Recent Advances in VLSI Layout

ERNEST S. KUH, FELLOW, IEEE, AND TATSUO OHTSUKI, FELLOW, IEEE

Research in layout design during the past decade has had a significant impact on the entire VLSI industry. Because of advances made in automatic placement and routing, application specific integrated circuit (ASIC) design now represents a sizeable portion of the chip market. The influence of computer-aided physical design will continue to be felt in the future; for example, in further popularizing the sea-of-gates and building-block design styles. Recent advances in floorplanning and placement and in global and detailed routing pertinent to these design styles are reviewed. Also included are the subjects of computational geometry and layout engines. The former has received a great deal of attention among computer scientists and has found interesting applications in gridless routing and compaction. The latter is becoming an interesting research topic along with other hardware accelerators and special-purpose engines. Although all these subjects are carefully reviewed, much greater emphasis is placed on the discussion of the authors' own research.

I. Introduction

Computer-aided design has made a major impact in advancing the state of the art of microelectronics in the past decade. This is especially true in the area of physical design, where automatic and interactive design tools have become indispensable to the layout of VLSI chips. The turnaround time for layout design has been reduced from months and weeks to days and hours. Standard-cell and gate-array technology are now prevalent in application specific integrated circuits (ASIC) design. Even for high-volume custom chips such as microprocessors, CAD programs for floorplanning, placement, and routing are essential. As chips become increasingly complex, larger in dimension, and smaller in feature size, more sophisticated algorithms and layout software are clearly necessary. The sea-of-gates design style, for example, calls for efficient algorithms which deal with hundreds of thousands of gates. Previously useful algorithms based on simulated annealing or other random methods have become unusable: the computation time is unbearable in spite of the growing accessibility of faster computers.

Manuscript received April 11, 1989; revised August 8, 1989. This work was supported in part by the Semiconductor Research Corporation, in part by the National Science Foundation under grant MIP-88-3711, and in part by the Japan Society for the Promotion of Science.

E.S. Kuh is with the Department of Electrical Engineering and Computer Sciences and the Electronics Research Laboratory, University of California, Berkeley, CA 94720.

T. Ohtsuki is with the Department of Electronics and Communication Engineering, Waseda University, Tokyo 169, Japan.

IEEE Log Number 8934098.

The current status of VLSI layout and the challenges in research for the future are the issues we will attempt to address. In an earlier special issue on computer-aided design, published in 1981, Soukup wrote an excellent review article on circuit layout [1]. Since then, several edited books on the subject have also appeared [2]–[4]. Because of page limitations we will not cover many of the problems and methods which appeared in these earlier review publications. For example, channel routing, although used extensively in current-day chip layout, is now a mature subject and will not be discussed. Others have become obsolete; gate matrix design, for instance, with the prevalent use of multiple metal layers. Simulated annealing, although useful in some applications, will not be included because its effectiveness in solving large and complex problems remains to be seen.

The standard-cell and gate-array design styles best illustrate the power of CAD. There are many good in-house, university, and commercial software packages. For example, the Timberwolf package [5], [6], which is based on simulated annealing, has proven to be an effective tool for small- and medium-size designs and is widely used. Therefore we will not review the methods used for placement and routing which are pertinent to standard-cell and gate-array designs. Instead, we have decided to tailor our discussions to sea-of-gates and building-block (macrocell or general cell) designs. The building-block design style, which was first introduced at NEC [7] in the mid 1970's, has gradually been accepted in industry, especially in Japan [8], [9] and among those companies that deal with silicon compilation. However, because of the complexity involved, there is a lack of commercial software that offers not only versatile interactive features but also the capability of obtaining efficient layout with an area usage comparable to manual designs. The potential of building-block layout clearly has not been realized.

Roughly speaking, VLSI layout refers to the transformation of the functional and logic specifications of a desired chip to physical realization of transistors, cells, and macros together with their interconnection pattern. This in turn leads to a full mask specification in multiple layers with detailed geometrical data and processing information for chip production. The task is complicated, and it is impossible to speak of the truly optimum design. What is possible is to cleverly break up the overall problem into subproblems which then can be managed and possibly optimized.

0018-9219/90/0200-0237$01.00 © 1990 IEEE

The well-known subproblems are partitioning, floorplanning, placement, global routing, and detailed routing. Each subproblem can then be formulated more precisely subject to the design styles. Within each subproblem, if suitable modeling is employed, there are problems that can be specified mathematically. Unfortunately, almost all such problems encountered in layout turn out to be NP hard. Roughly speaking, it is unlikely that the exact solution could be found in polynomial time. Furthermore, the objective function often does not reflect all the criteria that are ultimately desired. Therefore heuristics have played a major role in developing layout software. For the benefit of readers with no background in layout, we will try to give a concise overview of the subject, including necessary terminology and definitions, and whenever possible, a precise formulation of the problems. We will review some recent approaches but will emphasize our own work, which has yielded good overall results.

In Section II we will address placement and floorplanning for both the sea-of-gates and building-block designs. The former emphasizes the connectivity specification while the latter must also consider module shape and size. In Section III we will discuss global routing based on a method of successive cuts on a chip. This is a hierarchical top-down approach which is useful for both the sea-of-gates and the building-block designs. Next we will discuss a two-dimensional detailed routing problem. Finally, we will review the rip-up and rerouting problem.

In Section IV we will review the field of computational geometry and its application to layout—in particular, to gridless routing and compaction. Computational geometry has received a great deal of attention recently among computer scientists and could be a fruitful area of research in layout. In Section V we will discuss layout engines. Recent hardware advances and the increasing accessibility of parallel machines and supercomputers create an interesting debate as to the eventual fate of hardware engines for layout.

Finally, we will conclude with a brief mention of some of the topics of interest not treated in the paper.

II. Placement and Floorplanning

The placement problem for VLSI design is to place modules on a plane based on the given information on module interconnection. The purpose of placement is to facilitate routing with an overall objective; for example, to minimize the area of the resulting chip. The placement problem as stated is a two-dimensional problem; however, sometimes we encounter one-dimensional or linear placement. A typical example of linear placement is the placement of a linear array, such as in gate matrix layout. The placement problem is usually solved in two successive states: constructive placement and iterative improvement. Numerous methods have been proposed: clustering from bottom up, partitioning from top down, force directed, resistive network analogy, eigenvalue approach, etc. for constructive placement; and two-way or multiway exchange, simulated annealing, or other random methods for iterative improvement.

The interconnection specification is usually given by means of a *net list*. It specifies the precise topological relation between nets and pins of modules. Nets can be two-terminal or multiterminal depending on whether they are to connect two pins or many pins. By modules we mean gates, cells, macros, or pads. The term macro, or macrocell, is used here to designate large cells whose shape and area must be considered in placement. Thus the placement problem can also be classified into two kinds: point-module placement and macrocell placement. Constructive placement for gate-array and standard-cell design is usually treated as a point-module placement problem, i.e., we ignore the size of the gate or cell. However, for iterative improvement, we need to take into account the position of the pins of the cell. In building-block layout, we consider the macrocell placement problem. It is more difficult because, in addition to the topological specification of the net list, we must consider the geometrical specification of the macros or building blocks. Although some macros may be rectilinear, we consider only rectangular modules, for simplicity. Even for rectangular modules, macrocell placement is a difficult problem. It is related to the simpler bin-packing problem, which is known to be NP complete. In bin packing there is no net list specification; the sole purpose is to fit rectangular modules into a rectangle of minimum area.

The floorplanning problem is related to the macrocell placement problem in that some modules may be soft, i.e., they have a fixed area but a variable aspect ratio. In addition, pin positions may need to be optimized. Typical examples of soft modules are PLA and logic blocks created by standard-cell design. Thus macrocell placement may be considered a special case of the floorplanning problem in which all modules are hard, i.e., with fixed area and aspect ratio. Obviously, floorplanning is a harder problem since it offers additional degrees of freedom to optimize the total layout area.

The quality of placement or floorplanning cannot be easily measured. Ideally, we need to complete the routing and obtain the total area. This is clearly not feasible because routing cannot begin without placement. Therefore a criterion based on some measure of wire length is usually used. Since routing is almost always carried out with horizontal and vertical wires, we may use the Manhattan geometry. A commonly accepted measure for placement is the sum of net length based on the half-perimeter of the enclosing rectangle of all the pins which belong to a net. In the case of a two-terminal net, this measure represents the shortest direct route of a Manhattan connection. For a three-terminal net, this is the length of a rectilinear Steiner tree, as shown in Fig. 1; thus it also equals the shortest horizontal/vertical connection. For a multiterminal net, the half-perimeter length is, at best, a crude approximation. Fortunately, statistically speaking, over 75 percent of the nets of a typical chip are two-terminal or three-terminal. Thus the half-perimeter measure does give a good indication of the quality of the placement.

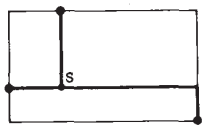


Fig. 1. Half-perimeter wire length is equal to length of Steiner tree for three-termimal net. *S* is Steiner point.

When point-module is used in the formulation, the net list must be modified first by merging all the pins which belong to a module. The resulting net list can be characterized by a 0–1 incidence matrix, $\boldsymbol{A} = [a_{ij}]$, where a_{ij} is zero if net i is not connected to module j; it is equal to one if net i is connected to module j. Another commonly used characterization for the net list is in terms of the connectivity matrix $\boldsymbol{C} = [c_{ij}]$; however, it is restricted to two-terminal nets. For an n-module problem, the $n \times n$ connectivity matrix is symmetric with $c_{ii} = 0$, and the off-diagonal term c_{ij} represents the total number of connections between module i and module j. The connectivity matrix has nice properties. As seen later, it makes the formulation of the optimization problem obvious. Therefore it is important to be able to use the connectivity matrix to treat a general net list which consists of not just two-terminal nets. The three possible representations of a multiterminal net in terms of equivalent two-terminal nets are illustrated in Fig. 2, where a weighted clique of a four-terminal net is shown in Fig. 2(a), a minimum

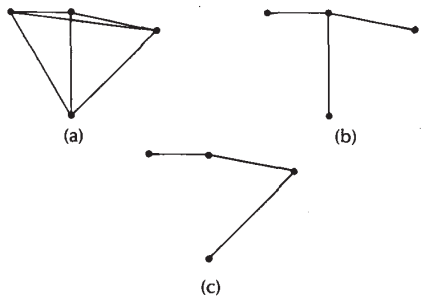


Fig. 2. Representations of four-terminal net in following terms. (a) Weighted clique. (b) Minimum spanning tree. (c) Linear tree.

spanning tree is shown in Fig. 2(b), and a linear tree is shown in Fig. 2(c). Depending on the application, one representation may be better than another. In linear placement, for example, the linear tree representation may be the best one to use.

With the introduction of the connectivity matrix, the point-module placement problem can be formulated in terms of an objective function which measures the sum of the squared wire lengths of two-terminal nets. Let x_i, y_i denote the coordinate of the ith module. The squared wire length objective function is defined as

$$L(\boldsymbol{x}, \boldsymbol{y}) = \tfrac{1}{2} \sum_{i,j=1}^{n} c_{ij} [(x_i - x_j)^2 + (y_i - y_j)^2] = \boldsymbol{x}^T \boldsymbol{B}\boldsymbol{x} + \boldsymbol{y}^T \boldsymbol{B}\boldsymbol{y} \tag{1}$$

where $\boldsymbol{B} = [b_{ij}]$ is a modified connectivity matrix defined by

$$\boldsymbol{B} = \boldsymbol{D} - \boldsymbol{C} \tag{2a}$$

and thus

$$b_{ij} = -c_{ij}, \quad \text{for } i \neq j \tag{2b}$$

and $\boldsymbol{D}$ is a diagonal matrix with

$$d_{ii} = \sum_{j=1}^{n} c_{ij}. \tag{2c}$$

The point-module placement problem is then to minimize $L(\boldsymbol{x}, \boldsymbol{y})$ subject to the slot constraint, i.e., all modules are to be located on a two-dimensional grid, called the legal position. Since the coordinate vectors x and y enter separately in the sum of two quadratic forms in (1), we may consider each coordinate independently, which amounts to considering the linear placement problem. The slot constraints for linear placement can be expressed by a set of n algebraic equations in $\boldsymbol{x}$, in terms of the coordinates of the legal positions [10]. Let P_i represent the ith legal position, then the constraint equations are

$$\begin{aligned} \sum_{i=1}^{n} x_i &= \sum_{i=1}^{n} P_i \\ \sum_{i=1}^{n} x_i^2 &= \sum_{i=1}^{n} P_i^2 \\ &\vdots \\ \sum_{i=1}^{n} x_i^n &= \sum_{i=1}^{n} P_i^n. \end{aligned} \tag{3}$$

The problem of minimizing the quadratic objective function $L = \boldsymbol{x}^T \boldsymbol{B}\boldsymbol{x}$ subject to the constraint equation in (3) is clearly difficult if not impossible.

The eigenvalue approach first introduced by Hall [11] can be shown to be equivalent to minimize the quadratic objective function $\boldsymbol{x}^T \boldsymbol{B}\boldsymbol{x}$ subject to the constraints of the first two equations in (3). By taking the two smallest nonzero eigenvalues of matrix $\boldsymbol{B}$ and their associated eigenvectors, one obtains the coordinates of the point-modules on a plane. We call the solution the *global placement*. The modules so obtained will not be on slots since only two constraint equations are used in the formulation. Thus they must be moved onto slots by successive partitioning and/or iteration. The method is useful for small or medium-sized gate-array placement. Since determining the eigenvectors of a large matrix is a difficult numerical problem it is clearly not feasible for sea-of-gates design.

A. Sea-of-Gates Placement

Sea-of-gates design differs from gate-array design in two aspects. First, in usual sea-of-gates designs, no routing channel is used, which means all routing is done over the gates. Second, the total number of gates could be very large, for example 50K or more, so computation algorithms that can handle very large problems efficiently are essential. Unlike the conventional gate array, there are very few software packages for sea-of-gates design.

The input to placement is in terms of a net-list for the cells. Each cell represents either a gate or an interconnected set of gates which form a basic logic element. The pads are placed at the periphery of the chip of I/O and power and ground connection. We assume that there are two or three metal layers for interconnection of the cells; however, throughout the chip there are blockages at various locations with one or more metal layers representing committed internal cell connection. The objective of the placement is to locate all the cells and pads in such a way that nets are uniformly distributed across the chip. To achieve 100-percent connection, which is required in gate-array technology, some of the active devices cannot be utilized because

their space must be reserved for routing. Thus the effectiveness of a sea-of-gates design can be measured by the percentage of gate utilization.

One of the main concerns in sea-of-gates placement is the size of the problem. To deal with more than 50K gates on a chip, hierarchical layout approaches have been proposed. One commonly used method is to divide the layout of a chip recursively into the layout of a set of smaller blocks, and place and route these blocks at a higher level using building-block techniques [12]. A series of steps needs to be addressed in the process: definition of layout hierarchy, area estimation, and aspect ratio assignment for each block; pin location assignments; block placement; and block routing. Some of these will be discussed later. In this subsection we briefly present the Proud placement program developed by Tsay at Berkeley [13]. The program is to be used either for a partitioned block of a chip or a chip itself if the gate count is not excessive. The constructive placement is based on a resistive network analogy which takes advantage of the sparsity of the net list specification. As we know, in circuit simulation sparse matrix algorithms have proven to be effective in handling circuits with many thousands of nodes.

Consider the electric network analogy of the placement problem. The objective function $\boldsymbol{x}^T \boldsymbol{B}\boldsymbol{x}$ can be interpreted as the power dissipation of an n-node linear resistive network, with $\boldsymbol{x}$ corresponding to an n-vector whose components represent the voltages at the nodes. The modified connectivity matrix $\boldsymbol{B}$ is exactly the indefinite admittance matrix of the n-node resistive network with $-b_{ij}$ equal to the conductance between node i and node j. The placement problem is then equivalent to that of choosing the node voltages of the n-node network for which the power dissipation is a minimum. In electric network theory it is well known that minimum power dissipation is implied by the two Kirchhoff laws. Thus we can reformulate our linear placement problem in terms of a linear resistive network problem provided that we include the I/O pad specifications. This allows us to model the I/O pads as fixed voltage sources applied to the network. The coordinates of the movable modules are then the node voltages to be determined. Therefore only sparse linear equations need to be solved. In the program Proud, the successive over-relaxation method is used to compute $\boldsymbol{x}$.

The result of the global placement gives the optimal solution in terms of the original objective function $L(\boldsymbol{x}, \boldsymbol{y})$. This is because it is a convex quadratic function, so there exists a unique global minimum. In the real situation, modules occupy finite non-zero area and their shapes vary. While the slot constraint is not pertinent to the global placement problem, replacing the point modules with real modules may cause module overlaps at this step. We will specifically address the overlap problem and propose an efficient partitioning method with iterations to improve the initial solution. The key feature is that we again resort to solving linear sparse equations. We repeat the process in a top-down hierarchy to obtain the final solution. At each hierarchy, the placement interactions between the two partitioned parts will be handled by the block Gauss–Seidel (BGS) iteration method to obtain a solution which is close to the global one. In the following, we will first elaborate the partitioning scheme.

As shown in Fig. 3, we introduce a *vertical* cut line to partition the chip into two. The location of the cut is determined by the area consideration based on the positions of the modules obtained from the initial global placement. We sort the modules from left to right and add up the module areas until roughly half of the total module area is reached. The cut is then made. If the cut line happens to coincide with the center line, we proceed to the next hierarchy level by performing horizontal cuts on each half. If not, we must move the cut line to the center in order to satisfy the area usage requirement. This calls for a modification of the placement in both halves. To achieve this, we introduce a simple heuristic, as follows.

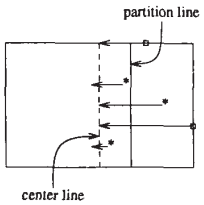


Fig. 3. Projecting modules from right half of plane to center line to determine left plane placement.

Assume without loss of generality that the cut line is to the right of the center, as shown in Fig. 3. All modules to the right of the center line are projected horizontally on the center line as shown. We then solve the global placement problem in the left half-plane by the method outlined in the aforementioned. Note that, among the projected modules, only those modules which lie between the cut line and the center line will be included in the set of movable modules, whereas the others will be grouped with the fixed modules. We next repeat the placement algorithm for the right half-region. However, before that, a temporary top-bottom two-way partition on the left half-plane is determined. These modules are then projected on the center line as fixed modules to solve the right half-plane problem. This process is repeated two or more times before the partition becomes fixed. We then reach the next hierarchy and proceed to find the global placements of the top and bottom quarters of each half. Eventually, after repeated partitioning, each block contains one and only one module with a legal location to which the module will be assigned. Since the partitioning scheme takes into account the available area, the resulting placement will usually have no overlaps.

The preceding concludes the constructive placement phase of the sea-of-gates Proud program. For iterative improvement, we consider actual pin position in evaluating an objective function based on the half-perimeter net length. Pairwise interchange and module insertion are used, but they are limited to within a window which covers only a portion of the entire chip. The size of the window is determined by a compromise between computation time and the quality of the final result.

For an experimental chip with 100K gates, the computation time of the Proud program on a VAX 8650 (a 6 MIPs machine) for the constructive placement phase is 50 minutes, and for the iterative improvement phase is about three hours. The proud program also contains features to place macros combined with basic cells. An algorithm has been developed to place the macros first. The pins of the macros must be considered and are treated like the I/O pads to

obtain a global solution. The hierarchical partitioning scheme must be modified to take into account the locations of the boundaries of the macros.

B. Building-Block Placement and Floorplanning

In Fig. 4 we show the result of placement and routing for a 33-block industrial chip done by the BEAR program, which was recently completed at Berkeley [14]. The placement part took 762 seconds on a VAX 8800. Note that in Fig. 4(a) routing space is provided based on a hierarchical wiring area estimation program. Crucial to building-block placement is the allocation of routing space on the routing plane between blocks. As a matter of fact, the problem of routing region

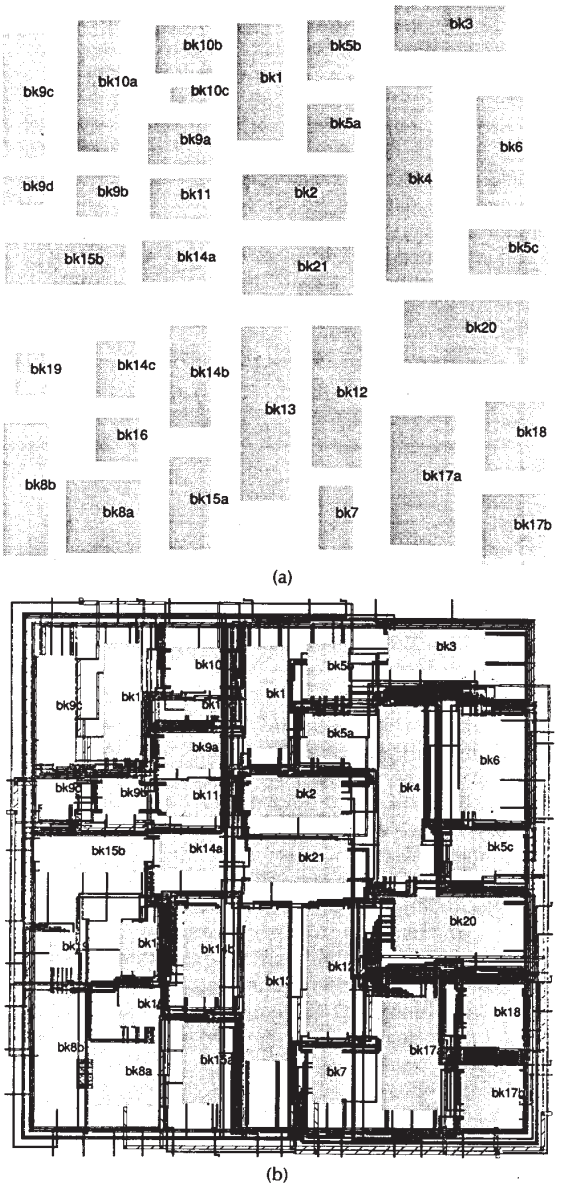


Fig. 4. Output of BEAR building-block layout system of industrial example, primBBL2, with 33 blocks and 123 nets. (a) Placement with routing area estimation. (b) Routing.

definition and ordering is an important aspect of building-block layout [15]. Furthermore, when the chip is routed, modules will be moved to accommodate the routing space required. Thus placement and routing interact; it is much more difficult to measure the quality of a placement alone, as in the case of the point-module placement. What really counts is the final total area after a chip is routed. We believe that wiring area estimation is an important part of placement and floorplanning using macrocells. Unfortunately, most published work ignores this important aspect. There has been, however, recent interest in wiring area estimation for layout design [16], [174], [175].

We will briefly review some pertinent work in floorplanning and placement; for example, the slicing structure, the top-down partitioning min-cut method, the polar graph representation, the shape function, the method of rectangular dualization, and the macrocell placement approach based on optimization. As indicated earlier, simulated annealing will not be included; however, it should be noted that programs like Mosaico [176] and others [177]–[179] have shown promise. We will briefly describe our own work used in the BEAR system, which combines bottom-up clustering with top-down partitioning. In BEAR, the problems of floorplanning and placement are treated in a uniform way, in hierarchical fashion, with only slight differences at the bottom hierarchy or the leaf level, where all hard modules must be used for placement.

The macrocell placement problem can be formulated as a nonlinear programming problem. Instead of using point modules, Sha and Dutton [17] introduced a line-segment representation of a rectangular macro to approximately depict its geometry. They used an objective function in terms of the squared wire length, and carried out the optimization subject to the nonoverlap constraints of the macros. The nonoverlap constraints are inequalities derived from the line-segment model. Rotation and mirroring of modules can be taken into account. A penalty function method and a sequential quadratic programming method have been proposed to find a solution. However, neither method guarantees a global minimum solution. Furthermore, because the nonoverlap constraints are based on a line-segment approximate representation, the method may even fail to ensure module nonoverlap. The required computation time is of the same order of magnitude as that of simulated annealing. Also, there is no routing-area estimation.

Placement based on min-cut partitioning [18] has been shown to be useful in standard-cell, gate-array, and building-block layout. Lauther [19] first demonstrated the effectiveness of min-cut partitioning in macrocell placement and, by introducing a graph representation, he established a close tie between the placement problem and floorplanning for the slicing structure. The basic idea is to group together those modules that are tightly connected and separate those that are minimally connected. This is accomplished by successive min-cut bipartitioning (two-way partitioning) in alternate vertical and horizontal directions. A modified Kernighan–Lin algorithm [20] is used. However, one key aspect of Lauther's method is to compute the sum of the areas of the macros of the two partitioned sets and make sure that the area difference between the two does not exceed a predefined threshold value. This criterion ensures that the min-cut partition will not lead to trivial

results, e.g., a single macro separated from the rest, and thus a uniform distribution of macros can be obtained. The same idea is used in the Mason floorplanner [21] developed at Carnegie-Mellon University. However, they introduce an additional feature called "in-place partitioning," which considers module connectivity in terms of terminal locations. At the end of the successive min-cut partitioning process, all modules are assigned to rectangular blocks which satisfy the area specification of each macro. Lauther's method contains a second phase which adjusts the shapes of the blocks to satisfy the aspect ratio of each macro and meanwhile does further optimization to reduce the total wire length by incorporating rotation and mirroring of the macros.

Clearly the first phase of Lauther's method is well suited to slicing-structure floorplanning since the area but not the dimensions of macros are considered in each min-cut bipartitioning. The use of slicing structure for floorplanning was first proposed by Otten and Szepieniec [22]. By slicing structure, we mean a floorplan which can be obtained by a successive slicing process. Fig. 5(a) illustrates

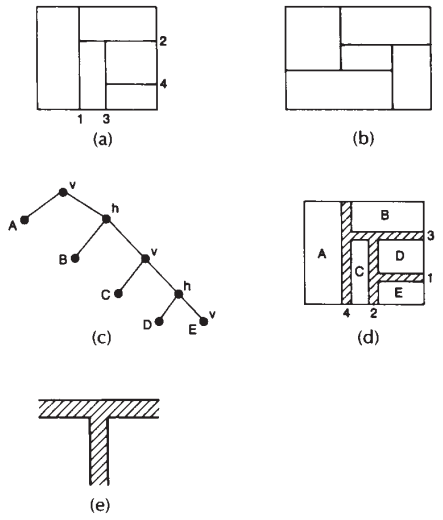


Fig. 5. Floorplanning for building-block layout. (a) Slicing structure with order of slicing indicated. (b) Nonslicing structure. (c) Tree representation of slicing structure. (d) Order of channel routing. (e) T-junction at two channels.

a slicing floorplan with 1, 2, 3, and 4 indicating the slicing sequence, while Fig. 5(b) gives the simplest structure, which is nonslicing. To be more precise, we start with a rectangular chip and cut it by $k \geq 1$ parallel line segments into $k + 1$ rectangles. The direction of cut may be either vertical or horizontal. The resulting $k + 1$ rectangles can then be cut again by parallel line segments in the perpendicular direction from the first. The process continues until, in the end, each rectangular block, sometimes referred to as a room, is fitted by a macrocell. Note that in the nonslicing example of Fig. 5(b), slicing cannot be done at the outset.

It is useful to represent a slicing structure by a slicing rooted tree. Fig. 5(c) gives the slicing tree representation of the slicing floorplan in Fig. 5(a). The root of the tree represents the whole chip. The set of nodes at the next level represents the resulting rectangles after the first stage of slicing. In the example in Fig. 5(a), it is done in the vertical direction. A marker *h* or *v* is used on each level of the slicing tree to depict the direction of slicing. Thus Lauther's method of min-cut bipartitioning always leads to a binary tree.

The slicing floorplan has a distinct advantage in that routing of the entire chip can be completed by using only a channel router. Furthermore, it is easy to demonstrate that the routing order turns out to be the reverse order of the slicing. For the example of the slicing floorplan in Fig. 5(a) its routing order is shown in Fig. 5(d). The essence of channel ordering is illustrated by a T-junction of two channels shown in Fig. 5(e). Clearly the vertical base channel must be routed first; its result, i.e., the width and floating terminal information, is needed to route the horizontal bar channel. In the case of a nonslicing floorplan, as in Fig. 5(b), channels cannot be defined unambiguously; there is always a cyclic conflict, and a channel router alone cannot complete the routing [15], [180].

A floorplan, whether of the slicing or nonslicing structure, can be represented by a pair of dual graphs, the vertical polar digraph and the horizontal polar digraph. The polar graph originated in the early 1940's in the problem of dissection of rectangles into squares [23]. Its use in IC design was first introduced by Ohtsuki, Suziyama, and Kawanishi [24]. Fig. 6 shows the vertical and horizontal polar digraphs

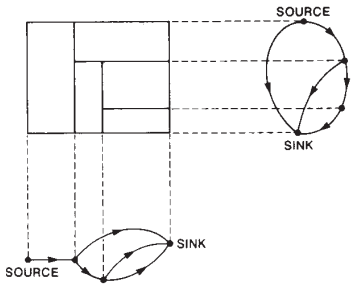


Fig. 6. Polar digraph representations of slicing structure.

of the floorplan in Fig. 5(a). In the vertical polar digraph, the top boundary of the rectangular chip is represented by a source and the bottom boundary of the chip is represented by a sink. Each slice in the horizontal direction is represented by a node. A directed edge exists between two nodes if there exists a block whose top boundary represents one node and whose bottom boundary represents the other node. The direction of the edge is from the node representing the top boundary to the node representing the bottom. The horizontal digraph is similarly defined, with the source representing the left boundary of the chip and the sink the right boundary. The edge direction is then from left to right. It is clear that both digraphs are acyclic and dual to each other. Furthermore, for a slicing floorplan, the digraphs are series-parallel graphs.

The floorplanning problem, as discussed in many published works, assumes a floorplan structure as represented by, for example, the polar digraphs. In other words, the relative placement is given. The objective is to find the minimum chip area subject to the constraint that each macro is specified by its area and perhaps a given shape function.

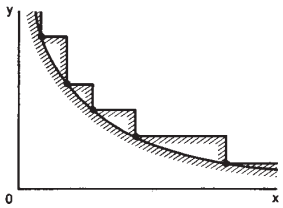


Fig. 7. Shape functions with continuous aspect ratio (parabola) and discrete aspect ratio (staircase).

For a given macro, if the area is fixed, its x–y dimensions can be plotted as a point on a parabola, as shown in Fig. 7. The function $f(\cdot)$, where $y = f(x)$ is called the shape function. If, in addition, the aspect ratio is specified by discrete numbers, as in standard-cell design, the parabola is replaced by a staircase shape function as shown in the figure superimposed on the parabola. Obviously, the shape function gives the lower bound of a final floorplan design. If in the final design every rectangular block has an aspect ratio which sits on the parabola of the shape function, the design is then optimum. To find the optimum solution is, in general, difficult; Stockmeyer showed that the problem is NP complete [25]. In the polar graph representation, the edges can be assigned a weight which is equal to the area of the macro. In their recent work, Wimer, Koren, and Cederbaum [26] used polar graphs to prove that if the shape function of the macros is continuous, there exists a unique optimum floorplan, and they give its necessary and sufficient conditions. This implies that there is always a minimum area floorplan, and thus, ideally speaking, it is possible to have no wasted area in floorplanning.

For the discrete aspect ratio situation, i.e., macros with a staircase-shape function, a branch and bound method has been proposed to find the optimum solution [27]. Earlier, Ohtsuki, Suziyama, and Kawanishi used a mathematical programming formulation based on the electric network analogy to solve the same problem [24]. A similar approach was used by Ziebert and Saal [28]. In the electric network analogy, each macro is represented by a linear resistor. The height and width of the macro are then the branch voltage and current, respectively. Thus the aspect ratio gives the resistance value, and the area represents the power dissipation. The whole chip corresponds to a resistive network whose power is to be minimized with respect to the resistances in the network.

For the special case of slicing structure, polynomial time algorithms have been proposed by Stockmeyer [25] and Otten [29]. In the slicing structure, it is easy to combine the shape functions of the macros by traversing the slicing tree, similar to the series and parallel combinations of resistors. It is simple to show that by traversing through the slicing tree from the bottom up, we can obtain the combined shape function for the whole chip. Then, using the information on the desired aspect ratio of the chip, we can then traverse the slicing tree from the top down to determine the aspect ratio for each macro. This is essentially the method used in the Mason floorplanner [21]. The Mason floorplanner also contains the important feature of routing area estimation, done by performing a global routing after preliminary floorplanning. The tree traversing algorithm is then repeated to divide the routing areas into blocks.

Up to now the discussion of floorplanning has assumed that the structure of the floorplan is known and that the only purpose is to minimize the chip area by determining the aspect ratio of the macros. One way to obtain a slicing structure placement is to use the bipartition process proposed by Lauther. Other methods can lead to a general structure. In the following we will briefly review the method of rectangular dualization which can incorporate the adjacent relations of functional and logic blocks into floorplanning. The objective is to find a relative placement of the modules which is compatible with the overall topological relations of the functional blocks to be realized in a chip. For example, module interconnection by abutting is attractive because no routing space is required. Thus in floorplanning it is desirable to keep track of the adjacency relations of the functional and logical blocks. This could also influence the module designers to assign pin locations to facilitate abutting.

A rectangular floorplan as represented by its dissection D is shown in the examples Fig. 5(a) and Fig. 5(b). It can be characterized in terms of a graph. For simplicity, we assume that the line intersections, with the exception of the four corners, are T-shaped. A "+" type intersection can be decomposed into two T-shaped intersections. A rectangular dissection D can be represented by a planar graph, called the floorplan graph: $G = (V, E)$, where V is a set of vertices representing the line intersections of D, and there exists an edge (v_i, v_j) in E if and only if the intersections corresponding to v_i and to v_j are adjacent. The inner faces of G are called rooms. To represent the adjacency relations between rooms of G, we construct the inner dual graph of G: $G^* = (V^*, E^*)$, where V^* is a set of vertices representing rooms of G, and there exists an edge (v_i^*, v_j^*) in E^* if and only if the rooms corresponding to v_i^* and to v_j^* have a common wall. The floorplan graph of Fig. 5(b) and its inner dual are shown in Fig. 8. The inner dual graph of a floorplan graph

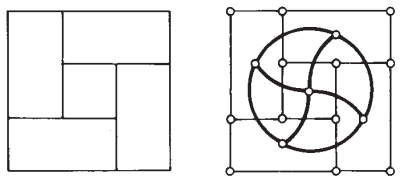


Fig. 8. Floorplan graph. (a) Inner dual graph. (b) Nonslicing structure.

is a plane triangular graph, i.e., a graph which can be embedded in a plane and every face of which is a triangle. A plane triangular graph has a rectangular dual if it is the inner dual graph of a floorplan graph representing a rectangular dissection. The application of dual graphs to IC layout was first introduced by Suziyama, Nemoto, Kani, Ohtsuki, and Watanabe in 1970 [30]. The use of rectangular dualization originated in architectural design in 1971 [31]. Its application to IC floorplanning was first developed by Heller in 1979 [32]. Since then a number of publications have appeared [33], [34], mainly due to its graph theoretical interest. To apply the method to floorplanning, several steps are necessary. First, the block diagram which depicts the global

functional and logic interconnections must be converted to a planar graph. Next, the planar graph must be reduced to a plane triangular graph. To guarantee that it has a dual floorplan graph, the plane triangular graph must not contain complex triangles [35]. In the aforementioned processes, edge deletion and node insertion may be necessary; furthermore, for a given proper plane triangulated graph there exist many rectangular duals. Thus for a large network, the complexity is too high to be appealing. Nonetheless, the method can be used as a guide in floorplanning for simple designs.

In the following we describe briefly the floorplanner used in the BEAR layout system [36]–[38]. The same method is also used for placement with only minor modifications. The method has several advantages and has proven to yield good building-block layouts on several real examples. It is general in that it includes the nonslicing structure. The method depends on bottom-up hierarchical clustering to obtain a clustering tree. A top-down planning is next performed. The method includes hierarchical wiring area estimation and other look-ahead features.

The placement and floorplanning method used in BEAR can be viewed as a generalization of Lauther's mincut bipartitioning algorithm; however, the number of parts resulting per partition is not limited to two. With a larger degree of freedom to group elements together, an artificial separation of related macros is less likely. In mincut bipartitioning a binary tree is generated by a top-down process, whereas BEAR uses a data-driven bottom-up clustering and thus preserves a natural cluster of macros as much as possible. This is not the case in mincut bipartitioning because a binary tree determines the eventual placement of some macros to a great extent at the very outset. For the final placement it is also crucial that at the very end of bipartitioning the shape of the rooms should match the shape of the macros as much as possible. Since this is very difficult to achieve, an iterative step [39] or restructuring of the tree is necessary [40].

BEAR employs a clustering algorithm based on graph partition using a combined criterion of connectivity and geometry. The latter is heuristic in that blocks in pairs are forbidden if their areas or if the lengths of their longer sides differ significantly. We allow the maximum size of a cluster to equal five at each step. The step is recursively repeated and thus produces the different hierarchical levels in a bottom-up fashion of the cluster tree. We limit the size of each cluster to five because: i) with five or less blocks, a simple pattern enumeration followed by an exhaustive search is feasible, and ii) a general nonslicing structure floorplan is included in our treatment.

Next the method proceeds top-down from the root of the tree, which represents the whole chip, with the chip's specified aspect ratio and possibly I/O goals. The known information at this stage is represented by the clusters of blocks at the next level, which includes area and possibly shape information passed on from the leaves towards the root of the tree. The floorplanner then searches a library of floorplan templates and considers all possible room assignments which meet the combined goals of aspect ratio and I/O pads. The library of templates for two to five rooms is illustrated in Fig. 9. Note that only in the five-room situation does a nonslicing structure exist. The cost function is again given by a combined criterion of connectivity and geometry. Fig. 10 illustrates a cluster tree with three possible floor plans whose chip aspect ratios equal 1 : 1, 2 : 1, and 4 : 1. The algorithms used at the leaf level and the nonleaf level are slightly different. The details are given in [38].

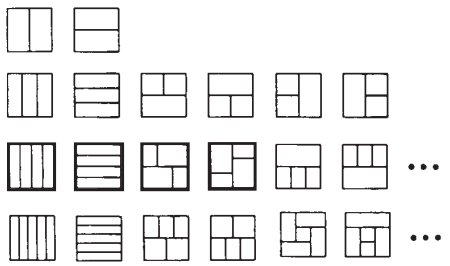


Fig. 9. Floorplan templates for two-room, three-room, four-room, and five-room configurations.

An important feature included is wiring area estimation. A simple top-down hierarchical method is used. Fig. 11 shows a three-block cluster with elements (0, 1, and 2) and I/O's (N, E, S, and W), where the shaded areas are called bottlenecks [41]. Each bottleneck has an estimated channel width which is calculated by the sum of possible nets going through the channel according to a probability measure. Consider the bottleneck between block 1 and block 2. The channel width is given by

$$w_{12} = t_{12} \sum_{i,j} p_{ij}^{12} C_{ij}$$

where C_{ij} is the connectivity between element i and element j, p_{ij}^{12} is the probability that the i–j connection goes through the bottleneck 1–2, and t_{12} is a heuristic weight factor which takes into account track sharing. The probability factors p_{1N}^{02} *and* p_{1N}^{2E}, for example, are each equal to 0.5.

The preceding estimation takes advantage of all the information gathered about the position and connectivity of clusters of blocks up to that level. Earlier in the top-down process space for global connections between different clusters has been provided. Later on, in the successive levels, internal connections within the clusters become visible. Because the path information is precomputed for the finite number of topologies, the wiring area estimation can be done efficiently and simultaneously with the template evaluation process. Routing area can be treated in the same way as the block area in the objective function, to obtain an optimal floorplan/placement with routing.

There is an additional feature of floorplanning in BEAR which is used after global routing. It is called the global shape optimizer, and is similar in philosophy to a chip-level compactor. The optimizer assumes that module orientations and the relative pin locations are fixed. Given the initial placement and a set of constraints on module shapes for each module, the shape optimizer alters the location of modules and the aspect ratio of the soft modules to further reduce the chip area. The algorithm performs X- and Y-axis optimization in one pass. It iteratively reduces the longest path by picking up a module lying on a critical path (in either direction) and resizing it so that the module dimension along the critical path is reduced. The process terminates when no significant improvement in the layout area can be achieved.

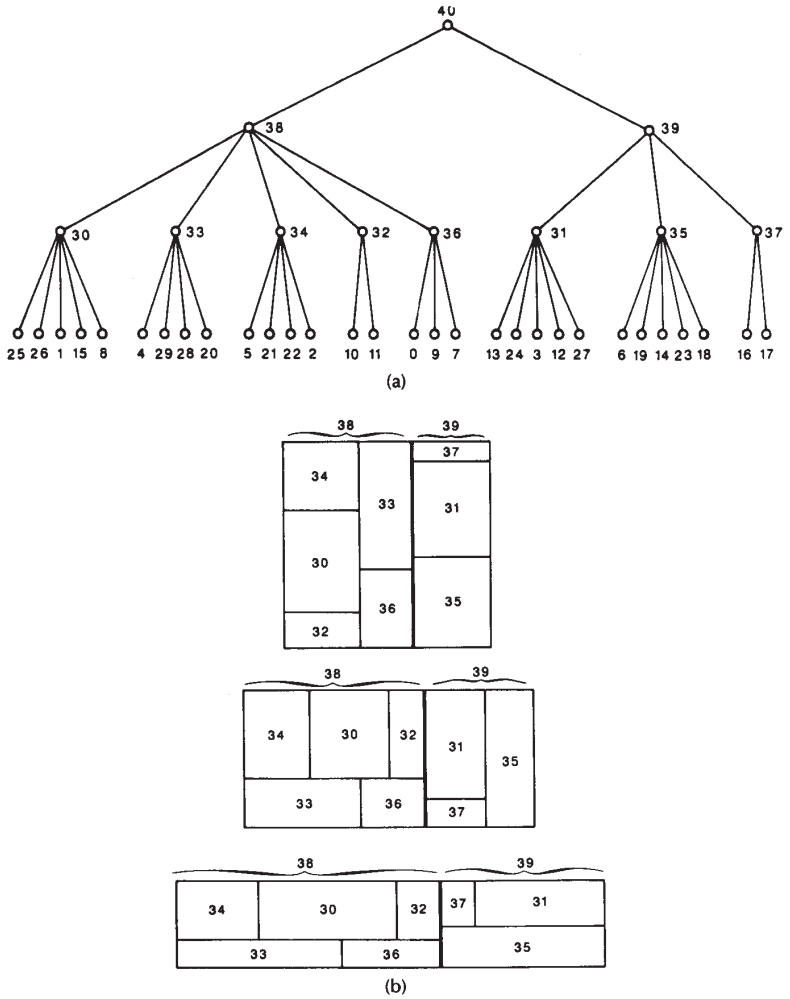


Fig. 10. Example of hierarchical floorplan enumeration. (a) Cluster tree. (b) Floorplanning with overall aspect ratio goals 1:1, 2:1, and 4:1.

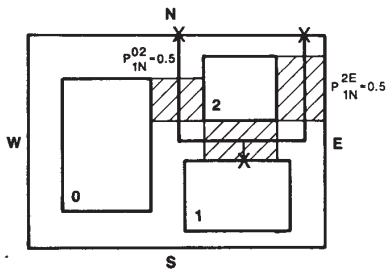


Fig. 11. Hierarchical routing area allocation.

III. Routing

After modules are placed, the remaining task in layout design is to make all the necessary interconnections. This means that pins of modules must be tied together by nets according to the net list specification, meanwhile satisfying both the physical constraints and design rules. The problem is usually referred to as *routing*. In the sea-of-gates design it entails the connection of many thousands of nets competing for the space throughout the chip that is left over from cell design. The principal requirement is to make 100-percent interconnection. In building-block layout, routing regions are defined after placement [15], though the areas of those regions are not yet fixed. The objective is to devise a method of routing which, when it is completed, minimizes the total chip area. Since the problem is complex, routing is done in two stages: global routing and detailed routing.

Global routing (sometimes called loose routing) calls for a routing plan in which each net is assigned to a particular routing region. The aim is not only to make 100-percent assignments but also to minimize the total wire length. Between neighboring regions, pseudo-pins are used to indicate the intersections of global paths with the boundaries of the local regions. These pins are sometimes called "floating," to imply that their precise coordinates on the boundary may not be fixed. Detailed routing takes over after

global routing and gives the precise recipe of net interconnections for each region in terms of net, via, and track assignments. Although detailed routing is sometimes done using only one layer, as in river routing [42], in most current practice two layers are used. As technology advances, routing on three or more layers will become important. Even now, multilayer interconnect is of real interest in packaging design.

The simplest approach to both global routing and detailed routing is *sequential routing*, i.e., to route one net at a time and to choose the shortest path whenever possible. The Lee–Moore grid expansion algorithm [43] and many of its modified versions have been widely used. The problem of finding the shortest connection for an n-terminal net is called the Steiner problem. It is related to the shortest spanning tree problem but it allows, in addition to the n given terminals, intermediate connecting points called the Steiner points. Unfortunately, the Steiner problem is NP complete. There exist simple heuristic algorithms, however, and for $n \leq 5$ the rectilinear Steiner tree is easily found [44]. Also, for global routing it is useful to consider the Steiner problem on a weighted graph [45]. For special graphs, such as the $2 \times N$ grid graphs and the partial two-trees [46] and plane-triangular partial three-trees [47], there are linear time algorithms. These graphs have been found useful in hierarchical routing and in combining floorplanning with global routing [36], [48].

The objective of routing is to make 100-percent connection that minimizes the chip area. Clearly it is not generally possible to route all the nets with shortest connection paths because they compete with each other for paths. The sequential approach updates the available routing space after completing each net, as in the Lee–Moore router. The routing order becomes crucial. There are arguments for routing the shortest nets first; there are also arguments for routing nets with the most terminals first. It is almost impossible to devise a general procedure which works the best in all situations. The other approach is to route all nets independently, and if some routing space becomes overcrowded, reroute the nets which cause congestion. This sometimes can be accomplished by rip-up and rerouting, which will be discussed in section III-C. Often the information obtained from independent net routing can be used to help in implementing a totally different routing strategy based on the global view it presents. We will discuss global routing in section III-A and detailed routing in section III-B.

A. Global Routing

Although the aim of global routing is to minimize the total wire length, the approaches used differ somewhat according to the layout design styles [49]. For gate-array design there are methods based on deterministic optimization, simulated annealing, hierarchical top-down and bottom-up schemes, and strictly constructive methods [49]–[52]. In standard-cell design, feedthroughs must be used to interconnect cell rows. Global routing can be treated as a multichannel optimization problem using spanning-tree or Steiner-tree algorithms to reduce the total number of routing tracks [53]. For building-block layout, a graph is usually used to represent the topology of the placement, and sequential routing is done on the graph. The edge weight of the graph, which depicts routing congestion, is updated after each net is routed [41].

In the following we will briefly discuss a method which is based on successive top-down cuts on a plane. The method was discovered independently by Marek-Sadowska for building-block layout [54] and by Lauther for sea-of-gates design [55]. Its special feature is that it is order-independent, and at each hierarchy, it attempts to simultaneously assign optimum net crossings at a cut-line. We have implemented the method in the BEAR program for building-block layout and in the Proud program for sea-of-gates design. We believe that the results are superior to any other available methods in terms of quality and computation time.

The essence of the method will be illustrated first with the sea-of-gates problem. We will assume that the routing capacity is uniform across the chip. The whole chip is divided into rectangular regions; at the boundaries of each region the capacity for net crossing is uniform. The end result of global routing is a specification for each region of the net list for detailed routing in terms of pseudo-pins and cell terminals. Clearly, with no overflow the number of net crossings must not be larger than the capacity at each boundary.

To illustrate, a horizontal cut is first made. There are altogether M sections evenly spaced on the cut-line, as shown in Fig. 12(a). We assume that there are N two-terminal nets

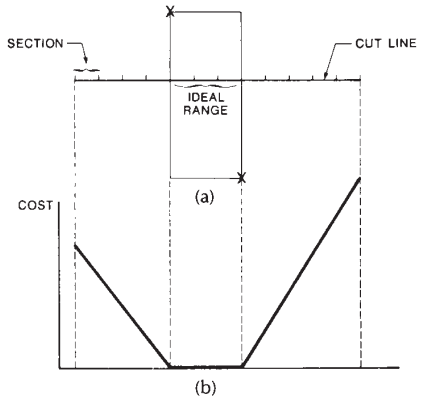


Fig. 12. (a) Illustrating cut-line, sections, enclosing rectangle of two-terminal net, and ideal range. (b) Cost function defined for net.

which must cross the cut-line from the top to the bottom. For simplicity, let us also assume that the capacity for each section is one and $N \leq M$. It is thus possible to assign one net to a specific section in such a way that some measure of wire length is minimized. Specifically, we want to consider the incremental length, i.e., the length exceeding the shortest route for each net. A cost function must be introduced for each net with respect to the sections of the cut-line. Let us draw the bounding box of each net. Clearly, if the sections are within the bounding box of a net, the cost is zero. We say that these sections are in the ideal range. For sections outside the bounding box, the cost is proportional to the horizontal distance away from the ideal range, as shown in Fig. 12(b) by finite-slope straight lines. The slopes in the cost function may be defined according

to the vertical distance from the terminal to the cut-line. Nets compete for sections in their ideal ranges; some nets must take detours and thus there is a higher cost. For terminals very near the cut-line, the net should not be routed with a long detour away from the ideal range. Other criteria can also be used; for example, in routing critical nets. With the cost functions specified, the problem can be solved by standard linear assignment algorithms. As mentioned earlier, net assignment is done all at once without net ordering. After the initial cut, we proceed to the next hierarchy in the upper plane and lower plane. The process is repeated for each half-plane with vertical cuts. It is terminated when we reach to lowest level of hierarchy, i.e., when we attain the regions for detailed routing.

In the preceding we have assumed that all nets are two-terminal nets and each net is to cross the cut-line exactly once. For multiterminal nets, some preprocessing must be done to decompose the net into subnets. The best way is to find the minimum Steiner tree or an approximate minimum Steiner tree for each net. A subnet denotes a subtree which crosses the cut-line only once.

The location of the cut-line affects the overall result of global routing. What seems to count is net congestion. We have found that the best result can be obtained if the cut is made in the most congested area. Thus we propose to first obtain a density profile after placement. This entails finding all the Steiner trees of nets independently. The order of cuts is then determined according to the density profile with the most congested area cut first. Cuts need not be done at the center in each hierarchy; neither is an alternative horizontal/vertical cut required.

The preceding discussion must be elaborated to fit the real design situations in both sea-of-gates and building-block layout. The sections on each cut-line have routing capacity specifications. Let us use building-block layout to illustrate first. When a cut-line is made it intersects both the macrocells and the routing regions. The sections are defined accordingly. If we assume that no over-the-cell routing is allowed, then the sections defined by macrocells will have zero porosity, i.e., the capacity is zero. The capacity of sections defined by the routing regions depends on the number of tracks predetermined by placement with routing area estimation. If over-the-cell routing is allowed on one layer in some locations, the routing capacity for those sections must be calculated. In sea-of-gates design, the capacity specification must include information on whether two or three layers are used for routing and on where layer commitments have already been made in cell design. Once the capacity is specified and cost functions are given, the global routing problem can be described as follows. For each cut, we have sections $j = 1, 2, \cdots M$, each with a capacity C_j. There are nets/subnets crossing the cut-line, $i = 1, 2, \cdots N$. The cost is defined by an $N \times M$ matrix $W = [w_{ij}]$, where w_{ij} is the cost of ith net assigned to the jth section. Let x_{ij} be a 0–1 variable, then we are seeking an assignment of nets to sections to minimize the total cost function F:

$$F = \sum_{i=1}^{N} \sum_{j=1}^{M} w_{ij}\, x_{ij}$$

subject to

$$\sum_{i=1}^{N} x_{i,j} \leq C_j$$

and

$$\sum_{j=1}^{M} x_{i,j} = 1.$$

That the variable x_{ij} is zero implies that net i is not assigned to section j. It is equal to one if net i is assigned to section j. This is a familiar network flow problem in which the integer min-cut maximal flow solution can be obtained by standard methods [56]. The time complexity is, on the average, of $O(M \times N)$. For a building-block layout example with 77 macros and 500 nets, it takes 20 seconds CPU time on a VAX 11/780 to find a solution.

B. Detailed Routing

In detailed routing a net list is given together with a specified routing region and available routing layers. For simplicity we assume that pins are accessible in any layer unless they are precisely specified. The primary requirement is to make 100-percent interconnection from the given net list using the space available. There are often additional objectives such as minimizing the total wire length or the total routing area if the routing region has flexible boundaries. Others may be imposed: minimizing the number of vias, considering prerouted nets such as power, ground, and bus lines, or treating critical nets specially, for instance.

The problem of detailed routing can be classified in terms of the number of available routing layers. For one-layer routing, the key constraint is planarity. There are well-established problems like river routing and single-row routing [57], [58]. For two-layer routing, a key strategy often employed, called H/V routing, is to route one layer with only horizontal wires and the other with only vertical wires. The connections between the two layers are made at vias. There exist numerous well-known algorithms for H/V routing. What is perhaps not so well-known is the concept of *collapsed routing* [59], which is a planar representation of two-layer routing. If knock-knee connections are excluded, the collapsed routing can easily be transformed to an H/V routing by inserting a via at every intersection. Horizontal connections can be routed all on one layer and vertical connections on the other. Well-known via-minimization algorithms can reduce the number of vias [60]–[64]. The result will have horizontal and vertical wires mixed on the two layers. The concept of collapsed routing is appealing because it initially disregards layers. We believe that this approach should be researched further to develop new routing algorithms.

The problem of via minimization has received considerable attention. It has both theoretical interest and practical significance. The problem can be classified into two: the constrained via minimization and the unconstrained, also called topological, via minimization. In the former, routing has already been done either on two layers or in the collapsed form. Via minimization then follows as a post-processor. Algorithms based on graph representation and clustering have been reported which show that via reduction of 20 percent or more from an initial routing can often be achieved. The unconstrained via minimization problem was first introduced by Hsu in 1983 [65]. The aim is to minimize the number of vias based on the net list information without considering either the geometry of the region or design rules. It is an NP complete problem [66], but there

are heuristic algorithms which find a topological routing [67], [68]. The conversion of topological routing to geometric routing, however, is difficult. The problem lies in mapping topological routes onto routing tracks to fit the space and boundaries of the given routing region. Much more research is needed before topological routing becomes useful in real layout design.

Detailed routing can also be classified according to whether the routing region is a channel or a general two-dimensional region. An ordinary channel is a rectangle with open ends on two opposite sides. There are pins on the other two opposite boundaries of the rectangle. Nets leave the channel through the open ends and are connected to other parts of the chip. In building-block layout, when four channels meet at a "+" (cross) intersection, the region of the intersection defines a rectangular routing region called a *switch box*. As far as the switch box is concerned, pins and pseudo-terminals exist on all four sides of the rectangle. The major difference between a channel and a switch box is that a channel is an open region and its width can be expanded or reduced according to the space required by a channel router. A switch box, on the other hand, is a closed region whose routing space is fixed. In channel routing, 100-percent completion is guaranteed and the main aim is usually to minimize the channel width. In switch box routing, though, the main aim is to make 100-percent connection. If, for a given net list, routing cannot be completed, the problem is said to be not realizable.

There are generalized versions which include rectilinear boundaries of both the channel routing problem and the switch-box routing problem. The channel can be L-shaped or Z-shaped [69]; the switch box becomes a two-dimensional closed region with a rectilinear boundary which may contain obstructions or rectilinear blocks. These generalizations are important in the detailed routing for both sea-of-gates design and building-block layout. In the following we will give a brief review of switch-box routing and its generalizations. We assume that vertical and horizontal routing grids are given in a closed two-dimensional region and design rules are automatically satisfied. The problem is to make 100-percent interconnection of the pins according to the specified net list.

Of course sequential routing in the form of either maze running or line search [70], [71] may be used. This should be combined, however, with other strategies such as a breadth-first search and minimum change in routing directions [72].

The hierarchical method developed by Burstein [73] can be used both for ordinary channels and switch boxes. It employs a divide-and-conquer technique and leads to excellent results for channel routing, but fails to complete the "Burstein difficult switchbox." Since the publication of the particular example in 1983, several routers have been developed which succeed in routing Burstein's difficult example.

The method introduced by Marek-Sadowska [59] is rule-based. It uses the collapsed routing strategy and conducts routing from the boundary of the region inwards. One key idea is to detect patterns of partial routing which call for a unique solution. A more elaborate rule-based router is Weaver [74], which is a knowledge-based expert systems router. Weaver produces excellent routing results, but takes an enormous amount of time even for simple problems.

The Mighty router [75] is based on an incremental routing strategy. It employs maze running but has the special feature of modifying the already-routed nets. The cost function penalizes long paths and those which require vias. Rip-up and rerouting may be necessary to complete the routing.

Another recently developed heuristic router, Beaver [76], has yielded excellent results. The algorithm consists of three successive parts: corner routing, line-sweep routing, and maze routing. It uses priority queues to determine net ordering. In addition, an important feature of track control is employed to prevent routing conflicts. At the beginning, a net is given control of the tracks that extend out part of the way from its terminals. A portion of the tracks is reserved for other nets, so controlled sections may overlap and routing is done on a first-come-first-served basis. This method tends to give a more global view and serves as a guide to the heuristic routing process.

An earlier routing approach proposed for printed circuit board design, called a pattern router [77], [78], is useful in two-dimensional routing. The idea is to enumerate all possible patterns according to cost functions to determine the best path for a two-terminal net. This and various other strategies have been combined recently in the development of a template-based router [79], [80]. As shown in Fig. 13, only four basic types of template are needed. Within each type there are four different configurations depending on the orientation of the templates. For the three-segment template, the span of the central segment depends on the spacing of a given net. The end sections can be characterized in terms of a single number d, which depicts the relative dimension of the sections. If $d = 0$, the three-segment template is reduced to an L-shaped pattern.

Routing is done sequentially, with the longest half-perimeter net routed first in the present implementation. A control structure similar to that in Beaver is used. Terminals control a specified percentage of the length of the tracks they face. For terminals on the edge of a routing region, the distance is set to a fraction f of the available length of the track perpendicular to the boundary where the terminal lies. Whenever a net is completed, it releases control of all the grid points which it controls. Terminals are also allowed in the interior of the routing region. For these terminals, the reserved track extends in a preferred routing direction on both sides of the terminal, with a distance equal to $f/2$ of the distance to the boundary. It is easy to see that due to the control structure a deadlock situation can occur where one net does not permit another net to be routed. A special feature in the algorithm is the dynamic updating of the control structure with automatic decrease of the control region. Having uncompleted nets after one pass through the routing will lead to an automatic decrease of the extent of control for the remaining unconnected nets by a fixed amount until the control reaches zero toward the end of the routing process, when maximum congestion starts occurring.

To select a template to be used for routing a net, a set of costs is defined. In the formulation, we assume that obstruction may exist within the routing region. The following defines the various costs.

- C_b Cost of routing on a grid that is blocked.
- C_c Cost of routing on a grid point controlled by a net other than itself.
- C_v Cost of a via.

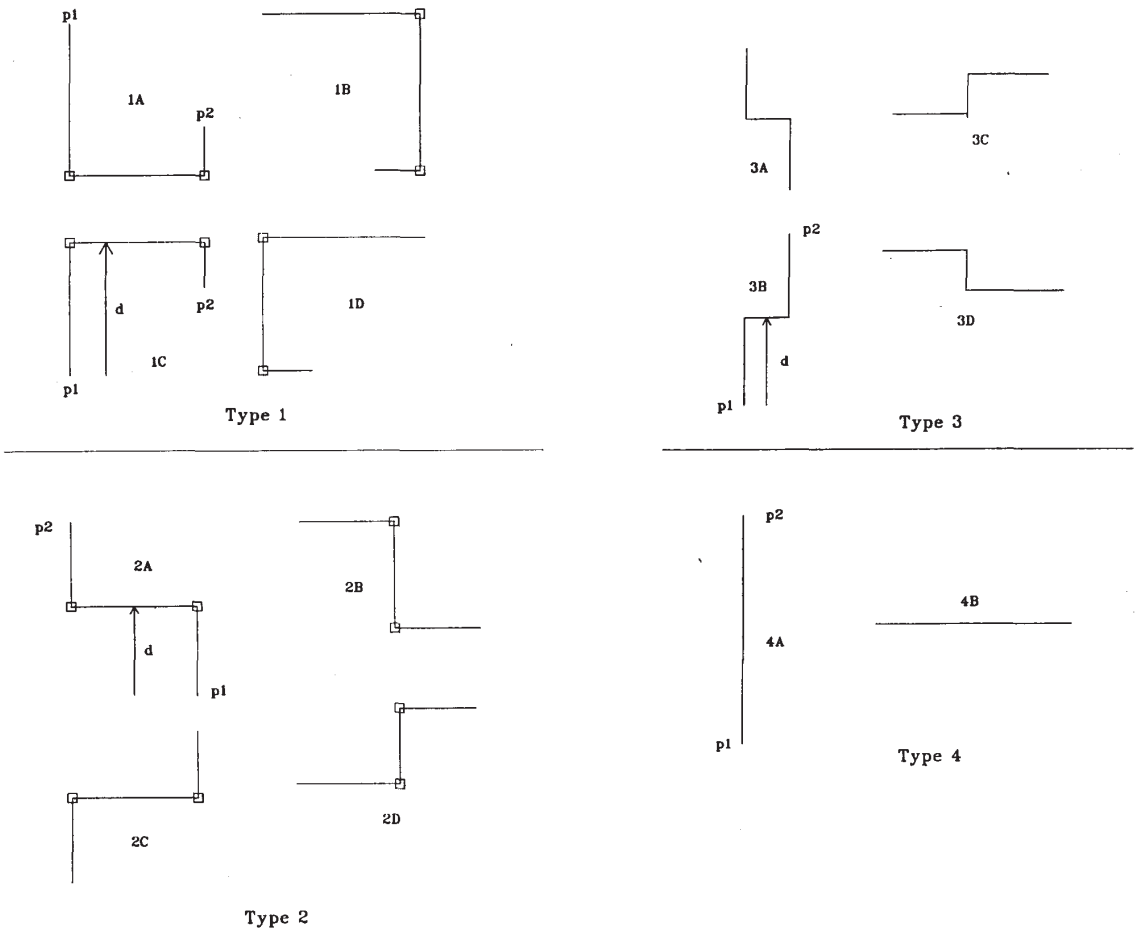


Fig. 13. Routing templates for template-based two-dimensional router.

C_f Cost of routing over an uncontrolled grid point.
C_s Cost of routing on a grid point controlled by itself.

Then C_b is set to be infinity, and $C_c \gg C_v > C_f > C_s$. The costs of various templates are evaluated for each net to be routed between two terminals. The template with the minimum cost is selected if its cost is less than C_c. The connection between the two points is uniquely identified by the template chosen and the characteristic dimension number d. These parameters are stored in the memory, so that if, at a later stage in the routing process, a certain connection needs to be ripped, only the template number and its characteristic dimension needs to be retrieved.

If there are no templates that have a cost less than C_c, then either blockage has been encountered or a template passes over a grid that is being controlled by another net. A special strategy is needed, depending on the nature of the conflict, to find a successful path [80].

Multiterminal nets must first be decomposed into two-terminal subnets. Because the router allows terminals inside the routing region, a minimum Steiner tree can be used to define the subnets. In the present implementation, the minimum spanning tree is used. In summary, the template-based router has several salient features. It is also very flexible in that if rip-up and rerouting is needed, it can easily be done because of the simplicity of the template's representation. Preliminary tests indicate that the router yields high-quality results in that most nets are routed with shortest paths. It is also extremely fast.

For the Burstein more-difficult switchbox example, the initial template-based routing completes 23 of the 24 nets, all with minimum wire length. The last net is completed by maze routing. The total wire length is 542 units and the number of vias used is 35, which is comparable to that of Mighty and Beaver [76]. The router has an additional special feature: it allows terminals to be prespecified on a particular layer.

The problem of general two-dimensional routing is clearly more difficult than channel routing. There are no theoretical results, such as those in channel routing, obtained from a graph theoretical approach. However, in special situations, there are some intersting results. Nishizeki, Saito, and Suzuki [81] considered routing in a two-dimensional region with convex grids that include knock-knee intersections on one or two layers. They proposed an exact solution with a linear time algorithm if all nets are two-terminal and pins are restricted to the boundary. The agorithm is based on planar multicommodity flows and finds edge-disjoint paths

when they exist. Onaga [82] researched the switchbox problem and found a method for multiterminal nets which obtained similar results.

C. Rip-Up and Rerouting

The primary goal of detailed routing is to achieve 100-percent interconnection using the space available. The conventional incremental routing algorithms suffer from a fatal disadvantage in that they provide no feedback or anticipation to avoid conflict between nets. Because of this blindness such autorouters often fail to connect all the nets and must be followed by manual rework to connect the remaining nets (usually less than 5 percent). What is typically done by designers for connecting the remaining few nets is to detect existing connections causing "blockages" and to rip up and reroute them so as to provide room for the blocked nets. The basic idea behind the recent rip-up and rerouting techiques is to simulate the human way of doing this. The goal of routing tools having the rip-up and rerouting ability is to achieve 100-percent interconnection as successfully as a human expert with shorter design turn-around time, by either improving existing autorouters or using interactive tools.

Recently, several CAD vendors have provided routers with rip-up and rerouting features, which achieves better performance. Until now, a few published papers [75], [83]-[85], [181] have reported rip-up and rerouting schemes, but they only ambiguously describe key strategies on deciding which existing nets should be ripped up and how their routes should be modified in order to complete the interconnection under consideration. Roughly speaking, these are the commonly used strategies: pay special attention to the neighborhood of the pins for subsequent interconnections [83]; push aside nearby existing nets so as to make room for a blocked net [75], [84]; remove some existing interconnections so as to connect the blocked net earlier [75], [83]; etc. The template-based router discussed in the last subsection uses a combination of the aforementioned schemes for rip-up and rerouting. In the following we will briefly discuss a graph-theoretic approach to rip-up and rerouting [86], [87] and its implementation on an interactive routing system, called WIRES [88], developed at Waseda University.

When a particular interconnection fails at an intermediate stage of incremental routing, it implies that some already-routed nets have blocked the pin pair. Such a blockage can be well characterized by the graph-theoretic term "*minimal separator.*" The example shown in Fig. 14 will help an understanding of this term, where nets 1, 2, --- , 5 have been routed and net 6 has been blocked. The essential minimal separators for our purpose are those consisting only of the cells occupied by existing interconnections. Among them, two particular ones, S_A (reachable from a pin of the blocked net 6) and S_B (reachable from the other pin) can be found by means of exhaustive wave propagation, i.e., an obvious modification of the Lee algorithm. Those cells in S_A and S_B are marked with O and X, respectively.

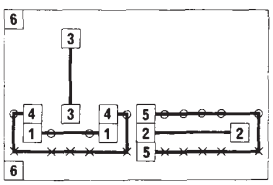


Fig. 14. Minimal separators blocking net 6.

In order to indicate the likelihood of rerouting particular nets to succeed, the following measure [86], [87] is derived from these two minimal separators. For each already-routed net W, let $N_A(W)$ and $N_B(W)$ be the number of cells of W contained in S_A and S_B, respectively. Then

$$P(W) = \min \{N_A(W) - L_A(W), N_B(W) - L_B(W)\}$$

is called the "reroute effect index" for W, where $L_A(W) = 1$, if the pin pair of W lies in the opposite sides of S_A, otherwise it is equal to zero. $L_B(W)$ is similarly defined. It should be stressed here that rerouting of W successfully makes room for the blocked net only if

$$P(W) \geqq 1.$$

In this particular example, we know that only net 4, with $P(W) = 2$, and net 5, with $P(W) = 5$, are worth trying for rip-up and rerouting. Although rerouting of net 4 alone is actually successful in this example, the reroute effect index is useful to narrow the range of trial and error. It is desirable to improve this index because a larger value means it is more likely to succeed. An approach in this direction is reported in [88].

Very often we encounter a case where a simple rip-up and rerouting is of no help. Fig. 15(a) is such an example; net

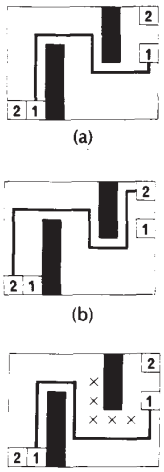


Fig. 15. Example in which X router is effective.

2 has been blocked by the already-routed net 1. If we connect net 2 by means of a shortest path after removing net 1, as shown in Fig. 15(b), then net 1 is blocked as well. Thus the partial reordering does not work. In such a case, the "X router" implemented in the WIRES system [88] is of great help to find an optimal rerouting path. The cells marked with X in Fig. 15(c), called "X cells," indicate the essential part of the existing interconnection which blocks the unconnected net. Then the X router tries to find a path for the removed net 1 without using X cells so as to make room for the blocked net 2. Note that those X cells can be found

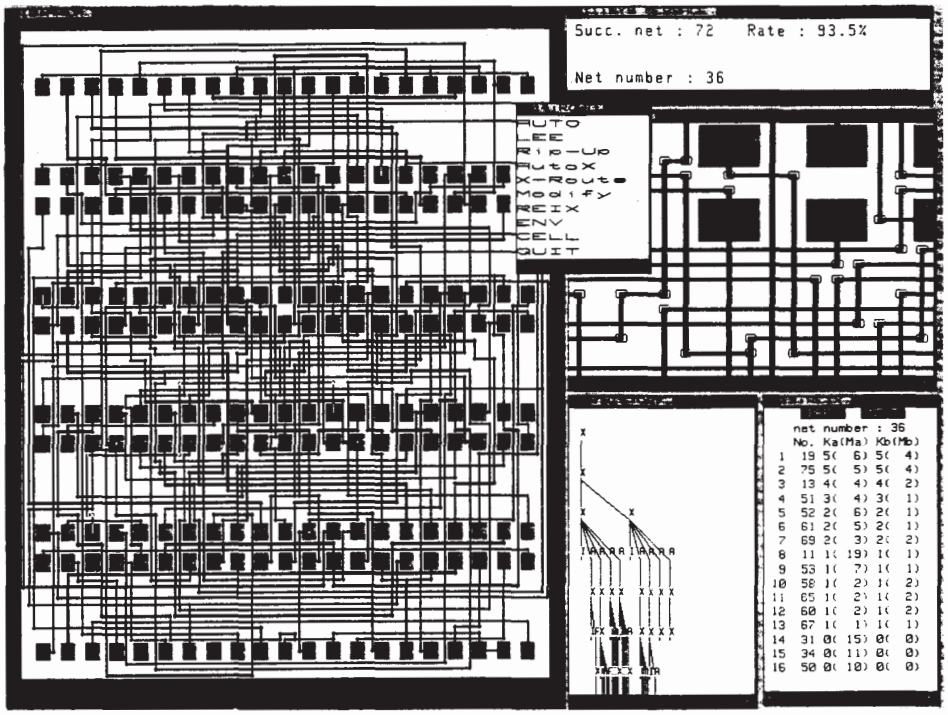


Fig. 16. Multiwindow display in WIRES system.

by means of a slight modification of the wave propagation search for obtaining the minimal separators [87], [88].

The sophisticated rip-up and rerouting is more suitable to implementation in a standalone interactive system than in an autorouter; the former has more flexibility to exploit expert designers' control. On the other hand, the interactive system must provide a pertinent set of commands that are executed very fast so as not to make the interactive operations tedious. The WIRES system has a backup hardware router to accelerate the execution of key commands, which will be briefly described in section V. The WIRES system, in addition to the ordinary incremental router, accommodates the commands for finding minimal separators, calculating reroute effect indices, temporarily removing an existing interconnection, generating X cells, invoking X router, etc. Fig. 16 shows an example of multiwindow display in WIRES. In this example the display consists of the whole wiring space, its partial zoom, command menu, reroute effect indices of blocking nets, and a managing tree representing a sequence of rip-up and rerouting trial.

Fig. 17 shows an example of two-layer wiring done by a graduate student using WIRES. There are 56 nets to be connected, but an incremental router left eight of them unconnected. The interconnection of the remaining nets was completed in 8 minutes (turn-around-time), where exhaustive wave propagation search, ordinary two-pin router, and X router were invoked 4, 14, and 6 times, respectively.

The rip-up and rerouting techniques are useful and promising for improvement of existing detailed routers. However, in order to solve large-scale problems in shorter time, research on faster backup accelerators and on rip-up and rerouting strategies based on gridless routing will be needed.

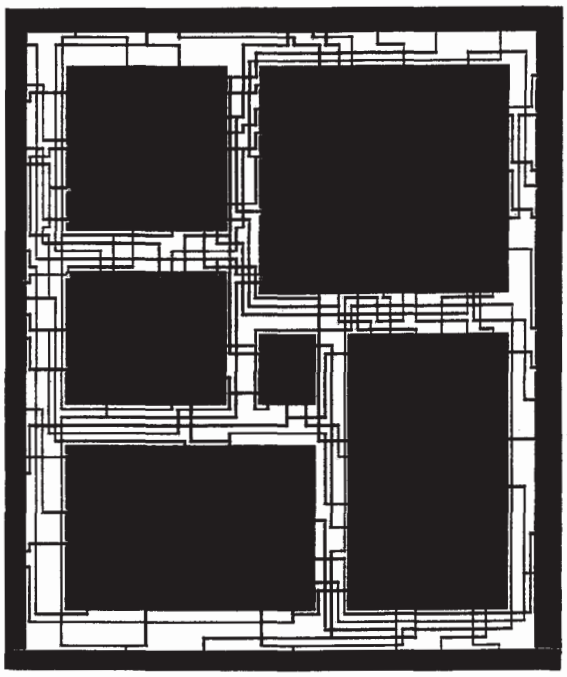


Fig. 17. Two-layer routing problem.

IV. Computational Geometry in VLSI Layout

Theoretical computational geometry has advanced, aiming at applications such as geographic information processing, computer graphics, structural databases, etc. [89].

In the past several years, applications of computational geometry to VLSI layout have also become popular [94]–[110], [114]–[118]. Some of these contributions are summarized in a review article [90]. In this paper we will discuss some more recent results on design rule checking, gridless routing, and layout compaction, after briefly reviewing basic algorithmic techniques and data structures relevant to VLSI layout.

The geometrical patterns encountered in VLSI design are usually simple, i.e., polygons in the mask planes, although we have to deal with extremely large-scale problems. Many of the relevant geometric search problems can be characterized as intersection problems in which layout objects such as rectangles, line segments, points, etc., lying on the digitized plane, are under consideration. These intersection problems are classified into two categories. One of them, called "batched mode," is to report all the intersecting pairs among a given set D of objects. The other, called "on-line mode," is to report items of D that intersect a particular query object. The on-line mode problems are further divided into two classes: static problems, in which the underlying set D is not changed, and dynamic problems, in which D is updated by intersections and deletions.

The most powerful and commonly used algorithmic technique in layout verification is the "worklist method," often called "plane sweep" by computer scientists. In this method, the IC mask plane is swept by moving a horizontal scan line from the top to the bottom. Only the layout objects currently intersecting the scan line are examined and partial results are reported. The objects lying above the scan line have finished their roles, and those below the scan line will play their roles afterwards. In batched-mode layout verification, the worklist method is used in such a way that only objects intersecting the current scan line are fetched from a sequential file, which accommodates the whole data, and therefore the main memory capacity can be saved to great extent. To be more precise, the method runs with $O(\sqrt{n})$ main memory space for a total of n layout objects in the plane, provided that they are uniformly distributed [183]. Note that a typical mask set of a VLSI chip includes polygonal patterns with a total of 10^6–10^8 edges. In theoretical computational geometry, the plane sweep is used to improve time complexity even if the whole data is within main memory capacity [89].

We shall examine the efficiency of the worklist method along with a rectilinear line-segment intersection problem, illustrated in Fig. 18. The vertical segments S_1 and S_2, which initially intersect the scan line, are stored in a work list. As the scan line goes down and hits the top end of S_3, it is inserted in the work list. When the scan line reaches the bottom end of S_2, it is deleted. When the scan line overlaps the horizontal segment S_4, (S_1, S_4) may be reported to be an intersecting pair. Continuing this way, everything desired can be reported once the whole plane has been swept out. A key point of the efficiency of this method is that any nonintersecting pairs need not be examined. The precise time complexity of the method depends on how the worklist data is managed.

The worklist method can be viewed as a technique of reducing a two-dimensional batched-mode problem into a sequence of one-dimensional, dynamic, on-line mode problems. The one-dimensional problem derived from the preceding example is as follows.

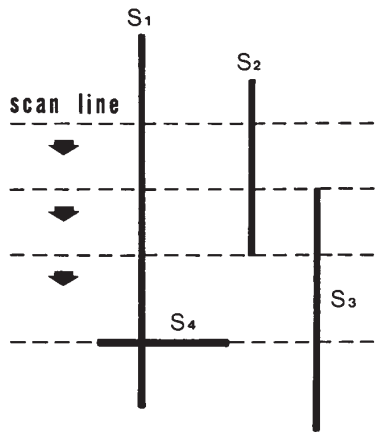


Fig. 18. Reporting intersecting pairs of line segment by means of plane sweep.

Problem A. For a set D of m numbers and a query interval I, enumerate all numbers of D contained in I.

A powerful and simple data structure for dealing with the preceding problem is a balanced binary search tree, e.g., an "AVL tree" [91]. By virtue of the balanced nature of the tree, the unit operations of inserting or deleting a number and accessing the smallest (largest) number above (below) the given key value can be done in $O(\log m)$ time. Note that in the process of reporting the line-segment intersections, the x coordinates of vertical line segments currently intersecting the scan line are kept in the AVL tree and x coordinates of the terminals of query interval I are used as keys to search all the numbers covered by I. From the aforetold arguments it is seen that the rectilinear line-segment intersection problem can be solved in $O(n \log n + k)$ time, where n is the total number of line segments and k is the number of enumerated intersecting pairs.

There is a wide variety of subproblems encountered in layout verification, but they can be represented by a combination of a number of basic geometrical operations among mask patterns, such as intersection of two polygonal patterns, shrinking or expanding polygons by given width, etc. [92]. Many IC suppliers have a long history of working in this area; they have been making an effort to develop batched-mode layout verifiers which run in almost linear time with $O(\sqrt{n})$ main memory space in practical cases. From the theoretical point of view, optimal algorithms with $O(n \log n)$ worst-case run time have been worked out for almost all subproblems encountered in layout verification, by means of the worklist method with pertinent data structures [90].

However, the minimum width/spacing check problem is an exceptional case where existing optimal algorithms are not acceptable in practice. To observe this point, Fig. 19 shows four possible cases to be considered for checking spacing errors in rectilinear polygonal patterns. By moving a horizontal scan line vertically, we cannot find spacing errors as in Fig. 19(b), but can find those shown in Fig. 19(a). Note that performing plane sweep in both horizontal and vertical directions still leads to an optimal algorithm, but it requires two sorted files (with respect to both x coordinates and y coordinates, separately). Spacing errors like

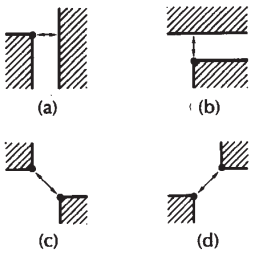


Fig. 19. Four possible cases causing minimal spacing error.

those of Fig. 19(c) and Fig. 19(d) are more difficult to check because Euclidean geometry is involved. Theoretically, those errors can easily be checked by means of a "Voronoi diagram" and an optimal algorithm to draw it in: $O(n \log n)$ time is available [93]. On the other hand, this algorithm requires that we keep the whole data in main memory.

Recently the "enhanced plane sweep" method [94] was presented to overcome these practical difficulties. A key consideration in the method is to prepare an additional worklist that stores the sequence of already-scanned vertices visible from the current scan line. This makes a single-plane sweep possible to detect the types of spacing errors shown in Figs. 19(a) and 19(b). Another key consideration is not to generate the complete Voronoi diagram but to update its partial cross section at the current scan line. This makes a single-plane sweep possible to detect the types of spacing errors shown in Figs. 19(c) and 19(d), with $O(\sqrt{n})$ main memory space. For this purpose, two more worklists are used to separately store southeast and southwest corners of the opaque polygons, which are visible from the current scan line, together with some additional data needed to update the partial Voronoi diagram.

The enhanced plane-sweep method was implemented in an experimental program written in PL/I and run on the Hitachi M680H computer. The CPU time for a 129K vertex example is 29.8 seconds, and that for a 256K vertex example is 60.4 seconds.

Unlike batched-mode layout verification, in which the plane sweep method can be exploited, interactive layout verification and gridless routing involve several two-dimensional on-line mode problems. Among them the following basic search problems are important in many applications; only rectilinear polygonal patterns are considered here for simplicity of our arguments.

Problem B. For a set D of n horizontal line segments and a vertical line segment query V:

1) report all line segments of D intersecting V, or
2) find the uppermost (lowermost) one among them.

Problem C. For a set D of n horizontal line segments and a rectangle query R:

1) report all line segments of D intersecting R, or
2) find the uppermost (lowermost) one among them.

Note that those equivalent to the preceding problems by symmetry are not stated here.

To deal with these problems, McCreight proposed an interesting and powerful data structure called a "priority search tree" [95]. Originally, it was designed to represent a dynamic set of points in the plane so as to efficiently search the points inside a query rectangle determined by the set of points (a, b) such that, for given three numbers x_1, x_2, and y, $x_1 \leqq a \leqq x_2$ and $0 \leqq b \leqq y$. The key consideration here is that a horizontal or vertical half-line, instead of a line segment, can be identified by two numbers. To manage the set of horizontal line segments, we bisect the plane by a vertical slice line in such a way that the number of line segments to the left and to the right of the bisector are balanced. Continuing the bisection recursively, the set D of horizontal line segments is represented by a two-level tree with $O(n)$ memory space. The external one is a binary search tree of $O(\log n)$ depth representing the hierarchical bisections. Each node of the external tree refers to two priority search trees of $O(\log n)$ depth representing the fictitious half-lines to the left and to the right of a bisector. Based on the sophisticated structure, Problem B can be solved in $O(\log^2 n + k)$ time, where k is the number of reported intersections. Moreover, an update operation of inserting or deleting an item can be done in $O(\log n)$ time, provided that the distribution of data to be managed are known in advance. McCreight's method is considered to be the best available to deal with the likes of Problem B among those requiring only linear memory space.

There is another data structure, called a "segment tree" [96], applicable to Problem B. It was originally designed to solve the one-dimensional version of Problem B in $O(\log n)$ time, in which the horizontal line segments are replaced by intervals and the vertical line-segment query by a point. By means of a similar technique of building a two-level tree, it achieves the same time complexity as a priority search tree. A modified version, called a "layered segment tree" [97], was proposed to improve the search time from $O(\log^2 n)$ to $O(\log n)$. Imai and Asano [99] have further extended the data structure, based on the idea proposed in [98], so that an update operation can be done in $O(\log n)$ time for the restricted case where line segments are updated by insertions only or deletions only. The preceding extension of segment trees is theoretically interesting and achieves better time complexity than priority search trees. It is not recommended for practical use, however, because it requires $O(n \log n)$ memory space and involves nontrivial programming overheads.

The aforementioned data structures, with some modifications, can also be used to solve Problem C, and they achieve approximately the same expected time complexity as when applied to Problem B. However, there is no known data structure for Problem C that achieves worst-case time complexity better than $O(n)$ within linear memory space in the dynamic case, where items are inserted and deleted at a random sequence.

In contrast to the preceding approaches taken from theoretical points of view, several techniques aiming at practical efficiency have been proposed. Edaharo *et al.* [100] proposed an algorithm for layout verification based on the *bucketing* technique, in which the plane is divided into $O(n)$ rectangles of equal size, called "buckets," and the global solution is obtained from local solutions within buckets by means of a divide-and-conquer technique. The line segments lying on several buckets are treated by a special operation, called "filtering." Hitachi is using a similar technique for interactive VLSI layout based on a "field-block" data structure [101]. In this technique, each field-block is determined by a minimal rectangle which encloses approxi-

mately the same number (several tens) of line segments. Both of these techniques are expected to search, insert, or delete an item in constant time for practical problem instances within linear memory space. Another technique proposed by NEC for its application to Printed Wiring Board (PWB) layout is based on the "slit tree" [102]. In typical two-layer wiring, one of the layers is dominated by vertical line segments and the other by horizontal ones. Then the slit tree represents recursive bisections of the layer dominated by vertical ones by means of vertical slice lines, and line segments intersecting or overlapping a common slice line are managed by a bilateral linked list. Another slit tree is used to manage the other layer as well. This simple data structure achieves $O(\log n)$ expected time for a search or update operation within linear memory space.

The techniques aiming at practical efficiency described in the preceding paragraph have poor worst-case time complexity as compared with the sophisticated data structures, such as priority search trees and segment trees. On the other hand, they are attractive in practice, because of their simplicity, and are efficient for problem instances of uniform data distribution.

A typical application of the search techniques for a dynamic set of line segments is gridless routing, which, unlike the classical routers, does not use grid graphs as in the Lee algorithm [43] but directly considers geometrical patterns. This approach is aimed at saving main memory and providing flexibility in accommodating complicated design rules. In the past ten years, routing algorithms in this direction have been proposed [76], [103]–[110]. Among them we will review algorithms for "gridless maze routers." By a gridless maze router we mean an incremental routing process, like the Lee algorithm [43] and the exhaustive line search [71], in which each net can be always interconnected unless blocked by the nets already routed. In those gridless maze routers, not only the path-finding process but also the postprocessing of updating geometrical information for subsequent interconnection occupies a major portion of total processing time for interconnecting nets one after another. Two-pin connection algorithms used for such gridless maze routers can be classified into four categories, as follows.

1) Visibility Graph Methods: The space for routing (assumed to be a polygonal region) is represented by a "visibility graph," in which each node is associated with a convex vertex of an obstacle and a branch is inserted if and only if the corresponding vertex pair is visible to each other. It is clear that the visibility graph can be constructed by a naive method in $O(n^3)$ time with $O(n^2)$ memory space for an n vertex polygonal region. Then a shortest path can be found in $O(n^2)$ time by means of the Dijkstra method. A more efficient method has been proposed to find a rectilinear shortest path in $O(n \log^2 n)$ time in the rectilinear geometry [106]. The key consideration here is to represent the visibility information by a modified graph which has only $O(n \log n)$ branches rather than $O(n^2)$. Disadvantages of this method are the memory overhead needed for maintaining the graph information and the complicated postprocessing after a path has been found.

2) Auxiliary Line Methods: As preprocessing, a pair of horizontal and vertical chords are drawn from each convex (90-degree) vertex of obstacle polygons, until they hit another obstacle. Note that the obstacles here can be rectilinear polygons of any shape. By associating a node with each point of interesection of these auxiliary lines (chords), a graph with $O(n^2)$ nodes and branches is derived. Then the Dijkstra method leads to a rectilinear shortest path in $O(n^2 \log n)$ time, with $O(n^2)$ memory space [107]. If we are satisfied with a minimum bend path rather than a shortest path, the former can be found much faster. For this purpose the preprocessing to calculate all the points of intersection in advance is not needed. By using priority search trees to store the auxiliary lines, Ohtsuki [90], [108] presented an algorithm, which after $O(n \log n)$ time preprocessing, finds a minimum bend path, including the postprocessing in $O(k \log^2 n)$ time with $O(n)$ memory space, where k is the searched and updated auxiliary lines. Note that the algorithm can be viewed as a gridless version of the exhaustive line search [71]. The methods using auxiliary lines, again, have a disadvantage in memory overhead for maintaining them.

3) Rectangular Partitioning Methods: The space for routing is partitioned into nonoverlapping rectangles, called "tiles" [184], by horizontal slice lines. Then a path is searched by exploiting the incidence relations between neighboring tiles [109], [110]. It is shown that, after $O(n \log n)$ preprocessing time, a minimum bend path can be found in $O(k \log k)$ time with $O(n)$ memory space, and the tile plane can be updated in $O(k)$ time, where k is the number of searched and updated tiles [109]. The algorithms of this category run very fast, but there is some difficulty in generalizing it to multilayer interconnection.

4) Area Search Methods: To reduce memory overhead in the implementation of gridless routers, it is desirable not to generate paths, auxiliary lines, partitioning lines, etc. as preprocessing. Although no algorithm in this direction which achieves good theoretical time complexity has been reported, the use of the practice-oriented data structures [100]–[102] in the routers of this category seems to be attractive for achieving better average processing time with less memory. The "line expansion" algorithm [111] has been suggested to work out such a gridless router. The basic operation in this algorithm, as shown in Fig. 20, is to search

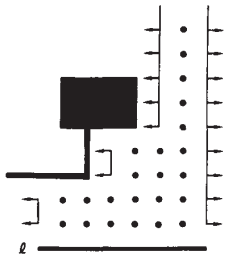


Fig. 20. Basic area search in grid expansion algorithm.

the area (grid points), which can be reached by moving a line segment or its portion in the perpendicular direction. Now it is seen that the gridless version of the search problem is reduced to Problem C, posed earlier. Thus various data structures for dynamic search problems can be applied. Moreover, when the CAM-based hardware engine, which will be described in section V, has become available, such a basic search problem can be solved in constant time, and therefore a path will be found in linear time [148].

In the last part of this section, we will review a few recent results on layout compaction in which the application of computational geometry was most successful. The enhanced plane sweep [94], which was successfully applied to design rule checking, is again becoming a powerful means for one-dimensional compaction. In contrast to previous compaction algorithms [112], [113], those based on plane sweep can bypass the use of constraint graphs and compaction grids, and therefore both processing time and space are saved to great extent. Furthermore, geometrical compactors make automatic jog insertion possible and provide intelligence not only for compacting a region of a chip, but also, if necessary, for expanding or adjusting it so as to match the layout of neighboring regions subject to a given design rule. A compactor possessing this intelligence is sometimes called a "spacer."

The channel spacer called Nutcracker [114], developed at the University of California at Berkeley, is based on the idea described in the preceeding paragraph. A similar result is also reported in [115]. The compaction algorithm implemented in Nutcracker essentially consists of two phases. In the first phase ("squeeze-down phase"), a channel routing result, followed by pertinent via reduction as shown in Fig. 21(a), is squeezed down subject to the design rule by scanning the channel from bottom to top. As a result, all possible jogs are inserted and the channel width is determined as shown in Fig. 21(b). In the second phase "(lift-up phase"), by scanning the channel from top to bottom, each wire with its vias lifted up so as to remove unnecessary jogs and reserve as much room in the lower part as possible for flexibility to linearize the remaining wires, Fig. 21(c) helps one understand what the output of the phase looks like. The second phase may still leave a small number of unnecessary jobs unremoved, as is seen from Fig. 21(c), which is caused by overreserving the space for the subsequent linearizations of the wires lying below. Such jogs can be removed by adding the third phase ("refinement phase"), which involves little overhead, and the final pattern, as shown in Fig. 21(d), is obtained [116].

Theoretically, the preceding algorithm runs in $O(n \log n + k)$ time with $O(n + k)$ memory space, where n is the total number of wire segments and vias, and k is the number of generated jogs. Practical efficiency of the algorithm was also confirmed by experimental results of Nutcracker, written in C and run on a VAX11/780. Many examples, with 69–252 nets, were tested, and the channel width was reduced at an average of 16.4 percent in less than 9 seconds.

The geometrical compaction based on plane sweep was extended to chip compaction [117]. Unlike the channel compaction, a difficult problem lies in that, whenever a block is moved, its terminals and all the wires connected to them must be moved simultaneously. To resolve the problem, a scheme for manipulating such a group of layout objects, called a "bag," is proposed in [117], and leads to an efficient chip compactor as well.

Another geometrical approach to compaction is reported in [118], where the tile plane representation of layout space as in the gridless maze routing [109], [110] is applied to "global spacing" of building-block layout. Here the term "global spacing" refers to placement modifications performed before detailed routing to modify the estimated channel capacity according to the result of global routing. The key operation for such placement modifications is to find an optimal compression ridge. The authors of [118], based on the tile plane, presented an $O(n)$ time algorithm for finding an optimal monotonic ridge, where n is the number of tiles. Note that previous methods require $O(n^2)$ time to find a monotonic ridge. Furthermore, the authors have generalized the compression ridges to be nonmonotonic

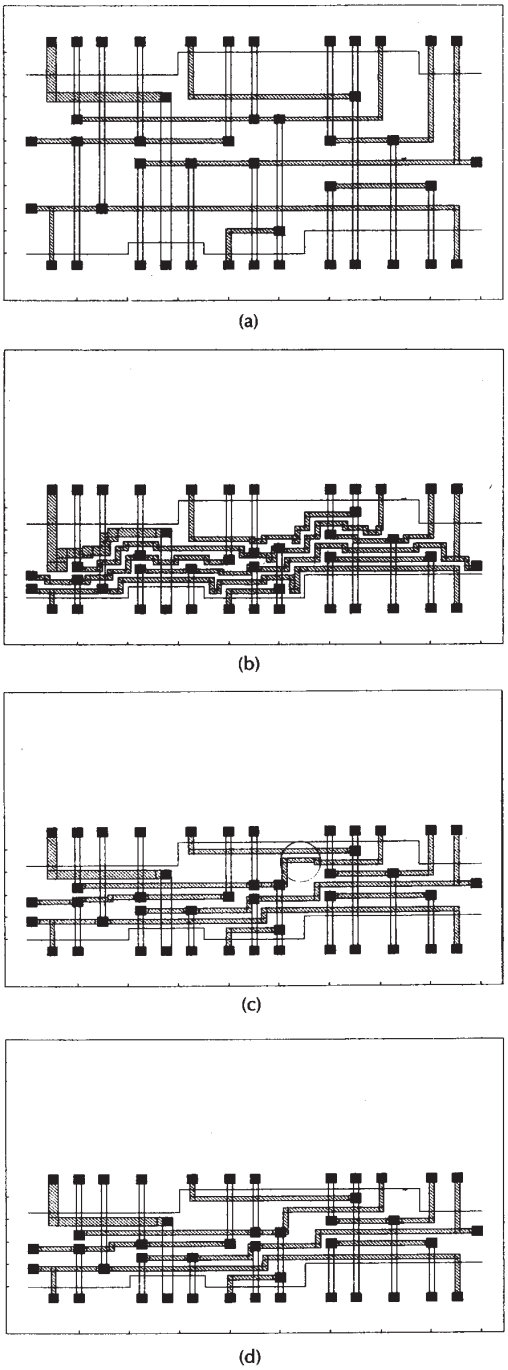


Fig. 21. Channel spacing based on plane sweep. (a) Channel routing result with via reduction. (b) After squeeze-down phase. (c) After lift-up phase. (d) Final refinement.

and developed an $O(n \log n)$ time algorithm to find an optimal one.

As has been reviewed, the one-dimensional compaction problems can be solved very fast by applying computational geometry. However, our ultimate goal is to establish efficient two-dimensional compaction algorithms. So far, only methods of repeating one-dimensional compactions to x and y directions alternatively or their slight modification are known [112], [113], and further research on the subject will be needed. Some recent papers that treated two-dimensional compaction can be found in [119]–[122].

V. Layout Engines

"Hardware accelerators," "backend processors," and "special-purpose engines" have become interesting research subjects in the past decade as viable solutions to a wide range of computer-aided design problems for VLSI. The motivation for developing such CAD engines is to overcome design size limitations of increasingly complex VLSI systems. On the other hand, recent advances in VLSI technologies have made it realistic to build such machines. Research and development on CAD engines, including commercial products, tools for in-house use, lab prototypes, and paper machines, cover almost all VLSI design phases. These results, with discussions on their problems and limitations, are surveyed in review articles [123], [124].

Unfortunately, CAD engines cannot be discussed without trade-offs [182]. A "point accelerator," i.e., an algorithm-built-in hardware dedicated to a very limited class of jobs, is one extreme that is most expensive and least flexible. The other extreme is to use a general-purpose computer with programs providing the greatest flexibility. It attains the least cost per application, through processing speed for a particular application is unsatisfactory. Taking the trade-off issue into account, it is becoming of a general understanding of CAD specialists that a "global accelerator," in which a multiprocessor architecture covers a wide range of design automation jobs, is most viable for business, considering the present and near future VLSI design environment. In addition, there seems to be another consensus, that point accelerators do not make sense in practice for anything but simulation. As a matter of fact, only simulation engines have been used to solve real problems in industry or brought into the commercial market. This is because simulation is the central part of the design process, is needed by a wide spectrum of users, and involves unbounded tasks the designers run into.

Despite the arguments against point accelerators for anything but simulation (from the viewpoint of economical viability), a wide variety of approaches has been reported: building engines for placement [127]–[135], routing [87], [136]–[145], [152], and design rule checking [146]–[152]. These contributions seem concerned with other kinds of viability as well; e.g., as a research topic, as a marketable product in the next generation design environment, etc. There seems to be a strong suggestion that, after the low-hanging fruit of simulation engines has been picked, the next reachable fruit should be targeted [182]. As such a target, layout engines are expected to reduce the intolerable computer time of software-implemented layout algorithms.

Recent contributions on layout engines are based on a wide variety of approaches and fall into several groups in terms of speed-up techniques and target applications. There have been many contributions on applying global accelerators with a processor-array architecture (such as the connection machine [125] and the hypercube [126]) to simulated annealing-based placement [127]–[133], [185] maze routing [136], [137], and design rule checking [146], [147]. Attempts to design point accelerators or special-purpose architectures with array or pipeline configurations for placement [134], [135], maze routing [87], [137]–[145], and design rule checking [147], [149], [152] have also been reported. As another approach, image processing engines are used to speed up design rule checking based on bit-map representation of geometrical patterns [150], [151]. Because of page limitations, we will not review these contributions but will focus the remaining discussion in this section on our own works [87], [152].

There have been several attempts [137]–[145] to build a hardware maze router which physically implements the Lee algorithm [43]. The primary idea here is to prepare an $N \times N$ array of identical "processor elements" (PE's) that has a one-to-one correspondence with the $N \times N$ grid plane, and to achieve linear run time for finding a path [137], [138]. Such a "full grid machine," though it obviously finds a path in linear time, has a serious disadvantage, in that it needs $O(N^2)$ PE's despite the fact that almost all of them are not utilized. The objective of the contribution in [87] was to build a new architecture that achieves linear run time with only $O(N)$ PE's.

The possibility of saving the number of PE's comes from the observation that only those grid cells on the current wavefront need to activate the corresponding PE's. In this sense, a hardware maze router is called a "wavefront machine" if it has only PE's to be assigned to wavefront cells [87]. It should be noted here that the average length of wavefronts for an $N \times N$ grid plane is of $O(N)$. Taking into account the trade-off between performance and processor utilization, and according to simulation, it is suggested in [87] that $N/2$ is the reasonable number of PE's for an $N \times N$ grid plane. The actual means of processor-to-cells mapping is exploiting the fact that a wavefront consists of diagonal line segments, and the mapping of PE's to any diagonal lines on the grid plane is optimized. Then, if there are $N/2$ PE's, the same PE appears at every $N/4$ PE's along any diagonal lines. Note that it is optimal because, when a PE is used to send a message during a wave propagation step, another PE must be simultaneously used to receive it. A figure showing how the $N/2$ PE's are mapped on an $N \times N$ grid plane is available in [87]. Such an optimal assignment can be realized by interconnecting PE's in a twisted-torus configuration.

Another key consideration for the proposed wavefront machine is to divide the PE's and grid cells into two groups, like the black and white squares of a chessboard. The grid cells on the current wavefront are all "black" or all "white," provided that a unit cost is assigned to each cell. Taking into account the bipartite structure of the grid plane, communication between the control unit and PE's is simplified.

A prototype machine with 64 PE's was developed in 1984 to implement the Lee algorithm on a 128×128 grid plane. The block diagram of its architecture is available in [87]. Roughly speaking, each PE is composed of an 8-bit microprocessor chip (NECμPD7800), an 8-kbyte RAM, an 8-kbyte ROM, a channel selector, and an ID number unit. The channel selector, made of CMOS switches, is added to allow

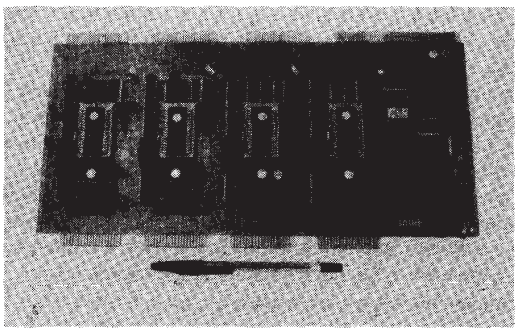


Fig. 22. Circuit board with four PE's embedded.

communication with the four neighbors. The ID number unit is used to identify which grid cells each PE is assigned to. Such a PE was built by eleven IC's. Fig. 22 is a photo of a printed wiring board with four PE's embedded, i.e., a total of 16 boards is used to assemble the wavefront machine with 64 PE's. For the control unit, we used a personal computer, NEC's PC8801, with a Z80 chip for its CPU.

The measured performance of the prototype machine for various sample problems is also available in [87]. Performance statistics taken on these sample problems are as follows. The "average multiple assignment factor" of a PE was observed to spread between 2 and 3, which suggests that the wavefront machine reduces the N^2 PE's needed in the full grid machine to $N/2$ with a speed degradation by a factor of 2–3. The average "processor utilization" of a PE was observed to be about 50 percent throughout all the sample problems. The average time spent to advance the wavefront one unit was about 26 ms, and the time for finding a path of length L was observed to be proportional to $L^{1.16}$, i.e., almost linear.

The prototype machine with 64 PE's is too small to be used in real design. A wavefront machine of reasonable size would be one with 1000 PE's to deal with a 2000 × 2000 grid plane. To realize such a machine, a custom chip must be developed. On the basis of recent VLSI technology, we estimate that the whole PE with ROM and RAM can be integrated in a single chip, and it results in a whole machine with about ten boards.

There is a critical issue against the viability of parallel maze routers in that some software speed-up techniques, e.g., those of [153], [186], would achieve almost linear expected running time for finding a path. It is true for connecting most of the nets. However, incremental routing tends to leave a small number of nets unconnected, and it usually occupies a major portion of the design turnaround time to connect the remaining nets by means of manual or interactive rework. The final clean-up phase involves the procedure to find the nets already routed which block an unconnected net. In this situation, any software speed-up techniques do not make sense because the wave must be propagated in the whole grid plane exhaustively, and the parallel processing on a wavefront can be well exploited to reduce $O(N^2)$ wave propagation time to $O(N)$. Based on this observation, the WIRES system [88] for interactive rip-up and rerouting was designed as described in section III-C, where the wavefront machine is used as a backend processor.

There is another issue, which is more fatal, against the viability of parallel maze routers. This is caused by the inherent nature of grid-based routing; i.e., no matter how the wavefront machine saves the number of PE's, we still need $O(N^2)$ memory space and, actually, the memory capacity must be at least doubled compared to software implementation. This negative feature of the grid-based approach has given rise to another class of routing methods, i.e., gridless routing, which was described in the preceding section.

To speed up gridless maze routers described in section IV, a new architecture [152] based on "content addressable memory" (CAM), elsewhere called "associative memory," was proposed. The "pattern matching" in hardware, the basic function of early bit-parallel CAM, can be extended to several more sophisticated word-parallel searches. Then, provided that the number of bits per word is fixed, hardware implementation of such searches achieves constant processing time independent of the number of words. Among them the following searches play an essential role for our purpose.

Equivalence search: For a given search key word K in the index register, enumerate all the words that match K within a partial content specified by the mask register.

Threshold search: For a given search key word K in the index register, enumerate all the words greater (or less) than K with respect to a partial content specified by the mask register.

Extremum search: Find the word with maximum (or minimum) value with respect to a partial content specified by the mask Register.

Fig. 23 shows the basic CAM configuration and how the equivalence search is done as an example. Note that the

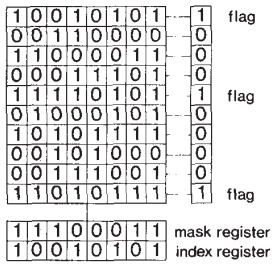


Fig. 23. Equivalence search on CAM.

partial content under consideration is specified by 0's in the mask register, i.e., 1's indicate the "don't care" bits, and the output is indicated by 1's in FLAG bit.

The preceding functions of CAM can be applied to the geometrical search problems posed in section IV. For example, Problem C1 can be solved in constant time by repeating the threshold search four times (with respect to each edge of the query rectangle R), where each horizontal line segment data is assumed to be stored in a CAM word in the format as shown in Fig. 24. In similar ways, almost all on-line mode search problems on polygonal patterns can be solved in constant time by virtue of the word-parallel nature of CAM. This implies that CAM makes all the gridless maze routers described in section IV possible for finding a path in linear time.

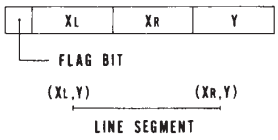


Fig. 24. Representation of line segment.

The outlined architecture of the proposed CAM-based layout engine is available in [152], which was designed as a back-end processor controlled by a host computer. It is intended to accelerate not only gridless routers but also a variety of geometrical problems encountered in VLSI design, such as design rule checking, layout compaction, artwork generation, etc.

A prototype machine named CHARGE (CAM-based hardware for geometrical search) was developed in 1987 to confirm the aforementioned idea. It consists of four major parts: CAM, sequencer, RAM, and ALU. In physical realization, too, there are four corresponding boards. A photo of the CAM board is given in Fig. 25. For the CAM part, we are currently using the 4-kbit CAM chips developed by NTT [154], which can be replaced by more recent 20-kbit chips [155]. The machine is designed to embed a maximum of 64 such chips. The 4-kbit CAM has 30 instructions, and each instruction can be carried out within a 140-ns clock cycle. The details of its specification are available in [154]. The sequencer part is used to drive the microcode. This part is very important, because the simple scheme, in which a CPU directly gives instructions to CAM, cannot follow that high-speed clock. RAM chips are used as shared memory for the host computer and the back-end processor (CHARGE). The ALU is used for preprocessing for generating input data to CAM and for other operations which are not suitable to CAM.

In implementing the geometrical search algorithms and the gridless routers described in section IV, two words (64 bits) of 4-kbit CAM chips are assigned to each rectilinearly oriented line segment. Then Problem B2, for example, can be processed within 300-cycle time units. Fig. 25 demonstrates the performance improvement CHARGE confers over software implementation. For software implementation, the priority search tree was used to represent line segments as the best available data structure. To make the comparison fair, the software program is implemented on a VAX11/785 computer with a 133-ns cycle time, whereas the instructions of CHARGE is driven by a 200-ns clock pulse. The performance for a small number of line segments was measured on the prototype machine with four CAM chips embedded, and that for large problems was extrapolated by means of the results on an emulator. The slight jump observed at the "hardware" curve in Fig. 26 is caused by the signal delay in two-stage configuration of priority encoder circuits for connecting several tens of CAM chips.

The CAM-based layout engine can be viewed as a point accelerator with a kind of SIMD configuration. Nevertheless, it provides the flexibility to deal with a variety of geometrical problems for VLSI design. Another advantage is that complicated coding for sophisticated data structures depending on subproblems is not needed, unlike software implementation. As a future direction, it is expected that higher density CAM chips and faster priority encoders will emerge.

Until now, a certain amount of effort and experience has

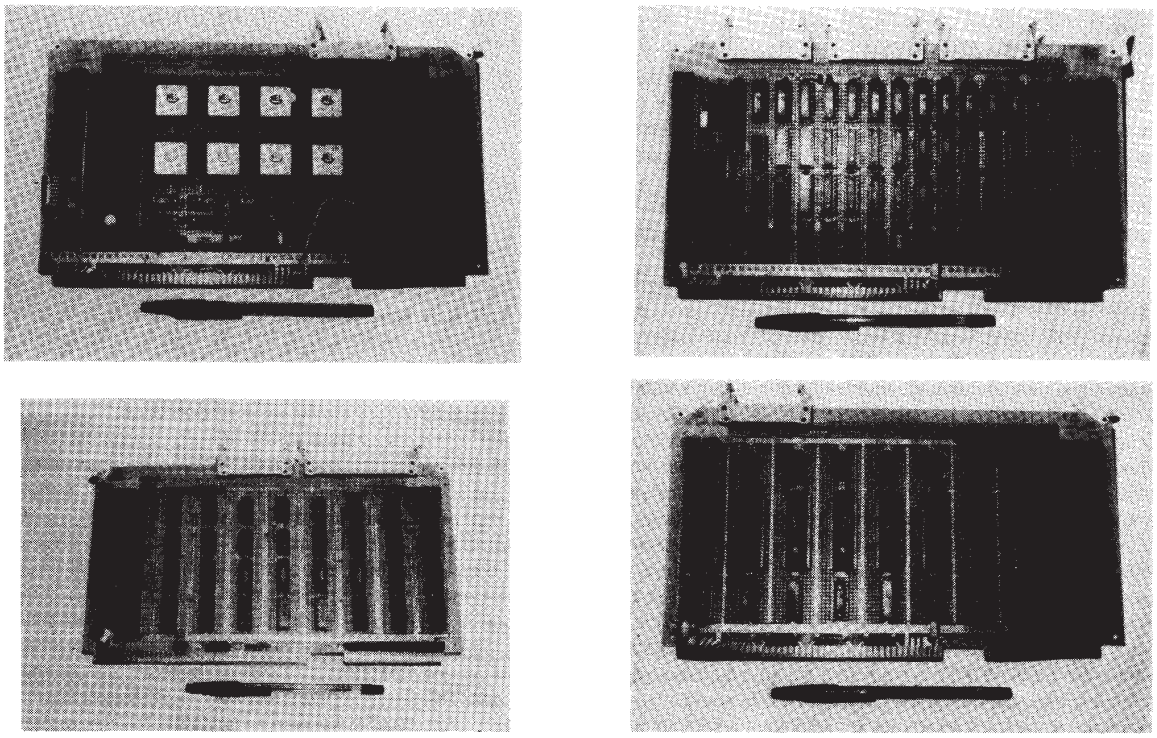


Fig. 25. CAM boards.

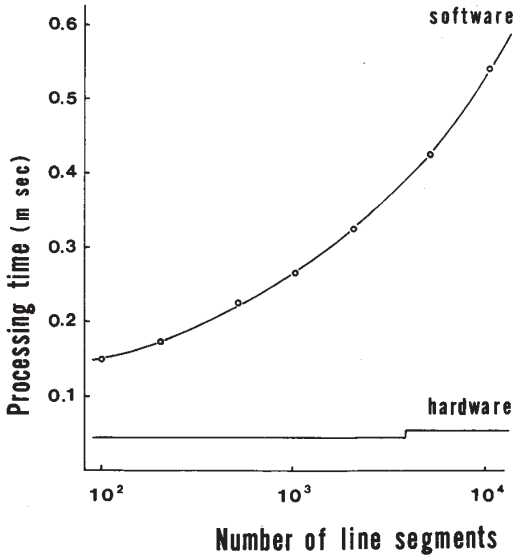


Fig. 26. Comparison of search time: software versus hardware.

been accumulated in research and development of CAD engines. According to those involved in the area, hardware assembly is not a major task, but software support, including debugging tools, have given rise to more critical issues. The advances needed for the next-generation accelerators include more flexible CAD engines, better parallel-programming environments, and more efficient parallel algorithms.

VI. Conclusion

We have discussed four major topics of current interest in VLSI layout. Unfortunately, because of space limitations, several other important topics were left out.

The subject of timing-driven layout is crucial in the design of high-density and high-speed chips. An automatic layout/simulation iteration loop has been proposed for standard-cell design at AT&T Bell Laboratories [156]. A hierarchical approach based on min-cut partitioning and placement for gate-array design which satisfies timing specifications has been proposed by Burstein and Youssel at IBM [157]. These approaches lack the ability to deal dynamically with chip timing in the paths during layout. Timing analysis is inherently path-oriented; physical design is net-oriented. The obvious way to do timing-driven layout is to optimize timing subject to net constraints. However, in doing so, one tends to overly constrain the problem solution. In [158], a new approach is introduced which dynamically optimizes the performance of the chip during placement. A linear-programming formulation is used, and the results on tested examples indicate that the approach is promising.

Current CAD efforts in routing concentrate on signal routing. In industry, manual routing of power and ground nets is still being used for most chip designs. Only in the past three years has some research been reported on automatic sizing of power/ground (P/G) segments [159], [160] and pad assignment [161]. A more general approach for automatic sizing of P/G networks has been reported recently [162]. In this approach, no restrictions are imposed on the network topology or the number of applying pads. Wire widths are optimally determined subject to the constraints of voltage drops and electromigration. As we enter the phase of future chip designs with multimetal layers, more innovative approaches are needed for P/G routing.

In interconnection and packaging design, considerable advance has been made in hardware development in recent years [163]. To name a few, there are surface mounting boards, multilayer ceramics, multichip and wafer scale integration, high-density multichip interconnect (HMDI) [164], and button boards [165]. While computer-aided design has had an enormous impact at the chip level, its impact on general interconnet and packaging has been limited. Even for IC's, three-dimensional integration is within sight [166]. There is a crucial need for basic research on three-dimensional interconnect [167]. In this connection, some recent work on multilayer routing has been reported [168], [169].

Another topic of interest and importance is analog circuit layout. Because of many inherently complicated constraints, the difficulties encountered are numerous; to name a few, a combinatorial choice of the geometrical configuration of passive devices, having to consider abutting and isolating a group of transistors, the effect of signal delay due to parasitics, the need to shield of sensitive signal nets, etc. These are good topics for future research. Nonetheless, there are recent efforts worth noting [170], [171].

Acknowledgment

The authors wish to acknowledge the research contributions and suggestions of W. Dai, B. Eschermann, C. P. Hsu, M. Marek-Sadowska, M. Pedram, M. Sato, A. Srinivasan, K. Suzuki, R. S. Tsay, and X. Xiong.

References

[1] J. Soukup, "Circuit layout," *Proc. IEEE*, vol. 69, no. 10, pp. 1281–1304, Oct. 1981.
[2] T. C. Hu and E. S. Kuh, Eds., *VLSI Circuit Layout: Theory and Design.* New York: IEEE Press, 1985.
[3] T. Ohtsuki, Ed., *Layout Design and Verification.* Amsterdam: North Holland, 1986.
[4] G. De Micheli, A. Sangiovanni-Vincentelli, and P. Antognetti, Eds., *Design Systems for VLSI Circuits.* Dordrecht, The Netherlands: Martinus Nijhoff Publishers, 1987.
[5] C. Sechen and A. Sangiovanni-Vincentelli, "Timber Wolf 3.2: A new standard cell placement and global routing package," in *Proc. 23rd DAC*, 1986, pp. 432–439.
[6] C. Sechen and K-W. Lee, "An improved simulated annealing algorithm for row-based placement," in *Digest of Tech. Papers, ICCAD*, 1987, pp. 478–481.
[7] H. Kato, H. Kawanishi, S. Goto, T. Oyamada, and K. Kani, "On automated wire routing for building-block MOS LSI," in *Proc. IEEE ISCAS*, 1974, pp. 309–312.
[8] S. Kimura, N. Kubo, T. Chiba, and I. Nishioka, "An automatic routing scheme for general cell LSI," *IEEE Trans. Computer-Aided Design*, vol. CAD-2, no. 4, pp. 285–292, Oct. 1983.
[9] T. Kozawa, C. Miura, and H. Terai, "Combine and top down block placement algorithm for hierarchical logic VLSI layout," in *Proc. 21st DAC*, 1984, pp. 667–669.
[10] C-K. Cheng and E. S. Kuh, "Module placement based on resistive network optimization," *IEEE Trans. Comput. Aided Design*, vol. CAD-3, no. 3, pp. 218–225, July 1984.
[11] K. M. Hall, "An *r*-dimension quadratic placement algorithm," *Manag. Sci.*, vol. 17, no. 3, pp. 219–229, Nov. 1970.
[12] C. P. Hsu, S. Evans, J. Tang, K. Chow, R. Perry, and J. Liu, "An effective hierarchical approach to high complexity circuit layout," in *Proc. IEEE Custom Integrated Circuits Conf.*, 1987, pp. 614–617.

[13] R-S. Tsay, E. S. Kuh, and C-P. Hsu, "PROUD: A sea-of-gates placement algorithm," *IEEE Design and Test of Computers*, pp. 44–56, Dec. 1988.
[14] W-M. Dai, H. H. Chen, R. Dutta, M. Jackson, E. S. Kuh, M. Marek-Sadowska, M. Sato, D. Wang, and X-M. Xiong, "BEAR: A new building-block layout system," in *Digest of Tech. Papers, IEEE ICCAD*, Nov. 1987, pp. 34–37.
[15] W-M. Dai, T. Asano, and E. S. Kuh, "Routing region definition and ordering scheme for building-block layout," *IEEE Trans. Comput. Aided Design*, vol. CAD-4, no. 3, pp. 189–197, July 1985.
[16] F. J. Kurdahi and A. C. Parker, "PLEST: A program for area estimation of VLSI circuits," in *Proc. 23rd DAC*, 1986, pp. 467–473.
[17] L. Sha and R. W. Dutton, "An analytical algorithm for placement of arbitrarily sized rectangular blocks," in *Proc. 22nd DAC*, 1985, pp. 602–607.
[18] M. A. Breuer, "A class of min-cut placement algorithms," in *Proc. 14th DAC*, 1977, pp. 284–290.
[19] U. Lauther, "A min-cut placement algorithm for general cell assemblies based on a graph representation," in *Proc. 16th DAC*, June 1979, pp. 1–10.
[20] B. W. Kernighan and S. Lin, "An efficient heuristic procedure for partitioning graphs," *Bell Syst. Tech. J.*, vol. 49, no. 2, pp. 291–307, 1970.
[21] D. P. LaPotin and S. W. Director, "Mason: A global floorplanning approach for VLSI design," *IEEE Trans. Comput. Aided Design*, vol. CAD-5, no. 4, pp. 477–489, Oct. 1986.
[22] A. A. Szepieniec and R. Otten, "The genealogical approach to the layout problem," in *Proc. 17th DAC*, 1980, pp. 535–542.
[23] R. L. Brooks, C. A. B. Smith, A. H. Stone, and W. T. Tutte, "The dissection of rectangles into squares," *Duke Math. J.*, vol. 7, pp. 312–340, 1940.
[24] T. Ohtsuki, N. Suziyama, and H. Kawanishi, "An optimization technique for integrated circuit layout design," in *Proc. ICCST—Kyoto*, 1970, pp. 67–68.
[25] L. Stockmeyer, "Optimal orientation of cells in slicing floorplan designs," *Inform. Control*, vol. 57, pp. 91–101, 1983.
[26] S. Wimer, I. Koren, and I. Cederbaum, "Floor plans, planar graphs, and layouts," *IEEE Trans. Circuits Syst.*, vol. CAS-35, no. 32, pp. 267–278, Mar. 1988.
[27] S. Wimer, I. Koren, and I. Cederbaum, "Optimal aspect ratios of building blocks in VLSI," in *Proc. 25th DAC*, 1988, pp. 66–72.
[28] K. Zibert and R. Saal, "On computer-aided hybrid circuit layout," in *Proc. IEEE ISCAS*, 1974, pp. 314–318.
[29] R. H. J. M. Otten, "Efficient floorplan optimization," in *Proc. ICCD*, 1983, pp. 499–502.
[30] N. Suziyama, S. Nemoto, K. Kani, T. Ohtsuki, and H. Watanabe, "An integrated circuit layout design program based on a graph-theoretical approach," in *Digest IEEE ISSCC*, 1970, pp. 86–87.
[31] J. Grason, "An approach to computerize space planning using graph theory," in *Proc. 8th DAC*, 1971, pp. 170–179.
[32] W. R. Heller, G. Sorkin, and K. Maling, "The planar package planner for system designers," in *Proc. 19th DAC*, 1982, pp. 253–260.
[33] K. Kozminski and E. Kinnen, "An algorithm for finding a rectangular dual of a planar graph for use in area planning for VLSI integrated circuits," in *Proc. 21st DAC*, 1984, pp. 655–656.
[34] S. M. Leinwand and Y-T. Lai, "An algorithm for building rectangular floorplans," in *Proc. 21st DAC*, 1984, pp. 663–664.
[35] S. Tsukiyama, K. Koike, and I. Shirakawa, "An algorithm to eliminate all complex triangle in a maximar planar graph for use in VLSI floorplan," in *Proc. IEEE ISCAS*, 1986, pp. 321–324.
[36] W-M. Dai and E. S. Kuh, "Simultaneous floor planning and global routing for hierarchical building-block layout," *IEEE Trans. Comput. Aided Design*, vol. CAD-6, no. 5, pp. 828–837, Sept. 1987.
[37] B. Eschermann, W-M. Dai, E. S. Kuh, and M. Pedram, "Hierarchical placement for macrocells: A 'meet in the middle' approach," in *Digest of Tech. Papers, ICCAD*, Nov. 1988, pp. 460–463.
[38] B. Eschermann, "Hierarchical placement for macrocells with simultaneous routing area allocation," Electronics Research Laboratory, University of California, Berkeley, memo. no. UCB/ERL M88/49, July 1988.
[39] B. T. Preas and P. G. Karger, "Automatic placement: A review of current techniques," in *Proc. 23rd DAC*, 1986, pp. 622–629.
[40] T. R. Muller, D. F. Wong, and C. L. Liu, "An enhanced bottom-up algorithm for floorplan design," in *Digest of Tech. Papers, ICCAD*, 1987, pp. 524–527.
[41] N. P. Chen, C-P. Hsu, and E. S. Kuh, "The Berkeley building-block layout system for VLSI design," in *Proc. VLSI 83*, Aug. 1983, pp. 37–44.
[42] C-P. Hsu, "General river routing algorithm," in *Proc. 20th DAC*, 1983, pp. 578–583.
[43] C. Y. Lee, "An algorithm for path connection and its applications," *IEEE Trans. Electron. Comput.*, vol. EC-10, pp. 346–365, 1961.
[44] M. Hanan, "On Steiner's problem with rectilinear distance," *SIAM J. Appl. Math.*, vol. 14, no. 2, pp. 255–265, 1966.
[45] S. L. Hakimi, "Steiner's problem in graphs and its application," *Networks*, vol. 1, pp. 113–133, 1971.
[46] J. A. Wald and C. J. Colbourn, "Steiner trees, partial two-trees, and minimum IFI networks," *Networks*, vol. 13, pp. 159–167, 1983.
[47] W-M. Dai, "BEAR: A macrocell layout system for VLSI circuits," Ph.D. thesis, EECS Dept., University of California, Berkeley, 1988.
[48] W. K. Luk, P. Sipda, M. Tamminen, D. Tang, L. S. Woo, and C. K. Wong, "A hierarchical global wiring algorithm for custom chip design," *IEEE Trans. Comput. Aided Design*, vol. CAD-6, no. 4, pp. 518–533, July 1987.
[49] E. S. Kuh and M. Marek-Sadowska, "Global routing," in *Layout Design and Verification*, T. Ohtsuki, Ed. Amsterdam: North Holland, 1986, pp. 169–198.
[50] M. P. Vecchi and S. Kirkpatrick, "Global wiring by simulated annealing," *IEEE Trans. Comput. Aided Design*, vol. CAD-2, no. 4, pp. 215–222, Oct. 1983.
[51] J. T. Li and M. Marek-Sadowska, "Global routing for gate array," *IEEE Trans. Comput. Aided Design*, vol. CAD-3, pp. 298–307, 1984.
[52] B. S. Ting and B. N. Tien, "Routing techniques for gate array," *IEEE Trans. Comput. Aided Design*, vol. CAD-2, no. 4, pp. 301–312, Oct. 1973.
[53] K. Aoshima and E. S. Kuh, "Multichannel optimization in gate-array LSI layout," in *Proc. IEEE ISCAS*, 1983, pp. 1005–1089.
[54] M. Marek-Sadowska, "Router planner for custom chip design," in *Digest of Tech. Papers, ICCAD*, 1986, pp. 246–249.
[55] U. Lauther, "Top-down hierarchical global routing for channelless gate arrays based on linear assignment," in *Proc. of IFIP VLSI 87*, 1987, pp. 109–120.
[56] R. E. Burkard and U. Deirgs, *Assignment and Matching Problems: Solution Methods with Fortran Programs.* Springer Verlag, 1980.
[57] B. S. Ting, E. S. Kuh, and I. Shirakawa, "The multilayer routing problem: Algorithms and necessary and sufficient conditions for the single-row, single-layer case," *IEEE Trans. Circuits Syst.*, vol. CAS-23, no. 12, pp. 768–778, Dec. 1976.
[58] E. S. Kuh, T. Kashiwabara, and T. Fujisawa, "On optimum single-row routing," *IEEE Trans. Circuits Syst.*, vol. CAS-26, no. 6, pp. 361–386, July 1979.
[59] M. Marek-Sadowska, "Two-dimensional router for double layer layout," in *Proc. 22nd DAC*, 1985, pp. 117–123.
[60] A. Hashimoto and J. Stevens, "Wiring routing by optimizing channel assignment within large apparatus," in *Proc. DAC*, pp. 155–169, 1977.
[61] R. Y. Pinter, "Optimal layer assignment interconnect," in *Proc. ICCC*, pp. 398–401, 1982.
[62] R. W. Chan, Y. Kajitani, and S. P. Chan, "A graph-theoretic via minimization algorithm for two-layer printed circuit boards," *IEEE Trans. Circuits Syst.*, vol. CAS-30, pp. 284–299, May 1983.
[63] K. C. Chang and D. H-C. Du, "Efficient algorithms for layer assignment problem," *IEEE Trans. Comput. Aided Design*, pp. 67–78, Jan. 1987.
[64] X-M. Xiong and E. S. Kuh, "The constrained via minimization

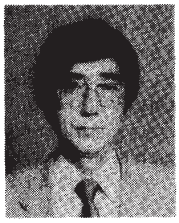
problem for PCB and VLSI designs," in *Proc. 25th DAC*, June 1988, pp. 573–578.
[65] C-P. Hsu, "Minimum-via topological routing," *IEEE Trans. Comput. Aided Design*, vol. CAD-2, no. 4, pp. 235–245, Oct. 1983.
[66] M. Marek-Sadowska, "An unconstrained topological via minimization problem for two-layer routing," *IEEE Trans. Comput. Aided Design*, vol. CAD-3, no. 3, pp. 184–190, July 1984.
[67] C-P. Hsu, "Theory and algorithms for signal routing in integrated circuit layout," Ph.D. thesis, University of California, Berkeley, 1983.
[68] X-M. Xiong and E. S. Kuh, "A unified approach to the via minimization problem," *IEEE Trans. Circuits Syst.*, vol. CAS-36, no. 2, pp. 190–204, Feb. 1989.
[69] H. H. Chen, "Routing L-shaped channels in nonslicing structure placement," in *Proc. 24th DAC*, 1987, pp. 152–158.
[70] D. W. Hightower, "A solution to the line-routing problem on the continuous plane," in *Proc. 6th DAC*, 1969, pp. 1–24.
[71] K. Mikami and T. Tabuchi, "A computer program for optimal routing of printed circuit connections," in *Proc. IFIPS*, vol. H47, pp. 1475–1478, 1968.
[72] C-P. Hsu, "A new two-dimensional routing algorithm," in *Proc. 19th DAC*, 1982, pp. 46–50.
[73] M. Burstein and R. Pelavin, "Hierarchical wire routing," *IEEE Trans. Comput. Aided Design*, vol. CAD-2, no. 4, pp. 223–234, Oct. 1983.
[74] R. Joobbani, *Weaver: A Knowledge-Based Routing Expert*. Dordrecht, The Netherlands: Kluwer Academic Press, 1986.
[75] H. Shen and A. Sangiovanni-Vincentelli, "Mighty: A rip-up and reroute detailed router," in *Digest of Tech. Papers, ICCAD*, Nov. 1986, pp. 2–5.
[76] J. P. Cohoon and L. H. Patrick, "BEAVER: A computational-geometry-based tool for switchbox routing," *IEEE Trans. Comput. Aided Design*, vol. CAD-7, no. 6, pp. 684–696, June 1988.
[77] J. Soukup and J. Fournier, "Pattern router," in *Proc. IEEE ISCAS*, 1979, pp. 486–489.
[78] T. Asano, "Parametric pattern router," in *Proc. 19th DAC*, 1982, pp. 411–417.
[79] A. Srinivasan and N. Shenoy, "Template-based pattern routing," EECS 244 class report, University of California, Berkeley, 1988.
[80] A. Srinivasan, "MOLE—A new template based router," Electronics Research Laboratory, University of California, Berkeley, memo. no. UCB/ERL M89/58, 1989.
[81] T. Nishizeki, N. Saito, and K. Suzuki, "A linear-time routing algorithm for convex grids," *IEEE Trans. Comput. Aided Design*, vol. CAD-4, no. 1, pp. 68–76, Jan. 1985.
[82] K. Onaga, "Distributed computing and network flow theoretic approach to simultaneous VLSI switchbox routing on two-dimensional processor arrays," in *Digest of Tech. Papers, ICCAD*, 1986, pp. 352–355.
[83] H. Mori, T. Fujita, M. Annaka, S. Goto, and T. Ohtsuki, "An advanced interactive layout design system for printed wiring board," in *Proc. Int. Conf. on Circuits and Computers*, 1980, pp. 754–757.
[84] W. A. Dees and P. G. Karger, "Automated rip-up and reroute techniques," in *Proc. 19th DAC*, 1982, pp. 432–439.
[85] I. Shirakawa and S. Futagami, "A rerouting scheme for single-layer printed wiring boards," *IEEE Trans. Comput. Aided Design*, vol. CAD-2, no. 4, pp. 261–271, Oct. 1983.
[86] T. Ohtsuki, "Maze-running and line-search algorithms," in *Layout Design and Verification*, T. Ohtsuki, Ed. Amsterdam: North Holland, 1986, pp. 99–131.
[87] K. Suzuki, Y. Matsunaga, M. Tachibana, and T. Ohtsuki, "A hardware maze router with application to interactive rip-up and reroute," *IEEE Trans. Comput. Aided Design*, vol. CAD-5, no. 4, pp. 466–476, Oct. 1986.
[88] T. Namekawa, K. Suzuki, H. Takano, and T. Ohtsuki, "Promoting efficiency of rip-up and reroute method and its evaluation (in Japanese)," IEICE monograph, VLD 88-91, pp. 39–46, Feb. 1989.
[89] D. T. Lee and F. P. Preparata, "Computational geometry—A survey," *IEEE Trans. Comput.*, vol. C-33, no. 12, pp. 1072–1101, 1984.
[90] T. Asano, M. Sato, and T. Ohtsuki, "Computational geometry algorithms," in *Layout Design and Verification*, T. Ohtsuki, Ed. Amsterdam: North Holland, 1986, pp. 295–347.
[91] N. Wirth, *Algorithms + Data Structure = Program*. Englewood Cliffs, NJ: Prentice-Hall, 1976.
[92] K. Yoshida, "Layout verification," in *Layout Design and Verification*, T. Ohtsuki, Ed. Amsterdam: North Holland, 1986, pp. 237–265.
[93] M. I. Shamos and D. Hoey, "Closest-point problems," in *Proc. 16th IEEE Symp. Foundations of Computer Sci.*, 1975, pp. 151–162.
[94] M. Sato, J. B. Kim, T. Awashima, and T. Ohtsuki, "A theoretically optimal and practically fast algorithm for VLSI geometrical design rule verification," in *Proc. IEEE ISCAS*, 1988, pp. 1445–1448.
[95] E. M. McCreight, "Priority search trees," *SIAM J. Comput.*, vol. 14, no. 2, pp. 257–276, 1985.
[96] J. L. Bentley and D. Wood, "An optimal worst case algorithm for reporting intersections of rectangles," *IEEE Trans. Comput.*, vol. C-29, no. 7, pp. 571–577, 1980.
[97] V. K. Vaishnavi and D. Wood, "Rectilinear line segment intersection, layered segment trees, and dynamization," *J. Algorithm*, vol. 3, pp. 160–176, 1982.
[98] H. N. Gabow and R. E. Tarjan, "A linear-time algorithm for a special case of disjoint set union," in *Proc. 15th Ann. ACM Symp. on Theory of Computing*, 1983, pp. 246–251.
[99] H. Imai and T. Asano, "Dynamic segment intersection search with applications," in *Proc. 25th Ann. IEEE Symp. on Foundations of Computer Sci.*, 1984, pp. 393–402.
[100] M. Edahiro, K. Tanaka, T. Hoshino, and T. Asano, "A bucketing algorithm for the orthogonal segment intersection search problem and its practical efficiency," *Algorithmica*, vol. 4, pp. 61–76, 1986.
[101] G. Suzuki and N. Hamada, "Practical data structure for incremental design rule checking and compaction," in *Proc. IEEE Int. Conf. on Computer Design*, 1987, pp. 442–447.
[102] I. Kato, S. Ohhira, and Y. Hisatomi, "A method of pattern data management of PWB layout system," (in Japanese) in *Proc. 35th Ann. Convention IPS Japan*, 1987, pp. 2429–2430.
[103] W. Lipski Jr., "Finding a Manhattan path and related problems," *Networks*, vol. 13, pp. 399–409, 1983.
[104] T. Ohtsuki, M. Sato, and I. Kojima, "Computational geometry approach to wire routing design," in *Proc. ECCTD 83*, 1983.
[105] T. Ohtsuki and M. Sato, "Gridless routers for two-layer interconnection," in *Digest of Tech. Papers, ICCAD*, 1984, pp. 76–78.
[106] K. L. Clarkson, S. Kapoor, and P. M. Vaidya, "Rectilinear shortest paths through polygonal obstacles in $O(n(\log n)^2)$ time," in *Proc. 3rd Ann. Symp. on Computational Geometry*, 1987, pp. 251–257.
[107] Y. F. Wu, P. Widamayer, M. D. F. Schlag, and C. K. Wong, "Rectilinear shortest paths and minimum spanning trees in the presence of rectilinear obstacles," *IEEE Trans. Comput.*, vol. C-36, no. 3, pp. 321–331, 1987.
[108] T. Ohtsuki, "Gridless routers–new wire routing algorithms based on computational geometry," presented at Int. Conf. on Circuits and Systems in China, 1985.
[109] M. Sato, J. Sakanaka, and T. Ohtsuki, "A fast line-search method based on a tile plane," in *Proc. IEEE ISCAS*, 1987, pp. 588–591.
[110] A. Margarino, A. Romano, A. De Gloria, F. Curatelli, and P. Antognetti, "A tile-expansion router," *IEEE Trans. CAD*, vol. CAD-6, no. 4, pp. 507–517, 1987.
[111] W. Heyns, W. Sansen, and H. Beke, "A line-expansion algorithm for the general routing problem with a guaranteed solution," in *Proc. 17th Design Automation Conf.*, 1980, pp. 243–249.
[112] Y. E. Cho, "A subjective review of compaction," in *Proc. 22nd Design Automation Conf.*, 1985, pp. 396–404.
[113] D. A. Mlynski and C. H. Sung, "Layout compaction," in *Layout Design and Verification*, T. Ohtsuki, Ed. Amsterdam: North Holland, 1986.
[114] X. M. Xiong and E. S. Kuh, "Nutcracker: An efficient and intelligent channel spacer," in *Proc. 24th Design Automation Conf.*, 1987, pp. 298–304.
[115] C. K. Cheng and D. N. Deutch, "Improved channel routing

by via minimization and shifting," in *Proc. Int. Workshop on Placement and Routing*, North Carolina, 1988, pp. 1–12.
[116] J. B. Kim, N. Nakajima, M. Sato, and T. Ohtsuki, "A fast intelligent channel spacer with automatic jog insertion and via reduction," (in Japanese) *Trans. IEICE Japan*, vol. J72-A, no. 2, pp. 349–358, 1989.
[117] X. M. Xiong, "Optimized one-dimensional compaction of building-block layout," technical report, University of California, Berkeley, memo. no. UCB/ERL M87/45, 1987.
[118] W. M. Dai and E. S. Kuh, "Global spacing of building-block layout," in *Proc. VLSI'87*, 1987, pp. 193–205.
[119] M. Schlag, Y. Z. Liao, and C. K. Wong, "An algorithm for optimal two-dimensional compaction of VLSI layouts," *Integration*, pp. 179–209, 1983.
[120] G. Kedem and H. Watanabe, "Graph-optimization techniques for IC layout and compaction," *IEEE Trans. Comput. Aided Design*, vol. CAD-3, no. 1, pp. 12–20, 1984.
[121] W. Wolf, R. Methews, J. Newkirk, and R. Dutton, "Algorithms for optimizing two-dimensional symbolic layout compaction," *IEEE Trans. Comput. Aided Design*, vol. CAD-7, no. 4, pp. 451–466, 1988.
[122] H. Shin, A. Sangiovanni-Vincentelli, and C. Sequin, "Two-dimensional compaction by zone-refining," in *Proc. 23rd DAC*, 1986, pp. 115–122.
[123] T. Blank, "A survey of hardware accelerators used in computer-aided design," *IEEE Design Test Comput.*, vol. 1, no. 4, pp. 21–39, 1984.
[124] N. Koike and K. Ohmori, "Design automation machine," in *Design Methodologies*, S. Goto, Ed. North Holland, pp. 465–499, 1986.
[125] W. D. Hills, *The Connection Machine*. Cambridge, MA: MIT Press, 1986.
[126] J. Palmer, S. Colley, J. P. Hayes, T. N. Mudge, and Q. F. Stout, "Architecture of a hypercube supercomputer," in *Proc. Int. Conf. on Parallel Processing*, 1986, pp. 653–660.
[127] M. Jones and P. Banerjee, "Performance of a parallel algorithm for standard cell placement on the Intel hypercube," in *Proc. 24th DAC*, 1987, pp. 807–813.
[128] A. Casotto, F. Romeo, and A. Sangiovanni-Vincentelli, "A parallel simulated annealing algorithm for the placement of marco-cells," *IEEE Trans. Comput. Aided Design*, vol. CAD-6, no. 5, pp. 838–847, 1987.
[129] F. Darema, S. Kirkpatrick, and V. A. Norton, "Parallel techniques for chip placement by simulated annealing on shared memory systems," in *Digest of Tech. Papers, ICCAD*, 1987, pp. 87–90.
[130] R. Jayaraman and R. A. Rutenbar, "Floorplanning by annealing on a hypercube multiprocessor," in *Digest of Tech. Papers, ICCAD*, pp. 346–349.
[131] C. Wong and R. Fiebrich, "Simulated annealing-based circuit placement algorithm on the connection machine system," in *Digest of Tech. Papers, ICCAD*, pp. 78–82.
[132] A. Casotto and A. Sangiovanni-Vincentelli, "Placement of standard cells using simulated annealing on the connection machine," in *Digest of Tech. Papers, ICCAD*, pp. 350–353.
[133] J. S. Rose, W. M. Snelgrove, and Z. G. Vranesic, "Parallel standard cell placement algorithms with quality equivalent to simulated annealing," *IEEE Trans. Comput. Aided Design*, vol. CAD-7, no. 3, pp. 387–396, 1988.
[134] C. P. Ravikumar and L. M. Patnaik, "Parallel placement by simulated annealing," in *Proc. IEEE ICCD87*, 1987, pp. 91–94.
[135] C. P. Ravikumar and S. Sastry, "Parallel placement on reduced array architecture," in *Proc. 25th DAC*, 1988, pp. 121–127.
[136] Y. Won and S. Sahni, "Maze routing on a hypercube multiprocessor computer," in *Proc. Int. Conf. on Parallel Processing*, 1987, pp. 630–637.
[137] O. A. Olukoton and T. N. Mudge, "A preliminary investigation into parallel routing on a hypercube computer," in *Proc. 24th DAC*, 1987, pp. 814–819.
[138] A. Iosupovicz, "Design of an iterative array maze router," in *Proc. ICCC*, 1980, pp. 908–911.
[139] M. A. Breuer and K. Shasma, "A hardware router," *J. Digital Syst.*, vol. 4, no. 4, pp. 393–408, 1981.
[140] S. J. Hong, R. Nair, and E. Shapiro, "A physical design machine," in *Proc. VLSI'81*, 1981, pp. 257–266.
[141] R. A. Rutenbar and D. E. Atkins, "Systolic routing hardware: Performance evaluation and optimization," *IEEE Trans. Comput. Aided Design*, vol. CAD-7, no. 3, pp. 397–409, 1988.
[142] T. Watanabe, H. Kitazawa, and Y. Sugiyama, "A parallel adaptable routing algorithm and its implementation on a two-dimensional array processor," *IEEE Trans. Comput. Aided Design*, vol. CAD-6, no. 2, pp. 241–250, 1987.
[143] T. D. Spiers and D. A. Edwards, "A high performance routing engine," in *Proc. 24th DAC*, 1987, pp. 793–799.
[144] Y. Won, S. Sahni, and Y. El-Zig, "A hardware accelerator for maze routing," in *Proc. 24th DAC*, 1987, pp. 800–806.
[145] T. Ryan and J. E. Rodgers, "An Isma Lee router accelerator," *IEEE Design Test Comput.*, vol. 4, no. 5, pp. 38–45, 1987.
[146] K. Kuwayama, S. Tsukiyama, and I. Shirakawa, "A design rule checker for VLSI on a multiprocessor," (in Japanese) IEICE monograph, CAS 87-209, pp. 7–11, 1988.
[147] E. C. Carlson and R. A. Rutenbar, "Mask verification on the connection machine," in *Proc. 25th DAC*, 1988, pp. 134–140.
[148] E. C. Carlson and R. A. Rutenbar, "A scanline data structure processor for VSLI geometry checking," *IEEE Trans. Comput. Aided Design*, vol. CAD-6, no. 5, pp. 780–794, 1987.
[149] R. Kane and S. Sahni, "A systolic design-rule checker," *IEEE Trans. Comput. Aided Design*, vol. CAD-6, no. 1, pp. 22–32, 1987.
[150] C. C. Jong, A. F. Pearmain, and P. R. Coward, "Dapmac: Design-rule checking using DAP," in *Proc. IEEE COMPEURO87*, 1987, pp. 422–425.
[151] K. S. Eo, S. S. Kim, and C. M. Kyung, "Ringtree: A VLSI architecture for fast image generation and processing," in *Proc. ISCAS*, 1988, pp. 801–804.
[152] K. Suzuki, T. Ohtsuki, and M. Sato, "A gridless router: Software and hardware implementations," in *Proc. VLSI'87*, 1987, pp. 153–163.
[153] J. Soukup, "Fast maze router," in *Proc. 15th DAC*, 1978, pp. 100–102.
[154] T. Ogura *et al.*, "A 4-kbit associative memory LSI," *IEEE J. Solid-State Circuits*, vol. SC-20, pp. 1277–1282, 1985.
[155] T. Ogura *et al.*, "A 20-kb CMOS associative memory LSI for artificial intelligence machines," in *Proc. ICCD'86*, 1986, pp. 574–577.
[156] A. E. Dunlop, V. D. Agrawal, D. N. Deutsch, M. F. Jukl, P. Kozak, and M. Wiesel, "Chip layout optimization using critical path weighting," in *Proc. 21st Design Automation Conf.*, 1984, pp. 133–136.
[157] M. Burstein and M. Youssef, "Timing influenced layout design," in *Proc. 22nd Design Automation Conf.*, 1985, pp. 124–130.
[158] M. A. B. Jackson and E. S. Kuh, "Performance-driven placement of cell based IC's," in *Proc. 26th Design Automation Conf.*, 1989.
[159] S. Chowdhury and M. A. Breuer, "Optimum design of IC power/ground nets subject to reliability constraints," *IEEE Trans. Comput. Aided Design*, vol. CAD-7, no. 7, pp. 787–796, July 1988.
[160] S. Chowdhury, "An automated design of minimum area IC power/ground nets," in *Proc. 24th Design Automation Conf.*, 1987, pp. 223–229.
[161] M. Marek-Sadowska, "Pad assignment for power nets in VLSI circuits," *IEEE Trans. Comput. Aided Design*, vol. CAD-6, no. 4, pp. 550–560, July 1987.
[162] R. Dutta and M. Marek-Sadowska, "Automatic sizing of power/ground (P/G) networks in VLSI," in *Proc. 26th Design Automation Conf.*, 1989.
[163] C. W. Ho, "High-performance computer packaging and the thin-film multichip module," in *VLSI Electronics Microstructure Science*, Vol. 5. New York: Academic Press, 1982, chap. 3, pp. 103–143.
[164] "The pacing technology," *Vectors*, vol. 30, no. 4, pp. 2–7, 1988.
[165] R. Smolley, "Connectorless high speed interconnect," presented at Government Microcircuit Applications Conf., Orlando, FL, Oct. 1987.
[166] Y. Akasaka, "Three-dimensional IC trends," *Proc. IEEE*, vol. 74, no. 12, pp. 1703–1714, Dec. 1986.
[167] F. T. Leighton and A. L. Rosenburg, "Three-dimensional circuit layout," *SIAM J. Comput.*, vol. 15, pp. 793–813, 1986.
[168] Y. K. Chen and M. L. Liu, "Three-layer channel routing," *IEEE*

Trans. Comput. Aided Design, vol. CAD-3, no. 2, pp. 156–163, Apr. 1984.
[169] D. Braun, J. L. Burns, F. Romeo, A. Sangiovanni-Vincentelli, K. Mayaram, S. Devades, and H-K. T. Ma, "Techniques for multilayer channel routing," *IEEE Trans. Comput. Aided Design*, vol. CAD-7, no. 6, pp. 698–712, June 1988.
[170] D. J. Garrod, R. A. Rutenbar, and R. L. Carley, "Automatic layout of custom analog cells in ANAGRAM," in *Digest of Tech. Papers, Int. Conf. on Computer-Aided Design*, 1988, pp. 544–547.
[171] H. Y. Koh, C. H. Sequin, and P. R. Gray, "Automatic layout generation for CMOS operational amplifiers," in *Digest of Tech. Papers, Int. Conf. on Computer-Aided Design*, 1988, pp. 548–551.
[172] A. Fujimura, "Automating the layout of very large gate arrays," *VLSI Systems Design*, vol. 9, no. 4, pp. 22–27, Apr. 1988.
[173] M. Igusa, M. Beardslee, and A. Sangiovanni-Vincentelli, "ORCA: A sea-of-gates place and route system," in *Proc. 26th Design Automation Conf.*, 1989, pp. 122–127.
[174] S. Sastry and Jen-I Pi, "An investigation into statistical properties of partitioning and floorplanning problems," in *Proc. 26th Design Automation Conf.*, 1989, pp. 382–387.
[175] C. Sechen, "Average interconnection length estimation of random and optimized placements," in *Digest of Technical Papers, Int. Conf. on Computer-Aided Design*, 1987, pp. 190–193.
[176] A. Casotto, F. Romeo, and A. Sangiovanni-Vincentelli, "A parallel simulated annealing algorithm for the placement of macrocells," *IEEE Trans. Computer-Aided Design*, vol. CAD-6, no. 5, pp. 838–847, Sept. 1987.
[177] D. Jepsen and D. Gelott, "Macro placement by Monte Carlo annealing," in *Proc. Int. Conf. on Computer-Aided Design*, Nov. 1983, pp. 495–498.
[178] D. F. Wong, H. W. Leong, and C. L. Liu, *Simulated Annealing for VLSI Design*. Boston: Kluwer Academic, 1988.
[179] C. Sechen, "Chip-planning, placement, and global routing of macro/custom cell integrated circuits using simulated annealing," in *Proc. 25th Design Automation Conf.*, 1988, pp. 73–80.
[180] H. Kawanishi, S. Goto, T. Oyamada, and K. Kani, "A routing method of building-block LSI," in *Rec. 7th Asilomar Conf. on Circuits, Systems, and Computers*, 1973, pp. 119–123.
[181] R. Linsker, "An iterative-improvement penalty-function driven wire routing system," *IBM J. Res. Develop.*, vol. 28, no. 5, pp. 613–624, 1984.
[182] T. Blank *et al.*, "Point accelerators versus general-purpose machines for electronic CAD," in (D&T Roundtable), *IEEE Design Test Comput.*, pp. 46–51, Oct. 1987.
[183] H. S. Baird, "Fast algorithm for LSI artwork analysis," in *Proc. 14th DAC*, 1977, pp. 303–311.
[184] J. K. Ousterhout, "Corner stitching: A data-structuring technique for VLSI layout tools," *IEEE Trans. Computer-Aided Design*, vol. CAD-3, no. 1, pp. 87–100, 1984.
[185] S. A. Kravitz and R. A. Rutenbar, "Multiprocessor-based placement by simulated annealing," in *Proc. 23rd DAC*, 1986, pp. 567–573.
[186] K. Vorm, "A production VLSI design CAD system," in *Proc. ICCD'83*, 1983, pp. 476–479.

Ernest S. Kuh (Fellow, IEEE) received the B.S. degree from the University of Michigan, Ann Arbor, in 1949; the S.M. degree from the Massachusetts Institute of Technology, Cambridge, in 1950; and the Ph.D. degree from Stanford University, Stanford, CA, in 1952.

He is a Professor of electrical engineering at the University of California, Berkeley. He joined the Electrical Engineering and Computer Sciences Department faculty in 1956. From 1968 to 1972 he served as Chairman of the department; from 1973 to 1980 he served as Dean of the College of Engineering. From 1952 to 1956 he was a member of the Technical Staff at Bell Telephone Laboratories in Murray Hill, NJ.



Tatsuo Ohtsuki (Fellow, IEEE) received the B.E., M.E., and Dr.Eng. degrees from Waseda University, Tokyo, Japan, in 1963, 1965, and 1970, respectively, all in electrical engineering.

In 1965, he joined the Nippon Electric Company, Ltd., Tokyo, Japan. From 1978 to 1980, he served as Research Manager, Application System Research Laboratory, at Central Research Laboratories. In 1980 he left Nippon Electric Company and joined Waseda University, where he is presently a Professor and Chair of the Department of Electronics and Communication Engineering.