Recursive channel router

W. Heyns and K. Van Nieuwenhove

Silvar-Lisco
Abdijstraat 34
B-3030 Leuven
Belgium

ABSTRACT

This paper describes an extended 'true' VHV-model 3 layer channel routing algorithm. The algorithm is based on the observation that, after having routed a track on either the top or bottom side of the channel, one can define a new channel routing problem and call the router recursively. The routing task then is reduced to making optimal use of a track. Routing of a track is done in 2 steps. For the first step, we will present a segment selection method, which aims at reducing the channel density. The second step further fills up the track and, at the same time, aims at reducing the heights of the vertical constraint graphs.

Introduction

Two different models are being used in 3 layer routing: VHV and HVH. In the VHV model, the router accepts pins on both the 1st and the 3rd layer, and uses the 2nd layer as the main horizontal routing layer. In the HVH model, the router only accepts pins on the 2nd layer, and uses both the 1st and the 3rd layer as horizontal routing layers.

If we take a look at the papers that have been written on 3 layer routing in recent years [1][4][5][9][10], it turns out that there seem to be far more papers on the HVH model than there are papers on the VHV model. This can partly be explained by the fact that channel routers are judged by the number of tracks in which they solve a routing problem as the Deutsch difficult example, and the fact that the HVH model scores very high in this area.

However, it is quite logical to select a VHV model in an environment where the 1st layer is a highly resistive layer, eg. poly. Normally, we will want to minimize the amount of routing on such a layer. It then suffices to notice that the 1st layer is one of the main routing layers in the HVH model. The HVH model only becomes interesting when all 3 interconnection layers are low resistivity layers, eg. metal. Furthermore, in a standard cell environment we might gain more by being able to put 3rd layer feeds on top of the standard cell rows when using a VHV routing model, than we might gain by using a HVH routing model, because this requires the insertion of feed cells in the standard cell rows.

Furthermore, those papers that present a VHV model, do not always present a 'true' VHV model. In a 'true' VHV model, the pin on the 1st layer and the pin on the 3rd layer do not have to belong to the same net. Often, we will find that one assumes that these pins do belong to the same net and that the router can select either pin to connect to. This assumption not only considerably reduces the complexity of the routing problem, it also seriously effects the applicability of the router to 'real world' routing problems. The 'real world' routing problem of a standard cell layout program that generates 3rd layer feeds over standard cell rows, requires a 'true' VHV model router.

In this paper we will present an extended 'true' VHV model routing algorithm. In fact, we could refer to our model as a (VH)H(VH) model, i.e. the 2nd layer is still the main horizontal routing layer, but the 1st and the 3rd layer are also used for horizontal routing.

One of the problems we immediately encounter in a 'true' VHV model, is how to realize connections between a 1st layer and a 3rd layer pin. Because of technological limitations, these usually can not be realized with a single 1st layer to 3rd layer contact. They have to be realized using separate 1st layer to 2nd layer and 2nd layer to 3rd layer contacts (fig.1). It should be noticed that, unless specific actions are taken to avoid them, 1st layer to 3rd layer contacts can appear in both HVH model and VHV model 3 layer routing results. However, most papers on 3 layer routing seem to neglect the technological problem and, hence, the presented routing results often contain 1st layer to 3rd layer contacts [10].

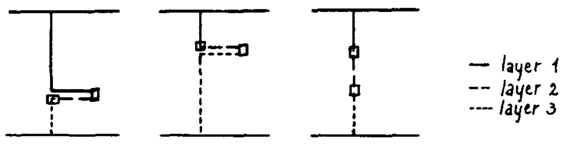


Fig.1: A 1st layer to 3rd layer connection can be solved by placing the 1st layer to 2nd layer and the 2nd layer to 3rd layer contacts either horizontally adjacent, using a planar segment on the 1st layer (a) or the 3rd layer (b), or else by placing them vertically adjacent (c).

Terminology

Before we can start explaining the basic principles on which the recursive router is based, we first have to introduce some definitions.

Given a set of vertical constraint graphs {VCG}, we will construct an extended vertical constraint graph EVCG, by introducing the nodes n_{top} and n_{bottom}, respectively representing the top and bottom channel border, and by adding an edge from n_{top} to every top node of the VCG's and an edge from all the leaf nodes of the VCG's to n_{bottom} (fig.2).

In EVCG we then define for every node n_i, representing segment i, the ascendant weight $w_a(i)$ as the length of the maximum length path from n_{top} to n_i. In the same way, we define the descendant weight $w_d(i)$ as the length of the maximum length path from n_{bottom} to n_i. Notice that the sum of $w_a(i)$ and $w_d(i)$ does only equal the height of EVCG for nodes that are on the critical path of EVCG.

From the definitions above, it should be clear that, when trying to decide which segments to route on the track under consideration, we can only select segments for which $w_a(i)=1$. We will refer to these segments as feasible segments.

Apart from the maximum channel density $d_{channel}$, we also introduce the local density d(i) of a segment $i=[x1_i,x2_i]$ and define it as the maximum channel density between $x1_i$ and $x2_i$.

With respect to channel densities we furthermore introduce the span concept. The span of the channel, $s_{channel}$, is defined as the number of positions in the channel where the maximum channel density $d_{channel}$ occurs. The span of a segment i, s(i), is defined as the number of maximum channel density positions that are crossed by the segment. Notice that the spans are deliberately defined as 'number of positions' instead of 'number of grid positions', to indicate that they do not imply the use of a grid.

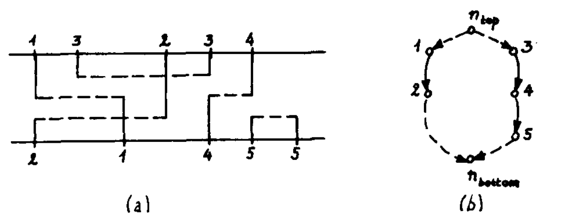


Fig.2: A routing problem (a) and its corresponding EVCG (b).

Recursive router

Recursion

We observed that it is possible to derive a new channel routing problem from a given routing problem in which we have routed a track on either the top or bottom side of the channel. It simply suffices to have the new routing problem inherit the pins to which a connection still has to be made, i.e. completely connected pins are not inherited and leave free positions (fig.3). The coordinates on which we have made a connection to a pin on the non-routing side of the channel, should be marked as blocked coordinates. This is required in order to guarantee that any subsequent step that searches for suitable dogleg or jog positions, will skip such coordinates.

Instead of continuing with the original routing problem, the router can then be called again with the new routing problem being the routing problem to be solved. The routing task itself is reduced to making optimal use of a track. The recursion stops when the remaining routing problem is an empty channel, i.e. when there are no more pins left to be connected.

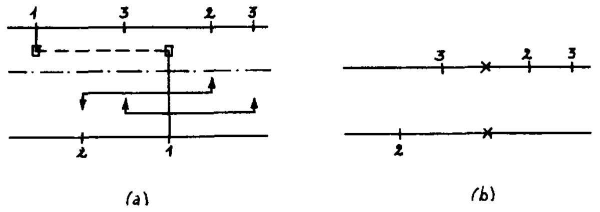


Fig.3: After having assigned segment 1 to the top track in the original routing problem (a), a new routing problem can be derived (b).

A nice advantage of routing algorithms like the 'greedy' router [2], the hierarchical router [3] and the 'Delft' router [6], is that no special processing step is required to solve cyclic vertical constraints. They get solved automatically during the routing process. The reason is that these algorithms, because of the way they work, automatically tend to insert jogs in interconnections and some of these jogs happen to be the jogs required to solve the cyclic vertical constraints.

In routing algorithms that do require a special processing step to solve cyclic vertical constraints, it is common practice to solve them once in the beginning of the routing process and to stick with this solution for the duration of the routing. Simplifying the problem, we could say that to solve a cyclic vertical constraint, we only have to find a suitable dogleg position. The problem is that, as the number of such positions is often limited, we often have to settle for poor solutions.

The algorithm we present requires cyclic vertical constraints to be solved before routing can start. However, due to the recursive approach, we always create new routing problems and, therefore, at every level of the recursion we will generate new solutions for the remaining cyclic vertical constraints. Again oversimplifying things, we could say that, when we search for suitable dogleg positions, we search for positions that are free on both the top and the bottom side of the channel on the layer of the dogleg. Because connected pins leave free positions on the next level of the recursion, it should be noticed that the number of free positions normally increases as we go down in the recursion. Now, since more free positions create more suitable dogleg positions, the probability that we will continuously find better solutions for our cyclic vertical constraints is high (fig.4).

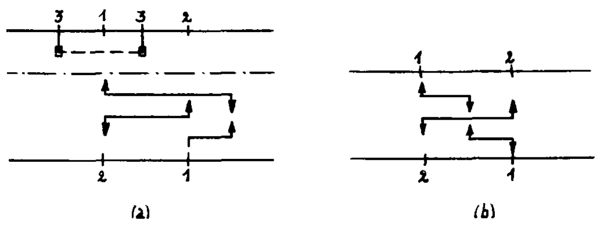


Fig.4: Although one had to settle for an external dogleg in the original routing problem (a), one will find an internal dogleg solution at the next level of the recursion (b).

Track routing

As will have become clear from the previous sections, on every level of the recursion the router only has to route a single track. It is the task of the router to make optimal use of the track.

In channel routing, there are two parameters that determine what routing results can be obtained: the maximum channel density and the maximum vertical constraint graph height. If only one horizontal routing layer is used, it should be clear that no solution can be found with a number of tracks less than the maximum channel density. On the other hand, if no doglegs are allowed, no solution can be found with a number of tracks less than the maximum vertical constraint graph height. Therefore, in our approach, track routing is done in two steps: the first step aims at reducing the maximum channel density and the second step at reducing the maximum vertical constraint graph height.

Step 1 The goal of the first step in the routing is to reduce the channel density. If we want to route a channel in density, we have to make sure that the channel density of the channel in the next level of the recursion is one less than the channel density of the channel on the current level. The problem is which segments from the set of feasible segments to assign to the track under consideration to achieve this goal. In view of the definitions given above, this translates to: from the set of feasible segments, select a feasible subset for which holds that no two segments overlap and *sum*(s(i))=$s_{channel}$. Notice that the subset can and normally does contain segments i for which s(i)=0.

Because of the vertical constraints it is not always possible to find a subset which fulfills the condition given above. Therefore, rather than trying to find a subset which fulfills the condition, we will search for a subset for which *sum*(s(i)) is maximal. It is always possible that this maximum happens to be $s_{channel}$, in which case we have fulfilled the condition above, but it can never exceed $s_{channel}$.

Although the condition given above seems to work in theory, it does not really work in practice. In general, *sum*(s(i)) becomes maximal for a number of different subsets. At first, it might seem as if it does not really matter which subset we pick. However, this is not true. We already indicated that, due to vertical constraints, it is not always possible to find a subset for which *sum*(s(i))=$s_{channel}$. For example, we can easily construct a channel in which none of the segments that account for the positions of maximum channel density, belong to the set of feasible segments, i.e. that $w_a(i)$<>1 for all segments for which s(i)<>0. (fig.5). It should now be obvious that the recursive router can easily create such a situation at one of his recursion levels, by selecting the 'wrong' subsets of segments on the higher levels of the recursion.

The problem with maximizing just *sum*(s(i)), is that this does not take into account the effects of vertical constraints. In order to overcome this deficiency, we introduced a priority scheme which, in its simplest form, will select segment i over segment j,

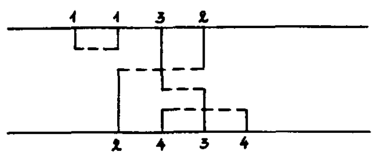


Fig.5: A routing problem in which none of the segments for which $d(i)=d_{channel}$, is a feasible segment when routing the top track.

although $s(i)=s(j)$, if EVCG contains vertical constraint edges from n_i and n_j to respectively n_k and n_l, and $d(k)>d(l)$. This priority scheme aims at having a maximum number of segments with non-zero span in the set of feasible segments at every level of the recursion.

As explained above, the priority scheme is based on a 1-level look-ahead. In practice, we implemented a l-level look-ahead. In this scheme, we calculate for every feasible segment i a priority value p(i). The priority value p(i) is an ordered set of values:

$$p(i)=\{s(i),p_l(i),p_{l-1}(i),..p_0(i)\}$$

The s(i) term still is the span as defined before. The other terms are calculated as follows: for every j in the subtree under n_i for which $w_a(j)=w_a(i)+y$ with $1<y<l$, set $p_x(i)=p_x(i)+1$ for $x=d(j)-(d_{channel}-y)$ if $d(j)>=(d_{channel}-y)$.

Instead of maximizing the sum of the spans, we now maximize the sum of the priority values p(i). To sum p(i) with p(j) we sum every term in p(i) with the corresponding term in p(j). Furthermore, we define p(i) to be greater than p(j), if s(i)>s(j). If s(i)=s(j), then p(i) is greater than p(j) if $p_l(i)>p_l(j)$. If these terms are still equal, then p(i) is greater than p(j) if $p_{l-1}(i)>p_{l-1}(j)$, and so on.

Step 2 While in the first step of the track routing we were concerned with reducing the channel density, in the second step we will be concerned with filling up the track even further while, at the same time, trying to reduce the vertical constraint graph heights. While in the first step we only selected complete segments, i.e. segments beginning and ending on a pin position, here we will look at partial segments, i.e. segments only beginning or ending on a pin position.

An important difference between routing complete segments and routing partial segments is that routing complete segments can only lead to the removal of pins from the next level of the recursion, while routing partial segments will always introduce a pin on this level (fig.6). We will call the position where this pin is introduced, the jog position of the partial segment, as we will always find a jog there in the final routing result.

In principle, there is little difference between finding suitable dogleg positions when we are trying to solve the cyclic vertical constraints, and finding suitable jog positions here. There are two basic constraints to be satisfied:

- at the routing side of the channel the jog position should be a free position, in order to avoid the blockage of a still unconnected pin
- if at the opposite side of the channel we do not have a free position, the vertical constraint to be introduced should not lead to a cycle in the vertical constraint graph

The first constraint only requires a simple check and, as we will see, the second constraint can easily be satisfied.

If we add a new vertical constraint to an existing graph without cycles, this can never lead to cycles if the vertical constraint edge is from a node n_i to a node n_j with $w_a(i)<=w_a(j)$. We can also phrase this as: a cycle can never be generated by introducing an edge from a node n_i to a node n_j with $w_d(i)>=w_d(j)$. The advantage of phrasing it this way, is that we not only avoid cycles, but that, at the same time, we effectively reduce vertical constraint graph heights. Also notice that, although partial segments are not introduced to solve cyclic vertical constraints, it is well possible that, because of partial segments, some of the cyclic vertical constraints that were present at one level of the recursion, disappear from the next level of the recursion (fig.7).

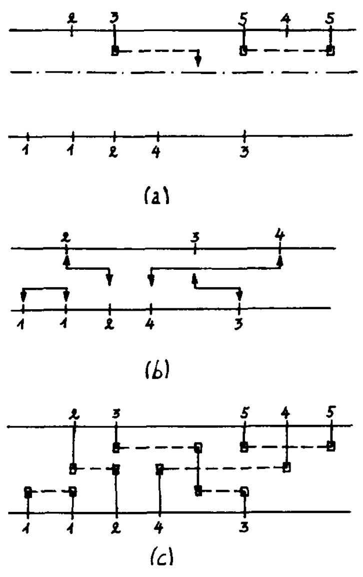


Fig.6: The partial routing of segment 3 on the top track in the original routing problem (a) introduces a pin at the next level of the recursion (b) at the position of the jog (c).

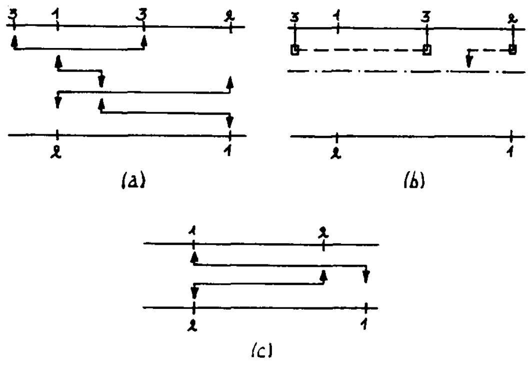


Fig.7: Although none of the segments of the original solution of the cyclic vertical constraint (a) is routed on the top track, the partial routing of segment 2 (b) has as side effect that there is no more cyclic vertical constraint at the next level of the recursion (c).

In practice, we will first scan the channel from left to right for pins which still have to be connected to the right. For each such pin we will try to find a suitable jog position. If such a position can be found, we will allocate the partial segment to the track. In case multiple suitable jog positions exist, we will take this position for which $w_d(i)-w_d(j)$ is maximal. After having arrived at the right end of the channel, we will repeat the whole process, this time scanning the channel from right to left searching for pins which still have to be connected to the left.

In a three layer environment, we simply repeat the previous steps for the secondary horizontal routing layers. In a (VH)H(VH) model this means that we repeat these steps for respectively the 1st and the 3rd layer. Notice that, when routing on the 1st or the 3rd layer, we have to check for horizontal constraints: we can never cross a coordinate on which there still is an unconnected pin on the routing layer on the routing side of the channel.

Comparitive analysis

It is interesting to compare the track routing algorithm presented in this paper with other routing algorithms.

In the left-edge algorithm, segments get assigned to a track in the order in which they are encountered while scanning the channel from left to right. This leads to optimal results in the absence of vertical constraints, but in the presence of vertical constraints we can only hope that the results will not be too bad. Every time the left-edge algorithm finds a segment that it can assign to the track being routed, it will do it without evaluating the effects this has on the rest of the routing.

In the greedy routing algorithm, we still scan the channel from left to right, but instead of routing only a single track, we now route all tracks at the same time. Every pin we encounter introduces a new segment on an empty track if it can not be connected to a segment previously introduced by another pin of the same net. New, compared to the left-edge algorithm, is that segments are not restricted to a single track. Before going to the next grid, a jog can be introduced on the current grid to shift a segment to a higher (lower) track, if the majority of still unconnected pins of the net can be found on the upper (lower) side of the channel. Basically, we have to conclude that a greedy router is governed by a set of heuristic rules which aim at some local optimization, rather than by a global objective.

There exist variations of both the left-edge algorithm and the greedy routing algorithm, in which we start from a place of highest channel density and then work towards the outer ends of the channel. These variations acknowledge that it is better to start from a problem area than to assign segments more or less at random. However, focusing on one hot spot area, still does not present a global approach to the routing problem.

In the hierarchical router we reduce the channel routing problem to a global routing problem in a 2xn routing space. There is a clearly defined objective and this objective is to route every net with minimum cost. Again, this is obviously rather a local than a global objective.

We believe that the approach taken in the 1st step of the track routing, represents a global approach. At every level of the recursion we have a clearly defined objective, which is that the channel density at the next level of the recursion, should be one less than the channel density at the current level. However, with the method we presented, the actions required to reach this local goal are evaluated in the context of the total routing problem by means of the proposed priority scheme. This priority scheme is introduced to take into account the effects of the vertical constraints on the routing problem. Therefore, while we are trying to reach our local goal, we are at the same time trying to maximize the chances that we can also meet this goal at the next levels of the recursion.

With respect to the partial segments that are routed in the 2nd step of the track routing algorithm, it should again be stressed that these are not introduced to solve cyclic vertical constraints. Cyclic vertical constraints should be solved before the track routing starts. Compared to, for example, the greedy routing algorithm, we therefore have better control over how they are solved. Furthermore, due to the recursive nature of the algorithm, we can continuously propose better solutions.

Results

As so many others before us, we used the Deutsch difficult example to evaluate different algorithms and to develop the recursive router. While evaluating other algorithms it became clear that we often had to fine tune implementations in order to get the reported results. Changing a simple 'greater than' condition to a 'greater than or equal to' condition, sometimes drastically influenced the results.

Furthermore, it is clear that a router should not only work well for the Deutsch difficult example, but for any routing problem, i.e. its performance should be insensitive to particularities of the input data. For example, if we present to a router the Deutsch difficult example turned upside-down, we should get comparable results. We noticed that the Deutsch difficult example usually seems to be even more difficult than it already was, when it is turned upside-down.

In our implementation of the recursive router we can route according to the following sequences:

- go top-down (TD)
- start at the top and alternate between top and bottom (TD+A)
- go bottom-up (BU)
- start at the bottom and alternate between bottom and top (BU+A)

First of all, we should mention that, although we implemented a 3 layer router, we treated the Deutsch difficult example as a 2 layer routing problem, i.e. we did not allow any routing on the 3rd layer.

Table 1 shows our results if only step 1 of the algorithm is used. It clearly demonstrates the influence of the lookahead approach. Notice that these results are obtained without partial segments, but that doglegging on pin positions was allowed. As far as we know, no solution in 19 tracks has ever been reported for this case.

Table 1a

	TD	TD+A	BU	BU+A
l=0	21 tracks	22 tracks	23 tracks	22 tracks
l=1	21 tracks	21 tracks	22 tracks	21 tracks
l=2	20 tracks	21 tracks	22 tracks	21 tracks
l=3	20 tracks	21 tracks	22 tracks	21 tracks
l=4	20 tracks	21 tracks	22 tracks	21 tracks
l=5	20 tracks	21 tracks	22 tracks	21 tracks

Table 1b

4 x	20 tracks
12 x	21 tracks
7 x	22 tracks
1 x	23 tracks

Table 2 shows the results with the full blown version of our algorithm. It clearly demonstrates the effectiveness of step 2 of the algorithm. It is well known that the Deutsch difficult example has a channel density of 19 tracks. Table 2 shows that we found different 19 track solutions (fig.8).

Table 2a

	TD	TD+A	BU	BU+A
l=0	20 tracks	19 tracks	20 tracks	21 tracks
l=1	20 tracks	19 tracks	21 tracks	21 tracks
l=2	20 tracks	19 tracks	21 tracks	22 tracks
l=3	20 tracks	20 tracks	21 tracks	22 tracks
l=4	20 tracks	19 tracks	20 tracks	22 tracks
l=5	20 tracks	19 tracks	20 tracks	22 tracks

Table 2b

5 x	19 tracks
10 x	20 tracks
5 x	21 tracks
4 x	22 tracks

For the sake of completeness, and to demonstrate that we have implemented a 'true' VHV model 3 layer router, we also included (a part of) a result generated by our router on a 3 layer routing problem (fig.9).

Conclusions

In this paper we presented a recursive channel router. Due to the recursive nature of the algorithm, the router always has to route only a single track. This is done in two steps. In the first step we try to achieve that the channel density at the next level of the recursion is one less than the channel density at the current level. In the second step we will fill up the track even further, while trying to reduce the maximal vertical constraint graph height. The merits of this approach were indicated and it was shown that very good routing results are achieved.

It should be noticed that the only intention of this paper was to present the basic principles of the recursive channel router. The fact that we did not discuss problems that are specific to a gridless environment with shaded areas, does not imply that they can not be handled by our algorithm. Our current implementation is gridless and supports shaded area routing. Last but not least, contour compaction has been implemented as a completely separate module.

Besides a 2 layer routing version, we also generated a 2 1/2 layer routing version starting from the current 3 layer version. For the sake of clearness, we should indicate that we define a 2 1/2 layer router as a 3 layer router which will only use the 1st routing layer to connect to 1st layer pins, i.e. a 1st routing layer segment can only contain a single 1st layer to 2nd layer contact. To shift from 3 layers to 2 1/2 layers, we only had to be able to handle n-pin segments instead of 2-pin segments in the 1st step of the track routing, as doglegging on 1st layer pins has to be restricted.

References

[1] W. Heyns, 'The 1-2-3 routing algorithm or the single channel 2-step router on 3 interconnection layers', 19th DAC, pp. 113-120, 1982

[2] Ronald L. Rivest, Charles M. Fiduccia, 'A "greedy" channel router', 19th DAC, pp. 418-424, 1982

[3] Michael Burstein, Richard Pelavin, 'Hierarchical channel router', 20th DAC, pp. 591-597, 1983

[4] Yun Kang Chen, Mei Lun Liu, 'Three-layer channel routing', IEEE Transactions on computer-aided design, vol. CAD-3, No. 2, pp. 156-163, april 1984

[5] Peter Bruell, Paul Sun, 'A "greedy" three layer channel router', ICCAD '85, pp. 298-300, 1985

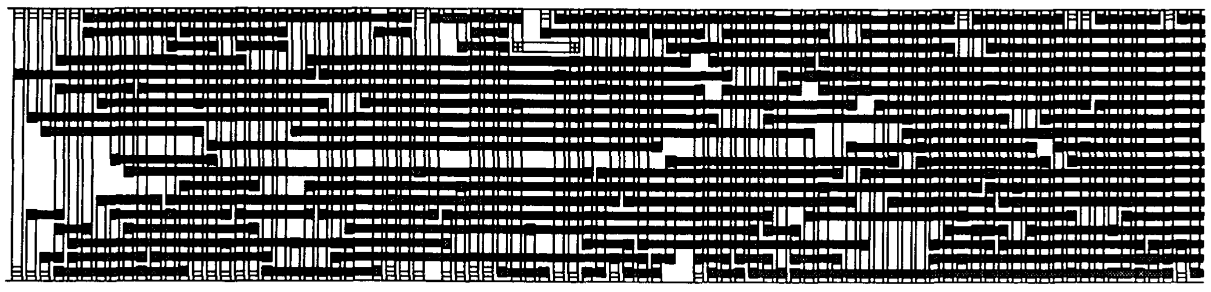
[6] H. Cai, P. Dewilde, 'Automatic routing in a general cell environment', The integrated circuit design book, Delft University Press, pp. 3.1-3.24, 1986

[7] Frieder V. Keller, D. A. Mlynski, 'A graph-theoretical channel-router for constraint-loop-problems', IEEE International Symposium on Circuits and Systems, pp. 311-314, 1986

[8] Howard H. Chen, Ernest S. Kuh, 'Glitter: A gridless variable-width channel router', IEEE Transactions ons computer-aided design, vol. CAD-5, No. 4, pp. 459-465, 1986

[9] Howard H. Chen, 'Trigger: A three-layer gridless channel router', ICCAD '86, pp. 196-199, 1986

[10] Jingseng Cong, D. F. Wong, C. L. Liu, 'A new approach to the three layer channel routing problem', ICCAD '87, pp. 378-381, 1987



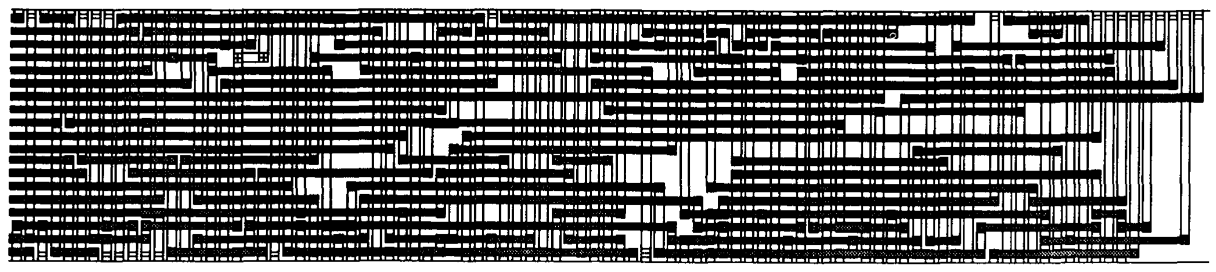


Fig.8: A 19 track solution for the Deutsch difficult example found by the recursive router.

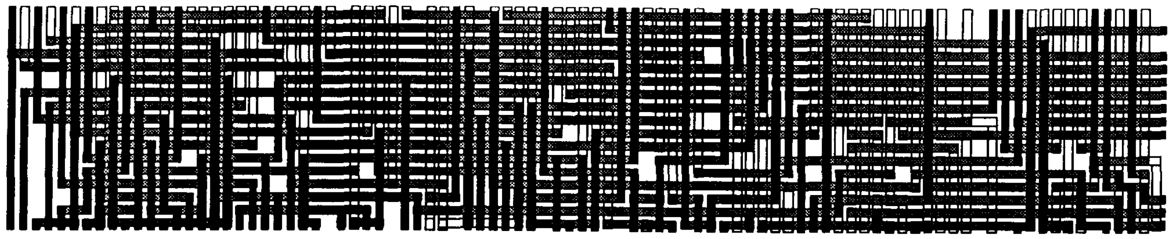


Fig.9: A 3 layer routing example routed by our 'true' VHV model 3 layer routing recursive router implementation.