Preprint from Proceedings of the Summer School on VLSI Tools and Applications,
Switzerland, July 1986, Kluwer Acadmic Publishers
W. Fichtner and M. Morf editors.

DESIGN AND LAYOUT GENERATION AT THE SYMBOLIC LEVEL

Carlo H. Séquin

Computer Science Division
Electrical Engineering and Computer Sciences
University of California, Berkeley, CA 94720

ABSTRACT

A pipeline of three tools for the construction of high-quality macro modules or library cells is described. TOPOGEN is a synthesis tool that takes a logic description at the gate level and converts it into a symbolic layout of a static CMOS circuit on a virtual (coarse) grid. EDISTIX is an interactive virtual grid editor for the creation or modification of symbolic sticks diagrams. ZORRO is a two-dimensional compactor using the concept of 'Zone refining' to generate the mask geometry from the symbolic layout. The generation of the final layout of a cell is a two-step process using an intermediate symbolic representation on a virtual grid. In this intermediate state, the user can interactively make changes.

1. INTRODUCTION

The creation of high-quality cells and macro modules is a corner stone of automatic and semi-automatic chip synthesis. This is true regardless whether a full custom or a semi-custom standard-cell approach is taken.

The rapid progress of VLSI fabrication technology renders existing standard-cell libraries obsolete rather quickly, so that they must be adapted periodically to a new set of mask layers and new design rules. Because of the repeated usage, density and performance of these cells is important, and thus a lot of effort is normally spent to obtain optimal cell layouts.

Emerging "Silicon Compilers" normally work in hierarchical stages. Hand-designed library cells and procedurally generated modules are assembled at the chip level by powerful placement and routing tools. The latter tools recently have started to outperform human designers for complicated tasks with many blocks. However, automatically generated cells and macro module rarely achieve the performance and density of hand-designs by a good designer. An exception are some special modules such as PLA's, but there the gain stems primarily from logic minimization and from topological folding rather than from the actual layout.

In the last two years we have concentrated some of our efforts on tools that make the production of high-quality cells and macro modules easier and more

automatic. The emerging system consists essentially of three parts. TOPOGEN is a synthesis tool that takes a logic description at the gate level and converts it into a symbolic layout of a static CMOS circuit on a coarse virtual grid. EDISTIX is an interactive virtual grid editor for the creation or modification of symbolic sticks diagrams. ZORRO is a two-dimensional compactor using the concept of 'Zone refining' to generate the mask geometry from the symbolic layout.

The generation of the final layout of a cell thus becomes a two-step process: conversion of the circuit into a good topology on the virtual grid, and then the fleshing out of the sticks elements and their geometric compaction into a dense layout in accordance with a given set of geometrical design rules. At the intermediate level, the designer has the option to review and possibly improve the topology of the cell with the interactive program EDISTIX. It is also in this intermediate format that the design of the cell should be stored for rapid generation of a new cell when there are small changes in the implementation technology.

2. THE ROLE OF SYMBOLIC REPRESENTATION

The direct conversion of a circuit into a dense layout is too big a step to be taken directly, — this is true for the human designer as well as for a computer program. The concerns of finding a good topology for the layout and of arranging the components to satisfy all design rules are independent enough, so that these two issues can be resolved separately in a two-step process. Between the two steps lies the symbolic representation of the layout in some sticks-like format.

We will briefly discuss the requirements for the representation at this symbolic level and then discuss our chosen representation.

2.1. Requirements for a Symbolic Representation

In the choice of the primitives at the symbolic level, one tries to combine various diverse goals.

The representation should be lean and uncluttered to make it easy for the designer to address the concerns he has at this stage of the design. These are to find an optimal topology for the module under construction that will produce a module of a desirable aspect ratio, place the connections to the external world at the proper sides of the module, and produce direct and minimal internal wiring as well as simple geometry for the well regions.

To be able to make reasonable choices on the topology, the symbolic representation must be expressive enough to render the tricks that are routinely used in the hand-layout of dense library cells. One such trick is to run metal wires over large transistors and to produce, if necessary, a cross-under for a signal that enters the drain/source diffusion on one side of this metal connection and gets picked up on the other (Fig. 1). This construction cannot be represented if the transistor at the symbolic level is viewed as a point device with only four possible connections, one each in the four major directions.

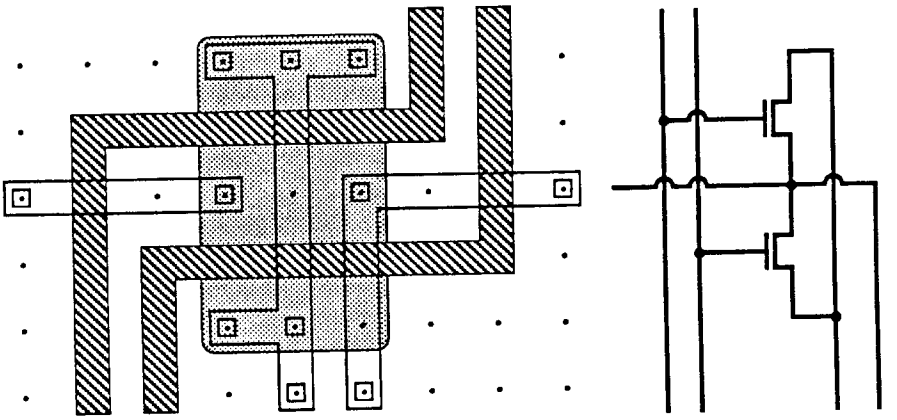

Figure 1. *Cross-under produced by a diffusion region between two transistors. This could not be expressed if the FET were a point device.*

And finally, the symbolic representation should be efficient. It must represent succinctly and unambiguously the geometrical and electrical properties of the circuit, so that the sticks diagram can be checked for functionality and evaluated qualitatively for the area required by the final cell. Of course, it is preferable to keep the size of the file describing the cell at the symbolic level as small as possible.

2.2. Virtual Grid and Raster Components

We have chosen a coarse virtual grid as the basic design space. It allows for a terse representation and makes the geometrical part of the data structure very simple. Further, the determination whether two components are actually connected is straight-forward; this makes the checks for possible illegal interference of components rather simple.

Every component in this representation occupies a number of grid points. The set of basic components selected for our symbolic representation is show in Figure 2. All components can be viewed as linear elements spanning one or more grid points. For wires, this representation is an obvious choice. It also applies for the port, a formal terminal that can serve as a connection to the outside world. If the port extends over more than one grid point, it is still considered an equipotential node. Contacts are also linear equipotential elements, typically represented as rows of contact holes spaced one grid unit apart.

Transistors are slightly more complicated. They occupy three rows of grid points next to one another, one each for the source diffusion, the gate, and the drain diffusion. They still fit the paradigm of a line element, as internally only the "stick" for the gate is represented explicitly, and the adjacent diffusion areas are implied and derived on the fly when needed for some check or for display on the screen. This gives the symbol set a cohesiveness that makes the various data

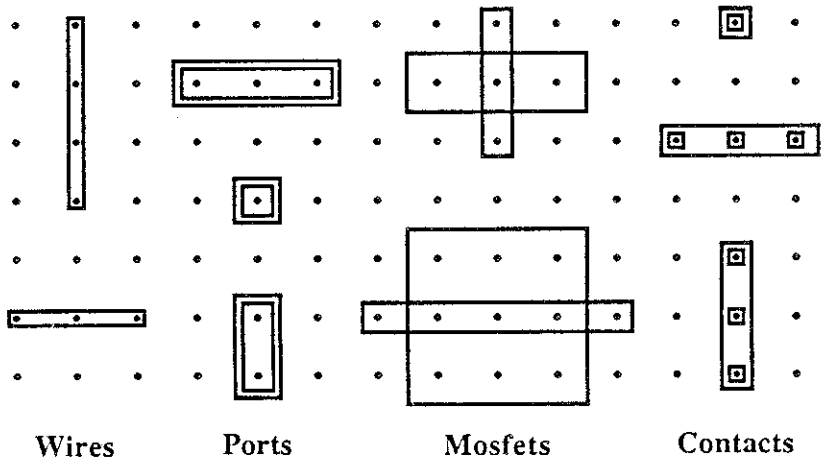

Figure 2. *Virtual grid components in EDISTIX.*

structure manipulations more regular.

In addition there is an auxiliary component called the joint. It is used wherever two or more wires join together. Joints are strict point elements. We first tried an implementation with a data structure that did not need these joints and connected wires directly to one another. The resulting data structure and its manipulation became rather cumbersome. The addition of the extra joints, where needed, simplified things. These joints need not be represented explicitly in the file that describes the circuit symbolically; they are introduced and deleted on demand whenever a wire end is not explicitly connected to a terminal, contact, or transistor.

Every tool described below has its own internal representation of these sticks elements that is most appropriate for the task that the particular tool has to perform. The information is passed between the various tools by means of terse ASCII files. The format of these files is very simple: Every element is represented by a keyword that implies its type and layer and the integer coordinates of its endpoints. In addition, ports and transistors can take names for identification. This decoupling through the use of these intermediate files makes the databases for each tool simpler and more efficient and permits separate tool development.

3. EDISTIX

The virtual grid components described above can give a reasonably accurate description of the layout organization and of the achievable packing of the components. This is necessary to allow the designer or an automated tool to find an optimal module topology. Because of the central role of the symbolic representation, we will first give more details on EDISTIX, rather than present the pipeline of tools in the sequence that an evolving design would see. EDISTIX acts as the glue between the other two tools, and its internal data structures are a good

example how one can deal efficiently with the described sticks components.

3.1. The Function

EDISTIX is an interactive virtual grid editor that relieves the designer of many of the chores associated with the modification of symbolic sticks diagrams. The purpose of this tool is to make it easy to enter symbolic designs from scratch or to inspect and modify the ones that come out of a tool like TOPOGEN.

In the first case, the goal is to make sticks entry as fast as sketching on a pad of paper, but with all the potential advantages of having a smart checking program looking over your shoulder and preventing you from making simple mistakes such as tying 'Power' and 'Ground' together. Particular attention was thus given to the user interface, with the goal of minimizing the necessary actions during the entry of circuit elements.

In the second case, the main goal in EDISTIX is to make it easy to change the topology of a layout without changing its connectivity. If a designer wants to improve the layout topology (in cases where TOPOGEN gives less than optimal results) he should be able to spend most of his attention on finding an optimum topological arrangement without having to worry that the interconnections might be changed accidentally in the process. Thus, in this mode, EDISTIX keeps the internal netlist unchanged and tries to reroute all interconnections accordingly when components are moved.

3.2. Data Structures

Considerable effort has been spent to find efficient data structures to represent the geometrical as well as the electrical aspects of a design.

In the *geometrical data structure*, because of the limited size of non-hierarchical macro modules or library cells, and since in good topological arrangements practically all vertical and horizontal grid lines contain at least one component, it is reasonable to represent all the rows and columns of the drawing area explicitly, rather than using sparse matrix techniques. Thus for every row and column we list all the vertical and horizontal line-elements, respectively. In each of the two directions, these elements are grouped into five linked lists sorted by element types (Fig. 3). Thus we store in separate lists: wires and links, contacts, joints, ports, and FET's. This makes it easier to search for a particular element type and to provide the different processing routines necessary for different element types.

Since the elements are either horizontal or vertical sticks, their geometry is fully captured with three numbers: their row/column number and two values for the second coordinates of their endpoints.

In the *electrical data structure*, a distinction is made between equipotential *nodes* such as ports, contacts, FET-terminals, or joints, and binary *connection elements* such as wires and links (Fig. 4). All nodes are connected in a linked list in

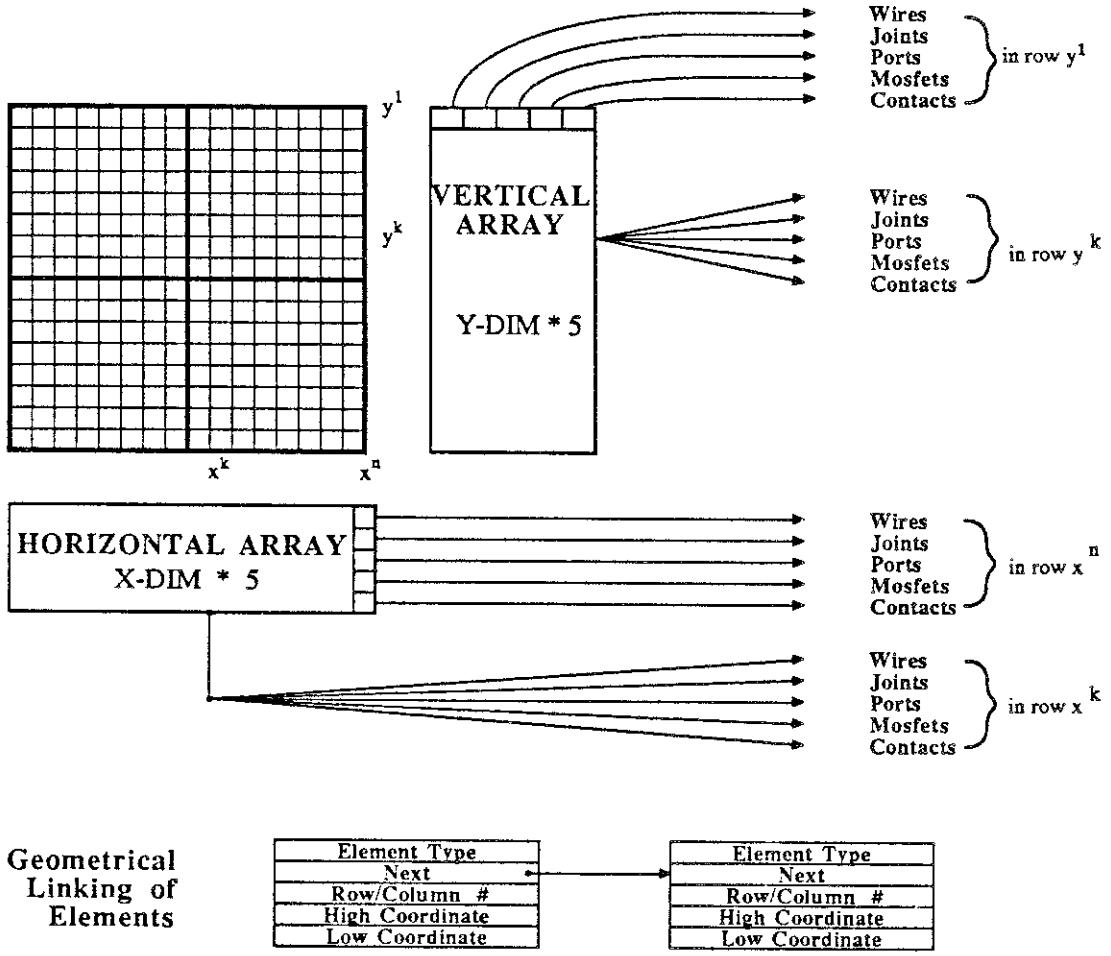

Figure 3. *Geometrical data structures in EDISTIX.*

the order they are created. They may carry an optional name. They also have a pointer that points to the first child, i.e., any attached wire or link, or is nil.

Wire or links are two-ended elements that are attached to two nodes. At each end they have two pointers, one pointing to the node to which they are connected, and the other pointing to any 'siblings', i.e. other wires or links attached to the same 'parent' node.

All the geometrical and the electrical information is contained in the same structure representing both aspects of an element.

3.3. Operations

The many possible operations can be grouped into various classes: edit operations, selection and query commands, clean up operations, rearranging the topology, analysis, and output.

Edit operations are used to build a circuit from scratch or to modify a given circuit. They include the standard operation to add or delete an element, and to

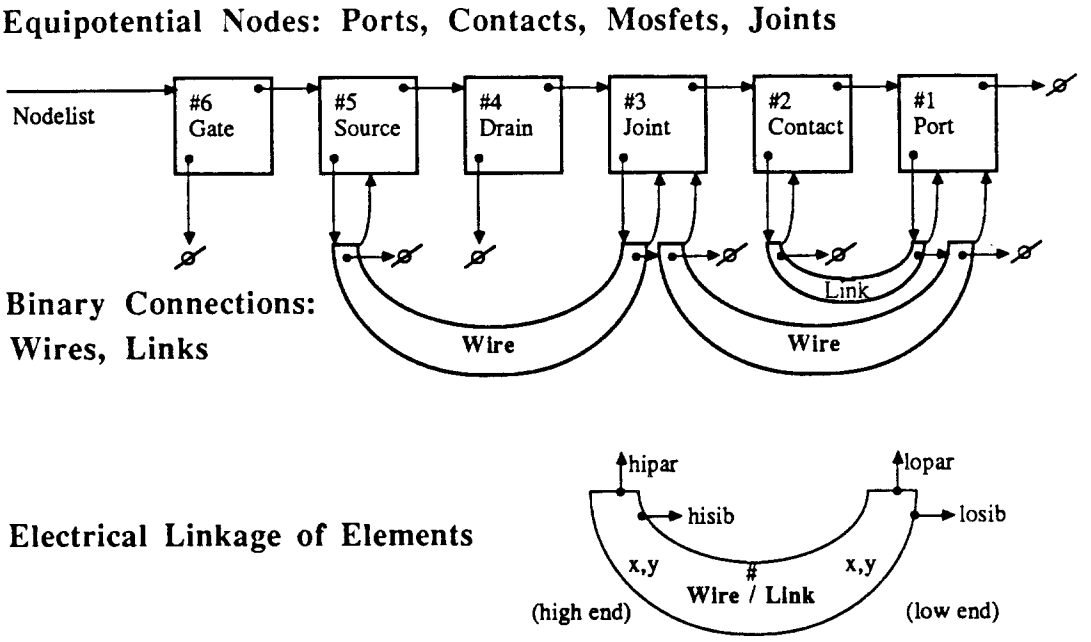

Figure 4. *Electrical data structures in EDISTIX.*

select and modify an element. When adding an element, the program watches for illegal constructs such as running a poly wire across a diffusion area, or it warns you of questionable configurations such as level crossings of wires that will lead to an implicit electrical connection. The program maintains up-to-date information about all electrical connections, and it will warn the user when nets with different names are connected or when a loop is formed in a net.

Selection and query commands permit the user to pick one or more elements on the screen and then see a listing of the detailed information on that element as well as a list of other elements it is connected to. Some fields such as the name can be changed.

Clean-up operations remove dangling wires and contacts and merge pairs of collinear wires on the same level. This brings the internal representation into a minimal consistent state.

Rearrangement commands allow the user to change the layout without changing the underlying circuit. These operations are using a generalized block move operation. A group of elements, selected individually or by an 'area select' command, are moved jointly by a given displacement vector. Connections that go beyond the selected area and connect to components that remain fixed have to be recreated. The system does as much rerouting as possible and shows the remaining connections that it cannot handle in contrasting color. It is then up to the designer to find a feasible implementation for these wires, or to make further changes that enable the wiring to be completed.

Analysis commands (not yet implemented) will eventually allow the user to interact directly with a simulator or a timing verifier. In this way it will be easy for the designer to verify functionality or to get a first estimate on performance. For the time being, the designer will have to produce an output file with one of the various drivers for a particular simulation tool, and then run that tool separately on this file.

Output of the stored data on the design can be viewed in many different forms. Elements can be listed in geometrical order, going through the various types of devices on a row by row and column by column basis. This is the default scan mode used when the user wants to write out a file of the database in the ASCII format. Alternatively, the nodes with all the attached children can be listed in the order in which they were generated. Finally the whole network can be traversed in a depth first manner; this is the mode that is used when one wants to create an output file in the format for simulators such as SPICE[1] or ESIM.[2] There is also a possibility to create a file in the format of the OCT data base[3] so that the other tools of the Berkeley Design Environment[4] can be run on the cells generated with EDISTIX.

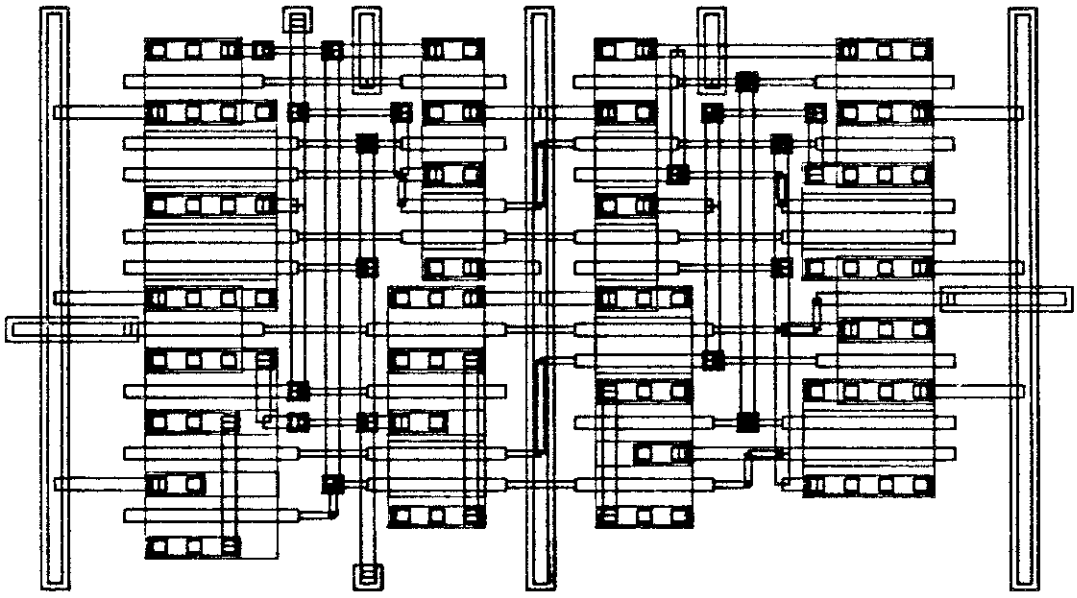

Figure 5. *Virtual grid representation of a flipflop composed with EDISTIX.*

3.4. Results and Discussion

Figure 5 shows the sticks representation of a flipflop as it would appear on the EDISTIX screen. EDISTIX has been under continued development for a couple of years. It has been rewritten from scratch at least four times, first a couple of times in Pascal, more recently in C. The general features discussed in this section have been rather stable over the last few versions, and we are confident that

they represent a good solution to capturing a symbolic layout. It gives a rather good idea of what the final layout might look like.

4. TOPOGEN

TOPOGEN is a generator program that takes a functional description at the logic gate level and converts this into a symbolic layout on a virtual grid. The first version of TOPOGEN is aimed at standard cells for a static CMOS family. So far, the layout style is restricted to a single row of transistor pairs with one diffusion strip each for the p-channel and n-channel FETs, respectively, TOPOGEN is organized in a modular fashion, so that one can experiment with different algorithms for the various steps mentioned below.

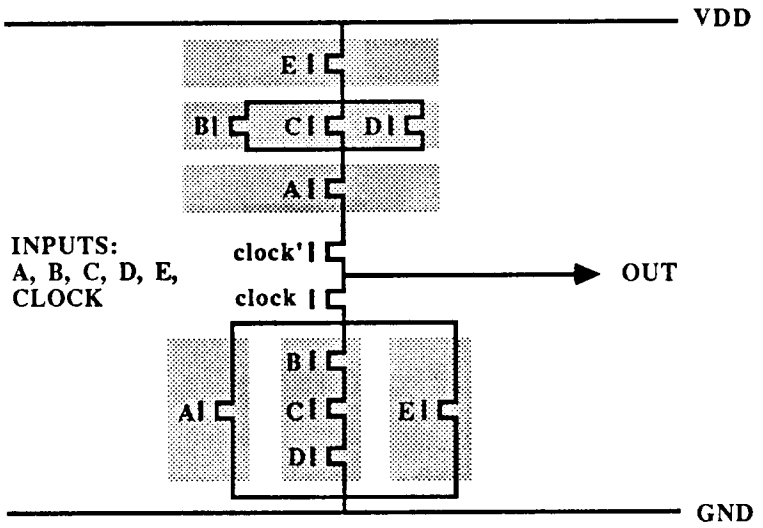

Figure 6. *Circuit generated form the following TOPOGEN input: (evalgate (output OUT) (npt CLOCK) (or A (and B C D) E)).*

4.1. Circuit Generation

The translation of the logic description into a corresponding circuit is straight-forward. TOPOGEN accepts nested AOI expressions that are converted to the corresponding series / parallel networks of transistors. The program looks at every AOI gate in the input stream separately. In the sequence in which the logic inputs appear in the original description, corresponding transistors are placed from left to right and from output rail towards the power/ground lines (Fig. 6). In addition, single or paired clocked switches can be specified. These clock inputs can be placed next to the output rail or next to the supply lines, depending on whether the clock input specification appears before or after the description of the Boolean logic block.

4.2. Gate Optimization

The circuits obtained in the manner outlined above are now arranged as a linear sequence of transistor pairs. In each gate the sequence of the transistor pairs is arranged so that the mutual sharing of the diffused drain/source areas is maximized and thus the length of the rows of transistors is minimized. This amounts to finding corresponding Euler paths through all the transistors of either polarity. We use the method of adding a pseudo input in every series / parallel block with an even number of components[5] since that makes the construction of an Euler path trivial. These pseudo inputs correspond to turned off gates or isolation zones in the final diffusion strips. Their number is minimized by permuting the sequence of the children at every node of the AOI tree (Fig. 7). Multiple adjacent isolation zones can then be collapsed into one.

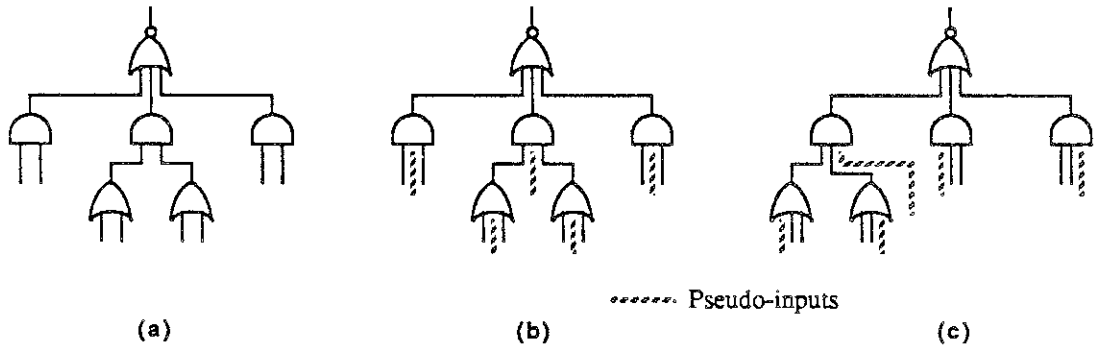

Figure 7. *Gate rearrangement to produce an Euler path through a circuit.*

4.3. Gate Placement

TOPOGEN subsequently rearranges these individual and-or-invert gates with the goal to minimize the width of the wiring channel between the two diffusion areas. A pairwise interchange algorithm is used to step through all the gate positions once, comparing the potential gains in exchanges with all the gates that lie ahead in the line. The cost function to be minimized is the width of the resulting strip, i.e., the maximum of the sum of the width of both the P and N transistors and the local density in the wiring channel. Since good channel routers can wire a channel without exceeding its density, this evaluation function is quite appropriate.

4.4. Wiring

When a suitable gate arrangement has been determined, all the necessary interconnections in the area between the two diffusion strips are generated. We use the latest channel router available to us. We have had good success with YACR II[6] and we are currently experimenting with others such as CHAMELEON[7] and MIGHTY.[8] TOPOGEN simply writes an ASCII file

specifying the routing problem in the particular format that the router needs and subsequently reads the generated file with the wiring description.

In trying to modularize our design environment, we are in the process of defining "standard" format for the description of a routing problem and for the generated solutions. To be general enough, we permit the routing region to be any arbitrary rectagon, signal input pins can lie on this rectagon boundary or inside, and there can also be obstacles inside the routing region on one or more layers. Issues that need to be resolved concern the transformation of the signals from the layers given by the original problem situation to the levels that the router is prepared to handle, and questions whether the router can introduce a level change right at the location of a signal pin.

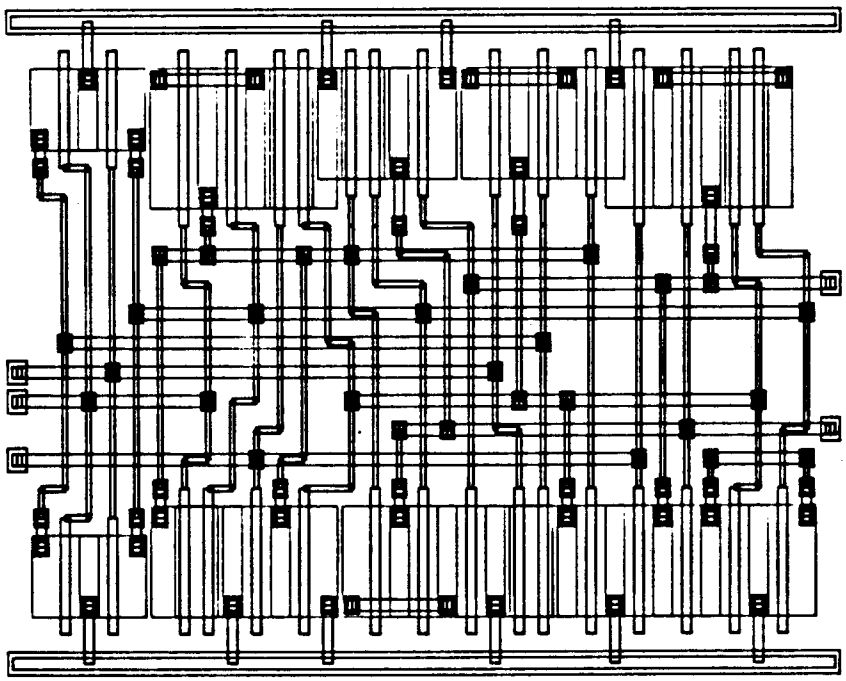

Figure 8. *Sample output generated by TOPOGEN.*

4.5. Output

The final phase is to write an output file in the format understandable by EDISTIX. This is fairly straight-forward since TOPOGEN internally has built up all transistor positions and wirings on the same kind of coarse virtual grid used by EDISTIX. A typical output for a small group of simple gates is shown in Figure 8.

4.6. Results and Discussion

In its current form, TOPOGEN is a useful tool for clusters of gates totaling about a hundred transistors. The layouts are not yet competitive with a hand design. The main reason is that TOPOGEN carries out each phase of the chosen design process without much concern for the other phases and without any iterative feedback loop. We are in the process of reducing this design gap by incorporating more sophisticated routing algorithms that can route over large transistors. In order to handle larger gate clusters, We have started to extend the basic approach to modules with multiple strips of complementary transistor pairs; in this case the gate placement is a harder problem that requires more sophisticated techniques than simple pairwise interchange with the goal to minimize channel density.

5. ZORRO

The third step in the generation of a standard cell is the production of the final mask geometry for the particular technology to be used for implementation, i.e. the compaction of the symbolic circuit representation with proper dimensioning and spacing of all elements. Most of the compaction or spacing programs in practical use today can alter only one coordinate of a component at a time. This leads to certain deficiencies in the compaction process that make the automatic spacing of layouts inferior to the work done by the human designer. The resulting inefficiencies are typically considered unacceptable for frequently used library cells.

Experimental two-dimensional compactors have been built with different approaches. One approach is to start with a totally collapsed layout and then remove the distance violations one by one.[9] G. Kedem and H. Watanabe[10] translated the compaction problem into a special form of a mixed-integer programming problem. An even more fundamental approach uses simulated annealing techniques[11] for the placement of the components.[12] All these approaches typically show non-polynomial growth in runtime for large circuits.

We have taken a less expensive approach to 2-dimensional compaction. Only a small part of the circuit is opened up for two-dimensional motion of the components. This open zone is swept through the precompacted layout in a strong analogy to the *zone refining* technique used in the purification of crystal ingots.

5.1. Zone-Refining

In close analogy to zone refining of crystals (Fig. 9a), we start from a circuit layout that has been "crystallized" by precompaction with a one-dimensional compactor. In our case the "impurities" that we want to sweep out of the "crystal" are the unnecessary voids between circuit components. Starting from the bottom, individual circuit components or small clusters of components are peeled off row by row from the precompacted layout and are reassembled after they have

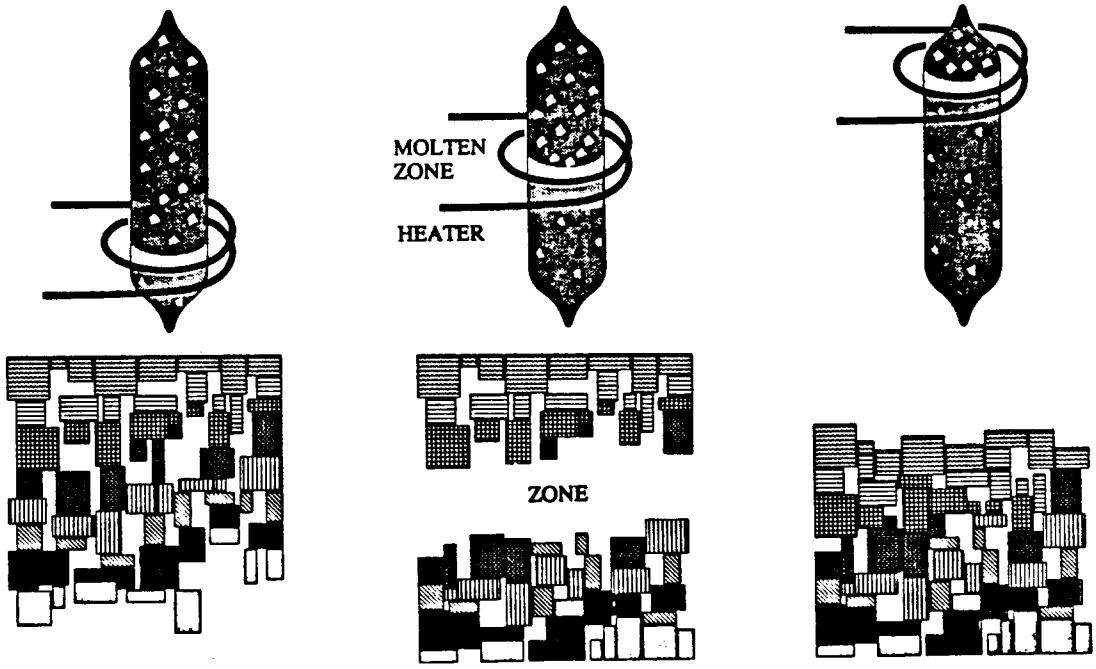

Figure 9. *Zone refining: (top) of crystal ingots and (bottom) of layouts; the preferred direction of compaction is vertical the direction of the sweep, but the blocks in the zone can also make lateral movements.*

been moved across an open zone (Fig. 9b). As the components pass this free zone, they can move laterally to a more advantageous position that will result in a denser layout. In the process of reassembling the components at the other end both coordinates of the moved components can be altered and jogs can be introduced in the connecting wires between the circuit components. These additional degrees of freedom permit a higher packing density in the newly formed part of the layout than can be achieved with a one dimensional compaction process.

The geometrical design rules are observed by maintaining and using the constraint graphs in both the x- and y-direction.

5.2. Data Structures

The main data structure is the *adjacency graph*, here illustrated on the simple example of a packing problem involving rectangular boxes (Fig. 10a,c,d). The positions of all blocks are represented in the nodes of the graph. All horizontal and vertical adjacencies are represented as two types of corresponding arcs between the nodes (Fig. 10b,e). These arcs are labeled with the minimal allowable horizontal or vertical separations between the centers of the block; this adjacency graph can thus be turned into a constraint graph for properly placing the blocks without overlap. For an actual circuit layout, the constraints attached to these arcs become more complicated and contain upper as well as lower bounds.

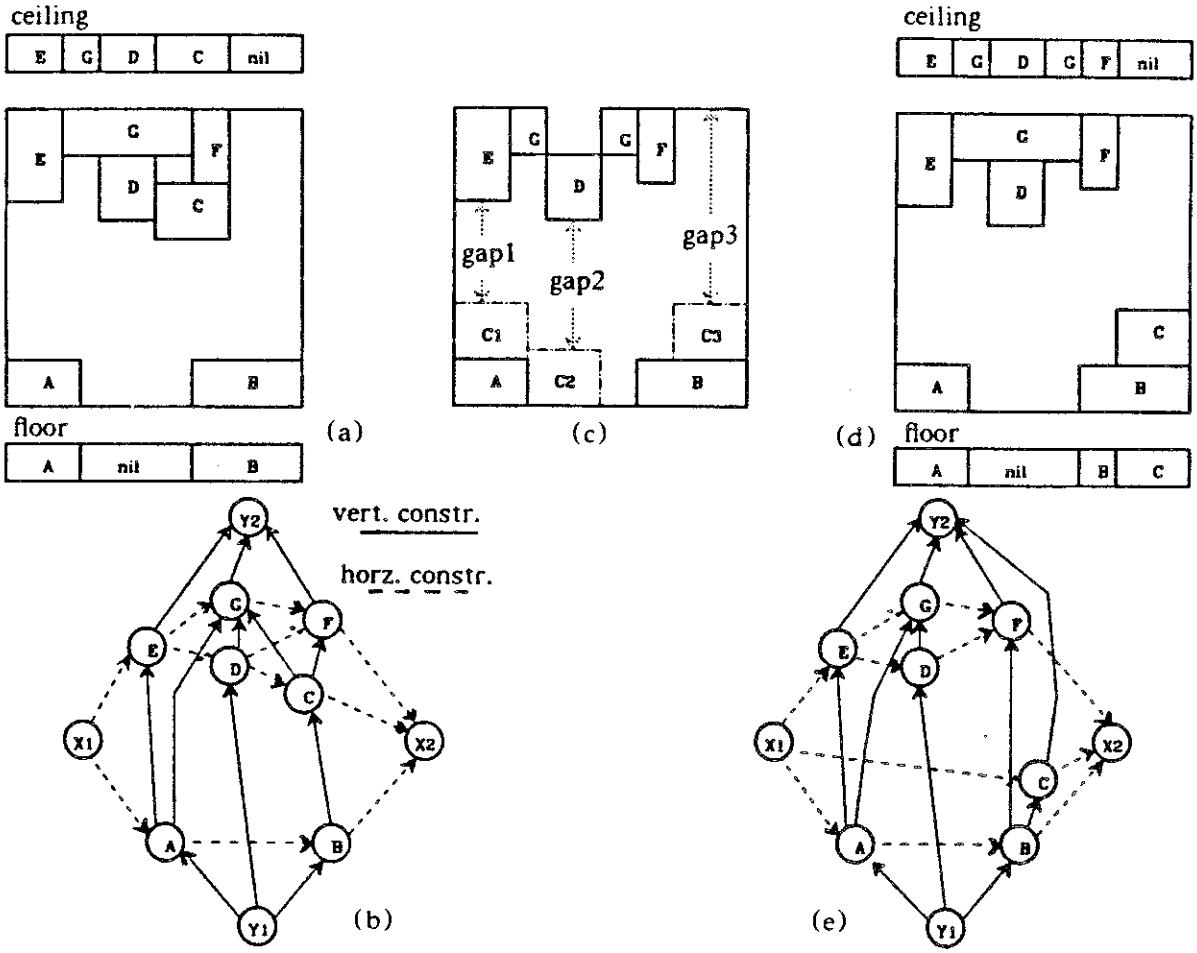

Figure 10. *Example of box packing in progress. (a) Intermediate constellation of boxes and floor and ceiling data structures. (b) Corresponding adjacency graph. (c) Box C has been selected to be moved; three candidate places C1, C2, C3 are evaluated. (d) Box constellation after box C has been placed and new floor and ceiling structures. (e) Updated adjacency graph.*

A second data structure is associated with the moving refinement zone and contains the currently active components that must be referred to frequently in each block move. All elements that form the boundary of the free zone, above and below, are joined together in the 'ceiling' and 'floor' data structures, respectively (Fig. 10a,d). They permit an efficient evaluation of the best position for the elements that are being moved across the zone.

5.3. Zone-Refining Algorithm

Elements are moved from the top part to the bottom part of the circuit across the open zone with the following algorithm. In the ceiling an element is selected that hangs farthest down. For simple box packing, an individual box is selected. For actual circuits, where the components are connected with wires, a whole cluster of components that is connected by horizontal wires without jogs in

them must be moved at once. The selected components are removed from the ceiling data structure and from the horizontal adjacency graph. They are now free to float around in the zone.

Now the best location for placing the component on the floor has to be found. We are looking for the position that maximizes the narrowest part of the zone, because then we know that the two halves of the circuit can fit together with minimum total height. In the case of box packing, all grid positions from the left extreme to the right extreme are evaluated. For circuits, the lateral motion is much more restricted. In the first version of ZORRO, components or clusters of circuits are only moved laterally within the freedom allowed by the attached wires. Wires can be moved to the extreme positions of terminals, and horizontal parts of wires can be stretched, but no new jogs are introduced at this point.

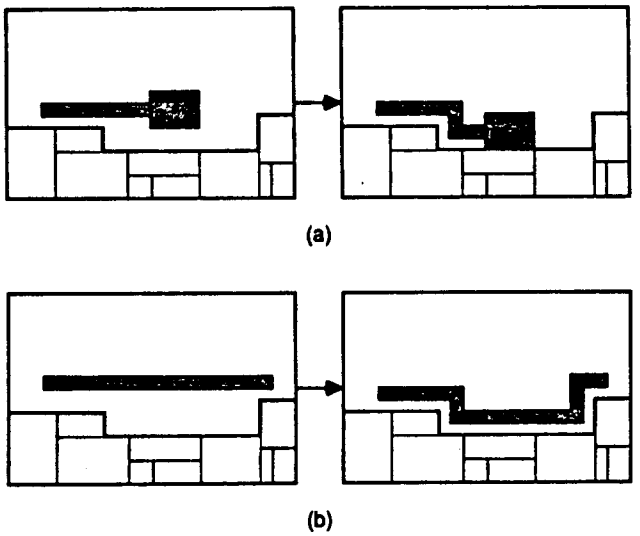

Figure 11. *Automatic jog introduction in horizontal wires.*

Once the optimal x-position has been found, the box or the circuit cluster is moved onto the floor and is properly integrated into the floor data structure and into the two adjacency graphs. Updating the horizontal and vertical adjacency graphs is done in an incremental manner. When a component is moved in the vertical direction, its horizontal arcs are removed. Once it is in the new y-position, the new adjacencies are detected by sweeping a scan line across the height of the component and checking what other components get intersected. New horizontal arcs are formed for all discovered adjacencies. Corresponding operations on the vertical adjacency graph are carried out whenever a component is moved horizontally.

For the case of circuit compaction, all attached wires have to be placed properly, once the best place for the moved component cluster has been found. Jogs may have to be introduced in the horizontal wires to permit the component to move all the way to the floor (Fig. 11a). To maximize vertical compression, we

will also bend some of the horizontal wires that span over large enough regions of empty space (Fig. 11b).

5.4. Results and Discussion

Figure 12 illustrates the zone refining process on a simple example of box packing. We start from a randomly generated array of rectangles and compact it in the upward direction; the overall height of the array is reduced from 80 units to 63 units. A first zone refining pass, where the boxes move downward across the open zone, reduces the height to 53 units. The second zone refining pass in the opposite direction brings the total height to 47 units. This is the limit; additional zone refining passes do not reduce the height of the constellation any further.

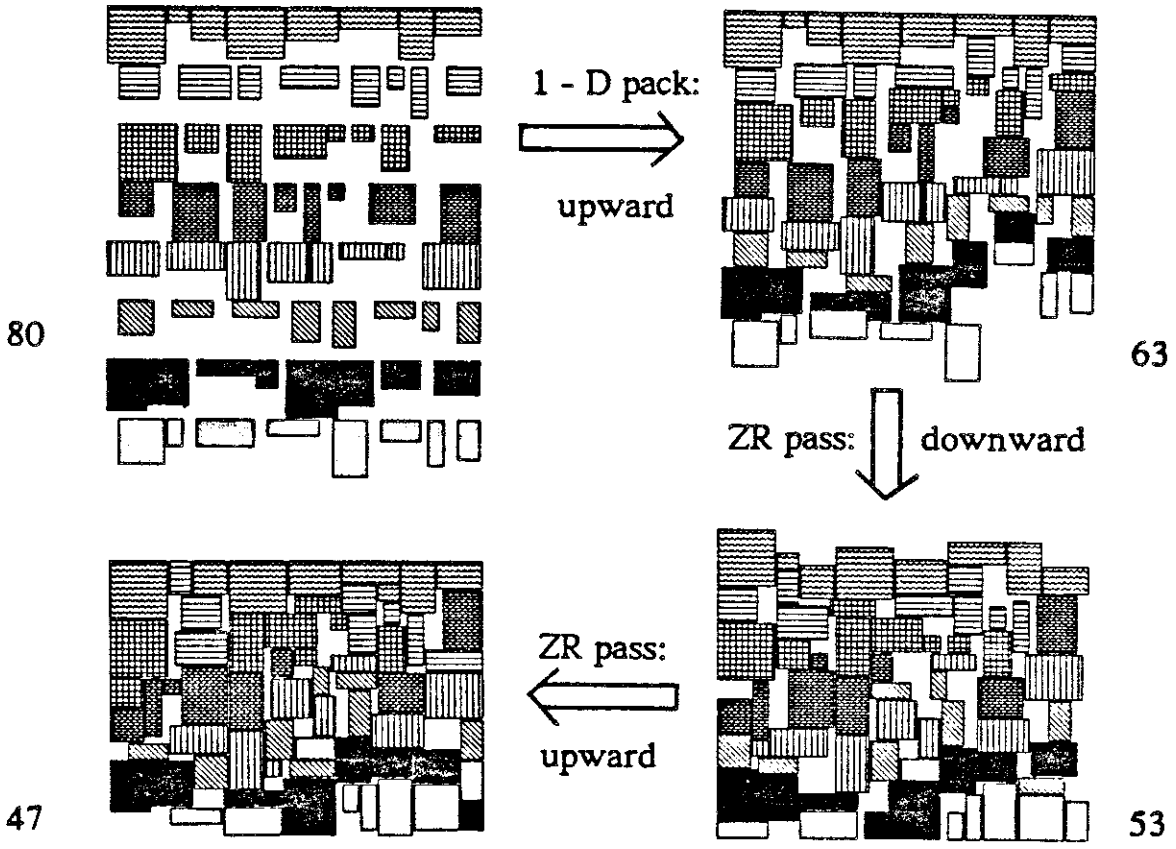

Figure 12. *Example of box packing with zone refining.*

Figure 13 shows various phases in the compaction process of a real circuit. First it shows the precompacted circuit with merged contacts and nets. The next two figures illustrate an intermediate and the final state of the first zone refining pass on this circuit. The last figure shows the result after four more zone refining passes in the vertical direction; these passes also include jog generation in horizontal wires. The obtained reduction in area is 33% compared to the result of simple one-dimensional compaction.

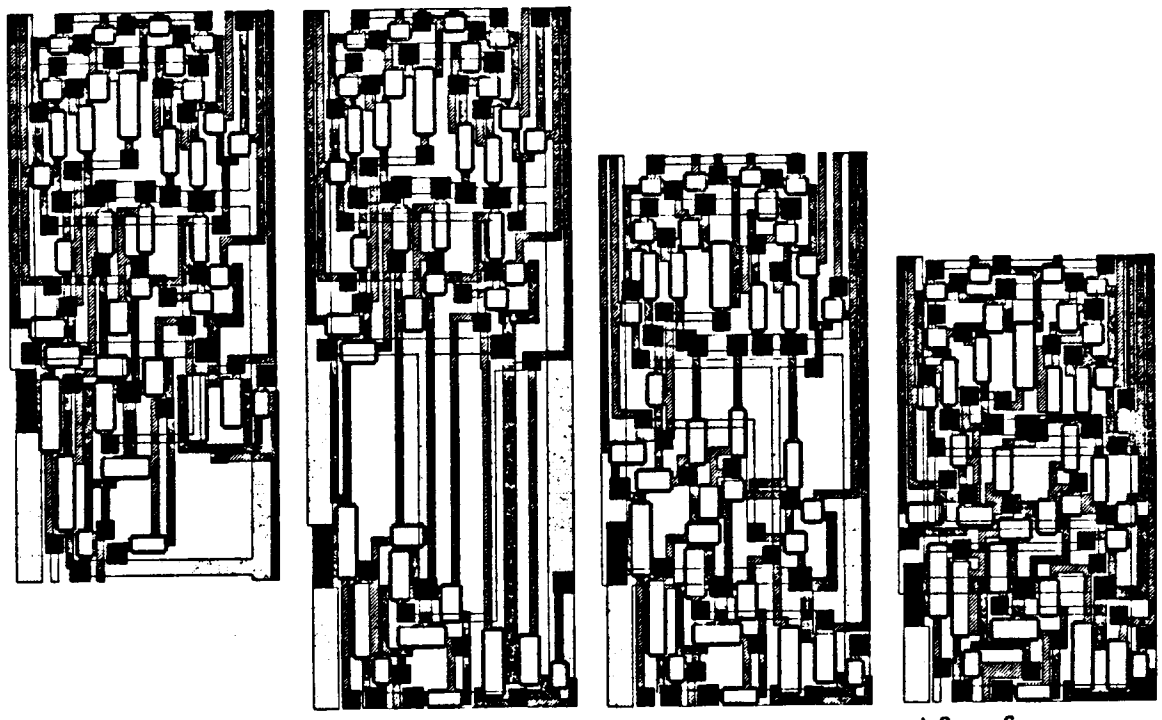

Figure 13. *Example of circuit compaction.*

For box packing problems, zone refining can reduce the area occupied by up to 30% beyond what a one-dimensional compactor can do, at a cost in total run time that is 10 to 30 times longer, depending on the number of passes. For circuits similar improvements have been observed, but because of the complications introduced by the attache wires, and the need for jog generation, total run time can be up to 100 times longer than that required for one-dimensional compaction.

Interconnections play a crucial role in the performance of the circuit, and the given topology of the circuit often has been chosen based on considerations at the micro-architecture level. Thus we do not want the compaction tool to make profound changes to the topology of the circuit; this is the task of a different kind of tool that can take properly into account concerns beyond observation of the geometrical design rules. Thus for the zone refining process we assume that we start from a good topology, given for instance in the form of a symbolic sticks diagram. The given basic ordering is maintained in the compaction process, distinguishing our approach from the more general problem of block placement and routing.

The advantage of the zone refining approach over global two-dimensional placement algorithms is that the number of components that must be considered

at any one time is dramatically reduced, and the complexity of the algorithm thus is only of polynomial complexity. In addition, just as in the physical zone refining process, the compaction process can be repeated if the results are not yet satisfactory after the first pass.

6. CONCLUSIONS

"Silicon Compilers" as well as human designers like to reduce design complexity by separating concerns, where possible. In the creation of dense library cells or macro modules, finding a good layout topology and observing all the geometrical layout rules for a particular implementation technology are two distinct concerns that can be addressed in subsequent design phases. A well-chosen symbolic representation to capture the design at the intermediate state is crucial to facilitate the design process and to obtain good results. The coarse-grid components used in EDISTIX seem to fulfill these needs quite nicely.

With this symbolic representation at the center, the design of a high-quality module becomes a two-step process. First, the gate or circuit-level description gets converted to a good sticks layout, then this symbolic representation gets compacted to a real layout. Both these steps can be automated. With TOPOGEN we have created a prototype of a generator that will produce acceptable topologies for clusters of CMOS logic gates. ZORRO is a first prototype of a new class of two-dimensional compactors that can convert sticks-representations to practical layouts. Before long, the process of module generation will be largely done by computers.

7. ACKNOWLEDGEMENTS

Over the last two years several people have worked on and contributed to the tools described in this paper. Special thanks go to the most recent set of developers who have also given me constructive criticism on this paper: Ping-San Tzeng, Glenn Adams, and Hyunchul Shin.

This work is supported in part by the Semiconductor Research Corporation and by the State of California under the MICRO program.

References

1. L.W. Nagel and D.O. Pederson, "Simulation Program with Integrated Circuit Emphasis," *Proc. 16th Midwest Symp. Circ. Theory*, Waterloo, Canada, April 1973.
2. C.M. Baker and C. Terman, "Tools for Verifying Integrated Circuit Designs," *Lambda*, vol. 1, no. 3, pp. 22-30, 4th Q. 1980.
3. D. Harrison, P. Moore, A.R. Newton, A.L. Sangiovanni-Vincentelli, and C.H. Séquin, "Data Management in the Berkeley Design Environment," *submitted to ICCAD-86*, Santa Clara, CA, Nov. 1986.

4. C.H. Séquin, "VLSI Design Strategies," in *Proceedings of the Summer School on VLSI Tools and Applications*, ed. W. Fichtner and M. Morf, Kluwer Acadmic Publishers, 1986.

5. T. Uehara and W.M. VanCleemput, "Optimal Layout of CMOS Functional Arrays," *Trans. Comp.*, vol. C-30, no. 5, pp. 305-312, 1981.

6. A. Sangiovanni-Vincentelli, M. Santomauro, and J. Reed, "A New Gridless Channel Router: YACR II," *IEEE Trans. Comp.-Aided Design*, vol. CAD-4, pp. 208-219, 1984.

7. A. Sangiovanni-Vincentelli, D. Braun, J. Burns, S. Devadas, H.K. Ma, K. Mayaram, and F. Romeo, "CHAMELEON: A New Multi-Layer Channel Router," *Proc. Design Autom. Conf., Paper 28.4*, Las Vegas, July 1986.

8. H. Shin and A. Sangiovanni-Vincentelli, "MIGHTY: A 'Rip-up and Reroute' Detailed Router," *submitted to ICCAD-86*, Santa Clara, CA, Nov. 1986.

9. M. Schlag, Y.Z. Liao, and C.K. Wong, "An Algorithm for Optimal Two-Dimensional Compaction of VLSI Layouts," *Integration* , pp. 179-209, 1983.

10. G. Kedem and H. Watanabe, "Graph-Optimization Techniques for IC Layout and Compaction," *IEEE Trans. CAD of ICAS,* , vol. 3, no. o 1, 1984.

11. S. Kirkpatrick, C. Gelatt, and M. Vecci, "Optimization by Simulated Annealing," *Science*, vol. 220, no. 4598, pp. 671-680, 1983.

12. R. Mosteller, "Simulated Annealing for IC layout," *privat communication*, 1983.