A Channelless, Multilayer Router

R. Eric Lunow

Lawrence Livermore National Laboratory
6000 East Avenue
Livermore, California 94550

Abstract

We have implemented a channel-less, gridless, multilayer router as part of the Magic IC layout system. The router is designed to handle new routing problems associated with emerging technologies such as Wafer Scale Integration and multilayered metal processes. Three features distinguish this router: rectilinear Steiner trees with *floating segments*, a *routing scheduler*, and a *corner-stitched* database.

1. Introduction

Advancing technologies present increasingly complex routing problems requiring support for multiple metal layers and wafer-scale integration. Conventional grid-based channel routing does not lend itself well to technologies providing three or more routing layers. This paper presents a new router with features including:

- Support for an arbitrary number of routing layers.
- Gridless Design.
- No restriction on the location of pins to be connected.
- Obstacle avoidance.

2. Highlights

This router incorporates three key ideas:

First, nets are represented as rectilinear Steiner trees with *floating segments*. Segments of a Steiner tree correspond to wire segments of a net and the relationship among segments defines the topology of the net. A floating segment is bound to a routing layer but not to a specific location within that routing layer – it may *float* within a range of values. As the routing progresses, floating segments are bound to fixed locations. By preserving the floating segment representation throughout the routing process, it is easy to allow nets to be partially ripped up, enabling the router to explore alternate routes.

Second, a *routing scheduler* applies a set of routing strategies to a queue of partially routed nets. Each strategy attempts to further the routing of a net. Simple strategies are applied first followed by more complex strategies to resolve difficult problems. The orchestration of routing strategies is controlled by a *strategy table*, which is a specification of a finite state machine. The use of strategy tables is analogous to the controlling expert in a system such as Weaver [1], but without the penalty typical of expert systems. Routing strategies are typically simple: generating coarse routes, laying wires in routing regions, maze routing through congested areas, and ripping up obstructing segments.

Third, a corner-stitched database [5] represents obstacles and wiring as the routing progresses. Corner-stitching is a technique for representing rectangular two-dimensional objects – it provides a representation for spatial relationships resulting in fast, efficient algorithms for searching, creating, and deleting objects. By using corner-stitching, the router can operate in the physical,

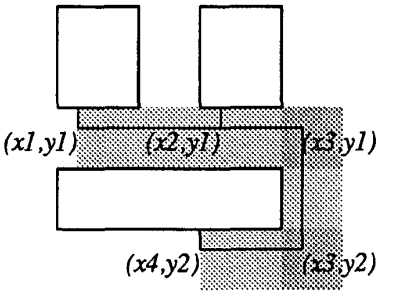


Figure 1. A rectilinear Steiner tree defining the coarse route of a three terminal net. The shaded areas delineate ranges of floating segments.

rather than the symbolic domain; this ability is crucial to the router's obstacle-avoiding role.

Our approach avoids several key limitations of conventional global-then-channel routing methods:

- The router works on a net at a time, as opposed to a channel at a time, but need not fully route a net before advancing to other nets. At any given time, a net may be partially routed or partially ripped up. The router thus has the flexibility to change its mind about the routing of a net as the routing problem becomes more fleshed out.
- Generating coarse routes is a routing strategy, not a distinct routing phase. New coarse routes can be generated dynamically to avoid congested areas.
- Routing regions are not constrained to have pins only on the top and bottom. Pins may be in the interior of routing regions, i.e., to connect terminals in the interior of a cell to a routing layer above the cell.

3. Fundamental Data Structures

Figure 1 elaborates on the concept of floating segments. Each segment is associated with a routing layer and two end points; connecting segments share common end points. Individual x and y coordinates float within the range illustrated by the shading. Floating segments gravitate toward the high, low, or middle of the range depending on the topology of the net. This *gravity* pulls segments in directions which reduce wire length.

The physical layout of nets and obstacles are represented in a corner-stitched database. Each routing layer corresponds to a separate tile plane. As floating segments are assigned locations, tiles are written to their respective planes to reserve space. Tiles are written with an area equal to the sum of the actual wire area plus inter-wire spacing required to satisfy design rule constraints. Hence, abutting tiles occupy sufficient space for both wires and inter-wire spacing, so the router generates design rule correct solutions without explicitly addressing design rule constraints.

4. Control Flow

Routing has three major phases:

- An initialization phase which decomposes the problem into routing regions and generates initial Steiner trees for each net's coarse route.
- A routing phase which executes the routing strategies.
- A post-processing phase which improves the quality of the routing.

4.1. Initialization Phase

In the initialization phase, the routing area is decomposed into regions on a per-layer basis. The router supports an arbitrary number of routing layers which are fixed and specified in a technology file. Each routing layer is assigned a primary orientation, either horizontal or vertical. Adjacent layers have alternating orientations.

For each routing layer, existing material is expanded, merged, and shrunk to create *protection frames* [2]. These protection frames define areas which cannot be used by the router. The protection frames are then classified into two groups: major obstacles and minor obstacles. Major obstacles are all larger than a given percentage of the largest obstacle; they are used to define the routing regions. All other obstacles are minor; these are represented as tiles within the routing regions.

The *routing regions* are the maximal horizontal and vertical space tiles on each routing layer after the major obstacles have been written to their respective tile planes. By merging pre-existing material into larger objects and focusing only on the largest objects, one achieves a reasonable per-layer decomposition without excessive fragmentation. A per-layer decomposition makes routing areas available within and above cells if there is no existing material on those layers.

Routing regions are further partitioned into *zones*. A zone is formed by the intersection of each pair of routing regions on adjacent layers; it provides a finer granularity for maintaining congestion and connectivity information.

Initial coarse routes are generated for each net. Coarse routes are represented as Steiner trees, which are generated using the algorithm described in section 4.2.1.

At the end of the initialization phase, nets are sorted in descending order of complexity and placed in a queue for processing during the routing phase. The complexity of a net is the maximum number of segments sharing a common coordinate and represents the greatest constraint placed on the net.

4.2. Routing Phase

The routing phase utilizes a number of routing strategies to bind floating segments to locations within routing regions. The *routing scheduler* orchestrates the execution of routing strategies. This scheduler is similar to an operating system job scheduler: it gets the next net from a queue, applies a routing strategy to that net, and then re-queues the net for additional processing based on the results of that routing strategy.

A routing strategy is an algorithm which, singly or in combination with other other strategies, effects the routing of a net. Typical routing strategies are described below.

- *Coarse Routing* generates new Steiner trees to avoid congested areas.
- *Region Routing* routes a net by placing straight line wires corresponding to segments of the Steiner tree.
- *RipUp Routing* removes previously laid wires to make room for other nets.
- *Jog Routing* resolves conflicts by introducing short orthogonal jogs in straight line segments.
- *Push Routing* plows previously routed wiring to make room for other nets.
- *Maze Routing* finds paths through congested areas.

The routing scheduler effects transfers of control by interpreting a *strategy table*. The strategy table is a specification of a finite state machine and queueing operations – it defines a set of states, transitions between states, and LIFO or FIFO queueing operations. Each state is associated with a routing strategy. Each transition corresponds to a return code from that routing strategy. As each net is processed, it is placed either on the front or rear of the queue for processing on subsequent scheduler cycles until it is fully routed or has exceeded its limit of scheduler cycles.

There may be many strategy tables. Different tables may be used for different types of routing problems. A typical finite state machine controlling the execution of routing strategies is shown in Figure 2.

Routing strategies are described in the following sections.

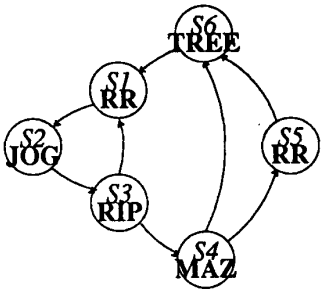


Figure 2. A typical finite state machine controlling the execution of routing strategies. In this example there are six states, *S1* through *S6*. *S1* is the initial state. Each state is labeled with an acronym indicating the routing strategy – **RR** indicates *Region Routing*, **JOG** indicates *Jog Routing*, **RIP** indicates *RipUp Routing*, **MAZ** indicates *Maze Routing*, and **TREE** indicates *Coarse Routing*. This particular FSM lays straight line wires, tries short orthogonal jogs, and then rips up pre-existing wiring, in that order. If nothing can be ripped up, it tries to maze route, and if that fails, generates a new coarse route and repeats the process again.

4.2.1. Coarse Routing Strategy

The *Coarse Routing* strategy generates new Steiner trees.

The algorithm for generating Steiner trees uses a combination of Lee's algorithm for maze routing via wavefront propagation [4] and Kruskals's algorithm for finding the minimum spanning tree of a graph [3]. The terminals in a net are assigned to initial points in routing regions. If extending a terminal into a routing region violates the orientation of the routing layer, vias are placed to locate the terminal in routing regions in the layers above and below. If this is impossible, orthogonal routes are allowed but penalized. Initially, each terminal is a separate *component*; a *component* is a set of one or more terminals which have been connected. This initial set of points constitutes a *wavefront* which is expanded into adjacent routing regions. As each component is expanded, it is compared with existing points of the same component in the same routing region. Expansion points of the same component and zone but higher cost are dropped. Following this expansion, affected routing regions are examined for two or more points belonging to different components. Points belonging to different components represent a potential segment of the Steiner tree. The minimum

cost segment is selected, the two components are merged into one, that portion of the Steiner tree is generated, and the intermediate Steiner points are re-assigned a zero cost. This process repeats until there is only one component outstanding or no components can be merged. Then the points on the wavefront are expanded again. The cost of the Steiner tree is the sum of the costs of each segment.

The wavefront propagation is similar to Lee's algorithm with the following exceptions:

- Wavefronts are expanded into routing regions, not grid points; costs are approximate but runtimes are reduced by several orders of magnitude.
- All components are expanded simultaneously.
- The net bounding box initially bounds the expansion of the wavefront.

4.2.2. Region Routing Strategy

The *Region Routing* strategy assigns straight line wires to floating segments. This involves the identification of floating coordinates, selection of candidate values, and examination of related segments for conflicts.

Each segment is defined by two end points, *(x1,y1)*, and *(x2,y2)*. The Steiner tree is searched for segments for which the assignment of one of the coordinates (*x1*, *y1*, *x2*, or *y2*) will fix both end points. These can be segments having one endpoint fixed and one coordinate of the other endpoint fixed, or vertical segments with both *y* coordinates fixed, or horizontal segments with both *x* coordinates fixed (the Manhattan geometry guarantees that vertical segments share *x* coordinates and horizontal segments share *y* coordinates).

The unassigned coordinate is identified and candidate values are considered in an order determined by the coordinate's *gravity* and *range*. For low gravity the range is searched from the minimum to the maximum. For high gravity the range is searched from the maximum to the minimum. For midrange gravity the range is searched from the center outward alternating toward the minimum and maximum.

As each candidate value is considered, segments referencing that coordinate are examined for conflicts. This may require examination of many segments on several routing layers. If no conflicts occur, the coordinate is assigned that value and tiles are written to the database reserving space for the wires and inter-wire spacing. The corner-stitched database enables these operations to be performed efficiently.

Not all points in the range are considered. A value is advanced according to conflicts encountered. Conflicts are classified as *soft* or *hard*. A *soft* conflict is caused by an obstacle which can be avoided by advancing the candidate value; a *hard* conflict is caused by an obstacle which cannot be avoided by advancing the value. *Soft* conflicts are resolved by advancing the candidate value beyond the boundary of the obstacle. *Hard* conflicts are resolved by backing off the last assigned coordinate in the conflicting segment. The algorithm always advances a value within its range – it never retries previously attempted values but may undo and advance a previous assignment.

The algorithm terminates when the net is fully routed, a value has been advanced beyond its range, or a *hard* conflict occurs which cannot be resolved.

4.2.3. RipUp Routing Strategy

The *RipUp Routing* strategy removes wiring blocking a net. It unconditionally rips up wiring preventing a terminal from entering a routing region and conditionally rips up wiring in other conflicting situations.

Wiring is ripped up by backing off either the *x* or *y* coordinate of the obstructing segment. This has the effect of shortening or moving the obstacle depending on the chosen coordinate and the segment's orientation. In situations caused by horizontal and vertical constraints, it exploits floating segments by ripping up in a way which moves the obstructing wire to one side, thereby eliminating the constraint. If this is not possible, it rips up in a way which shortens the obstructing wire.

Ripped up nets are placed in the routing scheduler's queue for subsequent re-routing. There is no distinction between ripped up nets and partially routed nets.

4.2.4. Jog Routing Strategy

The *Jog Routing* strategy provides obstacle avoidance by introducing jogs in straight wires. This is a simple non-destructive method for resolving constraint violations. Jogs are short wire segments running orthogonal to the primary wiring orientation or long wire segments running on adjacent routing

layers.

4.2.5. Push Routing Strategy

The *Push Routing* strategy moves wires to create open space. It moves adjacent wires in tandem while maintaining connectivity. This is comparable to *plowing* [6] or *weak modifications* used by the Mighty router [7].

4.2.6. Maze Routing Strategy

The *Maze Routing* strategy finds paths through congested areas. Selected segments of the Steiner tree are severed to produce independent, routable sub-nets. A Lee style point expansion algorithm finds paths connecting the sub-nets. The number of wavefront expansions is restricted. Simple solutions are quickly found and complex problems are deferred for an external point-to-point maze router.

4.3. Post-Processing Phase

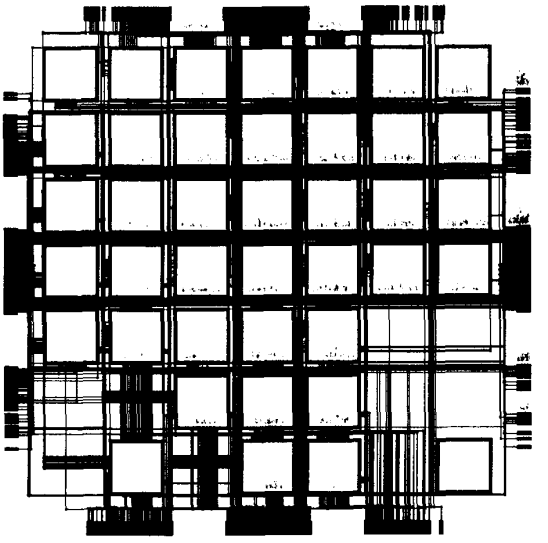
The post-processing phase completes routes deferred by the *Maze Routing* strategy and improves the quality of routing. An specialized point-to-point maze router completes routing for complex problems. Vias are minimized by moving wire segments between layers. Each net is examined for simple patterns in which a wire segment on one layer connects two segments on an adjacent layer. If the connecting segment is not obstructed by material on the adjacent layer, it is moved to that layer and the connecting vias are eliminated. Studies have shown this simple algorithm eliminates approximately 20% of the vias.

5. Conclusions

Results are shown in Example 1 below. The example is an experimental hybrid WSI design. 1,704 nets were routed on four layers in 1,198 cpu seconds on a SUN 3/260. Elapsed time was 23 minutes.

6. Acknowledgements

Mike Arnold and Walter Scott participated in discussions resulting in this work and provided comments on drafts of this paper. Berry Kercheval also reviewed drafts.



Example 1. A wafer-scale design.

7. References

[1] Joobbani, R. "An Artificial Intelligence Approach to VLSI Routing", Kluwer Academic Publishers, Mass., 1986.

[2] Keller, K. H. "A Symbolic Design System For Integrated Circuits", Proc. 19th Design Automation Conference, June, 1982.

[3] Kruskal, J. B. "On the shortest spanning subtree of a graph and the traveling salesman problem", Proc. Amer. Math Soc. 7:1, 48-50

[4] Lee, C. Y. "An Algorithm for Path Connections and Its Applications", IRE Transactions on Electronic Computers, VEC-10, September, 1961.

[5] Ousterhout, J. K. "Corner Stitching: A Data-Structuring Technique for VLSI Layout Tools", IEEE Transactions on Computer-Aided Design, Vol CAD-3, No. 1, January, 1984.

[6] Scott, W. and Ousterhout, J. "Plowing: Interactive Stretching and Compaction in Magic", Proc 21st Design Automation Conference, June 1984.

[7] Shin, H. and Sangiovanni-Vincentelli, A. "MIGHTY: A 'Rip-Up and Reroute' Detailed Router", Digest of Technical Papers, IEEE Conference on Computer-Aided Design, 1986.