tu/e
Research Report
ISSN 0167-9708
Coden: TEUEDE

Eindhoven
University of Technology
Netherlands

Faculty of Electrical Engineering

Gridless Routing for Generalized Cell Assemblies: Report and User Manual

by
L.P.P.P. van Ginneken

EUT Report 87-E-180
ISBN 90-6144-180-3
September 1987

Eindhoven University of Technology Research Reports
EINDHOVEN UNIVERSITY OF TECHNOLOGY

Faculty of Electrical Engineering
Eindhoven The Netherlands

ISSN 0167- 9708 Coden: TEUEDE

GRIDLESS ROUTING FOR GENERALIZED CELL ASSEMBLIES

Report and User Manual

by

L.P.P.P. van Ginneken

EUT Report 87-E-180
ISBN 90-6144-180-3

Eindhoven
September 1987

COOPERATIVE DEVELOPMENT OF AN INTEGRATED, HIERARCHICAL

AND MULTIVIEW VLSI DESIGN SYSTEM WITH DISTRIBUTED

MANAGEMENT ON WORK STATIONS.

(Multiview VLSI-design System ICD)

code: 991

DELIVERABLE

Report on activities: 5.3.C: Develop place- and route concept for logic family, and 5.3.D: Implement totally integrated cell generator and place- and route scheme.

Abstract: A new routing system for the general custom cell layout style is presented. The main features of the system are: construction of a floor plan topology by slicing, global routing by means of a Steiner tree heuristic and gridless channel routing with rigorous contour compaction, which allows variable width wires and easy adaptation to the design rules. The router handles arbitrarily sized cells and takes advantage of irregular channel boundaries. Power and ground lines are routed planarly with variable width. An O(nlogn) algorithm for making contours irredundant is given. Benchmarks of the channel router and application to a small chip are included.

deliverable code: WP 5, *task:* 5.3, *activities* C and D.

date: 21-08-1987

partner: Eindhoven University of Technology

author: L.P.P.P. van Ginneken

CIP-GEGEVENS KONINKLIJKE BIBLIOTHEEK, DEN HAAG

Ginneken, L.P.P.P. van

Gridless routing for generalized cell assemblies: report and user manual / by L.P.P.P. van Ginneken. - Eindhoven: University of Technology. - Fig. - (Eindhoven University of Technology research reports / Faculty of Electrical Engineering, ISSN 0167-9708; 87-E-180)
Met lit. opg., reg.
ISBN 90-6144-180-3
SISO 664.3 UDC 621.382:681.3.06 NUGI 832
Trefw.: elektronische schakelingen; computer aided design.

PREFACE

The work published in this report builds on the research of Ralph Otten in the area of layout design performed at Eindhoven University and IBM Yorktown during the early eighties. Together we presented the philosophy of his approach at the ICCD 84 conference in a paper called "Stepwise Layout Refinement" [3].

Stepwise refinement is a technique that has been shown to be effective in the development of computer programs. Niklaus Wirth formulated it explicitly in a now famous paper [9]. In this paper he viewed the design of a structured program as a sequence of refinement steps. Starting with clear problem statement, specifying the relation between the input and the output data, the task is progressively refined, by decomposing it into subtasks, each having an equally clear specification.

The principles of stepwise refinement obviously apply to any complex design task following a top down strategy rather than a process of combining independently developed subdesigns. Stepwise refinement can also be viewed as postponing implementation decisions. Each decision should leave enough freedom to following stages to satisfy the constraints it created, and at the same time rearrange the available data such that further meaningful decisions can be taken.

Presently I'm working towards an integrated floor planning and cell generation system according to these ideas. This routing program would be the last program in the sequence of designing programs. Some of this work was published before in [4] and some of it will be published in [10] at the ICCAD conference in Santa Clara.

I would especially like to thank Ralph Otten, whose ideas I found to be very valuable and inspiring, for introducing me into this area. Also I would like to thank Reinier van den Born for his patience in helping me formatting this manuscript. This work was supported by ESPRIT project 991, and by the Foundation F.O.M. under project nr. EEL 33.0417

Lukas van Ginneken

Eindhoven, 21 August 1987

CONTENTS

LIST OF FIGURES

1. LAYOUT DESIGN IN A SILICON COMPILER

Modern Integrated Circuit technology permits the design and production of increasingly complex circuits at low cost. Large designs typically contain hundreds of thousands of transistors, and designs of more than one million devices are already possible. This enormous increase in scale creates a design problem: design cost and design time exceed production cost and time.

To cope with these problems silicon compilers are being developed. A silicon compiler is a program that translates a functional specification of an integrated circuit into a layout design. Fig.1 gives a schematic flow diagram of a silicon compiler.

The silicon compiler accepts a functional description of the chip in formulated as an algorithm. The data flow optimization makes a mapping of this algorithm to the necessary hardware. This hardware can be optimized by logic optimization tools. Then cells are then generated and placed. The router has to connect the various cells together.

Since it is not possible to fit all designs into a single methodology it necessary to use an open system concept. This means that certain programs can be used independently, and that a reconfiguration of programs must be possible.

1.1 The output: a layout

The ultimate task of a layout design system is to produce a *layout*, a set of data that uniquely and completely specifies the geometry of the circuit. The term *mask* will be used for each plane with a pattern. A layout is translated into a sequence of processes that selectively change the characteristics of the silicon according to those patterns, thus realizing the functional specification available as input to the layout design procedures. For present day technologies the geometrical specification of eight to fifteen masks suffices to specify the layout.

Lithographic techniques have their limitations. Quite often only orthogonal artwork is acceptable. This leads to regions that are unions of iso-oriented rectangles. Rarely is a restriction to rectangles and combinations thereof detrimental, whereas the cell design algorithms profit from such a restriction.

The rectangle constraint is also accepted for the cells, and for the entire chip. Consequently, the floor plan will be a rectangle dissection, i.e. a rectangle subdivided into nonoverlapping rectangles. Choosing rectangles as the only constructs in the repertoire simplifies the formulation of design decisions, and lowers the complexity of deriving these decisions.

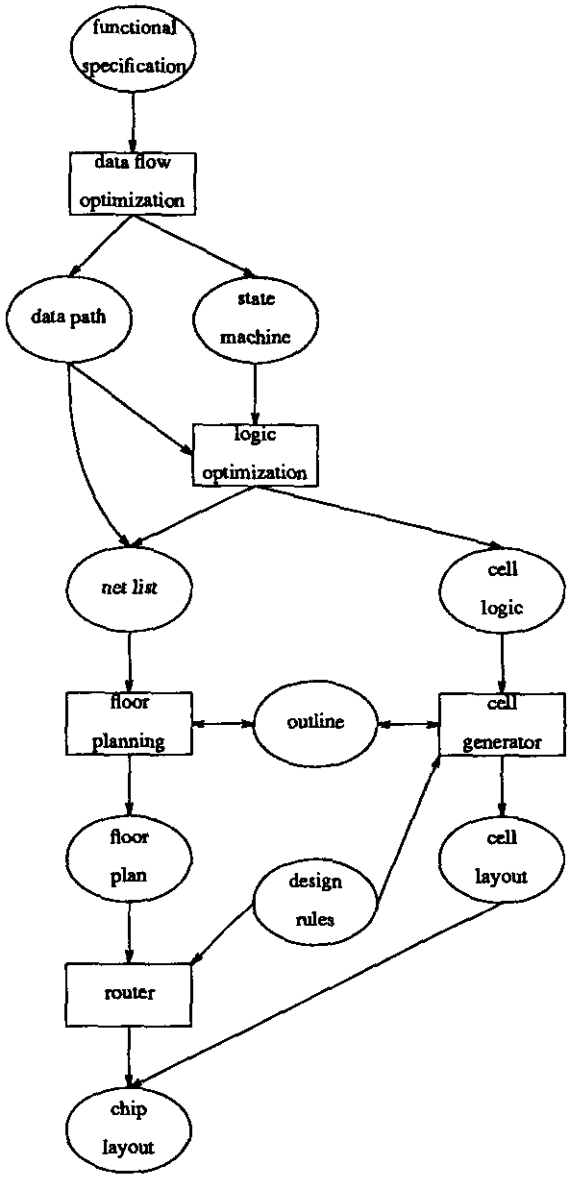


Figure 1. Schematic flow chart of a silicon compiler

1.2 Design rules

To improve chances for successful integration of the circuit, and increase yield when the circuit goes into production, patterns are required to satisfy certain rules, the *design rules*. Two classes of rules can be distinguished: numeric rules quantifying extensions of, and spacings between patterns in a mask and in combinations of masks, and structural rules, enforcing and prohibiting certain combinations. The numeric rules are almost exclusively specifications of lower bounds, because it is assumed that the layout design techniques will try to keep the total chip small. The router has been designed to take

maximum advantage of these rules by compaction.

1.3 Data representation

The single most important consideration in designing complex systems is conceptual integrity. An important aspect of this integrity is how to store the data of a design between the various stages. It is, for instance for simulation purposes, necessary that the results of the router are stored in way compatible with previous design stages.

The use of an open system concept means that a good routing program should be usable for almost any routing task. The input should be formulated in general terms, devoid of any irrelevant data. The open system concept makes flexible and extendable program interface standards necessary. For the net list the ICD standard [14] was adopted, while for the specification of masks the LDM standard [12] was used.

2. THE ROUTING SYSTEM

2.1 Design goals

One of the most time consuming tasks in manual layout design is the design of wiring. The router is probably in any automatic layout design system the most time saving program. The single most important design goal for this router was flexibility. The router has to be able to deal with the following conditions:

— Building cells with arbitrary rectangular shapes, and with terminal positions anywhere at the boundary.

— Routing on two layers, say poly and metal, with different design rules.

— Planar power and ground routing with variable width.

— *100% completion, under mild restrictions.*

Of course the resulting layout is required to be correct by construction. The system is allowed to move the cells to adapt the wiring space.

2.2 An overview

The system accepts a net list, a set of design rules and a layout file as input. The layout file contains the layout of the chip without wiring. This enables the user to modify the floor plan with an interactive layout editor. Also the positions of the terminals of the cells are specified in the layout file. To complete the description of the chip, the output of the wiring program is simply appended to this layout file.

As can be seen in Fig.2 the routing system consists of 3 programs named "entry", "global" and "local". The programs are written in portable Pascal, and communicate through files.

"Entry" parses the two input files, containing the layout description, and the net list. Then a slicing structure is (re)constructed from the layout geometry by a shrink factor technique. It writes two files: "florplan" contains the information about the floor plan topology and geometry, "mod2net" contains a list of terminals to be connected.

"Global" reads those files, and constructs a routing model of the floor plan. It finds for each net the Steiner tree. The nets are decomposed into two terminal segments and assigned to channels. These two terminal segments are stored in the file "channels". Extra pins are introduced where the segments are connected together. These pins are stored in the file "mod2net.app".

"local" processes the channels in a bottom up order. The wires in each channel are compacted before the next channel is routed. In contrast to [5] we used a more rigorous

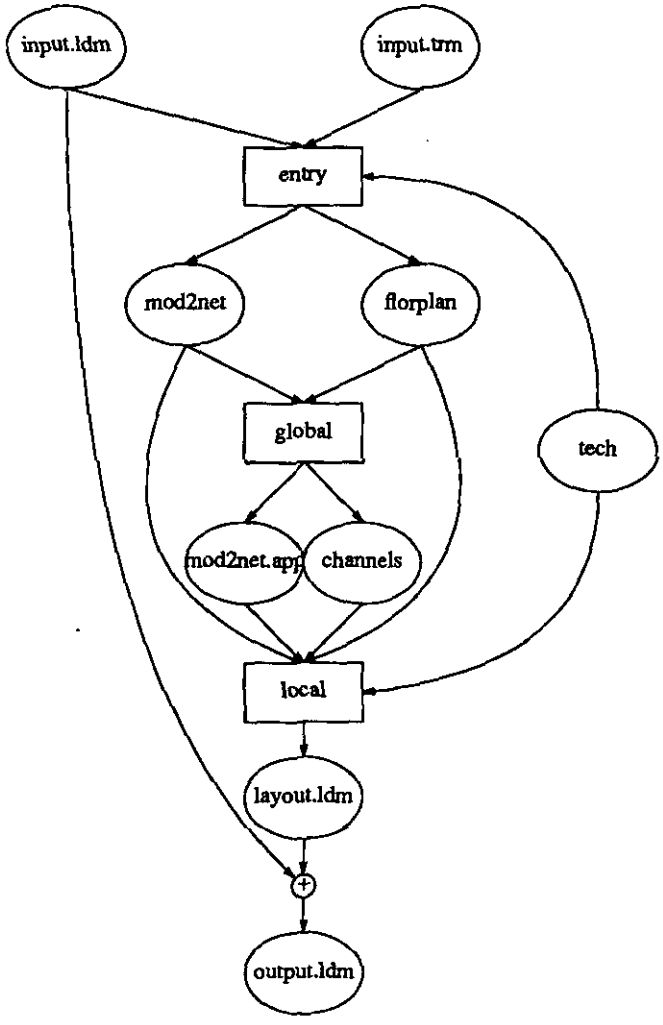


Figure 2. Overview of the routing system

compaction and a method of constructing the slicing structure. The layout of the channels is stored in the file "layout.ldm".

3. SLICING

To avoid channel routing order conflicts the topology of the floor plan is restricted to a *slicing structure* [3]. A slicing structure is a rectangle dissected by a parallel *slicing lines* into smaller rectangles. Each rectangle may in turn be dissected in the perpendicular direction into still smaller rectangles. A slicing structure can be represented by a *slicing tree* in which the leaves are the cells. The slicing structure can thus be seen as a hierarchical structure of the design.

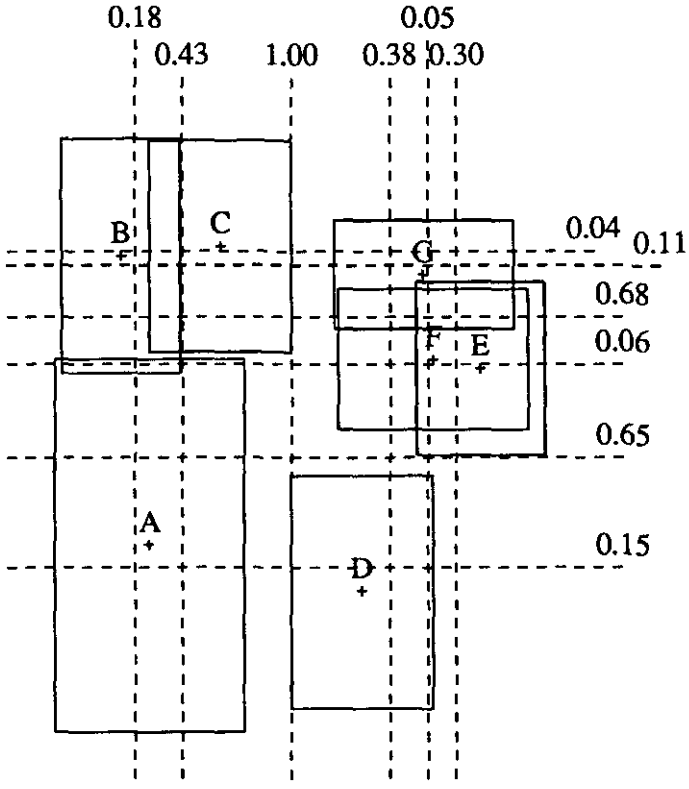


Figure 3. Floor plan scaled by the largest shrink value.

The slicing structure is constructed from the floor plan topology by subdividing the floor plan repetitively by orthogonal straight slicing lines. The selection of the slicing lines is determined by shrink factors. Let (x_i, y_i) be the center of cell i, with dimensions w_i, h_i. The *shrink factor* ζ is a function of a pair of cells, and the slicing direction.

$$\zeta_x(m, n) = \frac{2|x_m - x_n|}{w_m + w_n} \qquad \zeta_y(m, n) = \frac{2|y_m - y_n|}{h_m + h_n}$$

This shrink factor can be interpreted as a scale factor: when two cells are scaled by their shrink factor, they will touch exactly if they are close enough in the other direction.

A vertical slicing line that subdivides a slice into two sets of cells *M*, *N* has a *shrink value* Z.

$$Z_x(M, N) = \min_{m \in M,\, n \in N} \zeta_x(m, n) \quad Z_y(M, N) = \min_{m \in M,\, n \in N} \zeta_y(m, n)$$

This is the maximum scale factor allowed to be able to draw the slicing line as a straight line separating the two sets. The slicing line with the highest shrink value is selected for the next subdivision. After the slicing line has been selected the process is repeated for the subslices on each side. Notice that the set of possible slicing lines changes, and their shrink values have to be recomputed.

In fig.3 a floor plan is drawn together with all possible first slicing lines. The corresponding slicing tree is given in the figure below. The levels in the slicing tree represent the alternating slicing directions.

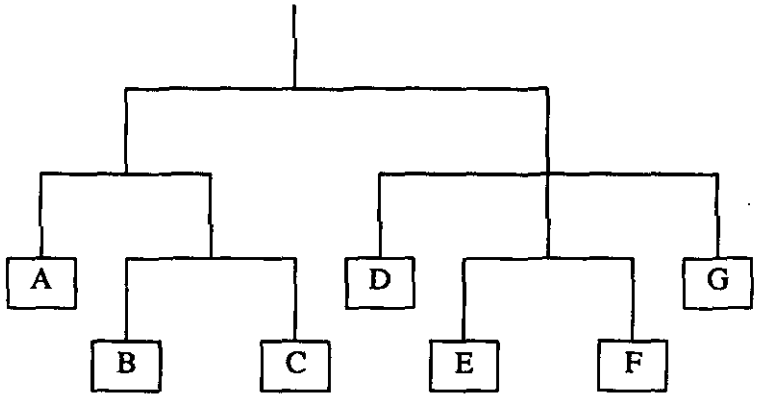


Figure 4. Slicing tree.

Computing the shrink value of a slicing line separating n/2 from n/2 other modules would cost $n^2/4$ operations. Using the following algorithm it is possible to compute all shrink values in only n^2-3n+2 comparisons. The shrink values Z(1) to Z(n-1) belong to the respective slicing lines. The cells are ordered by coordinate.

```
for i = 1 to n-1 do { Z (i) := ∞ ; }

for i = 1 to n-1 do
 {
    for j := i+1 to n do { Z(j) := min (Z(j), ζ (i, j)); }
    Z(i) := ∞;
    for j := i+1 to n do { Z(i) := min (Z(i), Z(j)); }
 }
```

Figure 5. Shrink value algorithm

Since the size of the cells is free, the cells will not fit exactly into the floor plan. The surplus area may be used for wiring since the channel router can handle irregular boundaries. After the channel assignment the channels are routed in a bottom up

sequence such that the the boundaries of the channel to be routed are known. A depth first search through the slicing hierarchy routes the channels in the correct order. The boundaries of the channel are determined by collecting the bounding boxes of all adjacent cells and channels. After the channel routing has been done, the width of the channel is adjusted by shifting some cells. The program updates the coordinates of all terminals, cells and channels that are in the slicing hierarchy to one side of this channel.

4. CHANNEL ASSIGNMENT

For the channel assignment or global routing the method of [4] is used. The objective of the channel assignment is to determine roughly the routes for the wires in a floor plan. The main criterion for the wiring is the length of the wires: all wires are to be kept as short as possible. The use of a slicing structure makes it easy to adapt the width of the channels to match any requirement. Controlling the congestion is therefore not a very important issue.

Channels are areas between the modules that can be used for wiring. The channels correspond one to one to the slicing lines of the slicing structure. The global wiring determines the channels that the wire uses. For each channel the wiring pattern is recorded. Multi terminal wires in the channel will be decomposed into several two terminal segments. Nets can be routed through porous modules to avoid detours.

The problem can be formulated as the problem of finding the shortest Steiner tree in a graph, which is NP-hard. Therefore the channel assignment is done by a Steiner tree heuristic. A new heuristic for solving the Steiner tree problem is presented, based on a spanning tree algorithm and an algorithm for finding an optimal Steiner point for the 3 point case. This heuristic first determines the topology of the Steiner points, by using a shortest spanning tree algorithm. Then it computes the optimal Steiner tree with this topology.

4.1 Routing model

Because of the insignificance of channel density the global routing runs with a background of constant data structures. The distances in the routing model are not changed after a net has been routed. The sequence in which the net are routed is irrelevant.

The incidences between the horizontal and vertical channels are the *T-junctions*. The routing model that is used is a graph $G(V,E)$ of which the vertices V correspond to the T-junctions. Nodes that are adjacent in the channel are connected by an edge. The length of the edges is derived from the input information containing the shapes of the modules.

The terminals to be connected are added to the graph as temporary nodes during the processing of one net. In fig.6 their temporary edges are indicated by dashed lines. Pins occur only at one side of the module and only get the edges that lead to the closest T-junctions on that side. The length of these edges represents as closely as possible the actual Minkowski-1 distance.

The modules may have multiple terminals on their periphery that connect to the same net. In that case a node is added to the graph for every terminal. If the names of those pins compare equal it is assumed that those terminals are internally connected and that

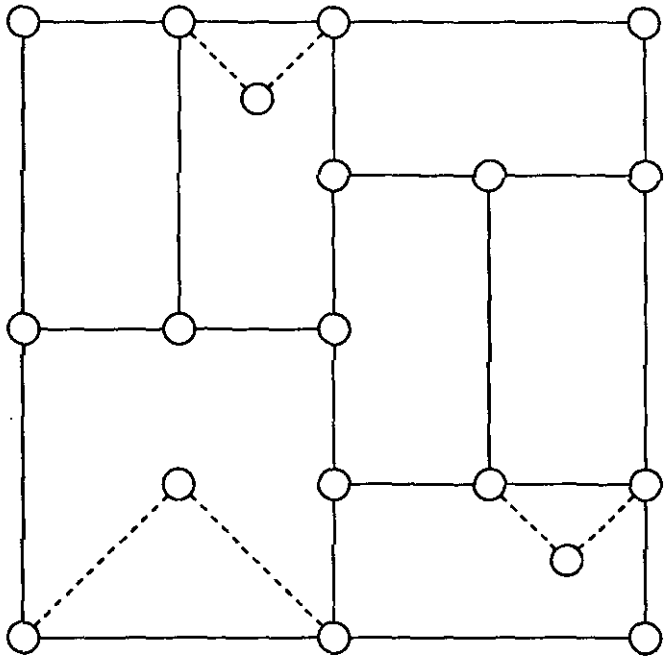


Figure 6. Graph used as global routing model.

only one of them actually needs to be reached. When the names are unequal both terminals will be reached.

In the routing model the nodes corresponding to those terminals are connected by an extra edge of minimum length. The global routing algorithm may use such a connection as a feed thru. This can be controlled by manipulating the length of the edges of the terminal nodes.

The T-junctions are stored in the matrix "Tjunc" of horizontal channels versus vertical channels. If two channels meet in a T-junction the number of this T-junction is stored there, otherwise 0. The channels in this array are ordered by coordinate, giving a symbolic picture of the floor plan. Unordered this array is only dependent on the topology of the floor plan. From this matrix information such as which T-junctions belong to the same channel, neighbor relations between T-junctions, etc. can be derived.

1	2	3	0	4
0	0	5	6	7
8	9	0	0	0
0	0	11	12	13
14	0	15	0	16

Figure 7. The channel incidence matrix Tjunc

For algorithmic efficiency it is necessary that the neighbors of a node can be found easily. Therefore the graph is represented as a node list, with each node having references to neighboring nodes. This means that each edge is present twice in the data structure. The distances between neighboring nodes or T-junctions are derived from the matrix Tjunc and the floor plan. To make the translation from this model back to the floor plan possible each node can represent a terminal in the net list. The pins corresponding with the T-junctions are created when they are needed.

This node list is the data structure that the Steiner tree heuristic runs on. In this way the algorithm for finding a Steiner tree is disconnected from the original problem.

The result of the global wiring is represented as a set of two terminal nets assigned to the channels that contain them. A nice property of these two terminal nets is are immune to changes in the geometry of the floor plan. Any changes in the sizes of the floor plan would not lead to disconnected nets.

4.2 Heuristic for finding the Steiner tree.

Two polynomial time algorithms exist that are commonly used in Steiner tree algorithms. They are the shortest spanning tree algorithm and the algorithm for the three point Steiner tree problem. Most existing methods for finding Steiner trees, exact and heuristic, are based on one of these algorithms. The presented heuristic uses both algorithms in two stages. First it determines the topology of the wire and then the optimal Steiner points for this topology. Even if the topology is not optimal then the shortest wire with this topology may still be close to the optimum.

An algorithm that finds the spanning tree determines the topology. From the spanning tree a binary tree is constructed that "matches" the spanning tree. The optimal spanning tree is found in $O(n^2)$ time. The shortest spanning tree cannot be longer then twice the length of the shortest Steiner tree.

The topology is represented as a binary tree. The leaves and the root of the tree represent the modules connected by the wire. The tree has (n-2) internal nodes corresponding to the possible Steiner points. The Steiner points can in the final result coincide with another node, giving a tree with fewer Steiner points.

For the second stage it is immaterial how this topology was found. The algorithm finds the optimum Steiner points for this topology in $O(n^2s)$ time, where n is the number of nodes in the graph and s is the number of modules to be connected. If the topology found in the first stage is the right topology then the result is the optimum Steiner tree. If it is not then still the shortest tree with this topology is found, which is usually not much longer than the optimum.

The second stage consists of two recursive depth first search procedures that follow the tree. The first one determines the length of all subtrees connecting to any node. From

the information provided by this procedure the length of the tree is determined. It takes advantage of the planarity of the routing graph to improve the computational efficiency. The second procedure traces back the shortest result in this tree, while choosing the Steiner points. During this phase the wire is split in several two terminal segments.

4.3 Power and ground net routing

The power and ground nets are routed planarly by following the slicing structure. The power nets stay to the upper/left side of the channels, and must have a bonding pad in the lower right corner of the chip. The ground nets stay to the lower/right side, and must have a pad in the upper left corner. The width of the power and ground wires is determined automatically.

5. LOCAL ROUTING

Each two terminal segment is realized by a horizontal piece called *trunk* and two vertical pieces called the *branches*. The vertical branches bring the wire into the channel. The horizontal trunk connects the branches together.

Because the terminals are not restricted to grid positions, a branch may have more than one opposite branch. Therefore gridless channel routing introduces more vertical constraints. A problem is that there may be cycles in the vertical constraint graph. In [6] a simple but less conventional approach was used, which guarantees that there will be no cycles.

5.1 Classification

In the approach of [6] the two terminal segments are divided into five classes, indicated by the letters A through E. Branches and trunks are realized in both layers. Fig.8 illustrates the different classes. The bottom layer is realized in polysilicon, the top layer is realized in metal.

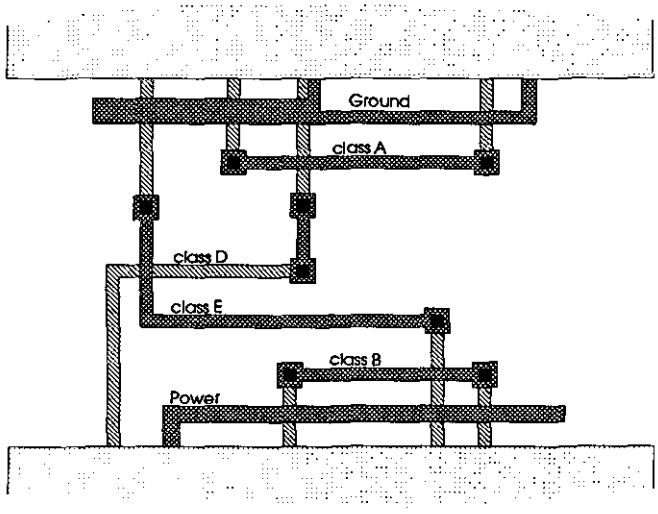


Figure 8. Different classes of two terminal segments.

Class C segments, that have terminals in exactly opposite positions of the channel, are handled as class E or D segments. The power and ground lines are routed at the top and bottom of the channel. In [6] the branches at the top of the channel use another layer than the branches at the bottom. However, to be able to connect to power and ground on the metal layer it is unavoidable that the branches connect to the sides of the cells on the polysilicon layer. Therefore the classes D and E use an extra via hole to change layer.

The segments will be ordered from the bottom of the channel to the top of the channel. The classes can be ordered in the sequence B - E - D - A. Within the classes A and B there are no vertical constraints, and any order is possible. Within class D the segments can always be ordered on the position of the bottom terminal from right to left. The

segments of class E can be ordered from left to right on the position of the top terminal. This always constitutes a valid order, in which the vertical constraints are satisfied.

A practical problem turns out to be that the vertical constraints in the classes E and D tend to form very long chains. This causes trunk ordering algorithms similar to [1] to become ineffective. It is necessary to use jogs to reduce the channel width. The number of vias is reduced by changing some branches or trunks to the other layer.

5.2 Compaction

The rigorous compaction method used introduces jogs anywhere where this is advantageous (see [2]). It also takes maximum advantage of the design rules. It uses different rules on different layers, in stead of deriving a track pitch.

The compaction is done by maintaining two *contours*, one for each layer, that indicate the area occupied by previous trunks. A contour C is defined as a set of *intervals* $[l,r]$ with an associated value t: $C \subset \{(l, r, t) \in R^3 \mid l<r\}$. The *functional value* of a contour C is defined as

$$f_C(x) = \max \{ t \mid (l, r, t) \in C \wedge l \leq x \leq r \}$$

All points (x, y) for which $f_C(x)>y$ are occupied and not available for routing.

Only the trunks and their vias are considered during the compaction; the branches are added once the layout for the trunks is determined. Trunks are processed one by one in the order that is determined by their class. The next trunk, with its associated vias, is placed against this contour, thereby using a minimum of space. The space used by this trunk and its vias is determined by the width ω of the trunk and the design rules, and is represented by the contour D. The set D has an interval for each new rectangle, representing the area occupied by that rectangle, plus its minimum separation design rule. The contour C is initialized with the shape of the channel boundary. When processing the next trunk, the area occupied by the present and previous trunks is described by $C:=C \cup D$.

An *irredundant contour* I is a contour with non-overlapping intervals. Also, the intervals of an irredundant contour are ordered: $I \subset \{(l_i, r_i, t_i) \in R^3 \mid l_i<r_i \wedge r_i \leq l_{i+1}\}$. A trunk can be generated straightforwardly from an irredundant contour: within the span of the trunk a rectangle is placed against each interval, and at each vertical slope between the subsequent intervals. Of course, the intervals in I should cover the whole span of the trunk. The vias are also placed against the contour.

After the new intervals D have been added to the contour C, it is no longer irredundant. To create a new irredundant contour an O(n log n) recursive algorithm, similar to mergesort, is used. It is based on a linear time algorithm to *merge* two irredundant contours J and K. Merging is defined as determining an irredundant contour I with the same functional value as the union of J and K:

$$f_I(x) = \max(f_J(x), f_K(x)) = f_{J \cup K}(x)$$

Because each interval can cut another interval in two, the number of intervals in I is maximally $2(\#J+\#K)-1$. Each interval in I can be covered by at most one interval in J and one interval in K. Since the intervals of J and K are ordered, the intervals of I can be generated by linearly traversing J and K. A redundant contour C containing n intervals is split up in n single interval contours. Obviously a contour containing only one interval is irredundant. At each stage contours are merged pairwise, so there are O(log n) stages. Since the result of merging contours J and K may be a contour with $2\#(J \cup K)-1$ intervals it seems that the number of intervals doubles in each stage. However, it is clear that there cannot be more than 2n-1 intervals in I. Therefore at each stage the amount of work is O(n) and the algorithm is O(n log n). A more straightforward implementation, that first determines the intervals of I, and then for each interval its value t, may need up to n^2 comparisons.

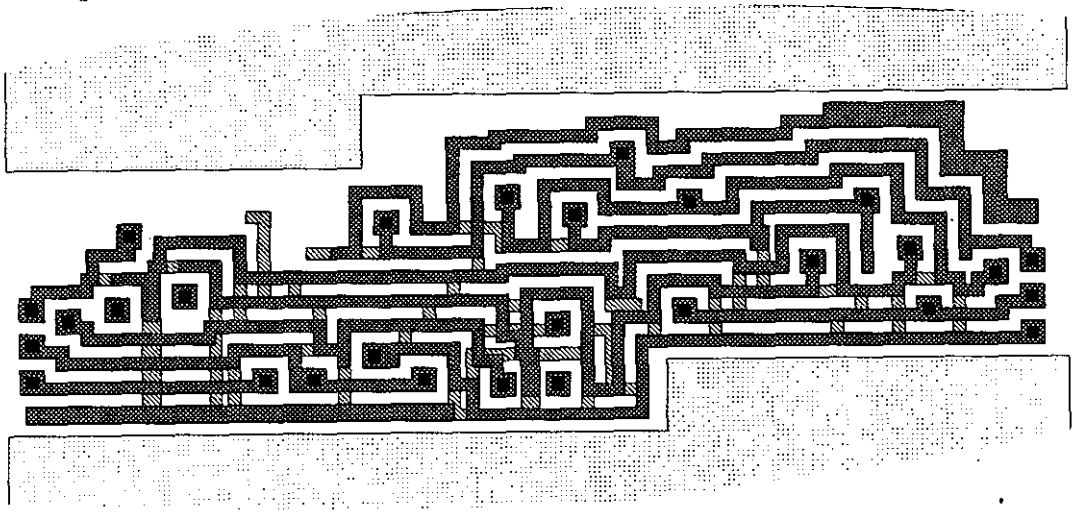


Figure 9. Routing after compaction phase.

5.3 Wire Straightening

As can be seen in fig.9 not all jogs that are generated this way are necessary to reduce the channel width. Each jog is a small reliability hazard due to electro migration. Also many jogs mean many rectangles in the layout. Wire straightening therefore increases the reliability of the chip and decreases the amount of disk space needed to store the result.

Two passes are needed to get rid of unnecessary jogs. The first compaction phase is described in the previous section. In this pass the width of the channel and the space available for each trunk is determined. All irredundant contours are saved, such that it is known exactly what area can be used for each trunk. In the second pass the actual mask generation is done. The trunks are processed in reverse order and the saved contours are mirrored. Each time the mask data for a trunk is generated there are two contours. The

bottom contour C indicates the area that is occupied by previous trunks. The contour U, which was saved during the first phase, indicates the area that is needed for the following trunks. The present trunk will be realized between those contours. To generate a trunk with a minimum number of jogs, the contour C will be replaced by a new straightened contour S, with a minimum number of intervals:

$$f_C(x) \le f_S(x) \le f_U(x) \ \wedge \ \#S \text{ is minimum}$$

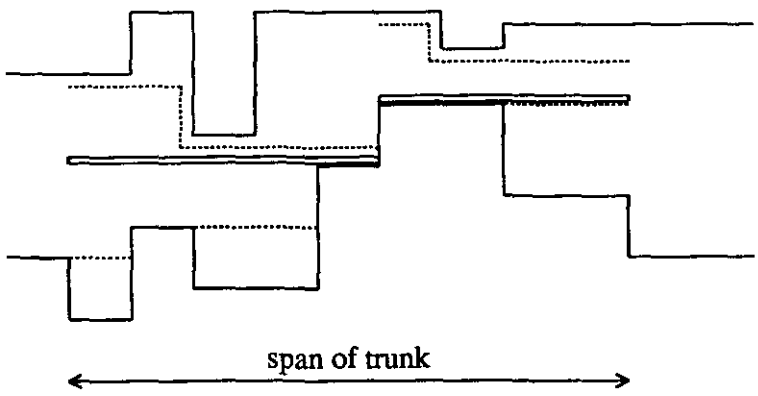


Figure 10. Straightening algorithm.

The contours C and U are irredundant. The straightening algorithm starts at the left side of the span, and works its way to the right. While going to the right, it generates intervals, trying to make each interval (l, r, t) as long as possible. For this purpose two bounds are maintained: $\beta \le t \le \tau$.

$$\beta = \max \{ y \mid (x_1, x_2, y) \in C \wedge x_1 < r \wedge x_2 > l \}$$

$$\tau = \min \{ y-\omega \mid (x_1, x_2, y) \in U \wedge x_1 - \omega < r \wedge x_2 - \omega > l \wedge \langle x_1, x_2 \rangle \cap \text{span} \neq \varnothing \}$$

The interval is extended by selecting the largest r for which $\beta \le \tau$. The interval is added to $S := S \cup \{ (l, r, \beta) \}$, and a new interval, with $l := r$, is started. The resulting contour S is irredundant. It is easy to see that this process will give us the minimum number of intervals. The only possibility would be to begin the next interval earlier. But, when passing the old breakpoint, the freedom in choosing t, limited by β and τ, cannot be larger. Therefore the next interval may have to be shorter than it could have been.

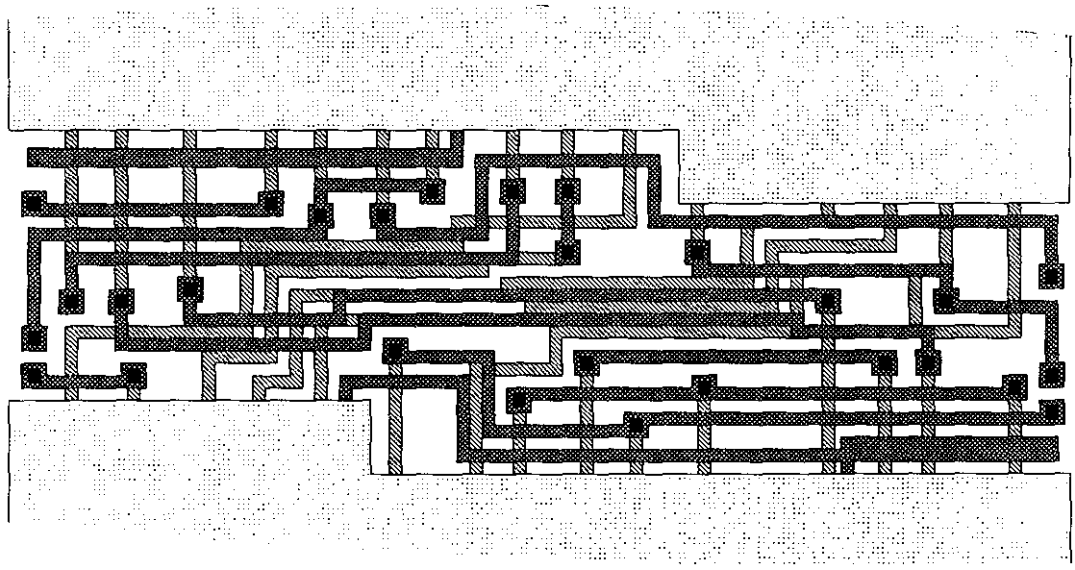


Figure 11. Routing after straightening phase.

6. EXPERIMENTAL RESULTS

A small number of experiments have been done using this router on two randomly generated channels and Deutsch's difficult example. We used Mead & Conway design rules for this experiment. Since our router does not have the concept of tracks it is only possible to compare the width of the channel in lambda's. The terminal positions were on a grid and the channel sides were straight. The pitch of the terminals is indicated over the columns, also in lambda's. For comparison the last column is added which gives the density of the channel times 6. This could be the performance of a good track based router.

Performance for three Channels						
example	gridless router *terminal pitch*					track router *track pitch*
	5	6	7	8	10	6
random1	128	120	118	108	106	132
random2	82	68	68	68	68	84
Deutsch	144	116	110	110	104	114

It can be seen that the width of the channel decreases as the pitch of the terminals increases. The results for Deutsch's Difficult Example are almost as good as those of a good track based router, and for random examples even better. In most practical floor plans the terminal density is much lower, and compaction is more rewarding than track assignment techniques.

The result of the routing system applied to a small chip is shown in fig.12. The chip contains 35 cells, including bonding pads, and 75 nets, including power and ground nets. The program required 37 seconds on an IBM PC/AT to perform the global wiring, and 72 seconds for channel routing. 15 seconds were used for parsing input files and constructing the slicing structure. The result contains 899 rectangles and 308 vias in 38 channels of which 16 are empty.

The routing system also has been applied to standard cell designs. In fig.13 a nMOS pluri cell macro is shown. It can clearly be seen that the power line width is adapted to the needs of the circuit.

Figure 12. Audio delay line chip

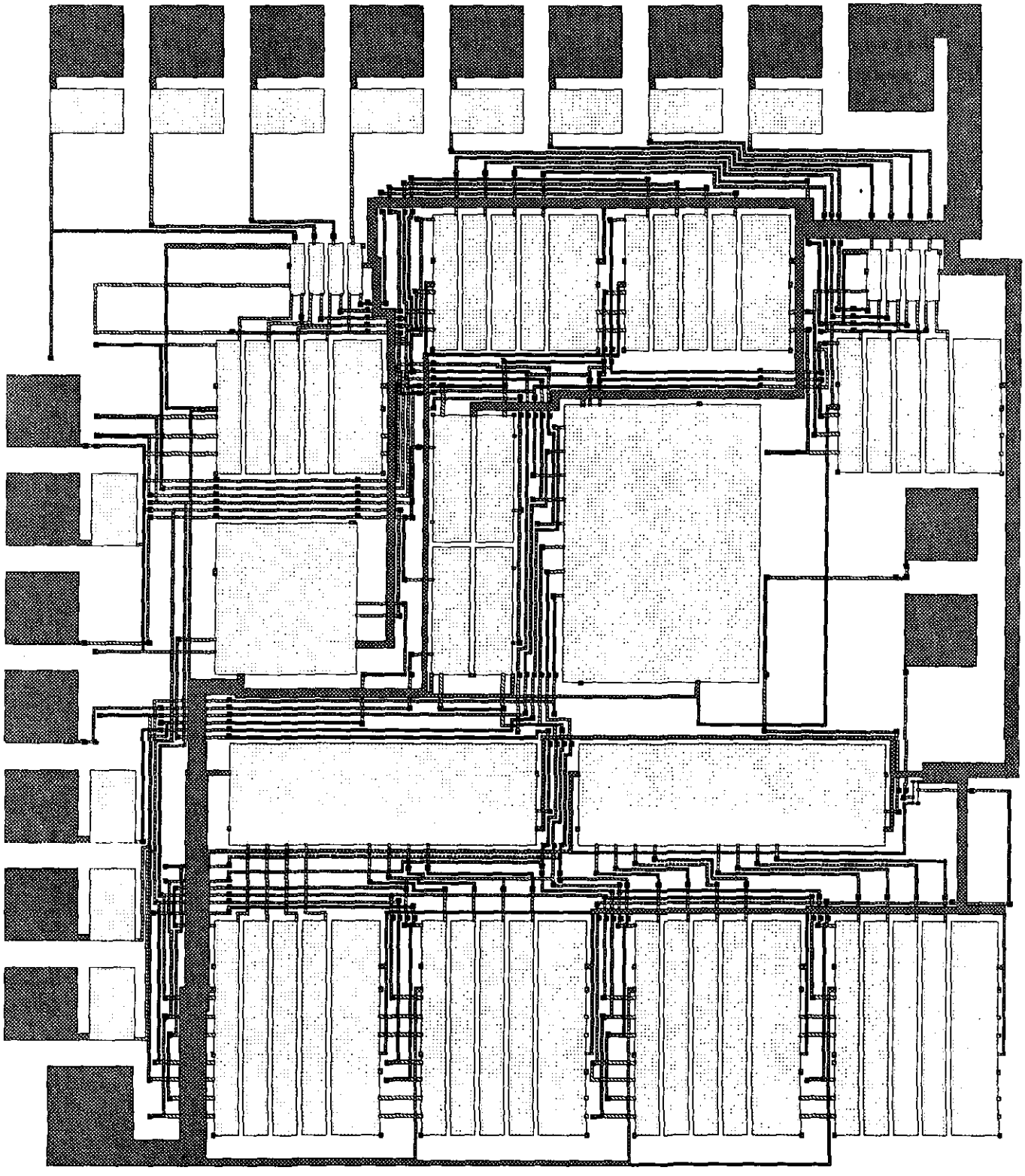
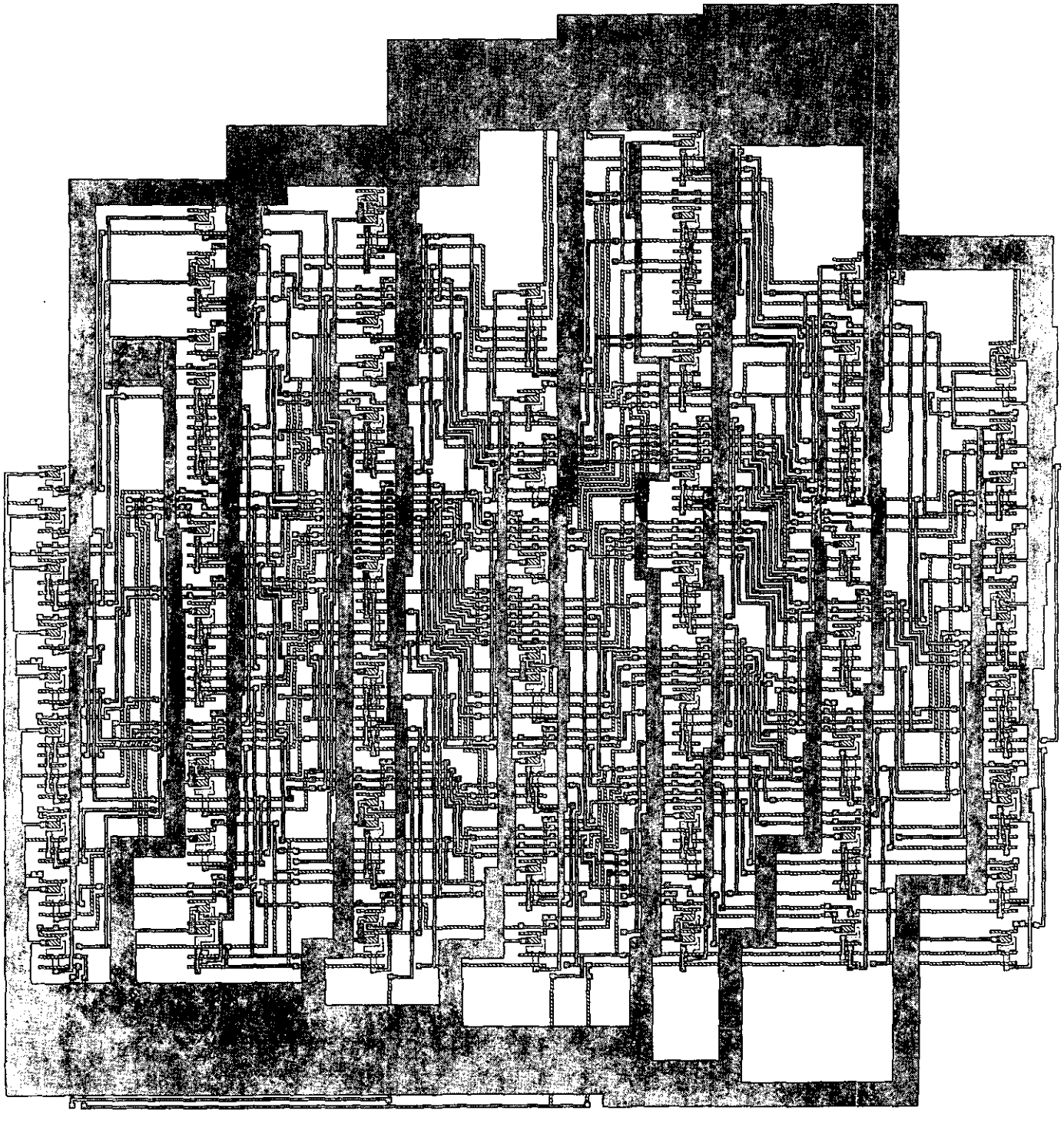


Figure 13. nMOS Pluri cell style random logic macro



7. CONCLUSIONS

It can be guaranteed that the system generates a valid layout provided that the terminals have sufficient spacing (it must be possible to place a via on each terminal). The use of slicing structures and the classification of the trunks help to overcome two well known problems. Many other systems that do not use these concepts cannot guarantee completion. The compaction proves to be a very powerful tool, responsible for a number of important features. It makes the router, as well as the wires (literally) more flexible. The power of the compaction was used also to simplify the channel routing. The concepts introduced make compact implementation of the routing system possible. It was coded in Pascal and needs about 5000 lines.

In combination with more advanced channel routing algorithms the results may still be better, although in real layout the terminals may be too sparse to be a significant nuisance. Possible future enhancements of the system may include the use of an extra routing layer. This would also mean that a more sophisticated way of determining the topology of the wiring in the channel would be needed.

REFERENCES

[1] Chen, H.H. and E.S. Kuh
A VARIABLE-WIDTH GRIDLESS CHANNEL ROUTER.
In: Digest of Technical Papers 3rd IEEE Int. Conf. on Computer-Aided Design (ICCAD-85), Santa Clara, Cal., 18-21 Nov. 1985.
New York: IEEE, 1985. P. 304-306.

[2] Deutsch, D.N.
COMPACTED CHANNEL ROUTING.
In: Digest of Technical Papers 3rd IEEE Int. Conf. on Computer-Aided Design (ICCAD-85), Santa Clara, Cal., 18-21 Nov. 1985.
New York: IEEE, 1985. P. 223-225.

[3] Ginneken, L.P.P.P. van and R.H.J.M. Otten
STEPWISE LAYOUT REFINEMENT.
In: Proc. IEEE Int. Conf. on Computer Design: VLSI in Computers, Port Chester, N.Y., 8-11 Oct. 1984.
New York: IEEE, 1984. P. 30-36.

[4] Ginneken, L.P.P.P. van and R.H.J.M. Otten
GLOBAL WIRING FOR CUSTOM LAYOUT DESIGN.
In: Proc. 18th Int. Symp. on Circuits and Systems, Kyoto, 5-7 June 1985.
New York: IEEE, 1985. P. 207-208.

[5] Lauther, U.
CHANNEL ROUTING IN A GENERAL CELL ENVIRONMENT.
In: VLSI 85: VLSI Design of Digital Systems. Proc. IFIP TC 10/WG 10.5 Int. Conf. on Very Large Scale Integration, Tokyo, 26-28 Aug. 1985. Ed. by E. Hörbst.
Amsterdam: North-Holland, 1986. P. 393-403.

[6] Marek-Sadowska, M. and E.S. Kuh
A NEW APPROACH TO CHANNEL ROUTING.
In: Proc. 15th Int. Symp. on Circuits and Systems, Rome, 10-12 May 1982.
New York: IEEE, 1982. P. 764-767.

[7] Otten, R.H.J.M.
AUTOMATIC FLOOR PLAN DESIGN.
In: Proc. 19th ACM IEEE Design Automation Conf., Las Vegas, Nev., 14-16 June 1982.
New York: IEEE, 1982. P. 261-267.

[8] Rabbie, H. and J. Jacobson
GRIDLESS CHANNEL ROUTING AND COMPACTION FOR CELL-BASED CUSTOM IC LAYOUT.
In: Proc. 8th IEEE Custom Integrated Circuits Conf., Rochester, N.Y., 12-15 May 1986.
New York: IEEE, 1986. P. 297-299.

[9] Wirth, N.
PROGRAM DEVELOPMENT BY STEP-WISE REFINEMENT.
Commun. ACM, Vol. 14(1971), p. 221-227.

[10] Ginneken, L.P.P.P. van and J.A.G. Jess
GRIDLESS ROUTING FOR GENERAL FLOOR PLAN.
IEEE Int. Conf. on Computer-Aided Design (ICCAD-87), Santa Clara, Cal., 9-12 Nov. 1987.

[11] Liedorp, J. and S. de Graaf
HET ICD SYSTEEM: Beginnershandleiding.
Internal Report. Laboratory of Network Theory, Faculty of Electrical Engineering, Delft University of Technology, 14 Jan. 1983.

[12] Annevelink, J.
LDM: Syntax and semantic description.
Internal Report. Laboratory of Network Theory, Faculty of Electrical Engineering, Delft University of Technology, 1983.

[13] Lodder, A. and M.T. van Stiphout, J.T.J. van Eijndhoven
ESCHER: Eindhoven SCHematic EditoR reference manual.
Department of Electrical Engineering, Eindhoven University of Technology, 1986.
EUT Report 86-E-157

[14] Graaf, A.C. de and G.L. Janssen
PROPOSAL FOR A NETWORK DESCRIPTION FORMAT IN THE ICD-SYSTEM.
In: The Integrated Circuit Design Book: Papers on VLSI Design Methodology from the ICD-NELSIS Project. Ed. by P. Dewilde.
Delft University Press, 1986. P. 1.49-1.60.

SUPPLEMENT: ROCOCO USER MANUAL

EINDHOVEN UNIVERSITY OF TECHNOLOGY
DEPT. OF ELECTRICAL ENGINEERING
AUTOMATIC SYSTEM DESIGN GROUP

ROCOCO
User Manual

by

L.P.P.P. van Ginneken

P.O.Box 513, 5600MB Eindhoven, The Netherlands
tel: 31-40-473352

Eindhoven
September 1987

CONTENTS

LIST OF FIGURES

1. INVOCATION

On a UNIX system the router can be invoked with

rococo [-vd -t<technology -o<output>] <layout> <net list>

The options are:

— v verbose: gives the full output listing of all phases. When This option is turned off only the error messages are printed.

— d database: only the cells that are generated by the router are put into the output file. This is useful when working with the ICD data base. When this option is turned of the program includes the standard library and the input file to generate a complete chip layout.

— t technology: you can specify the technology file to be used. If omitted the default is used. (Presently 6u nMOS)

— o output: the output file can be renamed using this option. When omitted the output file will be output.ldm

To run the program you have to prepare a LDM file containing the whole chip design, except for the wiring. Notice that also the bonding pads must be included. The power and ground lines must have bonding pads in the upper right, and lower left corner. The terminals of these bonding pads must be facing outward, up and down. The bonding pads are modules in any other respect.

The net list file must contain all connections that you want to make. That is also clock lines, power and ground lines, reset etc. must be explicitly present in the net list. The input files must be files in the current directory!

2. INPUT FILES

2.1 Technology File

The technology file controls the operation of the router. Tags may be specified in any order, with exception of the LIBRARY tag, which must be last in the file. Unrecognized tags are skipped as comments. The following tags are recognized:

FLEX

This tag specifies that the router is allowed to shift the block in the floor plan to reduce the wasted space. If FLEX was not specified and router has to create routing space, a warning message is generated.

LIBRARY

This tag is followed by an LDM description of the standard components of this technology. The router generates calls to the standard model rcontact, which represents the contact between the two routing layers. Notice that the origin of this model must be the lower left corner.

NAMES *<floor plan> <chip> <ground> <power>*

Specify the special names to be recognized. The <floor plan> name is the name of the top model which contains the layout of the placement. The resulting layout is put into a new top model with the name <chip>. <ground> and <power> are the names of the special nets that are to be routed planarly. Notice that these nets must have bonding pads in the upper right, and lower left corner. They must have terminals that are facing outward, respectively up and down. This tag is compulsory.

NEED *<real>*

This is the power need in mA per mm2 for the modules. It is used by the global routing to determine the width of the power lines.

POLY *<integer>*

When the program has a choice of layers, the poly silicon layer is prefered if the section is shorter than the specified value, and two via's can be eliminated.

THRU

When specified, the program uses equivalent pins as feedthru's. Otherwise a choice is made between the equivalent pins. Pins are equivalent if their names are equal, or if they differ only in the last character, which must be a capital.

WIRE *<sep> <width> <hole> <bbx_sep> <name>*

This tag should occur twice. It specifies the design rules and the names for the routing layers. The first layer is the top, metal layer, the second the polysilicon layer. The planar power and ground nets are placed in the first layer. The design rules are:

<sep> is the minimum separation between unconnected rectangles in the same layer.
<width> is the minimum width of the boxes.
<hole> is the size of the contact hole in that layer.
<bbx_sep> is the minimum separation to the bounding box.
<name> is the layer name used.

```
/* Technology for standard 6u nMOS process */
WIRE 6 6 12 6 nm
WIRE 6 6 12 6 np
NAMES floor chip vdd vss
POLY 200
FLEX
THRU
NEED 5
/* library contains the standard components for this technology */
LIBRARY
ms rcontact
box nm 0 12 0 12
box np 0 12 0 12
box nc 4 8 4 8
me
```

Figure 1. Example technology file in use at Eindhoven.

2.2 LDM files

LDM is is simple layout description language that originates from Delft University [11, 12]. It features hierarchy, module names, terminals with names and names for masks. The layout is described by rectangles. Module instances can be rotated or mirrored. Every new element is written on a separate line. This document describes an Eindhoven local version. Additions are: instance names, surrounding boxes and names for rectangles. Omissions are: wire and continuation statements, and array compound calls. All additions are indicated as being optional. The additions can be used to describe the interfaces of cells to programs. Furthermore, to facilitate upward compatibility with possible future versions, lines that start with an unknown keyword are comments. For the same reason also any text added to the end of a syntactically correct line is ignored. The syntax is given in SBNF. Notice that { .. }+ means one or more times. (*)

<layout> ::= { <module> }+ .

A layout consists of a set of modules. A module is declared before it can be

---

(*) *The numbers in the brackets refer to references on page 22 of the preceding main text of this report.*

instantiated. Also recursion is not allowed.

<module> ::= <header> { <element> }+ <tail> <eol> .

Each module has a header, a non empty sequence of layout elements, and a tail.

<header> ::= 'ms' <name> <eol> .

The name of the module is the template name.

<tail> ::= 'me' [<rectangle>] .

The tail contains optionally a specification of the surrounding box. The surrounding box encloses all layout elements in the module.

<element> ::= <box> | <module-call> | <terminal> .

There are three kinds of layout elements:

<box> ::= 'box' <layer> <rectangle> [<name>] <eol> .

Boxes are the layout primitives. The boxes have an optional name. This name may be used to indicate descendance.

<module-call> ::= 'mc' <name> ['mx' | 'my'] ['r3' | 'r6' | 'r9']
<integer> <integer> [<name>] <eol> .

A module call is used to instantiate a template. The first name is the template name. Then the coordinate at which the module is instantiated. The module can be transformed on instantiation. Transformations are relative to the coordinate specified. This coordinate is the origin (0, 0) of the module, not the center! The transformations are:

mx = mirror in X axis
my = mirror in Y axis
r3 = rotate 90 degrees anti clock wise
r6 = rotate 180 degrees
r9 = rotate 90 degrees clock wise

The optional name is the instance name, which may be required to identify the module uniquely.

<terminal> ::= 'term' <layer> <rectangle> <name> <eol> .

Terminals are used for outside connections, and will usually be on the edge of the bounding box. Terminals are not layout, so they must overlap with boxes. A wire that connects to two sides should have two separate terminals.

<rectangle> ::= <integer> <integer> <integer> <integer> .

The integers are respectively the left, right, lower and upper bound of the

rectangle.

<layer> ::= <name> .

Layer names are defined by the process. The layer names are not part of the definition of LDM and depend on the technology.

<integer> ::= ['-'] { <digit }+ .

Integers are as read by any decent program.

<name> ::= <letter> { '_' | <letter> | <digit> } .

Names follow the rules of normal identifiers. Case sensitive, but use only lower case, please. Names may have a maximum of 8 characters.

<eol> ::= { { <character> } '\n' }+

Lines are significant in this language, every line that does not start with a known keyword is a comment. The lexical elements of LDM consist of sequences of characters that are separated by spaces or new-lines.

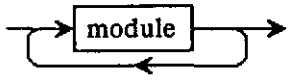
```
ms mosfet1
box ps 0 14 4 10
box od 4 10 0 14
term ps 0 6 4 10 gate
term od 4 10 0 2 source
term od 4 10 12 14 drain
me

ms usefets
mc mosfet1 15 10 r3
mc mosfet1 29 10 mx r9 fet2
me
```

Figure 2. Example of LDM file

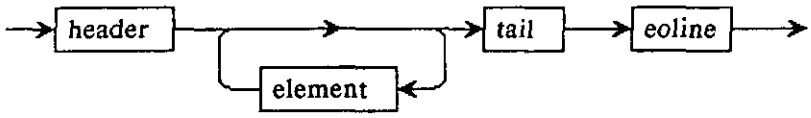
Compatibility problems may arise because of omissions of the language, use of non standard layer names, too long identifiers or identifiers with funny characters, and a different interpretation of coordinate transformations. For coordinate transformations use the -o option when using cldm.

Figure 3. Syntax diagrams of LDM

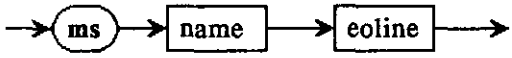
layout :



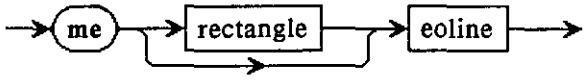
module :



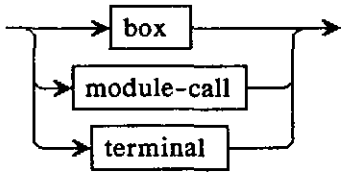
header :



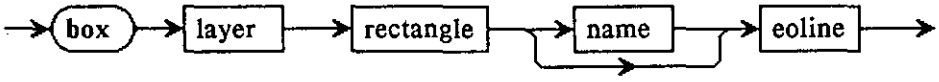
tail :



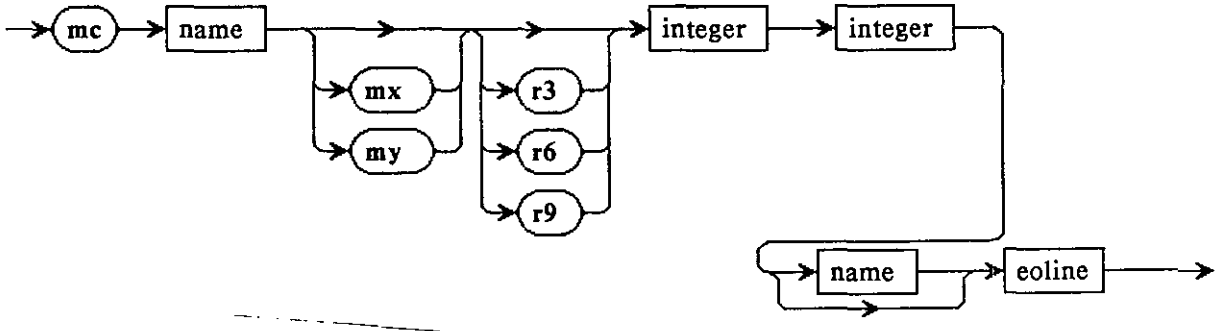
element :



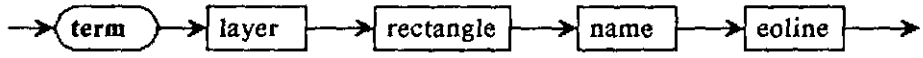
box :



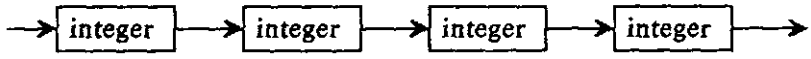
module-call :



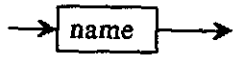
terminal :



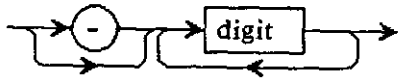
rectangle :



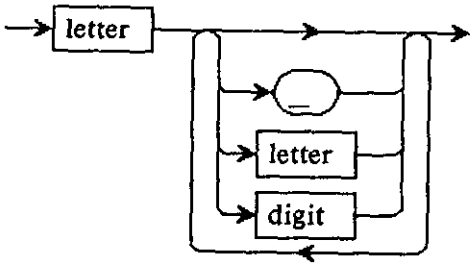
layer :



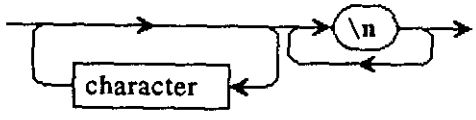
integer :



name :



eoline :



2.3 Network File

This is a description of the net work file used by the routing program. It is a simplified version of the ICD network data base [14], as generated by escher [13]. The file defined by the network description format is a sequential ASCII-file containing only printable characters, tabs and new-lines. The file consists of one or more records, which are variable length lines.

<network-file> ::= <record> { <NEWLINE> <record> } .

Each record consists of 3 fields. Fields are separated by white space.

<record> ::= <field> { <separator> <field> } .
<separator> ::= (<BLANK> | <TAB>) { <BLANK> | <TAB> } .

The network file gives the specification of the interconnection of the terminals of the instances. One record is meant to specify one terminal. All terminals with the same net name are connected to each other. The names of the terminals and the instances must occur in the layout file.

A record contains the fields:
<net> <instance> <terminal>
<net> ::= <String> . net description consisting of a netname.
<instance> ::= <String> . origin name of the terminal.
<terminal> ::= <String> . terminal name.

3. INTERMEDIATE FILES

The router consists of three passes that communicate with each other by means of files. This is a description of the format of these intermediate files.

3.1 Florplan

This file is written by the entry pass, and read by the global and local pass. It contains a record for each module, each channel and each slice (all nodes in the slicing tree). Each record has the following format:

<linenr>	number of the slice, increases from 1 by 1
<primogenitive>	structure of the slicing tree, link to first subslice
<nextsibling>	link to the next slice.

These field contain the neighbor relations For modules the references are absolute in the coordinate system, but for channels they are relative. For channels only the topjc and botjc are filled in.

<topjc>	Points to channel to the top of this module (higher y)
<botjc>	Points to channel to the bottom of this module (lower y)
<leftjc>	Points to channel to the left (lower x)
<rightjc>	Points to channel to the right (higher x)

Geometry of the floor plan.

<coordx>	X coordinate of lower left corner of bounding box.
<coordy>	Y coordinate of " " " " " "
<sizex>	Width of bounding box.
<sizey>	Height of bounding box.
<offsetx>	offset of bounding box relative to origin of LDM desc.
<offsety>	"
<orient>	orientation 0..7
<level>	level in slicing tree, negative means channel.
<name>	name of the module or channel.

3.2 Mod2Net

This file is for specifying terminals of modules and channels. Two files are generated with this format: mod2net and mod2net.app. The first one is generated by the entry, and

read by the sub- sequent passes. It contains the terminals of the modules. The second one is generated by the global routing and contains the terminals that interface the channels to each other. The format of each record is as follows:

<netnr>	The number of the net that the pin is connected to.
<same>	Points to the next equivalent pin, (linked in a circular list.)
<cside>	Side of the pin, absolute modules, but relative for channels.
<alt>	height (y coordinate) of the pin relative to the channel.
<left>	left x coordinate
<right>	right x coordinate
<layer>	layer of the pin, 1 or 2.
<name>	name of the pin.

The records are grouped together. Each group starts with the number of the module or slice that they belong to. This number is given on a separate line. Each group is terminated by an empty line.

3.3 Channels

This file specifies the two terminal segments in each channel. The records are grouped by channel, as in the mod2net.app file. Each record contains the following fields:

<frompn>	the two pins to be connected.
<topn>	
<netnr>	the netnumber.
<width>	the width of this wire segment; when smaller than the minimum design, the design rule is maintained.

4. DIAGNOSTICS

The program prints diagnostic messages in the following format:

{ <name> } '****' <mesg-kind> ['in line ' <nnn>] ':' <message> .

The <name>s may be names of nets, modules or pins that are refered to in the error message. <mesg-kind> can be Warning, Error or Bug. When the program is parsing a file while the error occurs, the line number is given.

4.1 Warnings

Warnings are intended to warn the user that the results are not necessarily what he or she intended, or to make him or her aware of certain peculiarities of the input.

Channel width extended.
The width of a channel had to be extended to accommodate the wires, but FLEX was not specified in the technology file.

Net nnn has only one module.
A net with only one module was encountered. This net will not be routed.

Empty net.
An empty net was skipped.

4.2 Errors

Errors are usually due to syntactic or semantic errors in the input, or due to memory limitations. The user should correct the input, or in the case of memory limitations recompile the program with larger array dimensions.

Invalid integer.
An integer was expected, but could not be read. The syntax for integers is ['-'] { <digit> }+ . Notice that there must be a space or an end-of-line on either side of the integer.

Invalid design rule.
A zero or negative value was specified in a WIRE tag in the technology file.

WIRE tag missing in tech file.

Two WIRE tags, one for each layer, must be specified in the technology file. Each tags should be followed by respectively the width, separation, size of the contact, the separation from the modules, and the layer name.

NAMES tag missing in tech file.

A names tag must be specified in the technology file. The tag must be followed by the floor plan name, the name of the output module and the power and ground names.

Boxes in floor plan layout.

In the floor plan of the chip only module call may occur.

Terminals in floor plan.

In the floor plan of the chip only module calls may occur. It is presently not possible to specify terminals to the outside of the floor plan.

Unknown terminal layer.

A terminal was specified in an unknown layer.

Call of an undeclared module.

Modules must be declared before used.

Unknown rotation mnemonic.

Allowed rotation mnemonics are r3, r6 and r9.

Module name must be unique.

A module was used twice in the floor plan. If you want to use the same module twice, you must make a copy under another name.

Too many modules in floorplan.

The program ran out of memory space to store the modules. Use less modules or recompile the program with more space.

Model start expected.

An 'ms' was expected but something else was encountered. boxes, calls and terminals may only be specified within a module.

Too many models defined.

Too many modules were defined, and the program ran out of space to store them. Remove unused models, or remove models that do not occur in the floor plan.

Nested models not allowed.

An 'ms' was encountered before a closing 'me' was encountered. Model definitions cannot be nested, so end the previous definition before starting the next.

Terminal outside bounding box.

A terminal was found outside the bounding box.

Terminals are too close.

Two terminals are so close that a valid routing cannot be guaranteed. Move the terminals further apart.

Floor plan must be last.

The last definition in the input ldm file must be the floor plan.

Terminal too narrow.

The terminal is narrower than the design rules in the technology file allow.

Too many terminals defined.

More terminals were defined than the program could store. Use less terminals by removing all unused terminals or recompile the program with a larger Ldim.

Unknown module.

A module is refered to that was not declared.

Number of nets exceeds limit.

More different nets were used than the program can handle. Use less nets or recompile the program with a larger Ndim.

Signal pins must be poly.

Signal pins must be in the second (usually polysilicon) layer.

Supply terminal at wrong side.

No power or ground supply terminal was found at the correct side of the module. Only a terminal at the wrong side was found. Power and ground terminals can only be reached from one side.

Terminal not found in layout.

A terminal name was encountered in the net list, but could not be found in the layout.

Degree too large.

The degree of a node in the routing graph has become too large. This may be a bug, or else recompiling with a larger maximum degree should help.

4.3 Bugs

These messages should never occur, and since they cannot occur in principle, it is hard to pin point their cause if they do. In general one can say that the bug need not be where the message is generated. A possible cause is that the intermediate files that the several programs use are corrupted or inconsistent. Please talk to your system administrator or contact

L.P.P.P. van Ginneken
Vakgroep ES, Dept. of EE
Eindhoven Univ. of Technology
P.O.Box 513
5600 MB Eindhoven
The Netherlands

tel: 31-40-473352
bitnet: elesluka at heithe5

Negative distance.

in NewEdge (global): tries to create edge in routing graph with negative distance.

in buildnodejc.

in BuildNodejc (global): Nodejc is inconsistent with Index.

Nodes not in same channel.

in Connect (global): nodes that are linked with an edge were not in the same channel.

Heap exhausted.

in GetHeap (global): tries to get a node from the heap, but no node is left.

Mod2Net ran out of space.

in CreatePin (global): tried to create a pin, but no space was left in Mod2Net. Increasing Lmax may help.

Not a valid node.

in GivePin (global): this node is not in the graph.

while Removing killed pins.

in RemoveKilled (global): only one pin of this net is left, so to what should this pin be connected?

while Making set.

in MakeSet (global): not all two terminal segments connected to the same pin carry the same netnumber.

internal references <> 1.

in Checking (global): internal references should be 1 after WireStraight.

external references are 0.

in Checking (global): externally unreferenced pins should have been killed by WireStraight.

reference to killed pin.

in Checking (global): two terminal segment refering to killed pin was found. This means a bug in WireStraight.

Quick sort, errno = n.

in QuickSort (local): sorting did not work properly.

Short Circuit.

in ChangeLayer (local): a short circuit was detected while changing layers.

Incorrect input data errno=n.

in chk (local): a pin was found to be on the wrong side of the channel boundary.

Terminal checking, errno=n.

in TermCheck (local): not all terminals were inside the modules.

Wrong tside.

in FixTerminals (local): a channel can only have terminals at the north/south ends.

Routing order conflict.

in MakeBoundaries (local): levels are inconsistent with neighbors.

Illegal transform.

in MakeRootBlock (local): transform must have a value in 0..7

Boundaries in wrong sequence.

in add (include): x coordinates of interval added to contour are in descending order.

Contour has overlapping block.

in check (include): contour must be irredundant. This is an error in the compaction phase.

Bug in merge.

in merge (include): this case cannot occur if all previous conditions are correct.

Bug n in Straight.

in straight (include): there was no area left to create the wire. This message can have many, many different causes. Maybe the error was caused by a mistake with design rules, contours, contacthole positions etc. Almost all subtle mistakes cause this error to occur.

5. IMPLEMENTATION NOTES

The program is designed to be as portable as possible. Only standard Pascal features were used, and a few extensions that are shared by many implementations. It proved to be possible to solve almost all out problems using only standard Pascal. To make this possible it was necessary to create an interface for reading files in entry.

The main thing that could not be done in a uniform way was giving names to files. This is done by the procedure NameFiles. For each implementation a separate procedure is given. The correct one can be selected by removing the comment brackets (* *). In this section also other non standard features can be put, like include files, compiler options, or non standard functions.

It was necessary to abstain from dividing the program into modules that can be linked, because this differently solved in various Pascal implementation. To create the advantage of separate compilation the system was implemented as three programs. This also has the advantage of being able to inspect the data at the interfaces of the programs. Also the three programs work like three overlays, thus saving memory usage.

Each program contains a number of constants, which can be increased, if larger routing problems are to be processed. These are the constants that the user may want to change:

constant	maximum number of	dependency
tokenlen	characters in a name	>= 8
Bmax	branches in one channel	>= Tmax
Cmax	contacts to the sides of one channel.	>= Tmax
Emax	edges to a routing node should be	> 6
Fmax	slices	= 3 * Mmax
Jmax	junction cells	= Mmax + 3
Kmax	nodes in routing graph	> 2 * Mmax
Lmax	terminals	>> 2 * Nmax
	should in global and local be approx. double	
Mmax	modules	depends on problem size
Nmax	nets	>> Mmax (depends)
Omax	objects	> Mmax
Pmax	power/ground pins in one channel	> 10
Qmax	pins in one net	> 10
Smax	two term seg in one channel	~ Nmax
Tmax	trunks in one channel	> Smax
Ymax	layers	2

Eindhoven University of Technology Research Reports
Faculty of Electrical Engineering

ISSN 0167-9708
Coden: TEUEDE

(162) Meer, A.C.P. van
TMS32010 EVALUATION MODULE CONTROLLER.
EUT Report 86-E-162. 1986. ISBN 90-6144-162-5

(163) Stok, L. and R. van der Born, G.L.J.M. Janssen
HIGHER LEVELS OF A SILICON COMPILER.
EUT Report 86-E-163. 1986. ISBN 90-6144-163-3

(164) Engelshoven, R.J. van and J.F.M. Theeuwen
GENERATING LAYOUTS FOR RANDOM LOGIC: Cell generation schemes.
EUT Report 86-E-164. 1986. ISBN 90-6144-164-1

(165) Lippens, P.E.R. and A.G.J. Slenter
GADL: A Gate Array Description Language.
EUT Report 87-E-165. 1987. ISBN 90-6144-165-X

(166) Dielen, M. and J.F.M. Theeuwen
AN OPTIMAL CMOS STRUCTURE FOR THE DESIGN OF A CELL LIBRARY.
EUT Report 87-E-166. 1987. ISBN 90-6144-166-8

(167) Oerlemans, C.A.M. and J.F.M. Theeuwen
ESKISS: A program for optimal state assignment.
EUT Report 87-E-167. 1987. ISBN 90-6144-167-6

(168) Linnartz, J.P.M.G.
SPATIAL DISTRIBUTION OF TRAFFIC IN A CELLULAR MOBILE DATA NETWORK.
EUT Report 87-E-168. 1987. ISBN 90-6144-168-4

(169) Vinck, A.J. and Pineda de Gyvez, K.A. Post
IMPLEMENTATION AND EVALUATION OF A COMBINED TEST-ERROR CORRECTION PROCEDURE FOR MEMORIES WITH DEFECTS.
EUT Report 87-E-169. 1987. ISBN 90-6144-169-2

(170) Hou Yibin
DASM: A tool for decomposition and analysis of sequential machines.
EUT Report 87-E-170. 1987. ISBN 90-6144-170-6

(171) Monnee, P. and M.H.A.J. Herben
MULTIPLE-BEAM GROUNDSTATION REFLECTOR ANTENNA SYSTEM: A preliminary study.
EUT Report 87-E-171. 1987. ISBN 90-6144-171-4

(172) Bastiaans, M.J. and A.H.M. Akkermans
ERROR REDUCTION IN TWO-DIMENSIONAL PULSE-AREA MODULATION, WITH APPLICATION TO COMPUTER-GENERATED TRANSPARENCIES.
EUT Report 87-E-172. 1987. ISBN 90-6144-172-2

(173) Zhu Yu-Cai
ON A BOUND OF THE MODELLING ERRORS OF BLACK-BOX TRANSFER FUNCTION ESTIMATES.
EUT Report 87-E-173. 1987. ISBN 90-6144-173-0

(174) Berkelaar, M.R.C.M. and J.F.M. Theeuwen
TECHNOLOGY MAPPING FROM BOOLEAN EXPRESSIONS TO STANDARD CELLS.
EUT Report 87-E-174. 1987. ISBN 90-6144-174-9

(175) Janssen, P.H.M.
FURTHER RESULTS ON THE McMILLAN DEGREE AND THE KRONECKER INDICES OF ARMA MODELS.
EUT Report 87-E-175. 1987. ISBN 90-6144-175-7

(176) Janssen, P.H.M. and P. Stoica, T. Söderström, P. Eykhoff
MODEL STRUCTURE SELECTION FOR MULTIVARIABLE SYSTEMS BY CROSS-VALIDATION METHODS.
EUT Report 87-E-176. 1987. ISBN 90-6144-176-5

(177) Stefanov, B. and A. Veefkind, L. Zarkova
ARCS IN CESIUM SEEDED NOBLE GASES RESULTING FROM A MAGNETICALLY INDUCED ELECTRIC FIELD.
EUT Report 87-E-177. 1987. ISBN 90-6144-177

(178) Janssen, P.H.M. and P. Stoica
ON THE EXPECTATION OF THE PRODUCT OF FOUR MATRIX-VALUED GAUSSIAN RANDOM VARIABLES.
EUT Report 87-E-178. 1987. ISBN 90-6144-178-1

(179) Lieshout, G.J.P. van and L.P.P.P. van Ginneken
GM: A gate matrix layout generator.
EUT Report 87-E-179. 1987. ISBN 90-6144-179-X

(180) Ginneken, L.P.P.P. van
GRIDLESS ROUTING FOR GENERALIZED CELL ASSEMBLIES: Report and user manual.
EUT Report 87-E-180. 1987. ISBN 90-6144-180-3

(181) Bollen, M.H.J. and P.T.M. Vaessen
FREQUENCY SPECTRA FOR ADMITTANCE AND VOLTAGE TRANSFERS MEASURED ON A THREE-PHASE POWER TRANSFORMER.
EUT Report 87-E-181. 1987. ISBN 90-6144-181-1