PSI: A symbolic layout system

by Rolf-Dieter Fiebrich
Yuh-Zen Liao
George Koppelman
Edward Adams

A symbolic layout tool, PSI, is described for use with IBM circuit technology. Significant features of PSI are used with multiple circuit technologies, adaptation to rapid changes of technology design rules, creation of nested designs, and extensive designer control over the spacing process.

Introduction

• *Objectives*

Reports of the development of automatic symbolic layout tools have made evident the potential of such tools to produce physical designs that are correct by construction with much smaller expenditure of resources than is required by more labor-intensive design methods [1–9]. Previously reported symbolic custom layout tools, STICKS [1], CABBAGE [2], MULGA [3], and REST [4], have for the most part utilized spacing rules or element positioning conventions that are more restrictive than might ultimately be aimed at for custom design of industrial products; however, they have shown that such tools can be made practicable.

Using this previous work as a point of departure, we have developed a symbolic layout tool for making custom circuit designs in any of several families of circuit technology used in IBM. Our main objectives in this development were the following:

1. The tool should guarantee that a design satisfies all purely geometric design rules.
2. It should allow nesting of cells and the easy incorporation in a cell of cells designed by other tools.
3. It should provide means for the designer to exert some control over the spacing process, e.g., to specify particular dimensions of the spaced layout, groupings of components, etc.
4. It should accommodate a number of different circuit technology variants and, for each, should incorporate sets of design rules complex enough to achieve good placements of circuits made in each technology.
5. It should provide quick, simple means to incorporate design rule changes in the tool and to automatically update a design in response to rules changes.
6. It should achieve for designs of typical complexity layout densities not greatly lower than those realized by a human layout designer using manual methods.
7. It should capture information needed to audit and ensure design integrity in existing design system data base environments.
8. It should run fast enough to be used in an interactive design environment.

• *Status of implementation*

We developed a first-phase prototype of a symbolic layout system suited to IBM technologies incorporating as many features as we could implement in about a year. We subsequently added to these as we gained experience with the first version of the tool.

Our prototype system, which we call PSI, is generically similar to STICKS, CABBAGE, and REST. However, it differs from the versions of predecessor systems with which we are familiar in many details, most importantly in providing for the explicit nesting of symbolic cells, in providing increased designer control of the compaction

process, and providing the capability to design in any of several circuit technologies.

The system we have at present meets many but not all of our initial goals. In the sections that follow we describe the state of the PSI prototype and our present perception of what is necessary in such a system beyond what was available in earlier STICKS systems for it to be useful in an industrial design environment.

System description

Our system consists of three main program packages. The interactive graphical editor for entry of designs exists in two versions, one using the 3277GA terminal attached to an IBM 3081, the other using a graphics processor terminal connected to a minicomputer. The spacer, which runs under VM/CMS on IBM 3081 equipment, provides means for a designer to make a compact placement of objects in a single-cell layout. The maskmaker program, which also runs only on 3081 equipment, generates specifications in several GL/I idioms for the masks to fabricate circuits described by PSI files.

In PSI a design description is provided as a set of files, a definition file for each independent cell of the design together with a number of auxiliary files prepared from these.

- *Data representation*

A cell definition file consists of a set of one-line records, each record defining either an object in the cell or some information about how objects are to be spaced. In the present implementation a definition file is a text file.

The principal section of this file contains a list of all objects contained in the prime cell. The objects that may be specified in a cell description are the following:

- cell boundary,
- cell terminals,
- transistors of several types,
- pullup devices,
- contacts,
- wires,
- rectangular shapes on layers,
- n-wells, p-wells,
- instances of other cells.

Other sections of the file contain information about when the cell was created and processed, technology and scale information, boundary descriptions of cells of which instances are incorporated, specifications of constraints on object placement, name equivalence information, and external connectivity information.

The symbolic representation used in PSI, like those in STICKS, CABBAGE, and REST, is similar to a circuit schematic, so such a schematic can be easily constructed from a PSI cell definition. However, each PSI cell also provides the geometrical information needed to define a physical layout, so that masks to make the circuit can be generated from the set of definition cells. Thus the PSI cells can easily accommodate the extraction of information for use in either logical or physical simulation.

The PSI programs can deal with multiple technologies, but they are technology-knowledgeable rather than technology-independent. They have specialized routines that embody knowledge of each type of device used in any of the technologies PSI can accept. A designer user makes entries in a small personalizing file to specify which technology is to be used for a particular design. By reading this file the programs determine which devices can occur in a particular design, and by reading tables in auxiliary files they determine rules on device dimensions and object spacings required by the technology.

This simple arrangement gives good flexibility in using PSI for different technologies within its scope.

Editor

FSE is an interactive graphical editor for preparing a symbolic representation of a circuit. Using FSE a designer sketches a circuit, showing the relative placement of all circuit components, and may also provide exact physical dimensions of components or component spacings which he wishes to specify. FSE has routines that can build any device used in any of several technology variants, and the designer need only specify this technology at the outset of a design. Four technology variants have been dealt with thus far, two NMOS and two CMOS technologies.

- *Displaying circuits*

FSE displays wires in a nonexpanded form as lines, but displays a point object in a form which shows its true size. When a cell is being edited, all its contents are displayed, but when an instance of a cell is used as a component of another cell, only an abstract boundary representation is shown. This representation consists of a bounding box and a set of terminals, where a terminal is an interval on some side of the bounding box to which an external connection can be made.

- *Editing cells*

More than one cell may be edited during a single session. When reference made to a cell during an editing session requires information about the contents of a cell, the cell description is automatically loaded from disk into main memory. When the designer moves to edit another cell, status information is preserved on all cells previously edited during the session, and the editing environment is restored upon return to a cell previously edited.

• *Creating objects*

An object is created by positioning the cursor at the desired position and pressing an appropriate key. In custom design the dimensions of objects may be varied to accommodate circuit requirements. PSI assigns default values to the physical dimensions and orientation of a newly created device, so the designer can defer entering these data until it is convenient. Terminals are created automatically when a wire is drawn to the boundary and automatically deleted if the wire is deleted.

Instances of cells used as components are created by a procedure that allows a final position adjustment after the cell instance is called forth.

• *Establishing connectivity of objects*

For interconnection purposes FSE, like CABBAGE, treats a transistor and/or contact as a *point object* located at a single point of the coordinate space. No matter what the actual size of a point object, all connections to it must be made to its "typical point" rather than to one of its edges. Natural conventions define which terminal a wire connected to an object is connected to.

Point objects (transistor, contact, terminal) must be connected by wires, and connections between objects and wires are deduced from coordinate values, so a wire and a point object are considered connected only if they have a common point on the same layer. Thus the possibility exists that two objects whose positions were entered by means of a graphical input device may appear connected without in fact having exactly the same coordinates.

FSE provides two modes for entering coordinates, which deal with the problem of establishing connectivity in different ways. When FSE is in grid mode of entry, any entered point is forced to lie on a grid of points that covers the design, with grid spacing such that neighboring grid points are visibly well separated. So long as all objects entered lie on the grid, points that seem connected are thus ensured to be so.

In many cases a design contains objects that are not ensured to be on the grid, either because the grid spacing has been changed or because they have been placed by the spacer. To ensure connectivity of objects or lines added to the design in such a case, FSE provides an object mode of editing. When the designer enters a point in object mode, the coordinates of the point "snap" to those of any object located within a critical interval of the cursor, this interval being shown graphically as a box which surrounds the center of the cursor. Thus, a point object is positioned exactly on a nearby wire, and a wire is caused to end exactly on a nearby point object.

• *Wiring*

Every wire must be either strictly horizontal or strictly vertical, but, when one enters the positions of the wire ends graphically, they are usually not perfectly aligned along a coordinate axis. In grid mode FSE draws the single wire, whether horizontal or vertical, that lines up with the initial point; in object mode FSE puts a jog in the wire if necessary to ensure that it connects to a point object at its end.

• *Workstations*

FSE is designed to run on either a raster color graphics system or a storage vector graphics terminal generally available in IBM. The vector storage terminal is called the 3277 Graphics Attachment. It consists of a 3277 text display terminal having an attached Tektronix 618 high-resolution vector storage tube in a dual-screen configuration, all connected via a high-speed connection to the 3081 host. The input device is a joystick. The terminal architecture is such that a keystroke may invoke a command and input coordinates at the same time if appropriate.

Symbolic objects are displayed in outline; the layer in which an object is implemented is shown by the line type of the outline.

The color graphics workstation is one we assembled from an Apollo DOMAIN node having 1M bytes of main storage, a local disk, and an attached Lexidata 3400 color graphics system. The workstation components are connected via a Multibus-compatible high-speed DMA interface, and the workstation communicates with IBM hosts via an RS232 connection. The graphics system has a resolution of 880 × 704 pixels and a refresh rate of 60 Hz noninterlaced, with four color planes, one text overlay plane, and a blink plane, and provides hardware pan and zoom. For graphics input a tablet with a four-button puck is used.

Both terminal and workstation are used in a dual-screen mode, one screen for display and graphical entry, the other for textual input and output. The basic software is such that the editor interfaces to the two are largely compatible. However, some functions are provided on the high-performance workstation that are not provided on the GA. Thus, commands may be given either through key presses, as on the GA, or via a hierarchical graphical menu making use of the puck and the puck buttons. As a result it would be straightforward to migrate FSE to a workstation using a single-screen display.

Spacer

The function of the spacer is to derive a physical layout for the design. To perform this function it must read design files in the external format created by the PSI Editor, construct the mask polygons implied by the symbols, create a graph that summarizes the constraints between object positions required by spacing design rules and designer-specified constraints, prompt for designer instructions regarding the compaction process, and then determine a placement of the objects that is tightly packed and design-rule-correct. During the compaction process, the spacer stretches or compacts the

interconnection components as necessary to preserve the topology and connectivity of the displaced objects. When the spacing operation is complete, it creates a new file in the external format describing the compacted layout. This file can now be modified by the Editor or used by the mask maker to produce mask descriptions in the language GL/I.

- *User commands*

The spacer may be with or without a graphics terminal. When used with a graphics terminal it provides several display and command functions that use graphical input. In particular, the designer can direct the program of compaction by specifying the sequence and direction of compaction steps.

Each compaction is parallel to either the x or the y axis. In any one compaction the designer specifies which axis the compaction is parallel to and the position toward which it should proceed, which may be toward either side or toward an internal line (implosive compaction). At least one compaction must be made parallel to each axis before design rule correctness is ensured.

- *Spacing of a nested cell*

When a cell is compacted, the boundary is contracted until it touches the objects inside. When a nested design is compacted, the spacer does not examine the objects inside the polygonal boundary of a cell used as an instance in a larger cell, so a space is left between the cell boundary and the edge of an external mask layer that is the maximum value of all the spacing rules observed by that layer in the design rules of the process technology.

- *Spacing rules*

The spacing rules in an IBM technology manual provide for many different spacings between a given pair of layers, depending on which objects the layers are in, so that the designer is free to use especially close spacings in all cases where the fabrication techniques permit. Typically there are 100–200 spacing rules in an IBM technology manual.

In developing the spacing rules for PSI, we simplified the rules considerably, so that PSI does not distinguish as many different cases as the full technology rules. Our present rules table involves 36 spacing rules for the NMOS and 51 spacing rules for the CMOS technology. If in future we found that particular configurations could be packed much tighter if special spacing rules applied between pairs of special regions other than those now recognized, we could incorporate the necessary code and increase the number of rules. However, we expect that PSI will always use simplified spacing rules.

- *Technology interface*

Technology rules regarding the spacings between objects on layers for a given technology are recorded in an external file as a table having rows/columns labeled by the edge types of the shapes that form the transistors, contacts, and interconnections, and entries equal to the required minimum spacings. We have used up to 51 such rules in the technologies we have dealt with so far.

Rules relating to the minimum size, enclosure, and overlap of the shapes that constitute the devices of a technology are recorded in another external file. These rules are also used by the editor and are enforced either at the time a device is created or at the time a circuit is spaced. We have used up to 39 such rules in the technologies we have dealt with thus far.

To specify the technology to be used in a design, a designer enters its name in a small personalizing file. To change the values of technology rules governing device structure or edge spacings, a technologist changes the entries in the technology tables; the changes are then enforced in the next processing of the design description. To change the specification of how a device is built out of shapes, a programmer modifies a module of the spacer code without other effect on the spacer. Thus the spacer as designed is readily adaptable to technology change.

Constraints on placement

The designer can specify two different kinds of constraint on object placements made by the spacer. One kind of constraint is a *fence*, as in Williams' STICKS. A user-plotted fence is a contour in the layout that no symbol element may cross during compaction. By specifying a set of fences the designer can require pieces of the design to remain bunched together through the compaction process. The other constraint is a requirement that the relative positions of a specified pair of symbol elements satisfy an equality or lower-bound or upper-bound inequality condition on their values.

- *Electrical connectivity*

In many cases, the spacing requirements for two mask polygons of the same type are different depending on whether or not they are electrically connected. Thus connectivity information must be extracted before assigning spacing constraints for mask elements. At present electrical connectivity is extracted by a special routine that will eventually be incorporated into the editor.

Each wire, contact, or cell-terminal is considered as a net node, while a transistor has three nodes corresponding to its gate, drain, and source. All nodes connected so as to have the same potential are assigned to the same net, and nets are given distinguishing numbers. The net tracer uses a fast scan line algorithm to find trees of connection, then traces the nets and assigns the proper net number to each object in the layout.

- *Building the constraint graph*

To place the objects in the layout the spacer solves a minimization problem on a weighted directed graph. Symbol

elements are mapped to the nodes and spacing constraints to the weighted edges of a directed graph. Then a longest-path search algorithm is used to find new locations of the elements such that the overall width of the layout in the compaction direction is minimized.

We describe the process for x-axis compaction of a constraint graph G: Objects are moved by expansion/contraction of interconnections. Each node in G corresponds to a group of objects which must move together. Each edge in G represents the spacing constraint between two groups. A group is formed by linking point objects and vertical wires which are physically connected on a common center line. Each cell instance, each vertical fence, and each boundary (right and left) of the prime cell is also considered a group.

After groups are mapped to the nodes in G, the spacer program assigns a spacing constraint to each edge in G by examining all of the rules/constraints constraining the pair of groups corresponding to that edge. A constraint may be of lower-bound, upper-bound, or equality type. A lower-bound constraint is expressed as a positive weight on a right-directed edge of G; an upper-bound constraint is expressed as a negative weight on a left-directed edge; an equality constraint is expressed as an upper-bound/lower-bound pair.

To build the graph for the layout the spacer determines what are the dominating constraints for each edge, considering both the user-specified constraints on selected pairs of symbols, and constraints deriving from the technology rules.

In principle, the number of edges in the graph is proportional to the square of the number of layout elements, but the number of edges that actually come into play in compaction is more nearly proportional to the number of elements for a large layout. That is because the distance within which spacing rules affect edges is similar in magnitude to the minimum line width of mask elements, so that most pairs of groups are automatically kept far enough apart because of the presence of groups lying between them.

Since the time and memory required to execute the compaction algorithm depend mainly on the number of edges in the constraint graph, it is important when building the graph to eliminate edges whose spacing requirements are satisfied automatically. This is done by the design-rule analyzer routine, which makes use of the concept of *updated layer-shells*.

There is a shell for each layer and a shell for fences. For compaction to the left, a shell is defined as the set of vertical edge segments on the right-side boundary of the union of all mask polygons of that layer that have already been examined. Initially, all the shells are empty, except the fence shell which contains the left boundary of the prime cell. As the analyzer successively examines groups from left to right, it determines the minimum spacing requirement between each group and other groups on its left and updates the shells.

As the analyzer examines a given group, it translates the elements in a group to the actual mask polygons, forms the left and right boundaries of every layer on the group, compares the edges in the shell to the vertical edges on the left boundary of the group, finds for each pair of edges having overlapped projections the correct spacing value required, and, finally by examining all spacing constraints affecting the edges of the group, obtains a spacing requirement between the group and the shell. In determining the constraints between two edges, the analyzer notes whether the corresponding layers belong to the same net and are allowed to merge.

After all left edges of a group have been checked, the design-rule analyzer updates the shells by adding to each shell the appropriate vertical edges on the right boundaries of the group, clipping out the shell edges of the same type, and merging the remaining shell edges to form new shells. The constraint graph is complete when each group has been examined and each necessary edge weight determined.

- *Symbol placement*

A graph-theoretic algorithm is used to find a placement satisfying the constraints of G and having minimum overall width in the compaction direction. Since mixed constraints are allowed, G can be cyclic, so the straightforward critical-path algorithm used in CABBAGE [2] cannot be used here.

We developed a new longest-path-search algorithm for cyclic graphs [10] that finds, e.g., in a leftward compaction, the lowest possible coordinate of each node in G consistent with the constraints of G. The algorithm does not in general find the layout with the minimum possible chip area. However, it quickly finds a fairly compact layout, which may be good enough to use or may be made so by manual modification by the designer, whereas an algorithm to find the layout of absolute minimum area would be so slow as to be impractical for a design of even moderate complexity.

By reversing the direction of all the edges in G, the same algorithm can compact the mask elements to the right boundary of the prime cell. For an implosive compaction, two compactions must be done, one to the right and one to the left; and since nodes corresponding to the groups on one side of the compaction center may interact with those on the other side, the designer must indicate which side is to be compacted first toward the compaction center.

- *Performance considerations*

In developing the spacing algorithm we considered speed of compaction important, but secondary to accuracy and function, so long as the spacer would be fast enough to compact the largest cell we needed to deal with in a few minutes. Adequate speed was ensured by designing all the critical algorithms to run in time proportional to $N(\ln N)$.

Probably the largest single cell one would want to make with a tool of this sort would have of the order 500–1000

devices. To test the speed of compaction of PSI we have run several circuits through the spacer with several cells of typical size. One was a personalized PLA we got from Steve Posluszny. For this circuit, which has 50 transistors and a total of 240 objects, the total compaction time was 0.89 CPU seconds on a 3081. We also compacted a layout of a tally circuit which we got from R. Montoye; this circuit had 480 transistors and a total of 2200 objects, and required nine CPU seconds. In addition we have compacted a number of circuits of various sizes prepared for us by Carol Gura and Marianna Clark. We found that the running time was approximately linear in the number of transistors. The high-speed memory required to compact larger circuits is about two megabytes. We estimate that with this much memory the spacer can compact a circuit with 1000 transistors.

As we add further features to PSI, we may complicate the work of the spacer and thus slow it down somewhat. Even so, we project that when in future we move PSI onto a fast workstation, the spacer will still be fast enough to be used as an interactive tool.

PSI maskmaker

The task of the maskmaker is to prepare from a symbolic description of a design a specification in the language GL/I of the masks needed to fabricate a circuit embodying the design. The standard GL/I representation accommodates nested cell descriptions but requires that the complete description of the design be in a single file. On the other hand, in the PSI representation descriptions of nested cells are in separate files. To build the GL/I description of a nested PSI design, one must first make a GL/I description for each individual cell, next assemble these descriptions into a single nested description of the whole design, and finally do any postprocessing needed to adapt to the notational conventions of the designer and to adjust the ideal mask shapes for the biases of the fabrication process.

The PSI maskmaker creates the GL/I descriptions of individual cells and combines these into a single cell describing the nested design. It accommodates a certain amount of technology-dependent and user-idiosyncratic level naming by use at mask-writing time of user-provided tables for translating level names. This permits, for example, combining several symbolic levels into the same physical mask level and the use of a variety of different naming conventions. It provides means to attach logical attribute information to particular shapes for later transfer to physical-to-logical checking tools or other final audit requirements. However, it does not perform postprocessing functions that are specific to any fabrication process; these functions must be done using other IBM design tools.

Discussion

In developing PSI we have adhered to the simple style of the early prototype symbolic layout tools as much as seemed practicable. However, as we worked with potential users of the system, who have assisted us in debugging and evaluation, we have decided that some features of PSI would need to be more complex. The present version of PSI meets many but not all of our original goals, so that further work on PSI is seen as necessary.

We discuss next our current ideas about the relative importance of various system features in a symbolic layout tool like PSI. We do this by reviewing features of PSI introduced to meet the needs of the designers we worked with and discussing other features not yet implemented that we believe are important to provide. As of this writing PSI has not had the extensive designer use needed to justify our choices of design features or to quantify the quality of design that PSI can produce, so the evaluations made thus far are based mainly on our impressions from designer use in a limited number of test cases.

- *Placement of contacts*

The simple STICK representation in which symbolic objects are connected by lines does not provide any natural way to connect to a wire at a point off center. However, off-center connections are useful in many circuit configurations. In PSI we provide the capability to construct such connections via a special point object called a connection strip. This object is essentially a flange that may be attached to a transistor or line and provides it with a visible dimension along which contacts can be placed.

The optimal shape and position of a buried contact are sensitive to the configuration of wires coming into it. PSI does some optimization of the dimensions and shape of a buried contact, but uses simplified spacing rules for placement. Since a buried contact uses a rather large area, it may be worthwhile in future to refine the procedure for constructing and placing these contacts.

- *Technology rules*

We require not only that PSI be usable with a number of related circuit technology variants, but also that it provide means to rapidly update a design to reflect recent changes in the design rules of a given circuit technology. By localizing technology-specific data in tables accessed at run time and localizing code that is specific to a technology or device in its own module, where it can be readily located and changed when necessary, we arrange that changes of rules can be made effective very quickly. This permits PSI to correct a spacing error immediately and to correct any dimension that would be below a required minimum. However, PSI cannot make coordinated changes to ensure that such conditions as electrical load matching or operating point are maintained.

The design rules for the circuit technologies we deal with provide for many different spacing rules between layers, depending on just what objects the layers are in, so that a designer can make the best possible placements of circuit

elements. In order to simplify the task of the spacer we have defined for PSI a set of design rules that are simpler than the complete technology rules, although more complex than, e.g., the Mead/Conway rules. We have no principle on which to determine which specialized design rules to keep: We simply selected what looked to be the cases where the most area was at stake and devised special rules to deal with them. We intend to add more rules if we find circuit configurations for which new rules would provide significant area savings. However, we judge that there are some technology rules that do not seem to be worth incorporating in PSI when we consider the cost of checking out that they are correctly dealt with in every situation.

An important special case of the point just discussed arises where two neighboring shapes are at the same potential. Two shapes on the same layer and at the same potential can often be merged and a good area saving realized. PSI currently performs such merges by initially suspending the requirement for a minimum spacing between the shapes, and then later checking whether a merge actually occurred and enforcing minimum spacing rules where it did not.

However, we found that if we make merges that are maximally aggressive, we may adversely affect the result of a subsequent compaction in the orthogonal direction. We are in the process of developing a merge procedure that gives a better trade-off between area squeezed out along the axis of compaction and the potential to squeeze out area in the orthogonal direction.

• *Hierarchy*

To be able to call cells as components of other cells makes a symbolic layout tool more broadly useful. However, the capability to nest cells, especially cells described in separate files, creates the following problems to be solved and capabilities to be provided that are not present when one deals with single-level cells:

1. PSI implements compaction at a single level of description; i.e., it treats every embedded cell instance as a rigid object. If one wishes to reshape a nested cell during compaction, one must de-nest it. It may be awkward to properly compact some designs under this limitation, but we think it would overcomplicate PSI to incorporate the capability to do multi-level spacing in the present notational framework. This is one of a number of subtle questions that require more study.
2. PSI presently compacts a nested design using a maximum design rule condition. This rule results in very poor area utilization for a design that has many small cells nested in it. In the next version of PSI the spacer will have information about the interior of an embedded cell instance so that it can use the space adjacent to the cell more efficiently.
3. It is often the case that one wishes to abut two cells that are compacted separately. At present there is no way for PSI to ensure that when these cells are abutted the design rules will be satisfied. The next version of PSI will permit the designer to tell the spacer that abutment of the two cells is planned and provide it information about the other cell needed so that it can ensure satisfaction of the design rules.
4. Designers need to attach names to terminals, nets, and sometimes devices; and in the case of net names it is necessary that the same name be associated to every node on the net at all levels of hierarchy. PSI presently allows one to assign a name to an object and it captures the minimum information needed to propagate names through the net. However, some new program may be needed to propagate names through a hierarchy in a fully satisfactory manner.
5. It is important to be able to incorporate into a cell being designed at the symbolic level an instance of a cell for which only a physical description is available and to deal with the incorporated cell efficiently in compaction. This capability too is planned for our next version of PSI.

• *Jogs*

Psi provides the capability to insert into a symbolic design pseudo-objects that tell the spacer where the designer wants to put a jog in a single line to achieve a more compact placement. This may prove to provide insufficient flexibility: Where one wants jogs in multiple parallel lines jogged, a more complicated implementation would be convenient. We need more design experience to determine whether multi-line jogs are necessary.

• *Non-grid-based editing*

Placement of all objects on a coarse coordinate grid can result in a substantial waste of area where customized device sizes are being used, so PSI has no such restriction on its placements. Thus, given our decision to compact at only one hierarchical level at a time, it is necessary to provide the object mode to edit devices. This might not be a requirement in other systems.

• *Control of compaction*

Most symbolic layout systems provide the designer a means to specify constraints on the placements the spacer makes, e.g., that a cell be of exactly a certain width, that its terminals have a specified separation, that certain components be in a particular configuration, etc. We have incorporated or plan to incorporate most of the capabilities of this sort that we know of. We have implemented or are in the process of implementing the following constraint capabilities:

1. Designer specifies constraints of equality or inequality between pairs of objects, including cell boundary objects.

2. Designer defines fences to cause subsets of cell components to be kept in a close group during compaction (as in J. Williams' STICKS).
3. Designer specifies a program of compaction, specifying as individual steps a dimension of compaction, X or Y, the side from which compaction should proceed, and the line toward which compaction should proceed, either a cell boundary or an interior line.
4. System captures a compaction sequence and can rerun it in batch mode (to be implemented).
5. Designer specifies that a cell is to be compacted so that if it is abutted to a specified cell, no spacing errors occur (discussed above).

• *Parameterized cells*

The power of a symbolic system would be greatly increased if one could define a generic cell, some of whose dimensions are parameters that are only specified when an instance of it is created, and could assign values to these parameters by means of a program that processes the design. We considered implementing such a capability, but decided that it would make PSI too complex, given the system framework as now defined. We have defined a limited type of parameterization capability that seemed feasible in our system framework, but have not as yet decided to implement it.

Acknowledgments

We wish to express our appreciation to a number of people in IBM who have helped us in various ways with the development of PSI: C. Wong for discussions of algorithms, B. Greene, B. Steinman, S. Posluszny, and D. Cummings for discussions of system implementation considerations, and C. Gura, J. Beckwirth, M. Clark, and R. Lembach for their help in system test and evaluation.

References

1. J. D. Williams, "STICKS—A New Approach to LSI Design," Master's Thesis, Massachusetts Institute of Technology, Cambridge, MA, June 1977.
2. M. Y. Hsueh, "Symbolic Layout and Compaction of Integrated Circuits," *ERL Memo UCB/ERL M79/80*, University of California, Berkeley, December 1979.
3. N. Weste, "Virtual Grid Symbolic Layout," *Proceedings of the 18th Design Automation Conference*, June 1981, pp. 225–233.
4. R. C. Mosteller, "REST: A Leaf Cell Design System," VLSI'81, J. P. Gray, Editor, Academic Press, Inc., New York, 1981, pp. 163–172.
5. M. W. Bales, "Layout Rule Spacing of Symbolic Integrated Circuit Artwork," Master's Thesis, University of California, Berkeley, May 1982.
6. K. H. Keller, A. R. Newton, and S. Ellis, "A Symbolic Design System for Integrated Circuits," *Proceedings of the 19th Design Automation Conference*, June 1982, pp. 460–466.
7. R. J. Lipton, S. C. North, R. Sedgewick, J. Valdes, and G. Vijayan, "ALI: A Procedural Language to describe VLSI Layouts," *Proceedings of the 19th Design Automation Conference*, June 1982, pp. 467–474.
8. R. K. Montoye and P. W. Cook, "Automatically Generated Area, Power, and Delay Optimized ALUs," *Digest of Technical Papers, IEEE International Solid-State Circuits Conference*, February 1983, p. 132.
9. W. L. Schiele, "Improved Compaction by Minimized Length of Wires," *Proceedings of the 20th Design Automation Conference*, June 1983, pp. 121–127.
10. Y. Z. Liao and C. K. Wong, "An Algorithm to Compact a VLSI Layout with Mixed Constraints," *IEEE Trans. Computer-Aided Design* **CAD-2,** 62–69 (April 1983).

Received February 1, 1983; revised April 26, 1984

Edward Adams *IBM Research Division, Yorktown Heights, New York 10598.* Dr. Adams received the B.S. in chemistry from Southwestern University at Memphis, Tennessee, in 1943, and the M.S. in physics (1947) and Ph.D. in theoretical physics (1950) from the University of Wisconsin at Madison. Dr. Adams is manager of VLSI symbolic layout tools in the Computer Science Department. He joined IBM in 1959, and has served twice on the staff of the Director of Research and as research director of engineering science, research director of systems and applications, and research director of computer-aided instruction. He has served on the faculty of the University of Chicago, in the Department of Physics and in the Fermi Institute, and has been a member of the Physics Department of Chicago Midway Laboratories. He served as research manager in solid state physics and semiconductors at the Westinghouse Research Laboratories, as visiting faculty at the Carnegie Institute of Technology, Pittsburgh, Pennsylvania, in physics, the State University of New York at Stony Brook in physics and engineering, at the City University of New York Graduate Center, New York, in educational psychology, and at the California Institute of Technology, Pasadena, in computer science. Dr. Adams is an original Atomic Energy Commission Fellow, 1948–1950, and a Fellow of the American Physical Society. He is a member of the Association for Development of Computer-based Instruction Systems, the Institute for Electrical and Electronics Engineers, and the International Federation for Information Processing.

Rolf-Dieter Fiebrich *Thinking Machines Corp., 577 Beaver Street, Waltham, Massachusetts 02154.* Dr. Fiebrich received his B.S. in electrical engineering (1973) and his Ph.D. in computer science (1977) at the Technical University of Munich, Federal Republic of Germany. He served as Assistant Professor of Computer Science at Ludwig-Maxmilian University, Munich, from 1973 to 1979 and as manager of data communications for the Leibniz Computing Center of the Bavarian Academy of Science, Munich, from 1978 to 1979. From 1979 to 1983 he was a Research staff member at the IBM Thomas J. Watson Research Center, Yorktown Heights, New York, working with software engineering and the development of symbolic layout tools.

George Koppelman *IBM Research Division, P.O. Box 218, Yorktown Heights, New York 10598.* Mr. Koppelman is a Research staff member in the Computer Sciences Department at the IBM

Thomas J. Watson Research Center. He received his B.S. degree from the University of Chicago, Illinois, in 1961. He joined IBM at the Watson Research Center in 1965 and has worked in pattern recognition, graphics, and interactive document scanning and, from 1981 to 1983, as a member of the VLSI design tools area in Josephson development, working on circuit simulation, design automation tools for master image logic chips, and in the microprocessor design group.

Yuh-Zen Liao *ECAD, Inc., 3255 Scott Boulevard, Santa Clara, California 95051.* Dr. Liao received the B.S. (1971) and M.S. (1973) degrees in electronic engineering from the National Chiao Tung University, Taiwan, China, and the Ph.D. degree in electrical engineering (1980) from the University of California, Berkeley. From 1980 to 1982, he was a senior staff engineer at the Link Flight Simulation Division, Singer Company, where he developed algorithms for removing aliasing effects in computer-generated moving images. From 1982 to 1984, Dr. Liao was a Research staff member at the IBM Thomas J. Watson Research Center working on an automatic compactor for VLSI symbolic layout, with research interests including image processing, computer graphics, and computer-aided design of VLSI layout.