RIVER ROUTER FOR THE GRAPHICS EDITOR CAESAR

Jaya Holla .
B.Sc. St. Xavier's College , Bombay . 1978
M.S. Case Western Reserve University , Cleveland .1982

A thesis submitted to the faculty
of the Oregon Graduate Center
in partial fulfillment of the
requirements for the degree
Master of Science
in
Computer Science

November, 1983

The thesis "River Router for the graphics editor Caesar" by
Jaya Holla has been examined and approved by the following Examination Committee:

Alan C. (Kit) Bradley, Thesis Research Advisor
Adjunct Assistant Professor,
Dept. of Computer Science and Engineering

Richard B. Kieburtz
Professor,
Dept. of Computer Science and Engineering

Robert G. Babb II
Assistant Professor,
Dept. of Computer Science and Engineering

Jon Orloff
Associate Professor,
Dept. of Applied Physics and Electrical Engineering

TABLE OF CONTENTS

TABLE OF CONTENTS

ABSTRACT

RIVER ROUTER FOR THE GRAPHICS EDITOR CAESAR

Jaya Holla.
Oregon Graduate Center, 1983

Supervising professor: Alan C. Bradley

A general river routing algorithm is described. It is assumed that there is one layer available for routing and the terminals are on the boundaries of an arbitrarily shaped rectilinear routing region. All nets are two terminal nets. No crossover is permitted between nets. A minimum separation must be maintained between wires to prevent design rule violations. The separation and default width for all nets are obtained from a parameter file. A command line option permits the user to change the width. The algorithm assumes no grid on the routing plane. The number of corners in a given route is reduced by flipping corners.

1. INTRODUCTION

The complexity of circuits on a single chip has increased drastically with the advances in VLSI technology and the software available for design automation. Circuits with tens of thousands of transistors have been designed and fabricated. These integrated chips usually have various sets of data busses which interconnect different circuit blocks on the chip. At the chip planning stage, the designer can, in general determine the order of input and output busses of each block. If the input and output busses are in the same sequence, it is possible to make the interconnections between the blocks in a single layer. The problem of interconnecting pairs of pins in two rows with the same sequence on a single layer is referred to as the " river routing problem " [1,2,3,4,7].

Ths thesis presents a routing algorithm which handles a more general and practical river routing problem. The algorithm can handle arbitrarily shaped contiguous rectilinear routing regions. The algorithm guarantees a solution if one exists. The existence proof is given in [1].

Section 2 describes the interface of Caesar[6], a VLSI layout program, to the river router. Section 3 describes the conversion of the Caesar rectangles to a contiguous routing polygon. Section 4 outlines the river routing algorithm. Section 5 deals with corner minimization.

1.1. TERMINOLOGY.

A *routing area* is a continuous area between blocks of circuitry that can be used for routing.

A *terminal* is either an input or output pin on the boundary of the routing region. It is characterized by a name and its location.

A *net* is a set of pairs of terminals to be interconnected by wires.

A *routing segment* is a horizontal or vertical wire segment, it is represented as a rectangle of a particular layer specified by its lower left and upper right coordinates.

A *route path* is a wire connecting two terminals and is characterized by a starting terminal a set of horizontal and vertical route segments, and an ending terminal.

A *solution* to a routing problem is a set of routing paths inside the routing area which connects the set of nets without design rule violations.

A *channel* is a rectangular space between circuit blocks.

2. THE CAESAR INTERFACE

Caesar is an interactive system for creating and modifying VLSI circuit designs . It is based on the Mead and Conway[5] style of design , and produces CIF [5] descriptions (Caltech Intermediate Form) suitable for chip fabrication. It is a geometry editor that allows picture painting of VLSI circuits and the combination of pictures hierarchically into larger designs.

It is a two screen system. One screen , called the text display, may be any standard CRT terminal capable of running the screen editor vi. Caesar is invoked from this terminal ; commands are typed at its keyboard and a command menu and several statistics about the chip design are displayed on the text display. The second screen is the graphics display and is used to display in color a piece of the circuit designed. A graphics tablet must be attached to the color display. For further information refer to "Editing VLSI Circuits with Caesar", User's Manual, John Ousterhost [6].

A cell is a piece of the design that can be stored and retrieved by name. A separate disk file is used to hold the contents of each cell. Caesar commands permit the designer to compose cells hierarchically into larger systems.

Fig 1 is an example of a hierarchically composed system for Caesar. The top level file main.ca has cells 1 through 6 included in it. Cell7.ca and cell8.ca are included in cell2.ca.

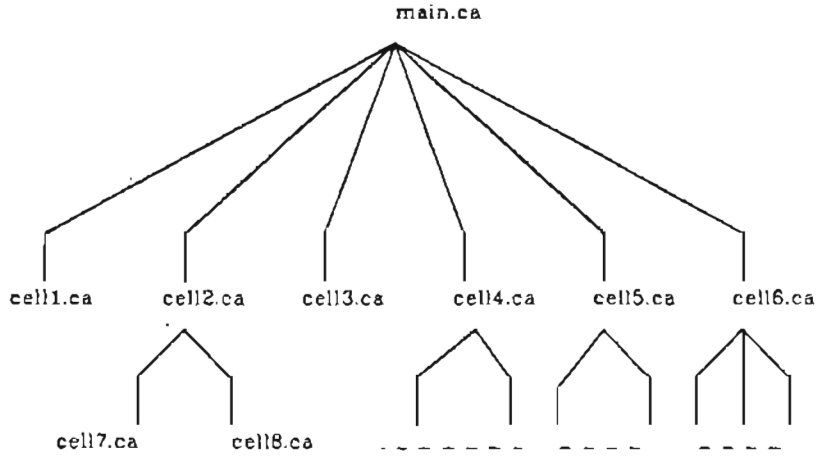


Fig 1. Cell Hierarchy.

The standard file format for a Caesar file is:

```
tech nmos
<< polysilicon >>
/* The rectangles that have to be painted  red */
<< diffusion >>
/* list of the rectangles that have to be painted green */
<< metal >>
/* list of the rectangles that have to be painted blue */
<< labels >>
/* Label names and their position */

<< end >>
```

2.1. CAESAR MODIFIED FOR THE RIVER ROUTER.

In preparation for routing the user creates a new cell and places the following information in it:

1) The named pairs of terminals to be connected.
2) The area available for routing.

Caesar had no provision to create a new cell without leaving the current editing session . A new function has been included in the list of long commands to create a new cell. The function created is called

":route cellname.ca". The new cell that is created has a rectangle of the routing layer included in the file created. The designer then gets the cell using the "getcell" command in Caesar. To incorporate the new cell into a design layout, the cell must be positioned and copied at the required spot. Choosing this as the current cell and subediting it now permits the designer to include the necessary routing information.

A new layer has been added to the nmos.tech file to include a layer called "routelayer". The routing area is specified by repeated positioning and painting of the Caesar box with the routing layer to obtain the desired rectilinear routing area. Terminal pairs to be routed are specified using the long command ": label name " Caesar places the label name and its coordinates in the routing file. "Return" gets the designer back to the original cell being edited. On exiting from Caesar all the routing information is stored in the file which was created by route. The file contains the rectangles of the route layer and the coordinates of the labels.
Rectangles are stored as :

rect x1 y1 x2 y2

The labels are stored as :

label name x1 y1 x2 y2 position(1 2 3 4)

The label's coordinates are stored as a rectangle. The Caesar box should be compressed to a point when specifying the terminals.

The router can then be invoked from the Unix[1] shell. The command is :

router -[p d m [integer]] filename

The layers available for routing are:

p - Polysilicon.
d - Diffusion.
m - Metal.

---

[1] Trademark of Bell Laboratories.

A integer value in multiples of lambda can be specified to override the default value in the parameter file. The route output will be included in the specified file. Refer to Appendix A for more information.

Assumption: This cell does not already contain any paint of the layer that will be used for routing.

3. CONVERSION OF THE RECTANGLES INTO A ROUTING POLYGON.

The Caesar file, read in by the program, contains rectangles of the routing area. The information contained within it has to be stored in data structures for easy manipulation within the program.

For its internal representation Caesar decomposes an arbitrary rectilinear polygon into horizontal rectangles. Fig 2 is an example of the decomposition. Horizontal rectangles are convenient for polygon fill with a particular color on the raster scan color display monitors. The rectangles have to be converted into a contiguous routing area.

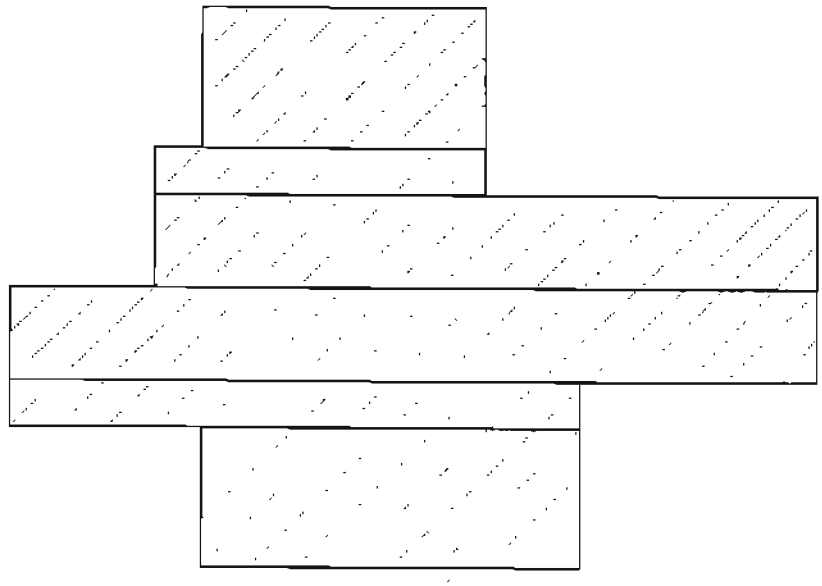


Fig 2. Arbitrary rectilinear routing region decomposed into rectangles.

A meaningful representation from a programming viewpoint can be obtained by viewing a rectangle as four directed line segments. There are four types of line segments that are possible with Manhattan style design.

The diagram below has the four possible cases. Note that by definition the line segments traverse the routing area in a counter-clockwise direction.

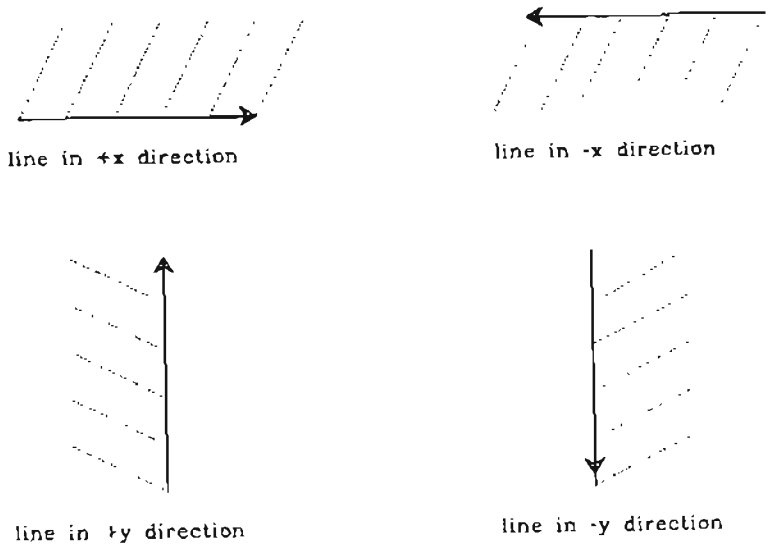


Fig 3. The four types of line segments.

To obtain a arbitrary rectilinear polygon area the line segments can be looked upon as dividing the space into two sections, one that could be included in a polygon and the other that cannot be included in the polygon. Only certain combinations of these line segments can be used to obtain a meaningful representation. A rectangle can now be viewed as four directed line segments which enclose an area.

The four directed line segments can give rise to eight types of corners. Fig 4 shows these eight types of corners.

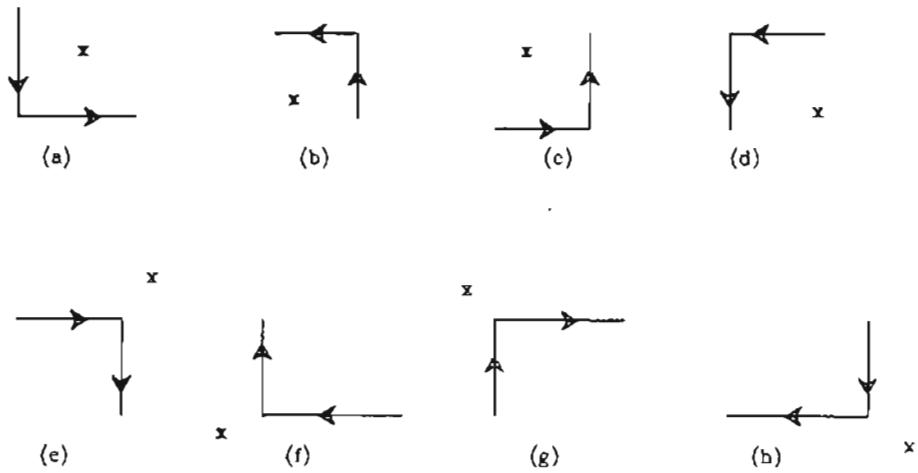


Fig 4. Eight types of corners that could occur.
The symbol "x" denotes the routing area

The first thing the routing program does is convert the list of rectangles from the Caesar file into a contiguous rectilinear polygon to represent the routing area. The rectilinear polygon or routing area is represented by a circular, doubly linked list. The links represent the line segments and the nodes represent the corners. Each node is tagged with the type of line segment that it represents and with the type of corner. Traversing the linked list in the forward direction is equivalent to following the routing area in a counter-clockwise direction.

The first rectangle from the Caesar file is set up in the linked list. Other rectangles can intersect this routing area only in a positive x segment or in a negative x segment. The order of inclusion of a rectangle with the route area depends on the type of intersection. As each rectangle is included it is marked as included. The list of rectangles is

repeatedly traversed as long as unmarked intersecting rectangles exist on the list. At the end of this algorithm the designer is warned of any rectangles that have not been included in the route area.

Algorithm for conversion of rectangles into a contiguous routing area:

```
K : Linked list of rectangles sorted in decreasing y-coordinate.
k : element on the above list.
L : circular doubly linked list representing the routing area.
Each entry in the list represents a vertex on the routing area boundary.

Initialize L to the first rectangle on the list K.
Mark first rectangle on K as INCLUDED.
change = TRUE ;
while change
begin
  change = false ;
  Initialize k to be the first element on the list K ;
  while ( k <> NULL )
  begin
   if rectangle k is NOT INCLUDED then :
      check rectangle for intersection with the routing area.
      if it intersects the routing area then
      begin
        if routing area segment is parallel to +x axis
        begin
          include points in list L in the order 4123 after point 1;
             ( See Fig 5 for inclusion order and definition of point1)
          mark rectangle k = INCLUDED ;
             change = TRUE ;
        end; { if }
        else
        if routing area segment has direction parallel to -x axis
        begin
          include points in list L in the order 2341 after point 1.3 ;
             ( See Fig. 6 for inclusion order and definition of point1.3)
```

```
                mark rectangle = INCLUDED ;
                change = TRUE ;
            end  { end if }
        end  { end if }
        next rectangle on the list K ;
    end { inner while }
end   { outer while }          /* Continue as long as there are rectangles
to be included */
```

The terminal labels are stored in a convenient form and require no further conversions to make it suitable for the router.

The data flow diagram below shows the interaction of the router with Caesar.

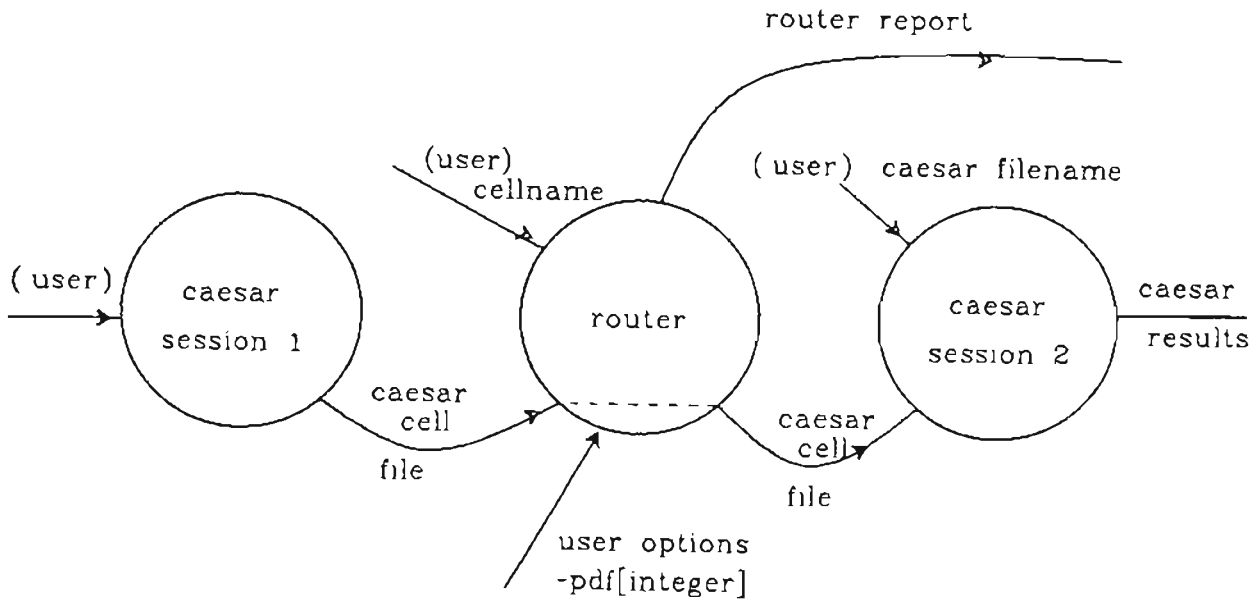
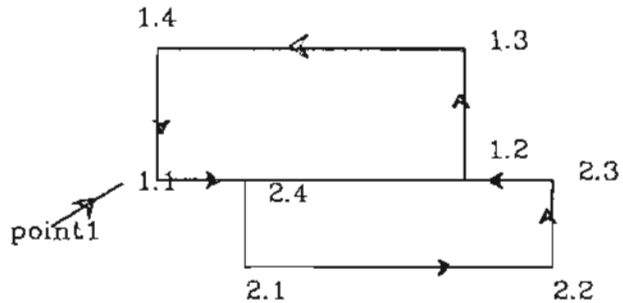
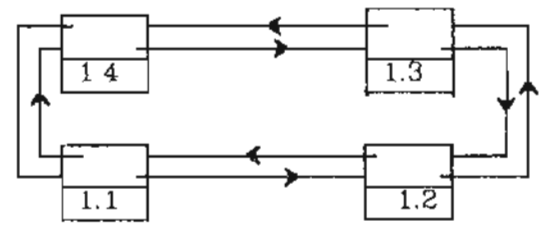


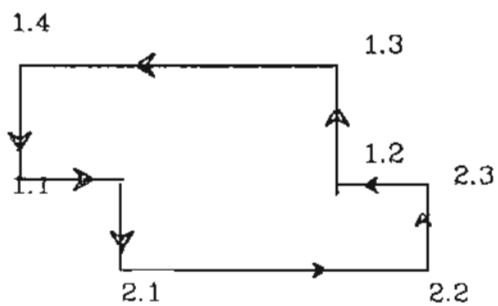
Fig 5. Top level data flow diagram for the router.



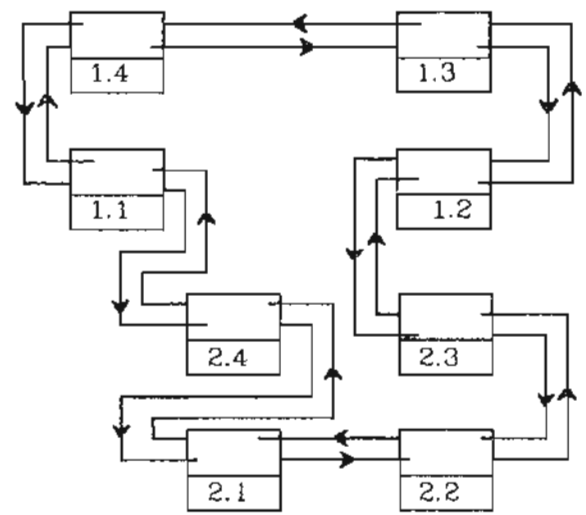
a) Rectangles for inclusion



c) Rectangle 1 included

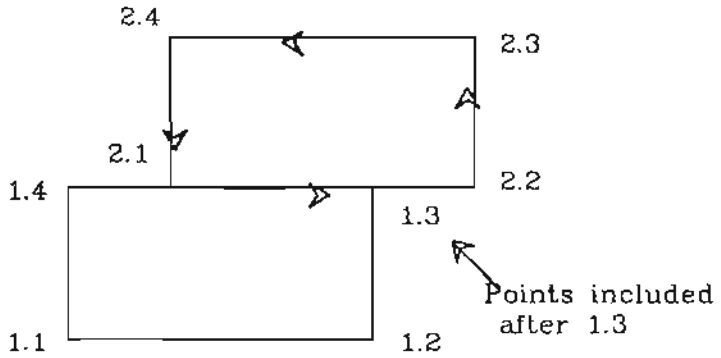


b) Directed line segments representing the route area boundary

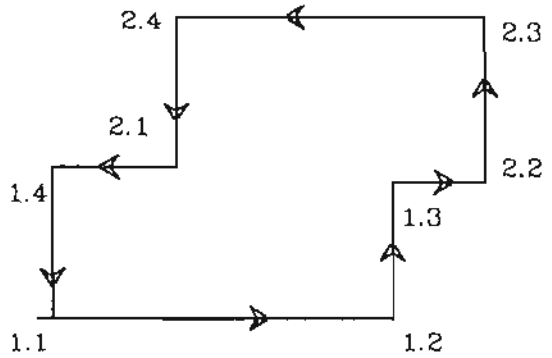


d) Rectangle 2 included in the doubly linked list.

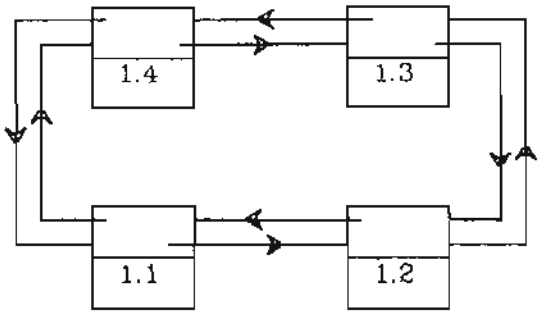
Fig 6. Inclusion of a rectangle intersecting the routing area in increasing x coordinate.



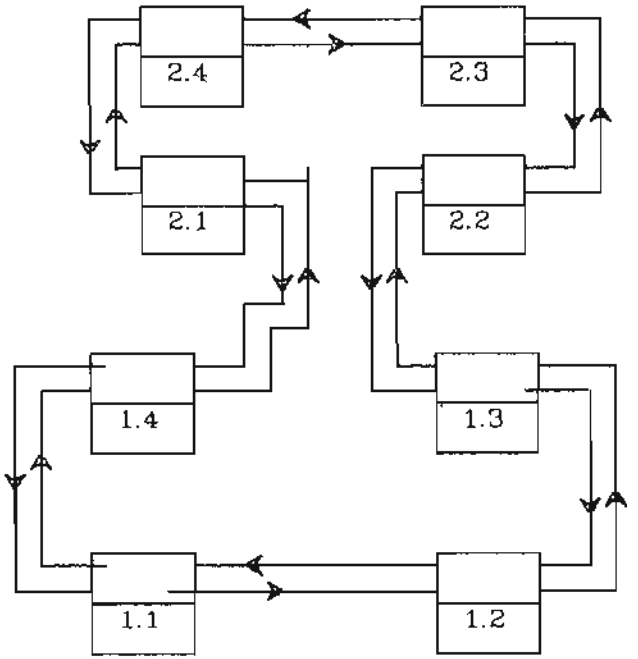
a) The two rectangles



b) Directed line segments representing the routing area



c) rectangle 1 included



d) Rectangle 2 included

Fig 7. Inclusion of a rectangle intersecting in decreasing x.

4. GENERAL RIVER ROUTING ALGORITHM

An ordered list of segments is continuous if the starting point of every segment, except for the first segment, is the ending point of the previous one . A routing path is a continuous list of alternating horizontal and vertical routing segments. A terminal connected with the first segment of a path will be called a starting terminal and the terminal connected with the last segment is an ending terminal. The widths of the routing segments are all the same and are predefined by the user.

The program checks for planarity of the terminal pairs to be routed. The start and end terminals for each route are identified while checking for planarity. The routing is done one net at a time while executing a modified stack routine. Path selection is done by routing each net in turn as close to the boundaries as possible.

After all nets have been routed unnecessary corners are removed by flipping corners.

4.1. STARTING TERMINAL ASSIGNMENT.

For river routing, each routing net has exactly two terminals to be connected. Assume that every routing path is counter-clockwise along the boundary. Every path has two possible choices along the boundary, corresponding to the two possible choices of the starting terminal. Fig 8. shows these two choices.

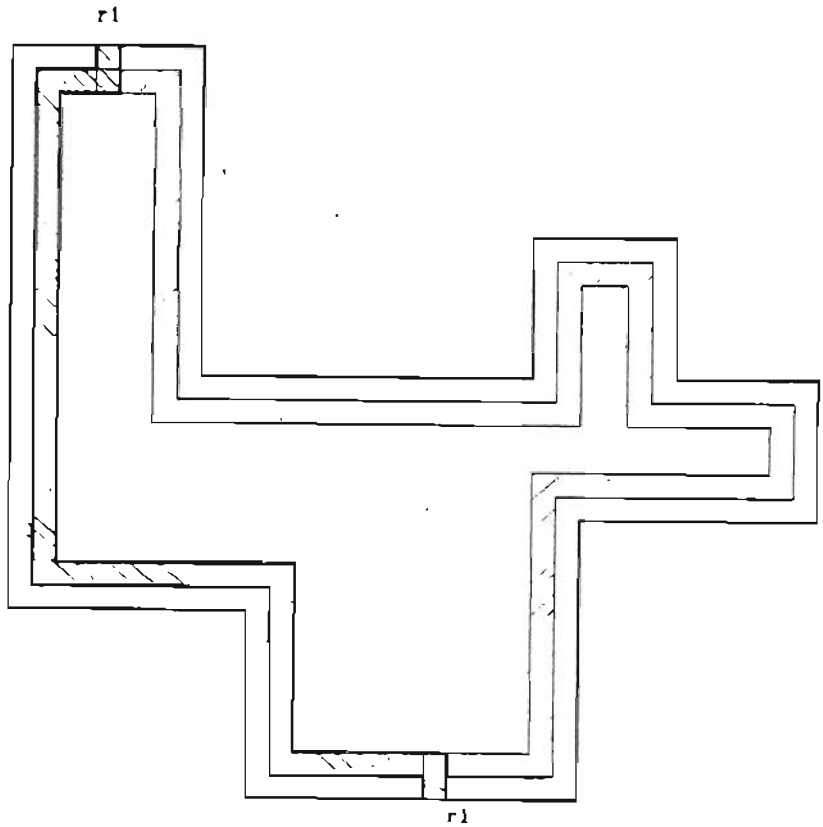


Fig 8. Two possible paths for routing r1.

The starting terminal for a routing net is chosen independently of all other nets, such that the shorter path is chosen. This is done by calculating the total length of the boundary segments counter-clockwise between the arbitrarily chosen starting and ending terminals, and then comparing to the length of the total routing perimeter. If the path length is less than half of the total length, the start terminal is marked as start, and the end terminal is marked end. If otherwise, then the assignment is switched.

4.2. PLANARITY CHECK

Next we need to determine if a single layer route is possible. Constraints are imposed on the order in which the nets are routed. A net is routed only after all nets have been routed which have both terminals on the counter-clockwise portion of the routing area boundary between the starting and the ending terminals of the net.

A simple check can be performed to check whether the terminals as they appear along the boundary of the polygon can be connected without intersecting lines. This is analogous to checking whether a set of parentheses is properly balanced, as formulated by the following algorithm[4]:

> Initialize a stack S to be empty.
> Cut the boundary of the routing polygon at any vertex and straighten it out.
> Scan the list of terminals from left to right. For each terminal compare the net identification to the one on the top of the stack. If they are equal then pop one, else push it on the stack.

If at the end the stack is empty, the nets are properly ordered. Otherwise they are not routable in a single layer.

Correctness[4]: The first occurrence of a terminal in a net is treated as an opening(left) parenthesis and the second occurrence as a closing(right) parenthesis. Regardless of where the boundary has been cut, the pattern is planar if and only if the parenthesis in the expression(obtained by the above interpretation) are properly balanced. The correspondence is demonstrated in Fig 9.

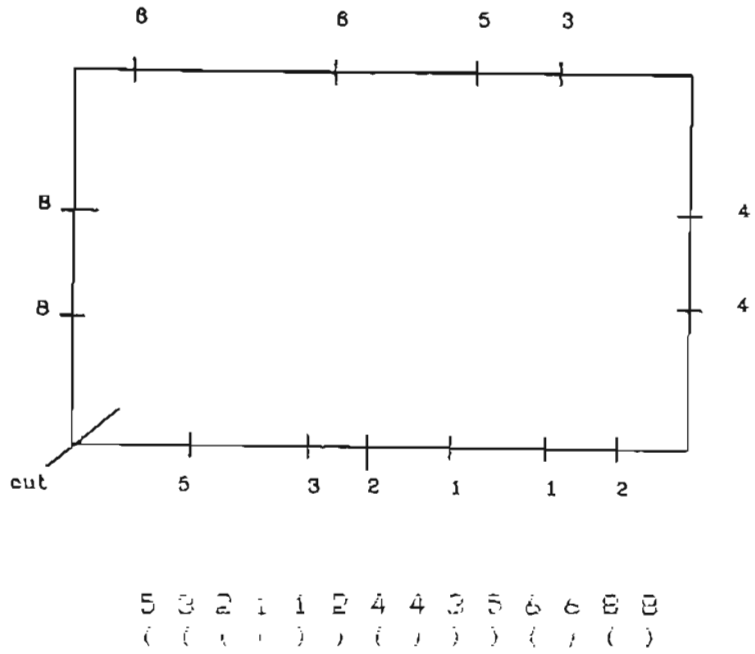


Fig 9. The planarity check is equivalent to balancing parentheses.

4.3. NET ORDERING.

A modified stack routine discovered by C. P. Hsu [1] is used to take into account the information about the starting and ending terminals. The algorithm eliminates the dependency of the routed nets on the spot where the cut was made in the routing area.

A circular singly linked list consisting of all terminals ordered in counter-clockwise direction according to their positions on the routing boundary is set up. The terminals have to pass the planarity check before this algorithm is executed.

The nets are routed in the order determined by the following algorithm:

```
Initialize stack S to empty.
i = 1 ;          /* Index for the nets */
N = Total number of nets.
T = any terminal in the circular list.
Every starting terminal is marked as NOT PUSHED.
Every ending terminal is marked as NOT MATCHED.
while i <= N
begin
  if T is a starting terminal that is NOT PUSHED,
  begin
    push T on S ;
    mark T as PUSHED ;
  end
  else
  begin
    if T is an ending terminal that is NOT MATCHED
    begin
      if T and the top element of S belong to the same net
      begin
          mark T as MATCHED
          pop the element from S;
          increment i ;
          route              /* Do the route here */
      end;
    end;
  end;
T = next terminal on the list ;

end;
```

4.4. ROUTING PARAMETERS.

The layers available for routing are polysilicon, diffusion and metal. To prevent short circuiting between wire runs in these layers a minimum separation between wires must be maintained.

The routes have to obey design rule constraints. The information about the minimum spacing between routes is read in from a parameter file. The routeparam file is:

```
lambda2        ; lambda = 2 microns
NP     2  2  ; width and spacing for polysilicon in lambda
NM     3  3  ; width and spacing for metal in lambda
ND     2  3  ; width and spacing for diffusion in lambda
```

Fig 10. Parameter File for the router.

4.5. PATH SEARCHING

At this stage the routing area is represented by a doubly linked circular list. Each time a matching pair of start and end terminals are identified in the modified stack routine the program goes through the following sequence:

1) Set up the route path. Perform design rule checks on the new vertex points generated.

2) Check path segments for design rule violations.

3) Create reduced route area if 1 and 2 were successful.

4.6. ROUTE PATH

The route path starts at the start terminal and follows the route area in a counter clockwise direction hugging the route area boundary while maintaining a certain distance from the edges. The section of the route area boundary between the start and the end terminal is checked for channels with a width less than the minimum width. Any channel which is too narrow is replaced by the appropriate line segment. Fig 11 shows how the minimum width for a channel is calculated and Fig 12 shows the replacement of narrow channels in the route area boundary.

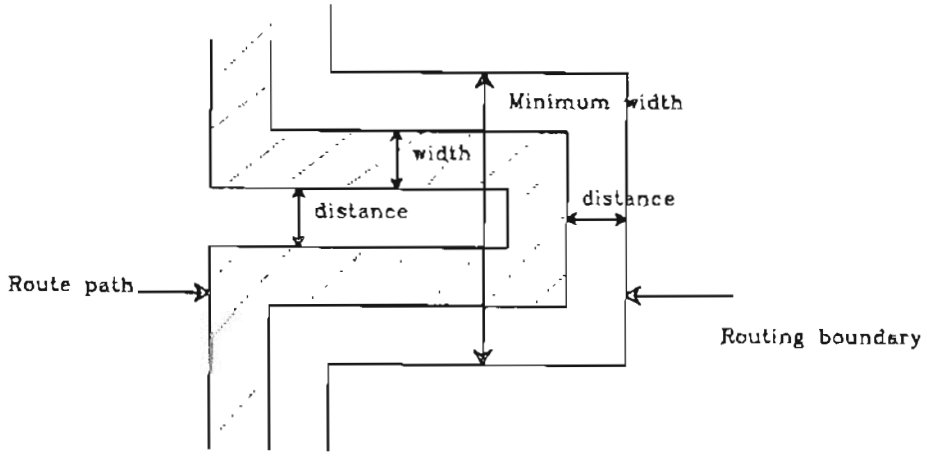


Fig 11. Calculation of minimum channel width.

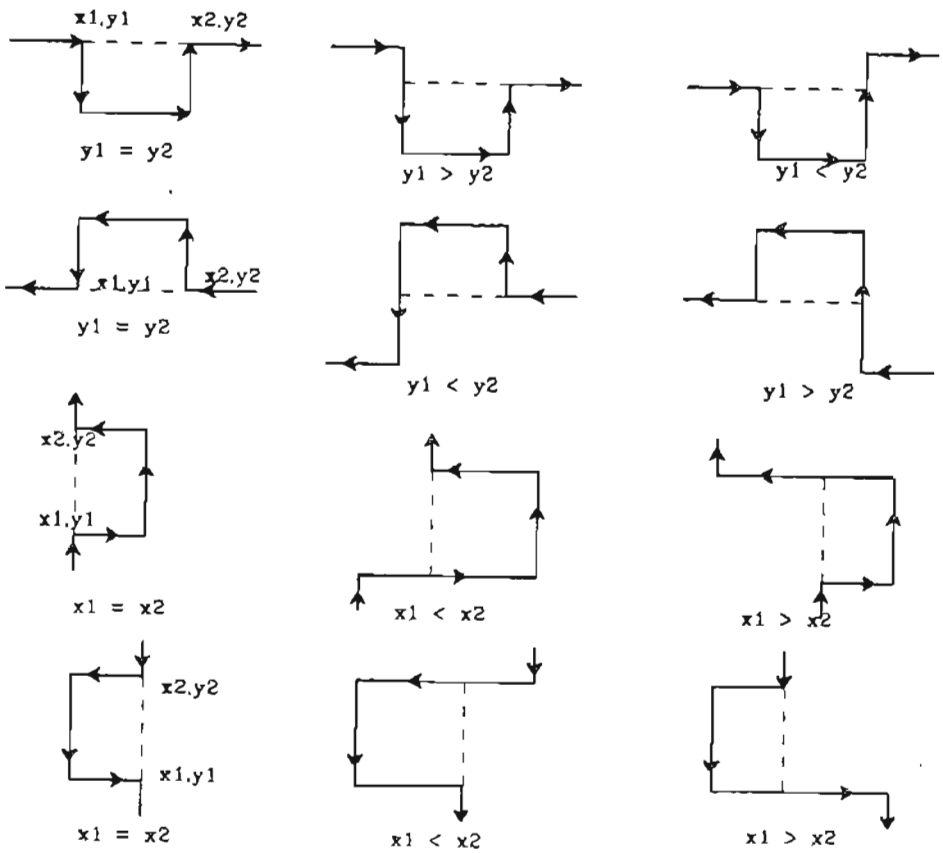


Fig 12. Different cases for narrow channel elimination of the route boundary. (Dashed lines represent the new segment.)

A route path point is obtained by calculating new vertex coordinates for each route area corner encountered while following the route area.

Two new points are generated for each corner point on the route boundary. One point is a minimum distance away and the other one is a distance plus the specified width away from the route area corner. This latter point is later used when creating new route area after this route has been successfully completed. A check for inclusion within the route area boundary is performed on these two points.

4.7. DESIGN RULE VIOLATION CHECKS

The route divides the routing area into two sections. Fig 13 demonstrates this division. One section reflects the corners in the route itself and the program derives these points by following the constraints imposed by design rules and the desired width The rest of the route area boundary, not reflected in the route will have to be checked for violations.

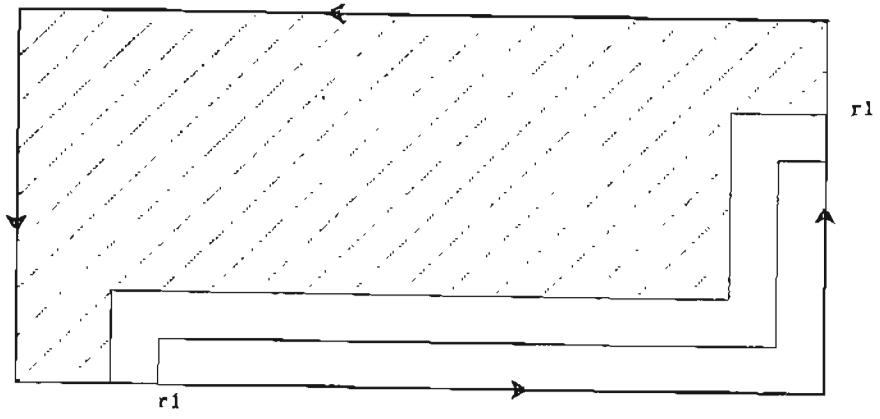


Fig 13.The route path divides the routing area into two sections.

The possible sources for the design rule violations can be categorized into two main categories. The violation could occur at the central portion or at the end points of the line segments that comprise the route path. Fig 14 below identifies the central portion and the end points of a line segment.

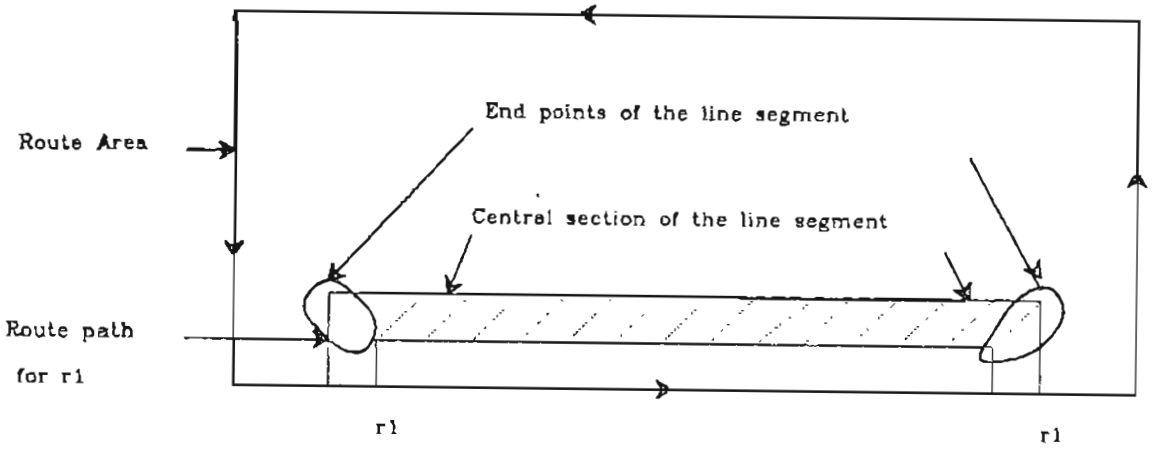


Fig 14. Sections of a line segment.

The check for end point violation is performed during the path search. A third point is generated which is a minimum distance away from the innermost point. If this lies within the route area then the corners do not have any violations. Refer to Fig 15 for a violation example.

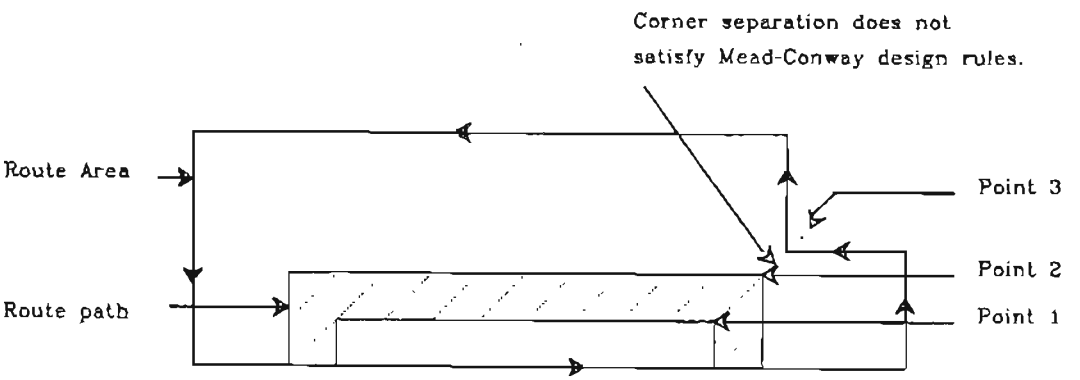


Fig 15. Example of corner violation.

Two types of checks are made for checking violations which could occur in the central portion of the route path. A route path is composed of several line segments. All segments of the route area which are parallel to and within the range of each line segment of the route should be a minimum distance away. The distance is obtained from the Mead Conway[5] design rules. See Fig 16. The second check is done to check for intersection of orthogonal lines of the routing area with the line segment of the route. A violation occurs if either of these checks turn out to be true. Refer again to Fig 16 for example of this second type of violations.

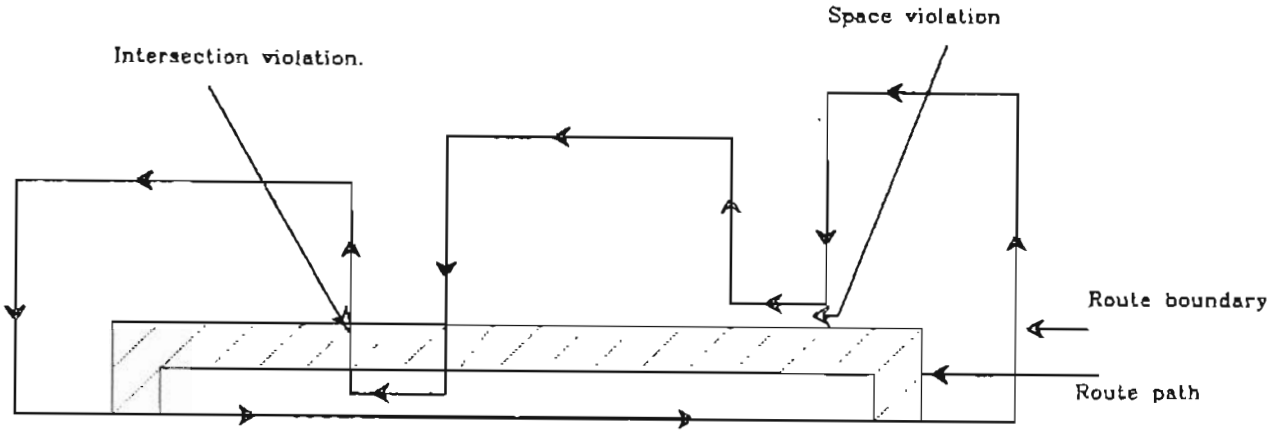


Fig 16- Violations in the central portion of a route path.

If any design rule violations occur while this section of the program is executed, a partial route up to the violating one, is included in the Caesar file. The designer can then add more space where necessary.

4.8. CREATION OF NEW ROUTING AREA

Each route alters the routing area available to the subsequent routes. The linked list representing the area available for routing is changed to reflect the new area available for routing. In the example below the shaded region represents the routing area for r2 after r1 has been routed.

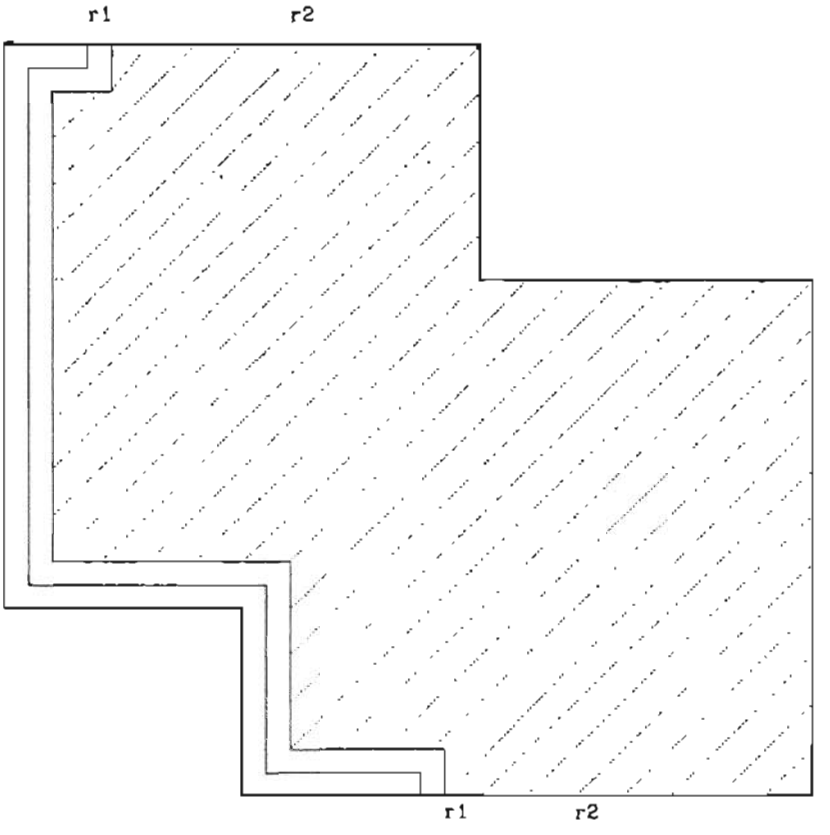


Fig 17. Shaded portion represents routing area for r2.

5. CORNER MINIMIZATION

After the path search has been successfully completed for all nets with no design rule violations, a solution exists. Every net has a unique path associated with it. All paths are pushed outward against the boundaries and the excess space remains in the center of the routing area.

5.1. CORNER FLIPPING AND CHECKING.

The corner minimization is done in a systematic way by flipping corners toward the inside of the routing region. The corners are minimized one at a time. The order in which the paths are minimized is in the reverse order of the path search sequence. The net routed last is minimized first.

Every corner of the path belongs to one of eight cases.

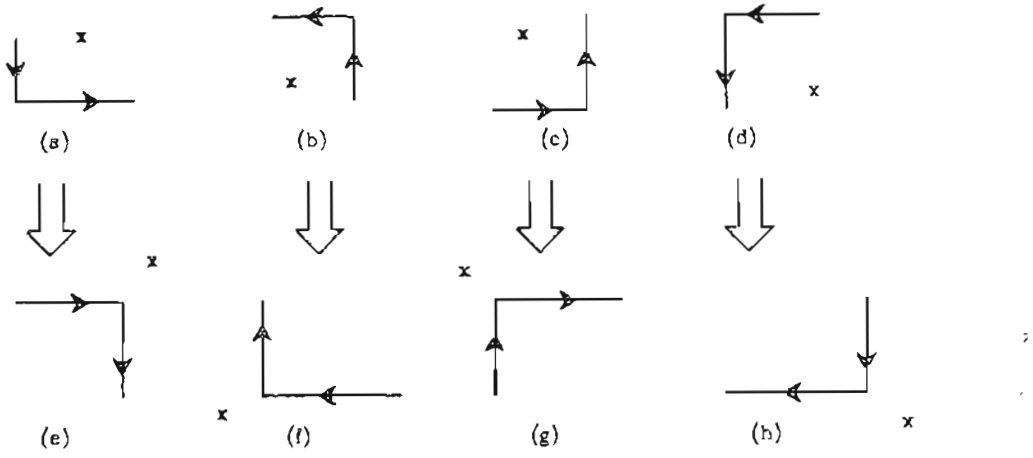


Fig 16. Eight possible cases of a corner.
(Cases (a)-(d) are flipped to the corresponding cases (e)-(h) below.)

Since every path is routed in the counter clockwise direction four of the cases can have their corners flipped toward the inside of the routing region. The top four corners Fig. 18 are converted into the corresponding ones below. A constraint segment list is generated to check for design rule violations. If the new corner violates any design rule, it is not flipped. Two corners are eliminated if the corner flip is successful. See Fig 19. All points on the new line segment are checked to ensure no design rules are violated. See Fig 20 for an example of a violation.

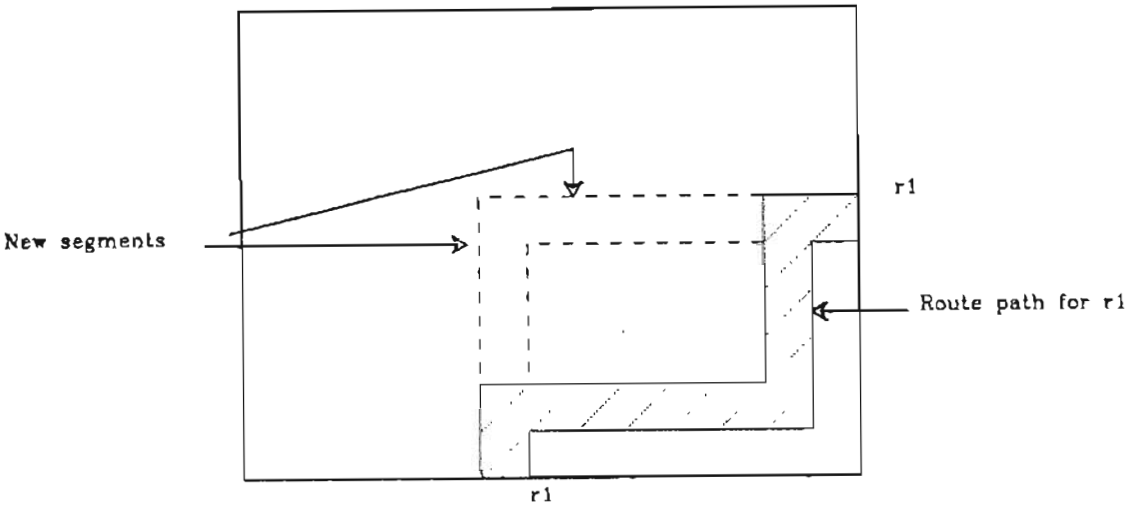


Fig 19. Dashed lines represent the new segments.

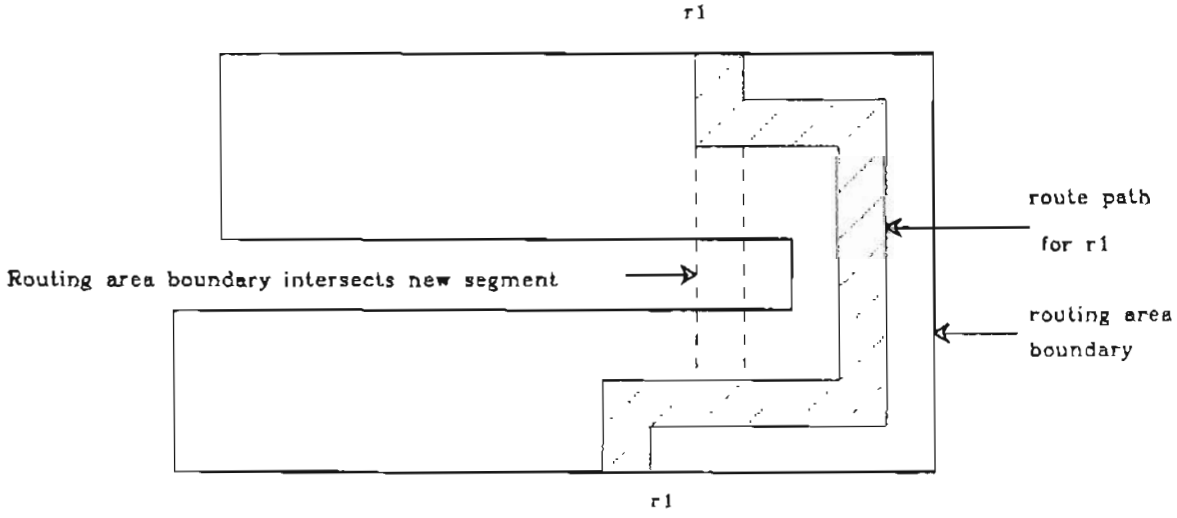


Fig 20 Violation in the new segment.

A different minimization routine is used if two concave corners occur one after the other in the route. There are twelve such cases. Fig 21 shows these twelve cases. The minimization routine generates one line segment as represented by the dashed line in the diagram below. The new segment is checked for design rule violations. If it violates a design rule then it is not kept.

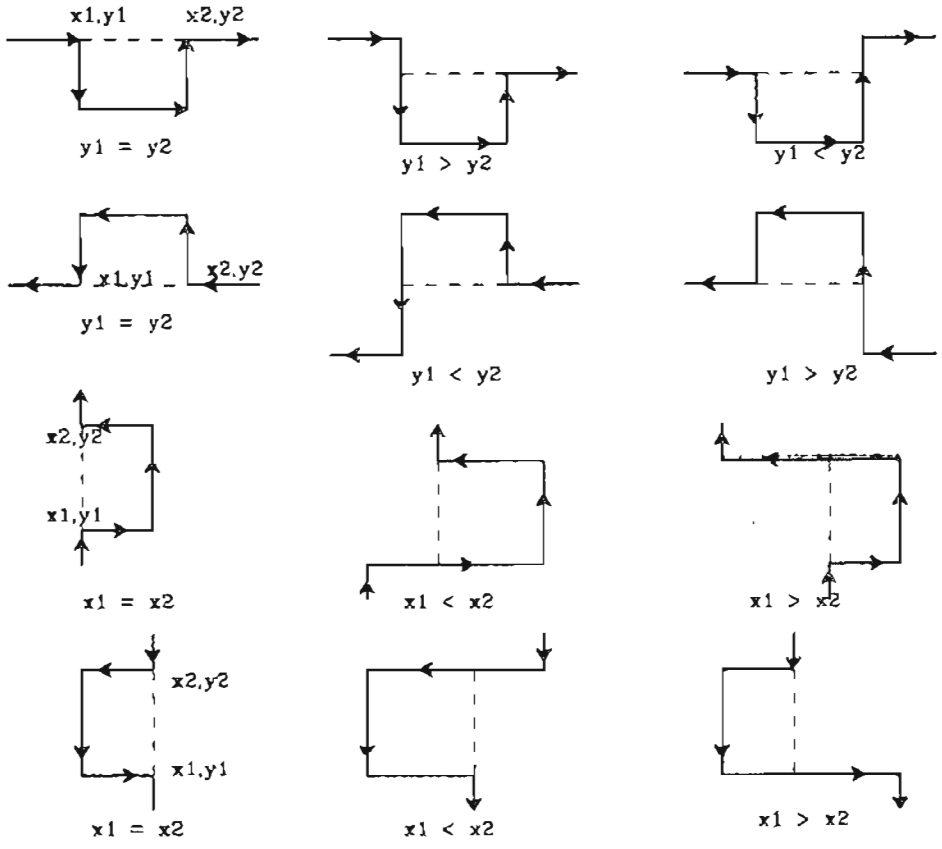


Fig 21. Twelve cases for corner elimination in a path.

5.2. ROUTING AREA FOR MINIMIZATION.

The routing area available for corner flipping for the first route path to be minimized is the routing area remaining after the last route . A new routing area for corner flipping is set up for each following net. Two linked lists are needed for the minimization routing area. One is the original routing area specified by the designer and the second one is the list representing the routing area for minimization. The original routing area is modified to include labels in the order that they appear on the routing area. Three types of points exist in the original list now, vertices from the original routing area, start label points and end label points.

For each net the start and end points are identified in the original routing area. The new linked list is created by traversing in the counter clockwise direction the route area, starting at the ending terminal of the route to be minimized, if a terminal point is encountered then the points of its route are included. The order of inclusion depends on the type of terminal encountered. If it is an end terminal then the route's points are included in the reverse order. If it is a start terminal then the route's points are included in the same order. Points on the original routing area between the start and end points of this net on the original list are skipped. The traversal of the original list continues from the net terminal and proceeds to the start point of the net being minimized. Finally points on the net being minimized are included in the routing area.

In Fig 22 the minimization routing area is set up for R2. The numbers indicate the order in which the points are included. The example illustrates the inclusion order when a start terminal is encountered while traversing in the counter-clockwise direction after the end point of R2. In Fig 23 the minimization routing area for net R1 is set up. The end point of net R2 is encountered while traversing the list in the counter-clockwise

direction. The order of inclusion is different from the first example.

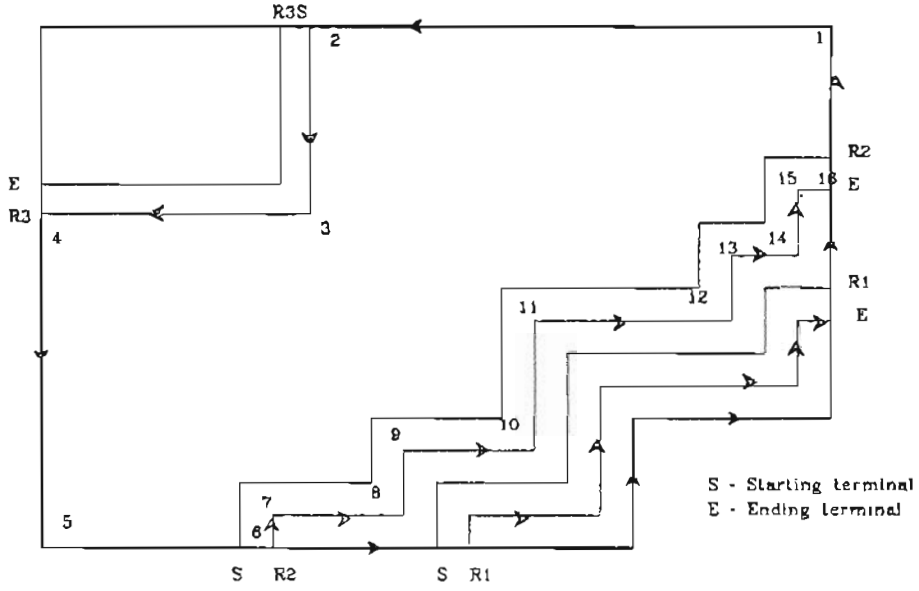


Fig 22. Setting up minimization route area for R2.
The inclusion order is denoted by the numbers.

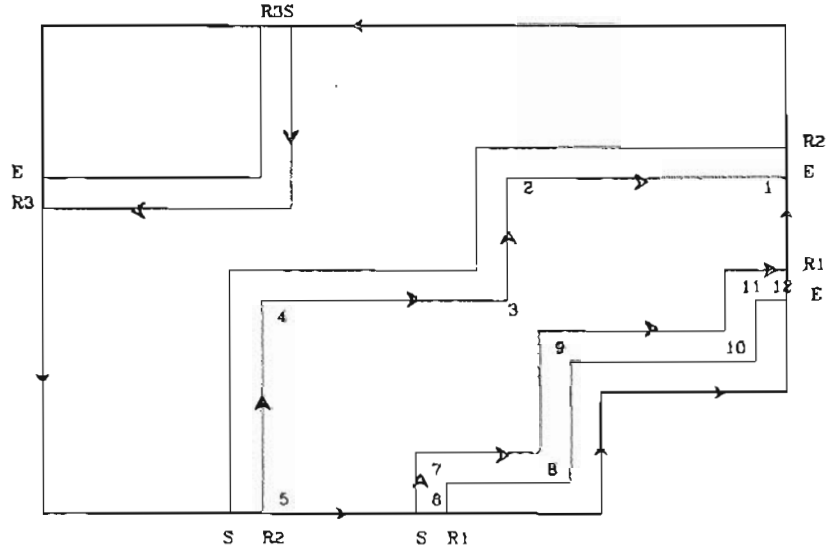


Fig 23. Minimization area for R1.

The example below has three terminal pairs to be routed and minimized. The route was done in the order: 3 2 1. The minimization is done in the reverse order, 1 2 3. In the example below the shaded portion represents the routing area available for minimizing net 1.

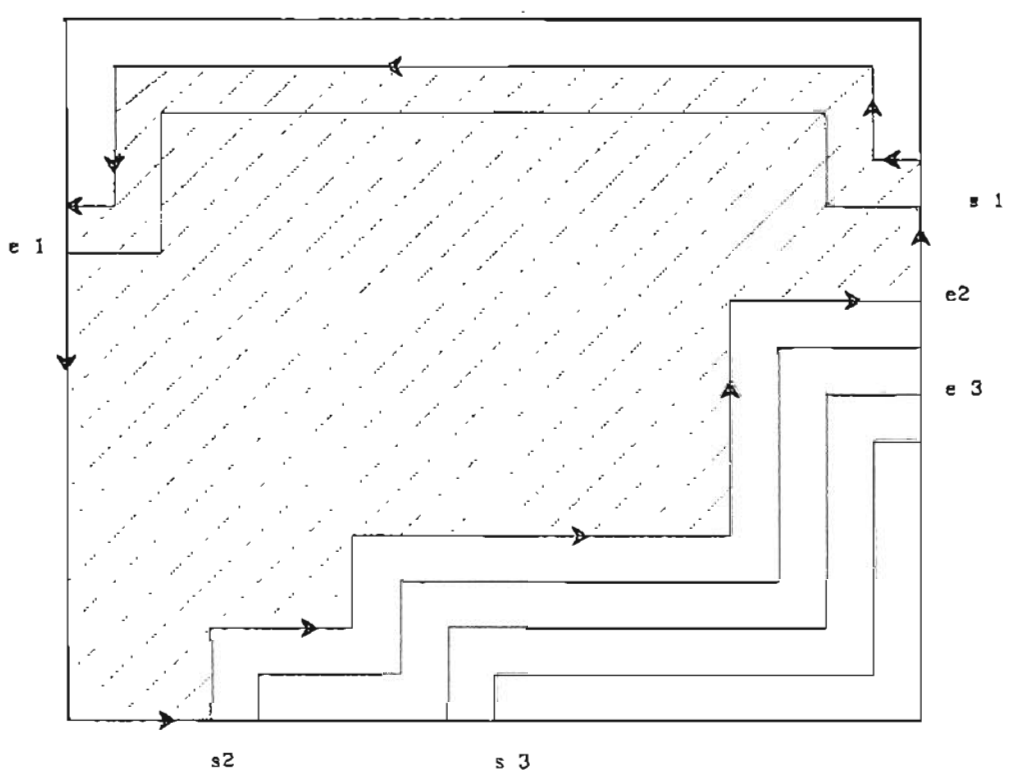


Fig 24. Shaded portion represents minimization routing area for net 1.

In the diagram below routes 1 and 2 have been minimized, the shaded area represents the route area available for the minimization of terminal pair 3.

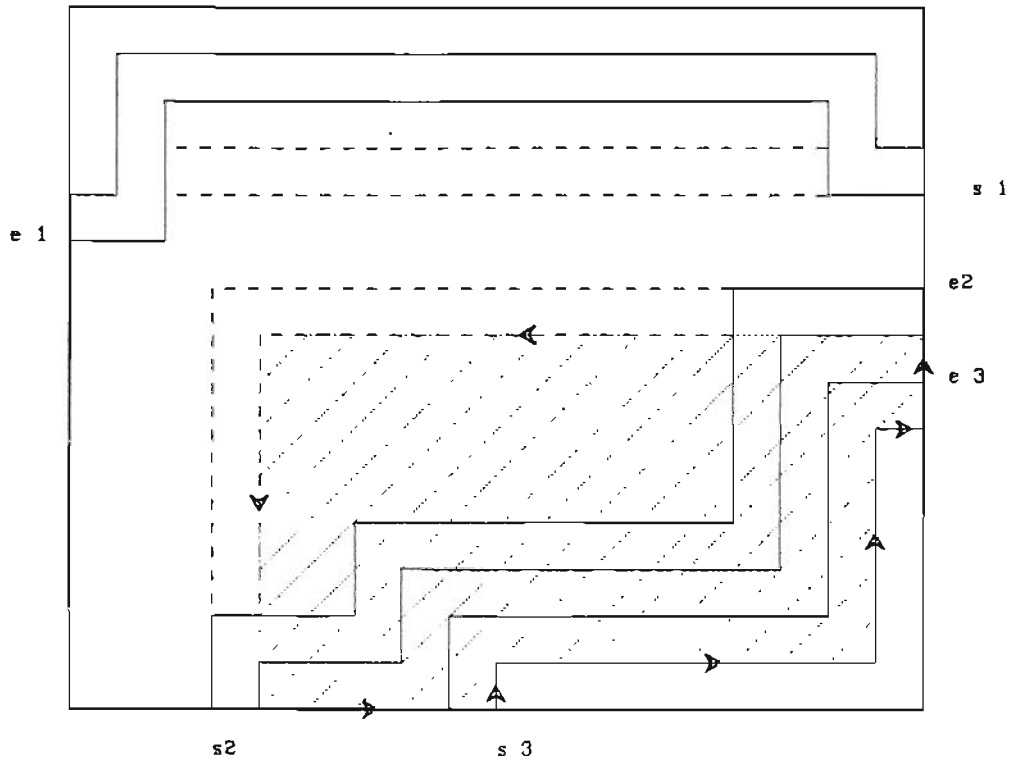


Fig 25. Shaded portion represents minimization routing area for net 3.

6. CONVERSION TO CAESAR FORMAT.

The route paths are stored as line segments and they have to be converted into rectangles suitable for Caesar. The line segments are converted into horizontal rectangles of the layer specified by the designer.

An example of a route done by the program before minimization is shown in Fig 26. Nets after minimization is shown in Fig 27.

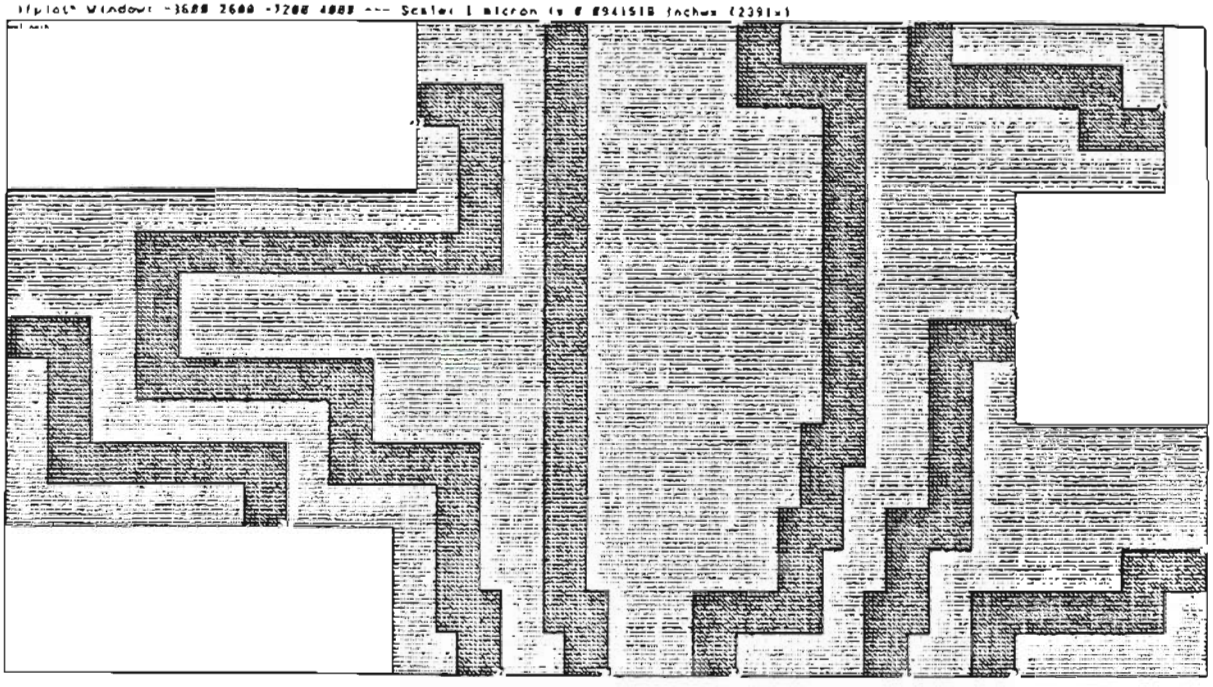


Fig 26. Nets before minimization.

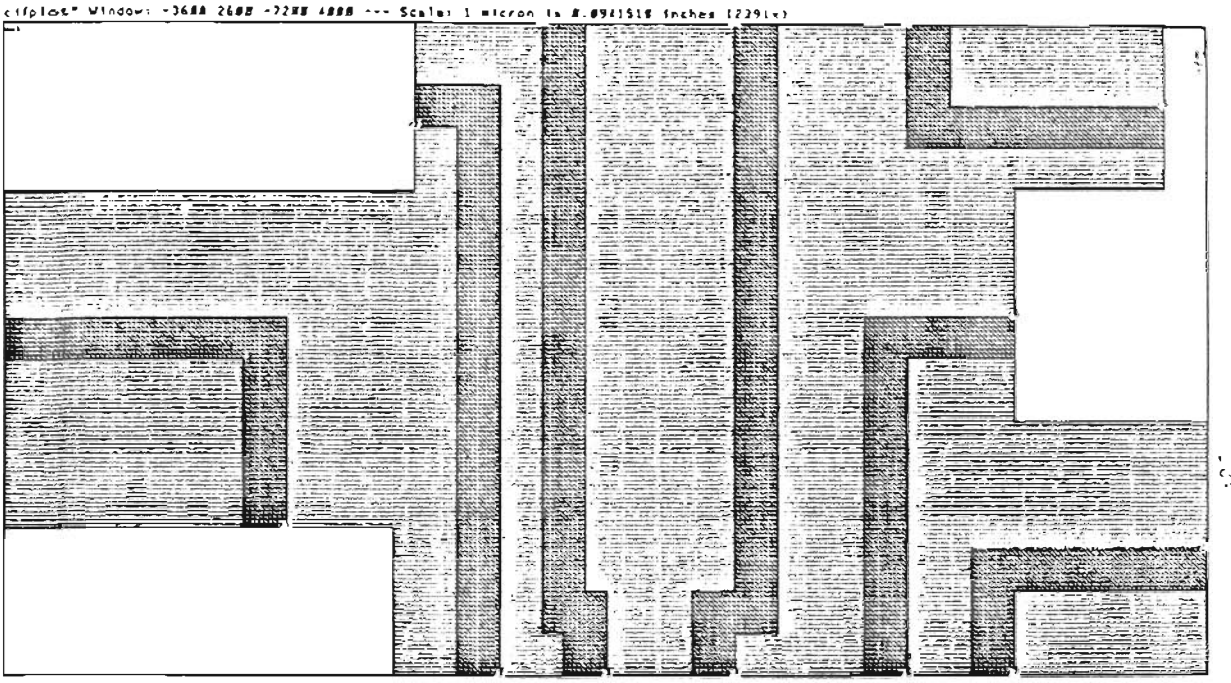


Fig 27. Nets after minimization.

7. CONCLUSION

In summary, the general river routing algorithm routes all nets against the boundaries of the routing area and then tries to minimize the number of corners. A planarity check ensures that the nets are routable in a single layer. The starting terminal assignment, tries to select a shorter path for each net and spread the nets against all boundaries. This assignment does not create any crossovers. Based on this assignment the net order is determined by using a stack. The nets are routed when a matching pair is discovered in the stack routine. The paths are generated by closely following the routing area in a counter-clockwise direction. Finally, the corner minimization is done in the reverse net order by flipping the corners toward the inside of the routing region. This step generates a final layout with paths distributed throughout the routing area.

7.1. IMPLEMENTATION.

The general river router was written in the programming language "C" and implemented on the VAX 11/780 running Berkeley Unix 4.1c.

The major part of the time developing the tool was spent in learning and implementing "C" and the rest of the time was spent in implementing the algorithm.

7.2. EXPERIMENTAL RESULTS.

The route in Fig 28 was produced by the routing program. There were 14 terminal pairs and the execution time was 4.7 seconds. A larger example with 46 terminal pairs for routing took 1 minute and 44 seconds for the router to route. See Fig 29.

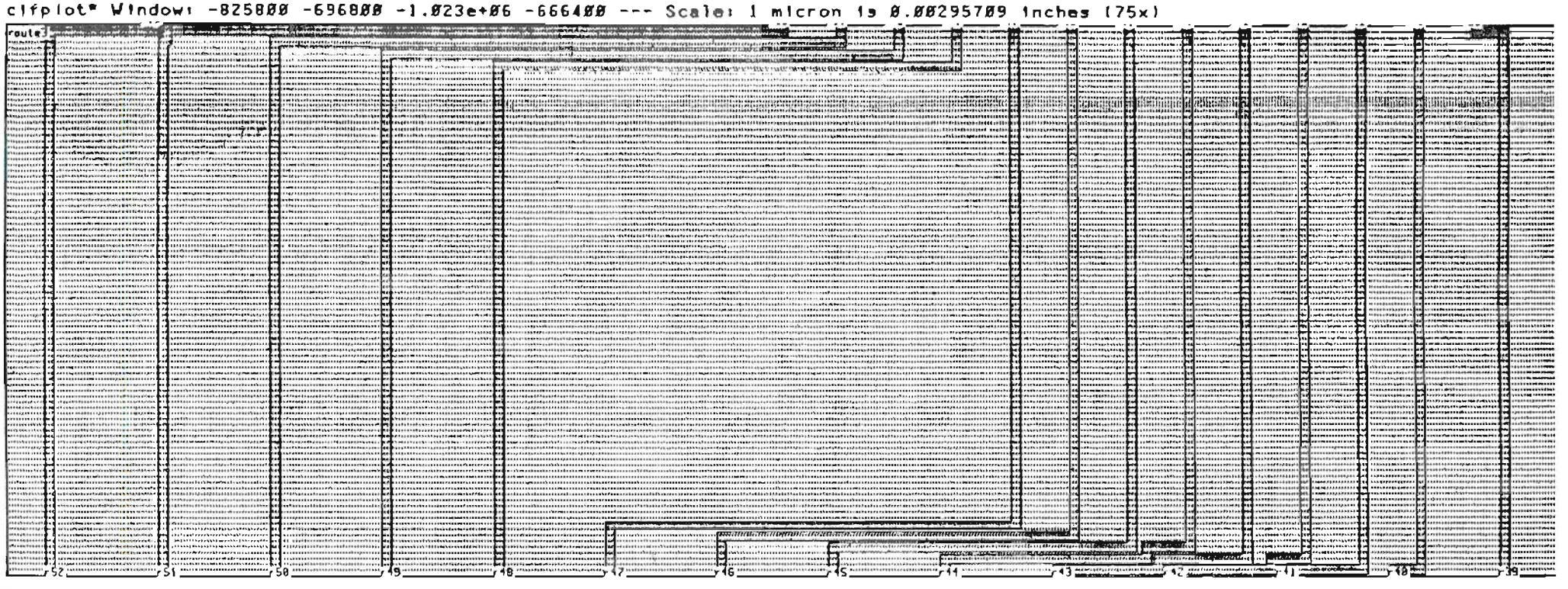


Fig 28. A routing example.

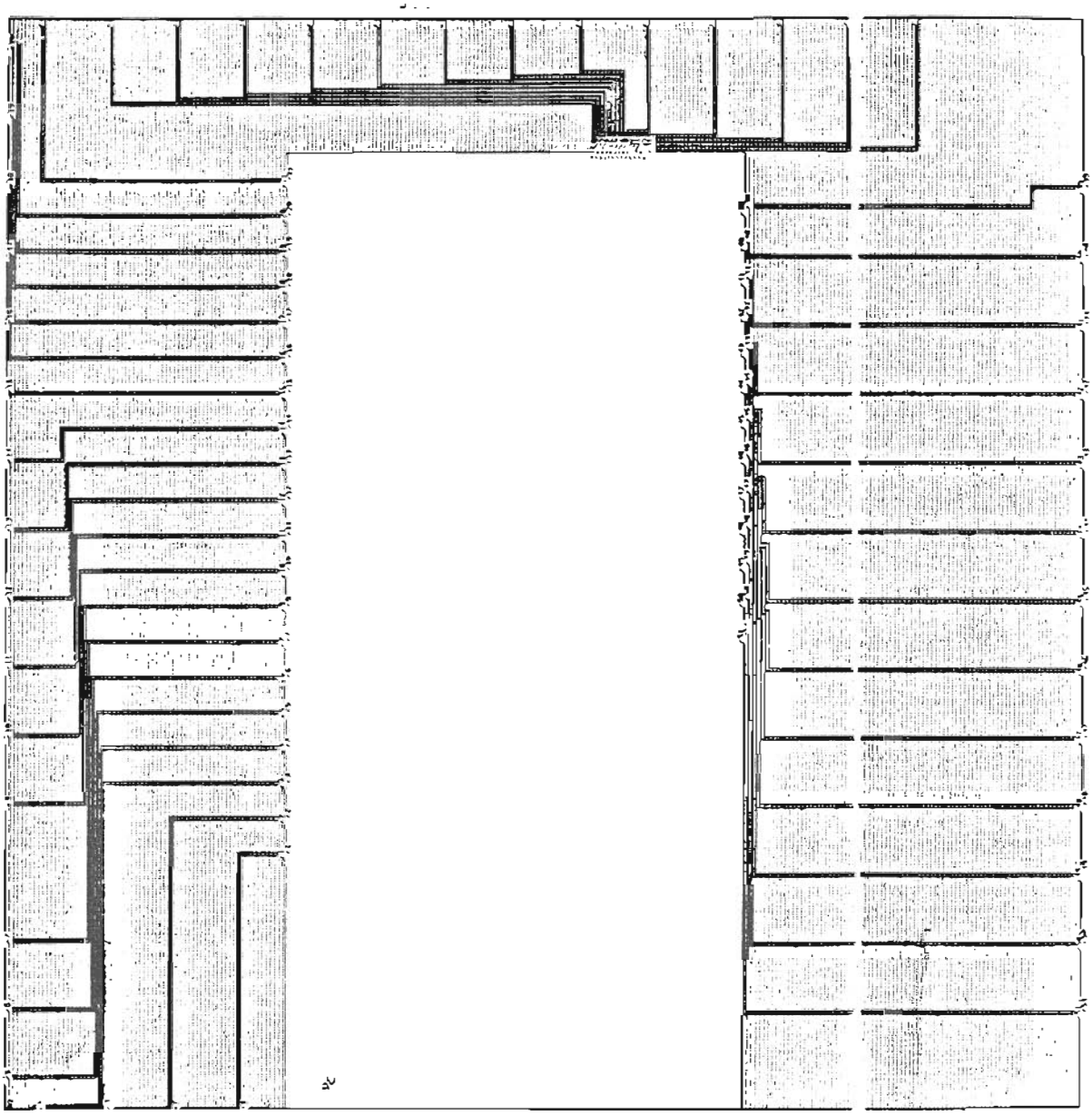


Fig 29. A larger example

REFERENCES

1. Hsu , C.P. : "General River Routing Algorithm", Proceedings of the twentieth IEEE Design Automation Conference, paper 37.1, pp. 578-583.

2. LaPaugh, A. S.:"A Polynomial-time Algorithm for Optimal Routing Around a Rectangle", Proceedings of the twenty first annual IEEE Symposium on Foundations of Computer Science, October 1980, pp. 282-293.

3. Leiserson, C. E. and R. Y. Pinter: "Optimal placement for River Routing". Proceedings of the C.M.U conference on VLSI Systems and Computations, October 1981, pp. 126-142.

4. Pinter, R. Y.: "River Routing Methodology and Analysis", Proceedings of the third Caltech Conference on VLSI, March 1983, pp. 141-163.

5. Mead, C and Conway, L: "Introduction to VLSI systems" Addison-Wesley Publishing Co., Reading, Ma 1980.

6. Ousterhost, John K.:"Caesar: An Interactive Graphics Editor for VLSI layouts", Caesar reference manual for Caesar Version VII.

7. Tompa, M.:"An Optimal Solution to a Wire Routing Problem", Journal of Computer and System sciences, Vol 23, pp. 127-150.

Appendix

Manual page for the router.

NAME

router – automatic river route generator.

SYNOPSIS

router -[p d m [integer]] filename.ca

DESCRIPTION

River Routing refers to the interconnection of pairs of terminals in two rows with the same sequence using a single layer, with no crossovers and no internal blockage

The information required by the router is:

1) The pairs of terminals to be connected.
2) Routing area encompassing the terminals and specifying the area available for routing. The terminals have to lie on the edges of the routing area. The routing area should not contain any totally enclosed patches of the routing layer.

The following long Caesar commands can be used to specify the routing information.

route filename : Creates a new file with a rectangle of the routing layer.

getcell filename : Gets the cell and displays it on the screen.

subedit : Subedit the current cell.

paint t : Paints the area under Caesar's bounding box with the routing layer.

label text : Places the text at the spot specified by the bounding box. See note below.

return : To get out of the subedit.

Note: Pairs of terminals with the same name are connected. Specification of the right edge (Right when looking into the routing area) of a route external to the routing area and ending on its edge, ensures continuity.

The bounding box should be squished to a point when specifying the terminal labels for routing. If not the lower left edge of the rectangle is taken as the coordinate for the terminal.

After creating the routing file , exit from caesar and at the command line give the route command : router -[pdm[integer]] filename.ca. The routing layer has to be specified in the command line. The width and spacing for that layer is read in from a parameter file called routeparam located in ~cad/lib. A default value for width from ~cad/lib is used unless specified in the command line. The route will be added in the input file. The next invocation of caesar will display the route.

The options are :

-p Route using the polysilicon layer.
-d Route using the diffusion layer.

-m Route using the metal layer.
Integral value in lambda of the required width of all routes. Default values are used if a value is not specified.

FILES

~cad/src/local/route.c
"C" Source code for the router.

~cad/bin/route
Executable object code file obtained from using the "C" compiler.

~cad/lib/routeparam
default parameter file for width and spacing of the routed paths.

SEE ALSO

caesar : Graphics layout editor.
M.S. thesis :"River Router for the Graphics Editor Caesar".

AUTHOR

Jaya Holla
Oregon Graduate Center.

DIAGNOSTICS

Error messages are generated if unrecognized flags are found on the command line.

The program checks for pairs of terminals to be routed. An error message is sent to stdout when a mismatch occurs, i.e. a river route in one layer cannot be performed. The contents on the route stack are included in the error message A partial route without minimization up to the mismatched point is written in the caesar file.

Ignores labels not on the routing edge and continues with routing if no other error condition occurs. However an error message gives label names and coordinates of the terminal.

Generates an error message if any routing layer rectangles from the Caesar file do not intersect with the rest of the routing area.

BUGS

The program may give an illegal route when the routing area includes non-contiguous area. e.g. is doughnut shaped.

BIOGRAPHICAL NOTE

The author was born on March 26, 1957 in Bangalore, India. She attended St. Columba High School, Bombay, India and graduated from high school in May 1972. She entered St Xavier's College, Bombay and received her Bachelor of Science degree in Physics and Mathematics in May 1978. The author moved to Cleveland, Ohio in May 1980, and received her Master's in Physics from Case Western Reserve University in May 1982. During that time she also taught a Physics Laboratory course.

In 1982 she began her studies at the Oregon Graduate Center where she completed the requirements for the Master of Science degree in Computer Science, in October 1983.

She is leaving the Graduate Center to accept a job with the Bipolar Design Automation Group of Advanced Micro Devices, Sunnyvale, Ca.