

An Obstacle-Avoiding Router for Custom VLSI

By

Gordon Taro Hamachi

A.B. (University of California) 1977

M.S. (University of California) 1982

DISSERTATION

Submitted in partial satisfaction of the requirements for the degree of

DOCTOR OF PHILOSOPHY

in

Computer Science

in the

GRADUATE DIVISION

OF THE

UNIVERSITY OF CALIFORNIA, BERKELEY

Approved:.....

Chairman

4/23/86

Date

4/23/86

4/23/86

.....

An Obstacle-Avoiding Router for Custom VLSI

Gordon Taro Hamachi

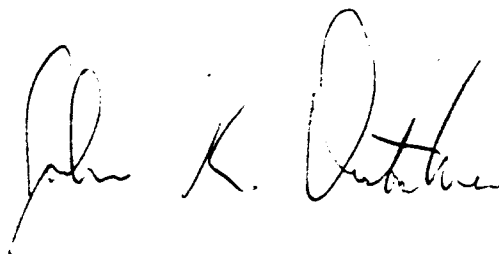
Abstract

Magic's automatic routing system combines the flexibility of hand-routing with the speed and quality of automatic channel routers, by allowing chip designers to prewire selected nets by hand. The router then works around this previously-placed layout, called obstacles, to automatically wire the remaining nets. This ability to partially hand-route an integrated circuit gives designers complete control over critical paths, power and ground routing, and other special nets. At the same time the router provides a fast way to make the remaining connections in the design.

The system's novel features include a fast channel decomposer, an obstacle-avoiding global router, and an obstacle-avoiding switchbox router. The router's channel decomposition algorithm relies on a corner-stitched data structure to efficiently produce a small number of large channels. The global router considers obstacles during path generation, trading-off net length and channel complexity to simplify the subsequent channel routing task. While able to cope with obstacles, Magic's switchbox router is still comparable to the best traditional (non-obstacle-avoiding) channel routers.

The router's obstacle-avoidance features rely on two underlying concepts: (1) a preferred direction for crossing an obstacle, and (2) hazards, or areas the routing should avoid. Crossing obstacles in the preferred direction minimizes the creation of blocked areas, which can not be crossed by other routing; this minimizes obstacles' impact on the automatic routing. Crossings in preferred directions are controlled by strategically-placed hazards adjacent to obstacles.

Measurements show that obstacle-avoiding routing is both useful and practical: hand-routing improves the electrical characteristics of the selected nets, while the hand-routed obstacles have only minor effects upon the routing quality of a design as a whole. The improvement in electrical characteristics is due to the decreased net length and increased attention to layer selection possible when nets are prewired by hand.



Dedication

In memory of Arnold Philip Tapie

Acknowledgements

Special thanks to John Ousterhout, who stresses the value of stating the most important ideas first. In his typical style, John comes very close to being the ideal research advisor. Working with him has been a very rewarding experience.

I would like to acknowledge a number of others for their technical assistance. First, it has been a great pleasure to work with the other members of the Magic group: Robert Mayo, Walter Scott, and George Taylor. Thanks to the many other tool builders for making my job easier, especially Michael Arnold, Dan Fitzpatrick, Howard Landman, and Barry Roitblat. Randy Katz served on my qualifying exam committee and provided valuable guidance. As members of my thesis committee, Alberto Sangiovanni-Vincentelli and Charles Woodson read several drafts of my dissertation and provided helpful and insightful comments on its content, organization, and presentation. Fred Obermeier and David Wallace also reviewed drafts of this thesis. David Patterson and Carlo Séquin supervised my academic progress and research early in my career as a graduate student. Franky Leung and David Ashkenas designed the chips used to debug the router and measure its performance.

Finally, I would like to express my gratitude to several persons for their important personal support and encouragement. I am indebted to my parents, Ted and Alice, who designed and fabricated me. Similar thanks to my brothers and sisters--Karolyn, Warren, Margaret, and Ken--who tested me. Thanks also to Dale Beucler, Scott Evans, David Spears, Lewis Yee, and Alan Yung, for their enduring friendship.

The work described here was supported in part by SRC under grant number SRC-82-11-008.

Table of Contents

CHAPTER 1. Introduction

CHAPTER 2. Approaches to the Routing Problem

2.1	Introduction.....	9
2.2	Net-At-A-Time Routers.....	10
2.2.1	The Lee Router.....	11
2.2.2	The Hightower Router.....	13
2.3	Channel-Based Methods.....	15
2.3.1	The Left Edge Algorithm	17
2.3.2	Deutsch's Dogleg Router.....	19
2.3.3	Yoshimura and Kuh	20
2.3.4	YACR-II.....	21
2.3.5	Burstein's Hierarchical Router.....	21
2.3.6	Rivest's Greedy Router.....	22
2.4	Lack of Flexibility.....	23

CHAPTER 3. Magic's Obstacle-Avoiding Router

3.1	Introduction.....	25
3.2	Magic's Obstacle-Avoiding Router.....	25
3.3	Obstacles and Hazards.....	27
3.4	Hierarchical Terminals	31
3.5	Netlists.....	32
3.6	Automatic Grid Alignment.....	33
3.7	A Fast Channel Decomposer.....	34
3.8	Magellan: An Obstacle-Avoiding Global Router.....	34
3.9	Detour: An Obstacle-Avoiding Channel Router	35
3.10	Visual Feedback	35
3.11	Technology Independence	35

CHAPTER 4. Channel Decomposition

4.1	Introduction.....	37
4.2	Background.....	38
4.2.1	VTI's Maximal Horizontal Strips.....	39
4.2.2	BBL's Bottlenecks.....	39
4.2.3	The PI System	41
4.3	Channel Decomposition Algorithm	41
4.4	Implementation.....	44
4.4.1	Identifying Convex Corners.....	45
4.4.2	Horizontal and Vertical Distances	48
4.4.3	Splitting and Merging Tiles	51
4.4.4	Setting Horizontal Status Flags.....	54
4.4.5	Cutting Corners.....	55
4.5	Analysis.....	55

CHAPTER 5. Magellan--Magic's Obstacle-Avoiding Global Router

5.1	Introduction.....	58
5.2	Global Router Overview	59
5.3	Shortest Path.....	61

5.4 Crossing Placement	64
5.5 The Penalty Function	67
5.5.1 Obstacles	68
5.5.2 Jog Avoidance	70
5.5.3 Pin Density and Corners	72
5.6 Net Ordering	73
5.7 Multi-Pin Nets	74
5.8 Equivalent Terminals and Feed-Throughs	75
5.9 Loop Prevention	76
5.10 Extensions	76
5.11 Evaluation	77
5.12 Summary	79
 CHAPTER 6. Detour--Magic's Obstacle-Avoiding Switchbox Router	
6.1 Introduction	81
6.2 The Column Sweep Approach	82
6.3 Obstacle-Avoidance Strategy	85
6.4 Switchbox Strategy	86
6.5 Hazards	88
6.5.1 Hazards for Obstacle-Avoidance	88
6.5.2 Hazards for End Connections	89
6.6 Revised Vertical Wiring Rules	90
6.7 Metal Maximization	92
6.8 Channel Rotation	93
6.9 Parameters	93
6.10 Results	94
 CHAPTER 7. Results	
7.1 Introduction	102
7.2 Hand-Routing Measurements	102
7.2.1 Net Length	105
7.2.2 Electrical Characteristics	107
7.2.3 Effects on Automatic Routing	108
7.3 Time To Reroute	111
7.4 Run Time Distribution	112
7.5 Small Examples	113
7.6 Code Size	114
7.7 Improvements	115
7.7.1 Technology Issues	116
7.7.2 Global Router	116
7.7.3 Switchbox Routing	117
7.7.4 Obstacle Locations and Sizes	117
7.7.5 Grid Alignment	118
7.7.6 User-Interface	118
7.7.7 Memory Usage	118
 CHAPTER 8. Summary and Conclusions	
8.1 Introduction	119
8.2 Summary	119
8.8 Conclusions	121
 REFERENCES	122

CHAPTER 1

Introduction

Integrated circuits are specified by layers of geometric patterns etched on silicon wafers (Figure 1.1). Components of these circuits are *cells*: functional blocks such as arithmetic logic units, registers, pads, and shifters. After the constituent cells of an integrated circuit are designed, they are arranged on a 2-d surface and connected by *wires*, which are also patterned onto silicon. Typically two or more layers of wires provide cells with power and ground, clock lines, control, and other signals. These wires run between connection points called *terminals* at the edges of cells.

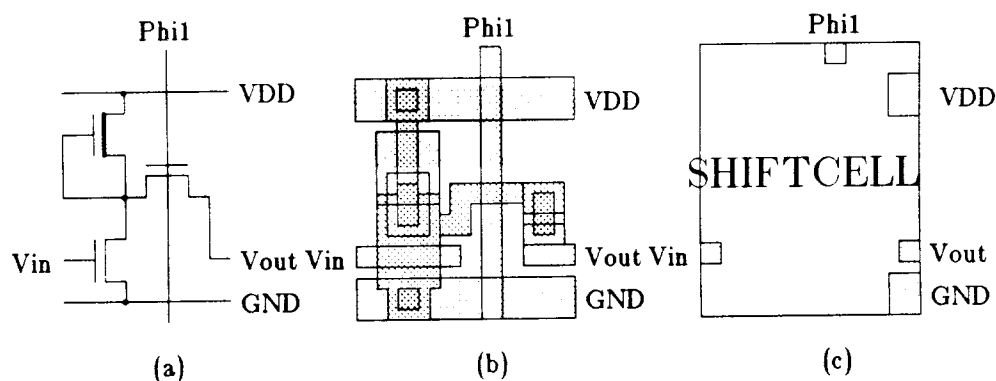


Figure 1.1. The circuit in (a) is implemented in an integrated circuit by the geometric patterns in (b). The geometry is encapsulated for routing as a subcell in (c), where terminals such as V_{out} are the only features visible.

The process of placing these wires to connect the cells together, called *routing*, is the topic of this dissertation. While the process of determining the locations of cells, called *placement*, has great influence on the subsequent wiring between these cells, it alone is a complicated task worthy of a dissertation. As such, I will ignore the placement problem and concentrate on the wiring together of pre-placed cells with fixed locations.

Integrated circuit routing is a difficult task. Current designs may have hundreds or thousands of *nets*, each containing two or more *terminals* to be wired together (Figure

1.2). The wires for each routed net must observe *design rules* specifying, for each wiring layer, the minimum width of the wires and minimum separation from other material necessary to ensure reliable fabrication. These design rules constrain the number and locations of wires that may be placed in a given area of the design. At the same time, since larger chips are more difficult to fabricate successfully, it is important to use area efficiently and place as many wires as possible in a given area.

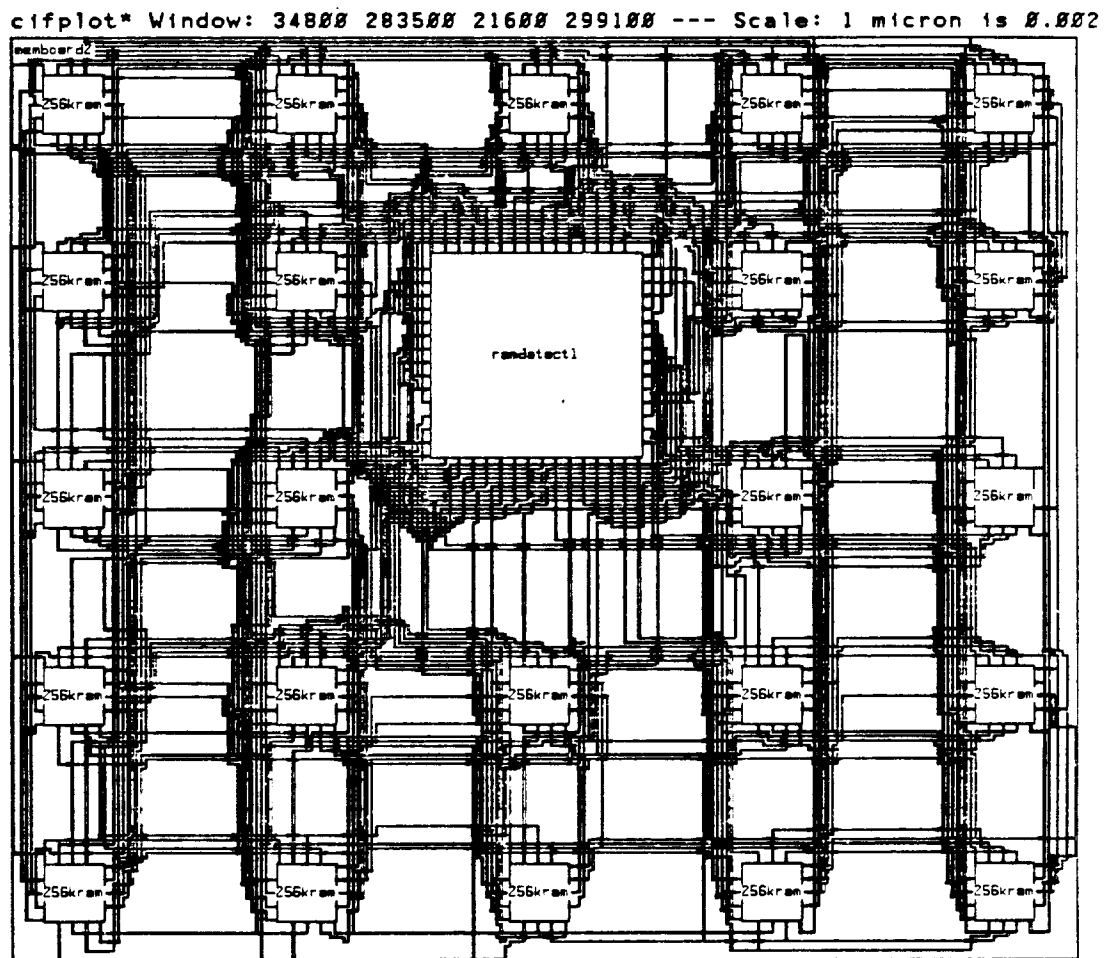


Figure 1.2 Routing places wires to connect subcells together. This example has 425 terminals and 60 nets.

The different electrical characteristics of the different routing layers are a further complication. Special care must be taken in selecting the routing layers used to connect

nets that are particularly critical in terms of their resistance, capacitance, or the propagation delay of the resulting wiring. These special nets may need to be routed on the routing layer with the best electrical properties, given priority over other nets, or connected using wires with larger than minimum widths.

One particularly important aspect that makes routing difficult is the interaction between nets. Since there is a limited number of routing layers (typically 2 or 3), once a net is placed it blocks some of the possible paths for the remaining nets to be routed; thus, a seemingly innocuous action during the routing of one net may have detrimental consequences for other nets.

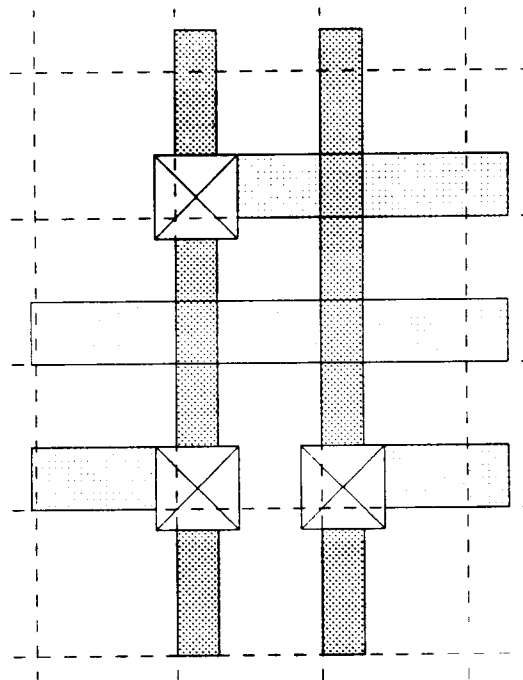


Figure 1.3. A wiring model for automatic routing places all wires on a uniform grid using one wiring layer in each direction. The grid spacing is determined by the design rule for minimum contact-to-contact spacing. This simplifies the routing process but can result in wasted area in cases where contacts are not adjacent.

The difficulty of the problem makes hand-routing very time-consuming. It can easily take months for designers to hand-route even the relatively simple Mead-Conway style custom designs produced in university environments, wherein many of the connections are

made by abutment. The chief designer for UC Berkeley's SOAR (Smalltalk on a RISC) microprocessor project estimates that hand-routing comprised 25 percent of the 12.5 person-months needed to complete the layout [Pendleton].

Because routing is so difficult and time-consuming, automated routers have been developed to manage the complexity and reduce design times. Automatic routers generally take a simplified approach, sacrificing the flexibility of hand-routing. One common wiring model places routing on a fixed grid whose spacing is determined by the closest possible spacing between adjacent contacts. To conserve area and maintain the uniformity of the routing grid, all wires are of the same width. To avoid layer assignment problems where nets cross over each other, one routing layer is used for all horizontal wires and another layer is used for all vertical wires (Figure 1.3).

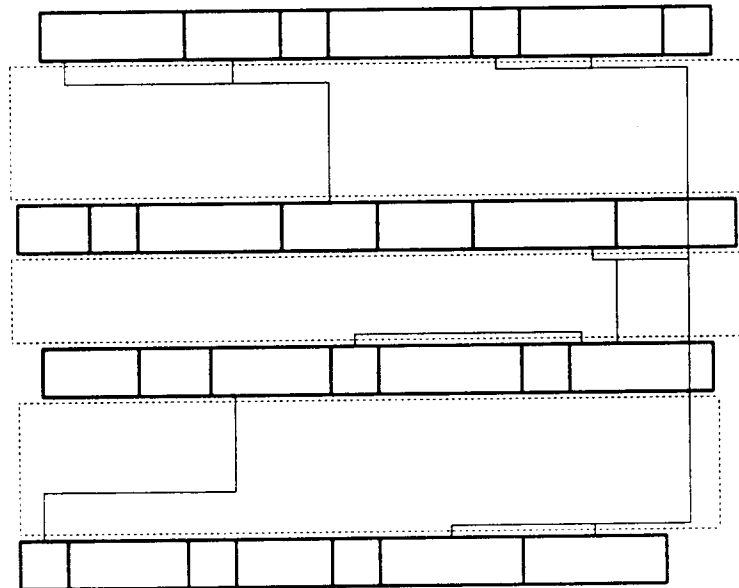


Figure 1.4. Standard-cell and gate-array designs consist of long parallel rows of subcells of uniform height. The dotted lines show the channels between the rows of subcells.

The simplifications taken by automatic routers extend to subcell placement. In the standard-cell and gate-array design styles chips consist of long parallel rows of subcells of uniform height, with signal routing between terminals on the edges of adjacent rows (Figure 1.4). Routing such designs is accomplished by routing each of the rectangular areas

between the rows of subcells. These rectangular regions with terminals on only their top and bottom edges are called *channels*; specialized *channel routers* are employed to route these regions.

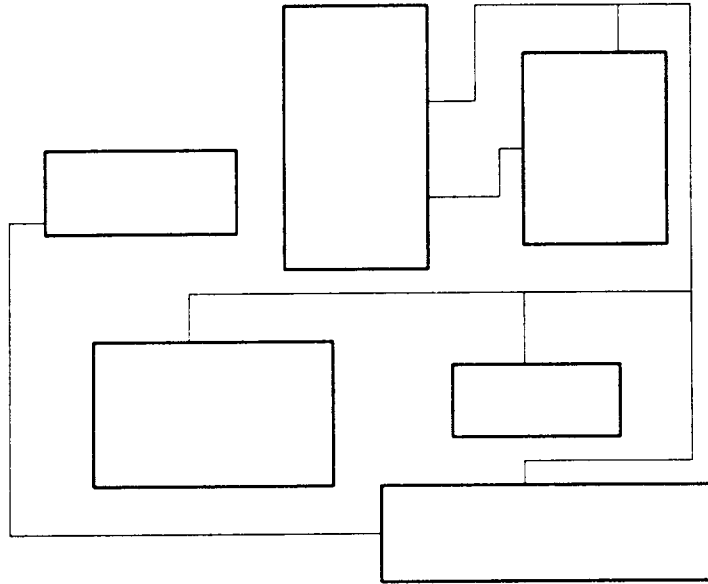


Figure 1.5. Custom chips have cells with arbitrary dimensions and arbitrary locations.

While most existing routing systems are designed for standard-cell or gate-array integrated circuits, high-performance custom designs present new and more difficult problems for automatic routers. In custom designs chips are less structured: cells have arbitrary dimensions, and placement is arbitrary rather than in rows (Figure 1.5).

The custom design style gives designers the freedom to build compact logic blocks of greater complexity and arrange them more efficiently, but the resulting layouts are more complex than for standard-cell designs. For example, connections are no longer limited to adjacent rows. Instead, the automatic router may have a large number of different choices for a path to connect terminals on one cell to terminals on another cell. In standard-cell designs, power and ground are distributed to the cells in a row through the cells themselves, by abutment; in this case power and ground routing consists of making simple connections to the ends of the rows. Custom designs require routers to make

power and ground connections to less-structured arrangements of cells, using wires whose widths vary according to the amount of current they must carry.

Custom designs frequently push the limits on chip size and performance. Consequently, high-quality routing is crucial. More aggressive performance goals require that the router give special treatment to the routing of special nets such as clock lines and critical paths, with respect to delay, layer selection, and resistance. The space constraints on custom designs require the flexibility to make trade-offs between wire length and area. For example, it may be possible to reduce chip sizes by routing non-critical nets by circuitous paths.

Traditional channel routing tools, intended for standard-cell and gate-array designs, are inadequate to meet the different needs of custom chips. They afford designers little or no control over special nets such as clock lines or critical paths. Instead, these nets are treated no differently from the larger number of non-critical nets. Traditional channel routers lack the flexibility to make tradeoffs between wire length and area, and have rigid restrictions upon which routing layer runs in each direction.

The heart of the problem is the inability of traditional channel routers to simply and efficiently incorporate the input of human designers into the routing process. Designers are more flexible and intelligent than the best automatic routers; however, hand-routing is slow and error-prone compared to automatic routing. Additionally, the human designer's expertise is generally needed for only a small number of the connections to be routed.

One way to efficiently incorporate designer input is to let a human designer wire the critical nets and let an automatic router route the larger number of non-critical nets. Combining designer input with automatically-generated routing allows designers to focus on the cases requiring special attention; the automatic router quickly routes the other nets that do not require human expertise. There are two approaches to combining designer input with automatic routing: *prewiring*, in which special nets are hand-routed before automatic routing, and *patching*, in which special nets are hand-routed after automatic routing.

Unfortunately, traditional channel routers do not allow prewiring. The use of such routers is typically an all-or-nothing proposition: they require that the routing areas be completely free of previously-placed layout. Designers must choose between routing everything automatically or routing everything manually. With such systems the only control the designer has over the routing is the connectivity. The other desirable features such as electrical properties and area/speed tradeoffs are completely dictated by the routing system.

Short of simply accepting the automatically-generated wiring, the remaining alternative to prewiring is postwiring: hand-patching the results of automatic routing to get the desired wiring. Hand-patching critical nets after channel routing is a poor alternative to prewiring, since the existing automatic routing interferes with hand-routing. Even if there is room to do the hand-patching, it is likely to be difficult. Furthermore, with traditional channel routers the hand-patching must be completely redone if the chip is rerouted.

My thesis is that it is important to build design tools that allow hand-generated input to be easily combined with automatically-generated results. To support custom designs and provide control and flexibility, automatic routers should allow designers to hand-route critical nets in advance and then allow this hand-routing to be efficiently merged with subsequent automatic routing. I call this previously-placed layout *obstacles*, and routing around the obstacles *obstacle avoidance*.

Work in other areas of Computer Science illustrates the general benefit of the ability to combine hand-generated input with automatically-generated results. In the area of programming languages some compilers allow assembly language to be combined in-line with compiled code. In place of (or to supplement) complex and sophisticated code generators, humans have the option to hand-code critical sections of programs. Short of artificial intelligence techniques on a par with the human designer, systems that combine hand-generated input and automatic input are faster, more flexible and more effective than would otherwise be possible. In such systems the designer does the critical work demanding human expertise, while the remaining non-critical work is left to the design tool. The design tool truly is a tool rather than an adversary to be outguessed or worked-around.

This dissertation presents the design of a routing system designed specifically to allow hand-generated input to be easily combined with automatically-generated results. The router, developed as part of the Magic VLSI layout editor [Ousterhout 85], works as an *obstacle-avoiding* system in which previously-placed layout such as hand-routing forms obstacles that the router knows how to route across and around. This allows designers to deal specially with nets that have special requirements, while providing a fast way to make the remaining noncritical connections.

Chapter 2 surveys previous routing systems. It explains how the early net-at-a-time routers are inadequate for routing large circuits, and how more powerful channel routers are an answer to these problems.

Chapter 3 presents an overview of Magic's routing system. It introduces the major results of this work, outlining techniques for combining automatic routing with hand-routing. The most fundamental of these results is the idea of a preferred direction for crossing an obstacle in a routing channel. From this it develops the idea of hazards, which guide Magic's global router as well as its channel router.

Chapters 4 through 6 present and analyze the three major parts of Magic's router. Chapter 4 presents Magic's channel decomposer, which uses a corner extension algorithm to divide the space not occupied by subcells into a small number of large rectangular channels. Chapter 5 presents Magic's global router, which incorporates crossing placement to choose specific channel-to-channel crossing points for each net. Chapter 6 describes Magic's channel router, which routes switchboxes and avoids obstacles while still producing results comparable to the best traditional channel routers.

Chapter 7 evaluates Magic's routing system. It presents results from routing a number of test chips as well as some real integrated circuits.

CHAPTER 2

Approaches to the Routing Problem

2.1. Introduction

Routing at the chip level involves much more than finding an optimal (shortest path) routing for each net. One reason is interactions between nets: an optimal routing for one net may block an optimal route for another, or even make it impossible to make the other connection. Obstacles present a further complication similar to net interactions: obstacle-avoiding routers must consider both the interactions with obstacles and interactions with other nets to avoid blocking connections for other nets. To find an optimal routing--one that minimizes chip area, minimizes wire length, or maximizes overall performance--requires a router to take a more global view, considering these interactions and resolving any conflicts.

The routing problem has been shown to be NP-complete for even a single multi-pin net [Garey and Johnson]. This means that routing is equivalent to any one of a large number of other problems for which no exact and efficient algorithm is known. Thus, research in automatic routing concentrates on less ambitious goals such as using heuristics to generate acceptable routing, or optimally solving special cases of the general problem.

This chapter surveys two general approaches to the problem of integrated circuit routing: net-at-a-time routing and channel routing. The approaches are distinguished by two major characteristics: the manner in which each approach decomposes the routing problem, and the manner in which each uses global information to guide the routing process. Net-at-a-time routers treat each net as a separate routing subproblem, while channel routers use a three-step problem decomposition. Net-at-a-time routers route each net without regard for other nets, while channel routers route many nets in parallel and consider interactions between nets.

Another important consideration in analyzing approaches to routing is their tradeoff in run time versus effectiveness. In routing chips for high volume production, die size dictates yield and thus economic viability. In this domain, designers are willing to spend large amounts of time to get marginally better routing. For lower-volume parts, die size may be less critical than design time. In this case the efficiency of the routing algorithm may be more important than the resulting die size. One particularly interesting example of this is wafer scale integration where, due to fabrication defects on the wafer surface, each wafer may require its own unique routing. A router that took days of CPU time to route each wafer would be too slow for such an application.

One final important criterion is flexibility--the ability to control the resulting routing. Designers often need to give special treatment to special nets such as critical paths, power and ground, and clock lines. Most routers do not provide for designer control over these special nets. One way for a router to provide this control is to allow automatic routing to be incorporated with previously-placed hand-routing. Net-at-a-time routers allow this; however, most channel routers prohibit partial hand-routing.

2.2. Net-At-A-Time Routers

The first category of routers is net-at-a-time routing, represented by Lee's wavefront router [Lee] and Hightower's line expansion [Hightower 69] router. With net-at-a-time routers a point-to-point router is used to make each of the many connections in a design, one at a time. The overall routing problem is decomposed into a number of subproblems, each of which consists of a single net. Each subproblem is routed without any consideration for the remaining nets to be routed. This is the simplest approach, and it uses the least global information to resolve net interaction conflicts.

Net-at-a-time routers inherently allow automatic routing to be combined with previously-placed hand-routing. Since they operate sequentially, the first nets routed become obstacles for subsequent nets to route around. A designer using such a router may wire special nets by hand; the router routes around these hand-routed nets just as it routes around other automatically-generated routing.

While net-at-a-time routers do an excellent job of routing individual nets, they do a poor job on the overall routing problem. This occurs because they route the current net without regard for the remaining nets to be routed. Since net-at-a-time routers do not consider net interactions, they may place a net in such a manner that subsequent nets will be blocked and cannot be completely routed.

2.2.1. The Lee Router

The Lee router was an early and significant approach to the routing problem. Operating on a grid superimposed over the routing area, the Lee router finds a shortest rectilinear path from one point to another by propagating a wavefront from a source point toward a destination point. At the start the wavefront consists only of the source point for the route (Figure 2.1a). At each iteration the wavefront expands to include all currently unvisited grid points immediately adjacent to some current portion of the wavefront (Figure 2.1b, c). For each point the router records the direction from which the given point was reached. The router terminates when the wavefront reaches the destination point, or when the search space is exhausted. If the route was successful the recorded direction information is traced backwards to the source to determine the exact path generated. (Figure 2.2).

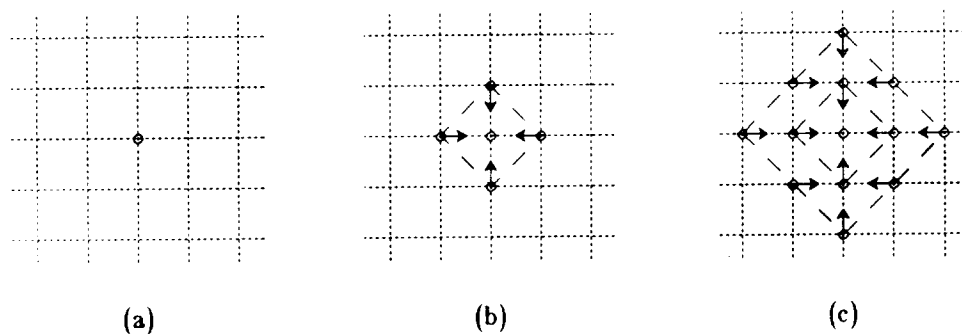


Figure 2.1. The Lee router finds a shortest path from one point to another on a rectilinear grid. On each iteration it expands a wavefront to include all currently unvisited points immediately adjacent to some current portion of the wavefront. At each point visited the router remembers the direction from which the point was first reached.

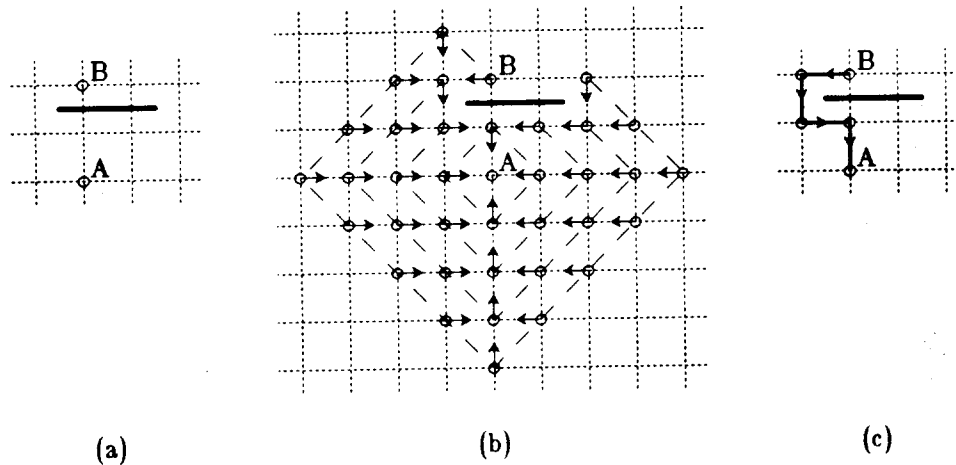


Figure 2.2. Figure (a) shows the start of a search from point A to point B with an obstacle (drawn in bold). In (b) the wavefront has propagated outward and intersected the destination point B. In (c) the shortest path information is obtained from the recorded direction information at each grid point.

The Lee router is effective. By considering layer switches in conjunction with the propagation at each wavefront expansion, the router will eventually generate all possible paths to every point reachable from the source point. Thus, the Lee router is guaranteed to find a solution if one exists. Since the wavefront expands outward uniformly from the source, the shortest path to the destination is guaranteed to be the first one found.

The Lee router is also flexible. By slightly modifying the algorithm the Lee router can obtain a number of special routing effects, such as crossing a minimal number of other nets or making tradeoffs between additional net length and routing on an unpreferred layer. This is accomplished by associating a penalty with each undesirable feature in a routing and remembering, for each point on the routing grid, the cost to reach this point from the source point. As the wavefront propagates it may generate multiple paths to any given point, each with a different cost. Special effects with their resulting flexibility are then achieved by remembering the current lowest cost to reach a point and the direction from which the lowest cost path to a point was reached.

Unfortunately, the Lee router has large memory requirements and slow run times for long connections. It requires a large amount of memory because it explicitly represents

each grid point in an array. The Lee router is slow because it must keep track of all of the points on the current wavefront, and at each point consider each of the possible alternative actions: switch layers, turn left, turn right, or run straight ahead. Also, since it propagates a wavefront in all directions, it must examine $O(N^2)$ grid points to find a path between two points N units apart.

2.2.2. The Hightower Router

Hightower's router also routes from point to point, one net at a time. It works by line expansion, expanding perpendicular escape lines from both end points of the route and then repeatedly extending the longest escape line from some previously-constructed line (Figure 2.3a). Escape lines are bounded by obstacles such as previously-placed wiring. The router terminates when a line from one end point intersects a line from the other point (Figure 2.3b).

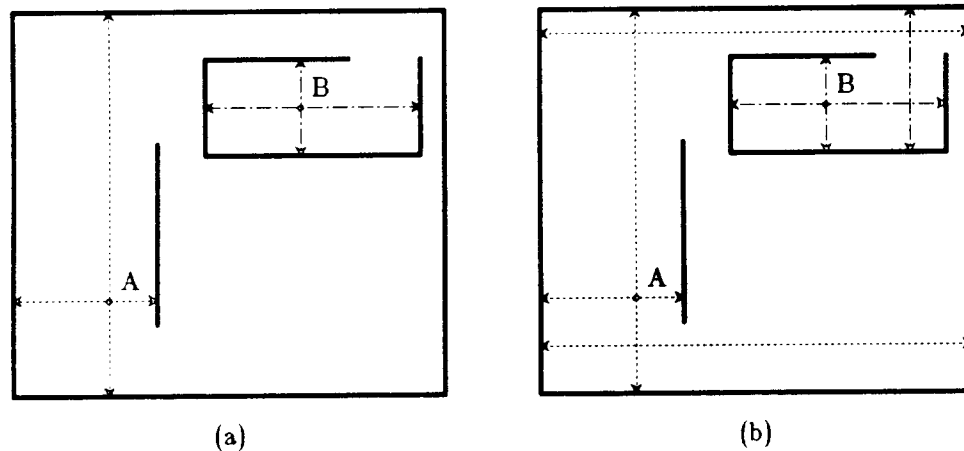


Figure 2.3. The Hightower router repeatedly expands perpendicular escape lines from both points (A and B) to be connected. The router terminates successfully (Figure b) when escape lines from the two end points intersect.

The Hightower router is fast for simple mazes. By extending perpendicular escape lines it takes advantage of the dimensions and locations of the features in the routing area. If objects are nearby, the router expands short distances much like the Lee router; however, where features are spaced far apart the router can extend large distances in a

single iteration. This ability to span large distances in a single iteration allows the Hightower router to avoid much of the work done by the Lee router.

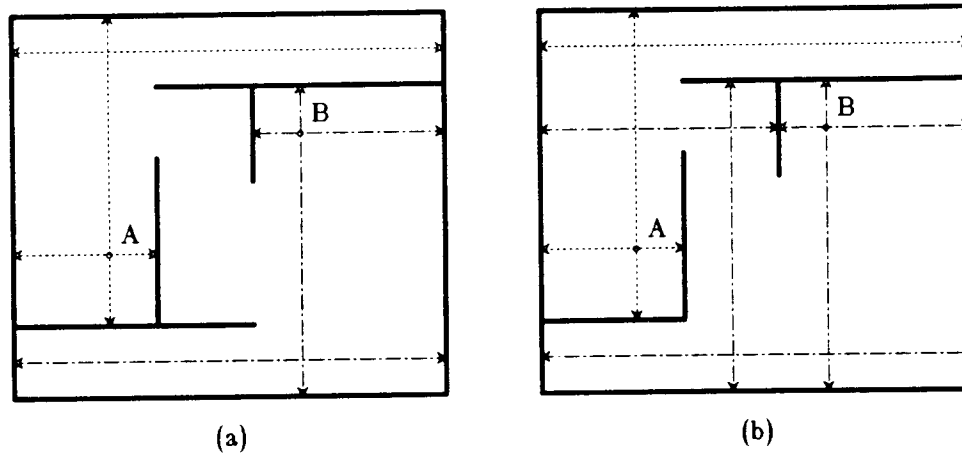


Figure 2.4. The Hightower router can not guarantee to find the best routing, or even to find a routing if one exists. It fails to find a solution in (a), where extending the longest perpendicular escape lines does not find the path from point A to point B. In another example (b), it fails to find the shortest path from point A to point B.

The comparison between the Hightower router and the Lee router shows a clear tradeoff between results and run time. Unlike the Lee router the Hightower router does not guarantee to find a path if it exists, although Hightower claims that in practice this is never observed (Figure 2.4). Also, the Hightower router does not guarantee to find the shortest path; instead, it finds the path with the fewest number of bends. This tends to be the shortest path; the distance penalty measured less than 10 percent for one set of examples [Hightower 74]. On the other hand, for one example the Hightower router was 2 to 3 orders of magnitude faster than the Lee router.

Memory requirements are much less than for the Lee algorithm. Rather than storing the entire plane as an array, the Hightower router stores only lines and points. For paths with few bends in them the memory requirements are correspondingly small.

Although the Hightower router is fine for routing individual nets, it suffers from the same short-sightedness as the Lee router. Since it routes a net without regard for subsequent nets, it may place routing that blocks these later connections.

2.3. Channel-Based Methods

The solution to the short-sightedness of net-at-a-time routers is to employ routing methods that consider many nets at once and make tradeoffs that allow all of them to be routed successfully. The channel routing technique, originated by Hashimoto and Stevens [Hashimoto and Stevens] is one such approach.

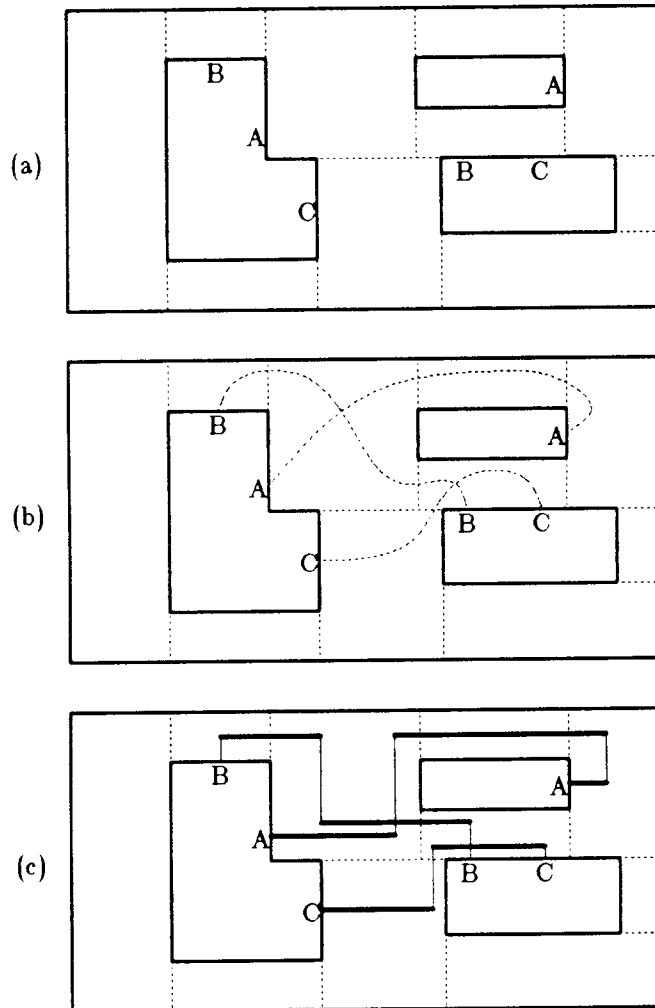


Figure 2.5. Channel routing techniques consider many nets at once. Channel decomposition (a) divides the area between subcells into rectangular *channels*. Global routing (b) determines which channels each net crosses. Channel routing (c) individually routes each channel, placing wires to make the connections specified in the global routing phase.

This approach decomposes the routing problem into three phases (Figure 2.5). The first phase, *channel decomposition*, divides empty areas between cells into rectangular, non-overlapping *channels*. The second phase, *global routing*, uses global information about the overall routing problem to determine which channels each net will cross to make its connections, without actually placing the wires to do so. The final phase, *channel routing*, separately wires each channel to make the connections specified by the global router. Channel-based approaches divide the difficult overall routing problem into a number of difficult but smaller problems: global routing and individual channel routing problems. Most current routing systems adopt this strategy, with minor variations.

Channel decomposition lays the groundwork for the subsequent steps by defining the dimensions of each of the channels. In general the remainder of the channel routing process runs more smoothly if channel decomposition generates a small number of large channels, as these are more easily used by both the global router and the channel router.

The global routing phase is an important planning step prior to channel routing. It determines the general path for each net without actually placing the wires to make the connections. The global router has the responsibility for generating channel routing problems that the channel router can successfully wire. Simultaneously, the global router must try to choose direct paths for nets, to minimize overall wire length. Global routers use global information about nets to resolve conflicts, handling overcrowding in channels by considering alternative paths through less-crowded channels. Global routing may process nets sequentially, or it may route many nets in parallel using more sophisticated techniques such as simulated annealing [Vecchi and Kirkpatrick] and hierarchical routing [Burstein and Youssef]; however, these techniques are generally much slower than net-at-a-time methods.

The channel routing phase separately routes each rectangular routing area as defined during channel decomposition, to make the connections specified during global routing. Routing areas with connections on two opposite sides may be routed using specialized *channel routers*. Routing areas with connections on additional sides are more difficult to route, requiring more general *switchbox routers*. Only some of the various approaches

handle the switchbox routing problem; nevertheless, throughout this dissertation I sometimes use the term *channel* in a general sense to denote the routing of any rectangular region.

One important characteristic of channel routers, compared to net-at-a-time routers, is that channel routers generally use a simplified wiring model. Many routers place all horizontal wiring on one routing layer, and all vertical wiring on the other; this constrains the routing and sometimes prevents channel routers from completing all connections. Some channel routers place all routing on a fixed routing grid that allows a contact to be placed at any grid intersection; this wastes space if two adjacent grid lines have no adjacent contacts (Figure 1.3).

2.3.1. The Left Edge Algorithm

Hashimoto and Stevens' routing system uses the classic *Left Edge* algorithm to individually route each channel routing problem generated by the global router. Scanning from left to right across a channel, the algorithm assigns each net to a single horizontal *track*, always selecting the net whose left edge comes closest to the right edge of some previously-placed net without overlapping that net (Figure 2.6). The process repeats until all connections are made. This algorithm is greedy, since it tries to make the most effective use of each track by packing as many nets as possible into each horizontal track within a channel. This reduces the total area needed to do the routing. In fact, the results are optimal if there is at most one terminal in each column of the channel, as in Figure 2.6.

This channel routing algorithm uses global information on a channel-by-channel basis. Rather than dealing with one net at a time, the Left Edge algorithm examines all nets in the channel to find one which makes the best use of the remaining wiring space within the current track. The results produced are superior to those for net-at-a-time channel routers.

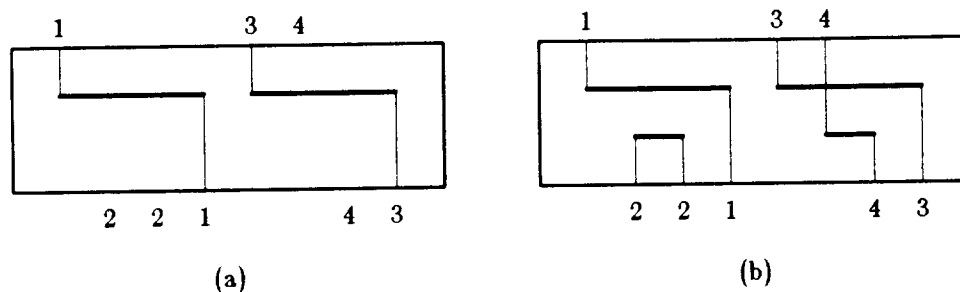


Figure 2.6. The Left Edge algorithm assigns nets to tracks, filling each track as fully as possible before proceeding to the next track. Since net 1 has the leftmost terminal, it is placed in the first track (a). Net 3 also goes into this track, since it extends further to the left than net 4. The remaining nets fit into the second track (b).

Unfortunately, the Left Edge algorithm does not work in channels where terminals for different nets share the same column (Figure 2.7a). Such terminals generate vertical constraints, requiring the net for the upper terminal to be placed in a track above the track assigned to the net for the lower terminal. The vertical constraints can be cyclic, requiring that a net be placed on a track above itself (Figure 2.7b)). This can be done by splitting a net so it occupies more than one track (Figure 2.7c); however, the Left Edge algorithm always assigns each net to exactly one track. Thus, channels with cyclic constraints can not be routed using the Left Edge algorithm.

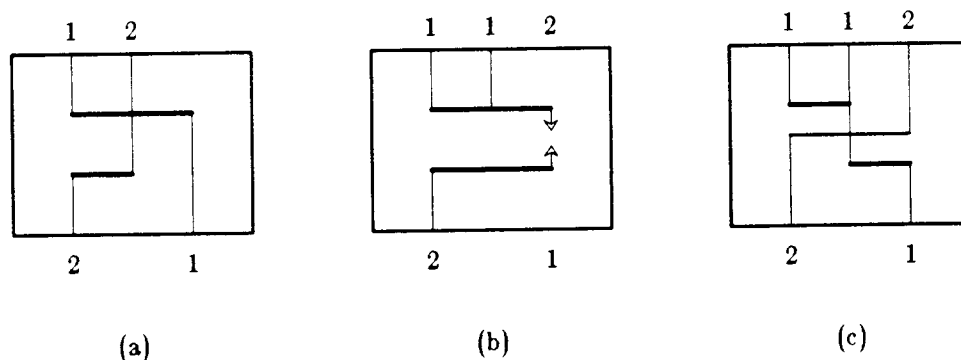


Figure 2.7. When two terminals share a column, vertical constraints require that one net be placed above the other (a). Cyclic constraints (b) may make it impossible to use the Left Edge algorithm. Such constraints can be resolved by splitting a net onto more than one track, as in (c).

2.3.2. Deutsch's Dogleg Router

Deutsch developed a router based on the Left Edge algorithm, whose goal was to handle channels containing vertical constraints. His *Dogleg* channel router [Deutsch] allows nets to switch from one track to another; locations where nets switch tracks are called *doglegs* (net 1, Figure 2.7c). Doglegs allow the router to eliminate long constraint chains and route using fewer tracks (Figure 2.8).

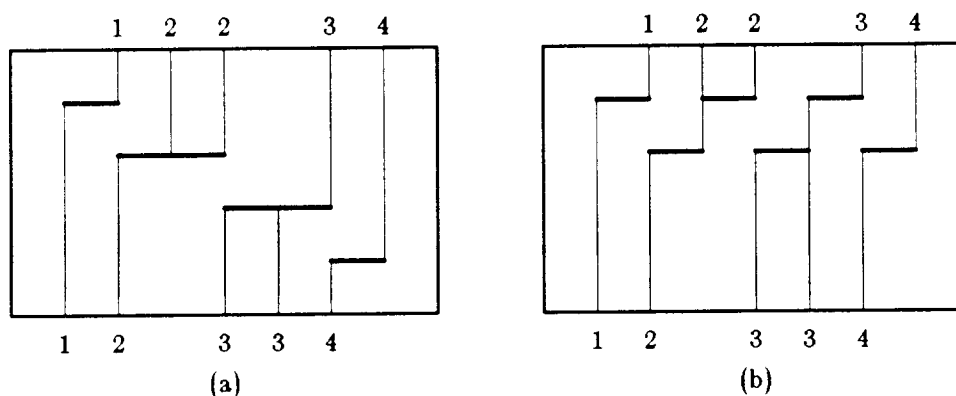


Figure 2.8. Doglegs can reduce the number of tracks needed to route a channel. The Left Edge algorithm, which does not use doglegs, produces the routing in (a). Deutsch's Dogleg router does the same routing using fewer tracks (b).

While the dogleg router can route channels where the Left Edge algorithm fails, there are cases where it too has problems. The cause of this is a restriction upon the locations for doglegs: to avoid producing unnecessary doglegs, the router allows them only at interior pin locations of multi-pin nets. Thus, the dogleg in Figure 2.9b is not allowed, since net 2 has no pin in the center column of the channel.

More fundamentally, the reason for this problem lies in the way channel routers closely based upon the Left Edge algorithm use information global to a routing channel. Although all such routers use global information to determine which net should be processed next, and some use vertical constraint information, in a sense all of these routers operate a net at a time since they do not fully consider the effect of subsequently-routed nets upon the current net. The dogleg router does not know in advance if adding a dogleg will reduce the number of tracks needed to route a channel (Figure 2.9a, b); in some cases

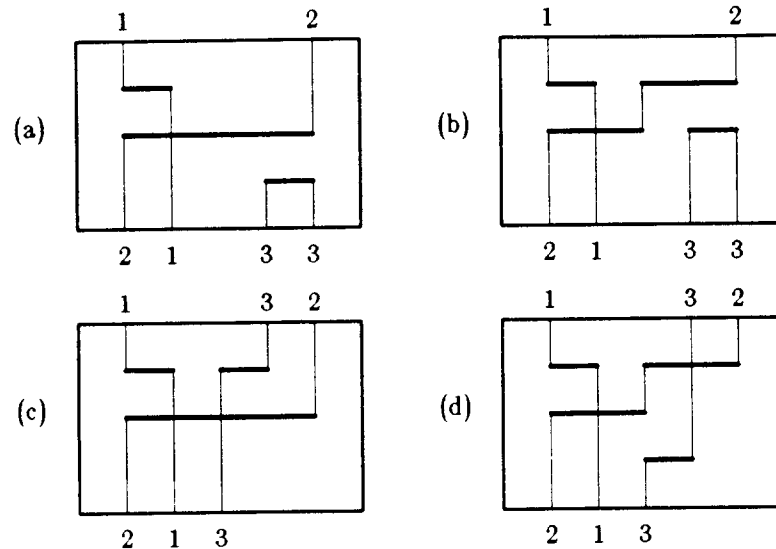


Figure 2.9. Unconstrained doglegs can have good or bad effects upon the number of tracks needed for routing. In (a, b) the dogleg in net 2 reduces the number of tracks, while in (c, d) it increases the number of tracks.

doglegs increase the number of tracks needed (Figure 2.9c, d). Consequently, the dogleg router constrains the number and locations of doglegs, with the effect that they are sometimes prohibited where they are needed.

2.3.3. Yoshimura and Kuh

Yoshimura and Kuh's algorithms [Yoshimura and Kuh] use graph reductions to merge nets onto tracks (Figure 2.10). Their algorithm #1 simply merges nodes on a vertical constraint graph, where each node represents a net, to minimize the length of the longest path in the graph. Their algorithm #2 takes a more complicated approach, to try to maximize the number of merges that can be made. When the algorithm concludes, all of the nets in a merged node are assigned to the same track; thus, merging nodes reduces the number of tracks needed to route a channel.

These algorithms share the Dogleg router's problem with regard to dogleg locations. Since doglegs are allowed only at terminal positions, they are sometimes prohibited even when they would benefit the routing.

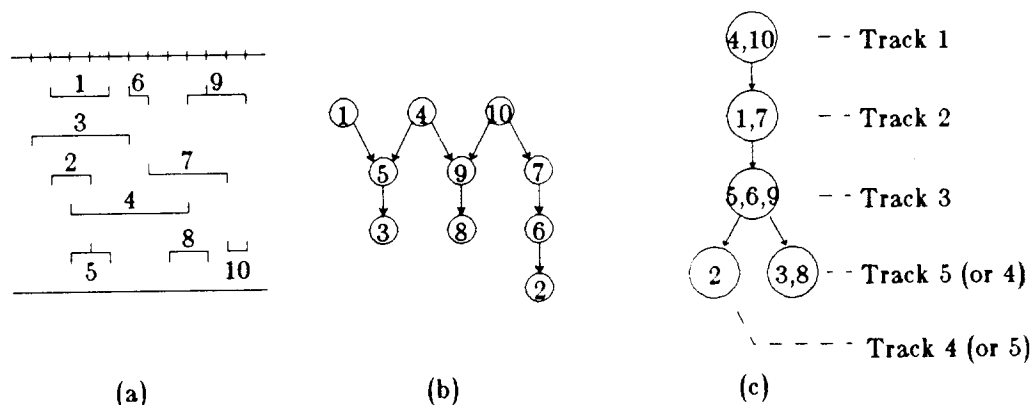


Figure 2.10. Yoshimura and Kuh's algorithms use graph reductions to merge nets onto tracks. Figure (a) shows a netlist. Figure (b) shows the corresponding vertical constraint graph. Figure (c) shows the final merged graph.

2.3.4. YACR-II

YACR-II [Sangiovanni-Vincentelli] is Yet Another Channel Router. YACR-II assigns nets to tracks and then copes with each vertical constraint by attempting to maze route around the vertically-constrained area, using some adjacent tracks and columns (Figure 2.11). To minimize the number of vias and avoid introducing additional tracks, YACR-II relaxes the layer-per-direction wiring model when resolving vertical conflicts (Figure 2.11a, c), allowing "wrong way" wiring.

YACR-II produces routing to equal the best channel routers, by making effective use of constraints and global information. It considers many nets at once, particularly the tradeoffs between various track assignments, selecting a track assignment that simplifies the overall routing for these nets. Also, unlike Left Edge algorithms, YACR-II focuses on the critical column crossed by the greatest number of nets, first assigning nets to tracks at this point.

2.3.5. Burstein's Hierarchical Router

Burstein's Hierarchical [Burstein and Pelavin] router takes the novel approach of mapping an m -row by n -column channel routing problem onto a coarse 2-by- n grid. Each net is routed through the grid with a minimum-cost Steiner tree, one net at a time. After

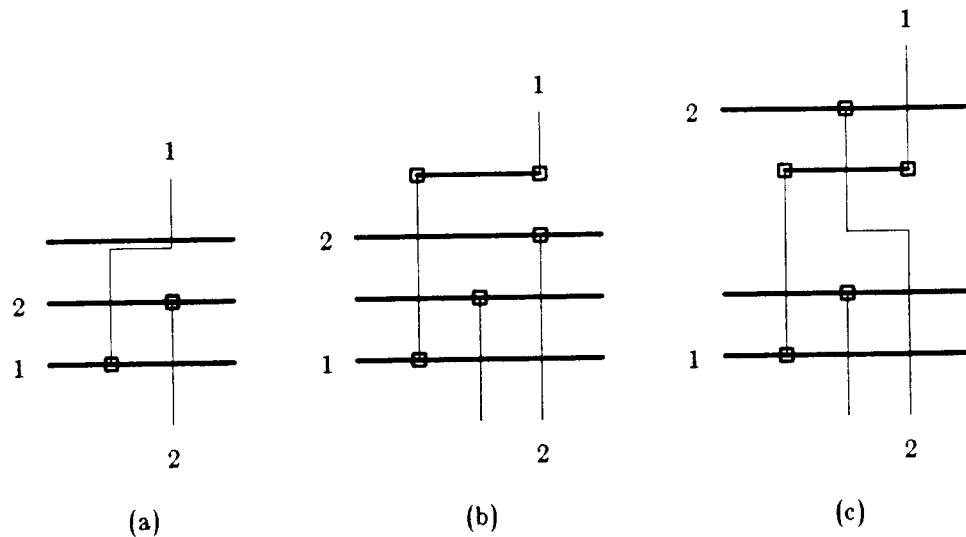


Figure 2.11. YACR-II uses maze routing to resolve vertical conflicts between nets 1 and 2. First, it tries to route using an adjacent column, in poly only (a). If this fails, it tries doglegs spanning more columns (b). If this fails, then both interconnections are jogged (c).

all nets are routed, each net is removed from the grid in random order and rerouted while the other nets remain in place. Following this the grid is refined by splitting it into 2 2-by-n subproblems, which are routed in a similar fashion. The successive refinement continues to the point where each point in the grid corresponds to a single point on the original routing grid (Figure 2.12).

Rerouting each net in random order allows it to adjust its path to accommodate the presence of the other nets. This gives the algorithm a degree of independence from the order in which nets are routed. It also is an example of using global information within the routing channel to choose routings that consider interactions between nets.

2.3.6. Rivest's Greedy Router

Since the Greedy router forms the basis of Magic's own channel router, a detailed description appears in Chapter 6. Consequently, this brief discussion omits many of the details, concentrating instead on higher-level issues such as its use of global information.

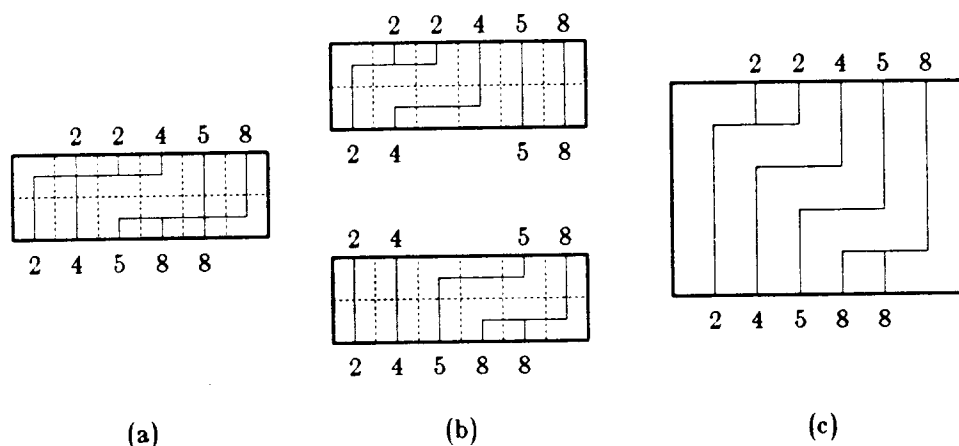


Figure 2.12. Burstein's Hierarchical router decomposes an m -by- n channel into a 2-by- n problem (a) and successively refines the grid (b) to obtain the final routing (c).

The Greedy router uses a column sweep approach, extending partially-wired nets from left to right across a channel. In each column heuristic rules control the placement of vertical wiring to make connections and simplify the remaining routing task.

The Greedy router is similar to YACR-II in the sense that it scans the length of a channel and is concerned with net assignments to tracks. Yet while YACR-II attempts to assign each net to some best single track, the Greedy router dynamically changes track assignments with vertical wiring. This allows the Greedy router to resolve or reduce vertical conflicts over a wide range of columns but tends to introduce extra vias and extra wire length.

2.4. Lack of Flexibility

Channel routers generally produce results far superior to net-at-a-time routers, but they are less flexible. While net-at-a-time routers allow partial hand-routing, channel routers generally assume channels are completely devoid of previously-placed layout. Because channel routers consider many nets at once rather than one at a time, it is not necessary for them to consider some nets as obstacles for other nets. Thus, the ability to combine automatic routing with previously-placed hand-routing is lost.

Since traditional channel routers can not handle obstacles, routing systems generally provide only a limited form of obstacle-avoidance. Some systems, such as BBL [Chen] and LTX [Persky], prewire power and ground routing by placing it at the edges of channels and then bridge other signals straight over this wiring; however, user-supplied hand-routing is not allowed. The PI system [Rivest] provides limited support for arbitrary, hand-placed metal obstacles, but each metal segment forms its own separate channel.

Magic's routing system provides a more general form of obstacle-avoidance. Based on the channel routing approach, it develops new techniques to support obstacle-avoidance at both the global routing and channel routing phases. These new techniques are described in the following chapters.

CHAPTER 3

Magic's Obstacle-Avoiding Router

3.1. Introduction

This chapter provides a high-level description of Magic's obstacle-avoiding routing system. It begins with a general introduction to the routing system, including its goals and its approach. Following this the chapter presents a major result of this dissertation: the *hazard* mechanism which forms the basis of the obstacle-avoidance techniques employed by the routing system. Next, it briefly describes the features and major components of the routing system, in the order of their invocation or use: hierarchical terminal names, netlists, automatic grid alignment, a fast channel decomposer, an obstacle-avoiding global router, an obstacle-avoiding switchbox router that produces results comparable to good channel routers, visual feedback, and technology independence.

3.2. Magic's Obstacle-Avoiding Router

The goal of Magic's automatic routing system is to provide flexibility by allowing automatically-generated routing to be combined with hand-routing. The system provides the designer with the flexibility to focus on the important parts of a circuit, performing critical wiring by hand to get routing with exactly the desired characteristics. After the critical nets are taken care of, the automatic router provides a fast way to make the remaining, non-critical connections, completing the wiring for a chip.

The router is an integral part of an interactive layout editor. It is invoked as a command from within Magic [Scott 86]. All of the router's inputs--a set of hand-placed subcells, hand routing or other obstacles, and netlists specifying the connections for the router to make--are produced using Magic's commands for layout creation, layout manipulation, and netlist editing. The router accesses Magic's internal database to get these

inputs, and paints the resulting routing directly into the database.

In this system the designer plays an active role, iterating over the design to obtain the desired results. If the results of a routing are unsatisfactory, the designer invokes a command to rip-up the automatic routing while preserving hand routing. Next the designer modifies the cell placement and adds hand routing, using commands such as *plowing* [Scott and Ousterhout]. Following this, the designer reinvokes the automatic router, completing the routing cycle.

Since the router is specifically intended for interactive use, the quality of the routing and the speed of operation are both important considerations. The router seeks a balance between speed and results, working quickly and relying on the designer for help if it gets "stuck". It does not strive for optimal routing; the quality of the routing is crucial for only a few critical nets and in a few "tight" routing areas. It completely routes moderate-size chips in 4 to 5 minutes of cpu time on a VAX 11/780, including the time to paint the resulting wires back into Magic's internal database and refresh the design on a display device. This is fast enough for designers to work with interactively.

This interactive approach contrasts sharply with other approaches to routing that attempt to completely remove the designer from the routing process. Magic's router accepts designer input in the form of hand routing; many other systems prohibit any designer input other than connectivity information, while attempting to provide a comprehensive, fully-automated answer to the routing problem [Chen] [Persky]. Another system, PI, provides limited hand routing for power and ground while stating that its objective is to "explore the limits of what can be accomplished algorithmically". In this sense Magic's router's objective is to explore interactions, particularly the division of labor, between designers and tools. More recent expert system research [Joobbani] attempts to build the flexibility and creativity of a human designer into a router; Magic's router avoids the complexity and enormous speed penalty of that approach by instead drawing upon the human designer for human expertise.

The Magic router has much in common with the traditional, non-obstacle-avoiding channel routing systems described in Chapter 2. Like those systems, it decomposes the areas between subcells into rectangular channels, globally routes nets through these channels, and then individually routes each of the channels.

The differences lie in the features added specifically to support obstacle-avoiding routing. The global router has special obstacle-avoiding features; The global router decides where each net crosses each channel boundary; this minimizes obstacles' effects on the routing, but requires that each channel to be routed as a switchbox. it considers the effects of obstacles as it evaluates alternative paths. The switchbox router is also designed to cope with obstacles; while normally observing the layer-per-direction model, it can switch routing layers to bridge obstacles.

3.3. Obstacles and Hazards

The router's unique feature is its ability to cope with obstacles. The major results of this dissertation are thus the ideas developed to deal with obstacles during routing. After first defining some terms and introducing some problems caused by obstacles, this section presents an important mechanism called a *hazard* to deal with these problems.

An obstacle is any layout present prior to automatic routing that affects the normal course of routing. For example, the results generated by one invocation of the router may become obstacles to subsequent invocations. Magic does not route over cells; thus, cells are not obstacles, and obstacles occupy the otherwise-empty areas between cells. Finally, material does not constitute an obstacle unless it occupies layers that match or interact with one of the two routing layers. For example, if the routing layers are metal1 and polysilicon, diffusion is an obstacle since it interacts with polysilicon, but metal2 is not an obstacle since there are no design rules or formation rules constraining interactions between metal2 and either of the routing layers.

Obstacles complicate the already difficult routing problem. They disrupt the simple wiring model which assumes that the routing channels are completely empty. The simple

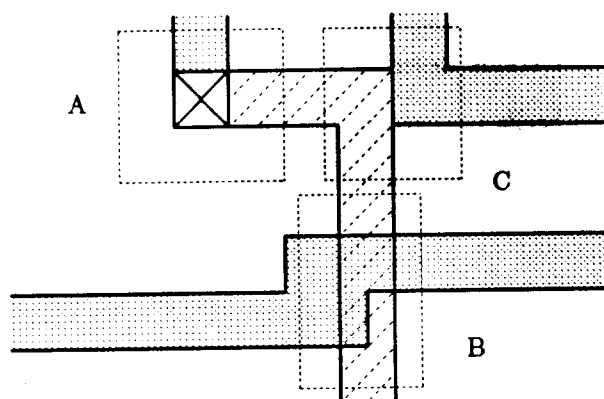


Figure 3.1. These obstacles form three blocked areas, outlined with dotted lines. Blocked areas are formed by contacts that block both routing layers (A), two-layer overlaps (B), and adjacent areas of different layers (C).

wiring model assumes it can always route in one direction on one routing layer and in the other direction on the other routing layer; this is no longer true when obstacles are present. In particular, it may be necessary for a net to switch routing layers to cross conflicting material. If routing channels are empty, design rules are an issue only at the boundaries between channels and cells; when obstacles are present in channels, design rule violations can occur anywhere within the routing area.

At their worst, obstacles occupy both routing layers in a given location, creating *blocked* areas through which no routing may pass (Figure 3.1). Blocked areas may be generated even where obstacles do not overlap each other. If obstacles on different layers are so close to each other that there is no room to place contacts between them to switch layers, then the obstacles effectively create blocked areas, since the router can switch layers to cross one of the obstacles, but can not switch layers again to bridge the other. One key problem of obstacle avoidance is recognizing these blocked areas and avoiding them by routing nets around them.

Obstacles occupying only one of the routing layers create a different problem. Areas covered by such obstacles are *obstructed*: routing may cross only on one of the two routing layers. In this case the router can simply switch routing layers, if necessary, to cross over the obstacle; however, since the combination of an obstacle on one routing layer and

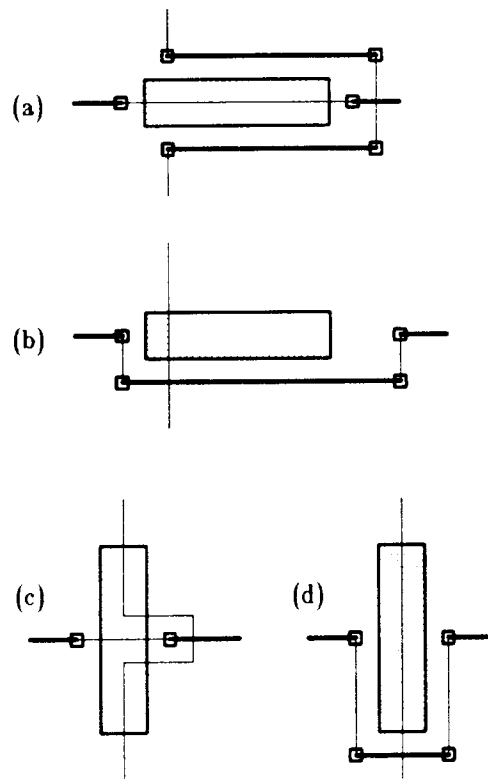


Figure 3.2. When an obstruction blocks one of the routing layers, either horizontal or vertical wires, but not both, may bridge the obstacle. If the obstacle is short and wide, as in (a) and (b), it is easier to jog the horizontal wire out of the way, as in (b). If the obstacle is tall and narrow, as in (c) and (d), it is easier to jog the vertical wire out of the way, as in (c).

a wire on the other routing layer creates a blocked area, the obstacle can be crossed either horizontally or vertically, but not in both directions. Thus the router must choose whether to cross each obstructed area horizontally or vertically.

The solution to this problem is to give preference to routing crossing the short dimension of a single-layer obstacle (Figure 3.2). Nets running perpendicular to the preferred direction (parallel to the obstacle's long dimension) are routed around the obstacle if necessary to allow other nets to cross in the preferred direction. This policy is advantageous; crossing an obstacle across its short dimension reduces interference with other nets by minimizing the size of the blocked area created in crossing the obstacle. Also, since the cross-section of the blocked area is small, nets diverted around the blocked area detour

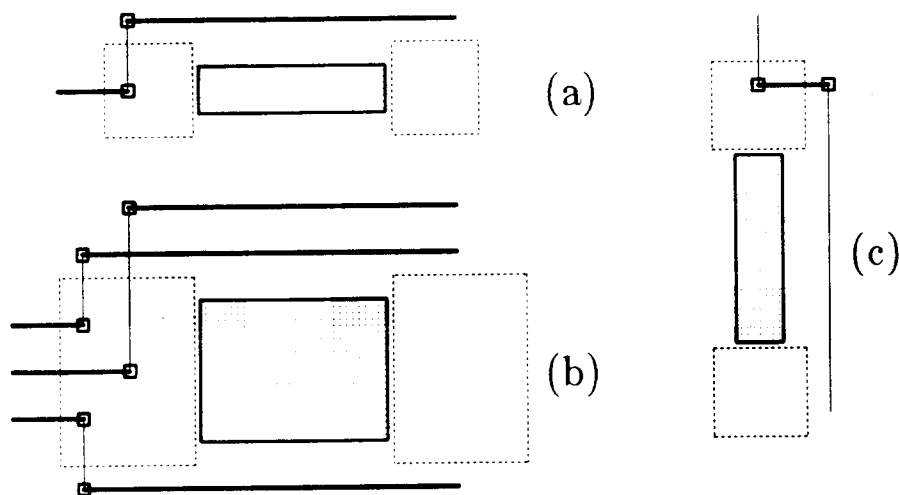


Figure 3.3. Hazards (marked with dotted lines) along one side of an obstacle are warning areas, indicating that nets approaching the obstacle from that direction should route around the obstacle. In Figures (a) and (b) nets extend from left to right toward an obstacle. Since a taller obstacle (b) may require more nets to jog out of the way, the hazard must extend further away from the obstacle to leave more room for nets to do this. In Figure (c) a hazard above an obstacle causes a descending net to jog around the obstacle.

only a short distance.

The router uses a mechanism called a *hazard* to deal with both single and double-layer obstacles. Hazards are warning areas near obstacles and are used in routing nets around blocked and obstructed areas (Figure 3.3). As the router extends a net through the routing area it may encounter a hazard; this indicates that it is approaching an obstacle that it should avoid. The router then takes appropriate actions to move the net out of the hazard area and around the obstacle.

Hazards guide the router by indicating the directions in which it should cross obstacles. The absence of a hazard adjacent to one side of an obstacle indicates that the router may cross the obstacle from that direction. Hazards completely surround blocked areas to indicate that these areas should not be crossed in any direction. Hazards adjacent to the short dimension of single layer obstacles indicate that nets should not cross these obstacles along their long dimension.

The size of a hazard has an important effect on routing quality. Moving nets away from an obstacle too soon wastes routing area, since it may leave empty spaces near the obstacle. On the other hand, moving nets away from an obstacle too late may cause some of them to run into the obstacle.

Accordingly, the router sets the size of a hazard to correspond to the urgency of avoiding the obstacle and the amount of area it estimates will be needed to jog nets around the obstacle. A hazard near a blocked area is made larger than a hazard near an obstructed area of the same size, since the router fails if it extends a net into a blocked area, while the router can simply switch routing layers to cross an obstructed area (with potential inconvenience for other nets). A hazard for a taller obstructed area is made larger than a hazard near a shorter obstructed area, since a larger number of nets may have to move around the taller obstacle, and this requires more area (Figure 3.3 (a) and (b)).

Magic generates hazards as one of the first routing steps. After channel decomposition but before global routing the router searches the channel areas and maps the location and type of each obstacle. It then analyzes the obstacles, determines the preferred direction for crossing each obstacle, and places hazards to indicate this.

3.4. Hierarchical Terminals

Terminals identify connection points at or near subcell edges. They are specified by rectangular labels, and include a textual terminal name, an area, and a layer. The router connects to a terminal at a convenient point along one of its sides; larger terminals give the router more freedom to choose connection points and to avoid generating jogs when aligning terminals onto the routing grid (Section 3.6). The terminal's layer specifies which of the layers at the terminal's location is to receive the connecting wire.

Terminal names are hierarchical, to avoid ambiguity when the same subcell appears more than once in the same design. A terminal name is composed of a sequence of cell use identifiers, followed by the name of the terminal itself. Each cell use identifier is unique

within its parent cell definition, so a hierarchical terminal name identifies which of possibly many uses of the same cell definition holds a terminal of interest (Figure 3.4). Terminals may appear arbitrarily deep within the cell hierarchy: they need not reside within the topmost subcells.

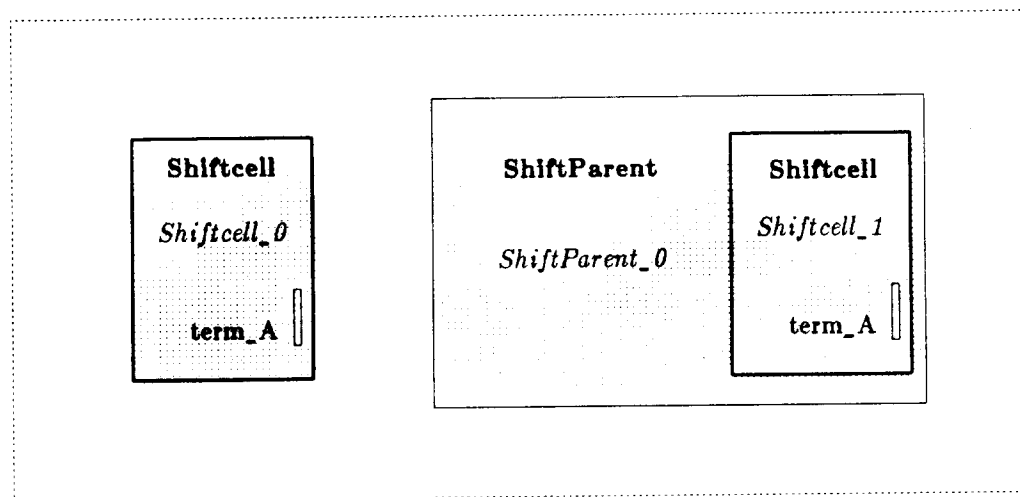


Figure 3.4. Hierarchical terminal names distinguish between terminals within copies of the same subcell. In the figure, subcell names are shown in bold, while instance names are in italics, and terminal names are in plain text. The hierarchical name of terminal *term_A* in the left subcell is "Shiftcell_0/term_A", while the name of the same terminal in the right subcell is "ShiftParent_0/Shiftcell_1/term_A".

3.5. Netlists

The designer creates netlists to specify groups of terminals to be wired together by the router. Netlists reference terminals by name rather than by location; this makes it easy to move cells, since the router finds terminals regardless of their coordinates. Magic uses this netlist information not only to specify connectivity during routing, but also to verify that all nets are fully-wired and to rip up wiring prior to rerouting.

Magic includes an interactive netlist editor that allows terminals to be selected, grouped into nets, or removed from nets. The netlist editor is graphically (rather than textually) oriented: terminals are selected and operated upon by pointing at them on the graphics screen and clicking a mouse button.

3.6. Automatic Grid Alignment

Although the router is grid-based, designers are not constrained to design cells whose cell edges or terminals match up with grid lines. The router automatically generates layout called *stems* to jog connections onto the routing grid from unaligned locations near the edges of subcells. Thus, the designer need only space terminals sufficiently far apart to ensure that it is possible to river route the terminals to grid lines; the router automatically takes care of the other grid alignment considerations.

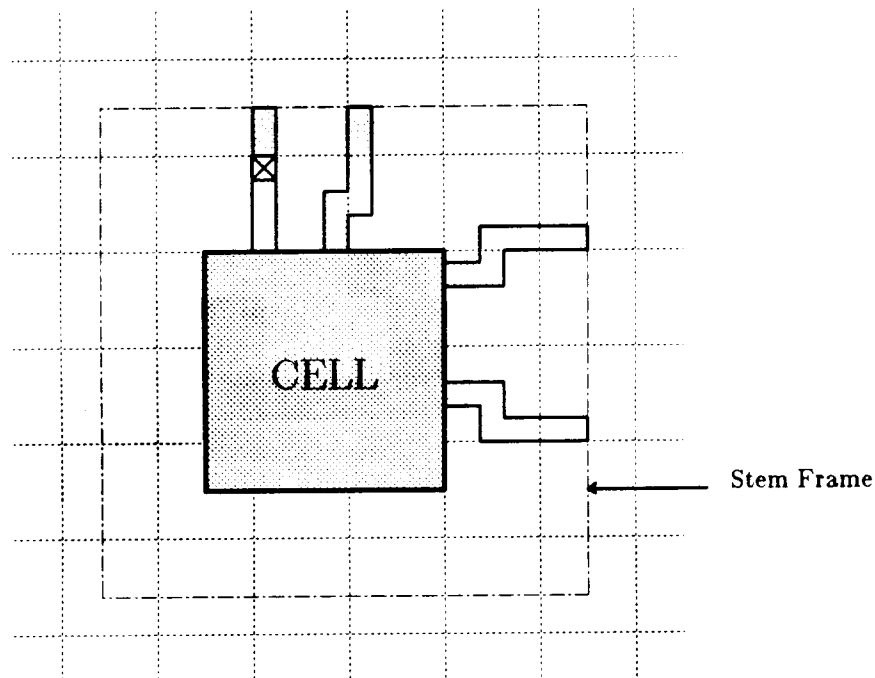


Figure 3.5. The router expands subcell dimensions to leave an empty frame around each subcell to hold *stems* used to jog terminals onto the routing grid. After creating the stem frame the router expands it outward to the next point between grid lines to force all channel boundaries to fall between grid lines and avoid terminal alignment problems.

Magic reserves an area called a *stem frame* around each subcell to allow room for the stems. The stem frame has to be large enough to allow wires to run outward away from the subcell edge and then make a right angle jog without generating design-rule violations. Figure 3.5 shows an example of a stem frame.

3.7. A Fast Channel Decomposer

The first major component of the routing system is a fast channel decomposer. The channel decomposer uses a heuristic to generate a channel structure with a small number of large channels; this simplifies global routing and channel routing. The channel decomposition algorithm owes its simplicity and speed to a data structure called *corner stitching* [Ousterhout 84].

Magic's channel decomposition algorithm is based entirely upon geometric considerations. Since Magic does not modify placement, the channel decomposer ignores channel definition and ordering techniques designed to provide "safe" routing [Kajitani] with revised placement. Instead, like the PI system [Rivest], it routes each rectangular channel area as a switchbox.

3.8. Magellan: An Obstacle-Avoiding Global Router

Magic's global router, named *Magellan*, is another key router component specially designed to deal with obstacles in the routing area. To avoid creating problems the switchbox router cannot successfully route, it considers the effect of obstacles on the resulting routing. Obstacles can completely block areas along channel boundaries; unlike other global routers which simply determine which channel boundaries each net traverses, Magellan is unique in that it examines crossing points along channel boundaries and assigns specific crossing locations to avoid such situations. Obstacles also cause nets to make extra jogs within channels, reducing their effective capacity; the global router must route fewer nets through such regions. These extra jogs increase the total wire length; the global router considers this addition to total net length when selecting a "shortest" path for a net.

The global router relies on hazards for obstacle-avoidance. When evaluating crossings it tries to avoid assigning nets to crossings within hazards, since the switchbox router will likely have to jog these nets out of the way of some obstacle immediately upon entering the channel.

3.9. Detour: An Obstacle-Avoiding Switchbox Router

Magic's obstacle-avoiding switchbox router, named *Detour*, is the third key component of the routing system. Unlike conventional channel routers, it routes rectangular regions that may contain obstacles; depending on the size, type, and aspect ratio of an obstacle, *Detour* may choose to route nets across or around it. *Detour* also handles difficult switchbox routing problems, where pin locations are fixed on all sides of the routing region. At the same time, since it is based on channel routing techniques (the "Greedy" channel router [Rivest and Fiduccia]), it still produces results comparable to the best traditional channel routers for conventional routing problems without obstacles.

Hazards are the key to the channel router's obstacle-avoiding function. They indicate when the channel router should route nets across obstacles and when it should route around them. When it is appropriate to route across obstacles, hazards reserve space for contacts needed to bridge the obstacles. Finally, the channel router uses a variation on the basic hazard concept to implement its switchbox routing mechanism.

3.10. Visual Feedback

Information and error reporting are visual as well as textual. Channel decomposition information is shown as bright white lines overlaying a design. If terminals are spaced too closely together for grid alignment, the global router puts feedback paint into the design at the location of the problem. Similarly, if the switchbox router is not able to complete all connections, it places feedback paint into the design at the location where the error occurred. Associated with each feedback area is a text string describing the nature of the error. This visual feedback generated by the router is viewed and manipulated using the general-purpose viewing facilities built into the Magic system.

3.11. Technology Independence

Aside from the limitation of 2 routing layers, the system is technology-independent. All information concerning routing parameters is contained in a Magic technology file that

is read at startup. The technology file describes the physical characteristics of the routing: the two routing layers, the widths of the wires on each of these layers, the minimum separation between these routing layers and other layers in the design, and the size of contacts between the layers. Routing in a different technology or changing the physical characteristics of the routing in the current technology is accomplished by changing these technology file routing parameters; no recompilation of the program is necessary.

CHAPTER 4

Channel Decomposition

4.1. Introduction

Channel decomposition divides the space between subcells into rectangular, non-overlapping areas called *channels*. The channel is the central data structure of the routing process; once defined, channels are first traversed by a global router to find approximate paths for nets, then individually routed by a channel router. The goal of channel decomposition is to create a channel structure that simplifies the remaining portions of the routing process.

Although what constitutes a good channel decomposition depends on the subsequent phases of the routing process, in general a superior channel decomposition generates a small number of large channels. A global router works faster with a smaller number of channels since there are fewer paths to consider from one point to another. A channel router also does better if channels are larger since this allows more options when choosing how to route nets within a channel. Also, since it typically takes slightly more than the optimal area to route each channel, this excess area accumulates more slowly if a given area is divided up into fewer channels.

Magic's channel decomposer meets this goal by using an effective heuristic, first employed by the PI system [Rivest], that minimizes the sums of the perimeters of the channels. The implementation, which relies on a corner-stitched data structure [Ousterhout 84] is simple and extremely fast.

Like the overall routing system, Magic's channel decomposer is capable of operating without user assistance, but it also allows the designer to examine its results and modify them before proceeding to the subsequent stages of routing. This provides a measure of control over the channel structure and takes advantage of the human designer's expertise.

Designers can control the channel decomposition process by modifying the cell placement. In typical usage the designer generates a channel decomposition, views it on a graphics display, modifies the placement, and generates a new decomposition. Slight placement changes can greatly improve the channel structure and simplify the remainder of the routing process. Channel decomposition typically requires only a fraction of a second of CPU time; this provides an interactive quality and makes it practical to take an iterative approach to the problem. Magic provides visual feedback, displaying the result of automatic channel decomposition by superimposing the channel structure over the chip layout. All of Magic's cell and layout manipulating commands are available to allow the designer to modify the channel decomposition by moving cells; if the cells have connecting wires, then an interactive operation called *plowing* [Scott and Ousterhout] may be employed to move subcells and their attached wiring. Once the channel decomposition is satisfactory, the designer invokes the rest of the routing system to complete the desired connections.

The chapter begins by examining previous approaches to channel decomposition. This is followed by a high-level presentation of Magic's channel decomposition algorithm. Details of this algorithm appear in the next section. The chapter concludes by analyzing the implementation's complexity and performance.

4.2. Background

This section examines the geometric channel decomposition techniques used by VTI's composition editor [Ng], the BBL system [Chen], and the PI system [Rivest]. Other techniques that generate separate channel structures for each routing layer [Hashimoto and Stevens], [Leblond], [Rothermel and Mlynski], and [Wiesel and Mlynski] are unsuitable for Magic's router, since they enforce a layer-per-direction wiring model, while in an obstacle-avoiding environment nets may need to switch layers to cross obstacles. Some previous work is concerned with "safe" orderings of channels, to allow automatic placement revisions to adjust channel widths ([Kajitani], [Chiba], [Dai]); however, since Magic does not modify placement, it does not use such techniques.

4.2.1. VTI's Maximal Horizontal Strips

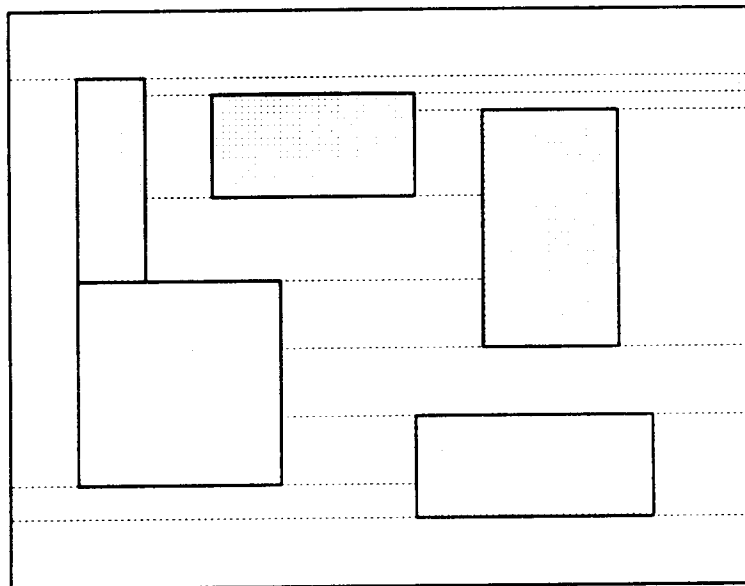


Figure 4.1. VTI's composition editor uses the channel structure defined by maximal horizontal strips. Shaded rectangles represent subcells. Channel boundaries are drawn with dotted lines.

VTI's composition editor divides the routing area between subcells into maximal horizontal strips (Figure 4.1). The vertical edges of all channels are bounded by subcells or by the edges of the routing area. Ng reports that this simple approach produces unsatisfactory results. An early version of Magic's router also used this approach, with similar results. One problem with the maximal horizontal strip approach is that it tends to generate long thin channels that are hard to route efficiently. Another is that it generates a large number of channels. Thus, the routing quality suffers compared to systems that use more sophisticated channel decomposition techniques.

4.2.2. BBL's Bottlenecks

The BBL system [Chen] uses the concept of a *bottleneck* to define routing regions (Figure 4.2). A bottleneck exists between two neighboring subcells or between a subcell and an edge of the chip if and only if there is no other subcell between them. Bottlenecks identify areas between cells where congestion is likely to occur. The bottleneck structure,

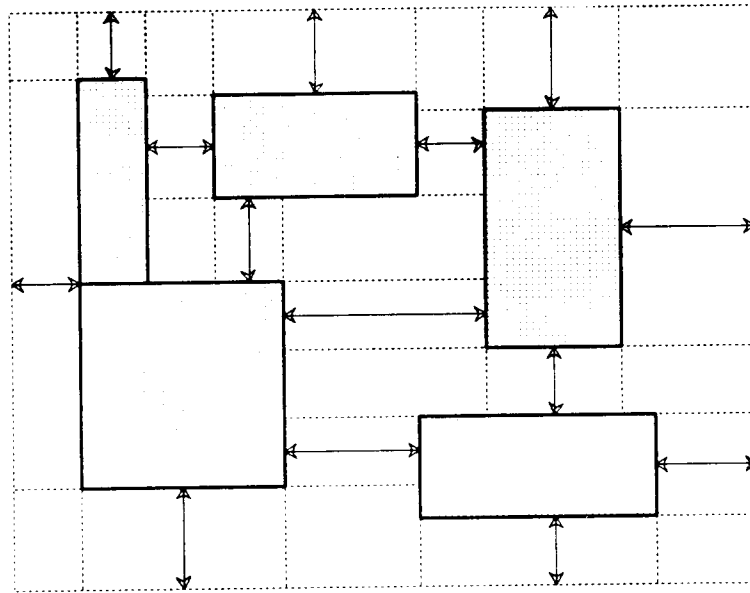


Figure 4.2. BBL derives its channel structure by generating bottlenecks (marked with heavy bold lines) between subcells. Each area containing a bottleneck is a channel, while the remaining regions are switchboxes.

which is used during global routing, allows BBL to revise the subcell placement to provide the amount of space needed by the routing. After global routing, BBL transforms the bottleneck structure into a more conventional channel structure composed of rectilinear channels and switchboxes.

BBL's bottleneck approach is unsuitable because it generates a much larger number of channels without providing any advantages for Magic's routing system. The channel structure that results from the bottleneck approach resembles a maximal decomposition into rectangles where each corner of each subcell generates two channel boundaries. Bottlenecks are useful in placement modification, while Magic does not modify subcell placement. With the bottleneck approach, BBL's global routing operates on a graph abstracted from the physical layout; however, Magic's global router, described in Chapter 5, needs to operate at the physical layout level to decide where to place crossings and to estimate the effects of interactions between particular crossings and nearby obstacles upon the eventual wiring.

4.2.3. The PI System

Published descriptions of the algorithm used by the PI system are very sketchy. In [Rivest] the algorithm given is nothing more than, "The channel-definition heuristic tries to produce a channel structure that minimizes the sum of the lengths of these edges." Another paper, [Soukup], provides only a little more, "The PI System as being developed at M.I.T. creates larger rectangles using only the shorter of the two possible edges for each corner."

Although Magic's channel decomposer is based on the heuristic used by the PI system, the two systems treat obstacles much differently. In the PI system each metal wire segment in the floorplan of a chip creates a *covered channel* which is treated like any other channel except that routing may only cross it on an alternate layer. If the design includes much hand routing, PI's approach generates many small channels which are difficult to route. Further, there is apparently no provision for handling obstacles on any routing layer besides metal. In contrast, Magic's channel router is designed to route around and across obstacles, so it is not necessary to make each obstacle segment a separate channel. The result is a better channel decomposition with fewer, larger channels. Finally, since Magic's implementation was derived completely independently of PI and is based on corner-stitching, the algorithmic details are likely very different.

4.3. Channel Decomposition Algorithm

Magic's channel decomposition algorithm takes as input a placement of subcells and a rectangular boundary for the overall routing area. It divides the area between subcells and within the routing area into rectangular non-overlapping channels. It does this by enumerating all convex corners of subcells, at each corner generating a horizontal or a vertical channel boundary, whichever is shorter (Figure 4.3). By choosing the shorter of the two alternatives at each corner, the sum of the perimeters of channels is reduced, and the result is a small number of large channels. The result of this channel decomposition algorithm applied to the placement of Figure 4.1 is shown in Figure 4.4.

1. For each convex subcell corner
 - 2. dh = horizontal distance to the nearest channel boundary
 - 3. dv = vertical distance to the nearest channel boundary
 - 4. if($dh < dv$)
 - 5. mark a new horizontal channel boundary
 - 6. else
 - 7. mark a new vertical channel boundary

Figure 4.3. A high level description of Magic's channel decomposition algorithm.

After generating and viewing a channel decomposition the designer may often simplify the channel structure by slightly modifying the placement. Figure 4.5 shows how the placement of Figure 4.4 can be adjusted to produce a smaller set of larger channels; this should make the example easier to route.

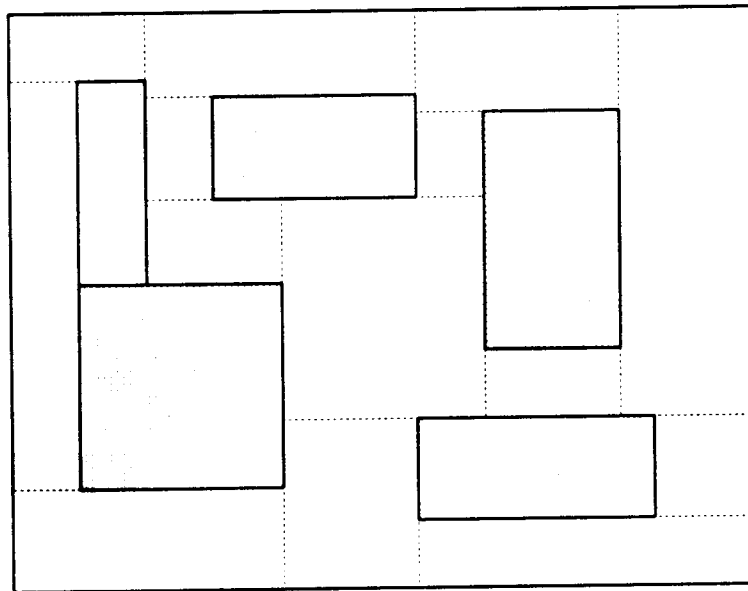


Figure 4.4. Magic's channel decomposer produces this structure when applied to the same subcell placement as in Figures 4.1 and 4.2. At each convex subcell corner it creates a channel boundary in the shorter of the two possible directions.

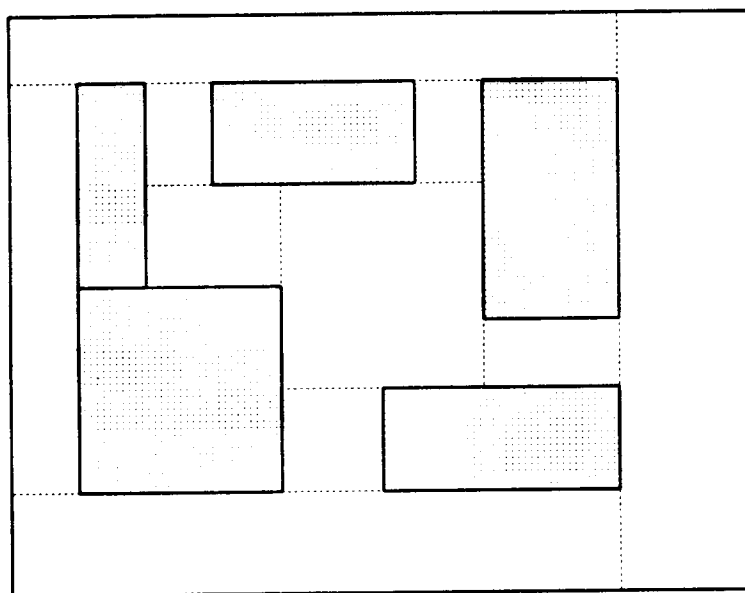


Figure 4.5. The simplified channel structure after the designer views the channel decomposition and modifies the placement.

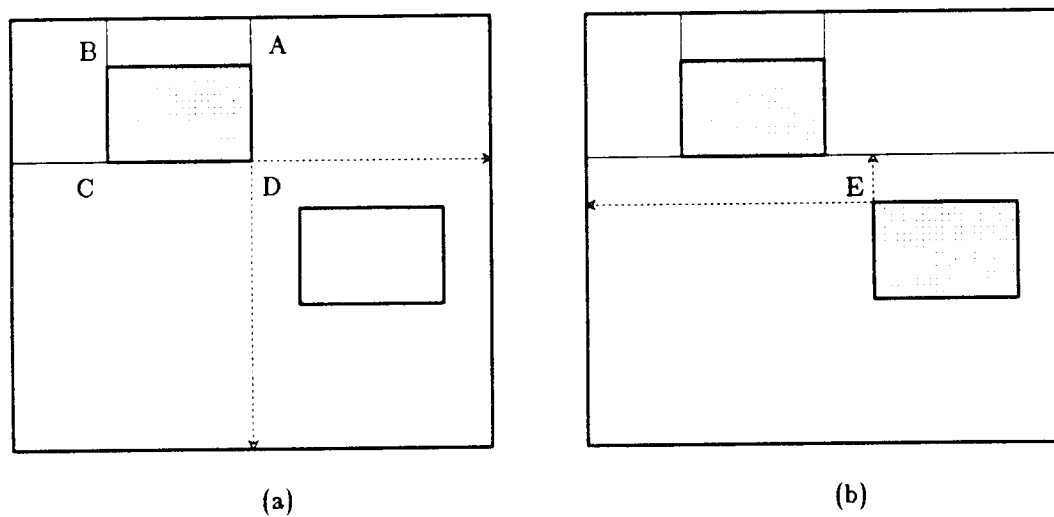


Figure 4.6. The channel decomposition algorithm searches horizontally and vertically from subcell corner D to find the nearest previously-defined channel boundary, as indicated by the dotted lines in (a). The shorter horizontal leg from corner D becomes a valid channel boundary for a subsequent search from corner E, as shown in (b).

The channel decomposition algorithm is feature-based, enumerating the corners of each subcell in the layout. At each convex subcell corner the algorithm searches both horizontally and vertically outward to find the nearest adjacent subcell, edge of the routing area, or previously-defined channel boundary, then chooses the channel boundary to be in the direction of the shortest edge. The channel boundaries established early in the decomposition are used in deciding which way to generate boundaries from later corners (Figure 4.6); thus, the algorithm's results depend on the order in which corners are enumerated. Since channels are strictly rectangular (no L-shaped channels are allowed), each convex subcell corner must lie on at least one constructed channel boundary (Figure 4.7).

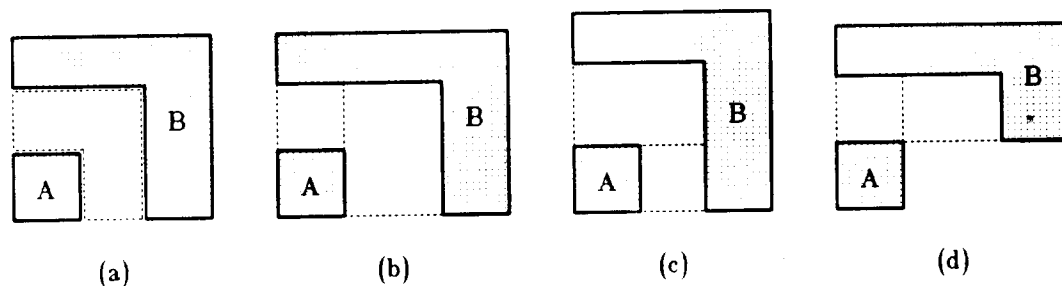


Figure 4.7. Since L-shaped channels are not allowed (a), each convex subcell corner lies on at least one constructed channel boundary. The boundary may be either vertical (b) or horizontal (c). A subcell corner may lie on two constructed channel boundaries as in (d), where the vertical boundary is generated by the northeast corner of subcell A and the horizontal boundary is later generated by the southwest corner of subcell B.

4.4. Implementation

This section describes Magic's implementation of the channel decomposition algorithm, which is based on corner stitching. Corner stitching [Ousterhout 84] is a data structure for Manhattan geometries that explicitly represents both material and space. Each corner-stitched plane is composed of non-overlapping rectangular tiles that completely cover the plane (Figure 4.8a). Each point lies within exactly one tile; a tile contains its left and bottom edges but not its top or right edges. Tiles of the same type are stored in maximal horizontal strips to prevent fragmentation and to provide a canonical form (Figure 4.8b,c). Each tile is linked to neighboring tiles with four pointers: two at the

upper right corner, and two at the lower left corner. Database routines traverse these pointers to provide fast point searching, fast area enumeration, fast neighbor finding, and fast database updates.

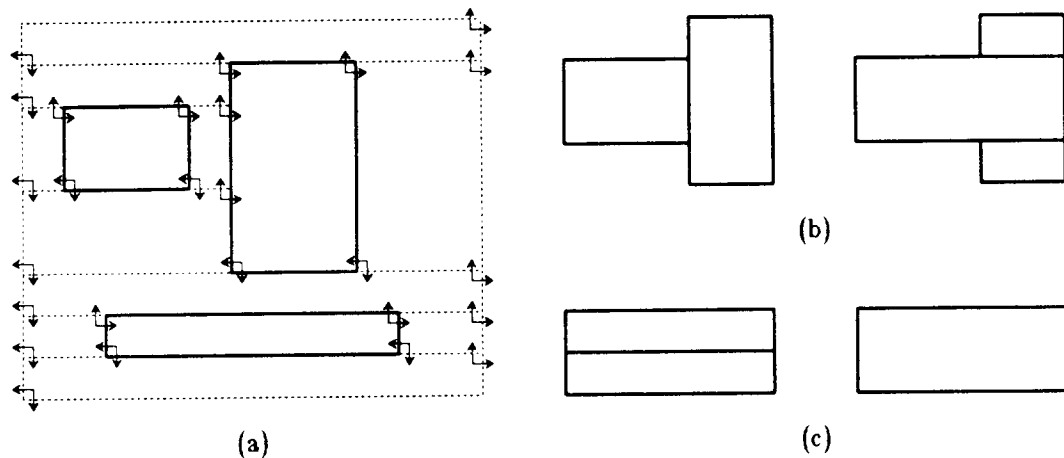


Figure 4.8. Corner-stitching explicitly represents both material (solid outlines) and space (dotted outlines). Each tile contains four pointers to adjacent tiles (a). Areas of the same type of material are represented with horizontal strips that are as wide as possible. In (b) and (c), the structure on the left is illegal and is converted into the tile structure on the right.

The channel decomposition algorithm uses a corner-stitched plane to represent the routing area. Initially, solid tiles represent subcells and space tiles represent the areas between subcells. The decomposition algorithm splits and merges horizontal strips of space tiles to produce a new set of space tiles with a reduced sum of perimeters. Each of these new space tiles represents a routing channel with the same location and dimensions (Figure 4.9).

4.4.1. Identifying Convex Corners

The outer loop of the channel decomposition algorithm is an enumeration of each of the convex subcell corners within the routing area. This is done using the standard *area enumeration* algorithm for corner-stitched tile planes, which visits a tile when all of the tiles above it and to its left have already been processed. As each subcell tile is enumerated its four corners are examined in turn to see if they are convex. The corners

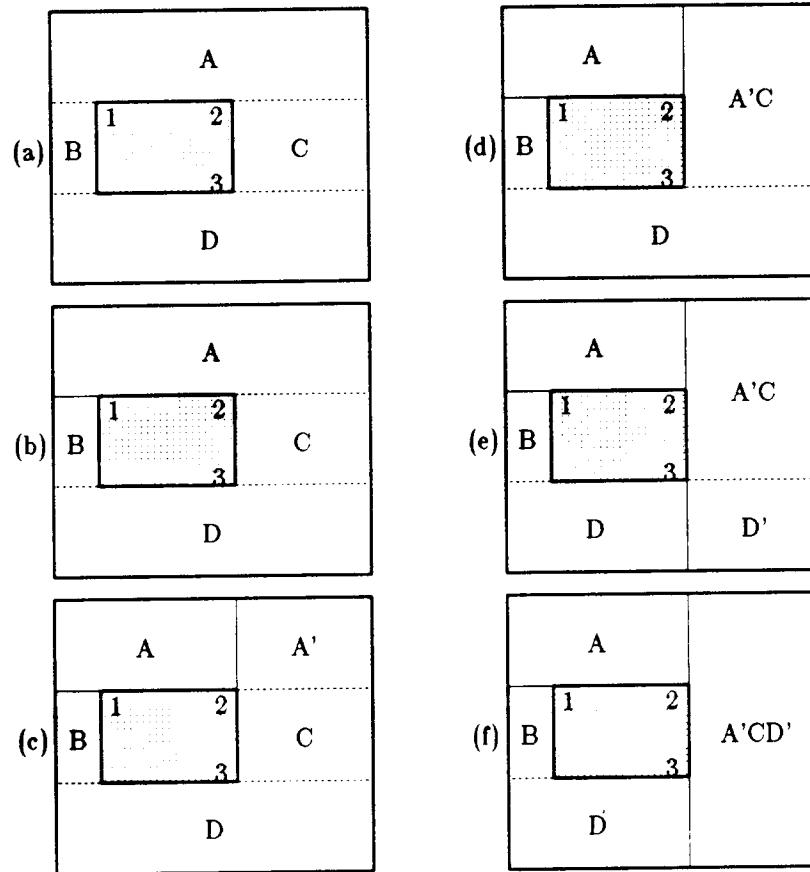


Figure 4.9. The algorithm splits and merges space tiles, creating a structure where each space tile represents a channel. *Initial edges*, drawn with dotted lines, are artifacts of the original formation as maximal horizontal strips. *Valid edges*, drawn in solid lines, are constructed channel boundaries. Figure (a) shows an initial tile plane. In (b) the edge between tiles A and B becomes valid after processing corner 1. As part of processing corner 2 in (c), tile A splits into A and A'. In (d), causing a vertical merge between A' and C. In (e) tile D splits into D and D' after processing corner 3. Tile D' merges vertically with A'C', forming tile A'CD' in (e). The last step, marking a valid boundary between tiles B and D, is not shown.

are enumerated in the following order: southeast, southwest, northwest, northeast.

The test for convexity, as shown in Figure 4.10 for a northeast corner, looks for material directly above the corner (Figure 4.10 (b)) and diagonally opposite the corner (Figure 4.10 (c)). If no material (in this case subcell tiles) is found either above or diagonally opposite the corner, then the corner is convex.

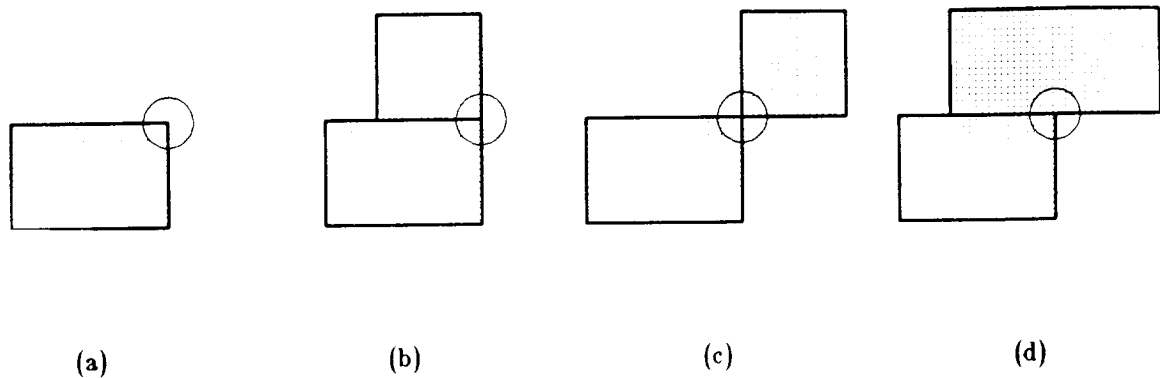


Figure 4.10. Testing a northeast subcell tile corner for convexity. Figure (a) is a legitimate convex corner. Figure (b) is not convex, since there is material above the corner. Figure (c) is not convex, since there is material diagonally-opposite the corner. Figure (d) is not convex, since there is material both above and diagonally-opposite the corner.

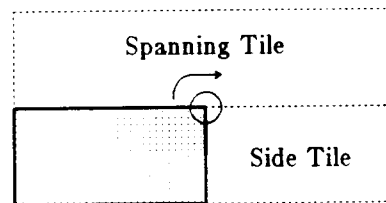


Figure 4.11. The test for a convex corner checks to see if the x-coordinate of the right edge of the spanning tile is greater than the x-coordinate of the corner in question.

The test for convexity is even simpler than suggested by Figure 4.10, through the use of *spanning* and *side* tiles. A spanning tile lies immediately above a north-facing corner or immediately below a south-facing corner, and may be either a solid tile or a space tile. A side tile lies immediately to the right of an east-facing corner or immediately to the left of a west-facing corner (Figure 4.11). Side tiles are always space tiles, since subcell tiles are stored in maximal horizontal strips.

The tests for material directly above and diagonally opposite the corner can be combined into a test of the corner's spanning tile. A corner is not convex unless the spanning tile truly spans the corner's x-coordinate. If, as in Figure 4.10 (b) or 4.10 (c), the

“spanning” tile has an edge directly above the corner, then the corner is not convex; this works regardless of whether the spanning tile is a subcell tile or a space tile. If the spanning tile really spans the corner, then the corner is convex if the spanning tile is a space tile; otherwise, the corner is concave (Figure 4.10 (d)).

4.4.2. Horizontal and Vertical Distances

The heart of the channel decomposition algorithm is the construction of a channel boundary at a convex subcell corner. A channel boundary can extend either horizontally or vertically from the corner location. It terminates when it runs into a subcell, an edge of the routing region, or some other previously-defined channel boundary. The channel decomposition algorithm examines both choices and selects the alternative whose boundary length from corner to termination point is the shortest. Since the algorithm always chooses the shortest of the two alternatives at each point, this tends to minimize the perimeters of channels.

Measuring the distance horizontally from a subcell corner to a valid channel boundary is easy in the corner-stitched tile plane. The distance is found by examining the x-coordinates of the corner's spanning tile and side tile. For the northeast corner in Figure 4.12 the distance is simply the minimum of the x-coordinates of the spanning tile and the side tile. If the spanning tile represents a subcell, then the corner is not convex, and the algorithm skips the corner.

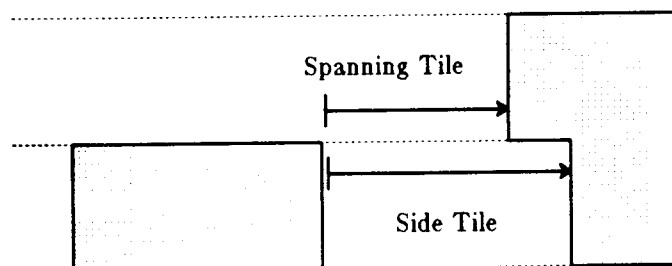


Figure 4.12. For a northeast corner the horizontal distance to a valid channel boundary is simply the minimum of the x-coordinates of the spanning tile and the side tile.

Measuring horizontal distance in this manner depends on the property that all vertical edges in the channel decomposition plane represent valid channel boundaries. At the start of the decomposition process the only vertical edges are those between subcell tiles and space tiles. Vertical edges are only added to the tile plane when space tiles are split as in Figure 4.9 to mark valid vertical channel boundaries. Thus, all vertical edges represent valid channel boundaries.

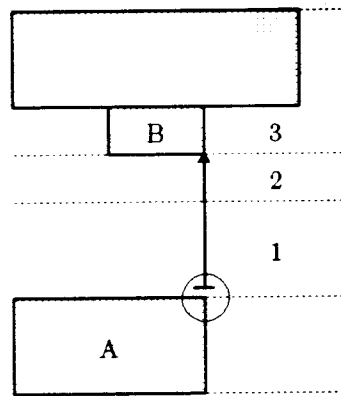


Figure 4.13. Determining the vertical distance from the circled corner to a valid channel boundary may involve traversing a number of space tiles and checking to see if any of their edges represent valid channel boundaries. When tile 3 is checked, the lower edge of tile B terminates the search.

Measuring vertical distances to valid channel boundaries is more complicated. The problem is that only some of the horizontal space tile edges represent valid channel boundaries. Other horizontal edges exist simply as artifacts of the original formation of the space tiles into maximal horizontal strips; these edges will eventually be made into valid channel boundaries or else disappear as tiles are split and merged during the course of the decomposition algorithm.

Figure 4.13 shows the vertical distance measurement from a northeast subcell tile corner. The search for a valid channel boundary progresses vertically upward beginning with the spanning tile (1) and progressing through space tiles directly above the originating corner (2 and 3), using the standard *point search* algorithm for corner-stitched tile planes to find the tile containing a given point. When a valid horizontal channel

boundary is located, the vertical distance is simply the distance from the corner to the channel boundary.

A valid horizontal channel boundary can be encountered in several ways. The horizontal edge of any subcell tile represents a valid channel boundary. The upper and lower edges of the routing region also represent valid boundaries. A horizontal boundary between two space tiles may also represent a valid channel boundary (how to determine this will be explained shortly). If the horizontal boundary is not valid, it is necessary to check the x-coordinates of the tile to the left (Figure 4.13, tile B) to detect the case shown in Figure 4.13 where a vertical subcell edge coincides with the line of search and the valid horizontal subcell tile edge would otherwise be missed (a tile includes its left edge but not its right). If none of these conditions apply, the vertical distance measurement advances to the next vertically adjacent tile and repeats the process.

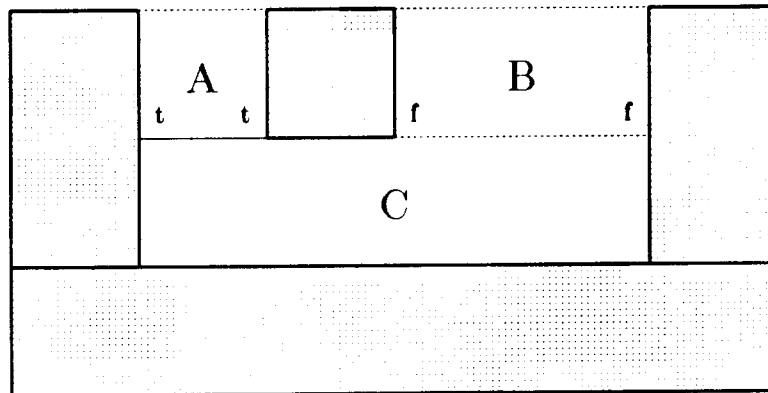


Figure 4.14. When one space tile's horizontal edge completely spans another's, the flags associated with the shorter of the two edges are identical and determine the status of the edge they share. Flags along A's lower edge are set to *t* to indicate that the edge between tiles A and C (drawn with a solid line) is a valid edge. Flags along B's lower edge are set to *f* to indicate that the edge between tiles B and C (drawn with a dotted line) is not a valid edge.

To identify the horizontal space tile edges that represent valid channel boundaries, Magic's channel decomposer associates two boolean flags with each horizontal space tile edge. When one space tile's horizontal edge completely spans another's, one bit is sufficient to determine if the edge is valid. In this case the flags associated with the

shorter of the two edges are identical and determine the status of the edge they share (Figure 4.14).

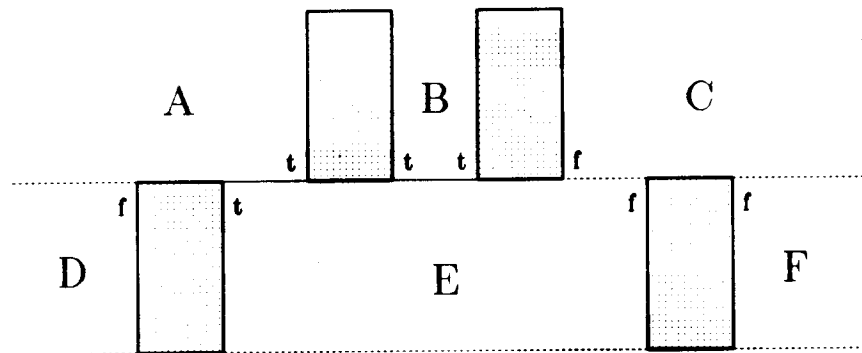


Figure 4.15. When space tiles partially overlap (tiles A, C, and E), two status bit per horizontal edge are needed to indicate the status of the overlapping edges. The edge between tiles A and E is valid; A's lower right flag and E's upper left flag are set to *t* to indicate this. The edge between tiles C and E is an initial edge; C's lower left flag and E's upper right flag are set to *f* to indicate that it is not a valid edge.

In some cases both status bits are needed to correctly mark each horizontal edge (Figure 4.15). This occurs when a horizontal space tile edge partially overlaps two other space tiles, one at each corner. In this case the horizontal edge requires two boolean flags: one for the horizontal edge's leftmost segment and one for its rightmost segment.

The algorithm for measuring the vertical distance is presented in Figure 4.16. This pseudo-code listing only shows the procedure for north-facing corners. Since at the start no horizontal edge is valid, the channel decomposition algorithm clears all edge status bits at the start of the decomposition process.

4.4.3. Splitting and Merging Tiles

Once the decomposer determines the direction from the corner to the nearest existing channel boundary, the algorithm creates a new channel boundary from the corner to that existing boundary. When the vertical direction is shorter the algorithm establishes a vertical channel boundary by splitting all space tiles between the corner and the nearest

```

1. nextTile=spanning tile;
   while(nextTile.type!=subcell)
   {
2.   currTile=nextTile;
3.   nextTile=tile above new currTile at corner's x-coordinate;
4.   if(nextTile.lower_y==routingArea.upper_y)
       break; /* Found top of the routing area */

5.   if(nextTile.lower_x==corner_x)
       break; /* Subcell edge or valid boundary */

6.   if((currTile.lower.x <= nextTile.lower.x) and
       (currTile.upper.x >= nextTile.upper.x))
       {
           if(nextTile's lower left horizontal status bit==1)
               break; /* Valid horizontal boundary */
       }
       else
7.   if(currTile.upper.x > nextTile.upper.x)
       {
           if(nextTile's lower right horizontal status bit==1)
               break; /* Valid horizontal boundary */
       }
       else
8.   if(currTile's upper right horizontal status bit==1)
       break; /* Valid horizontal boundary */
   }
   distance=nextTile.lower_y - corner_y;

```

Figure 4.16. The algorithm for measuring the vertical distance from a north-facing subcell tile corner to a valid horizontal channel boundary.

existing channel boundary.

If the resulting split tiles have upper or lower neighbors that are space tiles with identical horizontal span, then they are merged together as in Figure 4.17 (a) and (b). In some cases this may delete part of a previously-defined valid horizontal channel boundary. This is okay since the horizontal edge segment is completely redundant as it is bounded at both ends by vertical (valid by definition) edges.

If the tiles resulting from these vertical merges match horizontally-adjacent tiles (except for the edge representing the new valid channel boundary), further merging can be

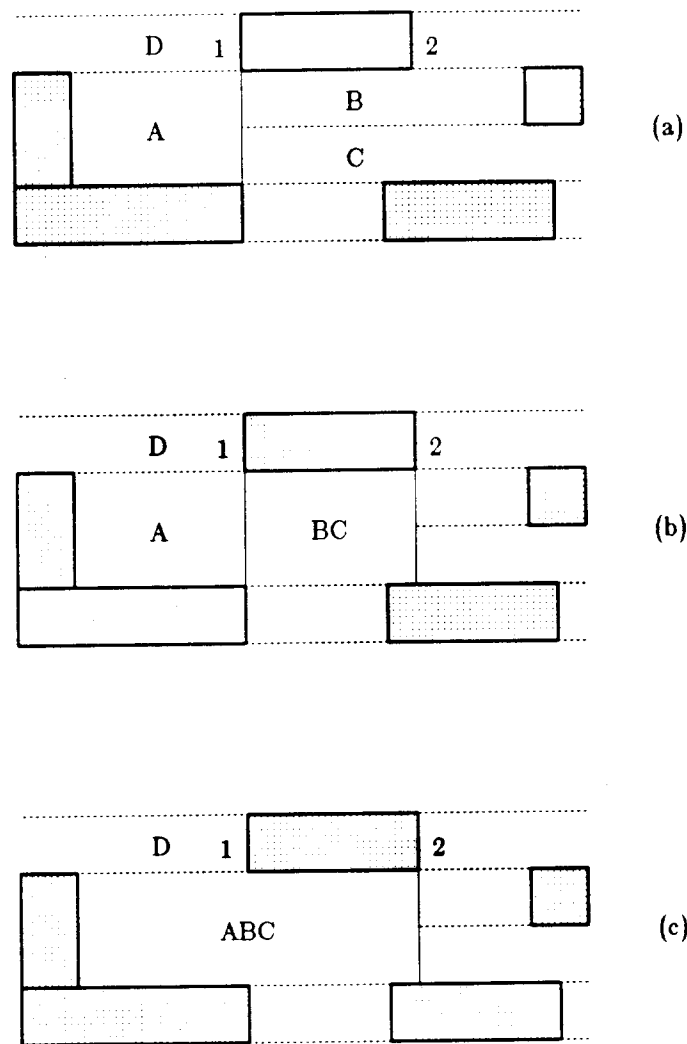


Figure 4.17. Splitting upward from corner 2 allows tiles B and C to merge (b) vertically. This merge allows tiles A and BC to merge horizontally, deleting the vertical edge from corner 1 (c).

done in the horizontal direction (Figure 4.17 (c)). Note that horizontal merging deletes a vertical edge, and vertical edges always represent valid channel boundaries. This is okay since by deleting a redundant vertical boundary the sums of the perimeters of all channels is reduced.

4.4.4. Setting Horizontal Status Flags

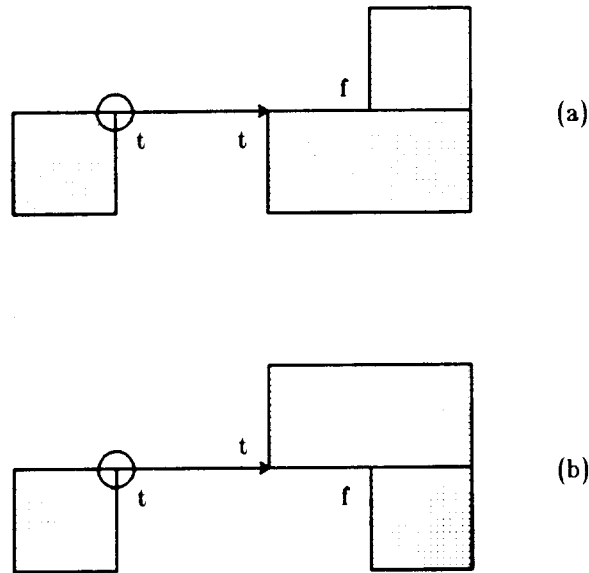


Figure 4.18. When creating a valid horizontal channel edge from the circled northeast corner to the right, set valid horizontal valid flags. Always set the northwest flag in the side tile. If the right edge of the side tile is to the left of the right edge of the spanning tile, also set the northeast flag in the side tile (a), otherwise, set the southeast flag in the spanning tile (b).

When the horizontal direction is shorter, the algorithm does not have to split tiles since there is already a space tile boundary from the corner to the existing channel boundary; however, to identify the horizontal tile edges that represent valid channel boundaries the algorithm sets status flags in the space tiles along the edge in question. The algorithm always sets a flag in the subcell corner's side tile. Where appropriate the algorithm also sets the flags in the spanning tile. The three rules for setting horizontal boundary flags are illustrated in Figure 4.18 for northeast subcell corners:

1. Always set the side tile's northwest valid flag.
2. If the spanning tile's upper x-coordinate matches or exceeds the side tile's upper x-coordinate, also mark the side tile's northeast valid flag.
3. If the spanning tile's upper x-coordinate is less than or equal to the side tile's upper x-coordinate, mark the spanning tile's southeast valid flag.

4.4.5. Cutting Corners

Part of minimizing the sums of perimeters of channels is avoiding the generation of unnecessary channel boundaries. As shown in Figure 4.5 a corner must have either a horizontal or a vertical channel boundary, but generating both unnecessarily increases the sums of perimeters of channels. Thus, the decomposition algorithm only processes those corners not already incident upon a valid channel boundary.

All edges of the routing region are valid channel boundaries, so the algorithm skips corners touching the edge of the routing region. It is also possible that while processing one corner the decomposition algorithm creates a valid channel boundary incident upon an as-yet-unprocessed corner; when that corner is later enumerated the presence of the valid channel boundary must be detected so the corner can be skipped (Figure 4.18).

Any vertical space tile edge represents a valid boundary; if a vertical tile boundary runs into a corner then the corner can be skipped. The test for this is to see if an x-coordinate of the spanning tile matches the x-coordinate of the corner. Otherwise, there must either be a valid horizontal channel boundary at this corner, or no valid channel boundary yet exists. The algorithm checks the status flag of the appropriate corner of the side tile to see if the horizontal boundary is valid. For example, when checking for a valid horizontal boundary while processing a northeast corner, the algorithm looks at the northwest status flag for the space tile immediately to the right of the corner. If the flag is set then the boundary is valid and the corner can be skipped.

4.5. Analysis

Compared to BBL or VTI's composition editor, Magic's channel decomposer produces better results: a smaller number of channels with dimensions that make them easier to route. Using an example subcell placement copied from a BBL report, BBL generates 25 channels (Figure 4.2), while Magic generates 14. Maximal horizontal strips also produces 14 channels, although some of them are so narrow that they can only be routed with some difficulty. The maximal horizontal strip method produces channels whose perimeter sum

is 171 units; Magic's sum of perimeters for the same placement is only 128. These measurements are summarized in Figure 4.19.

Method	Channels		Perimeter	
	Number	Ratio	Sum	Ratio
Max. Horiz. Strips	14	1.4	171	1.54
Bottlenecks	25	2.5	167	1.50
Magic	14	1.4	128	1.15
Magic, Adjusted	10	1.0	111	1.0

Figure 4.19. Comparison of channel decomposition methods for the placement of Figure 4.4. Magic's channel decomposer produces a significantly smaller number of channels and a significantly smaller sum of channel perimeters.

Typical run time complexity is $O(N)$, where N is the number of subcells in the layout; however, in the worst case the algorithm is $O(N^2)$. The typical linear time behavior occurs because the average subcell corner "sees" a constant number of space tiles during channel decomposition. The worst-case behavior occurs when subcells generate $O(N)$ space tiles, and each of the N subcells must search vertically through $O(N)$ space tiles (Figure 4.19). Processing the lower corners of the topmost subcell requires $O(N)$ space tiles to be examined; processing subsequent subcells requires $O(N-1)$, $O(N-2)$, $O(N-3)$, ... space tiles to be examined, yielding the stated worst-case behavior.

Corner stitching provides a very efficient data structure for channel decomposition. The horizontal and vertical searches for a nearest previously-defined channel boundary are simple traversals of neighboring tiles in the plane. Updating the data base during the splitting and merging of tiles representing channels is fast and efficient using corner stitching. Furthermore, the corner-stitched implementation runs very quickly, using less than 0.1 second of VAX 11/780 CPU time on every design tried to date.

Use of Magic's corner-stitched data base routines allows a simple, compact implementation of the channel decomposition algorithm. When stripped of comments the channel decomposition module is only 437 lines of C code.

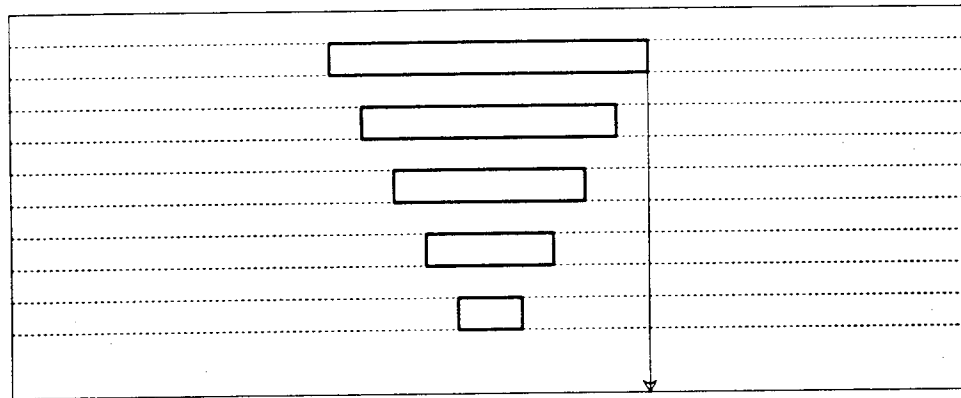


Figure 4.20. The channel decomposition algorithm runs in $O(N^2)$ time for this worst case layout, where subcell tiles are processed from the top down. The arrow crosses the $O(N)$ space tiles that must be processed by the topmost of the subcells.

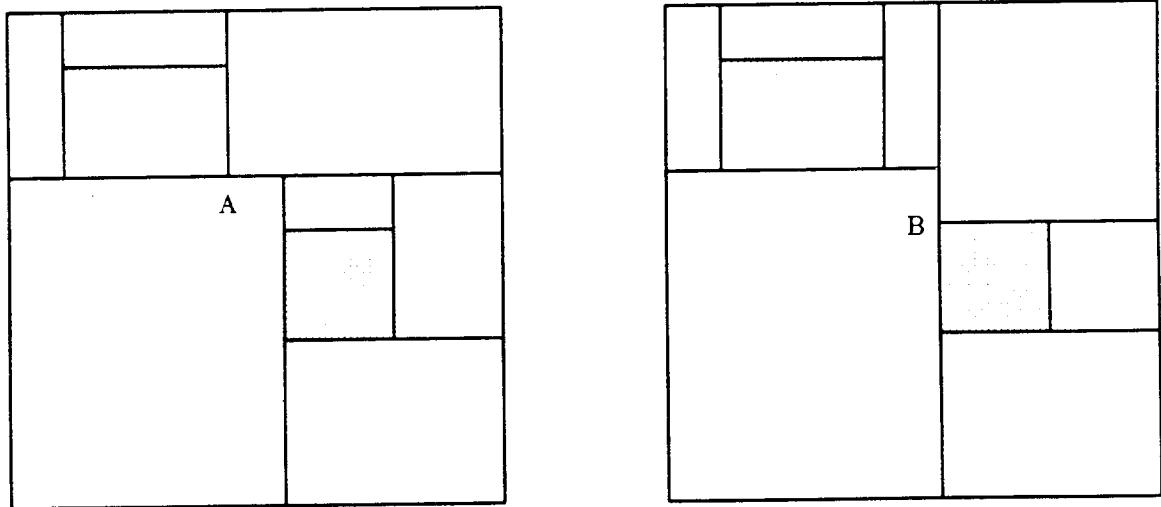


Figure 4.21. The channel decomposition algorithm is order-dependent. Depending on whether corner A or corner B is enumerated first, the final result may be either of these channel structures.

As a final observation, although the corner extension algorithm tends to minimize the sums of the perimeters of channels, it cannot guarantee this. The problem is that this procedure is order dependent: results depend on the enumeration order for the cell corners. Figure 4.21 shows two possible channel decompositions, depending on whether corner A or corner B is enumerated first.

CHAPTER 5

Magellan--Magic's Obstacle-Avoiding Global Router

5.1. Introduction

A global router generates a wiring path for each net. Each path specifies a sequence of channels through which a net will pass, without determining the exact placement of the net's wiring within those channels (a channel router does the actual wiring). Path selection considers the effect of choosing various alternative sequences of channels, with the overall goal of simplifying the resulting channel routing problems. Since longer wires occupy space that could be used to route other signals and also increase resistance and capacitance in circuits, a global router usually chooses the shortest path from one point to another. In some cases, however, it may choose a less direct path to avoid routing through congested channels whose wire capacities are already fully used.

Obstacles complicate the routing problem. In some places, they may occupy all of the routing layers; such areas will have to be avoided completely by the channel router. Single-layer obstacles limit the choice of routing layer; they may make the channel router's job difficult or impossible. The global router must ensure that the channel router will have enough space to route around obstacles, and must take the obstacles into account when computing the capacities of channels. In some cases, it may be better to choose a longer path than to attempt to pass through an area with many obstacles.

To operate in this obstacle-avoiding routing environment, Magic's global router named *Magellan* chooses not only the sequence of channels for each net, but also the exact points (and layers) where nets cross channel boundaries. These points are called *crossings* and the selection process is called *crossing placement*. Because the crossing points are fixed, the resulting routing problems are switchboxes rather than channels.

Crossing placement originated in the PI system [Rivest] as a separate step occurring after global routing. Unlike the PI system, Magic's routing system performs crossing placement during global routing. As it generates shortest paths for global routing, it evaluates crossing points along switchbox boundaries to assess the effects of obstacles. This allows the global router to select crossings that result in the simplest switchbox-routing problems. In cases where all of the available crossings are undesirable, the global router will consider alternate, "next-shortest" paths for the net.

5.2. Global Router Overview

The global router takes as input a cell placement, a collection of channels, and a net-list specifying desired connections between pins on cell edges. The cell placement is determined by the designer, and is not modified by the routing tools. The channel structure is determined automatically by the channel decomposer. The net-list is provided by the designer, and specifies one or more nets, each of which is a set of terminals to be wired together. The global router outputs a set of switchbox problems ready to be routed by Magic's switchbox router. After global routing, crossing points for each net are completely specified (Figure 5.1).

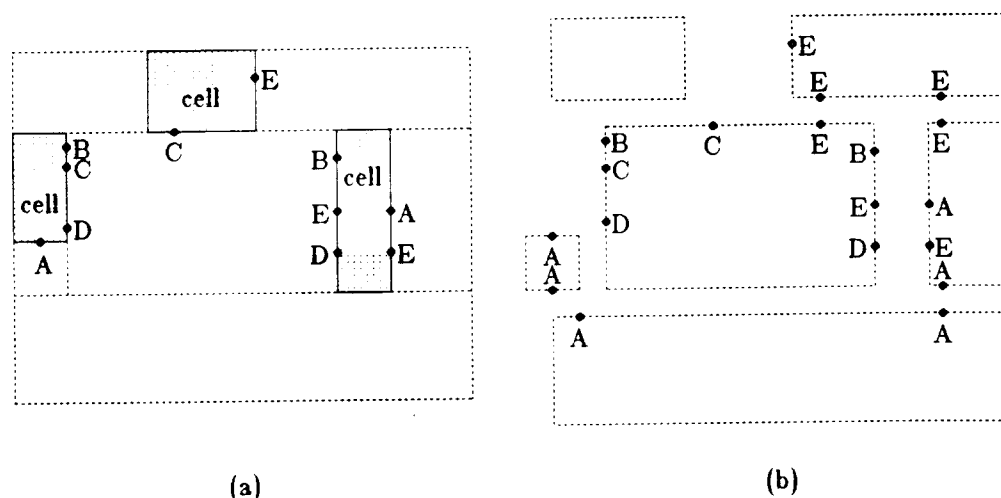


Figure 5.1. The global router is given a set of cells, a set of channels, and a net-list, as in (a). It produces a set of switchbox routing problems, as in (b), with the crossing points for each net completely specified.

The global router processes nets sequentially. It makes a rough estimate of the wire length of each net and sorts the nets according to the estimates (Section 5.6 discusses the sorting in more detail). Nets are then processed one-by-one, shortest net first. After the global routing for one net is completed, the next net is then considered.

```

0. sort nets;
1. for each net
2. { source points = (first pin);
3.   for each additional pin in the net
4.   {   repeat
5.       { find the (next) shortest path to target pin;
6.         if (path length > best total cost so far)
7.           break;
8.         adjust crossings;
9.         path total cost = path length + crossing penalties;
10.        if (path total cost < best total cost so far)
11.          save new path;
12.        }
13.      source points += new path points;
14.    }
15.  }

```

Figure 5.2: Overview of the global router. Lines 3 and 10 are used for nets with 3 or more pins, and are explained in Section 5.7.

Each net is processed in two steps, involving shortest-path generation and crossing placement. The process is outlined in Figure 5.2. For most of the discussion that follows, each net is assumed to have two terminals. Section 5.7 generalizes the algorithm to handle nets with more pins. The global router begins by finding the sequence of switchboxes through which to route the net in order to minimize wire length. As it creates this shortest path, it makes an initial crossing placement by choosing crossing points that will minimize the net length.

After the shortest path has been found, the global router re-examines the initial crossing points that were chosen for the path. Each crossing is penalized for nearby obstacles and other undesirable features that will complicate the task of the switchbox router.

The penalties are computed as distances. A penalty of 10 units means that it is preferable to choose a path that is up to 10 units longer in order to avoid this crossing. Where penalties are assessed, the global router will examine other nearby crossings and may replace the initial crossing choice with another crossing whose penalty is smaller. Section 5.5 describes how the penalties are computed.

After finding the best crossings along the path, Magic adds all the penalties for those crossings to the wire length for the path and uses this as the *total cost* of the path. The desirability of a path thus depends on both its length and the quality of its switchbox crossings. The shortest path may not be the best one if it has particularly bad crossings. For this reason, the global router does not stop with the shortest path. Instead it generates more paths, next-shortest first, in the hope that it will find one whose crossings are attractive enough to give it a lower total cost. Since paths are examined shortest-first, the algorithm can terminate when a path is found whose length alone is greater than the best total cost seen so far. When the global router finds the lowest-cost path for a net, it marks the path's crossing points as "in use" by that net, and then proceeds to the next net. The following section gives a more detailed discussion of this approach to crossing placement.

5.3. Shortest Path

One of the global router's key components is a multiple-source, single-destination shortest-path generator. The shortest-path generator is somewhat like Lee-type algorithms [Lee] in that it extends a wavefront outwards from source to destination. However, there are two significant differences between the Magic router and the standard Lee approach. First, the Magic router extends the wavefront one switchbox at a time [Clow] instead of one grid unit at a time. Second, Magic's shortest-path generator is directed: instead of extending the wavefront uniformly on all sides, it first explores the most direct route to the destination. These two differences result in a dramatic speed-up over the conventional Lee approach. This directed search technique, described in [Nilsson], is attributed to Hart, Nilsson, and Raphael.

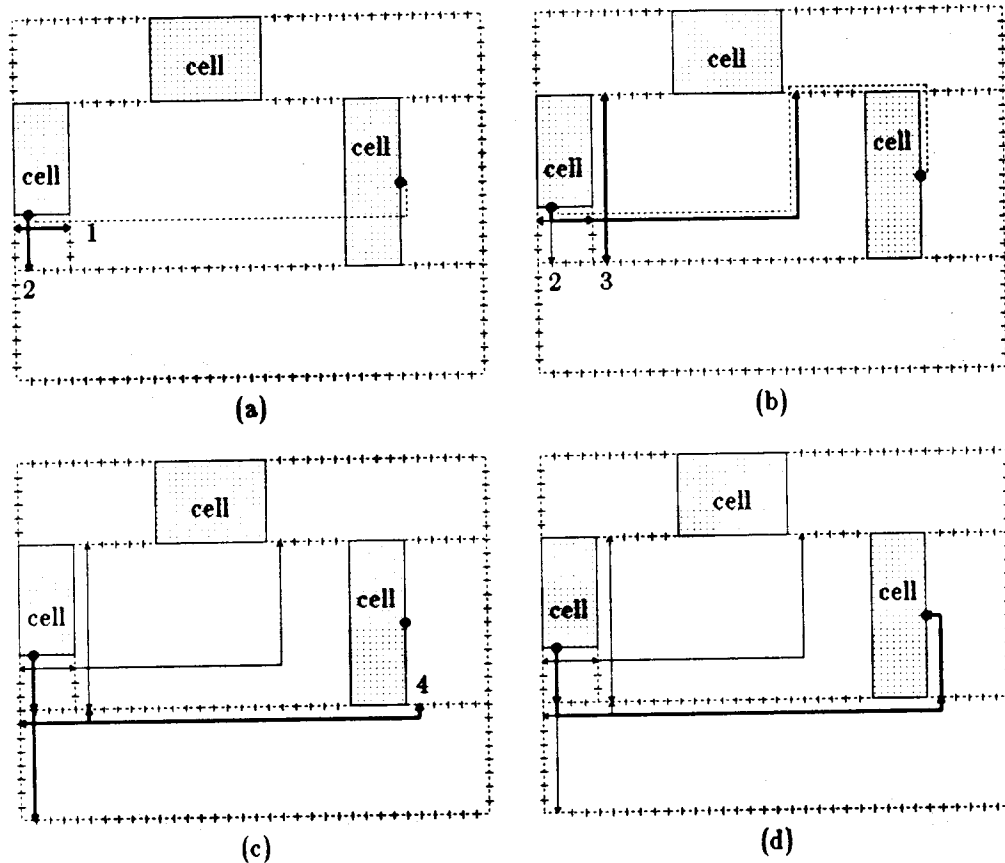


Figure 5.3. The shortest-path algorithm expands a search wavefront from the source point towards the destination. In each step it extends the most promising path, which is shown with dark lines. In (a), three partial paths are generated from the source point. Path 1 appears to be the most promising: it has the smallest sum of distance traveled from the source and estimated distance to the destination (the dotted line shows the distance used for comparison; in this case the estimate is optimistic since a subcell blocks the most direct path to the destination). In (b), path 1 is extended outward one switchbox in all directions. The dotted line in (b) shows the distance estimate for one of the new paths. The estimated costs for all of the new paths are at least as great as for path 2, so path 2 is extended in (c) (the choice between it and path 3 is arbitrary). In step (d) the most promising path reaches the destination, thereby ending the search.

The shortest-path algorithm constructs a number of partial paths through sequences of switchboxes from the source point toward the destination point, as shown in Figure 5.3. It stores these partial paths in a heap, sorted by estimated path length. The estimated length for a partial path is the actual distance travelled from the source point plus the Manhattan distance between the path's endpoint and the destination point. The

estimated distance is a lower bound on the path length: the actual distance to the destination may be greater than the estimate if there are cells blocking the most direct path (see Figure 5.3).

On each iteration the shortest-path algorithm removes the most promising partial path (i.e. the one with shortest estimated length) from the heap. If the path's endpoint is in the destination point's switchbox, then the shortest-path algorithm terminates successfully, returning this path. Otherwise, the algorithm extends the path to all neighboring switchboxes and estimates the length of each new path. The new partial paths are added back into the heap in sorted order, and the process repeats. If the heap ever becomes empty then no partial path can be extended, so the search terminates unsuccessfully. Figure 5.4 summarizes the algorithm.

```

1. for each source point P
2.   HeapAdd(pin, ManhattanDistBetween(P, target));
3. repeat
4.   { P = HeapRemoveTop();
5.   if no more points
6.     failure;
7.   if switchbox(target point) contains P
8.     success;
9.   for each switchbox C1 adjacent to switchbox(P)
10.    if !(loop created)
11.      { P1 = closest available crossing into C1;
12.      HeapAdd(P1, Dist(P) +
13.        ManhattanDistBetween(P, P1) +
14.        ManhattanDistBetween(P1, target));
15.    }
16.  }
17. until success or failure;

```

Figure 5.4: The shortest-path algorithm repeatedly extends the most promising partial path toward the target point, one switchbox at a time.

This procedure guarantees that the first path to reach the destination switchbox is the shortest. Since the estimated path length (the Manhattan distance to the destination) is a lower bound on the actual path length, this is a consequence of the guided search techniques of Hart, Nilsson, and Raphael. To see this, recall that the algorithm always extends the path whose estimated distance (distance travelled plus Manhattan distance to the destination) is minimum. When a path reaches the destination switchbox, its length is no longer an estimate: it is exact (there cannot be any cells blocking its path to the destination). All the other points on the heap have estimates at least as great as this, and their estimates are lower bounds on the exact lengths. Thus they cannot end up with shorter path lengths than the one that has already reached the destination switchbox.

Crossing locations are chosen by the shortest-path generator and later modified by the crossing placer. In the shortest-path stage of global routing, each new crossing is made as close as possible to the previous crossing in the path, subject to the availability of crossing points. This approach puts off additional wire length as long as possible and guarantees that the path length estimates are lower bounds. At a later stage of global routing the crossing locations will be reconsidered in order to avoid obstacles, eliminate jogs, or otherwise simplify the switchbox routing problems.

5.4. Crossing Placement

Magic's global router is unusual in that it does crossing placement during global routing. Other routers assign crossings after all global routing is complete, either by letting a channel router assign them or by executing a separate crossing-placement step before channel routing. Magic's mechanism has advantages over each of these other approaches.

The most common approach to crossing placement is to let a channel router assign crossings. When a given channel is routed, all of its crossings are fixed by the routing of the channel. The router is free to place unassigned crossings anywhere along the relevant channel border, but it must accept any crossings fixed while routing previous channels. This approach has the advantage of leaving the channel router maximum flexibility to put

crossings in the most convenient place. Also, in many situations the channels can be routed in order so that channels have fixed crossings only on their sides and never on their ends. In Magic, the router must handle switchboxes, with fixed crossings on all four sides.

A disadvantage of choosing crossings during channel routing is that it tends to simplify the problem at hand without considering the solution's global ramifications. A channel router may choose a crossing point that is convenient for the channel being routed but extremely inconvenient for the channel on the other side of the crossing; once the crossing is fixed, it will have to be used when the adjacent channel is routed. For example, if the channel router chooses a crossing adjacent to a large obstacle, it may be difficult or impossible to route the signal around the obstacle in the next channel.

The PI system [Rivest] and the VTI Composition Editor [Ng] perform crossing placement as a separate step between global routing and channel routing. They use global information during their crossing placement steps to adjust the crossings on the paths established during global routing. The aim is to maximize the number of crossings that face each other across switchbox edges so that the switchbox router will not need to insert jogs.

A problem with this approach is that the sequence of switchboxes traversed by each net is fixed before crossing placement occurs. When actual crossing locations are established, it may turn out that many of the crossings are undesirable because of obstacles. The router will not be able to consider alternative global routing paths to avoid the undesirable crossings, so it may have to use some of the bad crossings. This may make switchbox routing difficult or impossible.

Magic's crossing placement scheme avoids the problems with the above approaches. By assigning crossings during global routing, the characteristics of adjacent switchboxes can be considered in order to choose a crossing that makes each of the switchbox-routing problems as simple as possible. Since the global router repeatedly generates paths and adjusts crossings until the lowest-cost path is found, it can consider alternate paths during the global routing for a particular net.

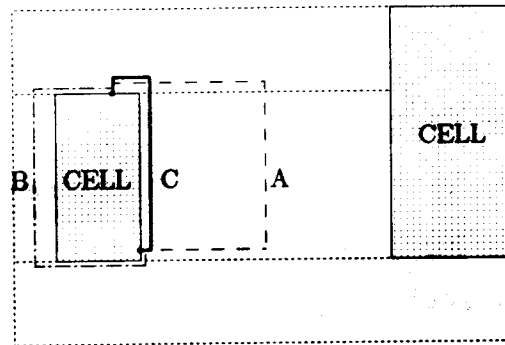


Figure 5.5. If net lengths are estimated using the centers of switchbox boundaries, unnecessarily long paths may be chosen. In this example, path B appears to be the shortest path if distances are measured using boundary centers. However, path C, which Magic will use, is actually the shortest route between the two points.

The Magic approach has other advantages over the above approaches. Since it knows where nets cross switchbox boundaries, it can make better estimates of path length. During global routing, the PI system assumes that nets all cross at the centers of switchbox boundaries. It can result in large overestimates of path length, as shown in Figure 5.5. [Ng] reports that this technique produces poor global routes.

In addition to avoiding obstacles, another of the goals in crossing placement is to permit straight-across routing in switchboxes by choosing crossings that face each other. In order to make as many facing crossings as possible, each crossing must be chosen with knowledge about the next crossing in the net. This information is not available until the shortest-path step is complete (when extending a path to a new crossing, the next crossing's location is not known). Thus, crossing placement cannot be done effectively during the shortest-path step of global routing.

Therefore, crossing placement is a two-step operation. During shortest-path generation, initial crossings are chosen to minimize net length. Once the complete path is established for a net, the router makes an additional scan over the crossings in the path. Each crossing is reconsidered in light of nearby obstacles, the locations of the preceding and following crossings in the net, and other information that is described in Section 5.5. If the crossing is undesirable, it may be replaced by a nearby crossing that is more

desirable, even if the other crossing results in a longer path length. Thus, crossing placement involves a tradeoff between net length and the difficulty of the resulting switchbox routing problems.

When reconsidering a crossing, the router first computes a penalty for the initial crossing, with the penalty expressed in units of distance. This penalty gives the maximum amount of additional net length it is worth incurring to avoid the problems at the crossing. If the penalty is not zero, then the global router examines crossings on either side of the initial crossing. Penalties are computed for each of these crossings, which also include the additional distance that will have to be travelled to use the crossings, and the lowest-cost crossing is chosen to replace the initial crossing. The penalty for the initial crossing limits how far to either side the router needs to look to find the best crossing. Figure 5.6 outlines the algorithm.

5.5. The Penalty Function

The global router's penalty function uses a number of heuristics to estimate the effect of a particular crossing upon the overall routing problem. The penalty function consists of a collection of rules, each of which assesses a distance penalty for a particular class of undesirable features at crossing points. Larger penalties are assessed for features likely to have the greatest impact upon the routability of specific switchboxes. Casting routing complexity into units of distance allows the global router to make tradeoffs between different combinations of length and complexity, and also different types of complexity, as it selects crossing points between switchboxes.

Since the crossing placement algorithm uses the penalty at its initial crossing to define the limits of its search for a lower cost crossing, each crossing penalty may be regarded as specifying how far to each side of the original crossing to look in an attempt to find a better crossing. The penalty function for a particular crossing considers a number of features described in the subsections below: obstacles, jogs, crossing density, and proximity to the corner of a switchbox.

```

1. crossing = closest pin;
2. penalty = evaluate(crossing);
3. if (penalty == 0)
    done; /* Nothing else could be better */
4. best so far = crossing;
5. current penalty = penalty;
6. for (dist = 1; penalty ≤ jog_penalty(dist); dist += 1)
7. { if(Save_Best(crossing - dist) == 0) done;
8.   if(Save_Best(crossing + dist) == 0) done;
9.   }

9. Save_Best(crossing)
10. { new penalty = evaluate(crossing);
11.   if (new_penalty < current penalty)
12.   { best so far = crossing;
13.     if (new_penalty == 0)
14.       return(0); /* Done. Nothing else can be better */
15.     current penalty = new_penalty;
16.   }
17. }
18. return(current penalty);
19. }

```

Figure 5.6: The crossing placement algorithm uses the crossing penalty at its initial crossing to define the limits of its search for a lower cost crossing.

Although the general framework is sound, the relative weights of the heuristics, as well as the rules themselves, are tentative. The specific rules and penalty values are likely to change with more experience with the system.

5.5.1. Obstacles

Several rules are concerned with avoiding obstacles. Obstacles come in two forms: double-layer and single-layer. Double-layer obstacles occupy both of the routing layers. There is no way to route any additional wires over those areas, so the router must avoid them completely. Single-layer obstacles occupy one of the routing layers; the router can still run wires over single-layer obstacles if it uses the other routing layer. However, the router normally expects to be able to use both routing layers for different signals, one horizontal and one vertical, so single-layer obstacles restrict the router's options: it can only

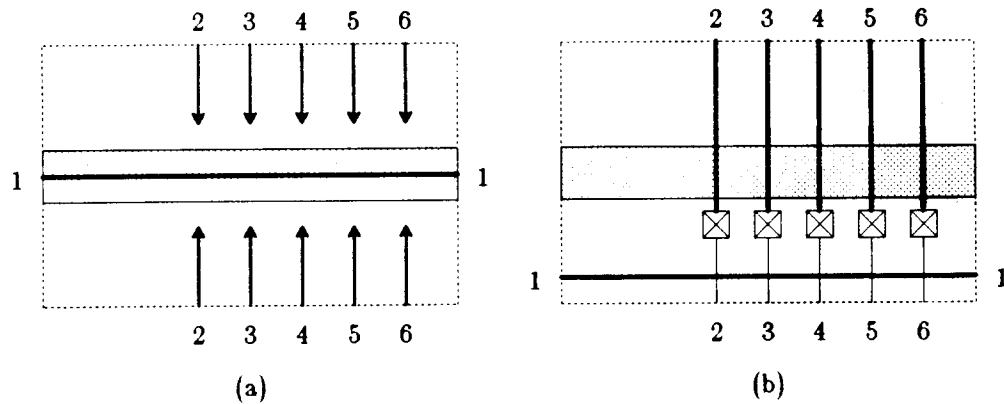


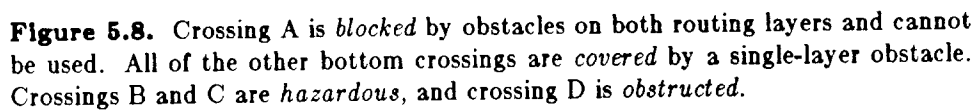
Figure 5.7. Wherever possible, the routing tools attempt to route across single-layer obstacles in the narrow direction. In this example, if net 1 were routed across the length of the obstacle, as in (a), all of nets 2, 3, 4, 5, and 6 would be blocked. By routing 1 around the obstacle, the other nets can cross the obstacle in the narrow direction and all the nets can be routed.

route in a single direction over them.

When choosing which way to route over single-layer obstacles the global and switch-box routers try to route across the narrowest dimension of the obstacle, as shown in Figure 5.7. This creates the least amount of two-layer blockage. If a signal would have to cross an obstacle along its long dimension, the routing tools try instead to jog the signal around the side of the obstacle so that other signals can bridge both the jogged signal and the obstacle.

The global router has four rules that deal with obstacles, illustrated in Figure 5.8. The first rule concerns crossing points that are covered by two-layer obstacles. These are called *blocked* crossings and cannot be used at all. They receive an infinitely high penalty.

The second rule deals with *covered* crossings. These are crossings that lie underneath single-layer obstacles. The goal is to penalize the crossing a lot if a wire connecting to that crossing would pass over the long dimension of the obstacle, and to use a smaller penalty if the wire would pass across the narrow dimension of the obstacle. The rule assesses a penalty equal to 3 times the size of the obstacle in the direction of the wire. Among covered crossings, this favors those covered by the narrowest obstacles.



Finally, a crossing point is termed *obstructed* if a straight run through that point would cross an obstacle. Such a crossing may cause a two-layer blocked area to be formed. These crossings are simply assigned a penalty of 5.

Another of the most important goals of crossing placement is to allow straight-across routing through switchboxes wherever possible. This involves three rules. The first rule penalizes crossings that are not facing. If a signal passes from one side of a switchbox to

the opposite side, the global router attempts to use crossings that are directly opposite each other. The penalty function assesses a fixed penalty of 5 to a crossing if it is on the opposite side of the switchbox from the previous crossing but is not directly across from the previous crossing.

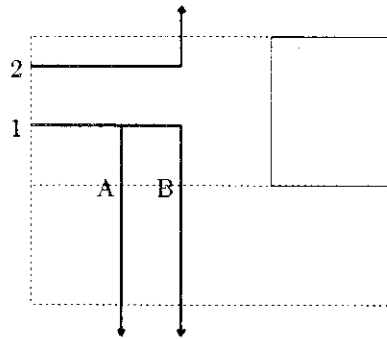


Figure 5.9. When choosing a crossing for net 1, the penalty function will penalize crossing point A, since it splits up a pair of facing crossings. Since the crossing opposite B is already in use by net 2, there is no penalty for using B.

The second jog rule preserves facing pairs of crossings when making turns. If a net enters a switchbox through one side and turns to leave the switchbox through an adjacent side rather than the facing side, then the router attempts to use crossings whose opposite numbers are already in use. See Figure 5.9. If a facing pair of crossings is broken up by a turning net, a penalty of 3 units is assessed.

The third jog-elimination rule is also applied when a net turns within a switchbox. When this occurs, the penalty function looks ahead at the next switchbox. If the next switchbox has a straight-through run, the router attempts to enter that switchbox at a crossing-point whose facing crossing is available, so that there won't be any need to jog within the switchbox. Any crossing whose opposite number is busy is assessed a penalty of 3 units. See Figure 5.10.

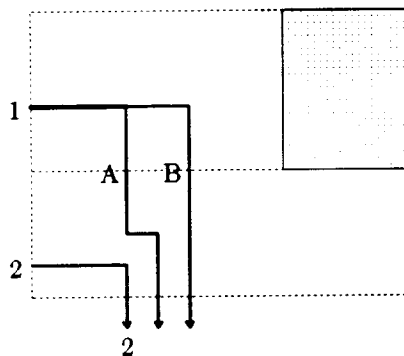


Figure 5.10. When net 1 makes a turn in the top switchbox, the penalty function will look ahead to the next switchbox. If the net runs straight through that switchbox, it will attempt to find a crossing whose opposite number is still available, as is the case with crossing B. Crossing A will be penalized since it would require a jog in the lower switchbox.

5.5.3. Pin Density and Corners

Magic's switchbox router can produce better routing if the occupied crossings are spread out over the length of the switchbox, rather than bunched together. For example, if only every other crossing is used then the router can virtually guarantee good routing. For this reason, the penalty function contains a rule that penalizes a crossing 5 units if both adjacent crossings are already in use, and 3 units if only one of the adjacent crossings is in use. This rule is only useful when a switchbox is undercommitted.

Crossing points near the corners of switchboxes also cause problems for Magic's switchbox router. For example, switchbox connections are harder to make if vertical wiring space in the last few columns must be used to make top and bottom connections. Therefore, a crossing that is the n th closest to the end of the switchbox is assessed a penalty of $5-n$ units. Only the five crossings at each end of the switchbox are penalized in this way.

5.6. Net Ordering

The global router processes shorter nets first. Before path generation, it sorts the nets according to the sum of the height and width of the bounding boxes containing their terminals. The height plus width of the bounding box containing a net's terminals approximates the length of the wires needed to connect the terminals. For two-pin nets (the most typical case), this number represents the Manhattan distance between pins.

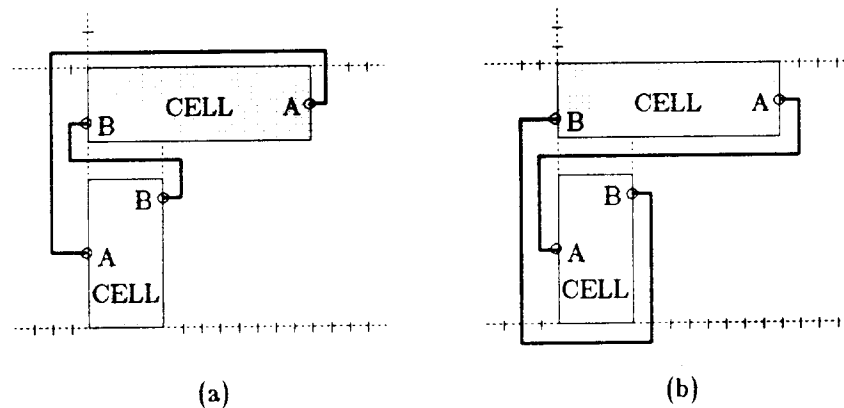


Figure 5.11. Shortest nets should be routed first, as in (a). If net A is routed first, the result is increased overall wire length (b).

This routing order for nets can have beneficial effects on the resulting wiring. Congestion created early during global routing may cause later nets to take less direct global routing paths; the same sized detour is a smaller percent increase for a longer net than for a shorter net. Shorter nets are less likely to interfere with each other; since longer nets cover more area, they are likely to interfere with many other nets. Designers may group cells during placement to reduce interconnection delays for critical paths; by routing the shortest nets first, the global router gives preferential treatment to these paths. The shorter nets will also get first choice of crossing points, while longer nets will use whichever crossings are available at the time they are routed.

The global router provides indirect support for user-defined net orderings. If designers want to give priority to arbitrary nets, they are free to pre-wire them in any

manner and then invoke Magic to make the remaining connections. This pre-wiring can be done by hand, with special-purpose routing tools, or even using Magic's automatic routing system.

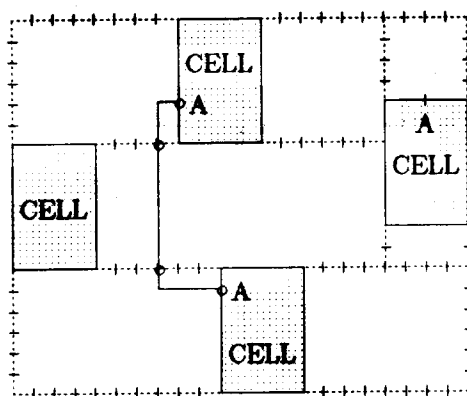


Figure 5.12. After the first two terminals of net A are connected, those terminals and all of the crossing points are used as starting points to connect to the third terminal in the net.

5.7. Multi-Pin Nets

Although the shortest-path algorithm described in Figure 5.4 dealt only with two-pin nets, it extends naturally to handle nets with three or more pins. The resulting algorithm generates a crude approximation to Steiner trees. If a net has more than two pins, the global router picks any two of them at random and applies the algorithm described above to generate a path from the source to the destination. After crossings have been assigned along this path, the algorithm then processes the third pin. That pin is used as the destination for the shortest-path algorithm. However, instead of searching from a single source, the shortest-path algorithm uses the previous pins, plus all of the crossings that have been assigned to this net, as starting points (see Figure 5.12). Each of these points is inserted in the heap with a zero cost, and the algorithm proceeds exactly as before: a shortest path will be found from one of these points to the new pin, and crossings will be assigned along that path. If there are still more pins in the net, the algorithm processes each of them in the same fashion, using all previous pins and crossings as starting points.

5.8. Equivalent Terminals and Feed-Throughs

The shortest-path algorithm also extends naturally to handle electrically-equivalent terminals. If two or more terminals on a subcell connect to the same node within the subcell, the global router connects to the subcell using whichever of the alternatives is the most convenient. Further, after making the initial connection to one of the electrically-equivalent terminals, a net can use the subcell's internal wiring and run out through another electrically-equivalent terminal to make additional connections for the net, reducing the amount of new wiring needed to connect the net.

The global router handles electrically-equivalent terminals by separately determining the lowest-cost path to each of the equivalent terminals. It uses the least-expensive path to any of these terminals. Each of the equivalent terminals is then added to the list of starting points used to connect to any additional pins for the net.

With electrically-equivalent terminals it is possible for the current global router to fail to find a path where one exists. This is because the global router starts off by picking two terminals at random from a net, but the only path connecting the terminals might be through electrically-equivalent terminals for that net. This problem can easily be solved by using a more generalized subcell feed-through mechanism that takes advantage of electrically-equivalent terminals.

Although not implemented, it takes very little additional work to extend the global router to use generalized subcell feed-throughs not associated with any particular net. This is done with a simple modification to step 7 of the algorithm of Figure 5.4. Instead of looking only at switchboxes adjacent to the current switchbox, the shortest-path algorithm must also consider subcells adjacent to the current switchbox. If the subcell contains an unassigned feedthrough terminal bordering on the current switchbox, then the shortest-path algorithm propagates through the feed-through terminal to all of its electrically-equivalent pins. Each of these points is added to the search point heap.

5.9. Loop Prevention

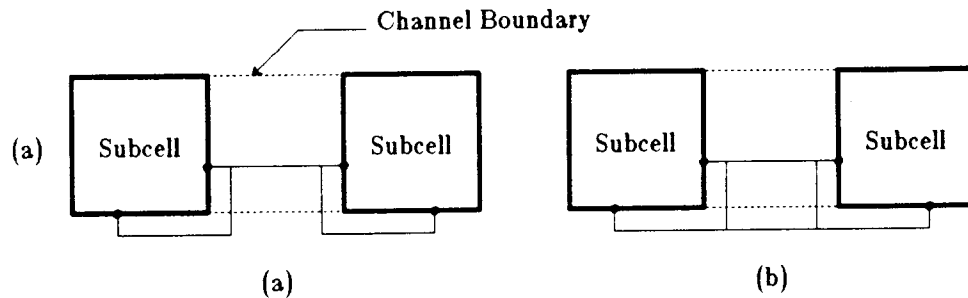


Figure 5.13. Different parts of a single net may need to pass through a switchbox without being connected within that switchbox (a). If this detail is not considered, the resulting routing may contain unnecessary closed circuits, as in the switchbox below the two subcells in (b).

One significant detail missing from the initial implementation of the global router was allowing different parts of a single net to pass through a switchbox without being connected within that switchbox. The result of this omission was closed loops of routing (Figure 5.13).

To avoid this problem, the global router treats each point-to-point connection as a separate subnet. The crossings for a subnet are stored as net/subnet pairs within the switchboxes it crosses. When a switchbox is routed, each net/subnet combination is regarded as a unique net. This allows the router to avoid generating the closed loops when distinct branches of the same net must pass through a single switchbox.

5.10. Extensions

A number of extensions to the current global router have not yet been implemented. Some of them--density checks and feed-throughs--have already been mentioned. This section describes the remaining extensions.

One possible extension is to remove order-dependency. Currently the global routing results depend upon the order in which nets are enumerated. One common technique to deal with this is to first route each net without regard for the other nets, then decide which nets to move from congested areas. Another possibility is to use the congestion

information from an initial routing to assign penalties for crossing through congested areas.

Another possible extension is to allow designers to give priorities to various nets. This would allow the designer to control the order in which nets are processed by the global router. Important nets could be given higher priorities to allow them to choose the most direct routes to make their connections.

The global router processes each net separately. There is currently no way to bundle nets together and route them along the same path. Such a "bus router" could result in higher-quality routing. It could reduce run times by obviating the need to route each of the nets in the bundle.

Since Magic currently uses a single switchbox router, the global router provides no support for river routing. If one were added, path generation and crossing placement would have to be modified to create switchboxes where the river router could be applied.

5.11. Evaluation

Experience with the global router is limited to a single Master's project designed by a graduate student at U. C. Berkeley. All of the results presented in this section are based on this one design. A thorough evaluation of the system will have to wait until there are more and larger test cases available. In particular, there is not yet enough information to evaluate the specific penalties attached to each of the rules. Consequently, these results are preliminary and only indicative of the general performance of the system.

The following results are based on an 8000 transistor custom NMOS chip called "Tester". Magic's router was used to make the global interconnections for Tester, consisting of 153 nets and 351 terminals. Hand-placed power and ground routing formed obstacles for the global router to work around.

Crossing placement effectiveness was measured by comparing results with and without crossing placement. When crossing placement was used the subsequent routing produced no bad connections; however, with crossing placement disabled the routing

produced 14 bad connections.

Penalty Disabled	Bad Connections	Paths Considered
None (normal)	0	289
All (no penalties)	14	198
All Obstacle Penalties	13	269
Straight Through Current	2	260
Straight Through Next	1	271
Neighboring Pins	1	269
Avoid Corners	1	276
Paired Orphans	0	272

Figure 5.14. Results of selectively disabling individual components of the crossing penalty function. The difference between the number of paths considered when all penalties are used and the number of paths considered when a penalty is disabled indicates the additional amount of work the global router does as a result of that penalty.

While processing Tester, the global router took advantage of its ability to select slightly longer paths in order to avoid areas with bad crossings. Of the 198 point-to-point connections, 151 could be made two or more different ways. Of these, 13 (8.6 %) connections used lowest-cost paths that were better than their initial shortest paths. Crossing placement was useful even when the global router used the initial shortest path to connect a net. On the Tester chip the global router moved 42% of crossings from their initial locations to reduce penalties and simplify switchbox routing.

The effectiveness of the individual components of the crossing penalty function was measured by selectively disabling them. The results are summarized in Figure 5.14. For the Tester chip, the obstacle penalties were the most useful of the various crossing penalties. With obstacle penalties disabled, Magic's router generates 13 bad connections while routing Tester. Of course, the obstacle penalties are only important in designs with hand-routing; when the hand-routing was removed from Tester, Magic had no trouble routing it even with no crossing penalties.

The other penalties had much smaller effects on Tester's routability. In particular, the second jog rule (Figure 5.9) had no effect on the number of bad connections. Since it is intended to make "tight" switchboxes easier to route, this rule should be more significant on chips with less free area than Tester.

Path Length	Nets	Cumulative % Nets	Points (Average)
1	75	38	1.00
2	37	57	2.14
3	19	66	3.37
4	20	76	6.50
5	16	84	8.69
6	15	92	9.53
7	8	96	11.38
8	4	98	37.50
9	2	99	49.50
10	2	100	26.50

Figure 5.15. The number of search points examined versus path length. For example, 15 nets had a path length of 6, and 92 percent of all nets had path length 6 or less. On average, paths of length 6 expanded 9.53 search points, or (9.53 - 6 =) 3.53 unnecessary search points.

The shortest-path algorithm appears to be working well. It was instrumented to count the total number of search points expanded during shortest-path searching (each extension of an existing path into new switchboxes counts as an expansion), and to count the number of switchboxes in the shortest paths. On average, each initial shortest path (crossing penalties cause the global router to generate additional paths) crossed 1.94 switchbox boundaries, and on average 5.17 points were expanded per shortest path found (Figure 5.15). Since each switchbox boundary represents one search point expansion and the search origin represents an additional mandatory search point expansion, this indicates that the shortest path algorithm typically examines only 2 search points not on the actual shortest path.

5.12. Summary

There are three key aspects to the Magic global router. The first aspect is that it combines crossing placement with global routing. This allows the global router to choose more circuitous paths to avoid particularly bad areas of the layout. Initial results indicate that the router does indeed find situations where it is better to choose a longer route. The second key aspect is the use of a rule-based penalty function. By evaluating the crossings with a set of rules, the global router can apply many different criteria in evaluating

crossings. As a result, the Magic crossing placer performs jog elimination as in the PI and VTI systems, and also applies different rules to keep wires away from obstacles and the ends of switchboxes. The third key aspect of the system is its use of a directed shortest-path algorithm, which tends to find the shortest path with very low overhead.

Experience with the penalty function is limited, and the specific numbers in use right now are fairly arbitrary. More experience is needed to understand the relative importance of the various penalties, and to extend the rule set with more knowledge about good and bad crossings. Nonetheless, even with this initial rule set crossing placement during global routing has been shown to be useful for effective obstacle-avoiding routing.

CHAPTER 6

Detour--Magic's Obstacle-Avoiding Switchbox Router

6.1. Introduction

Detour is a fast obstacle-avoiding switchbox router developed as part of Magic's routing system. Its goal is to support the routing system's obstacle avoidance capabilities by providing high-quality routing in switchboxes containing obstacles. This obstacle avoidance capability gives designers the option of prewiring special nets such as clock lines, Vdd, and GND, either by hand or using special-purpose routers. Detour routes around this wiring to complete the remaining connections.

Detour provides the same obstacle-avoidance capability as maze routers such as the Lee and Hightower routers (Chapter 2), which can also avoid obstacles on multiple layers. The problem with maze routers is that they consider only one net at a time. Since they completely route a single net before considering the next net, they can not consider interactions between nets as a channel is routed. Consequently, early nets tend to block later ones. For this reason these routers are inferior to true channel routers for channel routing [Soukup].

Detour's novel aspect is its ability to both avoid obstacles and consider interactions between nets. Since Detour is based on a channel-routing approach, it is able to consider the interactions between nets and thereby prevent some nets from blocking others.

This feature enables Detour to produce better routing than maze routers while also avoiding obstacles. In switchboxes or portions of switchboxes without obstacles there is no penalty: Detour produces results comparable to traditional channel routers. In areas containing obstacles, the router either jogs around the obstacles, or it switches layers and river routes across them. Detour thus combines good features from net-at-a-time routers and traditional channel routers.

Fast response time is important in an interactive design environment such as Magic's. Consequently, it is critical that the router run quickly, even at some expense in routing quality. Although it is a switchbox router and is capable of avoiding obstacles, Detour still routes large channels such as Deutsch's difficult example in less than 4 seconds, with total wire length and number of tracks within 5 percent of the best conventional channel routers.

Section 6.2 discusses the column sweep approach to channel routing, popularized by the "Greedy" router and adopted by Detour. Section 6.3 examines the problem of obstacle avoidance and develops a general strategy for dealing with obstacles. Section 6.4 explains Detour's techniques for routing switchboxes. Section 6.5 describes how *hazards* are used to implement both the obstacle-avoiding and switchbox strategies. Section 6.6 presents details of the obstacle avoidance algorithm. Section 6.7 describes a post processing step to increase metal and reduce vias. The chapter concludes with a discussion of the router's implementation and performance.

6.2. The Column Sweep Approach

Detour is an extension of Rivest and Fiduccia's "Greedy" channel router [Rivest and Fiduccia]. The original Greedy router was written in Lisp; Detour was independently written in C and enhanced to solve two difficult problems the Greedy router does not address: obstacle-avoidance and switchbox routing.

Like the original Greedy router, Detour uses a rule based, column sweep approach to channel routing. This section summarizes the column sweep approach and the specific wiring rules used by the original Greedy router. Subsequent sections describe and motivate the extensions to the Greedy router to avoid obstacles and to handle switchbox routing problems.

In the column sweep approach, the router works in a single left-to-right, column-by-column sweep across the channel (Figure 6.1). During the sweep it completely routes each column before proceeding to the next column. Routing a column consists of such things as

bringing nets into the channel from top and bottom pins, and jogging nets closer to their next top or bottom connection. When no more vertical wiring can be placed, the router extends still-active horizontal tracks into the next column, where the process is repeated. The router attempts to generate as much useful vertical wiring as possible in each column, hence the name "Greedy".

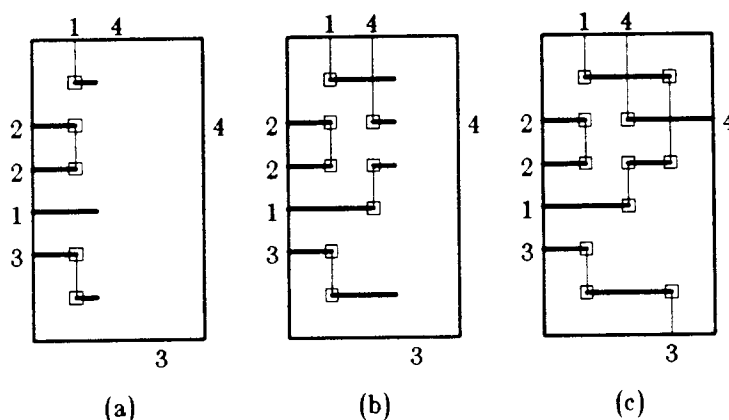


Figure 6.1. Three columns wired using the column sweep approach. In the first column (a) the router joins the two tracks assigned to net 2, brings net 1 into the channel, and jogs net 3 closer to its destination. In the second column (b) net 4 is assigned to the first vacant track, preventing the two tracks for net 1 from connecting; however, net 1's lower track moves closer to its upper track. In the last column (c) net 1's two tracks connect and net 3 makes its bottom connection.

Two features of the column sweep approach are of particular importance. First, unlike constraint-graph approaches, conflicting vertical wiring requirements are resolved by allowing nets to temporarily occupy more than one track at a time. These are called *split nets*. The ability to create split nets gives the router much of its flexibility, since it allows difficult wiring problems to be deferred to later columns where there is (hopefully) more room. Of course, split nets require extra tracks, so the router devotes considerable effort to collapsing them at the earliest opportunity.

The second important feature of the column sweep approach is its simple and straightforward control structure. Vertical wire placement within a column is controlled by an ordered set of rules. These rules are applied in order of importance to complete

crucial connections and to make subsequent columns easier to route. The router can easily be extended to achieve special effects such as obstacle avoidance, merely by adding new rules or changing the existing ones.

To ensure routability the router first takes those actions which if not done would cause it to fail. For example, it is essential to bring the nets of a column's top and bottom pins (if any) into the first available tracks that are either vacant or already assigned to the nets. If this step is not done immediately, vertical wiring might be placed so as to make it impossible to connect to the pins.

After taking care of essential tasks, the router generates vertical wiring to simplify the remaining routing problem. For the original Greedy router, this consists of collapsing split nets and "reducing the range" of split nets that could not be collapsed due to conflicts with other wiring.

A split net may be created by bringing a net into the first available track rather than into some other track already occupied by the net (net 1 in Figure 6.1). Split nets can fill up the available tracks, making it impossible to bring in additional nets; therefore, the router gives high priority to collapsing them. Since conflicting vertical wiring can make it impossible to collapse all split nets in a particular column, the router collapses split nets in the pattern that frees up the most empty tracks for use in the next column.

When not all of the split nets can be collapsed, the router simplifies the task of collapsing them in a later column by reducing the range of tracks occupied by these nets. It jogs each split net's highest occupied track downward and its lowest occupied track upwards. The remaining problems are easier because collapsing can be done with shorter vertical jogs which are less likely to conflict with other wiring.

Next, unsplit *rising* and *falling* nets are jogged upward or downward toward the edge of the channel with their next pin. This step anticipates the split nets that might be created when upcoming pins' nets are brought into the channel. It attempts to eliminate such split nets, or at least reduce their range.

The handling of split nets and rising and falling nets are examples of decisions based on interactions between nets. Among conflicting alternatives (a jog to raise a rising net may block a jog to lower a falling net) the router chooses the one that does the best job of simplifying the remaining overall problem.

6.3. Obstacle Avoidance Strategy

The original Greedy router assumes it can always extend active nets' tracks into the next column. It also assumes it can place vertical wiring wherever it is not blocked by vertical wiring it has previously placed. It runs horizontal wiring on one layer, and vertical wiring on the other layer.

When obstacles are present in the routing area, these assumptions no longer hold. Areas may be blocked (obstacles on both routing layers) or obstructed (obstacles on one routing layer), as explained in Section 3.2. Routing across a blocked area is impossible -- the router has to jog around it. As the router sweeps from left to right, it must move all nets from the blocked tracks before it reaches the blocked area. The router must also refrain from placing any column wiring in the blocked area.

Obstructions offer two alternatives to the router. One option is to jog around them as for blocked areas. However, since the obstruction only occupies one layer, the router can use the other layer to route either the track or column over the obstruction. In order to cross obstructions, the Detour router relaxes the strict layer-per-direction wiring model to allow routing in either direction to temporarily switch from the preferred layer to the other layer if that is necessary for it to bridge an obstruction.

One of the most important issues in obstacle avoidance is deciding whether to bridge an obstruction with column wiring or track wiring. If a track is allowed to bridge over an obstacle, then there is no way for columns also to pass over the obstacle since both routing layers are now blocked: the columns will all have to jog around the end of the obstacle. Similarly, if the obstacle is bridged by a column, the tracks will have to jog around. As mentioned in Section 3.2, it is generally easier to jog tracks around obstacles that are

very wide and around obstacles that are very tall.

The column-sweep approach used in Detour makes it easier to focus on the tracks, rather than the columns. By the time any given column is reached, it is too late to worry about where the tracks are, so they must be planned in advance. Before routing, Detour decides for each obstruction whether to bridge over it with the track or the column. If the track is to jog out of the way, Detour arranges to vacate the track before the obstacle is reached. No explicit action is taken to jog vertical wiring, except to avoid placing it in areas where both routing layers are blocked. If a track has not jogged out of the way before an obstruction is reached, this means that either it was supposed to bridge the obstacle or it was unable to get out of the way in time; in either case it will bridge over the obstacle, and there will be no place to put vertical wiring.

6.4. Switchbox Strategy

Detour handles routing problems with pins on all four sides. Furthermore, it allows nets to have arbitrary numbers of pins on each edge of the routing area. To route switchboxes Detour needs two new strategies, one to allow nets to move into the tracks they need to make end connections, and another to allow nets to split in order to make multiple connections at the right edge of the switchbox.

Detour extends the original Greedy router's notion of rising and falling nets to include not just top and bottom connections but also pins at the right edge of the switchbox. This makes nets gravitate towards the tracks they need for their end connections; however, the problem is complicated by the possibility of cyclic conflicts. For example, net 2 can occupy a track needed by net 1, while net 1 occupies a track needed by net 2.

The router resolves these conflicts by implementing a strategy that first tries to move rising and falling nets into tracks they need to make their own end connections, then tries to move them out of tracks needed by other nets to make their end connections. Initially no tracks are marked as needed to make end connections. When an unsplit net has passed its last pin on the top or bottom of the switchbox, it is ready to get into the track

it needs to make its end connection. Its final track is marked as needed by the net to make an end connection, and any net currently occupying that track attempts to jog out of the track.

If a net has more than one pin on the right edge of the switchbox, the router splits the net to connect to these pins. Since split nets occupy tracks that could otherwise be used to help route the switchbox, splitting to make multiple end connections is only done when the router gets close to the end of the switchbox. A parameter controls the number of columns from the end of the switchbox where net splitting begins. A typical value is four columns.

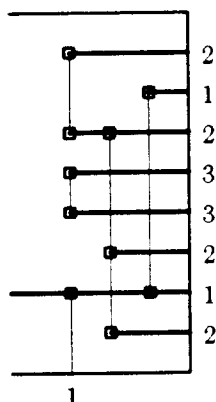


Figure 6.2. As the router approaches the end of the switchbox, nets with more than one of their pins on the right edge must be split to make those connections. If a net has pins only on the right edge (nets 2 and 3), the net must in some column be assigned to tracks for the first time. To avoid generating dead-end "stubs" of wire, this is done only if at least two tracks can be allocated and joined with vertical wiring. Note that the upper two tracks for net 2 are assigned earlier than the lower two tracks; this is because in the column where the upper two tracks are assigned, nets 1 and 3 prevent either of the lower tracks from connecting to any other track for net 2.

Nets with all of their pins on the right edge of the switchbox are another complication. As the router nears the right edge it has to decide when first to assign tracks to these *right edge* nets. Since there are no connections to previous pins, a right edge net is introduced only if it can be assigned to at least two tracks that can be joined by vertical wiring (Figure 6.2). Assigning a right edge net to a single track would be pointless, since

this would create a stub that doesn't connect to anything.

Going one step further, groups of two or more tracks for a particular right edge net may be introduced, even if the groups themselves can not immediately be joined. The task of joining these groups is easier, since the top track of one group need only be connected to the bottom track of a net's higher group.

6.5. Hazards

The strategies for obstacle avoidance and switchbox connections are both implemented using the *hazard* mechanism presented in Chapter 3. A hazard area is a location that nets should try to avoid. As the router extends tracks from left to right into a hazard, it attempts to make vertical jogs to vacate the hazards.

6.5.1. Hazards for Obstacle Avoidance

Hazards are generated to the left of all blocked areas, since it isn't possible to bridge them in either layer. Long, horizontally-oriented single-layer obstacles also generate hazards. This causes nets to jog vertically around such obstacles rather than route across them. Single-layer obstacles extending horizontally for only one column's width do not generate hazards since the vertical wiring gained in the obstructed area is more than offset by the vertical wiring wasted in jogging around the obstacle.

Depending on the height of the obstacle, many nets may have to be jogged around it. Since vertical wiring for one vacating jog may block other nets, not all nets can make vacating jogs in the same column. On the other hand, vacating tracks long before they near obstacles wastes routing area. In recognition of this, the distance at which Detour begins making vacating jogs around an obstacle depends on how high the obstacle is. Taller obstacles, which block more tracks, generate wider hazards, while shorter obstacles generate narrower hazards (Figure 6.3). The width of the hazard is the product of a parameter, *obstacle threshold constant*, and the height of the obstacle. This parameter allows some control over how soon the router attempts to vacate obstructed tracks. A

typical value for this parameter is 1.

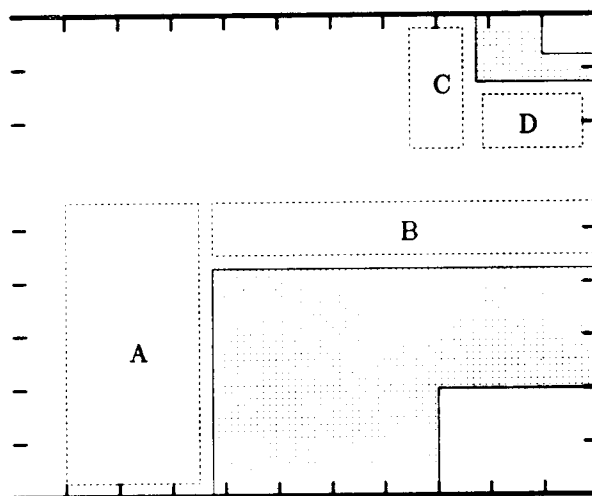


Figure 6.3. Taller obstacles may obstruct more nets, which require more columns to vacate a hazard; therefore, taller obstacles generate hazards (A and C) that are both wider and taller. Hazards (B and D) are also placed above and below obstacles which are on the vertical wiring layer, to leave space for contacts needed to bridge the obstacles with vertical runs.

If horizontal wiring were to run immediately adjacent to an obstacle blocking the preferred vertical routing layer, it would block all of the columns, since there would be no place to put a contact between the track running in one routing layer and the obstacle running in the other routing layer. Therefore, hazards extend one track above and below obstacles on the preferred vertical routing layer. This prevents horizontal wiring from running too closely to such obstacles (Figure 6.3).

6.5.2. Hazards for End Connections

The switchbox connection strategy is implemented with a slightly different form of hazards. Here rather than defining areas that are hazards for all nets, Detour defines areas that are hazards for all nets except the particular net that needs the track to make an end connection. See Figure 6.4.

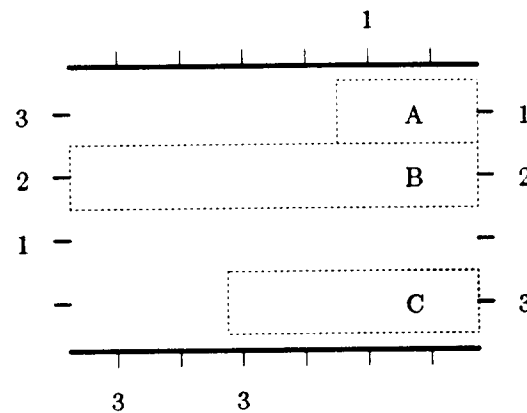


Figure 6.4. The areas marked with dotted lines are hazards for all nets except the specific nets making connections at the end of the switchbox. For example, area A is a hazard for all nets except for net 1. Tracks with end connections become hazardous for other nets at the column of the last top or bottom connection for the net making the end connection; thus, area B extends the length of the switchbox, while areas A and C do not.

6.6. Revised Vertical Wiring Rules

This section presents the complete set of rules the Detour router uses to control the placement of contacts and vertical jogs. All of the rules are modified to avoid placing vertical wiring in areas that are blocked, either by a two-layer obstacle or by a track bridging an obstacle. The rules are:

- a. *Place Track Contacts:* As the first step in wiring a column, place a contact in each unobstructed utilized track, if either the next column or the previous column has an obstruction in the preferred horizontal layer. The contact serves one of three purposes: (a) it switches the net from the preferred horizontal layer (metal) to the alternate layer (poly) when the net enters an obstructed region; (b) it switches the net from the alternate layer back to the preferred horizontal layer when the net leaves an obstructed region; or (c) it switches the track to the preferred vertical layer in preparation for jogging the net to another track.
- b. *Make Minimal Top and Bottom Connections:* A minimal top or bottom connection brings a net from a top or bottom pin into the nearest vacant track. Do not bring a net into a track that is blocked in the next column. This step may bring a net into a

hazard. If this occurs, step (c) will attempt to jog the net out of the hazard. Report failure if some net could not be brought into a track.

- c. *Vacate Tracks from Hazards*: Find horizontal wiring that is in a hazard, and try to jog it to the nearest empty track that is outside the hazard. Do not vacate to another track within a hazard unless the source track is blocked (ie. runs into a multi-layer obstacle) and the destination track is not blocked. Give preference to vacating jogs that move rising and falling nets closer to their next pin.
- d. *Collapse Split Nets*: Connect nets that occupy multiple tracks. Some combinations of connections may be mutually-exclusive; choose a pattern that creates the most empty tracks for use in the next column.
- e. *Reduce the Range of Tracks Assigned to Split Nets*: If a split net can not be collapsed, try to jog its highest track downward and its lowest track upward, to make it easier to collapse in some subsequent column. Make no jogs shorter than *minimum-jog-length* (a parameter of the router). Do not move a net into a hazard; however, it is okay to move a net from one hazard to another.
- f. *Raise Rising Nets and Lower Falling Nets*: Move unsplit rising and falling nets toward the edge containing their next pin. Enumerate nets in order of decreasing distance from their track to their "target edge". Make no jogs shorter than *minimum-jog-length*. Do not move a net into a hazard; however, it is okay to move a net from one hazard to another.
- g. *Split Nets to Make Multiple End Connections*: If within *channel end constant* (a parameter of the router) columns of the end of the switchbox, split nets onto multiple tracks to make multiple connections at the end of the switchbox. This is the opposite of the collapsing step (d) above. The best pattern is the one that splits the most tracks. If a net has pins only on the end of the switchbox, assign the net to vacant tracks, provided that at least two tracks can be assigned and connected.
- h. *Extend Active Tracks to the Next Column*: Report an error if some track is blocked in the next column.

6.7. Metal Maximization

After routing, Detour concludes with a metal maximization step. (Figure 6.5). Since the router already places metal horizontally wherever possible, this step replaces vertical wiring in polysilicon with vertical wiring in metal, subject to constraints imposed by wire crossings and obstacles in the switchbox.

Metal maximization improves routing quality by reducing the resistance of wires; for example, in one nMOS process the typical resistance for metal is 0.03 ohms/square, while the typical resistance for polysilicon is 15-100 ohms/square [Mead and Conway]. Metal also has lower capacitance; a typical capacitance value for metal is 0.3×10^{-4} pf/micron², while a typical capacitance value for poly is 0.4×10^{-4} pf/micron². Finally, metal maximization may often reduce the number of contacts, improving the circuit reliability.

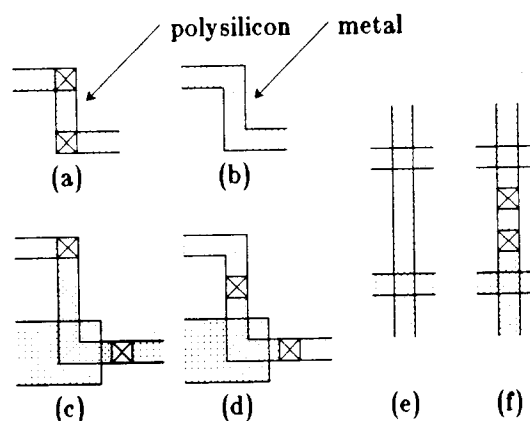


Figure 6.5. Layout before (a, c, and e) and after (b, d, and f) metal maximization. This step may add, delete or move vias.

Metal maximization is controlled by a user-supplied parameter specifying the cost of a via, VC . The Detour router will add a via to the routing (Figure 6.5f) if adding the via results in the conversion of at least VC wire lengths from polysilicon to metal. Specifying a large value for this parameter results in the insertion of no new vias; however, vias may still be moved (Figure 6.5d) or deleted (Figure 6.5b). Specifying 0 for this parameter allows the metal maximization step to convert polysilicon to metal wherever possible; since

the resistance of polysilicon is typically at least 500 times greater than for metal, small values for VC are reasonable.

Metal maximization only marginally improves the amount of metal in dense routing. Since almost all tracks are occupied in all places, most vertical polysilicon wires cross horizontal metal tracks; thus, the vertical polysilicon can not be converted to metal. In less dense areas the conversion of long vertical polysilicon runs to metal can be quite dramatic.

6.8. Rotation

Detour exhibits a directional bias. In some cases the router will fail on a particular routing problem but succeed when the same problem is mirrored or rotated. This occurs because of the column-by-column nature of the routing algorithm. A different orientation causes the router to process nets in a different order. The routing algorithm takes into account the direction and distance to the next pin in each net; these also change when a switchbox is processed in a different direction.

Magic's routing system takes this directional bias into account when using the Detour router. If a switchbox can not be routed in the default orientation, the routing system reroutes it in each of its other orientations. It quits when it finds a successful routing; otherwise, it uses the routing with the lowest number of connection errors. In practice rerouting in different orientations has been a useful recourse.

6.9. Parameters

The router accepts a number of numeric parameters. These parameters--the minimum jog length, the steady net constant, the channel end distance, and the obstacle constant--have major effects upon the resulting routing. Parameter values represent various kinds of tradeoffs. Higher values for minimum jog length and steady net constant may produce routing with fewer unnecessary doglegs; however, higher values may prohibit the router from making some necessary jogs, and the router fails. Higher values for the end constant may make the router overly cautious about making end connections in

switchboxes, prematurely tying-up tracks for end connections when they are still needed to make top and bottom connections. Lower values may not allow enough columns before the end of the switchbox to make all of the end connections. Higher values for the obstacle constant make the router overly-cautious about skirting obstacles; lower values may cause the router to be unable to avoid some obstacles.

Magic's default routing parameter values were selected to produce better completion rates, at some expense in terms of extra vias or wire length. In practice Detour does well with the default parameters; however, in some cases different parameter values, ones that might be expected to reduce routability, do have a beneficial effect. For example, with the default parameters Detour routes Deutsch's difficult example in 21 tracks; with a higher value for the steady net constant, it requires only 20 tracks.

Like other routers based on Rivest's Greedy router, Detour's parameter selection is very primitive. Currently, designers may specify different parameter values before the routing process begins, but those parameters are applied to all of the routing. If the router fails, the router does not modify the parameters and retry, although in most cases the useful parameter space is quite small. It might be useful to be able to interactively modify parameters and reroute selected areas; this would allow rerouting to improve routing quality as well as completion rate. Parameter selection need not stop at the granularity of the switchbox; parameters could be adjusted during the routing of a single switchbox in response to local conditions within various spans of columns.

6.10. Results

Figures 6.6 and 6.7 illustrate both the power of the greedy approach and the power of obstacle-avoidance. These figures show the result of applying Detour to Burstein's difficult switchbox [Burstein and Pelavin]. The hierarchical router was unable to completely route this switchbox; Burstein and Pelavin even conjectured that the problem was unroutable. The Detour router by itself was also unable to completely route the switchbox; however, by examining the incomplete solution, I found a solution by hand. In order to give the router a "hint", I routed one critical net by hand, and then Detour was again

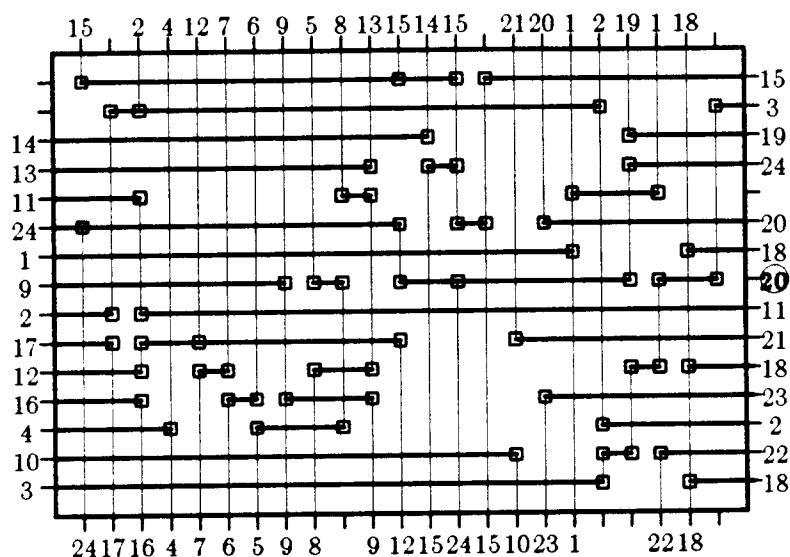


Figure 6.6. Partial routing of Burstein's difficult switchbox, routed by Detour. One pin for net 20 could not be connected.

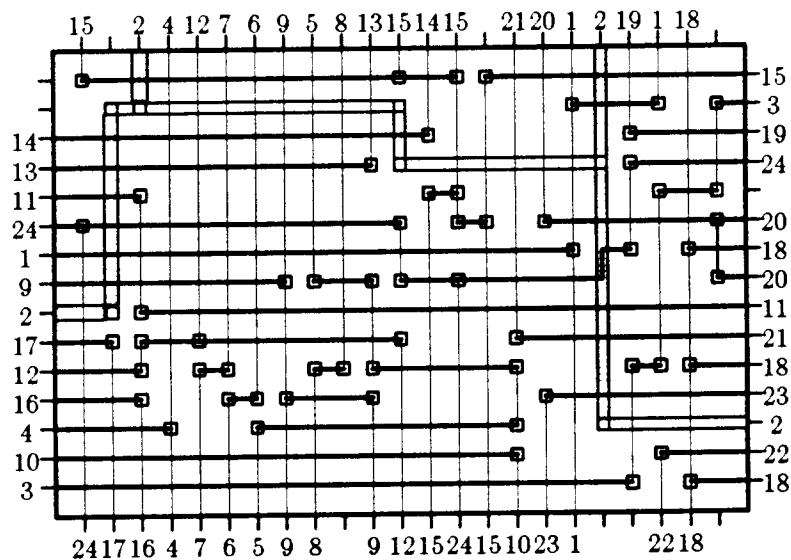
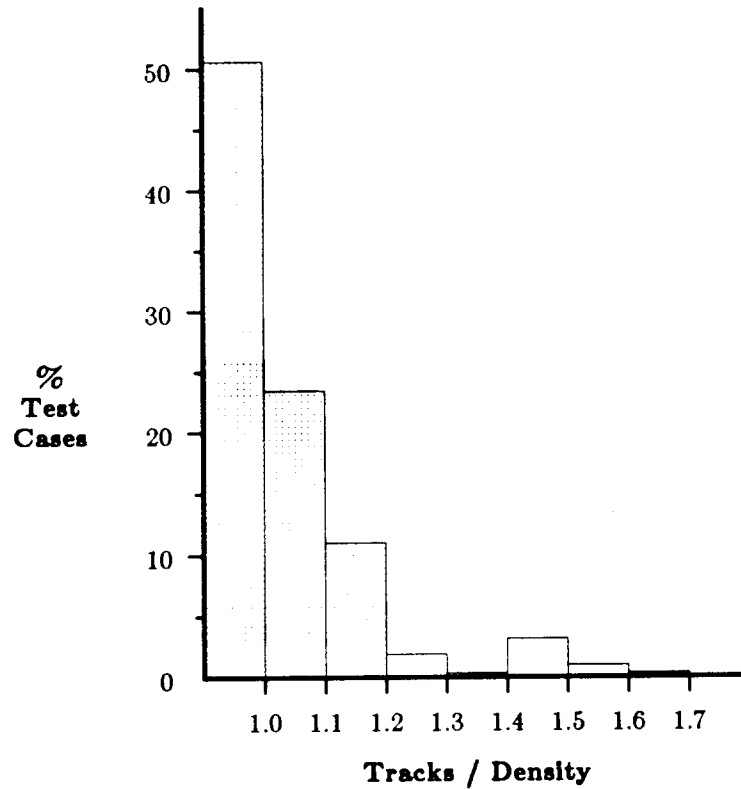


Figure 6.7. Complete routing of Burstein's difficult switchbox, routed by Detour. Net 2 was pre-wired by hand.

invoked. This time, it was able to complete the switchbox, while working around the hand routing (Figure 6.7).

In addition to illustrating the usefulness of partial hand-routing, this example also demonstrates the effectiveness of Detour's obstacle-avoidance techniques. Especially in such a difficult routing problem, one might expect obstacles to make the switchbox more difficult to route. Instead, Detour successfully routed the remaining nets around the obstacles, producing a fully-routed switchbox.



Tracks	Test Cases	Percent	Cumulative Percent
Optimal	86	50.89	50.89
Optimal + 1	44	26.04	76.92
Optimal + 2	18	10.65	87.57
Optimal + 3	7	4.14	91.72
Optimal + 4	2	1.18	92.90
Optimal + 6	1	0.59	93.49
Optimal + 9	1	0.59	94.08
Could not Route	10	5.92	100.00

Figure 6.8. Detour routed most of the 169 YACR test cases using an optimal or near-optimal number of tracks. The bar graph displays the number of tracks used as a fraction of the optimal number of tracks to route each test case, showing that most of the test cases were within 10 to 20 percent of optimal.

Test Case	Tracks	Test Case	Tracks
academic/11o	12	industrial/quali9.dat	13
academic/1b	2	stdcell/ex/ex2	15
academic/3c	18	stdcell/ex/ex3	11
academic/3cr	18	stdcell/ex1/route.in.1	12
academic/4	5	stdcell/ex1/route.in.12	22
academic/4r	5	stdcell/ex1/route.in.13	12
academic/4ra	5	stdcell/ex1/route.in.15	2
academic/5	5	stdcell/ex1/route.in.2	15
academic/chris2	50	stdcell/ex1/route.in.3	17
academic/chris3	64	stdcell/ex1/route.in.5	24
academic/error	1	stdcell/ex1/route.in.8	20
academic/ex2	15	stdcell/ex1/route.in.9	26
academic/ex3	11	stdcell/ex2/route.in.1	4
academic/nasty5.5	7	stdcell/ex2/route.in.11	10
academic/nasty7.7	9	stdcell/ex2/route.in.2	8
industrial/channel10.dat	26	stdcell/ex2/route.in.3	8
industrial/channel11.dat	22	stdcell/ex2/route.in.6	5
industrial/channel12.dat	22	stdcell/ex2/route.in.7	12
industrial/channel13.dat	19	stdcell/ex2/route.in.8	8
industrial/channel14.dat	23	stdcell/ex2/route.in.9	14
industrial/channel15.dat	21	stdcell/ex3/3c	18
industrial/channel16.dat	22	stdcell/ex4/route.in.1	8
industrial/channel17.dat	18	stdcell/ex4/route.in.3	17
industrial/channel18.dat	19	stdcell/ex4/route.in.4	10
industrial/channel19.dat	20	stdcell/ex5/route.in.1	10
industrial/channel20.dat	20	stdcell/ex5/route.in.11	32
industrial/channel21.dat	16	stdcell/ex5/route.in.18	13
industrial/channel22.dat	18	stdcell/ex5/route.in.19	12
industrial/channel23.dat	17	stdcell/ex5/route.in.2	13
industrial/channel24.dat	19	stdcell/ex5/route.in.3	16
industrial/channel5.dat	18	stdcell/ex5/route.in.5	17
industrial/channel6.dat	15	stdcell/ex5/route.in.9	27
industrial/channel7.dat	18	stdcell/ex6/route.in.1	2
industrial/channel8.dat	20	stdcell/ex6/route.in.10	7
industrial/channel9.dat	25	stdcell/ex6/route.in.11	7
industrial/ex2	15	stdcell/ex6/route.in.12	15
industrial/ex3	11	stdcell/ex6/route.in.3	10
industrial/quali10.dat	11	stdcell/ex6/route.in.4	9
industrial/quali11.dat	11	stdcell/ex6/route.in.5	9
industrial/quali5.dat	10	stdcell/ex6/route.in.7	9
industrial/quali6.dat	9	stdcell/ex6/route.in.8	8
industrial/quali7.dat	9	stdcell/ex6/route.in.9	6
industrial/quali8.dat	11	switchbox/t26	5

Figure 6.9. Detour routed these 86 of the 169 YACR test cases using the optimal number of tracks (density).

Test Case	Density	Tracks	Test Case	Density	Tracks
academic/0	6	7	academic/r4	15	17
academic/1	3	*	academic/r5	18	19
academic/11r	12	13	academic/yuck	2	8
academic/2	2	3	industrial/ex1	16	17
academic/2.a	2	*	industrial/ex4	19	21
academic/3a	15	18	stdcell/ex/ex1	16	17
academic/3b	17	20	stdcell/ex/ex4	19	21
academic/bad	4	6	stdcell/ex1/route.in.10	21	22
academic/burstein	4	6	stdcell/ex1/route.in.11	21	22
academic/c1	5	6	stdcell/ex1/route.in.14	13	14
academic/c2	4	*	stdcell/ex1/route.in.4	23	24
academic/c3	4	*	stdcell/ex1/route.in.6	38	39
academic/chris1	49	50	stdcell/ex1/route.in.7	25	26
academic/cycle.14	6	*	stdcell/ex2/route.in.10	5	6
academic/cycle.19	2	*	stdcell/ex2/route.in.4	7	8
academic/cycle.20	8	*	stdcell/ex2/route.in.5	11	12
academic/dd	19	20	stdcell/ex3/3a	15	18
academic/dd.rel	19	21	stdcell/ex3/3b	17	21
academic/dd2	32	33	stdcell/ex3/ddr	19	21
academic/dd2.fix	39	40	stdcell/ex4/route.in.2	17	18
academic/dd2.rel	32	34	stdcell/ex5/route.in.10	27	29
academic/dd3.fix	32	34	stdcell/ex5/route.in.12	32	33
academic/dd4.fix	35	36	stdcell/ex5/route.in.13	34	35
academic/dda	19	21	stdcell/ex5/route.in.14	29	30
academic/ddr	19	20	stdcell/ex5/route.in.15	25	26
academic/ex1	16	17	stdcell/ex5/route.in.16	25	26
academic/ex4	19	21	stdcell/ex5/route.in.17	20	21
academic/mit1	4	5	stdcell/ex5/route.in.4	16	17
academic/mit2	3	6	stdcell/ex5/route.in.6	24	25
academic/mit3	4	7	stdcell/ex5/route.in.7	25	26
academic/mit4	6	9	stdcell/ex5/route.in.8	22	23
academic/n1	20	21	stdcell/ex6/route.in.13	8	9
academic/n2	20	22	stdcell/ex6/route.in.2	10	11
academic/n3	16	17	stdcell/ex6/route.in.6	9	10
academic/n4	15	17	switchbox/t104	7	8
academic/nasty7.5	7	8	switchbox/t109	7	9
academic/nasty9.5	6	10	switchbox/t110	10	12
academic/nasty9.6	6	8	switchbox/t117	14	23
academic/nasty9.7	7	10	switchbox/t119	15	*
academic/r1	20	21	switchbox/t121	17	*
academic/r2	20	22	switchbox/t123	17	*
academic/r3	16	17			

Figure 6.10. Detour routed 83 of the 169 YACR test cases using a suboptimal number of tracks. An asterisk indicates that Detour was unable to route the test case.

Detour produces good routing in channels without obstacles. I applied Detour to 169 channels and switchboxes known as the YACR test cases, which were obtained from Alberto Sangiovanni-Vincentelli. The optimal number of tracks for the test cases ranged from 1 to 64 tracks. As summarized in Figure 6.8, Detour routed most of these test cases using an optimal or near-optimal number of tracks. Figure 6.9 lists those test cases that were routed optimally, while Figure 6.10 lists the results for those that could not be routed optimally.

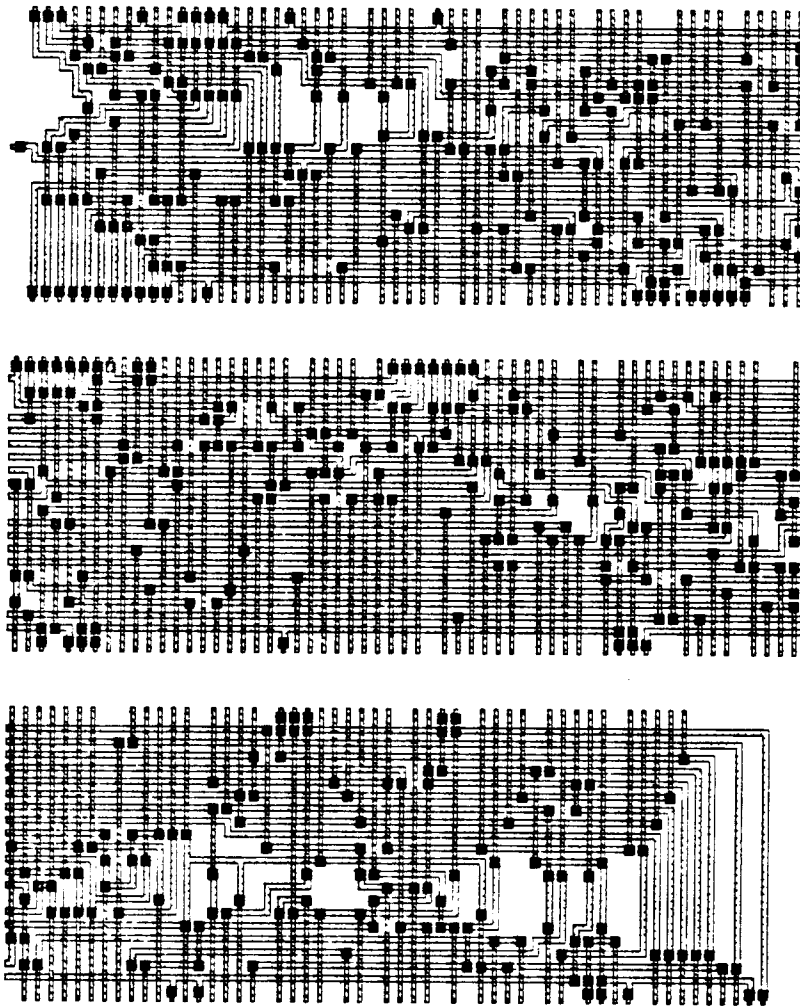


Figure 6.11. Deutsch's difficult example, as routed by Detour.

A more detailed comparison shows that Detour produces results similar to those produced by other good channel routers such as Yoshimura and Kuh's Algorithm #2

[Yoshimura and Kuh], YACR [Sangiovanni-Vincentelli], and Burstein and Pelavin's hierarchical router. Figure 6.11 shows the complete routing for the longer of the two versions of Deutsch's difficult example. The table in Figure 6.12 compares the Detour router to these other routers for this problem. All times are on a DEC VAX 11/780, except the Hierarchical router, which was timed on an IBM 370/3033.

Router	Tracks	Vias	Length	Time (sec)
Detour	20	412	5302	3.9
Y&K	20	308	5075	2.1
YACR	19	285	5060	6.1
B&P	19	336	5023	24

Figure 6.12. Router Comparison for Deutsch's Difficult Example

Figure 6.12 shows that Detour is competitive with other channel routers on Deutsch's difficult example. Detour is within approximately 5 percent of the other routers in total wire length and number of tracks; although it generates a significantly larger number of vias than the other routers, the effect of these vias upon chip yield is small.

Test Case	Optimal	Detour	YACR	Y & K
academic/r1	20	21	21	21
academic/r2	20	22	20	20
academic/r3	16	17	16	17
academic/r4	15	17	17	20
academic/3a	15	18	15	15
academic/3b	17	20	17	17
academic/3c	18	18	18	18

Figure 6.13. In comparisons on other test cases, Detour performed within one to three tracks of the best routers. The numbers for the other routers are from [Sangiovanni-Vincentelli 83b].

The results are similar on a wider range of test cases. Figure 6.13 compares Detour to YACR and Yoshimura & Kuh's router. Over all 7 of the test cases, Detour averaged 1.28 tracks per channel more than YACR, and 0.71 tracks per channel more than Yoshimura and Kuh.

Maximization	Vias	Metal Length	Percent Metal
None	471	2566	48.4
Unrestricted	467	2996	56.5
No new vias	412	2889	54.5

Figure 6.14. Metal Maximization for Deutsch's Difficult Example

Postprocessing to increase metal and remove vias significantly improves the quality of the routing (see Figure 6.14). The "unrestricted" line is when the router was allowed to insert additional contacts; for the "no new vias" data, the router was not permitted to introduce new vias. As an example of the range of problems handled by Detour, it can river-route the switchbox in Figure 6.15, which is completely covered with metal.

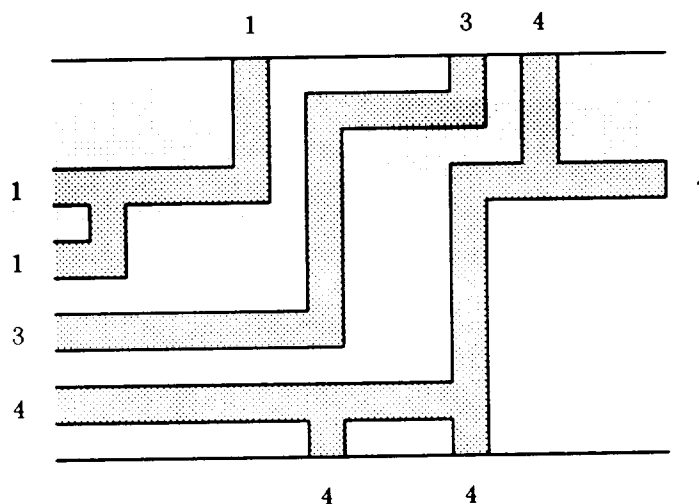


Figure 6.15. Detour river-routes in areas completely blocked in a single layer.

The router has proved to be sensitive to modifications to the routing rules, particularly for routing examples that are difficult to begin with. For example, after "improving" Rivest and Fiduccia's rules to take advantage of more vertical wiring (be more greedy), Detour would no longer route Deutsch's difficult example from left to right in 20 tracks--it took 21. After discovering this it could then route the same problem from right to left using only 20 tracks, whereas before it could not.

CHAPTER 7

Results

7.1. Introduction

Chapters 4-6 include measurements of the performance of the three major router components; this chapter supplements those results with measurements of the routing system as a whole. These measurements show that obstacle-avoiding routing is both advantageous and practical: hand-routing improves selected nets, while the hand-routed obstacles have little effect upon the routing quality for the design as a whole.

This chapter presents a number of measurements of Magic's router. It begins by comparing the lengths of carefully hand-routed nets with the lengths of the same nets when routed automatically, to show that the automatic router does a reasonable job of minimizing net lengths. These hand-routed nets show improved electrical characteristics resulting from reduced wire length, increased routing on the preferred routing layer, and fewer vias. The measurements also demonstrate the effectiveness of obstacle-avoiding routing, by showing that rather than hurting the quality of the subsequent automatic routing, partial hand-routing has a small and generally positive effect on the overall routing quality. Next, the results show that the router provides fast turnaround, as the time needed to hand-route a selected net and then reroute a chip is well under an hour. Small routing examples illustrate some of the fundamental obstacle-avoiding features. The chapter concludes with a discussion of code size and some areas for further work.

7.2. Hand-Routing Measurements

The first test of Magic's obstacle-avoiding routing techniques is a comparison of automatically-routed nets to hand-routed nets. This test shows that, while nets can be hand-routed with only small improvements in wire length compared to automatic routing,

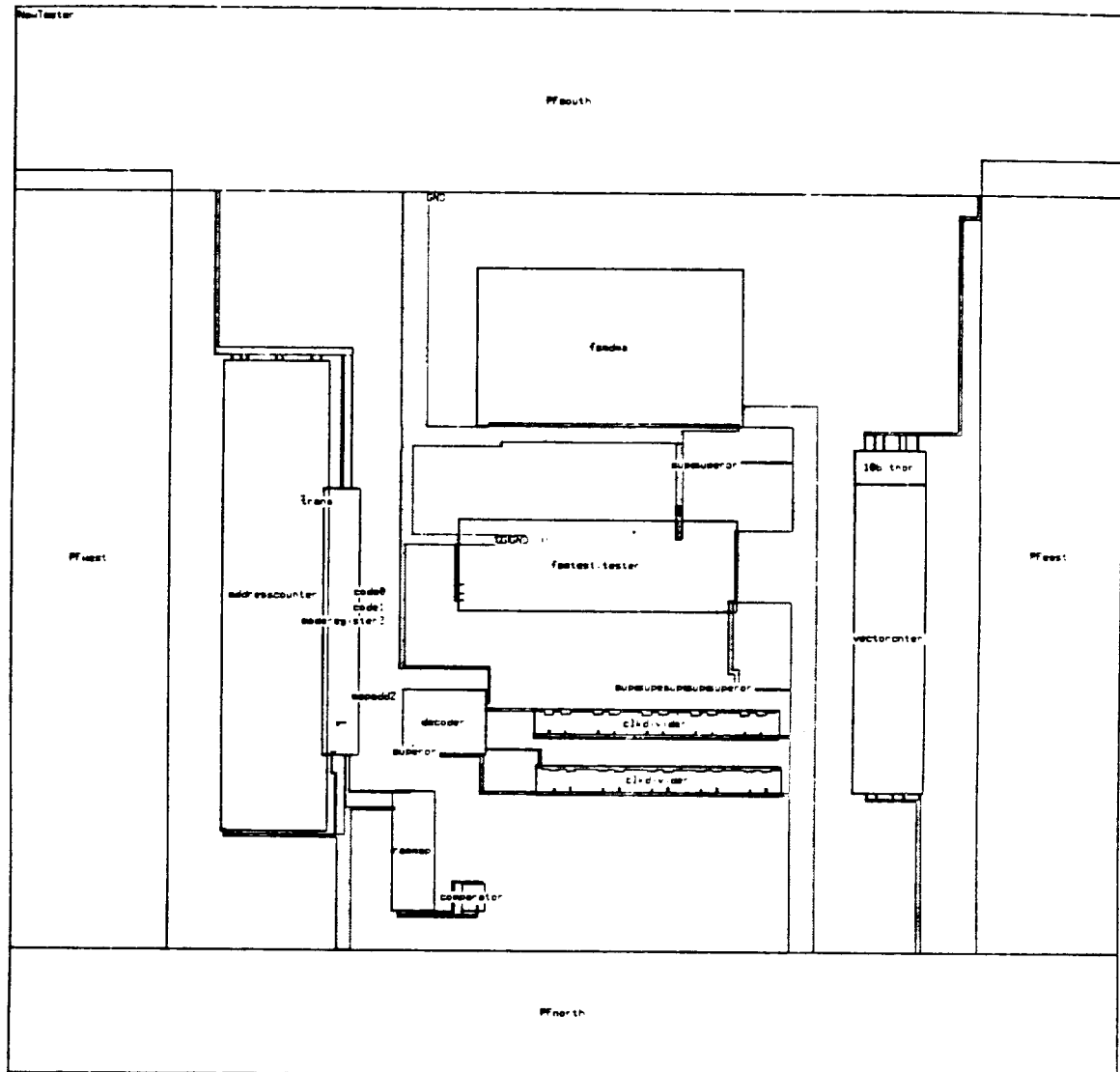


Figure 7.1. Comparisons of hand-routing versus automatic routing used the Tester chip, shown here before automatic routing. In addition to the hand-routing, Vdd and GND routing also formed obstacles for the router to work around. The completely-routed chip is shown in Figure 7.2.

significant improvements in electrical characteristics are typical.

The test corresponds to what a designer might do with a critical net: carefully route it by hand to reduce net length, resistance, and capacitance. The procedure was to route a chip using Magic's router, measure the length and electrical characteristics of one of the routed nets, delete all of the automatic routing, reroute the selected net by hand, and

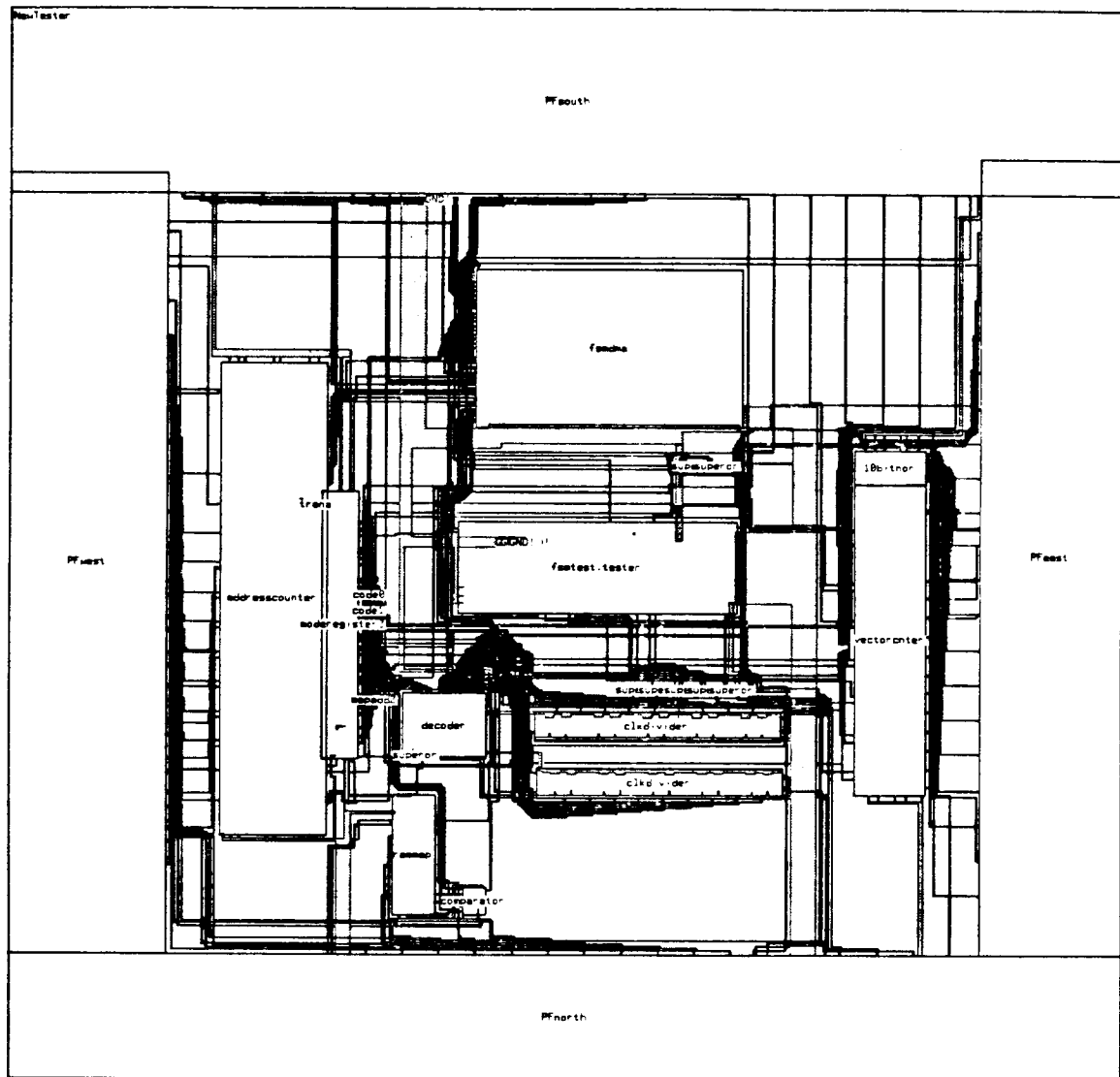


Figure 7.2. The Tester chip following automatic routing. Dense routing areas are above the decoder, to the right of the moderegister, and between the fsmdma and the large metal GND line.

then reroute the rest of the nets automatically. The hand-routed net and the power and ground routing formed obstacles for the automatic router to route around.

This test used the Tester chip previously described in Chapter 5 and shown in Figures 7.1 and 7.2. Fifteen of that chip's nets were tested out of its 198 total nets. To take net length into account (it should be easier to improve upon longer nets than shorter nets), I measured the 5 longest nets, the 5 nets with median length, and the 5 shortest

nets. The following sections describe the results of the test: net length improvement, delay improvement, and effects on automatic routing.

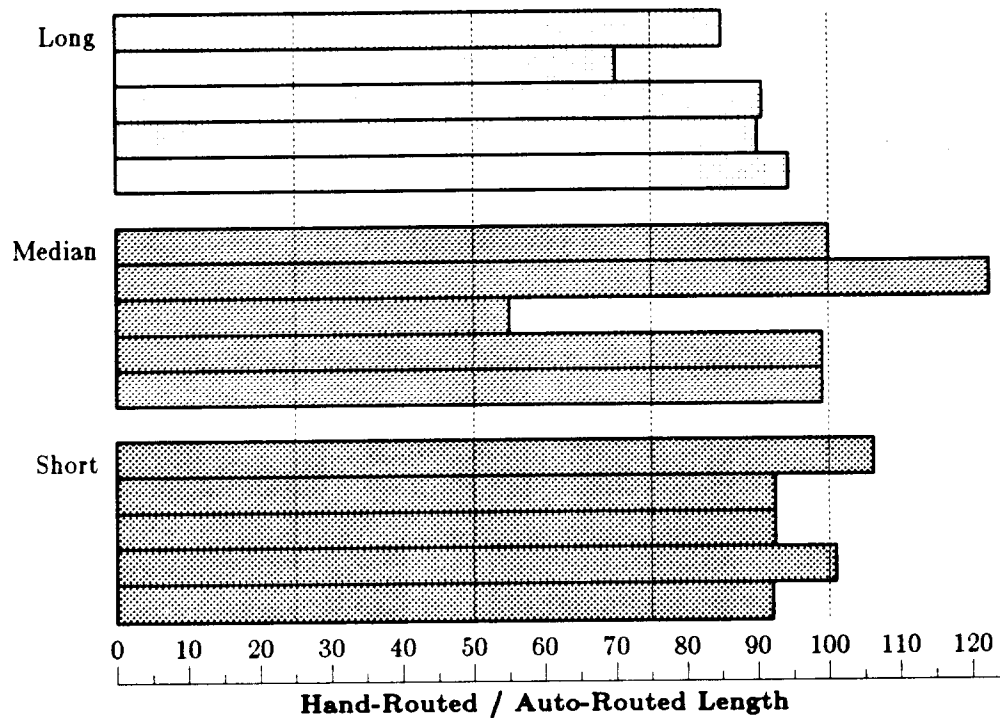
7.2.1. Net Length

The first measurement compares the length, percent metal, and number of vias in the hand-routing and the automatic routing for each of the 15 test nets. The table in Figure 7.3 summarizes the results, while the bar graph shows the ratio of the hand-routed length to the automatically-routed length for each net.

These results indicate that, while the router does a good job of finding shortest paths, designers can improve critical nets by hand-routing them. Twelve of the 15 nets could be hand-routed using less wire than was used by the automatic router; for these nets the hand-routing averaged 7 percent shorter than the corresponding automatic routing. In every case hand-routing improved the percentage of metal used to make the connection; the improvement ranged from 0.7% to 55.1%, with an average of 16%. The number of vias remained the same or was reduced in the hand-routing for each of the selected nets; in some cases the reduction was quite dramatic, ranging up to 100 percent.

The major reason for the length improvement of the hand-routed nets is that they are routed in advance of the automatic routing, so they do not have to route around the automatically-routed nets; in effect they have priority over the automatic routing. They can be assigned to the preferred routing layer, and avoid many of the vias that would otherwise be needed to route them across other wiring.

Three of the 15 nets could only be hand-routed using more wire than used by the automatic router. This occurred because the natural hand-routing paths for these nets cross highly-congested areas. Rather than route around the obstacles, the global router tried to reduce the lengths for other nets by placing them on top of the hand-routing. Because of this, the Detour router was forced to create two-level blocked areas completely blocking some of the nets. To avoid these problems, I hand-routed the nets around the congested areas, resulting in the increased net length.



Hand-Routed Net	Length		% Metal		Vias	
	Hand	Auto	Hand	Auto	Hand	Auto
addresscounter_0/a10	4470	5249	85.8	72.4	15	67
moderegister_0/regfor16bit_0[2]/\$9	4249	6070	94.1	75.6	11	55
PFnorth_0/\$41	4772	5249	92.1	72.4	11	67
PFeast_0/PADDATAINONLY_16[1]/\$1	4270	4736	94.5	78.4	11	45
vectorcncr_0/\$6	4145	4391	95.0	85.1	9	37
fsmdma_0/mack	1122	1123	89.6	86.0	3	7
decoder_0/output_3	1381	1115	97.5	87.1	4	8
clkdivider_1/\$10	611	1114	90.0	48.4	6	18
PFnorth_0/\$11	1103	1112	98.6	97.9	2	2
PFwest_0/PADADDRESSNOCOMP_12[1]/\$1	1049	1059	100	90.7	0	9
fsmdma_0/lnSt3*	124	115	96.8	41.7	1	5
comparator_0/\$2	97	105	67.0	52.4	4	4
rammap_0/\$33	97	104	67.0	59.6	4	4
comparator_0/\$4	98	97	66.3	56.7	4	4
moderegister_0/regfor16bit_0[0]\$2	63	68	92.1	82.4	1	1

Figure 7.3. In most cases, hand-routing improved the selected net's length, and in every case it improved the percentage of metal. The bar graph shows hand-routed net lengths as a percentage of the corresponding automatic routing. Nets in the bar graph and table are in three groups: *Long* refers to the 5 longest nets, *Median* refers to the 5 nets of median length, and *Short* refers to the 5 shortest nets on the test chip.

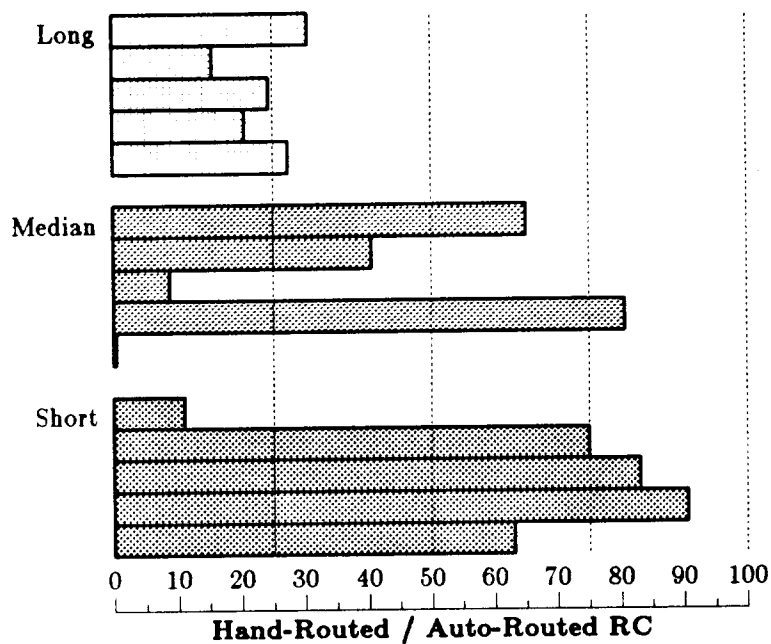
This problem is with the implementation rather than the general approach to obstacle-avoiding routing. The global router's penalty for selecting crossings already covered by hand-routing is too low, and should be increased. The switchbox router created blocked areas because of conflicts between wiring rules for avoiding hand-routing and rules for making switchbox connections; the rules can be modified to avoid the problem. I chose to hand-route around the problem simply to save the time necessary to make the changes and then redo the measurements.

7.2.2. Electrical Characteristics

From the viewpoint of critical-path improvement, a better measure of the effect of hand-routing is a comparison of the electrical characteristics of hand-routing with the electrical characteristics of automatic routing. Using the same nets as in Figure 7.3, I compared the resistance and capacitance of the hand-routing and automatic routing. These are summarized in Figure 7.4, which shows that hand-routing improves the RC product for the nets by a substantial margin, from 25 to 75 percent compared to the corresponding automatic routing. Figure 7.4a shows the routing's electrical characteristics assuming the electrical parameters for an NMOS technology, while Figure 7.4b shows the same figures for the MOSIS scalable CMOS technology.

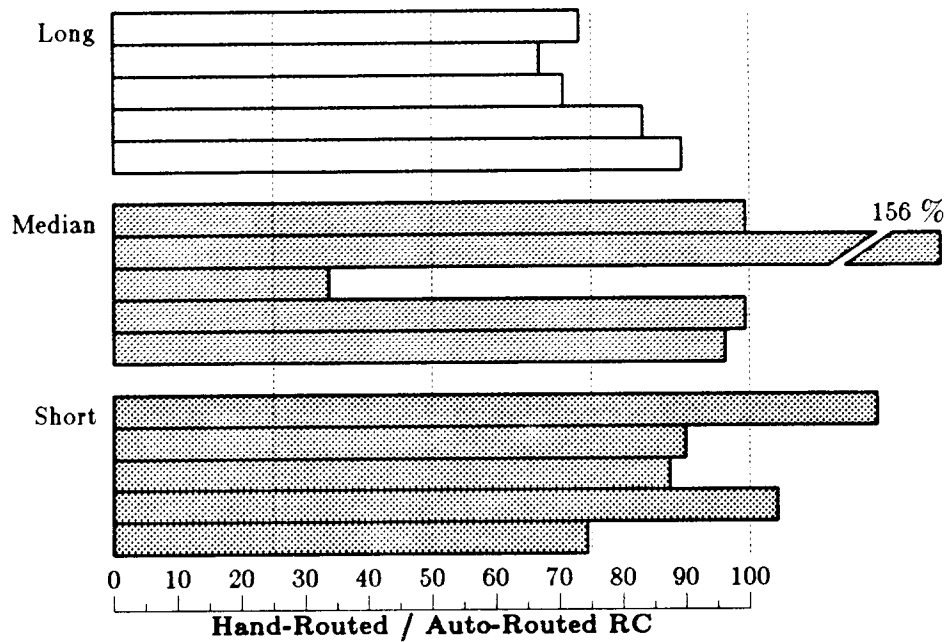
For the NMOS technology, hand-routing improved the electrical characteristics of each of the nets tested. The improvements ranged up to 75 percent for the longest nets, and generally less for shorter nets. Much of the RC reduction was due to better layer selection, since the selected nets could be hand-routed largely in metal and with little polysilicon.

For the CMOS technology the improvements are less dramatic because the two routing layers (metal1 and metal2) have nearly identical routing characteristics. In this case the results resemble the net length reduction results, with the same 3 hand-routed nets worse than the automatic routing, but with a general improvement overall. The degree of improvement is greater than just the net length improvement, since the designer hand-routing the nets can still minimize vias.



Hand-Routed Net	Resistance		Capacitance		RC x 10 ⁻¹²	
	Hand	Auto	Hand	Auto	Hand	Auto
addresscounter_0/a10	5843	15580	1.60	1.92	9367	29875
moderegister_0/reg-for16bit_0[2]/\$9	2677	13497	1.53	1.91	4106	25743
PFnorth_0/\$41	3656	12537	1.72	2.07	6277	25925
PFeast_0/PADDATA-INONLY_16[1]/\$1	2557	10844	1.54	1.72	3944	18671
vectorcenter_0/\$6	2214	7539	1.50	1.60	3311	12078
fsmdma_0/mack	1097	1677	0.40	0.41	442	682
decoder_0/output_3	548	1649	0.50	0.41	274	669
clkdivider_1/\$10	883	5577	0.23	0.40	198	2237
PFnorth_0/\$11	263	323	0.40	0.40	105	130
PFwest_0/PADADDRESS-NOCOMP_12[1]/\$1	10	1382	0.38	0.39	3	537
fsmdma_0/InSt3*	101	852	0.05	0.05	4	38
comparator_0/\$2	520	655	0.39	0.41	20	26
rammap_0/\$33	520	595	0.04	0.04	20	24
comparator_0/\$4	528	595	0.04	0.04	20	22
moderegister_0/reg-for16bit_0[0]/\$2	108	160	0.02	0.02	2	4

Figure 7.4a. Using the electrical parameters for a single-layer-metal NMOS process [Mead and Conway], hand-routing improved the electrical characteristics of all hand-routed nets. The parameters are: metal resistance 0.03 ohms per square, poly resistance 15 ohms per square, contact resistance 30 ohms, metal capacitance 0.3×10^{-4} pf/micron², poly capacitance 0.4×10^{-4} pf/micron², and contact capacitance 25.6×10^{-4} pf.



Hand-Routed Net	Resistance		Capacitance		RC x 10 ⁻¹²	
	Hand	Auto	Hand	Auto	Hand	Auto
addresscounter_0/a10	42.68	47.99	0.464	0.567	19.79	27.23
moderegister_0/reg- for16bit_0[2]/\$9	41.71	48.45	0.439	0.561	18.30	27.20
PFnorth_0/\$41	46.52	53.13	0.492	0.605	22.90	32.14
PFeast_0/PADDATA- INONLY_16[1]/\$1	41.98	44.18	0.441	0.505	18.50	22.29
vectorcnter_0/\$6	40.81	41.89	0.427	0.465	17.43	19.49
fsmdma_0/mack	10.85	10.74	0.116	0.118	1.26	1.27
decoder_0/output_3	13.71	10.71	0.143	0.117	1.96	1.26
clkdivider_1/\$10	5.93	9.33	0.065	0.112	0.39	1.14
PFnorth_0/\$11	10.99	11.05	0.113	0.114	1.25	1.26
PFwest_0/PADADDRESS- NOCOMP_12[1]/\$1	10.49	10.30	0.107	0.112	1.12	1.15
fsmdma_0/InSt3*	1.23	0.95	0.013	0.014	0.02	0.01
comparator_0/\$2	0.88	0.90	0.012	0.013	0.01	0.01
rammap_0/\$33	0.88	0.92	0.011	0.012	0.01	0.01
comparator_0/\$4	0.89	0.85	0.012	0.012	0.01	0.01
moderegister_0/reg- for16bit_0[0]/\$2	0.62	0.73	0.007	0.008	0.004	0.006

Figure 7.4b. Using a two-layer-metal CMOS process, hand-routing improved the electrical characteristics of 12 of the 15 hand-routed nets. These figures use the electrical parameters from the Magic SCMOSII technology file at U.C. Berkeley: metal1 resistance 0.03 ohms per square, metal2 resistance .027 ohms per square, contact resistance 0.03 ohms per square, metal1 capacitance 0.34×10^{-4} pf/micron², metal2 capacitance 0.26×10^{-4} pf/micron², and contact capacitance 8.5×10^{-4} pf.

7.2.3. Effect on Automatic Routing

While Magic's router avoids obstacles, the results show that it does so without a significant effect upon the overall routing quality for the entire design. This is a critical measure since, to be useful, hand-routing should improve the hand-routed net without having a severe impact on the automatic routing.

Hand-Routed Net	% Length Change	% Metal Change	% Via Change	Patches
addresscounter_0/a10	-1.5	-0.2	1.5	1
moderegister_0/regfor16bit_0[2]/\$9	-0.5	-0.3	5.2	0
PFnorth_0/\$41	-0.5	-0.5	1.4	0
PFeast_0/PADDATAINONLY_16[1]/\$1	-0.2	0	2.9	2
vectorcater_0/\$6	-0.7	-0.3	1.9	0
fsmdma_0/mack	0	-0.1	0.8	1
decoder_0/output_3	0	0.1	2.1	2
clkdivider_1/\$10	-0.4	0.2	0.6	1
PFnorth_0/\$11	-0.6	0.1	0.2	0
PFwest_0/PADADDRESS-NOCOMP_12[1]/\$1	0	0	-0.1	0
fsmdma_0/InSt3*	-0.1	0.4	-2.3	2
comparator_0/\$2	0	0	0	0
rammap_0/\$33	0	0	0	0
comparator_0/\$4	0	0	0.1	0
moderegister_0/regfor16bit_0[0]/\$2	0	0	0	0
Average for All Nets	-0.3	-0.04	0.9	0.6

Figure 7.5. Overall change in chip wiring characteristics, before and after hand-routing one net and auto-routing the rest. The numbers are for all nets, including the hand-routed net. Negative values for length change and via change indicate that the automatic routing produced better results after hand-routing than before. "Patches" indicates the number of incomplete nets that had to be hand-patched after automatic routing.

Figure 7.5 shows that hand-routing a single net has a small and usually beneficial effect on the total wire length and percent of metal wiring for the design as a whole. As might be expected, the automatic router had to use additional vias to bridge routing across the hand-placed obstacles; this increase was the largest for the longest nets, but averaged less than 1 percent increase over all 15 nets tested.

In spite of the router's obstacle-avoidance techniques, the automatic router was not always able to completely route designs containing hand-routing. When the only obstacles

were Tester's power and ground routing, the router was able to completely route the chip; however, when one net was hand-routed, in 6 of the 15 cases the hand-routed net caused the subsequent automatic routing to fail with one or two incomplete nets.

The primary reason for the router's failure to complete the routing was the lack of density calculations during global routing. The number of wires Magic will attempt to route through a switchbox is limited only by the number of crossing points at which nets can enter and exit; this is a fairly weak approximation to the switchbox's capacity. Further, there is no allowance for the negative effect of obstacles upon wire capacity. Thus, even after obstacles have been added, the router tried to run the same number of wires through a switchbox. The additional congestion caused incomplete nets that must be patched by hand.

In every case the incomplete nets could easily be hand-patched following the automatic routing. It was not necessary to reroute the entire net by hand; typically there was a problem within a single switchbox along a net's path. In some cases I patched the automatic routing by hand-routing one net on top of another for a short distance; Magic's switchbox router can cross obstacles, but it does not run one automatically-routed net on top of another. In other cases I routed nets within unused stem frame areas after looking inside the subcells to see that no design rules would be violated, or moved tracks closer together because they lacked adjacent contacts.

7.3. Time to Reroute

The router provides fast turnaround for a complete routing cycle that includes removing the old automatic routing, partial hand-routing, and automatic rerouting of an entire design. An initial goal was to be able to do this in less than a day [Ousterhout 84b]; measurements show that this takes less than an hour of real time.

As part of the hand-routing experiments in the previous section, I measured the time it took for one iteration through the hand-routing cycle on the Tester chip. Using the longest net in the design, the total time to run through the hand-routing cycle was 49

minutes. This included 7 minutes for the initial automatic routing, 2 minutes to rip-up the automatic routing, 23.5 minutes to hand-route the selected net, another 7 minutes to reroute the rest of the nets automatically, and 9.5 minutes to hand-patch one incomplete net. Routing the second-longest and third-longest nets took similar amounts of time, except that no hand-patching was necessary. All of these times were measured on a Sun-2 with 4 megabytes of memory.

Even using Magic's centerline wiring command, hand-routing proved to be very slow. Hand-routing the selected net took 48 percent of the total hand-routing cycle time, or more than three times as much time as was needed to automatically-routing the entire chip. If each of the 153 nets took the same amount of time to route by hand, it would take 7.5 8-hour days to completely route this chip by hand.

Hand-routing is probably even slower than this indicates. Since each hand-routed net forms an obstacle to other wires, each successive hand-routed net gets harder to route by hand. It might well take far longer than a week to hand-route the Tester chip.

7.4. Run Time Distribution

Figure 7.6 shows the router's distribution of run time over its various component parts, for two real designs. One is the Tester chip, with 153 nets and 338 terminals. The other is Memboard1, with 60 nets and 425 terminals (Figure 1.2).

Routing Phase	Tester		Memboard1	
	Time	%	Time	%
Channel Decomposition	0.28	0.1	0.75	0.3
Channel Initialization	32.48	15.0	24.72	10.9
Global Routing	43.87	20.3	52.08	23.0
Switchbox Routing	82.02	37.9	88.15	39.0
Paintback	57.58	26.6	60.35	26.7
Total	216.23	-	226.05	-

Figure 7.6. Distribution of run time over various parts of the router. All times are measured on a Sun 2 with 4 megabytes of memory and using a remote file server.

For each of the two designs the largest amount of time--38 to 39 percent--was spent in switchbox routing. The second most time-consuming part of the routing process was painting the routing back into Magic's corner-stitched database. This step includes metal maximization to convert polysilicon (or any other second routing layer with less-desirable electrical characteristics) into metal wherever possible. Global routing was the third most time-consuming part of routing. It was followed by channel initialization, which includes storage allocation and hazard generation. The least time-consuming part of routing was channel decomposition, which constituted less than 1 percent of the total routing time.

7.5. Small Examples

Figure 7.7 shows the effect of a simple obstacle upon the routing for a small example. Figure (a) shows how it is routed with no obstacles, while Figure (b) shows the effect of an obstacle upon the routing. It is impossible for four nets to avoid the obstacle, so the router simply places contacts, switches routing layers, and bridges the obstacle using the other routing layer. There are no other changes to the routing.

Although stem frames (Chapter 3.5) provide a simple mechanism to allow automatic alignment of terminals onto the routing grid, they also take up room that could be used to improve the routing quality. For example in Figure 7.7 the topmost connection for net 1 runs upward to clear the stem frame, and then left and down; however, it could have been routed horizontally through the stem frame to shorten the net's length.

Figure 7.8 shows the effect of a more complex obstacle upon the same example used in Figure 7.7. In Figure 7.8 nets 1 and 2 cross the obstacle by extending further to the left than in Figure 7.7, to avoid creating a long vertical strip with both routing layers blocked. If net 1 in Figure 7.8 had taken the same path as in Figure 7.7, it would have blocked the other net, preventing it from making its connection. Avoiding a long run across the obstacle also reduces the nets' parasitic capacitance with respect to the obstacle.

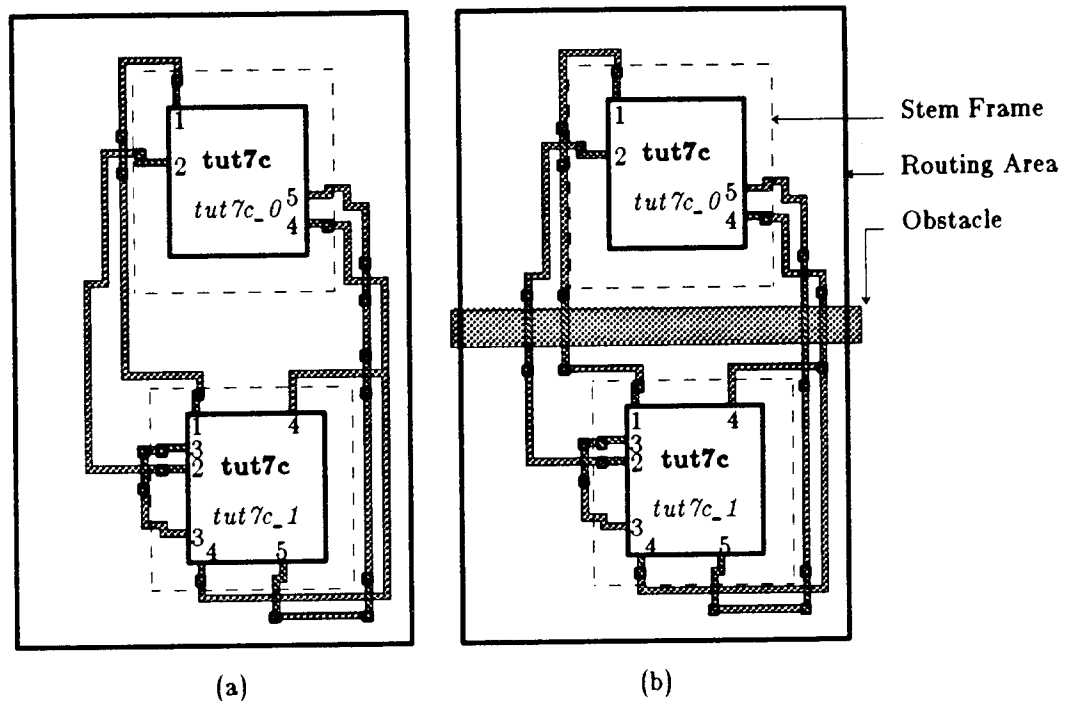


Figure 7.7. A small example routed by Magic. Figure (a) shows the routing when there are no obstacles. Figure (b) shows the effect of an obstacle passing through the routing area.

In Figure 7.8 (a) the routing area is limited, so it is impossible for the rightmost two nets (4 and 5) to clear the obstacle. They have to route across the obstacle, and because of the crossing ordering chosen by the global router, Magic could not completely route net 5. To avoid this problem when the routing area is expanded in that direction, the router chooses to avoid the obstacle by extending the nets further to the right (Figure 7.8 (b)).

7.8. Code Size

The router is written in C and runs under 4.3 Berkeley Unix. Its 18,716 lines of code constitute 16 percent of Magic's 117,039 lines, a substantial fraction indicative of the complexity of the routing problem. Approximately half of the source lines are comments or blank lines; when comments and blank lines are removed, the routing code is less than 10,000 lines.

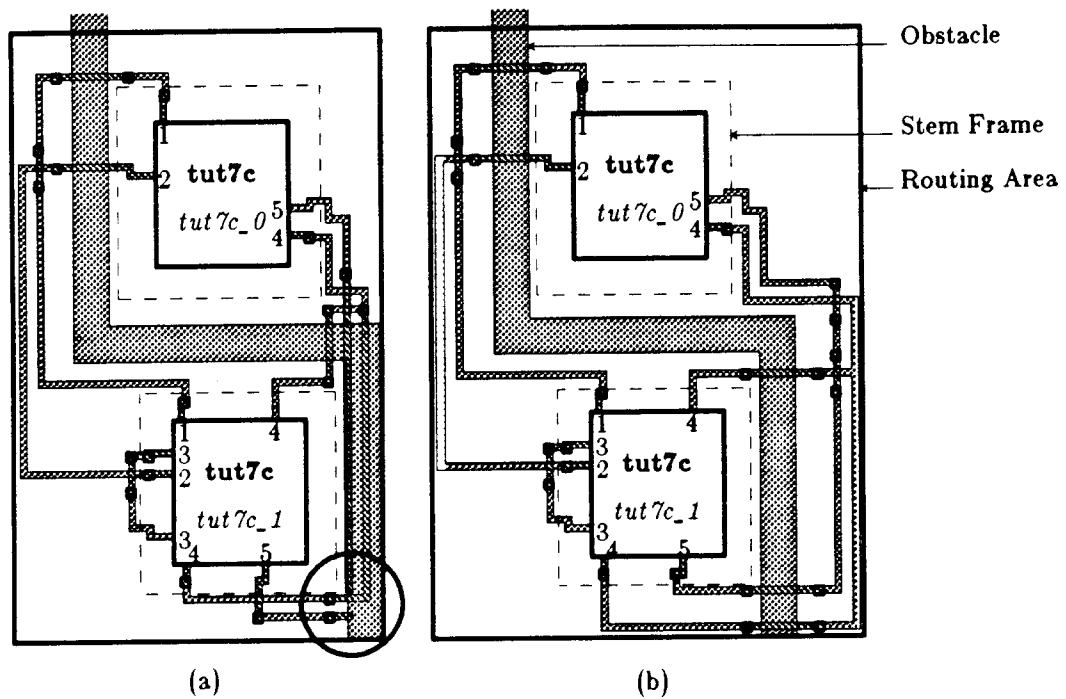


Figure 7.8. A more complex obstacle illustrates additional obstacle-avoidance techniques. When compared to Figure 7.7, nets 1 and 2 extend horizontally to the left to avoid crossing the obstacle along its long dimension. Net 5 is blocked in the lower right corner of (a) (circled area); when the routing area is expanded further to the right as in (b), nets 4 and 5 can extend further to the right and avoid crossing the obstacle along its long dimension.

The table in Figure 7.9 lists the sizes of the various parts of the router. Functions included under “miscellaneous” include channel initialization, feedback, stem generation, and paintback. Since the various parts of the router make use of Magic’s database, graphics, command-interpreter, and utilities routines, a stand-alone version of the router would be much larger than the total for all routing code.

7.7. Improvements

This section describes a number of possible improvements to the router. Some of these are straightforward additions to the current implementation, while others constitute areas for further research.

Router Module	Size (with comments)	Size (no comments)
Channel Decomposition	889	473
Global Router	2782	1259
Switchbox Router	4707	2930
Netlist Editing	6080	2872
Miscellaneous	4258	2242
All Routing Code	18716	9776
All of Magic	117039	54290

Figure 7.9. Code sizes, measured in number of lines, for each of the parts of the router. Approximately half of the lines in Magic source files are comments.

7.7.1. Technology Issues

Since many current IC fabrication processes provide three or more layers suitable for routing, it would be useful to modify the router to handle more than two routing layers. The greedy column-sweep approach used by Magic's Detour router is readily adaptable to additional layers; one 3-layer version has already appeared in the literature [Bruell and Sun]. Modifications to support obstacle-avoiding global routing with more than 2 routing layers would be more difficult, since the global router would have to pay more attention to layer assignments and obstacles on different layers.

Since the router assumes a uniform routing grid, there may be problems applying it to technologies where the two routing layers have widely differing pitches. If minimum-sized contacts are not square, then it is possible to gain additional tracks or columns by using different grid spacings in the x and y-directions; however, a grid-based obstacle-avoiding router cannot take advantage of this, since it needs to be able to switch layers to bridge obstacles. A non-gridded router would be more efficiently applicable over a wider range of technologies.

7.7.2. Global Router

The router's completion rate for fully-routed nets is quite sensitive to the placement of subcells and obstacles in a design. This occurs because the global router does not consider switchbox density, nor the effect of obstacles upon switchbox capacities. Because the global router does not check density, it can overcommit areas, creating routing problems

the switchbox router cannot successfully route. If density were considered during global routing, it would be possible to more evenly balance the load carried by each switchbox, and improve the completion rate for automatic routing.

7.7.3. Switchbox Routing

While providing great flexibility, Magic's switchbox router tends to introduce a large number of vias, unnecessary bends, and crossovers, when compared with hand-routing. Routing quality can be improved by the addition of more specialized channel routers, such as river-routers or more traditional channel routers, for those cases where the full power of an obstacle-avoiding switchbox router is not needed.

Additional techniques can improve the quality of the wiring. The router currently post-processes the wiring to convert polysilicon to metal wherever possible; this post-processing can be extended to a more general local optimizer, performing corner-flipping as described in [Hsu] to remove crossings, bends, and unnecessary vias within channels.

7.7.4. Obstacle Locations and Sizes

The router has problems with obstacles in certain situations. The problems are the worst at switchbox boundaries, where the routing is constrained. Since the global router is oblivious to wiring within switchboxes, it may choose crossing points that require wires to cross, even when river routing would be possible or necessary. This sometimes makes it impossible for the switchbox router to complete its routing if there is not enough area available to route around an obstacle (Figure 7.8).

To avoid these problems the designer must be careful when placing obstacles at switchbox boundaries, especially those adjacent to subcells. Obstacles should not run parallel to switchbox boundaries for long distances.

One possible improvement would be to classify obstacles according to size. Large obstacles, such as major power and ground lines, can be treated as river-routed areas, while small obstacles can be more efficiently-handled with the current obstacle-avoidance

mechanism.

7.7.5. Grid Alignment

Magic's stem generator could also be improved. Currently it reserves an area on all sides of each subcell, even those sides with no terminals or those whose terminals are already grid-aligned. Since it generates the stem for each terminal without regard for any adjacent terminals, closely-spaced terminals may result in design rule violations or short-circuits.

7.7.6. User-Interface

Since Magic operates in a single batch operation, it is difficult to redo the routing in specific areas. With the current implementation, the only way to do this is to remove all of the automatic routing from the design and then re-invoke the automatic router. Similarly, changes to the switchbox router parameters affect all of the routing regions rather than just the one containing the problem. A more interactive routing system would provide greater designer control over the routing. Such a system could use many of the same concepts described in this thesis; however, the implementation would be much different.

7.7.7. Memory Usage

One implementation issue is the amount of memory used by the router. Since it explicitly represents every routing grid intersection with two bytes of information, large designs use large amounts of memory. A 10,000 by 10,000 lambda design with a routing grid spacing of 7 lambda requires over 4 megabytes of memory, and the router uses additional storage for heap elements during shortest path generation. On our current workstations, with 4 megabytes of memory and a remote file server, routing causes Magic to spend a lot of time paging.

CHAPTER 8

Summary and Conclusions

8.1. Introduction

This chapter summarizes the major points of the thesis, and concludes with some observations on the routing problem.

8.2. Summary

This thesis has examined modes of interaction between human designers and automatic routers. These range from complete hand-routing at one extreme, to batch routers that attempt to provide complete, "hands-off" solutions to the routing problem. Magic provides an intermediate alternative that combines the flexibility of hand-routing with the speed and accuracy of automatic channel routers, by allowing designers to first prewire critical nets by hand and then invoke an obstacle-avoiding automatic router that works around the hand-routing to make the remaining connections. Complementary talents of the intelligent human designer and the fast automatic router combine to form an effective solution to the problem of handling critical nets in special ways.

As explained in Chapters 1 and 2, traditional approaches suffer from one or more of the following problems: (1) slowness; (2) poor routing quality; or (3) inability to incorporate designer input. Hand-routing provides direct designer control over all aspects of routing; however, it is difficult and time-consuming. Net-at-a-time routers improve design times, but the routing they produce is poor because they do not plan ahead to handle interactions between nets. Traditional channel routers provide high-quality routing, but unlike net-at-a-time routers, they do not allow designers to wire critical nets in advance of automatic routing.

Chapter 3 introduced the basic concepts and components of Magic's obstacle-avoiding routing system. The fundamental idea of a preferred direction for crossing an obstacle led to a mechanism called a hazard, which guides the router in dealing with obstacles.

Magic's channel decomposer (Chapter 4) uses a simple yet effective heuristic to create a small number of large routing channels: minimize the sum of the perimeters of channels. The implementation of this heuristic relies on a corner-stitched data structure to provide a fast and elegant implementation.

Magic's global router (Chapter 5) supports obstacle-avoidance by considering the effect of obstacles upon each globally-routed net. It tries to reduce the complexity of the routing problem by trading-off net length and channel-routing difficulty, sometimes choosing a longer path for a net in order to avoid routing it across an obstacle. As part of its obstacle-avoiding path selection, the global router selects the exact location where each net crosses from one channel to another; this is called crossing placement. Because each crossing location is fixed, the resulting routing problems are switchboxes.

Chapter 6 described Magic's switchbox router. It uses the column-sweep approach coupled with special heuristics to avoid obstacles and route switchboxes. Although able to avoid obstacles, the switchbox router still produces results within 5 percent of the performance of the best conventional channel routers, which do not avoid obstacles.

To be an effective obstacle-avoiding router requires that an automatic router produce good routing, that there be a significant benefit in hand-routing critical nets, and that the hand-routing cause no significant harm to the overall routing quality. Chapter 7 presented results supporting all 3 of these points. Magic provided automatic routing whose net length was typically within 7 percent of that for carefully-placed hand-routing; the fact that the automatic router's net lengths are close to hand-routed net lengths demonstrates its effectiveness. Meanwhile, careful hand-routing reduced the routing's *RC* delay by up to 75 percent. Finally, the hand-routing had a small but usually positive effect upon the characteristics of the overall routing for the test chip.

8.3. Conclusions

Obstacle-avoiding routing is both desirable and feasible. It is possible to hand-route critical nets on an integrated circuit and automatically-route the remaining nets. One would expect that the presence of obstacles should hurt the quality of the automatic-routing that interacts with it; however, using the obstacle-avoiding routing techniques developed in this thesis, the presence of hand-routed nets has a barely measurable effect on the overall routing quality for an integrated circuit. At the same time, hand-routing results in substantial improvements in resistance and capacitance for the selected nets.

Like the routing problem, developing and implementing the routing system described in this thesis was itself a difficult task. Routing is NP-complete, so optimal solutions cannot be obtained with reasonable efficiency. To make matters worse, no small number of heuristics is adequate to always produce routing as good as that produced by human designers. From a more practical viewpoint, implementation revealed numerous small details that were important to consider yet were not anticipated prior to testing the code on real routing problems.

There will always be a need for human designers to hand-modify their design tools' output. Design tools are not yet, and may never be, as intelligent as human designers; therefore, designers continue to find places where they can improve upon automatically-generated results. This thesis has explored ways of allowing them to so.

Many other approaches to obstacle-avoiding routing are possible. As an example, Magic's router is a batch system, and designers specify their input prior to invoking the router; however, as discussed in Section 7.7, a more interactive approach could prove to be more natural and produce superior results. Further, Section 7.7 presents only a partial list of improvements that could be made to the current implementation. Consequently, rather than offering Magic's router as the solution to providing designer-control over routing, I offer it as a starting point for further research in obstacle-avoiding routing.

References

- [Arnold]
Arnold, M., *Corner-Based Geometric Layout Rule Checking for VLSI Circuits*, Report No. UCB/CSD 86/264, UC Berkeley Computer Science Division, November 1985.
- [Bruell and Sun]
Bruell, P., and P. Sun, "A 'Greedy' Three Layer Channel Router", *Proceedings IEEE International Conference on Computer-Aided Design* (1985), pp. 298-300.
- [Burstein and Pelavin]
Burstein, M., and R. Pelavin, "Hierarchical Wire Routing", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, Vol. CAD-2, No. 4 (Oct 1983), pp. 223-234.
- [Burstein and Youssef]
Burstein, M., and M. Youssef, "Timing Influenced Layout Design", *Proceedings 22nd Design Automation Conference*, Las Vegas (1985), pp. 124-130.
- [Chen]
Chen, N. P., C. P. Hsu, and E. S. Kuh, "The Berkeley Building-Block Layout System for VLSI Design", ERL memo UCB/ERL M83/10, UC Berkeley, February 1983.
- [Chen H]
Chen, H., Private Communication.
- [Chen K]
Chen, K. A., et. al., "The Chip Layout Problem: An Automatic Wiring Procedure", *Proceedings 14th Design Automation Conference*, New Orleans (1977), pp. 298-302.
- [Chiba]
Chiba, T., N. Okuda, T. Kambe, I. Nishioka, T. Inufushi, and S. Kimura, "SHARPS: A Hierarchical Layout System for VLSI", *Proceedings 18th Design Automation Conference*, Nashville (1981), pp. 820-827.
- [Clow]
Clow, G. W., "A Global Routing Algorithm for General Cells", *Proceedings 21st Design Automation Conference*, Albuquerque (1984), pp. 45-51.
- [Dai]
Dai, W., T. Asano, and E. S. Kuh, "Routing Region Definition and Ordering Scheme for Building-Block Layout", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, Vol CAD-4, No. 3 (July 1985), pp. 189-97.
- [Deutsch]
Deutsch, D., "A 'Dogleg' Channel Router", *Proceedings 19th Design Automation Conference*, San Francisco (1976), pp. 425-433.
- [Garey and Johnson]
Garey, M., and D. Johnson, *Computers and Intractability*, W. H. Freeman and Co., San Francisco (1979).
- [Hamachi 84]
Hamachi, G. T., and J. K. Ousterhout, "A Switchbox Router with Obstacle Avoidance", *Proceedings 21st Design Automation Conference*, Albuquerque (1984), pp. 173-179.

- [Hamachi 85]
Hamachi, G. T., and J. K. Ousterhout, "Magic's Obstacle-Avoiding Global Router", *1985 Chapel Hill Conference on Very Large Scale Integration*, H. Fuchs Ed., Computer Science Press, 1985, pp. 145-164.
- [Hashimoto and Stevens]
Hashimoto, A., and J. Stevens, "Wire Routing By Optimizing Channel Assignment Within Large Apertures", *Proceedings 8th Design Automation Workshop* (1971), pp. 155-169.
- [Hassett]
Hassett, J. E., "Automated Layout in ASHLAR: An Approach to the Problems of 'General Cell' Layout for VLSI", *Proceedings 19th Design Automation Conference*, Las Vegas (1982), pp. 777-784.
- [Hightower 69]
Hightower, D., "A Solution to the Line Routing Problem on the Continuous Plane", *Proceedings Design Automation Workshop* (1969), pp. 1-24.
- [Hightower 74]
Hightower, D., "The Interconnection Problem: A Tutorial", *IEEE Computer* (April 1974), pp. 18-32. (1980) pp. 12-21.
- [Hightower 80]
Hightower, D., "A Generalized Channel Router", *Proceedings 17th Design Automation Conference*, Minneapolis (1980), pp. 12-21.
- [Horowitz]
Horowitz, E., and S. Sahni, *Fundamentals of Computer Algorithms*, Computer Science Press, Inc. (1978), pp. 183-188.
- [Hsu]
Hsu, Chi-Ping, "General River Routing Algorithm", *Proceedings 20th Design Automation Conference*, Miami (1983), pp. 578-583.
- [Joobbani]
Joobbani, R., *WEAVER: An Application of Knowledge-Based Expert Systems to Detailed Routing of VLSI Circuits*, SRC Technical Report No. T85044, July 1985.
- [Kajitani]
Kajitani, Y., "Order of Channels for Safe Routing and Optimal Compaction of Routing Area", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, Vol CAD-2, No. 4 (October 1983), pp. 293-300.
- [Kernighan]
Kernighan, B., D. Schweikert, and G. Persky, "An Optimum Channel-Routing Algorithm for Polycell Layouts of Integrated Circuits", *Proceedings 10th Design Automation Workshop* (1973), pp. 28-46.
- [Leblond]
Leblond, A., "CAF: A Computer-Assisted Floorplanning Tool", *Proceedings 20th Design Automation Conference*, Miami (1983), pp. 747-753.
- [Lee]
Lee, C. Y., "An Algorithm for Path Connections and its Application", *IRE Transactions on Electronic Computers* (Sept. 1961), pp. 346-365.
- [Mead and Conway]
Mead, C., and L. Conway, *Introduction to VLSI Systems*, Addison-Wesley Publishing Co., Reading, Mass., 1980.

- [Ng]
Ng, C. H., "A Symbolic-Interconnect Router for Custom IC Design", *Proceedings 21st Design Automation Conference*, Albuquerque (1984), pp. 52-58.
- [Nilsson]
Nilsson, Nils J., *Problem-Solving Methods in Artificial Intelligence*, McGraw-Hill, Inc., New York, 1971.
- [Ousterhout 84a]
Ousterhout, J. K., "Corner Stitching: A Data Structuring Technique for VLSI Layout Tools", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, Vol CAD-3, No. 1 (January 1984), pp. 87-99.
- [Ousterhout 84b]
Ousterhout, J. K., G. T. Hamachi, R. N. Mayo, W. S. Scott, and G. S. Taylor, "Magic: A VLSI Layout System", *Proceedings 21st Design Automation Conference*, Albuquerque (1984), pp. 152-159.
- [Ousterhout 85]
Ousterhout, J. K., G. T. Hamachi, R. N. Mayo, W. S. Scott, and G. S. Taylor, "The Magic VLSI Layout System", *IEEE Design and Test of Computers*, Vol. 2, No. 1 (February 1985), pp. 19-30.
- [Pendleton]
Pendleton, J., Private Communication.
- [Persky]
Persky, G., D. Deutsch, and D. Schweikert, "LTX--A System for the Directed Automatic Design of LSI Circuits", *Proceedings 19th Design Automation Conference*, San Francisco (1976), pp. 399-407.
- [Rivest]
Rivest, R. L., "The 'PI' (Placement and Interconnect) System", *Proceedings 19th Design Automation Conference*, Las Vegas (1982), pp. 475-481.
- [Rivest and Fiduccia]
Rivest, R. L., and C. M. Fiduccia, "A Greedy Channel Router", *Proceedings 19th Design Automation Conference*, Las Vegas (1982), pp. 418-424.
- [Romeo]
Romeo, F., and A. Sangiovanni-Vincentelli, "Probabilistic Hill Climbing Algorithms: Properties and Applications", *1985 Chapel Hill Conference on Very Large Scale Integration*, H. Fuchs Ed., Computer Science Press, 1985, pp. 393-417.
- [Rothermel and Mlynski]
Rothermel, H.-J., and D. A. Mlynski, "Routing Method for VLSI Design Using Irregular Cells", *Proceedings 20th Design Automation Conference*, Miami (1983), pp. 257-262.
- [Sangiovanni-Vincentelli 83a]
Sangiovanni-Vincentelli, Alberto, Private Communication, February 1983
- [Sangiovanni-Vincentelli 83b]
Sangiovanni-Vincentelli, A., and M. Santomauro, "YACR: Yet Another Channel Router", *Proceedings Custom Integrated Circuit Conference* (1983), pp. 327-331.
- [Sangiovanni-Vincentelli 84]
Sangiovanni-Vincentelli, A., M. Santomauro, and J. Reed, "A New Gridless Channel Router: Yet Another Channel Router the Second (YACR-II)", *Proceedings IEEE International Conference on Computer-Aided Design* (1984), pp. 72-75.

- [Scott 84]
Scott, W. S., and J. K. Ousterhout, "Plowing: Interactive Stretching and Compaction in Magic", *Proceedings 21st Design Automation Conference*, Albuquerque (1984), pp. 166-172.
- [Scott 86a]
Scott, W. S., R. N. Mayo, G. T. Hamachi, and J. K. Ousterhout, *1986 VLSI Tools: Still More Works by the Original Artists*, Report No. UCB/CSD 86/272, UC Berkeley Computer Science Division, December 1985.
- [Scott 86b]
Scott, W. S., "Compaction and Circuit Extraction in the MAGIC IC Layout System", Report No. UCB/CSD 86/269, UC Berkeley Computer Science Division, Spring 1986.
- [Soukup]
Soukup, J., "Circuit Layout", *Proceedings of the IEEE*, Vol. 69, No. 10 (Oct. 1981), 1281-1304.
- [Soukup 83]
Soukup, J., Private Communication, Dec. 1983.
- [Suen]
Suen, L., "A Statistical Model for Net Length Estimation", *Proceedings 18th Design Automation Conference*, Nashville (1981), pp. 769-774.
- [Wiesel and Mlynski]
Wiesel, M., and D. A. Mlynski, "An Efficient Channel Model for Build Block LSI", *Proceedings 1981 IEEE International Symposium on Circuits and Systems*, pp. 118-121.
- [Yoshimura and Kuh]
Yoshimura, T., and E. S. Kuh, "Efficient Algorithms for Channel Routing", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, Vol. CAD-1, No. 1 (Jan 1982), pp. 25-35.
- [Vecchi and Kirkpatrick]
Vecchi, M., and S. Kirkpatrick, "Global Wiring by Simulated Annealing", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, Vol. CAD-2, No. 4 (October 1983), pp. 215-222.