ROUTING ALGORITHM FOR GATE ARRAY MACRO CELLS

Atreyi Chakraverti

Department of ECSE
R.P.I
Troy, NY 12180

Moon Jung Chung

Department of Computer Science
Michigan State University
East Lansing, MI 48824

Abstract

We will present an efficient dynamic algorithm for routing pre-placed gate array macro-cells. A novel data-structure based on corner stiching is introduced to represent the routing environment in a general gate array, where an uniform grid cannot be superimposed on the basic-cells. The near-optimal routing is accomplished in iterations with an initial shortest-path routing followed by conflict resolution using a coloring procedure and net reordering.

1. INTRODUCTION

This paper presents an algorithm developed for "Automatic Book Generation". A dynamic algorithm is described for routing pre-placed gate array function blocks or macro cells, typically consisting of 2-10 basic cells. Routing involves the conversion of the net list of a circuit diagram of the macro cell into a set of metal wires in the gate array which realize the net connections. The resultant metal is called a **book** in industrial terminology. A **basic cell** is a set of transistors, resistors, etc., which is repeated in one or two dimensions to form a gate array.

The two major problem areas involved are the representation of the routing environment in the gate array and the resolution of routing conflicts between different nets. The first problem is dealt with the irregular grid set up for the routing using a corner stitched data structure with grid points. Corner stitched data structure is chosen for its efficiency both in terms of speed and storage requirements.

The second issue is dealt with using a first pass routing to find the shortest path routes for nets input in any arbitrary order. This is followed by conflict resolution using a coloring procedure. Coloring tries to eliminate conflicts between two nets by reducing order dependencies. It is a fairly fast method of verifying whether a wire can be re-routed before actually ripping it up. Coloring is therefore more efficient than attempting to route for all permutations of net ordering. Subsequent routing passes involve hueristic based net reordering. Eventually, vias are constructed for transferring the unresolved conflicts to the second metal layer.

The algorithm attempts to produce an optimal route based on the following criteria :

- The minimum usage of second level metal i.e. minimum number of cross overs.
- The minimum blockage of global channels.
- The minimum wire length, subject to the previous two criteria.

Most commercially available routers are grid-based i.e. they assume that gate array cell contacts and channels are so aligned that an uniform grid can be used to represent the entire cell. Many TTL and ECL gate arrays are exceptions to this category. Figure 1 shows three transistors T1, T2 and T3 in an ECL basic cell. Grid points for routing channels corresponding to T1 and T2 do not align with grid points from T3. This causes difficulties during routing because a large number of unevenly spaced grid points are required to represent grid points corresponding to different transistors. Further, to traverse from one point to another requires pointers between grid points. The number of pointers per grid point will not be fixed. To overcome all these problems, a data structure is required which allows for efficient routing without using excessive storage. The proposed solution is the corner stitched tile structure with grid points within each tile. This allows for efficient traversal of the routing environment while overcoming the problem of unaligned contacts and channels.

Net conflict resolution is another area which conventional routing methods like Moore [9], Lee [8] or line-search [6] algorithms have limited capacity to handle. In the global wiring approach, nets start of by having equal opportunity to compete for routing space. However, at a later stage when congested regions have to be re-routed, the process becomes order dependent [13] and the advantage is partially lost.

The technique adopted here is dynamic routing. This works by modifying one or more routed features and tries to wire unrouted configurations in the new environment. The approach has been successfully employed in the field of PCB routing. Some methods applied in conjunction with dynamic routing include usage of dummy grids [1] and repetition of deletion and routing processes, alternately [5]. Expert systems have also been developed in which blocking situations and their solutions are stored in a knowledge base [12].

2. CORNER STITCHING

The corner stitching data structure [10] is used to represent the routing environment. Its major advantage is that it provides for very fast geometrical operations like locating points, finding neighbors and enumerating areas. These functions form an integral part of the pre-processor as well as the routing algorithm. The data structure,as described here, is limited to Manhattan designs (with horizontal and vertical edges only). The algorithms for operations are generally linear in the number of local objects. Apart from the obvious advantage of speed, the data structure is particularly suited to this application because (i) it allows for compact representation of non-uniform (non-grid based) gate array cells where alignment does not exist between contact rectangles or channels and (ii) it provides a unique decomposition of the routing space.

The basic object in the structure is a **tile** or rectangle. The tiles are of two types, space (for the routing regions) and solid (for the contacts). A plane consists of a collection of tiles, linked together at the corners like a mosaic, as in Fig. 2. Tiles contain their lower and left edges, but not their upper or right edges. Hence, every point in the plane is present in exactly one tile. The entire plane is covered from the minimum to the maximum bounds of x and y. Coverage is achieved by extending the dimensions of the outermost space tiles. Another feature is that space tiles are organized as **maximal horizontal strips**. This means that no space tile has other space tiles to its immediate left or right. The horizontal-strip representation has the property of being unique i.e. there is only one decomposition of space for each arrangement of solid tiles.

The gate array basic cell is a collection of corner stitched planes. Usually, 3 planes can adequately represent the cell. The first plane contains the first level metal i.e. the transistor and resistor contacts as well as the horizontal global channels (for connection amongst different macro cells). The second plane holds the vertical global channels. When unresolvable routing conflicts occur, vias are created and one of the conflicting route segments is transferred to this layer. The third plane contains the power and ground metal lines as well as other fixed metal like reference and source voltages.

A corner stitched representation of the first plane of a 2 micron ECL(Emitter Coupled Logic) NOR gate cell is shown in Fig. 3. Dotted lines indicate space tile boundaries. The cell consists of seven bipolar junction transistors(BJTs) and five resistor banks. C, E and B indicate the collecter, emitter and base contacts of eachof the BJTs respectively. The resistor banks are like po- tentiometers in which connections can be made between any two of R1, R2, R3 to yield different resistance values. There are also 4 metal lines in the third plane i.e. VCC, VEE, VR and VS. VR and VS are reference and source voltages available in Emitter Coupled Logic Circuits.

A description of the basic cell topology is input as a set of diagonal coordinates (Xmin,Ymin,Xmax,Ymax) for each contact in the cell. A corner stitched data set up using this information.

3. IRREGULAR GRID CREATION

This algorithm operates on the corner stitched basic cell to create grid points along which route wires are laid. The non-uniformity and misalignment of the cell contacts does not permit a common grid for the entire routing space. Therefore, grids are created which are local to a corner stitched tile. This allows for two types of movement during route searches - intra-tile and inter-tile. Intra-tile movement is from one grid point to an adjacent grid point in the same tile. Inter-tile movement uses stitches to move to grid points in adjacent tiles.

The first step in grid modeling involves determining the location of **vertical channels** in the tiles. These channels must satisfy the design rule constraints of minimum width(MW) and minimum separation(MS) in space tiles. The x coordinate of these channels is stored in an array field of the tile. The x coordinates are sorted from left to right in ascending order of the array subscript. Due to non-uniformity, the location of **horizontal tracks** must be determined uniquely for each of the vertical channels. The track positions depend on the immediate top and bottom neighbors. Once these are found, grid points are created at the intersection of the tracks and the channels. The track position y coordinates are stored in an array sub-field of the channel (x) array field and are also sorted from bottom to top by subscript. Situations occur where horizontal tracks exist although there are no vertical channels. These cases are also handled.

The grid creation procedure deals separately with solid and space tiles. For solid tiles where more than one grid point cannot be accommodated, a grid point is created at the geometric center of the tile. After solid tiles have been gridded, space tiles are handled. A step wise description of the algorithm is given below :

(1) Enumerate the routing plane. For each space tile encountered, execute the following steps.
(2) Find neighbors along the top and bottom sides (vertical channels can emanate into the space tile from these two sides).
(3) For each bottom neighbor :
(A) Determine the range of X intersection i.e. max to min, with the space tile. The number v of vertical channels depends on whether the bottom neighbor is space or solid.
(B) If the bottom neighbor is solid, # of vertical channels is computed without leaving minimum spacing (MS) on either the left or the right side. That is $v = [(max - min + MS)/(MW + MS)]$. If space , MS is left on both sides, $v = [(max - min - MS)/(MW + MS)]$ These two cases are shown in Figure 4. In both cases, compute the x coordinate of these v different channels. For solid, $x = max - ((k * MS) + (k + .5) * MW)$; $0 \le k < v$ and for space, $x = min + (k + 1) * MS + (k + .5) * MW$; $0 \le k < v4$; If v = 0, take single x value at midpoint of tile. v is stored as the vertical capacity of the tile. (c) For each x computed in (b), find the corresponding top neighbor in the same X range. Determine location of horizontal track corresponding to that vertical channel x. Here, four cases exist.

Case I (the bottom neighbor and the top neighbor are space tiles) : horizontal track does not leave MS at either top or bottom.
Case II (bottom is a space tile but top is not): leave MS at top.
Case III (top is a space tile but bottom is not) :leave MS at bottom.
Case IV (Neither of top or bottom is a space title): leave MS at both top and bottom.

These four cases are illustrated in Fig. 5.
Accordingly, the values of horizontal capacity at a particular channel and y coordinates of tracks are calculated for each of the cases.
(4) All the steps in (3) are repeated for top neighbors with the distinction that the corresponding bottom neighbor is found in step (3)(c). This is required to determine any new vertical channels which may be emanating from the top neighbors.
(5) The x values obtained from (3) and (4) are sorted in ascending order i.e.from left to right.
(6) Each x value is compared with the x values to its immediate left and right. This is to check for conflicts arising from channels (from opposite sides) which are too closely spaced. Tags are set if conflicts occur.
(7) Finally, grid points are created at the intersection of tracks and channels.
(8) At higher levels, a global routing grid can be created if information about track and channel locations is known.

This algorithm sets up the grid structure for routing purposes.

It also provides key information about the vertical and horizontal capacities of a tile. Sideways conflicts between channels are detected in this stage.

4. SHORTEST PATH ROUTING AND NET ASSIGNMENT

The routing procedure is accomplished in two or more passes, as found necessary. Each pass consists of a shortest path determination stage and a conflict resolution stage. The first stage is described in this chapter.

Initially, the net lists of the "book" are input in any arbitrary order. The first pass involves using shortest path methods to assign these multi-terminal nets, in the order they are input. Within each pass, route conflicts between net pairs are resolved by a technique called **coloring**. Between successive passes, nets are reordered depending on the number of conflicts in the previous pass. Only those permutations of nets which result in a reduction in the number of conflicts are retained. The number of passes is bound to prevent unlimited computation.

4.1. SPATH Algorithm

The input to the algorithm is the net list of the macro cell to be implemented. The objective is to find minimal cost paths between members of each net. The search technique used is based on the A* Algorithm [4]. This is a heuristic based, branch-and-bound search method of complexity $O(N^2)$, where N is the total number of nodes in the circuit. The algorithm SPATH finds the shortest path between any two terminals using these principles.The search is constrained by the fact that no two net routes can occupy the same horizontal track or vertical channel at a particular grid point. However, it is possible for wires of different nets to intersect orthogonally at a grid point. This restricts the net conflicts to cross-overs between wires and simplifies the processes of conflict resolution and via assignment.

The SPATH search is based on the concept of minimizing a generalized cost function. In our case, the cost function is a measure of the wire length required for the route path. At any point, it is equal to the Manhattan distance between the source and destination terminals for a path through that grid point. The function can be extended to include a number of factors e.g. global channel blockage, number of jogs, etc.

The algorithm employs three lists i.e. **open**, **closed** and **node**. The first list, open, contains all grid points on the frontier of the search (from which the search can be expanded). The closed list consists of those points which have already been encountered during the search. The node list stores the x,y coordinates and the cost function value (md) for each grid point , whether on the closed or open list. This information is not duplicated in the other two lists.

To begin with, the source terminal is placed on the open list and its cost is computed. Next, all its **successors** are found and it is transferred to closed. Successors are all the grid points that can be accessed from the grid point currently being expanded. The Manhattan cost is calculated for each of them. These successors are added onto the open list following which, the open list is sorted in ascending order of cost. The search is now expanded from the first(lowest cost) element on the open list and the process repeats. If a successor is already present on the closed list, its old and new cost values are compared. The grid point is brought back on to open if its new cost is lower. The search progresses in this manner until the destination is reached or the open list becomes empty (failure). Grid point capacity information is updated when a path is determined.

To find the sucessor node, the grid point is expanded in all four directions i.e. +x,-x,+y and -y, for possible successors. This requires intra-tile or inter-tile movement. If the nearest grid point in a direction is within the same tile it can be accessed by simply changing the subscript of the channel or track array. Otherwise, the adjacent grid point is located in the nearest neighbor in that direction. Each grid point does not have four successors. This is because movement in one or more directions may be blocked. Such blockages can be detected by checking the horizontal and vertical capacities at the point. This eliminates a lot of unnecessary conflicts although the length of the path usually increases.

4.2. NETASSIGN

For routing a general multi-terminal net, the methodology is adapted from graph theory. The problem is analogous to that of generating a minimum-spanning-tree of a graph. NETASSIGN applies the nearest neighbor minimum-spanning-tree algorithm [11] to multi-terminal net assignment.

All the terminals in a net are divided into two categories, routed and unrouted. Initially, all net members belong to the unrouted category. The algorithm starts with any arbitrary terminal in the net and puts it in routed. Two arrays, **near(t)** and **dist(t)**, are maintained for the unrouted (t) terminals. The near(t) entry stores the member (terminal or grid point) of routed which is closest to the unrouted terminal t. The distance from t to near(t) is computed and stored in dist(t). The terminal with the smallest value of dist(t) is chosen as the next candidate for routing. SPATH is used to find a route between this t and its near entry. When a path is found, all grid points on the path are also entered in the routed category as they are now potential connection points. At this time, values of near and dist are recomputed and updated for the remaining unrouted terminals. If SPATH is unsuccessful, a route is attempted with the next nearest routed member. The process continues till all terminals in the are routed.

NETASSIGN produces a good initial route which can be modified for improvement. An attempt is made to resolve conflicts between two nets using the method described in the next chapter i.e. coloring.

5. COLORING AND CONFLICT RESOLUTION

One of the goals during routing is to minimize the number of cross-overs between wires and hence reduce the usage of second level metal. This involves the removal of conflicts between nets. This chapter describes the use of coloring to resolve conflicts between net pairs. Coloring has been adopted from Printed Circuit Board (PCB) Routing where it has met with good results [7]. Coloring is a part of the general dynamic programming procedure which can be outlined as follows :

For every route (r1) that has conflicts :

(1) Find a conflicting wire (r2).

(2) Check routability of r1 in the absence of r2.

(3) Check re-routability of r2 (by a new path).

(4) If the checks in (3) and (4) yield positive results, rip-up r2 and re-route.

5.1. Coloring

On the first routing plane, there are some space regions which remain vacant after the NETASSIGN procedure. The purpose of coloring is to identify adjacent vacant regions which share a common track or channel and assign the same color to them. This implies that after coloring, if any two vacant regions have the same color, it is possible to route between them. Hence, a feasibility check can be made before wires are actually ripped-up and re-routed. Prior to coloring, a **splitting** algorithm is executed to subdivide space tiles along existing route connections.

5.1.1. Splitting

During the splitting process, each tile along a route is subdivided into regions which are separated by a route wire. The corner stitched data structure remains the same. However, there is additional information stored in each tile now. A route passing through a tile may break it into regions which are not rectangles (see Fig. 6.). Whatever the shape of the region may be, it can always be stored as a combination of rectangles. Accordingly, the data structure to represent the regions is as described below.

Each tile contains the follwoing informations: (i) the number of different regions the tile contains (ii) an array which holds the details about the shape and color of each region in the tile.

The splitting algorithm is as follows :
(1) For each existing route segment, do the following steps.
(2) Scan tile by tile along the segment.
(3) For each space tile encountered, find the grid points from where the wire segment leaves or enters the tile.
(4) Find the sides of the tile which this segment intersects. Fig. 7. shows the possible six cases depending on the entry and exit sides.. For each case, we split into regions I, II and III. There are three possible configurations, one in which two regions result and the others with four regions.
(5) If the configurations in Fig. 8.a arise, they are treated in the manner shown for the first configuration,in Fig.8.b
(6) If the tile has already been split by another wire, find the region which the new wire passes through. Apply steps (3) and (4) with respect to this region instead of the whole tile.

5.1.2. Coloring Algorithm

The coloring algorithm is linear in the number of regions. It works as follows.
(1) Once all tiles have been split, enumerate the plane.
(2) For each space tile visited, consider all its regions. For each region do steps (3) and (4).
(3) Locate the region.
(4) From each boundary grid point of this region, proceed to all accessible grid points (in all four directions) in a manner similar to the successor generation procedure of SPATH. Locate the region for the accessible grid point. If this is different from the region found in (3), assign the same color to both regions.
(5) Return to (2) until all space tiles have been colored.

General Conflict Routing Scheme is invoked after NETASSIGN. The scheme uses the Splitting and Coloring algorithms. It operates as follows :
(1) Find all conflicts between two wires, r1 and r2.
(2) For each conflict, call the splitting algorithm, considering all wires except r1.
(3) Execute the coloring algorithm.
(4) Test segments of r1 in increasing size, originating from the conflict site. Locate the endpoints of the segment. If the endpoints belong to regions of the same color, perform a modified SPATH route. This altered routine sp, functions in the same way as SPATH except that it restricts paths to regions of one particular color.

5.2. Multiple Net Conflicts

Two approaches were considered for re-ordering the nets between passes in multiple conflict cases. One of the strategies involved ripping-up nets in the order of maximum conflict first. The other idea was to rearrange nets in terms of length. Since the latter approach does not leave much scope for iteration, the first method is used.

An array **order** stores the different permutations of the net ordering. It also stores the number of conflicts associated with each ordering. After NETASSIGN is applied to the initial net ordering we iterate the following loop.
(a) Apply the routing algorithm described in chap 4.
(b) If the current order is better than all previous ones, it is stored. Otherwise it is discarded. i.e. those orders that decrease conmin are stored.
(c) If current order is retained, its nets are sorted in increasing number of conflicts.
(d) The maximum conflict net is interchanged with the next highest net to obtain a new order.
(e) NETASSIGN is performed for this new order.
(f) Return to (a) with this as the current order, if loop condition is satisfied.

6. RESULTS

The corner stitched data structure represents all space explicitly. The utility of the data structure in terms of storage requirements is tested by the ratio of the number of space tiles to the number of solid tiles. In the worst case, it has been shown that a maximum of 3N + 1 space tiles is required for N space tiles. But ratios between 1 and 1.5 are observed for circuits with 2 and 4 basic cells. Each cell has 7 transistors and 5 resistor banks.

The timing requirements (on the SUN 2_160) for all the algorithms, applied on the 2 basic cell circuit in Fig. 9 are given in Table 1. In the 2 basic cell macro, the coloring procedure resolves 2 conflicts. For 4 basic cell macro with 4 initial conflicts, our algorithm routed it with it with 2 unresolvable conflicts. Both these compare favorably with manual route results. The executable file r has a bss (size of uninitialized data segment) of 16.13 Megabytes for the four basic cell circuit. The main memory requirement stems from the coloring procedure and the dynamic rerouting. The corner stitching data structure is efficient in terms of storage.

7. Conclusion

A new data structure and algorithm to perform automatic routing of gate array macro cells have been presented. These have been tested on ECL macro cells containing 2-4 basic cells. The route results are comparable to manual wiring with a great reduction in the time required for routing. The storage requirements of the data structure and the timing of the algorithms on the SUN workstation show the efficiency of the algorithm.

BIBLIOGRAPHY

1. Bollinger H., "A Mature DA System for PC Layout," *Proc. International Printed Circuits Conference*, 1979, pp.85-99.
2. Chang K.C. and Du H.C., "On Three Layer Via Minimization," *Proc. Int. Conf. on Computer Design*, 1987, pp.274-276.
3. Chen Ruen-Wu et.al., "A Graph Theoretic Via Minimization Algorithm for Two-Layer Printed Circuit Boards," *IEEE Trans. on Circuits and Systems*, Vol.CAS-30, No.5, May 1983, pp.284-299.
4. Clow G.W., "A Global Routing Algorithm for General Cells," *Proc. 21st. Design Automation Conference*, 1984, pp.45-51.
5. Finch A.C. et. al., "A Method for Gridless Routing of Printed Circuit Boards," *Proc. 22nd. Design Automation Conference*, 1985, pp.509-515.
6. Hightower D.W., "A Solution to Line-routing Problems on the Continuous Plane," *Proc. 6th. Design Automation Workshop*, 1969, pp.1-24.
7. Kawamura et. al., "Hierarchical Dynamic Router," *Proc. 23rd. Design Automation Conference*, 1986, pp.803-809.
8. Lee C.Y., "An Algorithm for Path Connections and its Applications," *IRE Trans. Electron. Comput.*, Vol.EC-10, 1961, pp.346-365.
9. Moore E.F., "The Shortest Path Algorithm through a Maze," *Bell Tel. Lab. Rep.*, 1959.
10. Ousterhout J.K., "Corner-stitching : A Data Structuring Technique for VLSI Layout Tools," *IEEE Trans. on CAD/ICAS*, Vol.CAD-3, No.1, Jan. 1984, pp.87-99.
11. Reingold E.M., et. al., *Combinatorial Algorithms*, Prentice Hall, 1977, pp.323-326.
12. Robert L.J., "An Expert Approach to Completing Partially Routed Printed Circuit Boards," *Proc. 22nd. Design Automation Conference*, 1985, pp.523-528.
13. Ting B.S. and Tien B.N., "Routing Techniques for Gate Array," *IEEE Trans. on Computer Aided Design*, Vol.CAD-2, No.4, Oct. 1983, pp.301-312.

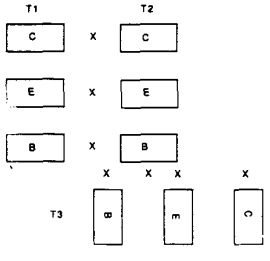


Figure 1 Off gridness in basic cell

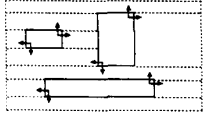
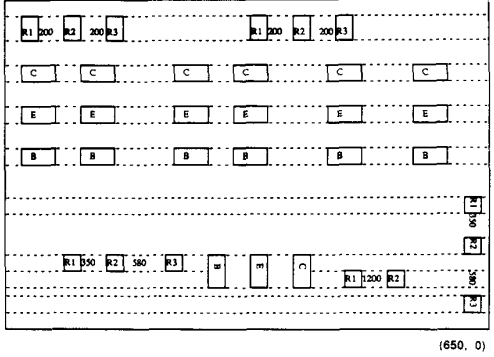


Figure 2. a corner stitched plane

Fig. 3.



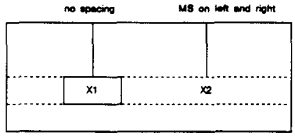


Figure 4. vertical channels

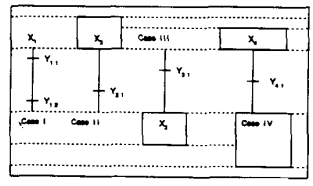


Figure 5. Location of horizontal tracks

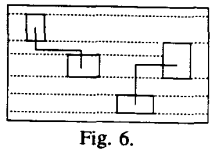


Fig. 6.

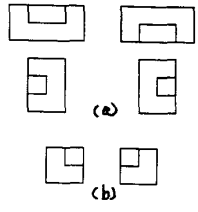


Fig. 8.

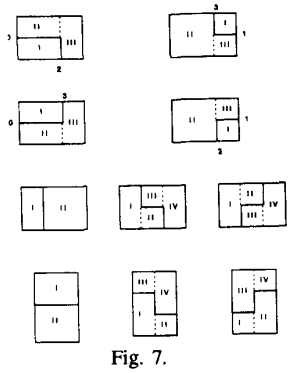


Fig. 7.

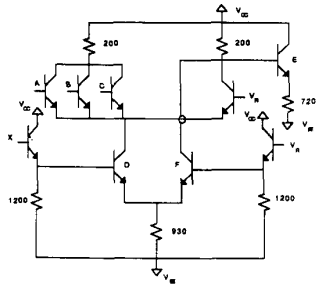


Fig. 9

Algorithm	CPU time (sec)
Tile Create (175 tiles)	26.9
Grid Create (105 tiles)	84.6
Color (105 tiles)	580.64
Netassign (23 connections)	226.72
Via (2 conflicts)	10.76

Table 1.

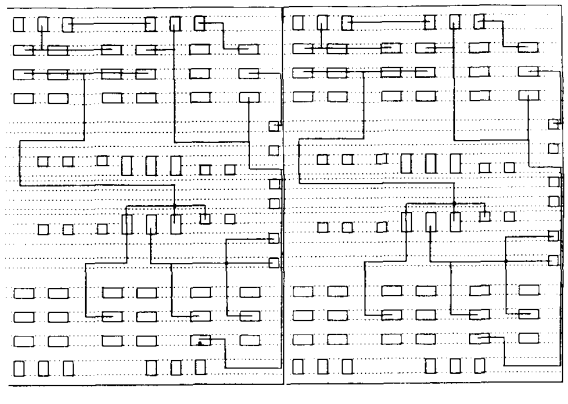


Figure 10 Final route of 4 cell macro