MOSAICO: AN INTEGRATED MACRO-CELL LAYOUT SYSTEM

by

J. Burns, A. Casotto, M. Igusa, F. Marron
F. Romeo, A. Sangiovanni-Vincentelli, C. Sechen,
H. Shin, G. Srinath, and H. Yaghutiel

Memorandum No. UCB/ERL M87/7

15 January 1987

MOSAICO: AN INTEGRATED MACRO-CELL LAYOUT SYSTEM

by

J. Burns, A. Casotto, M. Igusa, F. Marron,
F. Romeo, A. Sangiovanni-Vincentelli, C. Sechen,
H. Shin, G. Srinath, and H. Yaghutiel

Memorandum No. UCB/ERL M87/7

15 January 1987

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

Mosaico: An Integrated Macro-Cell Layout System

J.Burns, A.Casotto, M.Igusa,
F.Marron, F.Romeo, A.Sangiovanni-Vincentelli,
C.Sechen, H.Shin, G.Srinath ‡ , H.Yaghutiel,

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

ABSTRACT

In this paper a new, integrated, macro-cell layout system called Mosaico is presented. Mosaico implements a complete layout pipeline from high-level description to final layout. Well-tested tools like the channel routers Yacr and Chameleon are used together with recently-developed tools for power and ground routing, channel definition and ordering, and floorplanning and placement. The system handles macro-cells of any rectilinear shape and efficiently uses all the interconnect layers offered by the technology. Cells with floating pins and variable aspect ratios can be accomodated. Unlike other layout systems, no rectilinear slicing-structure placement is required; this is due to the new, generalized channel definition and ordering algorithm. Every tool in Mosaico runs from and generates symbolic-layout views of the design. A spacing program takes the results after detailed routing in symbolic form and produces mask geometries, while guaranteeing that the design rules are satisfied. The Oct data manager is used to store the design at each stage of the layout process. The amount of data stored at each stage is stated in a set of policies that are respected by all tools, making the system modular and extensible. Mosaico is tightly coupled with synthesis and verification tools in the Berkeley design environment.

Categories: 3, 4, 10

---

‡ G.Srinath was a visiting Industrial Fellow from AMD when this work was carried out. He is presently with Daisy Systems Corporation.

This research was supported by Semiconductor Research Corp. Grant SRC-82-11-008.

1. Introduction.

A critical component of an efficient IC synthesis system is a set of optimized placement and routing tools. Some of the early silicon compilers [1], [2] had a fixed floorplan resulting in serious inefficiencies with respect to the silicon area occupied by the design. Recently attention has focused on layout systems tightly coupled to logic synthesis [3]. In this paper Mosaico, a complete set of placement and routing tools tightly coupled with logic synthesis, module generation, compaction, verification, and timing analysis, is described.

One of the main requirements for a layout system in a general design environment such as the one being developed at Berkeley [4] is the ability to support a variety of layout styles. The ThunderBird system [5] has previously been developed to place and route standard-cell designs. The Mosaico system consists of a set of tools for the placement and routing of macro-cells, a design style that is rapidly gaining importance in IC design. Other tools have been developed over the years to place and route macro-cells, both in industry, e.g. the systems described in [6], and in universities, e.g. the PI system [7] and the BBL system [8]. Mosaico differs from other systems in several ways, the most important difference being that no constraints are imposed on the layout style used to implement the macro-cells. Furthermore Mosaico is tightly coupled to Berkeley's logic synthesis system [4] through the use of module generators. That is, each macro-cell is optimized at the logic level with the characteristics of the target implementation style accounted for. The Mosaico floorplanner is then used to produce specifications for the aspect ratio and pin positions of the macro-cell. The combination of the optimized logic equations and the geometric information from the floorplanner comprise the input to the module generator; the generator output is then used in the placement phase of Mosaico.

In Mosaico the floorplanning and the placement (F&P) tasks are considered in an unified framework; in fact the same program is used for both tasks. The process starts as floorplanning with some or all of the cells not fully specified, and becomes placement as

soon as all the macro-cells have fixed implementations. The tool used for F&P is based on simulated annealing. Simulated annealing has proven to be a robust optimization technique that consistently produces results with high area efficiency. In addition its flexibility makes it attractive for applications such as the macro-cell problem where there are many degrees of freedom.

The routing part of the system targets the fabrication technologies now emerging that provide multiple interconnect layers. To achieve design-rule independence the entire Mosaico system operates at the symbolic-layout level, rather than the physical-layout level. A spacing step is carried out prior to mask generation to ensure that the layout is design-rule correct.

Mosaico has been configured in a highly modular manner to ease the introduction of new tools into the system. The modularity has been achieved by using one data representation for the design, regardless of the point in the design cycle. No transformation between representation formats is required, and all data is stored using a common data manager.

The remainder of the paper is organized as follows. In section one the overall structure of the system is presented. In the following sections, the tools that comprise the system are described in the order in which they appear in the design flow. Results obtained on a number of test cases are then presented, followed by some of the directions for future development of Mosaico.

2. Mosaico Overview

An overview of the structure of Mosaico, the management of the design data, and the initialization of the system are given in this section.

2.1. Pipeline Structure

Mosaico consists of five main steps:

1. Floorplanning and Placement
2. Channel Definition
3. Global Routing
4. Detailed Routing
5. Spacing

The five basic steps are nominally executed in sequence, that is, as a *pipeline*. The pipeline is represented in Figure 1 together with the sequence of symbolic views that are stored at each step in the Oct data manager [9]. Each step is described in detail in the following sections.

In an ideal situation each step in the pipeline would be executed once only. In reality iteration of some or all of the steps is usually required. For example, if the routing area estimated in the placement phase is not sufficient, at least one of the macro-cells must be moved. This in turn may require that the unrouted areas be re-processed by the channel definition procedure. A description of the feedback loops in the Mosaico system is included later in this paper.

2.2. Data Management

Figure 1 illustrates one of the most important feature of Mosaico system, namely its modularity. Modularity is achieved by enforcing the rule that all the information produced at each stage of the pipeline must be stored in the data manager. Each of the tools in the pipeline reads its input data from a view in the data manager and produces as its output another view which contains the previous data updated with the information added by the tool. This modularity provides the Mosaico system with a great deal of flexibility since every tool can be replaced in the pipeline with another one that requires the same data as input and provide the same data as output.

The key issue in achieving this flexibility is to control the structure of and the amount of data stored at each stage of the pipeline. The set of data has to be rich enough to provide the tool with the information it needs while being general enought to allow interfacing different tools based on different of algorithms. This task is made easy by the characteristics of the design data manager Oct. Oct provides the user with a general way to store the data items and the dependency relations among them. The tools read and write Oct views using a library of procedures. As a result, the tools need not be concerned with the storage format used by Oct, nor are they affected by any changes to the internal structure of Oct.

The particular data items used, and the relations among them, are not part of Oct; rather, they are decided upon by the community of tool-makers. An example of such a decision is the manner used to describe the permutability of terminals. This set of decisions is referred to as the *policy* for the design. Since the policy is not part of Oct it can be changed easily; and since all concerned parties participate in the definition of the policy it provides them with an efficient protocol for storing and exchanging information at each stage of the pipeline.

Another important advantage of storing all intermediate results in the data manager is that the pipeline can be restarted at every level. Moreover all the intermediate results can be graphically displayed by Vem, a general-purpose graphics editor for Oct views [9]. Vem can be used to view the design at any point, from the floorplan stage down to the complete design after detailed routing and spacing. Finally, tools that complete more than one of steps outlined in Figure 1 can easily be inserted in the pipeline at the expense of loosing the ease with which all the steps of the layout procedure can be unbundled and re-executed.

2.3. Pipeline Initialization

The starting point for the design is a high-level description of the chip expressed in the Bdsyn language [3]. The designer determines the partitioning of the design into

macro-cells by means of this description. The initial set of data required for the Mosaico system consists of a set of instances of macro-cells connected by a net-list. The Bdsyn description is used to produce automatically the net-list and the instances, both of which are placed into a symbolic view of the chip named *unplanned*.

The implementations of the macro-cells are generated separately, either by an automatic module generator or by manual design. When the Mosaico pipeline is started not all of the macro-cells need to be completely specified. For some macro-cells the complete layout may be known while for others only estimations of their parameters or a set of constraints may exist. For example, some of the pins may have to be positioned on specific sides of the cell, or a target aspect ratio may be desired even if a range of aspect ratios is feasible.

Since several representations of each cell may be available, Mosaico searches the available views in a specific order so that view with the most complete information for the floorplanning phase is used. If none of the views is present an artificial view is created in which the macro-cell is represented as a "*soft cell*", i.e., as a cell for which only estimations of its area and of its aspect ratio are available . For such a cell the terminals are left free to move around the boundary of the cell unless ordering constraint are present. Such a description for macro-cells is suitable for the floorplanning step.

3. Floorplanning and Placement

In the Mosaico environment, the activity of floorplanning and placement (F&P) is handled in a unified framework: in fact there is no distinction between floorplanning and placement at the algorithmic level. At the floorplanning stage some of the parameters (aspect ratio, pin positions) of the macro-cells may be varied; at the placement level all the macro-cell parameters are fixed and the only remaining degrees of freedom are the positions and orientations of the cells. The F&P maintains a global view of the chip and directs the interactions with module generation and timing verification.

The F&P iterations are started by an initial floorplanning operation in which the most detailed description available for each cell is retrieved from the data manager. In this initial representation most of the cells are described only in terms of their input/output signals and estimated area. Some cells may be described at the logical or behavioral level. For such cells an area estimation is performed, based on the complexity of the cell and on the features of the module generator that will be used to produce it.

After each iteration the floorplanner produces as output the position and orientation of each cell and also the shape and pin positions for those cells that were not completely specified. This information is passed to the module generators, namely Gem [10], Wolfe [11], and Topogen [12]. The generators differ in the layout styles they implement. Gem produces macro-cells using a gate matrix approach featuring multiple row and column folding. Wolfe generates macro-cells by assembling standard cells using the TimberWolfSC package [13] combined with the channel routers Yacr and Chameleon. Topogen produces individual cells in the standard-cell style: these cells may be combined into larger cells using Wolfe.

The input to the module generators comes from two sources, one being the floorplanner which specifies geometric constraints as described above. The other data is the logic description produced by the logic optimization tool Mis [3]. If a module generator is unable to satisfy all the constraints imposed on a particular cell, the actual shape of the generated cell is fed back to the F&P and a new iteration is started.

The F&P also interacts with the timing analyzer Hummingbird [14]. The job of the timing analyzer is to critique the floorplan from the point of view of the timing. It performs two tasks: first it checks the arrival times of signals required for the correct functioning of the circuit. Then it marks each net with information about the maximum and minimum delay through that net. If the requirements are not met a new iterations of the floorplanner is necessary. The timing information is used to update the constraints imposed by the F&P on the maximum or minimum length for the specific nets. Correction of the

violations of the timing requirements may require more than just a new placement. In fact for some of the modules a transistor re-sizing may be necessary or some of the modules may need to be re-synthesized with the new delay targets in mind. In both these cases the intervention of the designer is presently required.

As the F&P task is repeatedly executed more and more refined representations of the chip are generated. The macro-cells gain their physical implementation. the information available to the F&P tool increases. and the floorplan task turns more into a placement task.

3.1. F&P Algorithm

The tool presently used for F&P is the TimberWolfMC package [15]. The program is based on the simulated annealing algorithm which provides the package the flexibility that is needed in order to use one tool for both floorplanning and placement.

A list of basic features includes:

- Cells may be represented by any Manhattan polygon
- Cells are allowed to have aspect ratios that vary over a continuous or discrete range
- Cells can have multiple implementations and the most suitable will be selected
- Cells may have variable pin positions
- Weights can be assigned to each net to bias the placement

During F&P the total estimated interconnect length is minimized. and a penalty function approach is used to drive the total amount of cell overlapping toward zero at the end of the annealing procedure. The length of each net is estimated using the half-perimeter of the bounding box of the pins connected to the net: the calculation is based on the exact pin locations. A dynamic algorithm [15] is used to estimate the routing area necessary around each cell to complete the routing. The use of this estimation algorithm has resulted in the generation of placements which require very little modification during detailed routing.

After the placement is completed TimberWolfMC performs a placement refinement based on a more accurate estimation of the routing area required. The estimation is

slices must have bends. no constraints need to be imposed to the placement. The lack of constraints is particularly important in Mosaico since the simulated-annealing-based F&P produces structures that are more general than the rectilinear slicing structure.

In Mosaico the slices are allowed to have bends. The configuration of the slice determines the type of its corresponding detailed routing problem. For example in the simple case where the slice has no bends. the problem is an ordinary channel-routing problem with two fixed sides (the top and bottom) and two open sides (the left and right). A single-bend slice produces an "L-shaped" channel.

It was proved in [17] that if only rectangular cells are present. an L-shaped slice is the most complex shape that can result from the slicing algorithm. However this result does not hold if the cells are general Manhattan shapes. In this case *k-bend* slices (k=0,1,2,3,...) are necessary to guarantee a feasible slicing of the placement for any combination of cells. In Figure 2 an example of a two-bend slice is presented.

The generalized slicing algorithm inherits all the properties of the ordinary slicing strategy and handles a larger class of problems. As in the simple case no switch-box router is necessary since all the generalized channels have floating terminals on two open sides.

4.1. Channel Definition Algorithm

A divide-and-conquer strategy similar to the one necessary to find a rectilinear slicing structure is used [16]. The differences compared to the standard algorithm are in the method used to determine the next slice. The Atlas algorithm consists of the following steps:

- Using a scan-line approach generate a floorplan graph [17]
- Repeat the slicing procedure until the chip is completely subdivided: i.e.. at each step find the minimal-cost slice through the graph.
- The routing order is LIFO (i.e.. The last slice found is the first to be routed)

The *cost* of a slice is determined by the number of bends in the slice, the number of orthogonal edges on the slice, and the number of external junctions [17] that are created by the slice. Note that the selection of a path with the smallest number of jogs and the maximum number of orthogonal edges to the path will eliminate as many potential jogs in future slices as possible. This is a greedy approach to find the set of slices with the smallest total number of bends. Presently a more sophisticated algorithm that should prodece a nearly "optimal" slicing is under development.

The hierarchy introduced by the slicing procedure corresponds to the binary tree that is used to store the data in Oct. This routing order is used in the router server described below, and the new Oct view is the input to the next step in the pipeline.

5. Global Router

After the channel definition and ordering step described in the previous section the data must be prepared for the global router. This operation calls for the construction of the channel graph. In the channel graph, nodes represent intersections between channels while edges represent channels or sections of them. Each edge in the graph is assigned a weight that represents the maximum number of tracks that can be accommodated in the channel. Once the channel graph is built all the connected terminals of the macro-cell instances are projected to the closest edge in the channel graph. The channel graph with the positions of the terminals on its edges and a net list is the information necessary for global routing. The weights on edges are interpreted as *capacity constraints*. It is important to note that the input to the global router is completely symbolic and therefore totally independent of the layout style.

When the global router finishes, each net consists of a sequence of subnets, each of which is assigned to one of the edges of the channel graph. When a net exits one channel to enter another one a *pseudo-terminal* is created.

Presently two different algorithms to perform the global route can be selected. The first algorithm is based on simulated annealing and it gives results that are marginally better at the expenses of a longer computation time. The second algorithm is faster and may be used interactively. The two algorithms are briefly described in what follows.

5.1. Simulated-Annealing-Based Global Router

The simulated-annealing global routing algorithm has been developed as part of the TimberWolfMC package [15]. The algorithm has the following features:

- No dependence on the routing order
- Multi-pin nets are handled in the same fashion as two-pin nets
- Electrically-equivalent pins are utilized to minimize the routing length

The algorithm has two basic stages. During the first stage it attempts to generate the M shortest routes for each net, a task which can readily be accomplished for two-pin nets [18]. For nets consisting of more than two pins, an algorithm has been developed which generalizes the approach in [18].

In the second stage of the algorithm selects a single route from the M alternatives for each net. Let n_i represent a net, where $i \in \{1, \cdots, N\}$ and where N is the number of nets. Furthermore, let n_i^k represent the k-th alternative route for net n_i, where $k \in \{1, \cdots, M\}$. A simulated annealing algorithm is then used to select alternative $n_i^{k_a}$ for each $i \in \{1, \cdots, N\}$ such that the total routing length is minimized subject to the channel-edge capacity constraints. This approach enables the global router to avoid the routing-order dependence problem. Rip-up and re-route strategies are never needed to complete or improve the global routing.

5.2. N-Layer Global Router

Nlagr is a global router that can handle n layers of interconnect and over-the-cell routing. Over-the-cell routes are represented by fixed-capacity edges on the graph while routing areas between cells are represented by edges whose capacities can be exceeded with a penalty. Nlagr has the following characteristics:

- The number of bends, and number of channels through which a net passes are minimized.
- User-specified critical nets are routed first. All other nets are routed based on (estimated) shortest-length first. The estimate used is one-half the perimeter of the minimum enclosing rectangle of the net.
- Nets can be weighted to prefer or avoid specific layers.

There are two different algorithms incorporated in Nlagr. The first uses an extended shortest-path algorithm; the extension accounts for multi-terminal nets. At each iteration a path is determined from the existing partial path to the nearest unconnected pin on the same net. The search for the new connection is performed by expanding first the nodes of the channel graph that are close to the existing path. The expansion proceeds until an unconnected pin is reached. In the expansion procedure no particular direction is privileged since no unconnected pin has been selected as the target. For this reason the method is referred to as *undirected search*. The results provided by the method depend on the choice of the first pin and several heuristics are provided to select it.

In the second algorithm, named *directed search*, the pins of the net are connected in a specified order. First a least-cost path is found from the first pin to the second pin. Next the least-cost *incremental* path is found to the third pin (i.e., the first pin among those not yet connected) and so on as in [19]. The order in which unconnected pins are processed is determined by sorting them according to a cost which consists of two terms. The first term is the cost of the path connecting the expanded nodes in the channel graph to the already-existing path. This cost is the same as that used in the first algorithm. The second term is an estimation of the distance between the unconnected pin and the closest expanded node. The presence of the second cost term makes the search proceed rapidly in the direction of the target. This algorithm is strongly dependent on the pin ordering. Several schemes are provided for ordering the pins prior to routing.

According to experimental results undirected search seems to be better than directed search, although slower. The dependency of directed search on pin ordering can be used to provide interactive optimization. In this mode of operation the global router is typically run first in the undirected-search mode. The order in which the pins were connected is

preserved and used to back-annotate the net list. Next the optional interactive mode is entered: the user may then modify the pin order and run the router in directed search mode.

6. Router Server

The router server Spider performs the detailed routing of the circuit based on the placement, the global router output, the channel order, and the design rules. Spider can handle k-bend routing regions with irregular edges, and wires with different widths, a crucial feature necessary to route special nets like power and ground along with the standard signal nets. Spider also chooses the best layer for floating pins whenever more than one layer is permitted.

The actual routing is performed one channel at a time. The order is determined by Atlas as described in Section 4 of this paper. Spider retrieves the channel order from the data manager by traversing the binary tree that represents the hierarchy in depth-first order. Then it selects one of the symbolic routers available according to the nature of the channel being processed. Presently the library of routers that can be used by Spider consists of three symbolic routers: the two-layers channel router Yacr2 [20], the multi-layer channel router Chameleon [21], and a general-area router called Mighty [22]. The selection of the particular router is based on a set of rules, such that the simplest router that can successfully route the area is selected. To make it easy to extend the tool library by adding new symbolic routers, the selection rules are kept separate from the core of the program by storing them in a file that can be modified by the user.

Once the symbolic router is selected, Spider prepares the input data in the suitable format. Since all of the routers in the Mosaico library are grid-based, Spider starts by defining a grid. The symbolic grid in the "vertical" direction (the columns) can be built in two different ways. The first choice is to use a uniform grid, where the spacing between grids is determined by the wire widths, the contact widths, and the spacing rules. The

second approach is a non-uniform grid. In this case the vertical grid lines are placed to obtain the best possible alignment with the actual locations of the pins. This method is especially useful in dealing with wires that differ in width. Regardless of the type of grid selected, it is not always possible for every pin to be located exactly on a grid line. In this situation an attempt is made to obtain a better alignment of the terminals on the two sides of the routing area by slightly varying the offset between the macro-cells on either side. After the alignment procedure each pin that is still not on a grid line will be connected to the nearest adjacent one by a jog.

In the "horizontal" direction, the grid lines (rows) are initially spaced uniformly. If the general-area router is used, the situation is a little more involved since an estimation of the space necessary to complete the routing must be computed. The spacing is definitively determined only after the routing is completed and the actual width of each wire has been computed. Presently, the non-uniform grid base is the default setting.

After the selected router completes, a post-processing step is performed to try to reduce the number of jogs by shifting wires while maintaining the design rules. A simple optimization step is then performed to minimize the number of vias.

When the routing of a particular region is completed, the compactor is invoked to space the region and the adjacent macro-cells according to the design rules. The combination of the cells and the routing is then considered to be a new, large macro-cell for the following steps in the layout process. A word of caution is in order about the use of the compactor in this channel-by-channel manner. By compacting one channel it is possible to generate a misalignment in the next channel to be routed, possibly increasing its density and possibly increasing the area of the chip. To avoid this problem a more global view of the routing problem is necessary, which is accomplished by looking at the next area to be routed and setting constraints on the compactor to avoid increasing the density in the next channel.

7. Power and Ground Routing

The routing of the power and ground nets is more involved than the routing of regular nets because of the following considerations:

- Voltage drops due to the finite conductance of interconnect and contacts have an adverse effect on the noise margins of the cells.
- The maximum current densities tolerated by an interconnect layer cannot be exceeded, otherwise metal migration might result.

The implications of the above are that the power and ground nets might have variable wire segment widths.

In a technology with only one layer of low-resistance interconnect (metal), a planar routing of the power and ground nets might be very desirable. However, in modern technologies two layers of metal are available. Therefore the policy used in Mosaico is to route power and ground using the existing global and channel routers. The exact procedure will be described later.

It is assumed that there are always power and ground rings around the chip. The existence of these rings is important for insuring correct functionality of the chip. If the power pads were not connected through a ring, some of them might be at different potentials due to wiring inductance: this in turn might result in latch-up problems in a CMOS design.

For the sake of the placement task, the part of the ring which passes under a pad is considered to be part of the pad cell itself. Therefore the location of the power and ground rings is automatically adjusted by movements of the pads. After the completion of the placement stage, the unconnected segments of the power and ground rings are tiled together with additional pieces of material of the same layer. Each pad cell has three pins, one for the pad itself and one for each of the power and ground connections. Thus the number of possible connections to each of the power and ground rings is equal to the total number of pads in the chip.

Treating the power and ground nets as regular ones imposes "false" constraints on the placement since power and ground are connected to every block of the chip. Furthermore since no assumption is made on the amount of current that can flow across a macro-cell without damaging the internal power (ground) connection it follows that each macro-cell must be connected directly to the rings.

The following steps describe the algorithm used to route power and ground.

1. Before the placement stage, decompose the power (ground) net into smaller nets. Each of these nets contains two sets of equivalent pins, one is the power (ground) pins of a given macro-cell, and the other is the set of all the power (ground) pins on the ring. Then the original power (ground) net is discarded, and the resulting nets are treated as regular ones with high priority in the placement and global-routing stages in order to keep their length as short as possible.
2. After the global routing stage, merge the power (ground) nets in such a manner that every channel contains at most two power (ground) nets. The upper limit of two happens when the two nets enter the channel from its opposite ends and both terminate inside the channel. An implication of this merging is that all the nets which stem from the same power (ground) pad pin are merged into a single net. This step is shown in Figure 3.
3. After the merging step, find the power requirements of the pseudo-pins of the power and ground nets at the boundaries of the channels.
4. After the symbolic detailed routing of each channel, find the subnet widths of power and ground nets in the routed channel. This step is shown in Figure 4.

8. Layout Spacing

As noted above, all steps in the Mosaico pipeline are carried out at the symbolic-layout level. In addition, the module generators used in the system generate symbolic layout as well. Symbolic layout spacing (or compaction) is thus an essential part of the system for several reasons. First, all designs must be spaced to ensure that they are design-rule correct; this has the advantageous effect of eliminating design-rule checking, since the layouts are correct-by-construction. Second, the use of generalized symbolic layout provides a mechanism for producing technology-independent designs. The spacing techniques used in Mosaico are capable of updating the symbolic layout primitives themselves (e.g., transistors, contacts), as well as the spacings between them. Also, by the use of a variety of spacing techniques various optimizations of the layout can be performed over a range of area/cpu-time tradeoffs. Presently two spacers are available in Mosaico; each is described below.

8.1. One-Dimensional Spacing

Sparcs [23] is a constraint-based spacer that, to meet the requirements of modern circuit design and fabrication technologies, has been designed to be as general as possible. Some of the features of Sparcs that help in achieving generality are:

- Full support of upper-bound and user-defined constraints
- Detection and identification of overconstrained elements
- Adjustable positioning of non-critical-path circuit elements
- Dependencies among constraints to enforce symmetry during spacing
- A hierarchy and technology-independent symbolic layout abstraction
- Selectable constraint modes, such as virtual grid and relative grid
- Terminal merging of arbitrary layout elements
- Automatic jog insertion

Sparcs consists of two major modules: the constraint graph builder and the constraint graph solver. The constraint graph is constructed from the relative placement of the symbolic layout elements and a table of spacing rules. The solver performs critical-path analysis, positions the elements that are not on the critical path, and identifies

overconstrained problems. Sparcs uses successive one-dimensional spacings in alternating directions. as do most other spacers and compactors.

The symbolic layout abstraction used throughout the system is appropriate for any level of the hierarchy: that is. any shape representable by Manhattan polygons with any number of terminals is allowed. Since Sparcs spaces designs represented in this manner. any layout from any level of the hierarchy can be spaced. It is not necessary to have one program for transistor-level layouts and another for cell-level layouts. Features such as terminal merging that work only on the transistor-level in other systems work for any level in the case of Sparcs.

8.1.1. Constraint Types and Representation

In addition to lower-bound constraints. upper-bound constraints are heavily used in Sparcs. For example two constraints. an upper-bound and a lower-bound. are used to describe terminals so that the wire segment connected to the terminal may slide along it. Fixed constraints (i.e.. an upper-bound and a lower-bound of equal value) are used in Sparcs to allow the user to space by either virtual or relative grid. Sparcs also supports a special type of constraint called an *active constraint*. which forces the relative spacing between one pair of nodes to be the same as the relative spacing between another. Active constraints are used to maintain symmetry among layout elements.

Two constraint graphs. $G_x(V, E)$ and $G_y(V, E)$. which represent the relationships among objects in the horizontal and vertical directions. respectively. are constructed. Each vertex $v_i \in V$ represents an element in the layout. and each directed edge $e_{ij} \in E$ represents a constraint between v_i and v_j.

8.1.2. Constraint Graph Solution

The constraint graph for a particular direction is analyzed to determine the locations of the layout elements in that dimension through a two-phase process. the first phase being a critical-path analysis. The elements that lie on the critical path determine the

8.2.2. Constraint-Graph Modification

Since the zone-refining process requires many local movements, constraint-graph modification dominates the cpu-time. Solving the longest path problem on the constraint graphs usually takes less than three percent of the total run-time, because only a small number of clustered components are moved in one operation.

As a result the current emphasis is on finding an efficient algorithm for updating the constraint graphs; this process is aided by a new data structure in which all the components are attached to cells in a coarse grid. With this grid approach, searching and sorting times are bounded by constants. Due to this new data structure, run-times have been reduced by factors of five to ten over those of the first implementation.

9. Pipeline Iteration

The two loops shown in Figure 1 are provided to account for situations in which the routing area available after placement is insufficient to complete the routing. This situation can occur both during global routing and during detailed routing. In both cases the extra routing area that is required can be provided by spacing apart the macro-cells delimiting the routing region. If the additional area can be provided without drastically changing the order and structure of the routing areas then the pipeline is started again from the point where the exception occurred. In this situation all of the work already completed is preserved.

If the addition of the extra area requires a change such that the organization of the routing area is no longer feasible then two possible solutions exist. In the first the channel definition procedure is re-executed to reorganize the areas not yet routed. The second is performed when the first solution fails; the pipeline is re-started at the placement stage using a more conservative estimate for the routing area requirements.

10. Interactive Capabilities

The Mosaico pipeline is operated by means of a script that execute the entire pipeline or subsections of it. The script-based approach provides the user with a great deal of flexibility: the script can be modified both in the sequence of the calls to the tools and in the parameters that are input to the tools, e.g., the parameters used in the simulated annealing procedures.

In addition to the tools described in the previous sections, a set of graphical interactive aids is provided to help the user judge the quality of the layout at any stage of the pipeline. One tool allows the user, for a selected macro-cell, to see a representation of its connections with the other cells drawn from cell center to cell center. Another aid allows the user to see the optimal position of a selected macro-cell as computed by a force-directed algorithm. Both of these programs may be used at any point following the placement step. A third tool uses the information available after the global router completes. It allows the user to select a terminal and highlight the path of the net connected to that terminal as determined by the global router. This aid is used to judge the quality of the placement combined with the channel definition and global routing, prior to the detailed routing.

These tools are closely linked to Vem [9] and are presented to the user as menu options. These aids together with the other Vem editing capabilities allow the user to modify existing views, quickly evaluate the effect of modifications on the quality of the layout, and determine whether or not to run the complete pipeline.

11. Applications and Results

The Mosaico system is presently under test on a set of macro-cell examples provided by industry. The examples, whose characteristics are summarized in Table 1, consist of macro-cells with fixed aspect ratios and fixed pin-positions. Table 2 contains the area statistics after placement, routing, and spacing. The cpu-time required to place and route

the examples is reported in Table 3 while the results are depicted in Figures 5, 6, and 7.

The interactions of Mosaico with the module generators and the timing analyzer is still in a preliminary test phase and significant results are not available at the present time.

12. Conclusions and Future Work

In this paper Mosaico, an integrated set of tools for macro-cell layout, has been presented. Mosaico is designed to interact closely with the other tools in the Berkeley design system. A major point of interaction occurs at the floorplanning stage, where Mosaico exchanges information with module generators and timing analyzers. All the layout procedures are carried out at the symbolic-layout level, which provides the system

Circuit	#Macro-Cells	#Pads	#Nets	#Pins	#Channels
ck1	8	5	29	58	17
ck2	12	39	262	691	54
ck3	23	17	129	458	43

Table 1. Characteristics of the test circuits.

Circuit	Placed	Routed	Compacted	Area savings (%) after compaction
ck1	1592x1415	1678x1562	1533x1433	16
ck2	17305x15620	17033x14982	15670x14982	1
ck3	2902x3607	4216x4080	n.a.	n.a.

Table 2. Area at various stages of the pipeline (in λ).

Circuit	Placement	Routing		Compaction
		Global	Detailed	
ck1	140	5	22	21
ck2	n.a	926	482	960
ck3	1203	74	140	n.a.

Table 3. CPU time (seconds, VAX-8650).

with technology independence. Furthermore Mosaico is closely coupled with the design manager Oct to achieve an open system with a great deal of modularity and flexibility.

In Mosaico well-tested tools like the detailed routers Yacr and Mighty cohabit with new tools that have been developed explicitly for the system, such as the channel definition and ordering procedure and the power and ground router.

Future development will occur on all steps in the pipeline. In particular the channel definition and ordering algorithm will be enhanced to better take into consideration the influence of the channel definition on the performance of the detailed routers. Also the interactions between the router server Spider and the spacers will be enhanced to provide for layouts in which particular spacing constraints between elements are present. In general the set of tools available will be enlarged to provide the user with the choice of several parallel paths for each stage in the pipeline.

References

1. D. Johannsen. "Bristle Blocks: A Silicon Compiler". *Proc. 16th Design Automation Conf.* 1979. 310-313.

2. J. R. Southard. "MacPitts: An Approach to Silicon Compilation". *Computers*. Dec. 1983.

3. R. Brayton. A. Cagnola. E. Detjens. S. Krishna. P. McGeer. L. Pei. N. Phillips. R. Rudell. R. Segal. T. Villa. A. Wang. R. Yung and A. Sangiovanni-Vincentelli. "Multiple-Level Logic Optimization System". *Proc. IEEE Inter. Conf. on Computer Aided Design*. Santa Clara . Nov 1986. 356-359.

4. A. R. Newton. A. Sangiovanni-Vincentelli and C. H. Sequin. "The Berkeley Synthesis Project". *ee299-cs292 Final Report*. Berkeley. May. 1986.

5. D. Braun. C. Sechen and A. Sangiovanni-Vincentelli. "ThunderBird: A Complete Standard-Cell Layout System". *Proc. 1986 Custom Integrated Circuits Conference*. Rochester. New York. May 12-14. 1986.

6. T. Sudo. T. Ohtsuki and S. Goto. "CAD Systems for VLSI in Japan". *Proceedings of the IEEE 71*. 1 (Jan. 1983).

7. R. Rivest and C. Fiduccia. "The 'PI' (Placement and Interconnect)-System". *Proc. 19th Design Automation Conference*. 1982. 475-481.

8. N. P. Chen. C. P. Hsu. E. S. Kuh. C. C. Chen and M. Takahashi. "BBL: A Building-Block Layout System for Custom Chip IC Design". *Proc. 1983 IEEE International Conf. on Computer-Aided Design*. Santa Clara. Calif. 1983. 40-41.

9. D. Harrison. P. Moore. R. L. Spickelmier and R. Newton. "Data Management and Graphics Editing in the Berkeley Design Environment". *Proc. IEEE Inter. Conf. on Computer Aided Design*. Santa Clara. Nov 1986.

10. S. Devadas. "Synthesis of Logic Networks in Silicon". *Masters Report*. Berkeley. Dec. 1986.

11. R. Rudell. "Wolfe: A Standard-cell Based Module Generator". *ee299-cs292 Final Report*. Berkeley. May. 1986.

12. G. Adams. S. Devadas. K. Eberhard. C. Kring. F. Obermeier. P. Tzeng. R. Newton. A. Sangiovanni-Vincentelli and C. Sequin. "Module Generation Systems". *ee290 Final Report*. Berkeley. May 1986.

13. C. Sechen and A. Sangiovanni-Vincentelli. "TimberWolf3. 2: A New Standard Cell Placement and Global Routing Package". *Proc. 1986 Design Automation Conference*. Las Vegas. Nevada. June 29 - July 2. 1986. 432-439.

14. N. Weiner. "Aspects of Pre-Layout Timing Prediction and Rectification". *ee299-cs292 Final Report*. Berkeley. May. 1986.

15. C. Sechen. "Placement and Global Routing of Integrated Circuits Using Simulated Annealing". *Ph.D. Thesis*. Berkeley. CA. 1986.

16. R. H. Otten. "Automatic Floorplan Design". *Proc. 19th Design Automation Conf.* 1982. 261-267.

17. W. M. Dai. T. Asano and E. S. Kuh. "Routing Region Definition and Ordering Scheme for Building-Block Layout". *IEEE Trans. Computer-Aided Design vol. CAD-4* (July 1985). 189-197.

18. E. Lawler. *Combinatorial Optimization: Networks and Matroids*. 1976. 102-104.

19. G. T. Hamachi. "An Obstacle-Avoiding Router for Custom VLSI". *Ph.D. Thesis*. 1986.

20. J. Reed. A. Sangiovanni-Vincentelli and A. Santomauro. "A New Symbolic Channel Router: YACR2". *IEEE Trans. on CAD 4* (July 1985). 208.

21. D. Braun. J. Burns. S. Devadas. H. K. Ma. K. Mayaram. F. Romeo and A. Sangiovanni-Vincentelli. "Chameleon: A New Multi-Layer Channel Router". *Proc. 1986 Design Autom. Conf.* Las Vegas. June. 1986.

22. H. Shyn and A. Sangiovanni-Vincentelli. "MIGHTY: A 'Rip-up and Reroute' Detailed Router". *Proc. IEEE Inter. Conf. on Computer Aided Design*. Santa Clara. Nov 1986. 2-5.

23. J. L. Burns and A. R. Newton. "SPARCS: A New Constraint-Based IC Symbolic Layout Spacer". *Proc. IEEE 1986 Custom Integrated Circuits Conference*. May 1986. 534-539.

24. H. Shin. A. Sangiovanni-Vincentelli and C. Sequin. "Two-dimensional Compaction by 'Zone Refining'". *Proceedings of 23rd Design Automation Conference*. June 1986. 115 - 122.

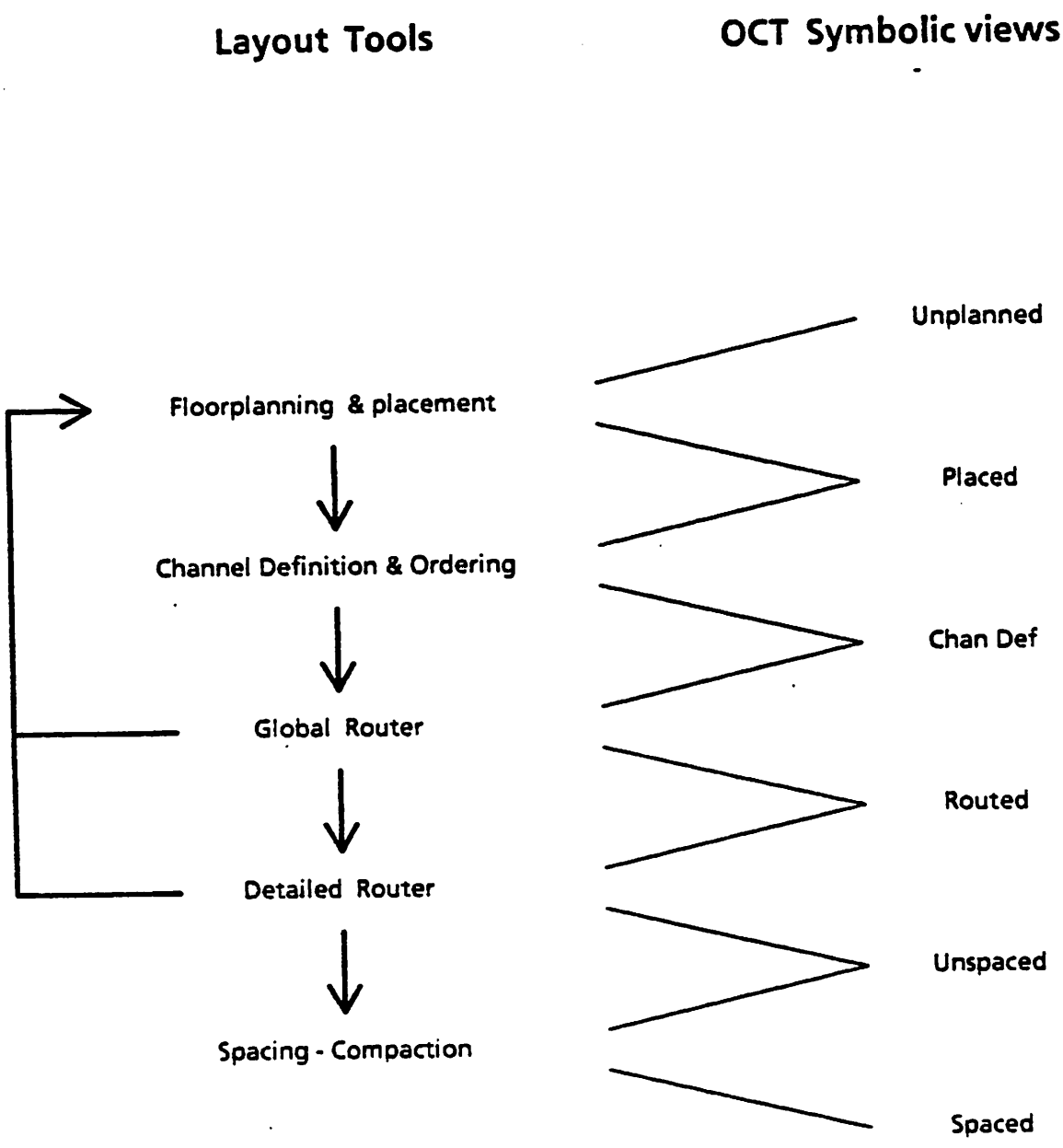

Figure 1. The Layout System and the correspondent OCT views.

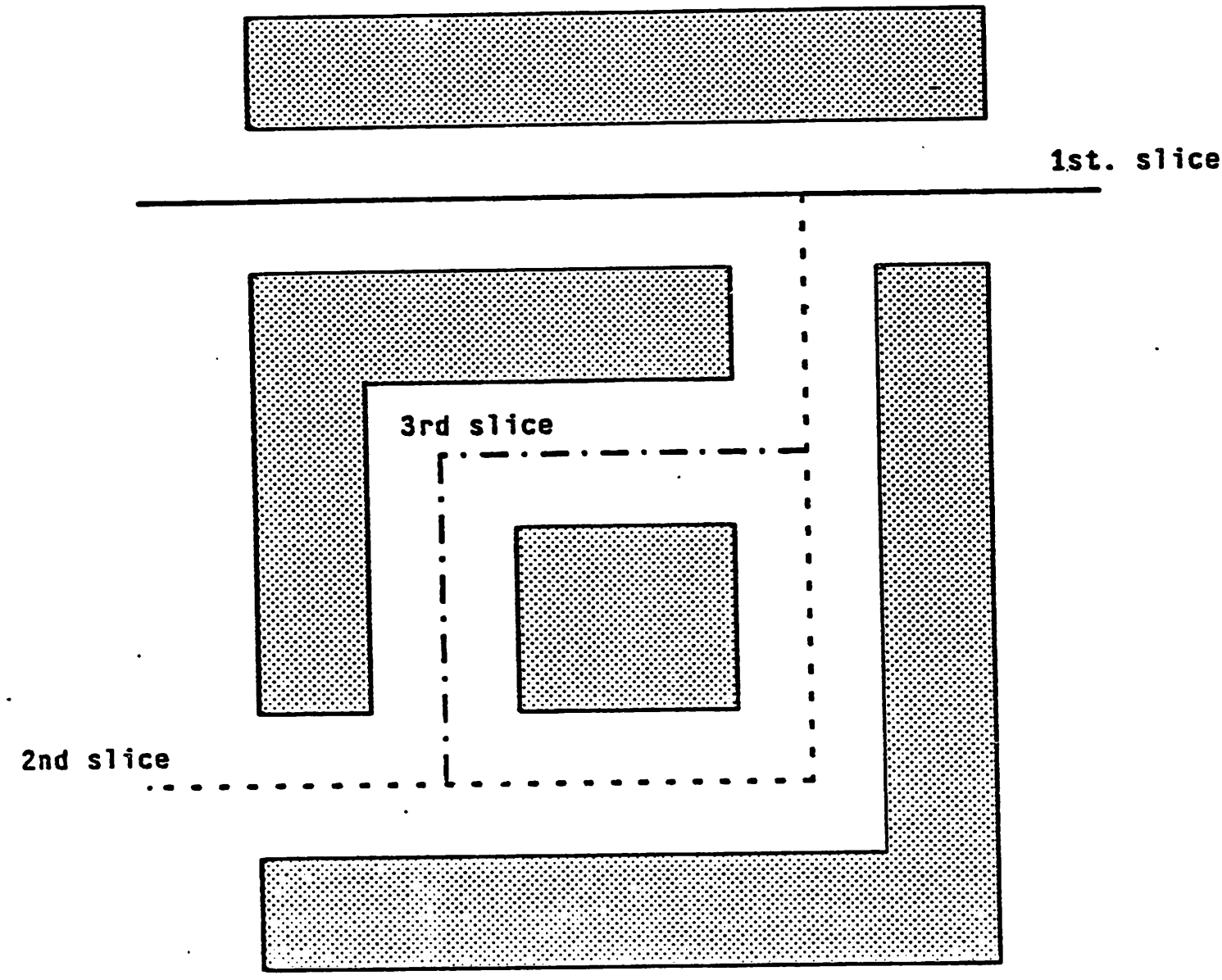

Figure 2. Example of zero-bends and one-bend slices.

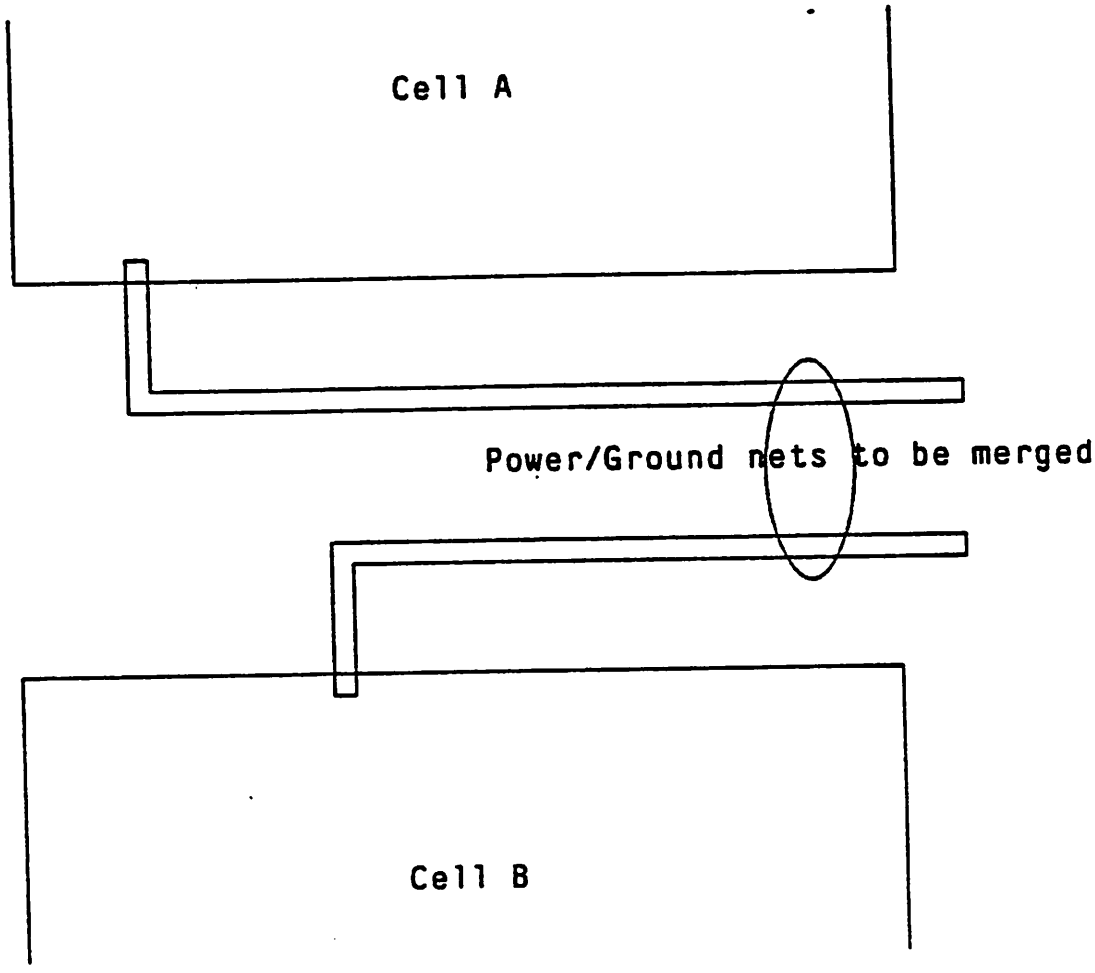

Figure 3. Merging of Power/Ground nets in the channel.

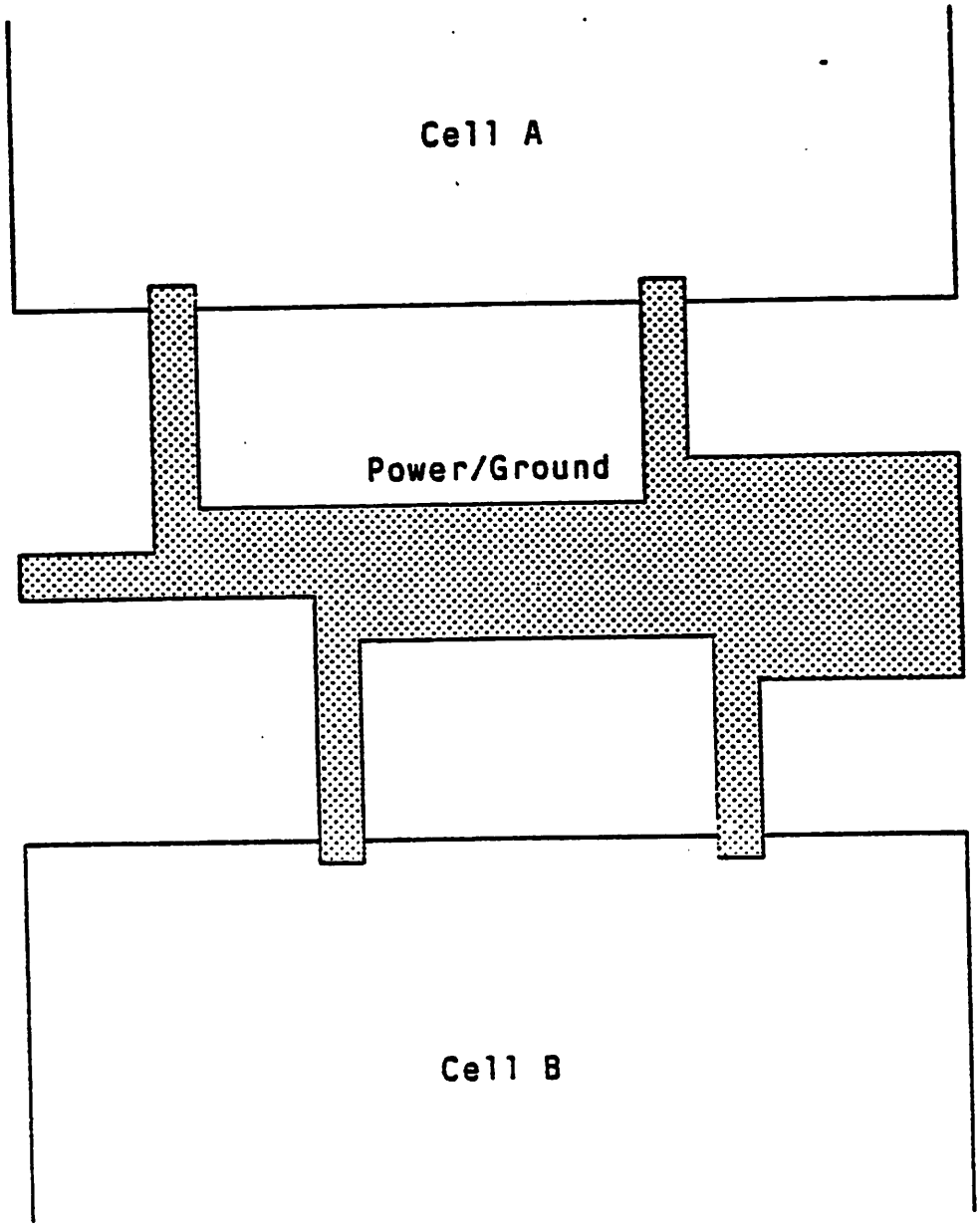

Figure 4. Power/Ground sizing.

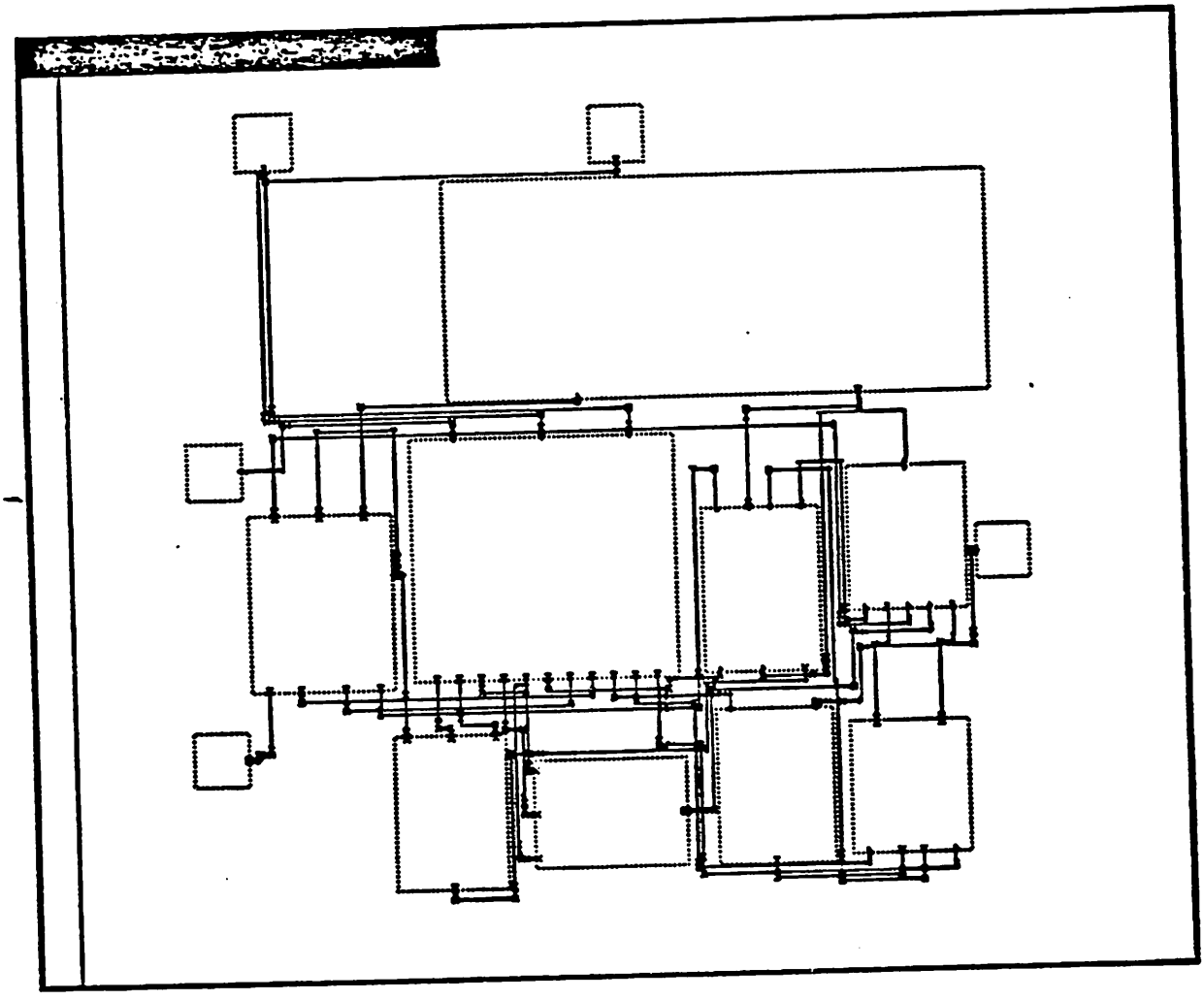

Figure 5. Circuit Ck1

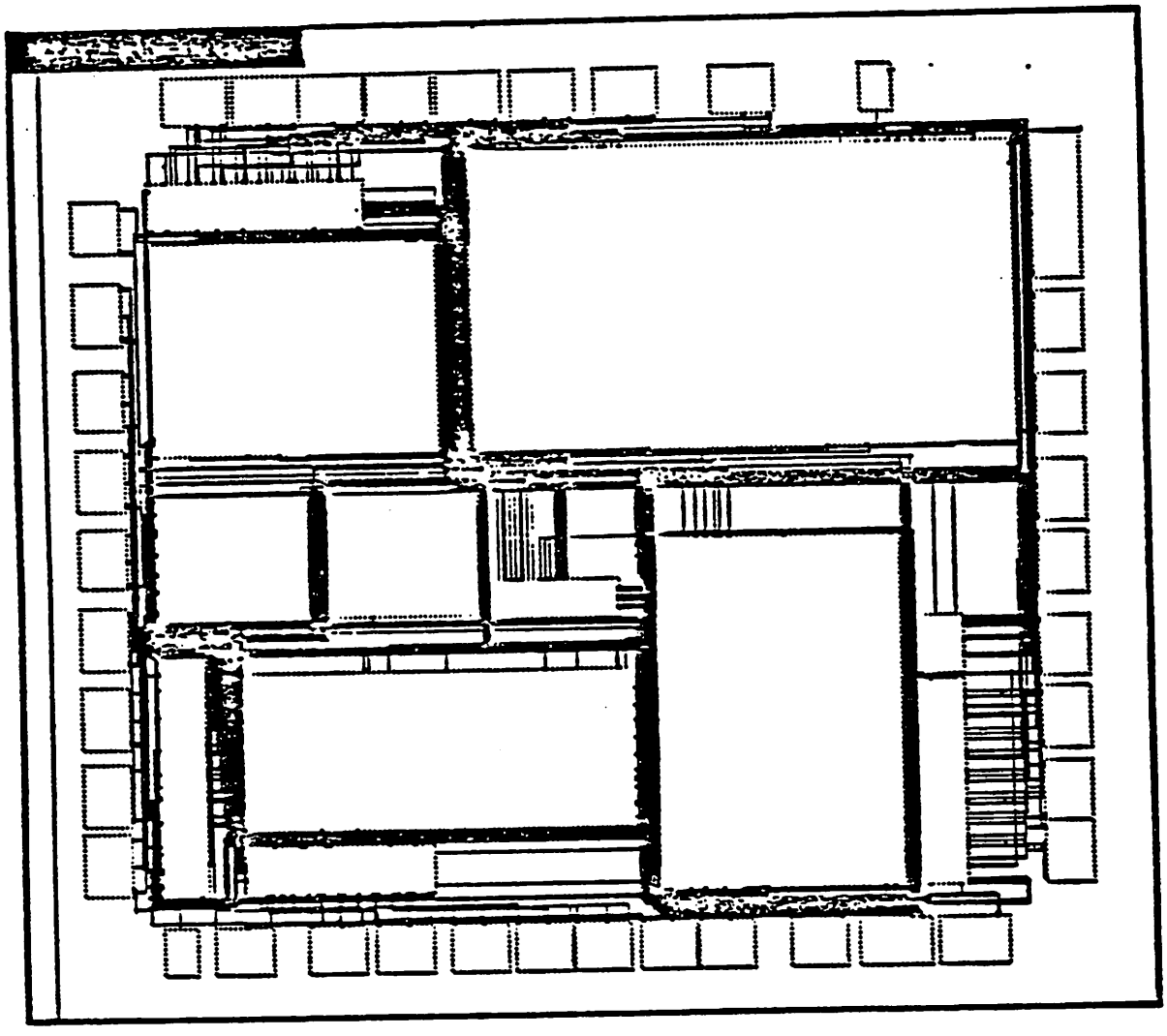

Figure 6. Circuit Ck2

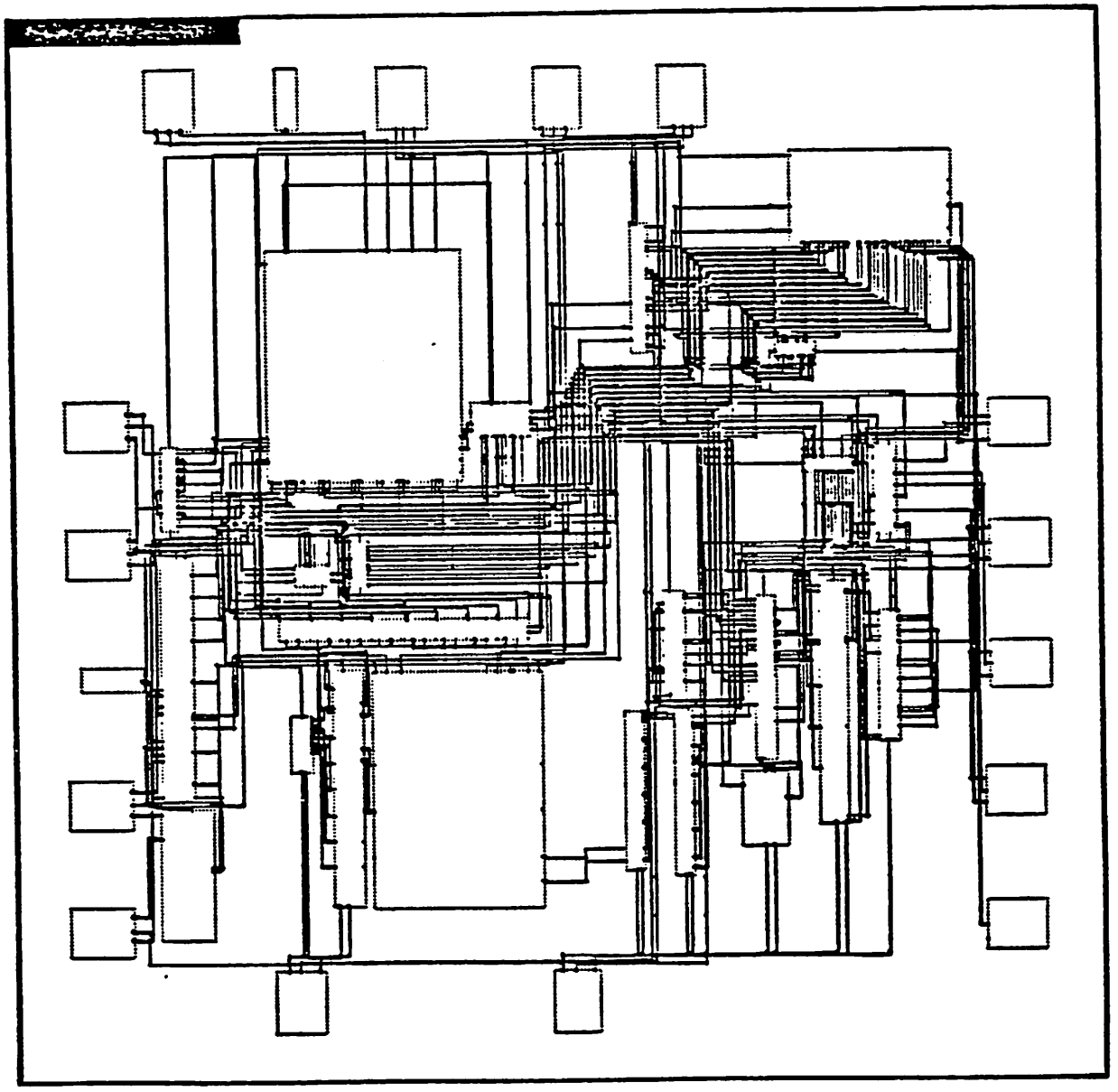

Figure 7. Circuit Ck3