A Multiple Layer Contour-Based Gridless Channel Router

PATRICK GROENEVELD

Abstract—**With the continuing advances of VLSI-technology, multiple layer channel routing has become increasingly important. A new n-layer gridless channel router is presented. Instead of the commonly employed grid it uses a set of contours as routing framework. The contours together with a set of reserved areas prevent short circuits and ensure routability of all nets. A compact routing can be expected since the framework is a very adequate model of the physical channel. We will show that it is also versatile enough to handle the classic vertical constraint problem. The inherent flexibility enables a design rule driven tradeoff between the number of vias, the wire length, and the channel height. The behavior of the algorithm has been investigated for many different channels, among which the well-known "difficult example" and its recent derivatives. Very competitive results have been achieved for channel height as well as the via count.**

I. Introduction

THE CHANNEL routing problem has been studied extensively over the past decade. In various VLSI design styles such as standard cell, macro-cell, or even sea-of-gates the routing area is decomposed into a number of channels. Near optimality, reliability, and efficiency are the main reasons for the popularity of the channel router today. More general detailed routers cannot cope with the complexity of the wiring problem.

Traditional channel routers use a grid matrix as the representation of the routing area. A very wide variety of grid-based routing algorithms have been published [11], [6], [18], [15], [1], many of which produce (close to) optimal results within this routing framework. Unfortunately, the discrete grid model of the routing area has a number of serious drawbacks.

1) The pitch of the grid is determined by the worst-case spacing requirement over all layers. With different design rules for each layer and with bulky vias this usually leads to a considerable waste of area.

2) A uniform wire width is required. Critical nets (e.g., power and clock) have to be treated separately.

3) The terminals are required to align on grid lines.

Several approaches have been published to get around the grid-related constraints. The pitch problem can be reduced by clever post-processing: variable track spacing [17], or compaction of the channel [5], [13], [3]. To evade the terminal alignment constraint a virtual grid [14], [12] and "gridding" the terminals by river routing [2], [13] have been proposed. Some of these "hidden-grid" routers obtained the unjustified appellation *gridless*. Only by abandoning the grid framework completely can the drawbacks be avoided. Although grid-free approaches to two- and three-layer channel routing have been published, multiple layer channel routing has remained the exclusive domain of the grid framework until now.

Many gridless variable-width channel routers (e.g., [4], [8], [16]) identify two-terminal wire segments, called *trunks*. Using a weighted constraint graph the trunks are ordered such that the channel height is minimized while the vertical constraints are obeyed. Finally, the trunks are converted into physical wires by a special purpose compactor. This symbolic routing framework is, of course, an inaccurate representation, the disadvantages of which must be compensated by the compactor. Vertical constraints pose another problem. In the grid framework, for each pair of "facing" terminals at most one vertical constraint is introduced because of the alignment in columns. For gridless routers the terminals are not required to align, which inevitably leads to additional vertical constraints. On the symbolic (trunk) level the resulting cyclic vertical constraints are hard to break. In the approach presented here we attempt to bring the algorithm "closer to silicon" by using a set of contours *directly* as the routing framework. More physical details of the channel can be taken into account in this way. The contour framework is also flexible enough to prevent most vertical constraints.

Our basic approach is to stack wires on top of each other. The nets are routed one at a time by a procedure which we called *net router*. It wraps the wires tightly around the contours to minimize the channel height. The contour framework and the net router will be discussed in Sections IV and V, respectively. In Section VI we will describe the *scheduler* which determines the order in which the nets are routed. It has an important role in handling cyclic vertical constraints. When the routing of all nets has been completed, the top side of the channel (including its pile of wires) is shifted as close as possible towards the bottom side. Finally the redundant wire jogs are removed by a wire straightening post-processor. This algorithm and other additions will be discussed in Section VII.

Manuscript received November 8, 1988; revised August 29, 1989. This work was supported by Philips Research Laboratories, Eindhoven, The Netherlands. This paper was recommended by Associate Editor A. E. Dunlop.

This paper is an expanded version of the paper presented at ICCAD-87.

The author is with the Faculty of Electrical Engineering, Delft University of Technology, 2628CD Delft, The Netherlands.

IEEE Log Number 9037114.

II. Problem Formulation

To make the discussion easier we assume that the routing area is a horizontal channel. The bottom and top boundaries of the channel can be irregularly shaped and their vertical offset is adjustable. Terminals can be specified at arbitrary positions on the horizontal edges of the bottom and top channel boundaries and at "floating" positions on the left and right boundaries. Each terminal can have any width and can be in any layer. A set of nets $N = \{N_i\}$, $i = 1, \cdots, n$ is provided to specify the connectivity of the terminals. The wire width of each net in each of the layers is given.

The problem of channel routing is to complete the routing of all nets such that the following parameters are kept small:

- the channel height;
- the number of vias;
- the total wire length.

Minimizing the channel height is still the main objective. Currently the via count has gained in importance because the yield of most production processes depends primarily on this parameter.

III. Design Rules

There are $m (m \geq 2)$ layers available for wiring. A layer is said to be *horizontal* if the orientation of the wires is mainly along the fixed boundaries of the channel. In a *vertical* layer the wires are mainly used to cross the channel. Within certain limits wires in the vertical direction are allowed in a horizontal layer and vice versa. A scheme of alternating horizontal and vertical layers is presumed (e.g., for three-layer routing either the HVH- or the VHV-type can be chosen).

In a conventional channel router the grid incorporates all process dependent parameters. For our more down-to-earth gridless router, however, the process design rules play a more intricate role in the routing process. We distinguish the following relevant design rule parameters.

1) SP_l the minimum spacing in layer l;
2) $WD_l(i)$ the (user-specified) wire width of net i in layer l;
3) $\mathrm{VIA}_{l1,l2}(i)$ the total width of the contact hole pattern of net i in layer $l1$, given that it connects the layers $l1$ and $l2$. Vias exist only between adjacent layers.

It is assumed that no additional design rule constraints exist between layers. In this description we also assume that the orientation of all wire segments is rectilinear.

IV. Routing Framework

The routing framework must provide the limits within which a new wire can be laid. It should prevent short circuits and design rule violations. It must also ensure that routing a net does not make other nets unroutable. Basically we reduce the two dimensions of the routing area into the $1\frac{1}{2}$ dimensions of a curved contour. During the routing process the channel will be split into a top part and a bottom part, each of which is a more or less independent environment. Routing can take place on both sides of the channel independently since the height of the channel is adjustable.

4.1. Contours

We define a *contour* as a set of line segments shielding the upper limit of previously laid wires in a layer. It can be seen as a "protective cover" for existing wires. Future wires will have to be laid at minimum spacing SP_L from the contour.

We make a fundamental distinction between horizontal and vertical layers. In a vertical layer two contours are used on each side of the channel (see Fig. 1).

1) The *bottom contour* covers the wires touching the boundary on this side of the channel. If a wire crosses to the opposite side of the channel, the contour around it will bend up to "infinity." Future wires in the corresponding layer are not allowed to pass this contour obstacle. Initially the bottom contour is wrapped around the channel boundary.

2) The *top contour* is wrapped around wires coming from the opposite side of the channel and ending above the bottom contour. Each top contour is initialized as a single horizontal line spanning the entire length of the channel and of which the height is "infinite."

A new wire in a vertical layer must be placed between the bottom and top contours. It will be blocked if the spacing between the contours is too small. In this way "forbidden" intervals can be introduced.

In horizontal layers only one contour is used on each side of the channel: the bottom contour. Part of our strategy to ensure 100% completion is to require that the contour in a horizontal layer does not contain any unpassable obstacles (that is, forbidden intervals). In this way any pair of two horizontal positions in the channel can always be reached by a single continuous wire in a horizontal layer. A wire in a horizontal layer may not cross to the opposite side of the channel since this would introduce a forbidden interval.

4.2. Territories

Short circuits with previously routed nets are prevented by the contour framework. To ensure the routability of the still unrouted nets a *territory* is allocated around each terminal. The territory of a terminal is a column-shaped area which is just wide enough for a via and of infinite height. It is only valid in a vertical layer on the same side of the channel as the terminal itself. Wires of other nets in this layer are not allowed to enter the territory until the terminal has been routed. The dashed lines in the example channel in Fig. 2 indicate the territory boundaries of the terminals of net 4. Since we do not allow a horizontal layer to contain forbidden intervals the terminals in horizontal layers obtain a territory in one of the adjacent ver-

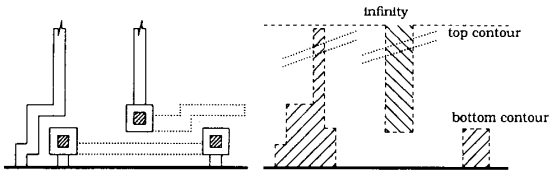


Fig. 1. The bottom side in a 2-layer example channel (left). The shaded areas (right) are protected by the contours of the vertical layer.

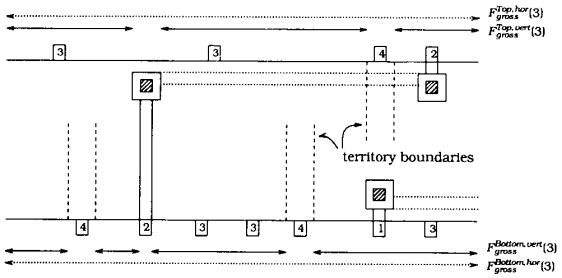


Fig. 2. A partially routed channel in VH-style. The intervals $F_{gross}(3)$ of net 3 are indicated by arrows. The vertical layer is drawn using solid lines, the horizontal layer using dotted lines.

tical layers. Thus area is reserved for a via connecting the terminal to a vertical layer.

If the spacing between each pair of adjacent territories is at least SP_l all terminals can be connected to a wire in a horizontal layer, which guarantees routability. We will later show how to deal with the terminal spacing constraint emerging from territory overlap.

V. Net Router

The nucleus of this sequential contour router is the algorithm that connects the terminals of a single net to one another. It consists of three parts. First a *wire interval graph* is constructed. From the contours and the territories a set of horizontal intervals can be derived in which a wire of the net could be laid without causing design rule violations or unroutability of nets later on. The idea is to capture the adjacencies of these intervals in a graph containing all possible topological paths of the wire. Using the wire interval graph as framework a *global router* determines roughly the path of the wire. The weights of the edges in the graph enable a tradeoff between the number of vias, the channel height, and the wire length in each of the layers. Finally, the *detailed router* generates the actual wires based on the path in the wire interval graph.

5.1. Wire Interval Graph

The goal is to make the wire interval graph $G_i(V_i, E_i)$ an as accurate as possible representation of the free wiring intervals for net i. The free area is limited by contour obstacles in the channel and by territories of other nets. Contour obstacles occur if the spacing between a pair of bottom and top contours is too small to place the wire without causing design rule violations. More formally, a wire of net i is not allowed in those (horizontal) intervals where the (vertical) spacing between the pair of contours is smaller than

$$WD_{\text{layer}}(i) + 2 \times SP_{\text{layer}}.$$

We define:

$$C^{\text{side, layer}}(i)$$

to be the set of forbidden intervals for which the spacing is too small on *side* of the channel and in *layer*.

The territories of other (still unrouted) terminals also introduce forbidden intervals. We declare T_j to be the set of intervals protected by the territories of net j. Obviously, $T_j = \varnothing$ if net j has already been routed. The intervals of

$$T^{\text{side, layer}}(i) = \bigcup_{\{j \in N \mid j \neq i\}} T_j$$

are the forbidden areas caused by territories on *side* of the channel and in *layer*. It is the union of the territories of all other unrouted nets.

Removing all forbidden intervals from the channel yields the *gross free space* of net i:

$$F_{\text{gross}}^{\text{side, layer}}(i) = \overline{C^{\text{side, layer}}(i)} \cap \overline{T^{\text{side, layer}}(i)}.$$

At this point the minimum spacing between an obstacle and the wire has not yet been taken into account. The *net* free space $F^{\text{side, layer}}(i)$ is derived from the gross free space by shrinking both bounds of each interval by SP_{layer} and deleting those intervals of which the resulting width is smaller than $WD_{\text{layer}}(i)$. This set contains the intervals within which layout patterns for net i can be placed on top of the contour.

Let $F(i)$ be the set of all free intervals in all layers. The wire interval graph $G_i(V_i, E_i)$ is constructed by determining the adjacencies of the intervals. With each interval of $F(i)$ a set of vertices in $G_i(V_i, E_i)$ will be associated. We also assign an x-position within the corresponding interval of $F(i)$ to each vertex $v \in V_i$. One of the ways to obtain the wire interval graph from $F(i)$ is shown in the following and is illustrated in Figs. 3–6.

Wire Interval Graph Construction

1) Represent each interval in $F(i)$ by a pair of connected vertices. The edge between the two vertices corresponds to a horizontal wire.
2) Find the set of horizontal overlap intervals

   $$O^{\text{side, layer 1} - \text{layer 2}}(i) = F^{\text{side, layer 1}}(i) \cap F^{\text{side, layer 2}}(i)$$

   for all pairs of adjacent layers and on both sides of the channel. For each of these intervals insert two pairs of connected vertices in the graph: one pair for each end of the overlap interval. Of a vertex pair one vertex is assigned to the corresponding interval of $F^{\text{side, layer 1}}(i)$, the other to the interval of $F^{\text{side, layer 2}}(i)$. The edge between them represents a possible location for a via. If the size of the overlap interval is too small for a via no vertex pair will be generated.
3) Find a similar set of vertical overlap intervals

   $$O^{\text{layer}}(i) = F^{\text{bottom, layer}}(i) \cap F^{\text{top, layer}}(i)$$

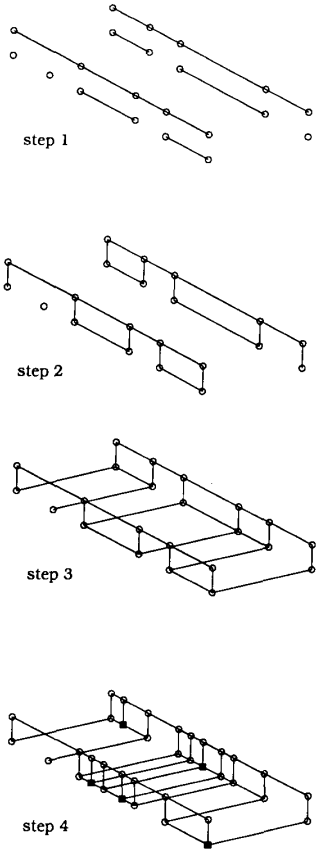


Fig. 3. Illustration of the major steps during graph construction for the channel in Fig. 2.

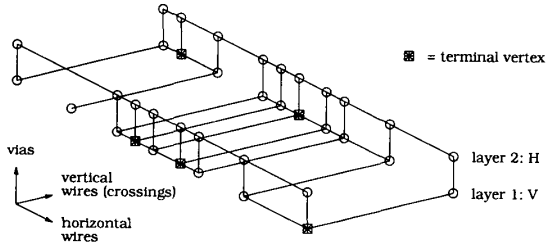


Fig. 4. The final wire interval graph G_3 for net 3 in the channel of Fig. 2.

for all vertical layers. For each of these intervals generate two pairs of connected vertices. The pairs are placed at opposite sides of the overlap interval. Of a vertex pair one vertex is assigned to the corresponding interval of $F^{\text{top, layer}}(i)$, the other to the interval of $F^{\text{bottom, layer}}(i)$. The edge between them represents a possible location for a wire to cross the channel. No vertex pair will be generated if the size of an overlap interval is too small for a wire ($< WD_{\text{layer}}(i)$).

4) To represent the terminal locations of net i we insert "terminal vertices" in G_i. A terminal vertex obtains the x-position of the terminal and is assigned to the corresponding interval (and layer) of $F(i)$. The accuracy of the graph can be enhanced by inserting vertices and edges at aligning positions on the opposite side of the channel and in adjacent layers.
5) Terminals which are already connected to one another outside the channel are called *electrically equivalent*. Only one connection per set of electrically equivalent terminals is required. These terminals could also serve as "feedthrough" to shorten the paths to other terminals of the net. This property can be represented adequately by zero-weight edges between the vertices of electrically equivalent terminals.

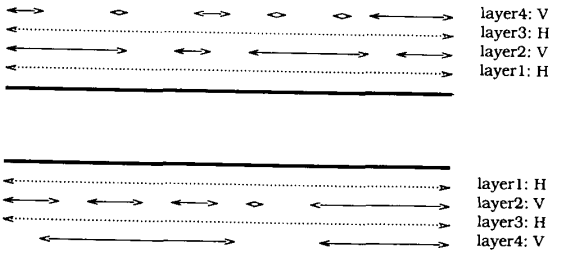


Fig. 5. Intervals $F_{\text{gross}}^{\text{side, layer}}$ in a 4-layer example.

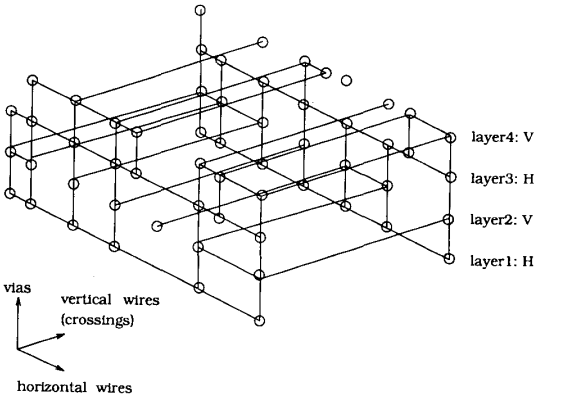


Fig. 6. A 3-D view of the resulting 4-layer wire interval graph.

5.2. Global Routing

The global routing of a net i is performed on its wire interval graph $G_i(V_i, E_i)$. An edge in this graph may represent a via, a horizontal or a vertical wire segment. A number of heuristics can be applied to determine the weights of the edges in this graph. For instance, the weight can depend on the following factors:

- an estimate of the wire length in combination with the resistance in the layer;
- the amount of crosstalk we allow between wires in adjacent layers;
- the amount in which the wire will contribute to the critical path for the channel height. The wiring density can be spread evenly over all layers by adjusting

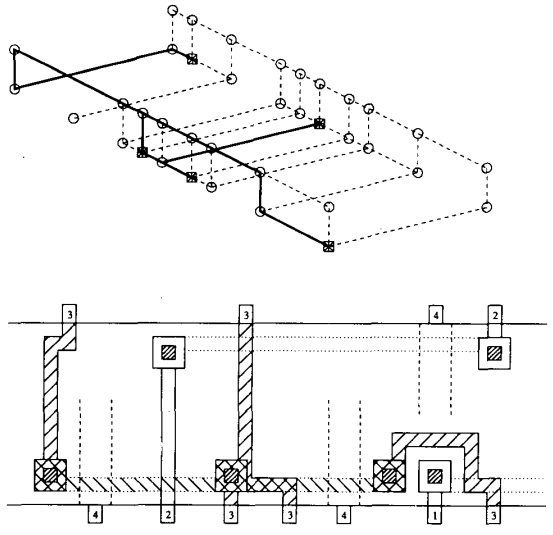


Fig. 7. A path found for net 3 in G_3 and the corresponding detailed routing.

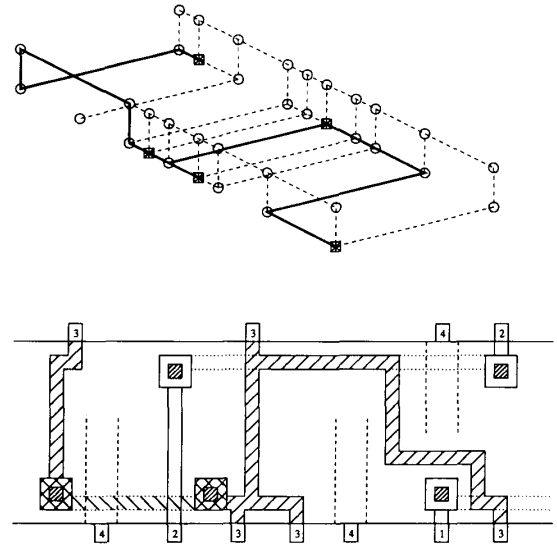


Fig. 8. The cost penalties of the edges can be used to reduce the number of vias.

this cost parameter dynamically during the routing process.

The weight parameters can be tuned to make a tradeoff between conflicting quality factors. By increasing the cost of a via vertex, for instance, more wires will be routed in a vertical layer, thereby obviating as many vias as possible (see Figs. 7–9).

The task of the global router is to find the minimum-cost path connecting all terminal vertices in $G_i(V_i, E_i)$. For arbitrary numbers of terminals this is an *NP*-hard Steiner tree problem. We used a modified version of Dijkstra's shortest path algorithm to find a path between the terminals. It produces acceptable results and never generates a cycle. The worst-case complexity of the algorithm is $O(n^2)$ in which n is the number of vertices in V_i.

5.3. Detailed Routing

During the detailed routing phase the actual layout will be generated, based on the topological path in the wire interval graph. First *trunks* are identified. In our context a trunk is a set of connected vertices in the same layer and on the same side to which the path of the net belongs. We define a *crossing* to be a wire segment which crosses the channel in a vertical layer. A quite straightforward sequence of steps to perform the detailed routing is shown below.

Detailed Routing

1) *Trunk identification.*
2) *Via placement*: A number of heuristics are applied to place the via(s) of each trunk. The via position depends mainly on the height of the contour in the horizontal layer. To prevent detours the positions of the other connections of the trunks to be connected are taken into account.

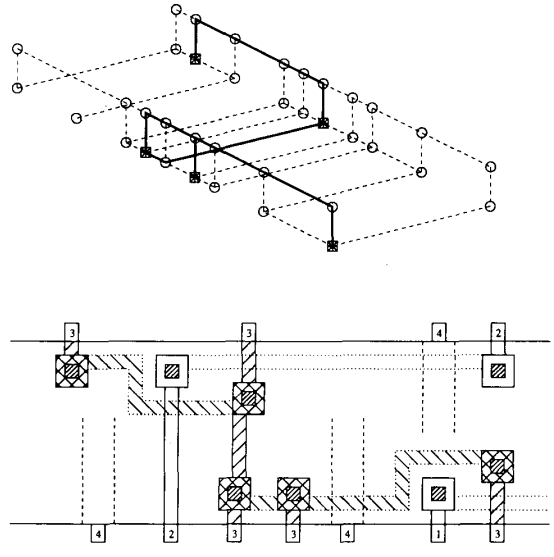


Fig. 9. It is also possible to adjust the weights of the edges such that the wire length in the vertical layer is minimized. As a result the number of vias will increase.

3 *Crossing placement*: The position of a vertical wire crossing the channel is determined according to the (local) contour heights on each side of the channel. Again the position of the other connections is taken into account.
4) *Wire generation in vertical layers*: Once the crossings and the vias have been placed they can be connected by a horizontal wire. In each trunk this wire is wrapped as close as possible around the contour. Some useless wire bends can be prevented by filtering the contour envelope.

5) *Layer conversion of terminals in horizontal layers*: In the following step a wire in a horizontal layer will be generated. Since there are no territories in a horizontal layer this wire could cover some unrouted terminals in the layer. Routability is maintained by diverting the threatened terminals to an adjacent vertical layer. The diversion is performed by generating a via which "shifts" the terminal into the layer of its territory.
6) *Wire generation in horizontal layers*: As in the vertical layer a horizontal wire is generated which spans all the terminals and vias of a trunk. The wire is bent at minimum spacing around the contour to minimize the channel height. Useless wire jogs will be removed by a wire-straightening post-processor.
7) *Contour update*: The contours generated in the previous steps cover the new wires of each trunk exactly. These new contours will be merged with the existing contours.
8) *Weight factor adjustment*: For performance reasons the wire interval graph G_i is derived from a "master interval graph" G. The critical path in each layer is found by comparing the contours on the bottom and top. The weight of the edges in G which correspond more or less with the critical path is increased.

VI. Scheduler

6.1. Ordering Constraints

A graph that has traditionally been associated with channel routing is the *vertical constraint graph* (VCG). It captures the wire ordering constraints of the nets. Each vertex of the VCG represents a net. A directed edge from vertex a to vertex b indicates that a horizontal wire segment of net a must be placed above a segment of net b. A vertical constraint edge originates from a pair of "facing" terminals on opposite sides of the channel. If their x-positions overlap, the wire which connects to the terminal on the top must be placed above the wire which connects to the bottom terminal. In grid-based routers the facing terminals share the same column.

The main routing problem which can be derived from the VCG is the *cyclic vertical constraint* (Fig. 10). The cycle in the VGC can be broken by splitting a net into a number of subnets. The subnets are connected to one another by a *dogleg* pair of vias (Fig. 11). The majority of channel routers must apply separate cycle detectors and removers since their search space for a wire realization does not include doglegs. In our algorithm the interval graph G_i contains all possible realizations of net i, including numerous detours and doglegs. A special constraint breaker, therefore, is not required.

To detect vertical constraints a reserved-layer model is applied. If no horizontal wire segments are allowed in vertical layers however, all "facing" terminals of which the x-coordinates overlap will introduce a vertical constraint. Solving the numerous constraints on a higher symbolic level is very cumbersome. A VCG-based scheduling also lacks accuracy to find the best way of breaking a vertical constraint loop. In the algorithm presented here the constraint problem is shifted towards the contour framework which is closer to the silicon. On this level the majority of the constraints can be solved easily and efficiently by wire jogging. Better use of the available wiring resources in all layers can be achieved in this way.

6.2. Net Ordering Algorithm

The main task of the scheduler is to determine the order in which the nets will be presented to the net router. As discussed in the previous section the vertical constraint graph cannot be used for this purpose since it does not properly represent the ability to jog wires in a vertical layer. In our framework there are usually only a few nets which cannot be routed immediately due to the large search space in G_i. In the example of Fig. 10, for instance, all nets can be routed at once. Some of the nets, however, contain detours or doglegs. Their routing could be postponed to allow the more efficient nets to be routed first. There is a fair chance that by removing territories better paths become available. To capture the efficiency of a net i we derive a "quality value" from its path in the graph G_i. The length of the detours and the relative amount of vias per terminal could be used to compose this value. Fig. 12(a) shows an example of the initial quality values for the nets in the channel of Fig. 10. In this channel net 10 should be routed first since it is the only net that does not require a dogleg or a detour. Therefore, it has the highest quality value. Due to the routing of this net the quality values of nets 8 and 9 improve (Fig. 12(b)). A straightforward algorithm for net scheduling would be to select each time the net with the best quality value. The final result of this procedure is shown in Fig. 13.

Determining the quality of a single net i is rather "expensive" because it requires the construction of a wire interval graph G_i and a global routing. A total of $\frac{1}{2}n(n+1)$ of these calculations are required since the quality of all unrouted nets will have to be redetermined after each routing. To reduce the CPU-time consumption we apply a more "greedy" scheduling algorithm. Instead of deriving an accurate quality value from the path in G_i a rough estimate of the quality is used, based on the following criteria.

- The "single-side" criterion. All terminals of single-side nets are either at the bottom or at the top side of the channel. Therefore, the path of these nets does not have to cross the channel. Single-side nets are more likely to have a high quality value since they can always be routed without detours and doglegs, provided there is no overlap of territories. Moreover, routing these nets removes territories without introducing new obstacles. In this way the number of possible sources of constraints decreases and, with that, the wiring freedom increases.
- A user-specified ordering of the net. In this way a global wire ordering (see [10]) can be specified to prevent unnecessary wire twisting in adjacent channels.
- Additional heuristics such as the number of terminals of a net.

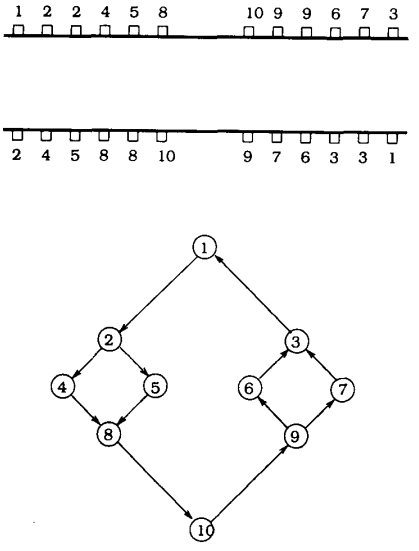


Fig. 10. A gridded example channel. The corresponding vertical constraint graph is cyclic (after [1]).

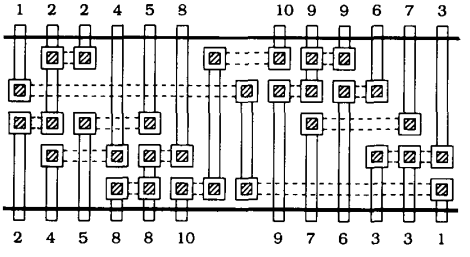


Fig. 11. The original gridded and nonoptimized result of [1]. It contains doglegs to break the cyclic vertical constraint and to reduce the channel width.

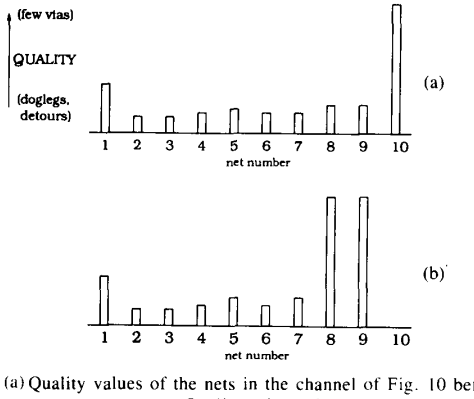


Fig. 12. (a) Quality values of the nets in the channel of Fig. 10 before any net was routed. (b) Quality values after net 10 was routed.

The nets will be presented to the net router in this order. We will now distinguish two cases in which the net router does not perform detailed routing.

1) *Hard Constraint:* The net cannot be routed because there is no path in G_i which connects all terminals of i.

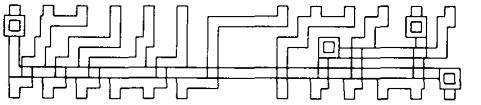


Fig. 13. Routing produced by this algorithm. By jogging in the vertical layer most vertical constraints can be solved without using vias.

2) *Soft Constraint:* The implementation of the path the net i does not come up with a certain value for the quality. This "quality standard" is initially such that only nets without detours and doglegs are routed.

Due to the net ordering a successful routing is more likely to occur. If the routing of a net fails, due to a constraint, the next net in order will be routed. We apply a simple heuristic to determine the time to retry the routing of a rejected net. As soon as a terminal adjacent to a terminal of the rejected net is successfully routed the rejected net will be reattempted. The routing of an adjacent terminal has removed a territory which may have created the space for a better path for the rejected net. The process continues until all nets are routed or all remaining (unrouted) nets have a constraint. In the latter case the quality standards for soft constraints must be temporarily lowered to enlarge the search space in G_i. Although the worst-case complexity of this greedy algorithm is still $O(n^2)$ the behavior for most practical channels is much better.

Both algorithms discussed above usually achieve 100% completion. For the majority of channels (including the "difficult example") even doglegs are not required. Only in a few cases do all remaining unrouted nets have a hard constraint. If the hard constraints are not caused by terminal spacing violations, the channel must be made longer in the x-direction. In this very rare case the problem is caused by the absence of edges in G_i to cross the channel. By enlarging the channel it is always possible to create vertical crossing edges in G_i. The only other (and more likely) source of hard constraints are spacing violations between territories. This problem can be solved by shifting terminals apart (see Section VII-7.1).

VII. Extensions

7.1. Overlapping Territories

In order to ensure 100% completion, it is required that the spacing between two adjacent territories is at least SP_l. All terminals can be connected to a horizontal layer in this way. This results in an additional spacing constraint between terminals which have territories in the same layer. In many cases 100% completion can be achieved even if the spacing between some territories is smaller. This is usually not possible if there are arrays of overlapping territories. Our solution is to remove all wires from the channel if some nets are unroutable due to a territory spacing violation. Prior to the second attempt to route the channel the overlapping territories will be shifted apart to fulfill the spacing constraint. A *fan-out* wire pattern connects

each original terminal to its new territory (see Fig. 14). The territories can be shifted such that the height of the required fan-out pattern is minimal. Obviously the sum of the widths of the territories must be smaller than the length of the channel, otherwise the channel length must be increased.

7.2. *Terminals on Vertical Channel Boundaries*

It can be useful to allow terminals on the vertical edges of horizontal channel boundaries. In our macro-cell router [9] we observed that the congestion at the critical channel intersections can be reduced by placing terminals on vertical edges (see Fig. 15). Until now vertical terminals on fixed boundaries have been the exclusive territory of (grid-based) switch-box routers. Approaches with switch-box routing areas at channel junctions have been published (e.g., [19]). Within limits it is also possible to use a (variable wire width) channel router for this purpose. A preprocessor can be implemented to convert vertical terminals into horizontal ones with nonoverlapping territories. A wiring pattern similar to the previously mentioned fan-out pattern is required. The maximum number of vertical terminals on an edge is limited by the nearest obstacle on the channel boundary.

7.3 *Wire Straightening*

Each wire is wrapped tightly around the existing contour to obtain the smallest contour height. This process of "greedy" compaction introduces many useless wire bends. By removing the jogs a considerable reduction in wire length and memory usage can be achieved.

Our wire straightening algorithm uses two contours which mark the upper and lower limits of the area within which a wire can be reshaped. These contours traverse through the pile of wires in reverse order as the wires were routed. Each time the wire is stretched between the contours such that it contains the fewest number of bends. Special care has to be taken while shifting a via since it connects to another layer. The "shiftability" of each via is determined by scanning the contours in the adjacent layer. During the wire straightening process it is also possible to reduce the length of wires in critical layers such as polysilicon. Vias can be shifted such that the wire length in these layers is minimized.

VIII. Experimental Results

A VHV-type version of the algorithm has been implemented in C under the UNIX operating system. The program is integrated in our macro-cell routing system [9] and in a number of layout assembly tools of commercial partners. A considerable number of large real-life chips in various processes have been routed by the program. The weight factors of the algorithm were tuned such that a tradeoff between the number of vias and the channel height was achieved. No emphasis was put on reducing the wire length in certain layers. The quality measure for soft constraints was set such that all nets with doglegs are rejected.

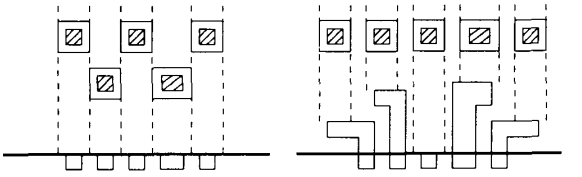


Fig. 14. If spacing between the territories is too small (left) there is not enough room for the vias. A *fan-out* pattern (right) is required to shift the terminals apart until the territory overlaps are removed.

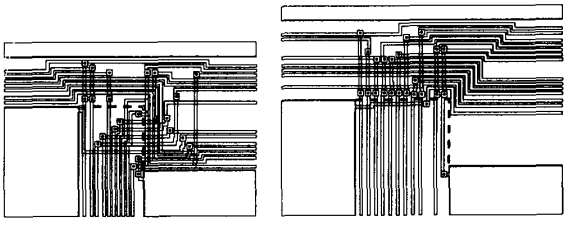


Fig. 15. Two versions of the same channel intersection in a real-life chip. The vertical terminals (left) enable a better area utilization which results in a considerable decrease of channel width. The channel boundary is marked by the dashed line.

TABLE I
Comparison of Various Routers for "Deutsch's Difficult Example"

Terminal pitch = 4 sources: [14,5,16,3]				
Reference		no. of tracks	compacted height	no. of vias
[6]	("dogleg")	21	49	n.a.
[18]	("efficient")	20	47	308
[15]	("greedy")	20	48	347
[1]	("hierarchical")	19	46	354
[14]	("YACR2")	19	47	287
[5]	("compacted")	19	45	n.a.
[16]	(manh.)	-	46	271
[16]	(octa.)	-	44.33	271
[3]		19	42	275
	This work	-	45	265

Although the main strength of the algorithm is gridless variable width routing, it also produces very competitive results for "traditional" two-layer gridded channels. Table I shows the results of our routing of "Deutsch's Difficult Example" [6] compared to a number of well-known routers. The design rules of [5] were used for all results in this table.

The channel height of this gridless router depends very much on the terminal pitch. With a 2.0 terminal pitch the territories overlap and a massive fan-out pattern is required. The minimum pitch to prevent overlapping territories is 3. In general the results will improve with increasing terminal pitch because better paths come available. This effect is illustrated in Table II.

Recently two new "more difficult examples" were suggested by the author of the original problem [7]. They are derived from the original channel by shifting the top half of the channel either half a terminal pitch to the left or to the right. In this way the number of vertical constraints increases considerably. To our knowledge no results for these "more difficult examples" have been published up

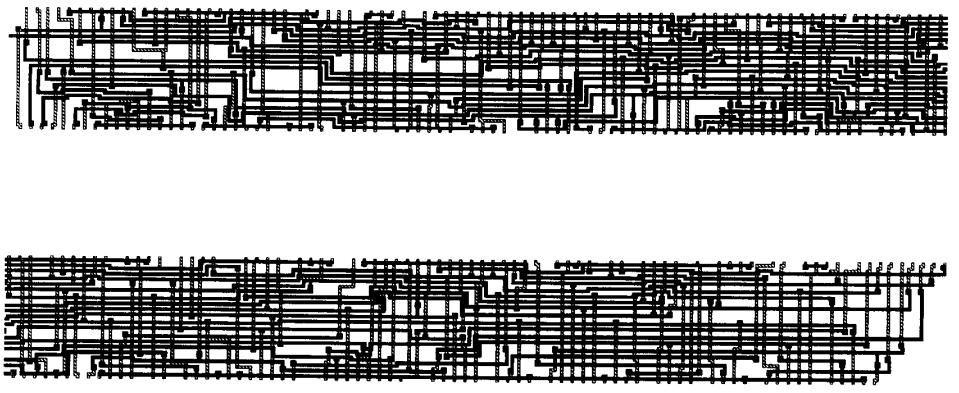


Fig. 16. Our result for Deutsch's new difficult example with $+\frac{1}{2}$ pitch shift of top side.

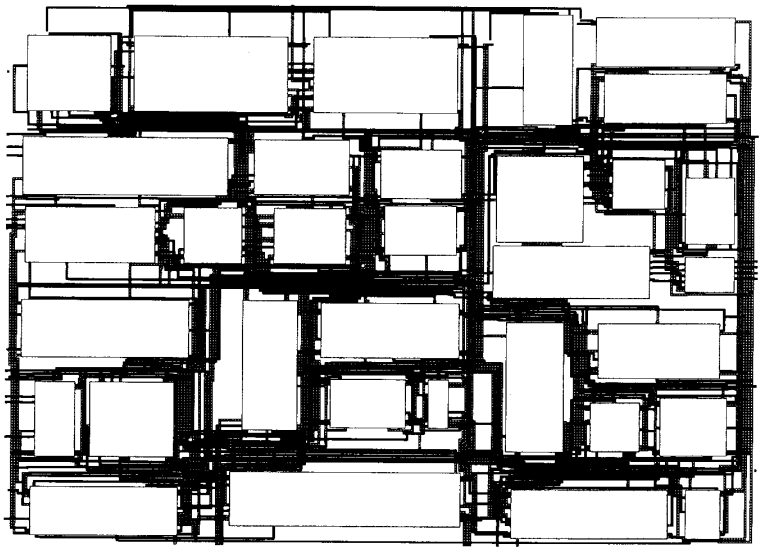


Fig. 17. MCNC benchmark circuit AMI33 routed by this channel router.

TABLE II
HEIGHT OF "DEUTSCH'S DIFFICULT EXAMPLES" FOR VARIOUS TERMINAL PITCHES

terminal pitch	original channel	+ ½ pitch shift	− ½ pitch shift
2	353	356	357
3	48	52	54
3.5	48	49	53
4	45	46	46
5	43	45	46
6	43	44	44
7	42	43	43
10	41	41	41

to now. Tables II, III, and Fig. 16 show our results for these channels.

We also routed the benchmark circuits used at the 1988 MCNC workshop on placement and routing with this channel router. Although the results depend very much on the placement of the blocks, the channel router contributed considerably to the good results (see Table III). Fig. 17 shows a plot of the AMI33 benchmark circuit.

TABLE III
RESULTS ON 1988 MCNC MACRO-CELL BENCHMARK CIRCUITS

system	AMI33		XEROX	
	area	no. of vias	area	no. of vias
This work	2.60	967	26.56	925
Seattle Silicon	2.73	862	28.63	1235
VITAL	3.21	763	31.71	1029
MOSAICO	3.10	885	29.01	1173
BEAR	3.05	1092	28.47	1101

In Table IV the results of a number of channels are shown. The channels marked 6) were obtained from a large real-life chip which was processed in a double-metal 1.6-μm CMOS process. They have rugged channel boundaries and variable-width wires for power and clock wires. A few "fan-out" wire patterns were required in most of them to separate territories or to route overlapping

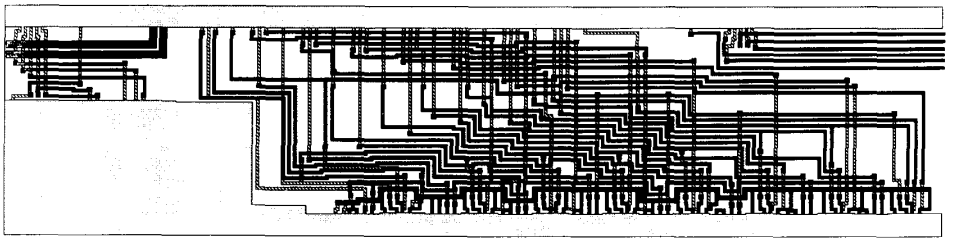


Fig. 18. A real-life channel with overlapping terminals.

TABLE IV
DATA AND RESULTS OF VARIOUS CHANNELS

channel	no. of nets	no. of terms	channel height	no. of vias	total wire length 1)	accept. ratio 2)	CPU-time 3)
diff. 4)	72	302	48	264	12955	61%	11s/16s
diff. +½ 4)	72	302	52	271	13417	16%	42s/45s
diff. -½ 4)	72	302	54	272	13616	10%	44s/47s
CK2 5)	21	75	271	57	17456	100%	2s/5s
chan1 6)	40	94	7)	85	2343	95%	6s/8s
chan2 6)	17	38	7)	21	1205	92%	4s/5s
chan3 6)	44	99	7)	92	1771	95%	7s/9s
chan4 6)	48	142	7)	130	2317	63%	14s/18s
chan5 6)	56	114	7)	3	2346	100%	5s/5s
chan6 6)	11	22	7)	18	3146	93%	2s/3s
chan7 6)	10	73	7)	61	5890	85%	5s/7s

1) The total wire length after straightening.
2) The acceptance ratio of the net scheduler. This is the number of successful attempts as percentage of the total number of attempts to route a net.
3) CPU-time on an Apollo DN-3000 work station (≈1 MIPS) including all overhead such as database I/O. The first value is for routing without wire straightening. The second value includes wire straightening.
4) Deutsch's Difficult Example, terminal pitch = 3.
5) The second example channel published as Fig. 6 in [4]. The units are millimeters.
6) Variable wire width channels, see text. The units are microns.
7) Irregular channel boundaries.

terminals. Fig. 18 shows one of these channels. As can be derived from Table III the CPU-time consumption depends very much on the complexity of the problem. If many obstacles are present the number of failed attempts to route a net will increase due to hard and soft constraints. The execution time also depends on the number of events (contour-bends) present in the channel. The contour bends are caused by terminals. We used a number of test channels to measure the CPU-time consumption under controllable conditions. The channels were generated using a random generator such that they only differ in the number of terminals and as little as possible in other properties. The boundaries of the channels are packed with terminals at minimum spacing. No significant differences were detected between channels with 2, 3, or 4 terminals per net. Fig. 19 shows the results of the tests.

IX. Conclusions

An n-layer gridless variable-width channel router has been presented which can take advantage of different design rules in each layer. The contour-based routing framework has proven to be very effective and robust. Due to this framework vertical constraints can be avoided by jogging in a vertical layer. This gives the router some of the characteristics of a river router. Its immunity to vertical constraints makes the algorithm especially useful for complex channels with terminals at nonaligned positions. One of the major characteristics of the algorithm is that

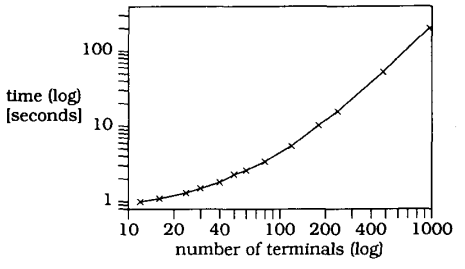


Fig. 19. The total execution time versus the number of terminals in random channels. Measured on a DN-3000 workstation.

only very few constraints are imposed on the channel. Terminals are allowed at any position and in any layer on the (rugged) channel boundary. The wire width of each net can be specified. Weight parameters enable a tradeoff between the number of vias, the wire length in each layer and the channel height. Very competitive results have been achieved for various types of channels.

Acknowledgment

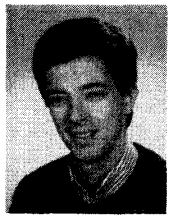
The author would like to thank Prof. Ralph Otten and Dr. Hong Cai for their valuable comments on this topic.

References

[1] M. Burstein and R. Pelavin, "Hierarchical channel router," in *Proc. 20th Design Automation Conf.*, 1983, pp. 591-597.
[2] H. Cai, H. Verheyen, and P. Dewilde, "An automatic routing system for general cell VLSI circuits," in *Proc. Custom Integrated Circuits Conf. '85*, May 1985, pp. 68-71.
[3] C. K. Chen and D. N. Deutsch, "Improved channel routing by via minimization and shifting," in *Proc. 25th Design Automation Conf.*, 1988, pp. 677-680.
[4] H. H. Chen and E. S. Kuh, "Glitter: A gridless variable-width channel router," *IEEE Trans. Computer-Aided Design*, vol. CAD-5, pp. 459-465, Oct. 1986.
[5] D. N. Deutsch, "Compacted channel routing," in *Proc. Int. Conf. Computer-Aided Design '85*, Nov. 1985, pp. 223-225.
[6] —, "A dogleg channel router," in *Proc. 13th Design Automation Conf.*, 1976, pp. 425-433.
[7] —, "Two new and more difficult channel routing problems," *IEEE Trans. Computer-Aided Design*, vol. CAD-8, p. 448, Apr. 1989.
[8] L. P. P. P. van Ginneken, "Gridless routing of general floor plans," in *Proc. Int. Conf. Computer-Aided Design '87*, November 1987, pp. 30-33.
[9] P. Groeneveld and H. Cai, "The Delft placement and routing system," in *Proc. Int. Workshop on Placement and Routing*, May 1988, p. 6.2.
[10] P. Groeneveld, "On global wire ordering for detailed routing," in *Proc. 26th Design Automation Conf.*, June 1989, pp. 155-161.
[11] A. Hashimoto and S. Stevens, "Wire routing by optimizing channel

assignment within large apertures," in *Proc. 8th Design Automation Conf.*, 1971, pp. 155–169.
[12] C. H. Ng, "A gridless variable-width channel router for macro cell design," in *Proc. 24th Design Automation Conf.*, 1987, pp. 633–636.
[13] D. B. Polk, "A three-layer gridless channel router with compaction," in *Proc. 24th Design Automation Conf.*, 1987, pp. 146–151.
[14] J. Reed, A. Sangiovanni-Vincentelli, and M. Santomauro, "A new symbolic channel router: YACR2," *IEEE Trans. Computer-Aided Design*, vol. CAD-4, pp. 208–219, July 1985.
[15] R. L. Rivest and C. M. Fiduccia, "A 'greedy' channel router," in *Proc. 19th Design Automation Conf.*, 1982, pp. 418–424.
[16] J. Royle, M. Palczewski, H. Verheyen, N. Naccache, and J. Soukup, "Geometrical compaction in one dimension for channel routing," in *Proc. 24th Design Automation Conf.*, 1987, pp. 140–145.
[17] R. Y. Tsui, "Variable track space channel routing," in *Proc. Int. Conf. on Computer-Aided Design '86*, Nov. 1986, pp. 200–203.
[18] T. Yoshimura and E. S. Kuh, "Efficient algorithms for channel routing," *IEEE Trans. Computer-Aided Design*, vol. CAD-1, pp. 25–35, Jan. 1982.
[19] N. P. Chen, "Building block routing—A symbolic approach," in *Proc. Custom Integrated Circuits Conf. 88*, May 1988, p. 11.2.

*



Patrick Groeneveld received the M.S. degree in electrical engineering from Delft University of Technology, Delft, The Netherlands, in 1987. He is currently working towards the Ph.D. degree at Delft University of Technology.

His research interests include algorithms for computer-aided design, in particular placement and routing strategies for VLSI circuits.

Mr. Groeneveld received a Best Paper Award at the 26th Design Automation Conference in Las Vegas, in 1989.