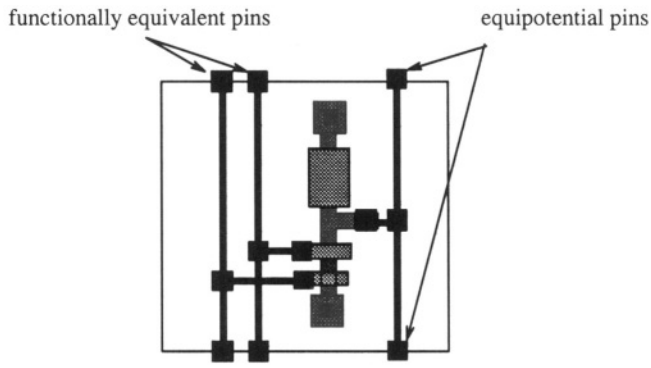

Figure 6.4: Functionally equivalent and equipotential pins.

put signal can be connected on any side. Figure 6.4 shows both functionally equivalent pins and equipotential pins.

6.3.1 Problem Formulation

The purpose of pin assignment is to optimize the assignment of nets within a functionally equivalent pin groups or assignment of nets within an equipotential pin group. The objective of pin assignment is to reduce congestion or reduce the number of crossovers. Figure 6.5 illustrates the effectiveness of pin assignment. Note that a net can be assigned to any equipotential pin within a set of functionally equivalent pins. The pin assignment problem can be formally stated as follows: Given a set of terminals $T_1, T_2, \ldots, T_n$ and a set of pins $P_1, P_2, \ldots, P_m$, $m > n$. Each T_i is assigned to pin P_i, $i = 1, 2, \ldots, n$. Let $\mathcal{E}_{P_i}$ be the set of pins which are equipotential and equivalent to P_i, the objective of pin assignment is to assign each T_i to a pin in $\mathcal{E}_{p_i}$ such that a specific objective function is minimized. The objective functions are typically routing congestions. For standard cell design, it may be the channel density.

6.3.1.1 Design Style Specific Pin Assignment Problems

Pin assignment problems in different design styles have different objectives.

1. **Full custom design style:** In full custom, we have two types of pin assignment problems. At floorplanning level, the pin location along the boundary of the block can be changed as the block is assigned a shape. This assignment of pins can reduce routing congestions. Thus, not only we can change pin assignment of pins, we can also change the location of pins along the boundary. At placement level, the options are limited to assigning the nets to pins. Notice that in terms of problem formulation,

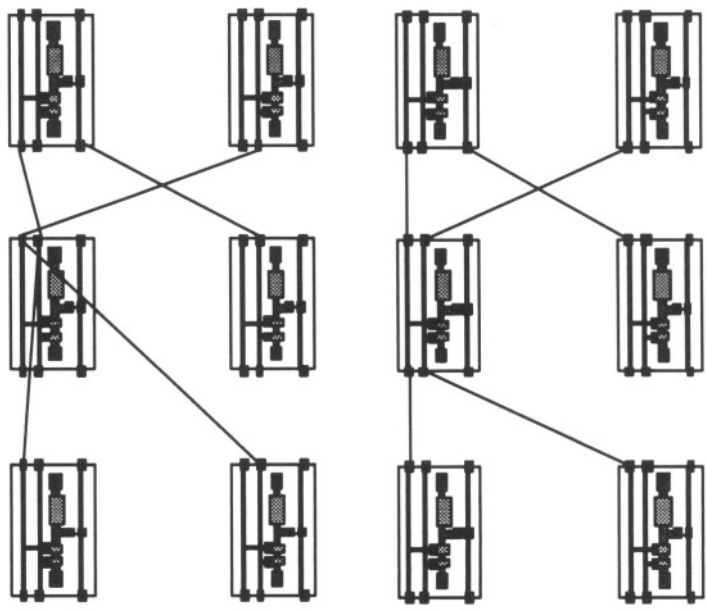

Figure 6.5: Impact of pin assignment.

we can declare all pins of a flexible block as functionally equivalent to achieve pin assignment in floorplanning.

2. **Standard cell design style:** The pin assignment problem for standard cells is essentially that of permuting net assignment for functionally equivalent pins or switching equipotential pins for a net.

3. **Gate array design style:** The pin assignment problem for gate array design style is the same as that of standard cells.

Assignment problems mostly occur in semi custom design styles such as gate arrays or standard cells.

In gate array design, the cells are pre-fabricated and are arranged on the master. Pin assignment problem in this type of design style is to assign to each terminal a functionally equivalent slot such that wiring cost is minimized. Slots in this case are the pin locations on pre-designed (library) cells. In standard cells, however, equipotential pins appear as feedthroughs. Since no wiring around the cell is needed, the wire length decreases with the use of feedthroughs.

6.3.2 Classification of Pin Assignment Algorithms

The pin assignment techniques are classified into general techniques and special pin assignment techniques. General techniques are applicable for pin

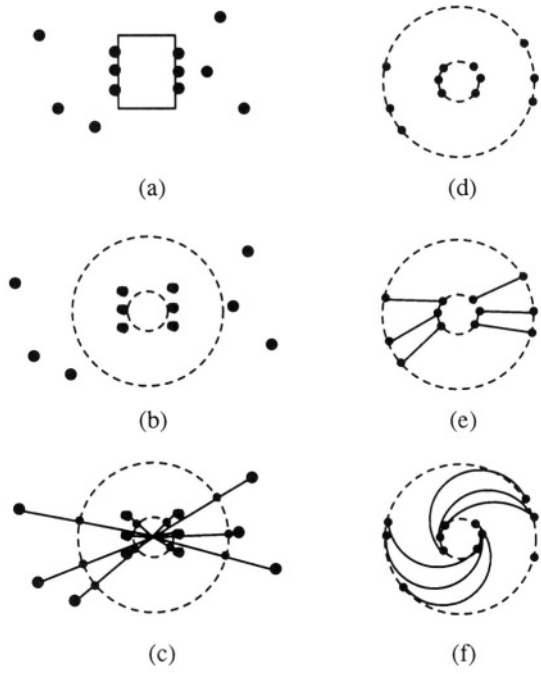

Figure 6.6: Concentric circle mapping.

assignment at any level and any region within a chip. Such techniques are applied at board level as well as chip level. On the other hand, the special pin assignment technique can be used for assignment of pins within a specific region such as channel or a switchbox.

6.3.3 General Pin Assignment

There are several methods in this category as discussed below:

1. **Concentric Circle Mapping:** To planarize the interconnections, this method models the pin assignment problem by using two concentric circles [Kor72]. The pins on the component being considered are represented as points on the inner circle whereas the points on the outer circle represent the interconnections to be made with other components. The concentric circle mapping technique solves the pin assignment problem by breaking it into two parts. The first part is the assignment of pins to points on the two circles and in the second part the points on the inner and outer circles are mapped to give the interconnections.

   For example, consider the component and the pins shown in Figure 6.6(a). The two circles are drawn so that the inner circle is inside all the pins

on the component being considered while the outer circle is just inside the pins that are to be connected with the pins of this component. This is shown in Figure 6.6(b). Lines are drawn from the component center to all these pins as shown in Figure 6.6(c). The points on the inner and outer circle are defined by the intersection of these lines with the circles (Figure 6.6(d)). The pin assignment is completed by mapping the points on the outer circle to those on the inner circle in a cyclic fashion. The worst and the best case assignment are shown in Figure 6.6(e) and (f).

2. **Topological Pin Assignment:** Brady [Bra84] developed a technique which is similar to concentric circle mapping and has certain advantages over the concentric circle mapping method. With this method it is easier to complete pin assignment when there is interference from other components and barriers and for nets connected to more than two pins. If a net has been assigned to more two pins than the pin closest to the center of the primary component is chosen and all other pins are not considered. Hence in this case only one pin external to the primary component is chosen. The pins of the primary component are mapped onto a circle as in the concentric circle method. Then beginning at the bottom of the circle and moving clockwise the pins are assigned to nets and hence they get assigned in the order in which the external pins are encountered. For nets with two pins the result is the same as that for concentric circle mapping.

3. **Nine Zone Method:** The nine zone method, developed by Mory-Rauch, is a pin assignment technique based on zones in a Cartesian coordinate system [MR78]. The center of the coordinate system is located inside a group of interchangeable pins on a component. This component is called *pin class.* A net rectangle is defined by each of the nets connected to the pin class. There are nine zones in which this rectangle can be positioned as shown in Figure 6.7. The positions of these net rectangles are defined relative to the coordinate system defined by the current pin class.

6.3.4 Channel Pin Assignment

In design of VLSI circuits, a significant portion of the chip area is used for channel routing. Usually, after the placement phase, the positions of the terminals on the boundaries of the blocks are not completely fixed and they still have some degree of freedom to move before the routing phase begins. Figure 6.8 shows how channel density could be reduced by moving the terminals. Figure 6.8(a) shows a channel which needs three tracks. By moving the pins, the routing can be improved such that it requires one track as shown in Figure 6.8 (b). The channel pin assignment problem is the problem of assigning positions for the terminals, subject to constraints imposed by design rules and the designs of the previous phases, so as to minimize the density of the channel. The problem has various versions depending on how the pin assignment

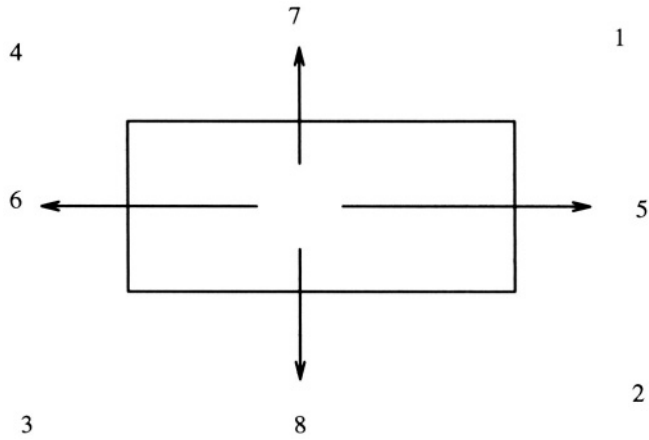

Figure 6.7: The nine pin zones.

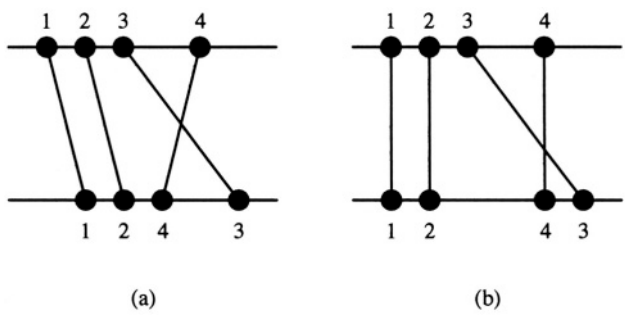

Figure 6.8: Reducing channel density by moving terminals.

constraints are specified. Many special cases of this problem have been investigated. In [GCW83], Gopal, Coppersmith, and Wong considered the channel routing problem with movable terminals.

In [YW91] the channel pin assignment problem in which assignment of terminals is subject to linear order position constraints is solved using a dynamic programming formulation by Yang and Wong. Their method is described briefly below. Except minor changes for clarity, the discussion is essentially the same as it appears in [YW91].

Since the terminals are linearly ordered we have a set of terminals at the top given by TOP in which the terminals $t_1 < t_2 < \ldots < t_p$. Similarly the terminal set for terminals at the bottom is given by BOT in which the terminals are ordered $b_1 < b_2 < \ldots < b_q$. Each terminal t_i on the top and b_i on the bottom have a corresponding set given by T_i and B_i which indicate the possible

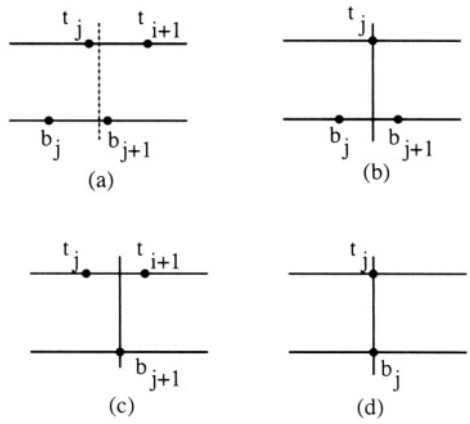

Figure 6.9: Four types of (i, j, k) solutions.

positions these terminals can occupy. A solution to this problem is called an (i,j,k)-*solution* if it assigns exactly $t_0, t_1, t_2, \ldots, t_i, b_0, b_1, b_2, \ldots, b_j$ to the first k columns of the channel where t_0 and b_0 correspond to a auxiliary column and two trivial nets which consist of only one terminal are introduced. The main idea used by the algorithm is to first compute a density function using dynamic programming and then use backtracking to reconstruct an optimal solution.

Let L be the length of the channel and N be the number of nets to be routed. The set of terminals on the top boundary of the channel is denoted as TOP, and the set of terminals on the bottom boundary of the channel is denoted as BOT. In this implementation, separation constraints and position constraints are not considered and but the terminals are definitely allowed to be within a certain position only i.e. the length L of the channel. The (i, j, k)solutions can be classified into the following four types, according to pin assignment at column k as illustrated in Figure 6.9.

- Type 0: No terminal is assigned to either endpoints of column k as shown in Figure 6.9(a).
- Type 1: Only t_i is assigned to the top endpoint of column k as shown in Figure 6.9(b).
- Type 2: Only b_j is assigned to the bottom endpoint of column k as shown in Figure 6.9(c).
- Type 3: Both t_i and b_j are assigned to column k as shown in Figure 6.9(d).

Let $d(i, j, k)$ be the density of the channel considering i terminals at the top and j terminals at the bottom after consideration of k columns. Let $x(i, j, k)$, $y(i, j, k)$ and $z(i, j, k)$ be the local densities (*crossing numbers*) at

column k considering i nets at the top and j nets at the bottom for Type 1, Type 2 and Type 3 solutions respectively. Let $R_1(i,j)$ denote the set of nets with one terminal in $\{t_1, t_2, \ldots, t_{i-1}, b_1, b_2, \ldots, b_j\}$ and one terminal in TOP $\cup$ BOT - $\{t_1, t_2, \ldots, t_i, b_1, b_2, \ldots, b_j\}$, and the net containing t_i, if it is not trivial. Let $R_2(i,j)$ denote the set of nets with one terminal in $\{t_1, t_2, \ldots, t_i, b_1, b_2, \ldots, b_{j-1}\}$, and one terminal in TOP $\cup$ BOT - $\{t_1, t_2, \ldots, t_i, b_1, b_2, \ldots, b_j\}$, and the net containing b_j, if it is not trivial. Let $R_3(i,j)$ denote the set of nets with one terminal in $\{t_1, t_2, \ldots, t_{i-1}, b_1, b_2, \ldots, b_{j-1}\}$, and one terminal in TOP $\cup$ BOT -$\{t_1, t_2, \ldots, t_i, b_1, b_2, \ldots, b_j\}$, and the net containing t_i or b_j, if it is not trivial and if they do not belong to the same net. Hence we have

$$x(i,j,k) = \begin{cases} +\infty & \text{if } K \notin T_i \\ |R_1(i,j)| & \text{otherwise} \end{cases}$$

$$y(i,j,k) = \begin{cases} +\infty & \text{if } K \notin B_j \\ |R_2(i,j)| & \text{otherwise} \end{cases}$$

$$z(i,j,k) = \begin{cases} +\infty & \text{if } K \notin T_i \cap B_j \\ |R_1(i,j)| & \text{otherwise} \end{cases}$$

The algorithm for optimal channel pin assignment is shown in Figure 6.10. It is easy to see that the time complexity of the algorithm Linear-CPA is $O(pql)$.

6.4 Integrated Approach

The various stages in the physical design cycle evolved as the entire problem is extremely complex to be solved altogether at once. But over the years, with better understanding of the problems, attempts are made to merge some steps of the design cycle. For example, floorplanning was considered as a problem of just finding the shapes of the blocks without considering routing areas. Over the years, the floorplanning problem has been combined with the placement problem [DEKP89, SSR91, WL86]. The placement problem is sometimes combined with the routing problem giving rise to the 'place and route' algorithms [Esc88, FHR85, SSV85, Sze86].

In this section, the approach used by Dai, Eschermann, Kuh and Pedram in BEAR [DEKP89] is described briefly. BEAR is a macrocell-based layout system. The process of floorplanning is carried out in the following three steps:

1. **Clustering:** In this step, a hierarchical tree is constructed. Blocks that are strongly connected are grouped together in a cluster. Each cluster can have a limited number of blocks within it. The clustering process considers the shapes of the blocks to avoid a mismatch within the cluster. This step is repeated to build the cluster tree.

2. **Placement:** In this step, the tree is traversed top-down. The target shape and terminal goals for the root of the cluster tree is specified. This information is used to identify the topological possibility for the

```
Algorithm LINEAR-CPA()
begin
    (* initialize *)
    for i = 0 to p do
        for j = 0 to q do
            d(i, j, 0) = +∞;
    for k = 0 to L do
        d(0, 0, k) = 0;
    COMPUTE-CROSS();
    for k = 1 to L do
        for i = 0 to p do
            for j = 0 to q do
                (* type 1 solution *)
                D1 = max{d(i − 1, j, k − 1), x(i, j, k)};
                (* type 2 solution *)
                D2 = max{d(i, j − 1, k − 1), y(i, j, k)};
                (* type 3 solution *)
                D3 = max{d(i − 1, j − 1, k − 1), z(i, j, k)};

    if d(p, q, l) = +∞ then
        return φ is not feasible;
    else
        (* backtracking for constructing optimal solution *)
        i = p; j = q;
        for k = L down to 1 do
            if d(i, j, k) = D1 then
                f(i) = k; i = i − 1;
            else if d(i, j, k) = D2 then
                g(j) = k; j = j − 1;
            else if  d(i, j, k) = D3 then
                f(i) = k; g(j) = k;
                i = i − 1;j = j − 1;
        return π = (f, g);
end.
```

Figure 6.10: The optimal channel pin assignment algorithm

```
Procedure COMPUTE-CROSS()
begin
    for (i = 0 to p)do
        for (j = 0 to q) do
            COMPUTE( R1(i,j) );   (* Compute |R1(i,j)| *)
            COMPUTE( R2(i,j) );   (* Compute |R2(i,j)| *)
            COMPUTE( R3(i,j) );   (* Compute |R3(i,j)| *)
            for (k = 0 to L) do
                x(i,j,k) = +∞;
                y(i,j,k) = +∞;
                z(i,j,k) = +∞;
    for(i = 0 to p) do
        for(j = 0 to q)do
            for( k ∈ Ti ) do
                x(i,j,k) = |R1(i,j)|;
            for( k ∈ Bj ) do
                y(i,j,k) = |R2(i,j)|;
            for( k ∈ Ti ∩ Bj ) do
                z(i,j,k) = |R3(i,j)|;
end.
```

Figure 6.11: The COMPUTE-CROSS procedure for the channel pin assignment algorithm

clusters at the level below. This in turn sets the shape and terminal goals for the immediate lower levels in the hierarchy till at the leaf level the orientations of the blocks are determined. For each of the topologies, the routing space is determined. The selection of a particular topology is based on the area and the shape of the resulting topology and the connection cost. The system is developed so as to allow the user to control the trade off between the shape, the area and the connection costs. This strategy works well in case the blocks at the leaf level are flexible so that the shapes of these blocks can be adjusted to the shape of the cluster. On the other hand, if the leaf level blocks are fixed then this top-down approach can give unfavorable results. This is due to the fact that the information of the shape of these blocks at the leaf level are not considered by the objective function when determining the cluster shapes at higher levels of the cluster tree. This is rectified by passing the shape information from the leaves towards the root of the tree during the clustering phase. In addition, during the top-down placement step, a look-ahead is added so that the objective function can examine the shapes generated during clustering, at a level below the immediate level for which the shape is being determined.

3. **Floorplan optimization:** This an improvement step that resizes selected blocks iteratively. The blocks to be resized are identified by computing the longest path through the layout surface using the routing estimates done in the previous step.

6.5 Summary

Floorplanning and Pin assignment are key steps in physical design cycle. The pin assignment is usually carried out after the blocks have been placed to reduce the complexity of the overall problem.

Several placement algorithms have been presented. Simulated annealing and simulated evolution are two most successful placement algorithm. Although these algorithms are computationally intensive, they do produce good placements. Integer programming based algorithms for floorplanning have been also been successful. Several algorithms have been presented for pin assignment, including optimal pin assignment for channel pin assignment problems. The output of the placement phase must be routable, otherwise placement has to be repeated.

6.6 Exercises

1. Given the following 14 rectangles with their dimensions specified, write a program that will arrange all these rectangles within 5000 sq. units of area, if possible, or otherwise minimize the area required. The dimensions (width x height) of the rectangles are $R_1 = 15 \times 15, R_2 = 25 \times 15, R_3 = 10 \times 30, R_4 = 30 \times 20, R_5 = 10 \times 15, R_6 = 20 \times 5, R_7 = 10 \times 25, R_8 = 30 \times 15, R_9 = 10 \times 65, R_{10} = 10 \times 25, R_{11} = 20 \times 20, R_{12} = 10 \times 20, R_{13} = 30 \times 15, R_{14} = 40 \times 15$.

2. Recall that the aspect ratio of a block is the ratio of its height and width. If each rectangle in problem 1 can have three different aspect ratios, find the appropriate aspect ratio for each rectangle so that the area occupied by the rectangles is minimized. The set of aspect ratios for R_i, is R_i^a, is, $R_1^a = \{1.0, 1.2, 2.0\}$, $R_2^a = \{0.8, 1.5, 1.9\}$, $R_3^a = \{0.6, 2.0, 3.0\}$, $R_4^a = \{0.8, 1.2, 1.5\}$, $R_5^a = \{0.75, 1.2, 1.5\}$, $R_6^a = \{0.3, 0.9, 2.0\}$, $R_7^a = \{0.4, 1.2, 1.5\}$, $R_8^a = \{0.5, 1.0, 1.5\}$, $R_9^a = \{0.8, 3.0, 4.0\}$, $R_{10}^a = \{0.4, 1.2, 1.8\}$, $R_{11}^a = \{0.5, 1.0, 1.2\}$, $R_{12}^a = \{0.5, 0.9, 1.2\}$, $R_{13}^a = \{0.5, 1.0, 1, 5\}$, $R_{14}^a = \{0.4, 1.6, 2, 5\}$.

3. Use the lowest and highest aspect ratios for each rectangle in problem 2 as lower and upper bounds respectively and generate a placement which occupies minimum area.

† 4. Apply Simulated Annealing algorithm for pin assignment problem. In each iteration, pins of each block are permuted and routing congestion is estimated.

† 5. Develop an algorithm for pin assignment of a full custom layout based on concentric circle mapping.

‡ 6. Implement the channel pin assignment algorithm. Discuss the constraints, based on functionally equivalent and equipotential pins.

Bibliographic Notes

A floorplanning system designed to work within a hierarchical design environment supporting multiple design styles has been discussed in [MTDL90]. A technique for floorplanning and pin assignment of general cell layouts has been developed in [PMSK90]. A global floorplanning approach has been discussed in [PD86]. The approach is based on a combined min-cut and slicing paradigm. A pin assignment algorithm for improving the performance in standard cell design by improving the longest delay has been discussed in [SL90]. A pin assignment problem for macro-cells is discussed in [YYL88]. An approach which combines pin assignment and global routing has been developed in [Con89]. In [KK95], yield issues are considered. Authors demonstrate that for large area VLSI chips, especially those that incorporate some fault tolerance, changes in the floorplan can affect the projected yield. In [KK97], the authors have demonstrated that the floorplan of a chip can affect its projected yield in a nonnegligible way, for large area chips with or without fault-tolerance.

In [MAC98], a floorplanner for RF circuits based on a genetic algorithm that supports simultaneous placement and routing has been developed. In [MK98], Sequence-pair based placement method for hard/soft/pre-placed modules has been discussed (also discussed in chapter 7). In [ITK98], a new approach for the minimum area floorplanning is proposed where the shape of every module can vary under the constraint of area and floorplan topology. Simulating the air-pressure mechanics, the algorithms iterate to improve the layout to decide the shapes and positions of modules. In [CF98], a convex formulation of the floorplan area minimization problem is presented.