THE "PI" (PLACEMENT AND INTERCONNECT) SYSTEM*

by Ronald L. Rivest**
MIT Laboratory for Computer Science
Cambridge, Mass. 02139
March 1982

Abstract

"PI" is an advanced LISP-based placement and interconnect system for custom NMOS or CMOS (single-layer metal) designs. When fully implemented, PI will handle placement of arbitrarily-sized rectangular modules, routing of power and ground, signal routing, and compaction. In this paper we briefly review the structure of PI, and present details on the signal-routing heuristics, focusing on the definition of "channels", the global router, the "crossing placer", and the channel routers. The signal router is fully operational; the rest of PI is currently being coded and will be more fully described in later papers and theses.

Objectives

Our goal is to completely automate the placement and interconnect phases of IC design for custom NMOS or CMOS designs created in the "Mead-Conway" style [MC80]. We take as a model an optimizing compiler for a high-level programming language: PI should quickly produce high-quality output with a minimum amount of human interaction.

Working Assumptions

- The fabrication technology is single-layer metal NMOS or CMOS. This assumption introduces difficulties that an extra layer of metal would alleviate. For example, the routing of power and ground would become trivial, and the signal router wouldn't have to worry about minimizing the length of wires in the non-metal layers.
- The current-carrying limits of the metal layer are significant for the power/ground routing (wide buses may have to be used), although not for the signal routing (minimum-width wires can be used).
- The logic modules and pads can be adequately modelled as rectangular "black boxes" having pins on their perimeters. The pins may be in arbitrary locations and on arbitrary layers. (See "Input to PI".)
- I/O pads must be placed on the periphery of the completed chip.

Major Design Decisions

- Like most compilers, PI is basically a "batch"-type system. Our objectives are to explore the limits of what can be accomplished algorithmically, not to provide "automated drafting tables" for a human designer. As a consequence, PI does not provide means for a designer to partially specify a placement or a routing. (However, he may specify an entire placement or an entire power/ground routing, if we wishes, and can also specify just the placement of the I/O pads.) But note that PI can run within "DPL" [BMSSW81], which provides many of the interactive capabilities that PI lacks.
- PI uses channel routers ([HS71], [Hi74], [PDS77], [So81], [RF82]) to do the detailed routing. This commonly-used technique is quick and highly effective. Note below, however, that PI uses a non-standard notion of "channel" – the width of the channel is known before channel-routing is begun.
- PI works with absolute coordinates; there are no "symbolic" channel widths to be computed after the signal routing is completed. This decision was made to achieve a high-quality result; we feel that maintaining symbolic positions compromises too much quality of the final product. Of course, a couple of iterations of the signal router may be needed to find the maximally compact result, or to even find a solution if there were unroutable channels. PI will automatically resize channels for successive iterations.
- PI is not a "hierarchical" router, although the placement phase may involve some explicit manipulation of the design hierarchy.
- PI is implemented in LISP (i.e. MACLISP or LISP-Machine LISP [WM81]). There are three reasons for this choice: (1) LISP is an excellent language for quickly developing prototype systems, due to its object oriented nature, flexible macro facility, and run-time interpreter. (2) The "DPL" IC design system

* This research was supported by DARPA grant N00014-80-C0622, Air Force grant AFOSR-F49620-81-0054, and NSF grant MCS-8006938.

** The following people have participated in the design and development of the PI system: Alan Baratz, C.S. Chow, David Christman, Ali Ghaznavi, Alain Hanover, David Jilk, Jim Koschella, Andy Moulton, Ron Pinter, Ramana Rao, Flavio Rose, Gerry Roylance, Alan Sherman, Susmita Sur, and Alok Vijayvargia.

[BBSSW81] is implemented in LISP Machine LISP; using LISP allowed us to interface easily with this sytem for graphic I/O, module specifications, etc. PI is thus one of the many tools available in this design system. (3) LISP is by far the best-supported and best-documented programming language at MIT.

- PI works only with rectangles oriented parallel to the axes. There are no "45°" wires, and all modules are rectangles that will be placed parallel to the $x-y$ axes.
- PI will not route nets "over" unrelated modules (it can't, since it doesn't know the wiring inside a module), although it may route a net "through" a module, if that module has more than one pin for that net.
- PI will treat signal nets uniformly; there is no way to mark any signal lines (including clock lines) for special treatment.

High Points

PI incorporates a number of innovative features:

- A placement algorithm that gracefully combines "min-cut" and "bottom-up" approaches.
- An effective technique for producing compact single-layer interdigitated trees for power/ground routing.
- A novel "channel-definition" algorithm that produces large, natural-looking channels.
- A "crossing-placer" that runs after global routing is completed that determines exactly where each net crosses every channel-to-channel edge. This permits a global approach to reducing channel-routing complexity, and makes the channel-routing problems entirely independent.
- A novel "channel-router" (which could also be called a "switchbox router" [So81]) based on a left-to-right scan of the channel [RF82].

System Organization

PI divides the "silicon compilation" process into the following steps:

- Input the problem specification and check its validity.
- Place the logic modules.
- Route the power and ground buses.
- Route the signal nets, as follows:
  Channel Definition: Divide the area of the chip outside of the modules (including the power/ground buses) into non-overlapping rectangular *channels* in which the wiring will be done.
  Global Routing: Decide for each signal net the set of channels it will pass through, thus determining the approximate path for each net.
  Crossing Placement: Decide exactly where, and on which layer, each net will cross from one channel into the next. This phase, unique to PI, attempts to globally minimize signal line crossover and jogging. It also decomposes the remaining routing problem into the *independent* subproblems of routing each channel.
  Channel Routing: Route each channel. To maximize the completion rate PI has several channel routers.
- Resize Channels. If the routing produced was satisfactory, stop. Otherwise, resize the channels to enlarge over-congested channels (ones that could not be routed) and to shrink overlarge channels (ones that had too much empty space). Restart with a new global routing phase.

In the following sections we describe each of the phases a little more precisely.

Input to PI

The designer using PI specifies a layout problem by describing:

- The logic *modules*: their size (length and width, since they are rectangles), terminals, pins, power consumption, and whether they are logic blocks or I/O pads.
- The signal *nets* to be routed,
- The desired *size* of the completed chip.

A *pin* on a module is specified as a rectangle of material (on some layer – e.g. metal, poly, or diffusion) inside the module. Typically, the pin is a degenerate rectangle (a point) on the border of the module. If it is not on the border, PI will add direct wiring from the point to the nearest edge of the module (or another edge if the designer so specifies). This facility is useful with modules that are only approximately rectangular in shape; the pins are automatically extended to lie on the "bounding box" of the module. If the pin is a true rectangle (and not just a point), PI may be able to connect to the pin in several ways (e.g. the rectangle might cover one whole side of the module – as is common for power and ground), giving additional flexibility.

A *terminal* is specified as a set of pins on a module; usually just one pin is given. In contrast to the physical notion of a pin, the notion of a terminal is a logical one. It is possible to specify several pins as alternative choices for that terminal; PI will choose the most convenient pin. Since the pins of a given terminal are assumed to be electrically connected inside the module, PI may route a signal net *through* a module by entering a terminal via one pin and leaving through another.

A signal *net* is specified as a set of terminals to be interconnected.

Similarly, power and ground nets are specified as part of the input to PI. The specification lists for each of power and ground the terminals that must be joined together.

Since only the signal routing stage is now implemented, the user of PI must currently specify a module placement and power/ground routing. (The placement and power/ground routing algorithms have been designed and should be implemented within a couple of months.)

Placement

The placement heuristic uses a combination of "min-cut" ([Br77], [La79]) and "bottom-up" (successive pairing) approaches. During placement PI maintains a tree indicating the current decomposition of the chip into nested rectangular regions. The leaves of this tree are the modules being placed.

For the placement of the logic modules, at each step a node of the tree is selected, and it is "refined" by either a min-cut step or a bottom-up step. These steps continue until all modules are placed. (The I/O pads are placed separately by a special-purpose *ad hoc* routine.)

A min-cut step will divide the associated region in two and assign the logic modules of the region to each subregion in a way that attempts to minimize the number of wires cut by the dividing line. This technique is similar to the approach suggested by Lauther [La79]. The min-cut paritioning algorithm used is due to Fiduccia and Mattheyses [FM82].

A "bottom-up" step identifies two modules that "fit well together" and combines them to form a "supermodule" replacing the two component modules in the tree. The purpose of this routine is allow PI to discover and take advantage of situations where two modules have been designed to fit nicely together – e.g. a bus may leave one module, cross a small channel, and then enter the other module. By carefully aligning these modules the bus can be run straight-across the channel. PI later takes these connections into account when resizing channels, so as not to disturb this nice alignment if at all possible.

Figure 1 illustrates the way the decomposition tree is modified by a min-cut step on node X; Figure 2 illustrates how a bottom-up step combining modules A and B affects the tree.

The placement heuristics will be more fully described in later papers.

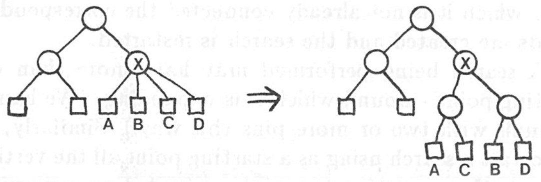


Figure 1. Min-Cut step

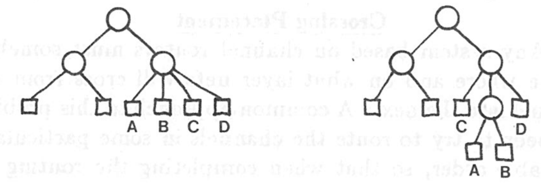


Figure 2. Bottom-Up step

Power/Ground Routing

Given a placement of the logic modules and I/O pads, the next step is to route the power and ground nets entirely in the metal layer.

The key observation in our power/ground routing heuristic is that to each possible power/ground routing structure there is a unique Hamiltonian path through the modules that does not cross the power or ground supply trees. (We assume here that each module has one pin each for power and ground; this assumption is easily removed.) This correspondence permits the use of heuristics for finding short Hamiltonian paths to be applied to this problem. The distance metric used should be a good estimate of the actual amount of wiring needed to connect both power and ground from one module to the next.

This heuristic will be more fully described in later papers.

Figure 3 illustrates the key observation showing a Hamiltonian path through modules A, B, C, and D.

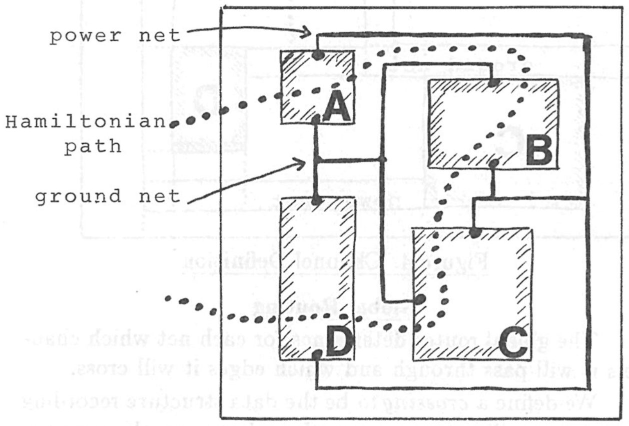


Figure 3. Power/Ground Heuristic

Channel Definition

The chip is partitioned into nonoverlapping rectangules, each of which is either

- a module,
- a routing region free of any previous logic or wiring (a "free channel"), or
- a routing region covered by metal (under a power or ground bus) but free on the other layers (a "covered channel").

Our definition of a *channel* is nonstandard since our channels have fixed height and fixed width, and may have fixed pins on all four sides to be connected up. Such channels have also been called "switchboxes" [So81].

The channels are separated from each other and from the modules by horizontal and vertical *edges*. Each edge is on the border of exactly two distinct channels or modules, unless the edge is on the border of the chip.

The channel-definition heuristic tries to produce a channel structure that minimizes the sum of the lengths of these edges. This heuristic tends to give large, "natural"-looking channels.

This phase also identifies "narrow" channels: channels so narrow that no jogs are possible in them; wires can only run straight across these channels.

This original channel structure is then refined so that each covered or narrow channel is enclosed by exactly four edges. This simplifies the channel assignment phase of the system.

Figure 4 illustrates the result of channel definition on a small example.

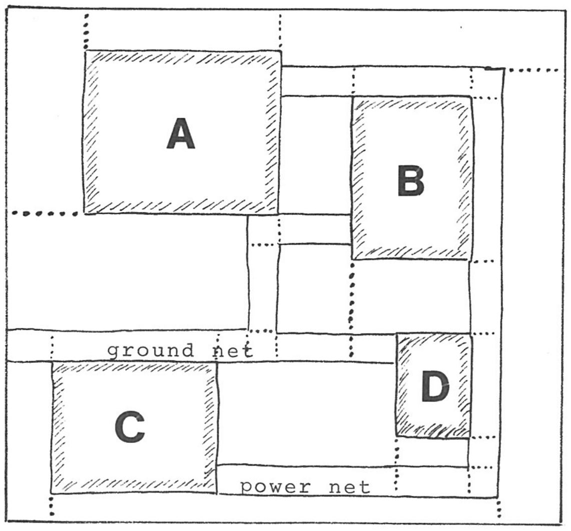


Figure 4. Channel Definition

Global Routing

The global router determines for each net which channels it will pass through and which edges it will cross.

We define a *crossing* to be the data structure recording that a net will cross a given edge. Later on the *crossing placer* will assign each crossing an actual position on the edge, as well as a layer.

We define a *strand* to be the data structure corresponding to a connected portion of a net which lies within a given channel. (Note: a net may have more than one strand within a channel, although this is unlikely.) Each strand specifies which net it is part of and which crossings it connects. For strands connecting more than two crossings the channel router will have to produce a tree-like wiring within the channel. The branch points of the tree are *Steiner points*.

The output of the global router thus consists of strands and crossings.

The global router processes the nets in order of increasing length of the perimeter of the bounding box of the net's terminals divided by the number of terminals of the net, so a many-terminal net may be routed early, although in general the nets are routed approximately in order of increasing net length.

The global router uses an explicit graph representation of the pin/channel/edge structure. The vertices of this graph are (1) the mid-points of the edges in the channel structure, together with (2) the pins of the net currently being processed. Two vertices are adjacent if and only if they lie on a common channel. Figure 5 illustrates a graph created for a simple routing example for a net having 3 pins on module A and 2 on module B.

PI estimates the *length* of each arc in the graph as the rectilinear distance between its endpoints, plus penalties for turning corners, traversing covered channels, or reaching an edge that is nearly "full" of crossings.

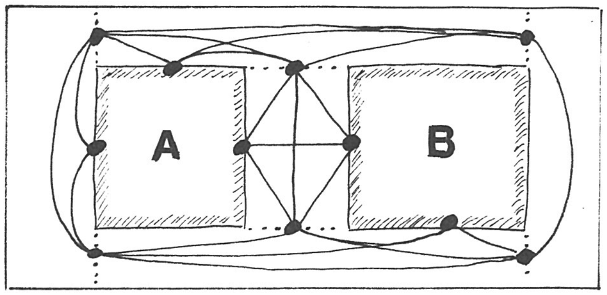


Figure 5. Graph for Global Router

In this graph PI then finds for each net a tree which connects together at least one pin from each terminal of that net and which approximately minimizes the sum of the arc lengths of the tree (the estimated net interconnect length). The arcs found determine which *channels* the net traverses, the vertices of the subgraph determine which *edges* it crosses. For each arc of the path a strand is created, for each vertex a crossing.

For two-pin nets such a tree is the shortest path between the two corresponding "pin vertices". The routing of two-pin nets is therefore performed by Dijkstra's shortest-path algorithm, beginning at one pin and searching until the other is reached.

For multi-pin nets the desired tree must have Steiner points. Here the search is begun simultaneously at all pins. These searches expand outwards from each starting point like waves from a number of pebbles dropped simultaneously into a pond. When one search finds another pin to which it is not already connected the corresponding strands are created and the search is restarted.

A search being performed may have more than one "starting-point" around which it is expanding. (We handle terminals with two or more pins this way.) Similarly, we will restart a search using as a starting point all the vertices on the path connecting two pins connected on a previous iteration.

The global router is described in more detail in [Ba81].

Crossing Placement

Any system based on channel routers must somehow decide where and on what layer nets will cross from one channel into the next. A common approach to this problem has been to try to route the channels in some particularly favorable order, so that when completing the routing for one channel it becomes determined where the nets from that channel cross into the next. This approach has at least two well-known difficulties: the router may make the problem of routing the next channel unnecessarily difficult by injudiciously choosing the crossing points when completing the current channel, and (worst of all) it is not very difficult to create channel structures for which no "particularly favorable" routing order exists. One is often left with the situation of routing a channel which has the crossings fixed on three or four sides, whereas the usual channel router can only handle channels with crossings on two opposing sides.

Our point of view is that this "crossing placement" problem is too important to be left to the lowly channel router. The choice of crossing placements has a major impact on overall wire length, number of vias used, and the routability of each channel. We feel that a separate, global determination of crossing placements is the best path to successful, high-quality routings.

The benefits to be gained by a global crossing placement routine are an increased likelihood that each channel routing problem will be simpler, and an expected decrease in overall wire length and number of vias used.

A potential disadvantage of this approach is that *every* channel now becomes a "switchbox" [So81] with crossings fixed on all four sides; new, more capable channel routers are thus required. It is also possible that by fixing the crossing positions one might so constrain the solution space that certain channels become unroutable that otherwise might be routed. The crossing placement routine is designed to minimize this likelihood.

The crossing placer is a unique innovation of the PI system. PI tries to organize the crossings in a "global" manner that will maximize the probability that the channel router will succeed on each channel and will be able to produce a routing that minimizes the amount of wire in the non-metal layers. PI also tries to maximize the number of crossings that will be "facing" each other across a channel so that the corresponding strand can be implemented as a straight run across the channel.

A further advantage of including an explicit crossing-placer is that it cleanly decomposes the remaining problem into a set of *independent* channel-routing problems.

The input to the crossing placer is the set of edges of the channel structure, each of which contains a number of crossings. An edge between a module and a channel already has its crossings placed, since these crossings represent the module's pins. An edge between two channels has "floating" crossings to be placed by the crossing placer. Note that each floating crossing will belong to two strands - one for each channel the edge is on.

The crossing-placer uses an iterative relaxation method where every edge is considered in turn and the crossings on that edge are adjusted in position. Initially every floating crossing is placed at the center of its edge. At each step a new position for a crossing is computed as the weighted average of the old position for the crossing and the projections of the other crossings on the same strands onto that edge. Crossings close to the crossing being placed are weighted heavier than those further away, and a crossing which is "facing" the edge is weighted much heavier. After a new position for each crossing on the edge is computed the positions are "discretized" so that they lie on a "global grid" whose spacing equals the standard track-center-to-track-center distance. (The only exception is that a crossing may be off the global grid if it is facing a pin of a module.) Crossings which face each other across narrow and/or covered channels are treated as a unit for purposes of crossing placement, since we want to guarantee that they will always be directly opposite each other.

The above simple procedure has the effect of "sorting out" bus structure in the desired manner; after several iterations of the global crossing placement procedure the wires of a bus will be ordered in the same way that they leave the modules, forming a nice clean "bus" structure. This works well whether the bus is a logical part of the design or just an accidental grouping of wires that follow a common route.

The layer-assignment phase of the crossing-placer assigns the metal layer to all floating crossings, except for crossings which are on an edge of a metal-covered channel – these crossings will be assigned poly or diffusion. Furthermore a crossing which is connected to a non-metal pin across a narrow channel will be assigned the same layer as the pin.

Figure 6 illustrates a typical "crossing-placement" situation. Here crossing X will be attached to crossing (pin) A with a strand in channel R, and to crossings B and C with a strand in channel S. The new position for X will be a weighted average of positions A', B', C' and X.

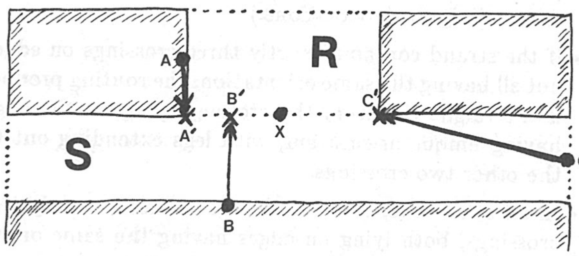


Figure 6. Crossing Placement

Channel Routing

At this point each channel is a self-contained routing problem: the channel has a definite size and shape and there are a number of crossings of fixed position and layer on its perimeter. Furthermore, a number of "strands" have been defined for the channel which indicate, for each strand, which crossings need to be connected together with wiring internal to the channel. While some of the crossings may correspond to pins on modules and others to floating crossings that have been placed, at this time they can all be treated on an equal footing.

The wiring must be placed internal to the channel in a way that will not interfere with wiring in adjacent channels or modules; the wires may not be near the edge of the channel unless the wire is running to the edge of the channel to make a connection to a crossing.

We believe that the channel-routing problem is best attacked with a variety of routers with different philosophies and capabilities. In this way we can maximize the chances of a successful routing by trying each router in turn on the channel until one succeeds.

At the moment we have implemented three routers: a "quick and dirty" router, a slower, more sophisticated "slice" router, and a "Lee"-type router [Le61] that is rarely used. We find that the "quick router" succeeds 50-80% of the time, decreasing the overall running time.

Before routing the channels, each channel is assigned a "preferred direction" (vertical or horizontal) giving the direction in which the metal wiring will be run; wires in the other direction will be in poly or diffusion where they cross the metal wires. This is chosen so as to maximize the amount of metal wiring used.

The "Quick" Channel Router

This router tries a quick but naive strategy for routing and then does a simple design-rule check to see if it worked. It fails in many simple situations, such as when there exists a strand connecting more than three crossings.

The routing strategy is the following:

- If the strand contains exactly two crossings which face each other across the channel, the natural straight-across routing is produced.
- If the strand contains exactly two crossings, one on a horizontal edge and one on a vertical edge, the quick router creates the natural "L" routing. (This is independent of any other wiring already placed, so this may create design-rule violations.)
- If the strand contains exactly three crossings on edges not all having the same orientation, the routing produced is a straight run from the crossing lying on the edge having unique orientation, with legs extending out to the other two crossings.
- In all other cases (i.e. the strand contains exactly two crossings, both lying on edges having the same orientation, where the crossings are not directly opposite each other, or the strand contains exactly three crossings all lying on edges having the same orientation) there is a need to find a "jog track" where a portion of the routing can be placed. These jog tracks are determined in a way that minimizes the chance that a design-rule violation will be produced (i.e. by looking for an empty track). Furthermore, they are found in a way that attempts to minimize the number of wire crossovers produced for the common "river routing" situations. Once the jog track is found the routing for the strand is straightforward.

Once all the wire segments are placed, they are assigned layers in a way that maximizes the amount of metal wiring used. A single long segment may be assigned different layers along its length to achieve this objective.

Finally, a design-rule check is used to see if there are any conflicts between the routings produced for the individual strands. If so, the quick router reports failure.

The "Slice" Channel Router

The "Slice" channel router is a more powerful and effective channel router. At present, if the quick channel router fails, we depend primarily on the "slice" channel router to produce a successful channel routing. This router is based on the "greedy" philosophy espoused in [RF82], which proposes a left-to-right, column-by-column (or "slice-by-slice"), scan of the channel, routing each column completely before proceeding to the next.

The basic components of this router are:

- A "jiggler" which can be used to adjust the track positions of the wires as they enter on the left or right.
- An iterative "slice router" that successively routes within the sequence of vertical "slices" that the channel is broken into. A given slice may contain connections to crossings on the top and/or bottom of the channel.

The router tries to avoid "conflicts" (when the wires extending from pins opposing each other in the channel would have to cross) by making small local jogs to the wire as it enters the channel, so that the conflicting pins are in essence no longer opposing. This will succeed if the density of opposing pins is not too great. More precisely, a conflict occurs when two nets enter the channel at the same horizontal position – one from the top and one from the bottom – where the tracks in the channel already assigned to these nets are such that the track for the net coming in from the bottom is above the track for the net coming in from the top. Conflicts are avoided by assigning the nets of the conflict to different slices; small horizontal jogs may be made by these nets from the top and bottom as they enter the channel. Conflicts are also avoided by the techniques given in [RF82], where nets may occupy more than one track for several slices.

This router is described in more detail in [Ko81].

Resize Channels

If PI fails to route all of the nets successfully, the user will be notified that some set of channels and/or edges is over-congested and more space must be provided. PI will then modify the placement accordingly by resizing the channels and restart the routing. In practice several such iterations may be required before a final layout is obtained. Resizing of channels is also used to compact the layout. The resizing routines are particularly careful not to shrink any covered channels too much (these channels form the power/ground bus structure, after all), and not to misalign any modules that were paired by the "bottom-up" placement routines (these placements enable many nets to run straight-<across from one module to the next).

Results

The current implementation is about 200 pages of LISP code.

The PI system has been used to route several project chips at MIT. On a typical sized project chip (e.g. 10 modules and 100 nets) a routing is obtained in 3-4 minutes on the LISP machine. We feel that the system produces high-quality solutions, and look forward to having the complete system up and running (i.e. placement as well).

Figure 7 illustrates the performance of PI on a small "toy" example.

References

[Ba81] Baratz, A. E., "Algorithms for Integrated Circuit Signal Routing," Ph.D. Thesis, MIT Department of Electrical Engineering and Computer Science. (August 1981).

[BMSSW81] Batali, J., N. Mayle, H. Shrobe, G. Sussman, and D. Weise, "The DPL/Daedelus Design Environment," in *VLSI 81, Very Large Scale Integration,* (Academic Press 1981), 183-192.

[Br77] Breuer, M.A., "Min-Cut Placement," *Journal of Design Automation and Fault-Tolerant Computing,* Vol. 1, No. 4, (1977), 343-362.

[FM82] Fiduccia, C. and R. Mattheyses, "A Linear-Time Heuristic for Improving Network Partitions," *Proc. 19th Design Automation Conference,* (Las Vegas, 1982)

[HS71] Hashimoto, A. and J. Stevens, "Wire Routing by Optimizing Channel Assignment within Large Apertures," *Proc. 8-th Design Automation Workshop,* IEEE (1971), 214-224.

[Hi74] Hightower, D., "The Interconnection Problem: A Tutorial," *Computer* 7,4 (April 1974), 18-32.

[Ko81] Koschella, J., "A Placement/Interconnect Channel Router: Cutting your PI into Slices," Bachelor's Thesis. MIT Department of Electrical Engineering and Computer Science. (May 1981).

[La79] Lauther, U., "A Min-Cut Placement Algorithm for General Cell Assemblies Based on a Graph Representation," *Proc. of the 16th Design Automation Conference,* IEEE (1979), 1-10.

[Le61] Lee, C. Y., "An algorithm for path connections and its applications," IRE Trans. on Electronic Computers, (Sept. 1961), 346-365.

[MC80] Mead, C., and L. Conway. Introduction to VLSI Systems. Addison-Wesley (1980).

[PDS77] Persky, G., D. Deutsch, and D. Schweikert, "LTX – A Minicomputer-Based System for Automated LSI Layout," *Journal of Design Automation and Fault-Tolerant Computing* 1,3 (May 1977), 217-255.

[RF82] Rivest, R., and C. Fiduccia, "A 'Greedy' Channel Router," *Proc. 19th Design Automation Conference,* (Las Vegas, 1982)

[So81] Soukup, J., "Circuit Layout," Proc. of the IEEE, Vol. 69, No. 10(Oct. 1981), 1281-1304.

[WM81] Weinreb, D., and D. Moon. Lisp Machine Manual. (Fourth edition.) MIT Artificial Intelligence Laboratory (1981).

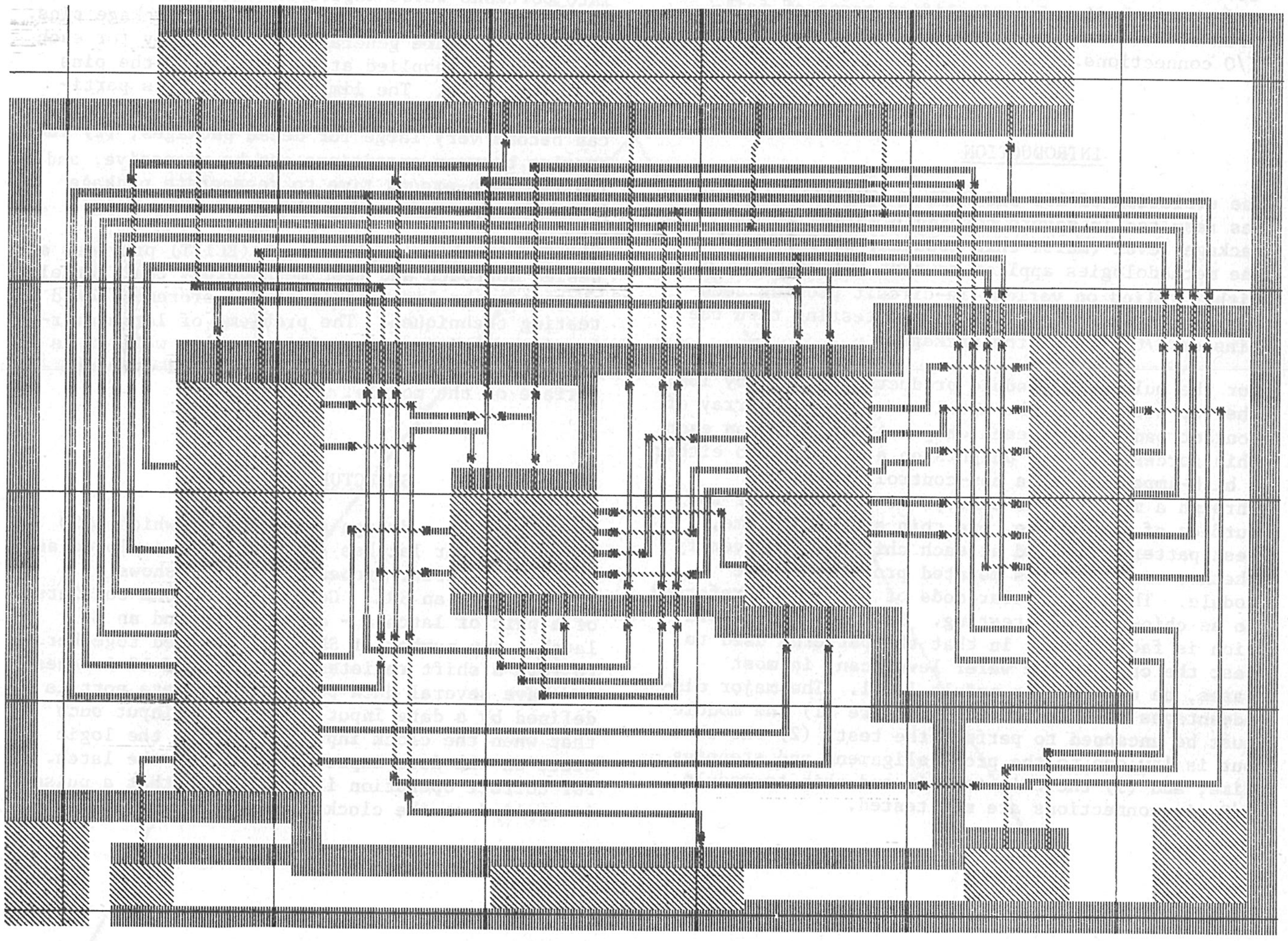


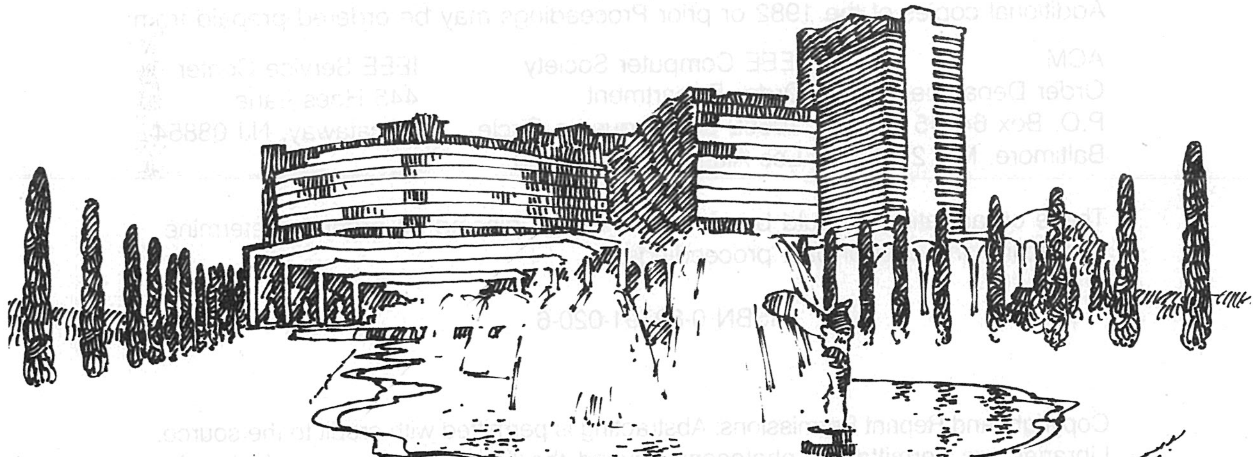
Figure 7. Performance of PI on a small example.

ISSN 0146-7123
IEEE Catalog No. 82CH1759-0
Library of Congress No. 76-150348
IEEE Computer Society Order No. 416
ACM Order No. 477820

ACM IEEE Nineteenth Design Automation Conference Proceedings

Caesars Palace
June 14-16, 1982

Las Vegas, Nevada



acm SIGDA

IEEE COMPUTER SOCIETY-DATC

PROCEEDINGS OF THE DESIGN AUTOMATION CONFERENCE

9th (1972) Contains	47 Papers	379 Pages
10th (1973) Contains	36 Papers	288 Pages
11th (1974) Contains	47 Papers	379 Pages
12th (1975) Contains	57 Papers	448 Pages
13th (1976) Contains	63 Papers	501 Pages
14th (1977) Contains	70 Papers	507 Pages
15th (1978) Contains	72 Papers	493 Pages
16th (1979) Contains	94 Papers	586 Pages
17th (1980) Contains	81 Papers	656 Pages
18th (1981) Contains	134 Papers	917 Pages
19th (1982) Contains	125 Papers	928 Pages

Prices (1982) ACM or IEEE Members:	$39.00 prepaid
All Others:	$52.00 prepaid

ACM Order Number 477820
IEEE Catalog No. 82CH1759-0
Library of Congress No. 76-150348
IEEE Computer Society Order No. 416
ISSN0146-7123

Additional copies of the 1982 or prior Proceedings may be ordered prepaid from:

ACM	IEEE Computer Society	IEEE Service Center
Order Department	Order Department	445 Hoes Lane
P.O. Box 64145	10662 Los Vaqueros Circle	Piscataway, NJ 08854
Baltimore, MD 21264	Los Alamitos, CA 90720	

These organizations should be contacted before placing an order to determine availability and cost of past proceedings.

ISBN 0-89791-020-6