Macro-cell generation for combinatorial logic

Tullemans, H.H.

Award date:
1986

Eindhoven University of Technology
Department of Electronic Engineering
Laboratory Automatic System Design

MACRO-CELL GENERATION FOR COMBINATORIAL LOGIC

H.H. Tullemans

Thesis performed during 1.1.1985 - 19.12.1985
by order of prof. dr. -ing. J.A.G. Jess
and supervised by dr. ir. J.F.M. Theeuwen.

ABSTRACT

An automatic design system is implemented in PASCAL to obtain from a set of Boolean expressions a complete layout in NMOS-technology with a standard-cell architecture.
Several tasks of this system were already completed such as the simplification and optimalisation of the boolean expressions, and their realisation in NMOS random logic.
The remaining tasks to obtain a complete layout were the generation of cells, placement of these cells and finally routing. These remaining steps are implemented. The theory, implementation and the results of the several approaches are described.

CONTENTS

1. PROJECT ENVIRONMENT

The results of this thesis report are a contribution to a international project supported by the European Communities, Information Technologies Task Force "ESPRIT". The official title of this project is:

> **Coopearative Development of an Integrated Hierarchical and Multiview VLSI-Design System with Distributed Datamanagement on Intelligent Workstations.**

The project is a cooperation of British Telecommunications research department, PCS a workstation manufacturer from West-Germany, ICS a software house from the Netherlands and two universities, i.e. the Universities of Technology from Delft and Eindhoven.
The final result of this project will be a complete, integrated design-system that can be implemented on intelligent workstations.
To achieve this final result several goals have to be reached. The first goal is the development of new, modern design tools which make an efficient use of the structure of the integrated circuit. Second, development of a design database through which the design tools can be used more efficient and user-friendly. Also internationally accepted software- and design-standards must be used which will allow the system to run on a large variety of workstations.
The project is subdivided in six main Work Packages which capture the goals stated above. The tasks are the following:

1. Work Package 1: System architecture and distributed datamanagement.
2. Work Package 2: Low-level verification of submicron circuits.
3. Work Package 3: Distributed workstation architecture for VLSI-design.
4. Work Package 4: Software components for a silicon compiler for custom VLSI.
5. Work Package 5: Logic synthesis.
6. Work Package 6: Development of quality software and support.

The Eindhoven University of Technology is responsable for Work Package 5. This Work Package has four tasks i.e.:

Task 1 : Construction of precendence graph editor.
Task 2 : Logic editing and state assignment.
Task 3 : Designing built-in test modules for CMOS and NMOS based architectures.
Task 4 : Generating layouts for random logic.

The main part of this thesis report is dealing with task 4 in Work Package 5.

2. INTRODUCTION

Since the invention of the transistor there has been a continuous progress in the area of electronics, specifically digital electronics, and today we are dealing with VLSI.
This progress is coupled with an increase in the complexity of the function implemented on the chip and an increase in performance.
Because of the increase in complexity and performance the design time has to be shorter. Otherwise it is not possible to implement a VLSI system successfully. In order to achieve a shorter design time, the design has to be structured (top-down, bottom up) and the design has to be carried out in a number of well- defined hierarchical levels. At each level extensive use is made of design and simulation aids.

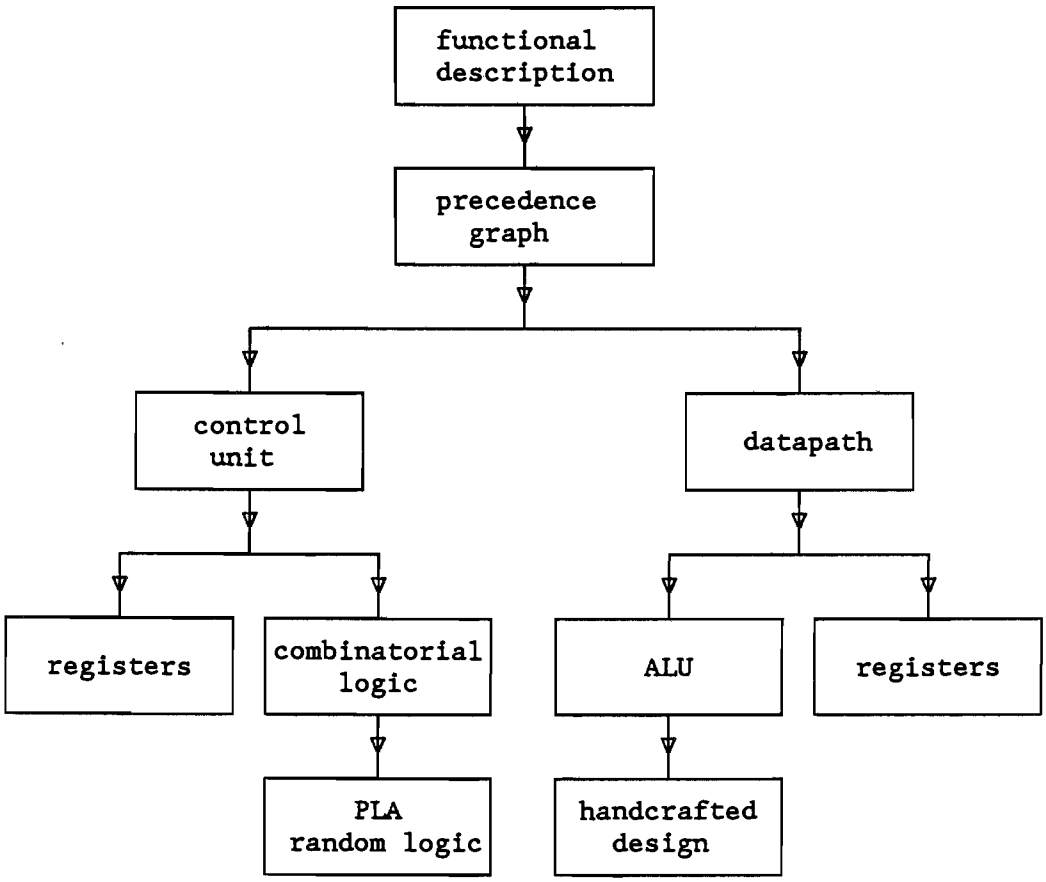


Figure 2.1

To design a digital VLSI-circuit a top-down approach will be used in the ESPRIT- project (fig.2.1). It starts with a functional description that tries to capture an algorithm in terms of

expressions and control statements. By using such an algorithm the designer is capable to describe the function of a desired circuit correctly, reliable and more easily.
The first design-step is the development of parts of the description into precedence graphs. In the precedence graph vertices are associated with expressions and data-flow with directed edges. Many architectural decisions can be discussed in terms of the precendence graph if technological data is available. Numeruous subjects, such as estimating the amount of wiring, critical path analyse, optimal scheduling etc. can be computed or evaluated.
Editing, without changing the functional description, of the precedence graph yields the control and the datapath structure of the system.
The next step derives the control unit from the functional description and the structure of the datapath. Essentially the control unit consists of a set of finite state machines. Those finite state machines are implemented by registers and combinatorial logic. The logic can be built by using for example PLA's or random logic.
Finally the control unit and the datapath must be realized on a chip. Most datapaths have a regular structure and can most appropriately be designed by layout-editors supporting hierarchy and repetition. For control units PLA-, gate array-, standard cell- or slicing-architectures can be used. The control unit can be generated automatically in order to reduce the design-time. For automatic generation of a layout, place and route programs must be used.

The combinatiorial logic that is part of a finite state machine can be described in a set of boolean expressions. The design system, that will be described in the next chapters, transforms these boolean expressions automatically into a layout.
The boolean expressions are implemented in random logic with a standard cell architecture.
Several steps are neccesary to obtain a complete layout from a set of boolean expressions. First of all the boolean expressions must be simplified and optimized to remove all the redundancy and to obtain expressions with a reduced complexity.
Due to several constraints of the used NMOS-Technology a transformation of the simplified and optimized boolean expressions has to be performed in order to obtain realisable boolean expressions.
The next step is the realisation of each realisable boolean expression into a layout of a standard cell. This step is called the cellgeneration.
After the cellgeneration the cells are placed in columns in such a way that the wiring, necessary to connect the cells, is minimal.
The last step is routing of the channels in between the cells. The goal of the routing phase is to minimize the total chip-area

while all connections between the cells are established and all designrules are obeyed.
When the routing is finished the layout of the channels is known. Together with the layout of each cell produced by the cellgenerator the complete layout is obtained.

3. SYSTEM DESCRIPTION

The design-system has to perform several steps (fig.3.1) in order to obtain a complete layout in NMOS-Technology from a set of boolean expressions. Each part of this system will be described shortly in the next paragraphs.

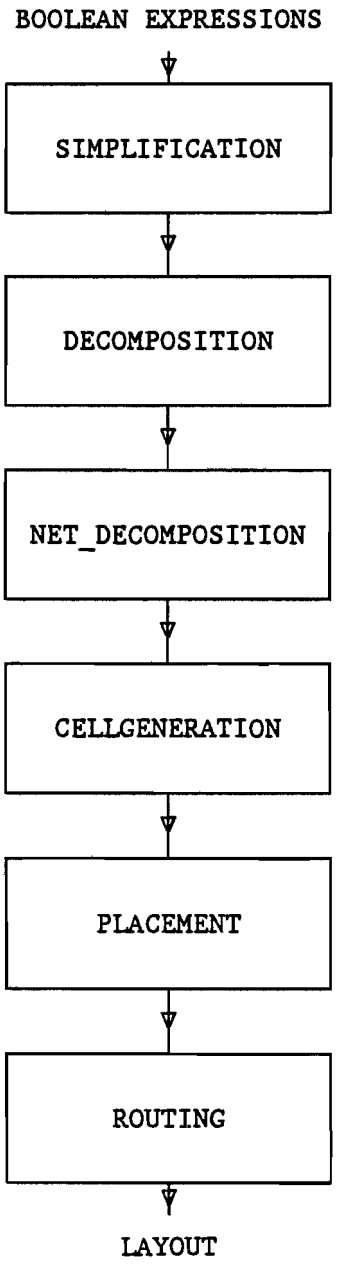


Fig.3.1

3.1 SIMPLIFICATION

As mentioned before, the input is a set of boolean expressions. It is possible that these boolean expressions contain certain redundancies, i.e. the expressions could be simplified. The probability that redundancy exists is specially high for "hand-made" expressions. The main reason for simplification is that the second step, the decomposition phase, does not deliver optimal results if the boolean expressions are not simplified.
Simplification is the task of the first program in the system. The result of the simplification will be a set of simplified boolean expressions. A detailed description of the simplification and its implementation can be found in a thesis report of A.J. Smits [1].

3.2 DECOMPOSITION

Simplification operates only on a single function. But a conscientious study of the whole set of expressions can lead to determination of common subexpressions. A new variable can be introduced for such a common subexpression. The result of this step is a less complex set of boolean expressions extended with a set of new variables.
An algebraic technique is used to find these common subexpressions and substituting them by new variables. This technique and also its implementation can be found in a thesis report of P.T.H.M. Van Paassen [2]. This part, called decomposition, is the second step in the system.

Simplification and decomposition together, are also known as the Logic Editor. The input for the Logic Editor is a set of boolean expressions and the output is a set of simplified and optimized boolean expressions. The Logic Editor is a part of the design-system but it can also be used for other applications.

3.3 NET_DECOMPOSITION

The third step is an interface between the boolean expressions and the used technology.
The final goal of the system is to produce a complete layout in NMOS-Technology. The reason for choosing the NMOS-technology is that this process is running in the EFFIC, the IC-fabrication laboratory of the Eindhoven University of Technology.
Circuits can be constructed with NAND, NOR and INVERTER configurations. Also combinations of these configurations can be used in order to construct more complex circuits.
Each expression can have a maximum number of 3 drivers in series or parallel. So, if the expression is represented as a-sum-of-products, there can be maximally 3 productterms and each productterm can consist of maximally three variables.

The task of the third step, called net_decomposition, is to transform the set of simplified and optimized boolean expressions into expressions that are realisable in NMOS-Technology under the constraints mentioned above. A detailed description and the implementation can be found in a thesis report of P.T.H.M. Van Paassen [2].

3.4 CELLGENERATION

The next step, the cellgeneration, realises the cell-layout of each expression obeying the design-rules for the used technology. Cells in standard-cell technology have a constant width and a variable height depending on the expression to be realized. The input- and output pins are situated along the sides of a cell. Each input-pin agrees with a variable or its negation. The output pin agrees with the negation of the function
A special characteristic of the generated cells is that pins have electrical equivalences on the opposite side of a cell (fig.3.2).

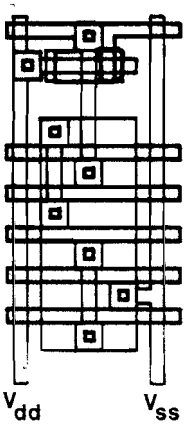


fig.3.2

The electrical equivalences for the v_{dd}- and v_{ss}-line are necessary because otherwise the cells could not be stacked into columns. The electrical equivalences of the other pins result in great advantages in the routing phase.
A detailed description of the cellgenerator and its implementation can be found in a training report of A.T.H.A.J. De Groot [14].

3.5 PLACEMENT

After cellgeneration almost everything is known about a cell, except its position on the chip. A characteristic of the standard-cell concept is that there are a fixed number of columns, choosen by the user. By that he has a great influence on the aspect-ratio of the chip.
Since a detailed routing is not known at the moment of placement, the placement is not capable to fix the positions of the cells exactly. The only thing placement can do is putting a cell on a certain position within a column. The relative position of the cells within a column is fixed. This position is relative because it is possible that special cells, i.e. feed-throughs, are inserted during the routing-phase. After the detailed routing the positions of the columns are exactly known and also the exact positions of the cells within the columns.
The goal of placement is to find positions for the cells in a two-dimensional space in such a way that the expected number of nets connecting the cells is minimal. A mathematical construction, described in articles of R.H.J.M. Otten [3,4], is used to find optimal positions of the cells in a two-dimensional space. A characteristic of this approach is that cells are forming "clusters" in the two-dimensional space. Each cluster is formed by cells which are heavily connected.
Because a standard-cell approach is used the two-dimensional positions of the cells have to be transformed into column-numbers and positions within the columns. So each cell-position in the two-dimensional plane is transformed into a columnnumber and a position within the column.
The mathematical approach, the column-construction and its implementation are described in chapter 6.

3.6 ROUTING

Traditionally routing is divided into two stages, i.e. global and local routing. After the routing phase the layout is constructed. The layout generation is accomplished in the routing module in order to avoid storage-operations on the routing-data.

3.6.1 *GLOBAL ROUTING*

Global routing is a preliminary planning stage for the final routing. The global router decides through which channels individual connections will run. Some connections may be routed around to avoid critical bottlenecks.
The main goal of the global router is to minimize the total wire-length necessary to connect the cells.
The preliminary stage of global routing involves the definition of the routing area. The global router which is used in the system cuts the area into a set of rectangles (fig.3.3).

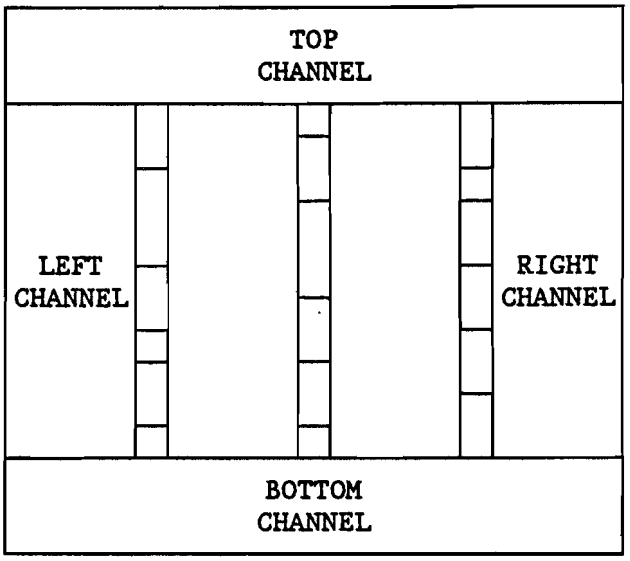


Fig.3.3

The number of rectangles or channels equals the number of columns + 3. There are #columns+1 vertical channels and 2 horizontal channels. The global router uses the electrical equivalences of pins on both sides of a cell. Using these pins eliminates connections, through the top- or bottomchannel, for nets which interconnects more than one channel. Where electrical equivalent pins are not available the global router is allowed to insert feed-through cells. So it is possible for each net to "travel" from a vertical channel to another, without using the top- or bottomchannel. If there are no suitable pins for a net in a channel and for some reason no feed-through cells can be used, the global router must decide to make connections through the top- or bottomchannel.
The goal of global routing is to find an optimal way for each net through the channels using electrical equivalences and feed-through cells. It also cuts the routing are into a number of rectangular channels.
The global router delivers the netlists which are used in the local router. Each channel can be looked upon as a separate routing problem that can be handled by a channel router. The channel router determines a detailed way for each net within a channel.
The global router also delivers the exact positions of the cells and the feed-throughs within a column. The exact position of a column is determined by the local router.
The global router is described in chapter 7.

3.6.2 *LOCAL ROUTING*

The task of the local router is to make a detailed routing within a channel with a minimum number of tracks used, obeying the design-rules of the used NMOS-Technology.
Before the local router can start, the positions of the IO-pins must be known. In the present implementation of the system the IO-pin positions are asked after the placement stage and before the routing of the first channel. There are four possible sides for the IO-pins, i.e. left, right, top and bottom. The user can decide at which side an IO-pin enters the chip. He can fix the y-coordinates at the left or right-side or the x-coordinates of IO-pins at the top- or bottom-side. The exact positions of the IO-pins are known after the routing of the last channel.
For the local routing a "greedy" channel router is implemented. The "greedy" channel router was introduced by R.L. Rivest and C.M. Fiduccia [7]. The main advantage of this router is that it always succeeds usually using no more than one track more than required by channel density. The greedy router wires the channel in a left-to-right, column-by-column manner, wiring each column completely before starting the next. Within each column the router tries to maximize the utility of the wiring produced, using simple, "greedy" heuristics.
The greedy router always succeeds because it may occassionaly add a new track to the channel, to avoid "getting stuck". He is also allowed to make connections "off the end" of the channel if necessary.
The greedy router is described in chapter 7.

Because the routing area is divided into rectangular channels the channel router must route each channel. This has to be done in a certain sequence. The vertical channels have to be routed first. After this the netlists are known for the horizontal channels and they can be routed too.
The local router is described in chapter 7.

3.6.3 *LAYOUT GENERATION*

After the routing process the exact measures of the channels are known so it is possible to give the final positions on the chip to the channels and the columns.
While generating a layout also the power-lines have to be constructed, i.e. the v_{dd}- and the v_{ss}-lines.
The layout generation is also described in chapter 7.

4. THE STANDARD CELL APPROACH

The standard cell approach is one of the most straightforward for automating the layout generation. The logic cells are positioned by the layout software in columns, between which wiring channels are situated (fig.4.1). Power and ground wires are laid out along the sides of the chip so that it does not interfere with the data signals.

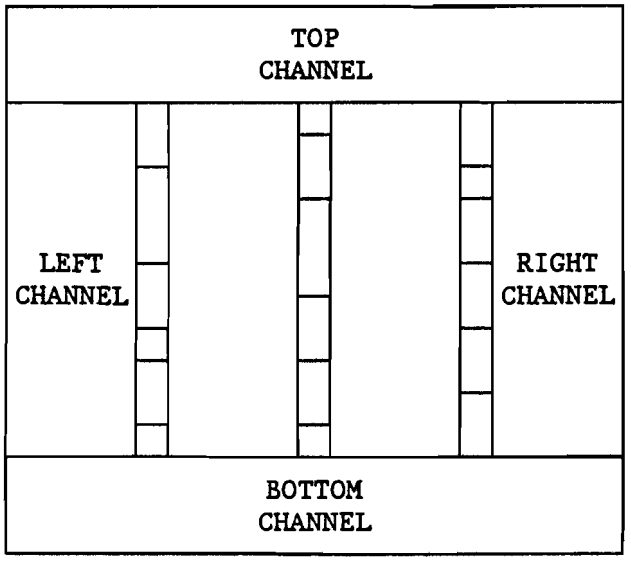


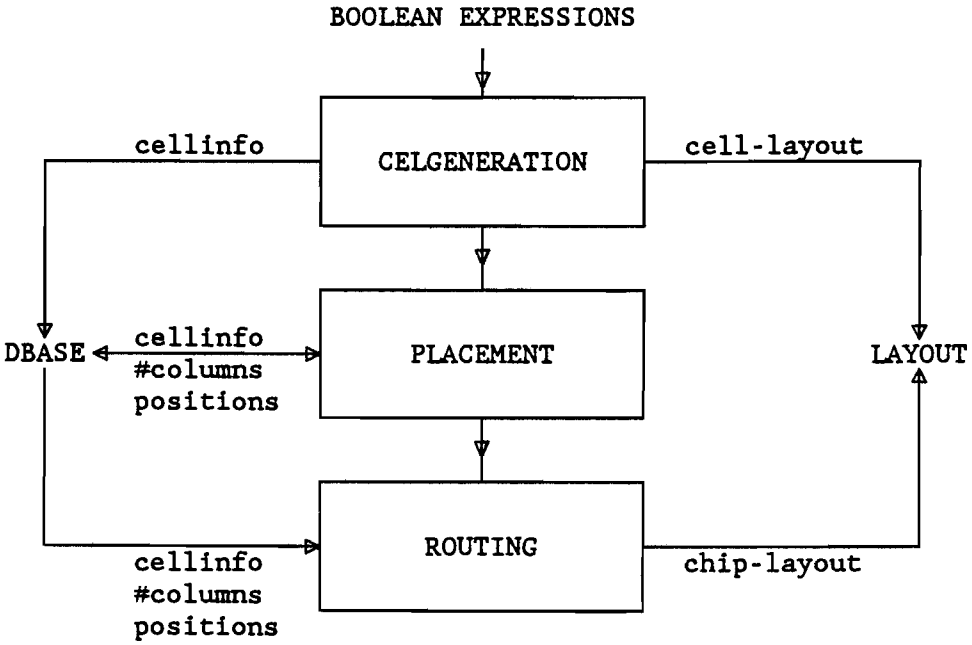
Fig.4.1

Input to standard cell systems are in the form of so-called netlist - a list of cells and connections between cells - usually derived from logic diagrams or logic equations. The standard cell system examines the netlist, determines the position of each of the cells on the chip, and creates the wiring pattern to connect the cells into a working system.
The two steps to a standard cell layout process are placement and routing. The goal of placement as well as routing is to keep the channel widths as small as possible.
In principle no human interaction is required in layout design systems based on the standard cell approach. It is fast because placement of the cells and routing of the wiring are done entirely by computer. Another advantage is that designers need not know details of the IC-technology that will be used to execute the design. The layout rules for CMOS, NMOS and other technologies are built into the algorithms. A design may be reimplemented when more advanced fabrication technologies become available.
The standard cell approach is capable of always achieving a layout with all connections completed and all design-rules

obeyed. The automatic placement and routing system may not be capable to produce a chip within the disired size, performance and power-consumption limits. When this occurs, a human designer must modify the placement and routing, and much of the advantage of automated layout is lost.
The final conclusion might be that standard cell programs perform very well, especially when design time dominates other cost factors such as yield, signal delay and power requirements.

5. THE DATABASE

A database is used to store information of the cells and some global information of the chip. This database is necessary because the last three steps of the macro-cell generation consists of independent programs. These programs have to read and/or write information to the database. The programs using the database are cellgeneration, placement and routing.
The contents of the database are read or changed with procedures which are called during the cellgeneration, placement or routing phase. Each program has its own database-procedures in order to reduce the time needed for interactions with the database.



DATABASE / LAYOUT
INTERACTIONS
Fig.5.1

The cellgenerator reads the boolean expressions from a file produced by the net_decomposition. The layout of the cells is generated and information of the cells is written to the database. The width, height of each cell is stored and also the positions and signalnames of the pins on the boundary of a cell.
The placement program reads the signalnames of each cell and constructs with this information a distance matrix. After the calculation of the positions of the cells, the columns are constructed. The cells are "pushed" into a number of columns. The

number of columns is fixed by the user.
A position of a cell is characterized by a columnnumber and a position within the column. This information together with the number of columns, is written to the database.
Finally the routing program reads the number of columns and cellinformation from the database. This information together with the positions of the IO-pins placed by the user and some additionally parameters for greedy routing, are sufficient to start the routing phase and to construct a complete layout.
A detailed description of the database can be found in a training report of C.H.H. Janssen [15].

6. PLACEMENT

An optimal solution for the placement problem cannot be found within polynomial time. Instead heuristic techniques are employed to find sub optimum solutions. The heuristic used in this chapter can be found in the article 'Automated Floorplan Design' written by R.H.J.M. Otten [3].
Before defining the placement problem some basic definitions are necessary. First wires are attached to pins which carry the electrical signals and are located on the boundary of a cell. Various subsets of these pins, called signal sets, must be interconnected to form nets.
In this treatment the positions of the pins of a cell are not taken into consideration. Instead a cell is regarded as one point that is carrying several signals.
An object function must be optimized to find an optimal location for each cell. The object function must contain data that relates a cell to another. Since the only available initial information is an area and netdata, the only data relating a cell with another is the global interconnection count, i.e. the expected number of nets connecting that cell with another.
The placement problem can now be formulated as follows:
Given a set of cells with signal sets defined on (non-disjoint) subsets of these cells, assign positions in a two-dimensional space to the cells so that the global interconnection count is minimal.
The initial information must be transformed into a distance space, i.e. associating with each pair of modules a unique non-negative real number and with each one-element subset of the set of cells the number 0. A distance space of m objects is completely specified by a matrix D_{mm}, its distance matrix:

$$i \neq j : \delta(\{m_i, m_j\}) = d_{ij} = d_{ji} \in \mathbb{R}^+$$

$$\delta(\{m_i\}) = d_{ii} = 0$$

The distance matrix is a non-negative, symmetric matrix with zeros on the principal diagonal.
The task is to find a suitable metric distance space, because not any metric distance space can be congruently embedded in a euclidian space of some dimension, let alone in a two-dimensional euclidian space.
Schoenberg addressed the problem of congruent embeddings of a distance space into a euclidian space of given finite dimension in a paper published in 1935 and gave necessary and sufficient conditions for the existence of such an embedding.
In the next paragraphs the Schoenberg theory will be explained and also which metric is chosen for the placement problem.

6.1 SCHOENBERG CONSTRUCTION

The m rows of a real matrix Q_{mr}, called a configuration matrix, are to be interpreted as the positions of m points in an r-dimensional euclidian space. Since congruent transformations of their configuration in this space are not of interest, the centroid is assumed to be at the origin, which means that the column sums of Q are 0. Let $\hat{d}_{ij}$ be the distance between the i-th and j-th point, and let $\hat{d}_{0i}$ be the distance of the i-th point from the origin. The cosine law says

$$q_{i.} q_{j.}^T = 1/2 \; (\hat{d}_{0i}^2 + \hat{d}_{0j}^2 - \hat{d}_{ij}^2) \qquad (1)$$

$q_{i.}$ is representing row i in the configuration-matrix Q.

In order to eliminate the distances of the individual points from the origin these distances are expressed as interpoint distances:

$$\hat{d}_{0i}^2 = 1/m \sum_{k=1}^{m} \hat{d}_{ik}^2 - 1/2m^2 \sum_{h=1}^{m} \sum_{k=1}^{m} \hat{d}_{hk}^2 \qquad (2)$$

using the fact that

$$\sum_{k=1}^{m} q_{k.} = 0 \qquad (3)$$

substituting (3) into (2) gives

$$q_{i.} q_{j.}^T = -1/2 \; (\hat{d}_{ij}^2 - 1/m \sum_{k=1}^{m} (\hat{d}_{ik}^2 + \hat{d}_{jk}^2) + 1/m^2 \sum_{h=1}^{m} \sum_{k=1}^{m} \hat{d}_{hk}^2) \qquad (4)$$

or in matrix form with $Z_{mm} = I(j_m) - 1/m \; J_{mm}$: (5)

$$Q \; Q^T = - 1/2 \; Z \; \bar{\hat{D}^2} \; Z \qquad (6)$$

A bar $\bar{}$ above a operation denotes an elementwise operation.

where I is a diagonal matrix with the values of the argument on the diagonal and zeros off the diagonal, J is the matrix, and j the vector of all ones, and D is the matrix with zeros on the diagonal and $\hat{d}_{ij}$ in the i-th row and the j-th column.
The result obtained is that (6) applied to a distance matrix $\hat{D}$ of a configuration of m points in the r-dimensional euclidean space yields a positive semidefinite matrix of which the rank does not exceed r. But also the converse is true: whenever a distance matrix D substituted in (6) gives a positive semi-definite matrix, there exists a congruent embedding of this distance space into a euclidean space. So there exists an orthonormal matrix U such that

$$U\ I(\Lambda_m)\ U^T = -\ 1/2\ Z\ \hat{D^2}\ Z \tag{7}$$

where Λ_m is the vector of eigenvalues of the matrix in the right hand side (in decreasing order). By setting (all eigenvalues in Λ are nonnegative)

$$Q = U\ I(\Lambda^{1/2}) \tag{8}$$

the configuration matrix of the configuration realizing the interpoint distances given by D is obtained. The columns of U are the normalized eigenvectors of (6), so Schoenbergs theorem states: A distance space with distance matrix D is embeddable in an r-dimensional euclidean space fif its Schoenberg matrix

$$-\ 1/2\ Z\ D^2\ Z \tag{9}$$

is positive semidefinite and has rank r or less.
The matrix of which the columns are the eigenvectors of the Schoenberg matrix with length equal to the square root of the associated eigenvalue, is the configuration matrix of the configuration realizing the distances of D.
This theorem is a complete solution to the problem of embedding a given distance space congruently into a given euclidean space. It is now possible to answer the question whether a certain distance space derived from the cel and net data available, can be embedded in the euclidean plane.
In general this will not be the case. Quite often the structure is almost embeddable in the plane, i.e. there is an embedding in the two-dimensional euclidean space realizing distances d'_{ij} instead of d_{ij} such that the function

$$\phi = \sum_{i=1}^{m} \sum_{j=1}^{m} (d_{ij}^2 - d'^2_{ij}) \tag{10}$$

is relatively small. To obtain such an embedding in case there is some euclidean space in which the given distance space can be congruently embedded, the r-dimensional configuration might be projected into a suitable two- dimensional subspace. If T_{rr} is the matrix of an orthonormal transformation of which the first two columns span the projection plane, the value of ϕ is

$$\phi(T) = \sum_{i=1}^{m} \sum_{j=1}^{m} \sum_{k=3}^{r} (q_{i.}^T t_{.k} - q_{j.}^T t_{.k})^2 \tag{11}$$

which is minimal if the other r-2 columns of T span the same subspace as the eigenvectors associated with the r-2 smallest eigenvalues of the Schoenberg matrix. Or, equivalently, if the projection plane is spanned by the two eigenvectors associated with the two largest eigenvalues of the Schoenberg matrix. The

corresponding two-dimensional configuration is given by the first two columns of the configuration matrix.

6.2 THE DUTCH METRIC

The dutch metric yields a distance space which is congruently embeddable into a finite-dimensional euclidean space, has no distance exceeding 1, takes all connections of the cell involved into account (not only the nets they have in common), and allows for weighting individual nets (for example to force critical nets to be short).
With m cells and n nets the netlist can be represented by a (0-1) matrix P_{mn}. There are one-to-one correspondences between the rows and the cells, between the columns and the nets, and between the pins and the nonzero entries of P. An entry p_{ij} equals 1 if the corresponding cell and net share a pin. Otherwise $p_{ij}=0$. Adopting the same order for the nets as in P the weights assigned to the nets can be represented by a positive vector w_n. The higher the weight of a net, the closer the cells connected by this net should be together. From P and w the distance matrix D is derived.
It is better to base the matrix D on the presence of connections, and not on the absence of connections. Therefore it is more natural to speak about proximity and a nonnegative proximity matrix C_{mm}, than about distances. The elements on the diagonal of C should be the same, and greater than all the off-diagonal elements. If the diagonal elements are chosen to be 1, a monotone increasing function of $1-c_{ij}$, bounded on [0,1] and vanishing at the origin might serve as a distance function.
A cell having many pins needs a large perimeter and, possibly, claims proximity to many other modules. In the two-dimensional chip environment this cannot be realized by arbitrarily short center-to-center distances. Besides, those other cells do not claim proximity only to that module. A measure for proximity should therefore express a balanced sharing of the neighbourhoods of the cells. Moreover, since it is also desirable to express the fact that certain nets should be kept short the following mutual proximity measure seems to be appropriate:

$$c_{ij} = \frac{\Sigma \{ w_k \mid p_{ik}=1 \wedge p_{jk}=1 \}}{\Sigma \{ w_k \mid p_{ik}=1 \vee p_{jk}=1 \}}$$

Clearly, proximity thus defined is a number in the closed interval [0,1]. The dutch metric, defined by

$$\tilde{D^2} = J_{mm} - C$$

This metric yields a distance space with positive semi-definite

Schoenberg matrix. A proof for this statement can be found in [3].

6.3 PLACEMENT IMPLEMENTATION

Before describing the implementation of the placement program, the datastructure is considered.

6.3.1 *DATASTRUCTURE PLACEMENT*

Information about the cells is stored in two arrays. The first array, i.e. totsig_in, is a two-dimensional array that contains the input-signalnames of each cell. The output-signalname is equal to the number of the cell. Signalnames are represented as integers.
The remaining information such as cellnumber, cellwidth, cellheight, cellcolumn and cellposition is stored in the array cellinfo. The array cellinfo is of type celldata. The type celldata is as follows:

```
totsig_in : array[1..maxcell,1..maxpin] of integer;
cellinfo  : array[1..maxcell] of celldata;
celldata  = record
              cellnumber : integer;
              width      : integer;
              height     : integer;
              column     : integer;
              ypos       : integer;
            end;
```

6.3.2 *IMPLEMENTATION*

The first step in the placement phase initialises the datastructure. Cellinformation is read from the database and put into the datastructure. By using this cell-information the Schoenberg matrix is constructed. The next step is the calculation of eigenvectors and eigenvalues of the Schoenberg matrix. With these results the cell-configuration-matrix can be calculated in a n-dimensional space. In par.6.1 is showed that the first 2 columns of the cell-configuration-matrix are representing the best obtainable positions in a 2-dimensional plane.
Hereafter the cells are placed in a number of columns, given by the user. Because the height of each cell and the number of columns is known, the mean-height for a column is calculated. The placement program tries to construct a column with approxiamately mean-height.

The cells are sorted on rising x-coordinates. By starting with the cell with the smallest x-coordinate, the cells are assigned to columns. The column-assignment is done by using a mean-height criterium.
This criterium assigns a cell i to a column j if :

height cel i + height column j ≤ mean-height+1/2.height cel j

If cell i satifies this criterium the height of column j is increased with the height of cell i. The column-assignment proceeds with the next cell. Again is tried to assign this new cell to column j. If cell i does not satisfie the criterium then column j is full. Cell i will be the first cell that is assigned to column j+1.
This column-assignment is continued until all cells are assigned to columns.
All cells having the same columnnumber are ordened on rising y-coordinates. The cell (suppose cell i) with the smallest y-coordinate is placed on y-pos cell i = 0. The next cell is placed on:

ypos cell j = ypos cell i + height cell i

This process continues untill all columns are constructed.
The last step in the placement phase is the storage in the database of the columnnumber and the y-position of each cell. Also the number of columns is stored in the database.
The whole placement procedure is also described below in pseudo-pascal.

```
procedure placement:
  begin
    read number_of_cells from database;
    for i:=1 to number_of_cells do
      begin
        read heigth, width and signalnames of cel i;
      end;

    calculate matrix for "dutch metric" with celinformation;
    calculate matrix for Schoenberg-construction;
    calculate eigenvalues and eigenvectors of Schoenberg-matrix;
    calculate cell-configuration-matrix in a n-dimensional space;
    destillate 2-dimensional configuration from n-dimensional configuration;

    read number_of_columns given by user;
    sort cells on x-coordinates;
    assign cells to columns with a mean-heigth criterium;
    for i:=1 to number_of_columns do
      begin
        sort cells in column i on y-coordinates;
        give cells new y-coordinates according to their heigths;
      end;

    for i:=1 to number_of_cells do
      begin
        write column and y-coordinate of cel i to database;
      end;
    write number_of_columns to database;
  end;
```

Placement procedure

7. ROUTING

Routing is divided into two stages, i.e. global and local routing. These stages will be described in the next paragraphs. But first the grid is described used for the routing phase.

7.1 THE ROUTING GRID

The greedy router works on a grid. The grid is determined by the designrules of the used technology and by the requirement that the greedy router is allowed to introduce all vias between a horizontal and a vertical wire that are necessary.
Normally two polysilicon-layers and two aluminium layers are separated by a distance of 6 μ. But when a via is introduced the distance between two wires must be greater. In the worst case two vias are introduced next to each other either vertically, horizontally or both. This means that constraints exists for the grid in both horizontal and vertical direction.
The dimension of a via is 12μ x 12μ. Two vias next to each other must be separated by a gap of 6μ. This implicates (fig.7.1) that the grid in both vertical and horizontal direction must be 18μ.

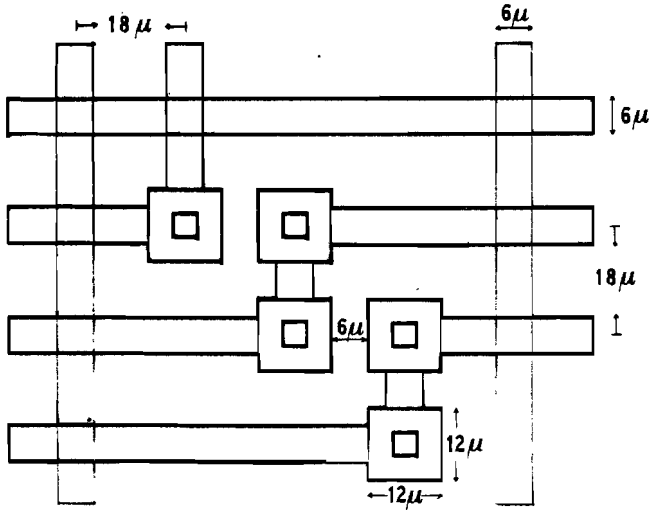


Fig.7.1

7.2 GLOBAL ROUTING

After placement of the cells, global routing has to be performed. During the global routing phase the system determines through which channels each net has to be routed. If for instance two pins in different channels have to be connected, the connecting wire can use

1. a feedthrough.
2. the top channel.
3. the bottom channel.

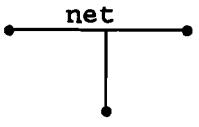
The global router has to choose the best alternative.
If a number of pins have to be connected, the global router has to determine which connections have to be made.
For instance:

The objective of the global router is to minimise the length of the nets. In this chapter firstly the global routing problem will be formulated. Next the theory of this problem will be addressed. At last the details about the implementation will be given.

7.2.1 *PROBLEM DEFINITION*

A wire is a connection between two pins.

A net is a connection between two or more pins. So a net can also be a wire.



A signal is the collection of all the pins which have to be connected to each other. (All these pins have the same name.)

Because the placement of the cells has already been performed, the y-positions of all the pins in the layout are known. Because we don't know the width of the channels, the x-position of each

pin is however not known yet. So a guess of the width of the channels has to be made.
The global router routes the layout signal by signal. This means that all the pins with the same signal name are considered together in one global routing step.
The nets between the pins have to go through the channels. There is however also a possibility to create feedthroughs between two cells.
The global wiring problem can be described as:
Given a number of pins, connect these pins in such a way that the total wire length is as small as possible. All the connecting nets, have to run through the channels and/or through the feedthroughs.

7.2.2 *THEORY*

The problem can be mapped into a graph G(V,E). V being the set of vertices in the graph and E denoting the edges in the graph. Each vertex $v \in V$ in the graph corresponds with a pin in the layout. Each edge $e \in E$ in the graph corresponds with a wire in the layout. With each edge a number c is related. This number denotes the cost of that edge, being an estimation of the length of that wire.
For each signal a full graph can be constructed. A full graph is a graph in which each pair of vertices is connected by one and only one edge. The cost of each edge can be computed. In this full graph we can construct a minimum cost spanning tree. The edges in this minimum cost spanning tree are the wires which have to be routed in the layout.
Given a number of points in the X-Y plane, the minimum cost spanning tree is in general not the shortest possible connection between these points. The shortest connection is found by constructing a Steiner tree [5]. Finding a Steiner tree is a NP-complete problem. Because of the simple channel structure it is not necessary that the minimum spanning tree solution is always worse than the Steiner tree solution.
For the time being the minimum cost spanning tree algorithm is implemented.
The minimum cost spanning tree algorithm can be found in [6]. In the algorithm the edges are sorted with respect to their cost. The solution is initialized by assigning a tree to each node in the graph, so the whole graph is covered by a number of trees (a forrest). Then the cheapest edge is added to the forest such that this edge joins to different trees in the forrest. This is repeated until only one tree (the minimum cost spanning tree) is left.

7.2.3 *COMPUTATION OF THE COST OF AN EDGE*

For each edge in the full graph the cost has to be computed. This is done in the following way: If a wire stays in the same

channel the cost is equal to the sum of the difference in x positions and the difference in y position (estimated length of the wire). If the two pins are not in the same channel, there is tried to go from one channel to the other channel with a feedthrough. The cost of the edge is equal to the estimated length of the wire.
Because not all the cells have the same height, the heights of the columns of cells will not be equal. (the heighest column has the height "maxcolheight"). Because of the free space in some columns, the global router is able to make feedthroughs between two cells in a column (shifting up the cells and feedthroughs above the new feedthrough). The user is able to allow a number of extra feedthroughs in the heighest column.
If a wire has to go from one channel to an other channel and no feedthroughs in that column are allowed any more, the global router routes the wire through the top-channel (if the y-position of the wire at that moment is greater then 1/2 * maxcolheight) or through the bottom channel (if the y-position of the wire at that moment is less than 1/2 * maxcolheight).

7.2.4 *DATA DELIVERED BY THE GLOBAL ROUTER*

The global router has for each channel to deliver the lists of pins which have to be routed. Pins with the same name will be connected by the local router.
So if two pins have to be connected they have to be incorporated in the lists of pins to be routed and they have to have the same name (number). If a wire goes for instance through the bottom channel, pins have to be created at the appropriate sides of the channels, to let the local router route the signal to the appropriate place. Also if a feedthrough is created pins have to be incorporated in the corresponding lists.

7.2.5 *NAMES OF THE PINS*

If two pins have to be connected by a wire, the pins have to have the same name. However, if there are two wires from the same signal within the same channel, the pins for each wire are not allowed to have the same name because then there will be introduced an extra connection between the two wires. So for each wire, different names for the corresponding pins have to be generated. If however a pin is incident with two wires, all the three pins incident with the two wires have to have the same name.

7.2.6 *ALREADY CONNECTED PINS*

It is possible that a cell has a number of pins which are connected internally in the cell. The global router doesn't need to connect these pins to each other any more. The internal connections of two pins are indicated by a pointer "connect" (in

the database) pointing to a internally connected pin. This pointer list is cyclic, so if all the internally connected pins are scanned, the pointer points again to the first element. If a number of pins are internally connected, they are, during the generation of the minimum cost spanning tree, initially placed in the same sub tree. Because of this the algorithm treats them as already connected pins.

7.3 LOCAL ROUTING

Consider a rectangular channel; the four sides of the channel are labeled T(op), B(ottom), L(eft) and R(ight) (figure 7.5).

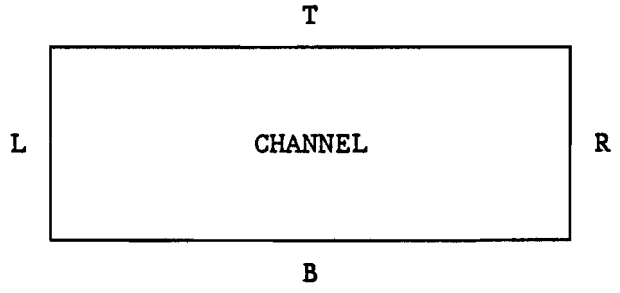


Fig.7.5

Along the sides of a channel terminals may appear on each gridpoint. A number is assigned to each terminal. Terminals with the same number i must be connected by net i, while those with number 0 designate unconnected terminals.
Two layers are available for routing, horizontal tracks are on one layer and vertical tracks on the other. Nets are laid on tracks. Horizontal tracks are isolated from vertical tracks, and connections between them are made through via holes.

A channel-routing problem is specified by giving (fig.7.6):

- A channel-length λ. Most of the routing will lie within the channel whose left end is at x=0 and right end is at x=λ+1, on the vertical columns at x-coordinates 1,...,λ, although columns outside the channel may also be ususeed.

- Top- and bottom connection lists $T = (T_1,\ldots,T_\lambda)$ and $B = (B_1,\ldots,B_\lambda)$. T_i (respectively B_i) is the net number for the pin at the top (respectively bottom) of the i-th column (at x=i), or is 0 if no such pin exists.

- The left and right connection sets, L an R, specifying which nets must connect to the right and left ends of the channel. They are sets since a net need connect at most once to an end of the channel, and because the relative ordering of such connection may be chosen by the channel router.

A solution to a channel-routing problem specifies:

- The channel width ω, i.e. the number of horizontal tracks used. These tracks are at y-coordinates $1,\ldots,\omega$. A channel router tries to minimize ω.

- For each net n, a set of connected horizontal and vertical wire segments whose endpoints (x,y) with $1 \leq y \leq \omega$. Except that segments with endpoints (i,0) or $(i,\omega+1)$ must be included if $T_i = n$ or $B_i = n$. Endpoints with $x<1$ or $x>\lambda$ are legal but should be avoided. A net in L (respectively R) must have a segment touching the line $x=0$ (respectively $x=\lambda+1$). Two segments in the same direction are on the same layer, so they may not touch if they are for different nets. Two segments for the same net in different directions that touch a grid point are said to be connected by a contact or via at that point. If the segments were for different nets there has to be a crossover.

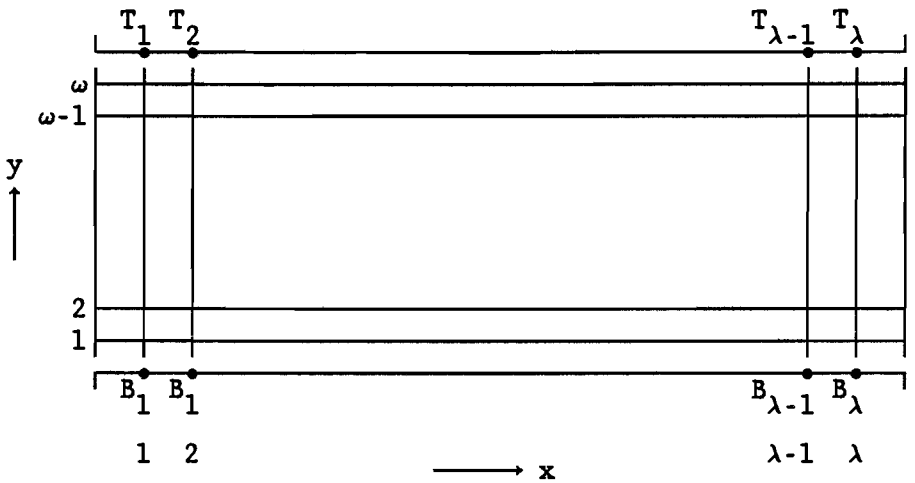


Fig.7.6

7.3.1 *A 'GREEDY' CHANNEL ROUTER*

Channel routing has become a very popular method of routing integrated circuits. Typically, the wiring area is first divided into disjoint rectangular channels. A global router then determines which channels each net traverses. Finally a channel router computes a detailed routing for each channel. This approach is effective because it breaks down the overall problem into a number of simpler problems and simultaneously considers all nets traversing each channel.
Since the general channel-routing problem is NP-Complete [8] there is a need for good practical heuristics. The heuristics presented in this paragraph can be found in [7].

The 'greedy' algorithm exploits a novel control structure: a left to right column-by-column scan of the channel, where the router completes the routing for one column before proceeding to the next. In each column the router acts in a 'greedy' manner trying to maximize the utility of the wiring produced.
The first step of the algorithm is to make connections to any pins at the top and bottom of the column. These connections are minimal; no more vertical wiring is used than is needed to bring these nets safely into the channel, to the first track which is either empty or contains the desired net.
The second step tries to free as many tracks as possible by making vertical connecting jogs that collapse nets that currently occupy more than one track. This step may complete the job of bringing a connection from a pin over to a track that its net currently occupies.
The third step tries to shrink the range of tracks occupied by nets still occupying more than one track, so collapsing these nets later will be less of a problem. Since freeing up tracks has high priority, jogs made here have priority over jogs made in the next step.
The fourth step makes preference jogs to move a net up if its pin is on the top of the channel, and down if its next pin is on the bottom. The router chooses longer jogs over shorter ones if there is a conflict. This tends to maximize the amount of useful vertical wiring created. These jogs are effective at resolving upcoming conflicts, even though no explicit consediration of these conflicts is made.
The fifth step is only needed if a pin could not be connected up in step one because the channel is full. Then the router adds a new track to the channel between existing tracks, and connects the pin to this track. The old tracks are renumbered.
When the routing of a column is complete, the router extends the wiring into the next column and repeats the same procedure. The following paragraphs make precise the algoritm just sketched.

The input for the greedy router consists of

- a specification of a channel-routing problem
- three non-negative integer parameters:
  - initial-channel-width
  - minimum-jog-length
  - steady-net-constant

The greedy router starts with the given initial-channel-width. A new track is added whenever the current channelwidth becomes unworkable. The router does not begin over when a new track is added, so different initial widths may give different results.
The router will make no jogs shorter than minimum-jog-length. A higher setting reduces the number of vias and thus produces more acceptable solutions, while a lower setting tends to reduce the number of tracks used.

When routing a given column, the greedy router classifies each net which has a pin to the right as either rising, falling or steady. A net is rising if its next pin after the current column will be on the top of the channel (say in column k), and the net has no pin on the bottom of the channel before column k+steady-net-constant. Falling nets are defined similarly. Steady nets are the remaining nets. A larger value for the steady-net-constant reduces the number of times a multipin net changes tracks.
By running the router several times, with different initial parameter settings, the best solution obtainable, for the given channel-routing problem, can be determined.

7.3.2 *THE GREEDY ROUTER*

In this paragraph a detailed description of the greedy router algorithm will be given. The operations of the router are illustrated using a set of before/after figures of each step. These figures describe what happens in a single column, and should be interpreted as follows. Nets entering a column from the previous column are shown extended up to the current column. If the net has pins to the right of this column, the net is shown extended toward the next column with an arrowhead. Otherwise (if the net has no pins to the right), no arrowhead is shown. A '+', '-', or '+/-' may be shown next to an arrowhead to denote rising, falling, or steady nets.

Assign tracks to nets at left end.

For each net n in L give n a distinct track in the range 1,...,initial-channel-width on which to enter the channel from the left end. Put the rising nets above the steady nets above the falling nets and generally group the nets at the centre of the channel.

Route channel from left to right.

For each column i, for i=1,2,..., until i≥n and no split nets remain to be collapsed do the following:

Step 1: Make feasible top and bottom connections in a minimal manner.

If T_i or B_i is nonzero, bring in that net if possible to the nearest possible track which is either empty or already assigned to this net, by running a vertical wire from the edge of the channel to the desired track (figure 7.7). Note that if there is an identical net on a track somewhere in the current column the net to be brought in is not routed to this track if there is an nearer empty track. This means that n is temporarily assigned to an additional track (figures 7.8 and 7.9). Also note that a new net cannot be brought into a full channel in this step (figure 7.10).

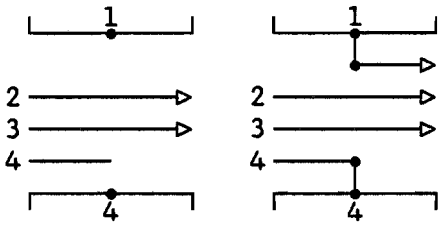


Fig.7.7

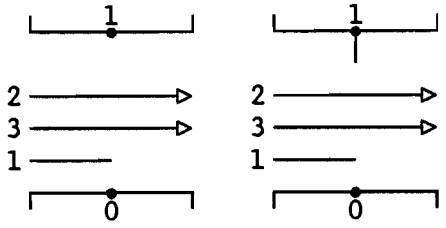


Fig.7.8

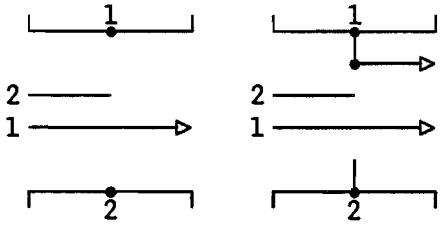


Fig.7.9

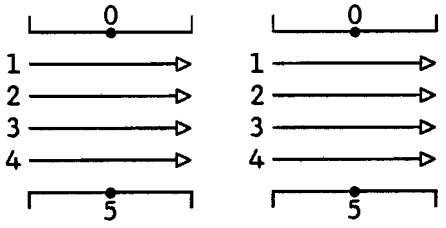


Fig.7.10

If T_i and B_i are both nonzero and $T_i \neq B_i$ try to bring in both nets. But if there is a conflict (overlap) bring in the net which can be brought in with the least wire, and leave the other net to be brought in the third step (figure 7.11).
As special case, if there are no empty tracks, and net $T_i = B_i \neq 0$ is a net which has connections in this column only, then run a vertical wire from to bottom of this column (figure 7.12).

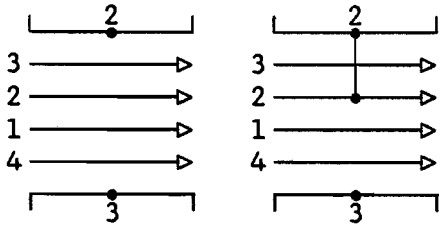


Fig.7.11

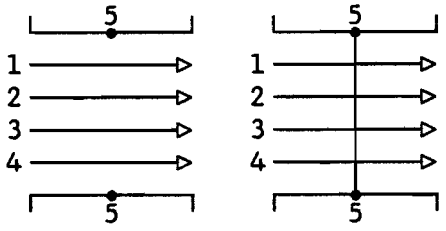


Fig.7.12

Step 2: Free as many tracks as possible by collapsing split nets. Add vertical segments in this column to collapse split nets in a pattern that will create the most empty tracks for use in the next column. Define a collapsing jog to be a piece of vertical wire which connects two tracks holding the same net without crossing another net holding that net.
Define a pattern to be any set of collapsing jogs for which no jogs overlap any vertical wiring placed in the first step.
The number of such pattern to consider may be exponential in the number of collapsing jogs there are to consider. Find the pattern which creates the most empty tracks by a small but complete

combinatorial search (figures 7.13 and 7.14).

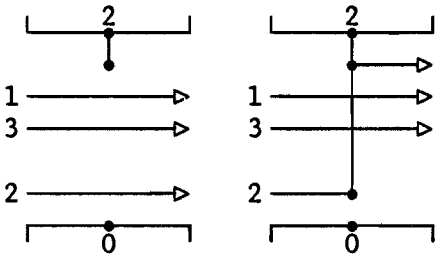


Fig.7.13

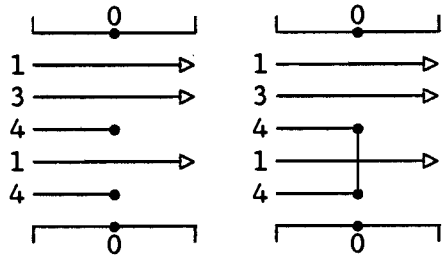


Fig.7.14

A pattern will free up one track for every jog it contains, plus one additional track for every net it finishes. The pattern finishes a net n if it totally connects up the dangling end for n and n has its last pin by column i.
Resolve any ties between pattern that free up the most tracks by choosing the pattern which leaves the outermost uncollapsed split net as far as possible from the channel edge. If necessary consider the second outermost such net, etc (figure 7.15). Resolve any remaining ties by choosing the pattern with largest sum of jog lengths (figure 7.16).

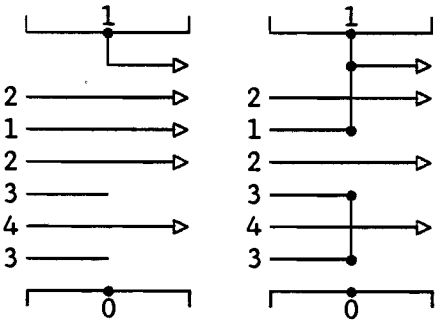


Fig.7.15

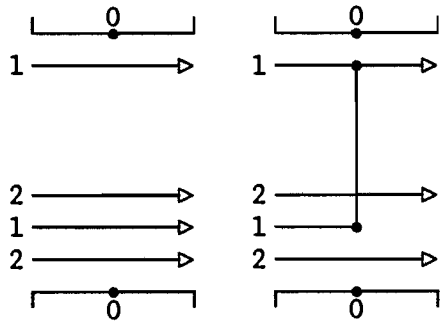


Fig.7.16

Add appropriate vertical wiring for each jog in the winning pattern, and for each such jog which connects a track y_1 to a track y_2 (assume $y_1 < y_2$) for some net n, free up track y_1. This is an arbitrary choice that might get modified in the next two steps.
Note that this step will typically collapse a net that was temporarily brought in the first step when that net had a previously assigned but more distant track.

Step 3: Add jogs to reduce the range of split nets.
For each uncollapsed split net, try to reduce the range of tracks assigned to the net by adding vertical jogs that have the effect

of moving the net

- from the maximum track which is occupied by that net, to the lowest possible empty track.
- from the minimum track which is occupied by that net, to the highest possible empty track (figure 7.17).

Because of step 2, no collapsing will occur, but the difficulty of collapsing the remaining split nets may be reduced.
Make no jogs which are shorter than minimum-jog-length or which would be incompatible with vertical wiring already placed in this column by previous steps.
If a jog for net n is made from track y_1 to track y_2, free up track y_1.

Step 4: Add jogs to raise rising nets and lower falling nets.
Consider all the unsplit (i.e. nets found only once in the current column) rising and falling nets being routed in order of decreasing distance from their track to their target edge (the upper edge of the channel for rising nets and the lower edge of the channel for falling nets).
Try to add a vertical jog to move that net to an empty track which is as close as possible to its target edge (figure 7.18).

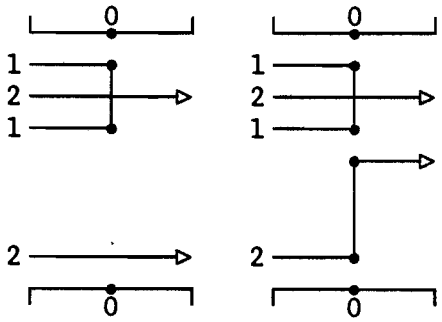


Fig.7.17

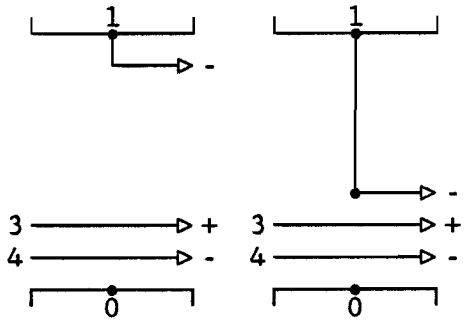


Fig.7.18

Make no jogs which are shorter than minimum-jog-length or which would be incompatible with vertical wiring already placed in this column by previous steps.
If a jog for net n is made from track y_1 to track y_2, free up track y_1.

Step 5: Widen channel if needed to make previously not feasible top or bottom connections.
If a net T_i or B_i could not be brought in to a track in the first step, create a new track for this net and bring the net in to this track. Place this track as near the centre of the channel as possible between existing tracks, subject only to the constraint that the desired connection to the edge of the channel

can be made (figure 7.19).
If a new track lies between tracks previously numbered by k and k+1, all old tracks at y-coordinates k+1 and greater now have their y-coordinates increased by one.

Step 6: Extend to the next column.
Each net n that occurs only once in the current column and n has no pins after column i, is finished. The track of this net can be freed up.
Then for each track y which has a net, that is not finished, extend the dangling end along track y into column i+1 with appropriate horizontal wiring (figure 7.20).

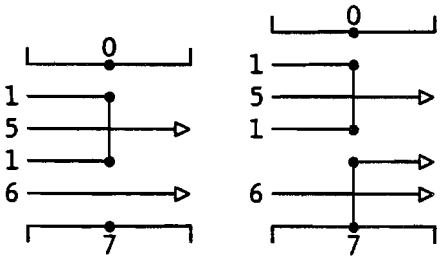


Fig.7.19

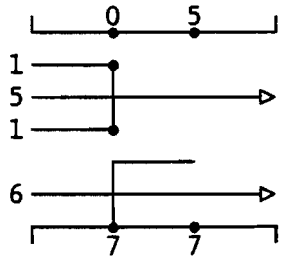


Fig.7.20

7.4 ROUTING IMPLEMENTATION

Before describing the implementation of the routing, the necessary datastructure is considered.

7.4.1 *DATASTRUCTURE ROUTING*

Information about each channel has to be stored in the datastructure. The main datastructure can be seen in fig.7.21. The structure contains #columns+3 records. Each record contains information and pointers to linked lists. A channel record consists of the following fields:

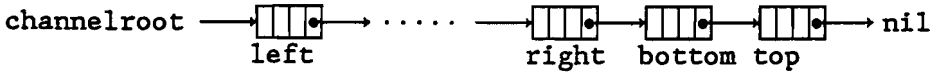


Fig.7.21

```
number           : number of channel
xl,yb            : left-bottom position of channel in layout.
top              : pointer to top-netlist of channel.
bottom           : pointer to bottom-netlist of channel.
left             : pointer to left-netlist of channel.
right            : pointer to right-netlist of channel.
density          : channeldensity
first_net_pnt    : array of pointer to tracklists
first_column_pnt : pointer to columnlist
signalroot       : pointer to signallist
initwidth        : initial width, necessary for the greedy router
width            : width of the channel
length           : length of the channel
via              : number of vias used in the channel
PS_length        : total length of poly-silicon wires used
                   in the channel
IN_length        : total length of aluminium wires used
                   in the channel
next             : pointer to next channel record
```

The fields left, right, top and bottom of the channelrecord are containing pointers to a linked list of net-records. The netlists delivered by the global router are stored in these lists. A netlist record contains of the following fields (fig.7.22; this figure shows the netlist of the top-side of a channel):

Fig.7.22

```
number   : number of net
ypos     : position of net
connect  : boolean for connection y/n
next     : pointer to next net
```

The connect-field can be removed because all the nets which have not to be connected, are not stored in the netlists.

The pointers first_net_pnt and first_column_pnt are pointing to resp. the first horizontal and the first vertical wire in each channel. The horizontal and vertical wires in a channel are stored separately.

The horizontal wires (aluminium) are stored by using a tracklist-array (fig.7.23).

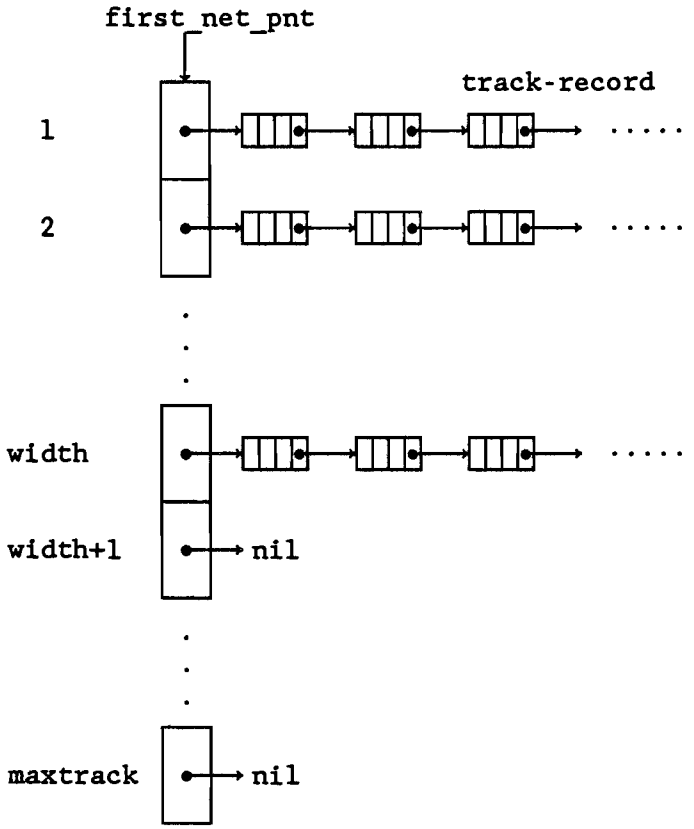


Fig.7.23

Each element of this array contains a pointer. This array can contain a maximum number, i.e. maxtrack, of pointers. The number of pointers which don't have the value nil equals the width of the channel. Each "not-nil"-pointer points to a linked list of horizontal wires in the same track.
A horizontal wire (aluminium) that begins and ends with a via without containing a via in between, is stored in the linked list of the concerned track.
A record describing a horizontal wire (trackrecord, fig.7.23) contains the following fields:

```
startcolumn : startcolumn of horizontal wire
endcolumn   : endcolumn of horizontal wire
netnr       : netnumber
next        : pointer to next trackrecord in the same track
```

An exception is made for nets "touching" the channelside, i.e. the left or rightside. If a net starts at the leftside the startcolumn-filed has the value 0 and if a net ends at the rightside the endcolumn-field has also the value 0. Nets starting at the leftside and ending at the rightside without vias in between are forbidden.

The vertical wires (poly-silicon) are stored in a linked list of column-start-records (fig.7.24). Each column has its own list of vertical wires except the columns that have no vertical wires.

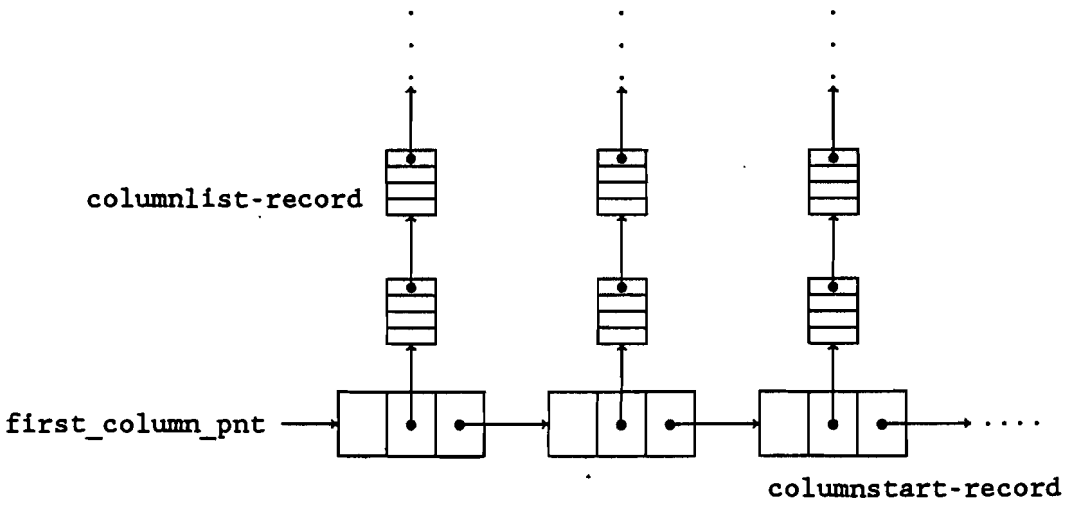


Fig.7.24

```
columnstart-record
   columnnr  : number of column
   first_net : pointer to first vertical wire in column
   next      : pointer to next columnstart-record

columnlist-record
   starttrack : starttrack of vertical wire
   endtrack   : entrack of vertical wire
   netnr      : netnumber
   next       : pointer to next columnlist-record
                in the same column
```

The columnstart-record contains a pointer that points to the first vertical connection (poly-silicon) in the column. A vertical wire that begins and ends with a via without a via in between, is stored in the linked columnlist of the concerned column. An exception is made for the vertical wires "touching" the channelside, i.e. top- or bottomside. If a net touches the top- or bottomside the endtrack resp starttrack contains the value 0. Vertical connections between the top and bottom in the same column are allowed.

All signals appearing in the channel are stored in the signallists (fig.7.25). A signal info-record contains pointers to the first and last appearance of the concerning signal. If a signal occurs more than once in a channel then the pins containing the same signal have to be ordered. The pins are ordered on increasing positions.

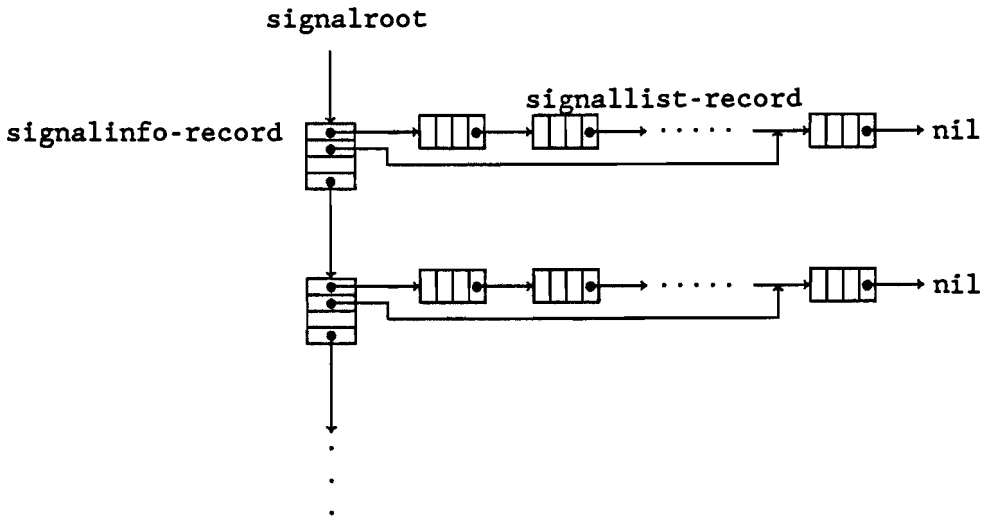


Fig.7.25

If a signal appears in the leftlist then it is the first record in the linked signallist. If a signal appears in the rightlist it is the last record. All signals in between are ordered according to their positions on increasing coordinates. It does not matter if a pin is part of the top- or bottomlist.

7.4.2 *DATASTRUCTURE OF THE GREEDY ROUTER*

The greedy router works in a left-to-right, column-by-column manner. So when the router works in a column i, the columns 1 to i-1 are finished. The vertical wires in these columns can be stored in the datastructure. The horizontal wires have to be treated differently, because it could be possible that a horizontal wire starts in a column that is already finished and ends in a column that is not yet reached by the greedy router. So a check has to be made if a connection is established in the column currently under construction by the router. The check that has to be made is if a via exists on one or more tracks in the workcolumn. If this is the case then there are three possibilities:

1) a horizontal wire starts in the current column.
2) a horizontal wire ends in the current column.
3) a horizontal wire end and starts on the same track again.
   (these wires are containing the same net)

According to these three possibilities different actions have to be carried out on the datastructure. In case 1 a horizontal wire has to be opened. This means a record is linked at the tail of

the tracklist in the concerned track. The startcolumn is known but the endcolumn not. The horizontal wire is opened now.
In case 2 a opened record has to be closed because the endcolumn is known.
In case 3 a record has to be closed and a new record has to be opened.
An advantage of the used datastructure is that vias have not to be stored, because the start and end of a wire implicates a via, except the wires that are "touching" the channelsides.
The datastructure is already described in the previous paragraph. The tracklist array contains pointers to a linked list of horizontal wires. The linked list of columnstart-record contains pointers to a linked list of vertical wires.

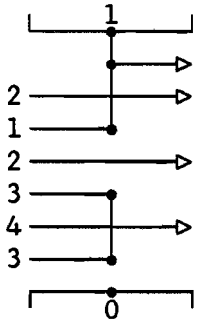
The datastructure for the greedy router must be separated in two parts. The first part is the structure that is already finished by the router. The second part is the workstructure of the router.
The already finished columns and all closed horizontal wires are forming the first part. The second part consist of a workcolumn and all the opened horizontal wires.
The workcolumn of the greedy router consists of four arrays describing the current situation of the tracks in the workcolumn.

1) **workarea**
   An array of numbers describing which net is available on which track. (e.g. workarea[3]=2, net 2 appears on track 3)

2) **vert_connect**
   An array that is describing which kind of vertical connection is realised. The vertical connection types are:

- nocon : no vertical connection
- botwire : connection with bottom without a via on the concerned track
- botcon : connection with bottom with a via on the concerned track
- topwire : connection with top without a via on the concerned track
- topcon : connection with top with a via on the concerned track
- topbotwire : connection with top and bottom without a via on the concerned track
- topbotcon : connection with top and bottom with a via on the concerned track
- jog : part of a connection between two tracks without a via on the concerned track
- jogcon : part of a connection between two tracks with a via on the concerned track

Example :



workarea[7] = 1	vert_connect[7] = topcon
workarea[6] = 2	vert_connect[6] = topwire
workarea[5] = 1	vert_connect[5] = topcon
workarea[4] = 2	vert_connect[4] = nocon
workarea[3] = 3	vert_connect[3] = jogcon
workarea[2] = 4	vert_connect[2] = jog
workarea[1] = 3	vert_connect[1] = jogcon

3) status_vert_connect
An array describing if a connection between a vertical wire and a horizontal wire is definite or temporarily. Tracks having no vias in the workcolumn have no status. So the status possibilities are:

- definite
- temporarily
- no_status

It is necesssary to give a connection a status, because a net

could be brought into the channel on an empty track. In this case a net has a temporarily status. A temporarily connection can change in a definite one or in a connection with no status. A definite connection implicates a via on the concerned track.

4) **next_workarea**
An array describing the nets in the next workcolumn. The final next_workarea in each column initialises the array workarea for the next column.

As said before these 4 arrays are the datastructure for the greedy router while a column is under construction. In each step of the greedy algorithm the contents of these 4 arrays are changed according to the used heuristics. After a column is finished the column- and tracklists are updated.

7.4.3 *IMPLEMENTATION*

The best way to explain the routing phase is to use pseudo-pascal procedures.

```
procedure Route;
  begin
    Initialisation;
    GlobalRoute;
    LocalRoute;
    PutLayout;
  end;
```

This procedure is divided into four parts, i.e. initialisation, global routing, local routing and the generation of a layout-file.
Each part is described in the next paragraphs.

procedure Initialisation;
Before starting with the routing the IO-pins have to be defined. The user is allowed to fix the y-coordinates of an IO-pin at the left and right-side of the chip. Also the x-coordinates at the top- and bottomside can be fixed. The remaining coordinates of the IO-pins are known after the last step of the routing phase, the layout generation. short description of the procedures."

procedure Global Route;
The global router is sub-divided in a number of procedures which will be described shortly.

procedure initglobal:

This procedure reads the cells with their position and size from the database, end composes for each column a list of cells in that column. Maxcolheight is else computed in this procedure.

procedure distance:
This procedure computes the cost of an edge in the full graph.

procedure add_trace:
This procedure keeps track of the route of a wire. This is necessary, because if an edge becomes a member of the minimum cost spanning tree, we have to know how the wire associated with that edge has to be routed. Make_vertical_channel_lists." This procedure makes the top, bottom, left and right lists for each channel. The procedure gets its input from a list of wires to be routed (rout_list).

procedure add_list:
This procedure adds a pin to be routed to the top, bottom, left or right list of a channel. This procedure is used by make_vertical_channel_lists.

procedure make_tree:
This procedure generates for each signal the full graph and sorts the edges with respect to their cost.

procedure route_signals:
This is the main procedure. The procedure constructs the minimum cost spanning tree, and fills a list of wires to be routed (rout_list). The procedure make_vertical_channel_lists is called and the feedthroughs are processed (these are stored in feed_list).

```
procedure Local Route;
  begin
    InitLocalRoute;
    channel := channelroot;
    for number := leftchannel to rightchannel do
      begin
        MakeSignalListChannel;
        if channel^.signalroot <> nil then
          begin
            InitialiseGrid;
            InitChannelLength;
            CalculateChannelDensity;
            InitChannelWidth;
            ChannelRoute;
          end;
      end;
    MakeNetListBottomChannel;
    MakeSignalListChannel;
    if channel^.signalroot <> nil then
      begin
        InitialiseGrid;
        InitChannelLength;
        CalculateChannelDensity;
        InitChannelWidth;
        ChannelRoute;
      end;
    channel := channel^.next;
    MakeNetListTopChannel;
    MakeSignalListChannel;
    if channel^.signalroot <> nil then
      begin
        InitialiseGrid;
        InitChannelLength;
        CalculateChannelDensity;
        InitChannelWidth;
        ChannelRoute;
      end;
  end;
```

The first step is the initialisation of the local routing. The user is asked for the parameters steady_net_constant and min_jog_length which are necessary for the greedy router. Because the greedy router gives different results with different startconditions, the user has to decide how many times a channel has to be routed. The best result of these attempts is stored in the datastructure.
The vertical channels, i.e. the leftchannel up to and including the right-channel, are routed first. Here after the netlists for the top- and bottomchannel can be generated, so that these channels can be routed also.

For each channel to be routed some initialisations are necessary. A signallist is built that tells on which side and on which position a signal appears. Also the greedy router wants to know the length and width of the channel to start with.

```
procedure ChannelRoute;
  begin
    repeat
      InitChannel;
      for column := 1 to channellength do
        begin
          BringInNet;
          CollapseSplitNets;
          LowerRaiseUnsplitNets;
          WidenChannel;
          ExtendToNextColumn;
        end;
      repeat
        Check for extra_column;
        if extra_column then
        begin
          CollapseSplitNets;
          LowerRaiseUnsplitNets;
          WidenChannel;
          ExtendToNextColumn;
        end;
      until not(extra_column)
    until channelready;
  end;
```

The first column of the channel has to be initialised by using the results of the global router. the global router gives for each channel the netlists. A netlist is available for the left-, right-, top- and bottomside of the channel. The netlist of the leftside is used to initialise the first column.
After the initialisation the channel is routed in a left-to-right, column-by-column manner. In each column greedy heuristics are used as described in chapter 7.4.
The heuristics which are successively performed are:

- bring in a net from the top- and/or the bottomside.
- collapse split nets.
- lower or raise unsplit nets.
- widen channel if necessary.
- extend to next column.

If the last column is finished it is possible that not all connections are established. It may therefore necessary to make some connections in one or more extra columns. In these columns the same heuristics are applied except the first one because

there is no pin at the top- or bottomside available.

```
procedure PutLayout;
  begin
    Open Layout-file;
    Generate feed-through cell;
    Calculate position for first channel;
    if leftchannel <> empty then put channel rotated over 90
                                 degrees on the calculated
                                 position
    for number := (leftchannel+1) to rightchannel do
      begin
        Put cells in column;
        Calculate position for channel;
        Put channel rotated over 90 degrees on the
          calculated position;
        Make vertical connection between top of column
          and the v_dd-line;
        Make vertical connection between bottom of column
          and the v_ss-line;
      end;
    Make the v_dd-line;
    Make the v_ss-line;
    Calculate position for bottomchannel;
    Put BottomChannel;
    Calculate position for topchannel;
    Put TopChannel;
    Make ps-connections between bottom-side of topchannel and
       right-sides of the vertical channels;
    Make ps-connections between top-side of the bottomchannel
      and left-sides of the vertical channels;
    Close Layout-file;
  end;
```

The last routing part is the generation of a layout. The datastructure for each channel is known and also the positions of the cells within the columns. Now it is possible to generate the layouts for the channels and putting them together with the layout of the cells into a layout-file. The layouts of the cells are already generated by the cellgenerator.
Also attention has to be given to the power-lines, v_{dd} and v_{ss}, which enables each cell to perform each logic function.
First the layout of the vertical channels is generated and the layouts are positioned together with the cells, at their final coordinates. After the construction of a column the v_{dd}-line is extended upwards and the v_{ss}-line is extended downwards. The upward-extension of the v_{dd}-line is not the same for each column because not every column has the same height. The downward extension of the v_{ss}-line is everywhere the same because the lowermost cells in each column have the same vertical

coordinates.
After the layout-generation of the last vertical channel, the total width of the chip is known. The horizontal v_{dd} and v_{ss}-lines can now be constructed from left to right. Here after it is possible to generate the layout of the top- and bottomchannel and the positions for these channels are calculated.
Because of the v_{dd}- and v_{ss}-line "ps-bridges" have to be built in order to realise connections between the topside of the bottomchannel and the leftsides of the vertical channels or between the bottomside of the topchannel and the rightsides of the vertical channels.
At this moment the layout is completed. The result of the automatic layout- generation can be seen by using a layout-editor.

It is possible that one or more channels are empty. For example the topchannel can be empty when there is no IO-pin at the topside and when no connections are established between the vertical channels by using the topchannel. Whenever a channel is empty it will not be placed in the layout.

8. SYSTEM FEATURES

The macro-cell generator offers some features which give the designer significant influence on the aspect-ratio of the layout and on the positions of the IO-pins. The user can also control the greedy router by giving two parameters. These parameters will have a direct influence on the total used area.
The designer is able to fix the number of columns. With this feature he is able to steer the aspect-ratio of the layout. It is possible to get from the same set of boolean expressions, a square layout or a more rectangular layout. Of course a square layout is the most preferable shape when the total layout-area has to be minimized. The disadvantage of a rectangular layout is that the mean distances for the connections between the cells are longer. In a standard cell technology this will result in an increasing number of feed-throughs and in channels which are requiring more tracks. These disadvantages are resulting in an increased layout area compared to a square layout.
The second feature is that the user is able to fix the IO-pins at one of the four sides of the chip. At the top- or bottomside the x-coordinates can be fixed. Also at the left- or rightside the y-coordinates can be fixed. The y- and x-coordinates of an IO-pin at resp. the top-/bottomside and the left-/rightside are known after the layout is completely finished.
However an IO-pin can not be placed everywhere. The exact allowed sides are:

1 - the topside of the topchannel
2 - the bottomside of the bottomchannel
3 - the topside of the leftchannel
4 - the bottomside of the rightchannel

The sides 1 and 2 are belonging each to one channel, so no problems have to be expected at this sides. But the left- and rightside of the chip are distributed over three channels. The leftside, for example, is formed by:

- the leftside of the topchannel
- the topside of the leftchannel
- the leftside of the bottomchannel

It is not possible to put at the same time IO-pins in these three parts of the left-side. The user wants to fix the global y-coordinates, but the width of the top- and bottomchannel are not known at the moment of IO-pin positioning. To give the user the greatest opportunity for the positioning of the IO-pins, the topside of the leftside is choosen and the two other parts of the leftside are forbidden.
An analog reasoning can be given for the rightside of the chip.

The aspect-ratio control and the fixation of the IO-pins are examples of global control. Also some local control is possible. The greedy router needs two parameters, i.e. steady_net_constant and min_jog_length. Changing these parameters has influence on the number of vias used. When these parameters are increased less vias will be used but the width of a channel will grow. So these parameters have a direct influence on the total area used for the chip. The user has to find the correct values for the parameters so that an acceptable balance is found between the total used area and the total number of vias used.
Beside these two parameters the greedy router needs also an initial value for the width of the channel. Different results are obtained with different parameters and different initial widths. Before starting the routing of a channel it is not known which values the parameters and the initial width must have in order to obtain optimal channels.
The parameters steady_net_constant and min_jog_length are fixed by the user. The user has to decide how many times a channel is routed (suppose n times). The minimum width (suppose ω) of a channel that can be reached, can be calculated. The greedy router is started n times. The first time the initial width ω-1/2n is used and the last time ω+1/2n. The best result, i.e. the result having a minimum number of tracks and vias is stored in the datastructure.
By increasing the value n the user can obtain a better routing result. The disadvantage is that the computation time for each channel is increased.

9. CONCLUSIONS

The macro-cell generator for combinatorial logic is now in operation. The whole path, starting with a set of boolean expressions and finishing with a complete layout, can be passed through.
However the macro-cell generator can be perfectionated. Several things can be done to reduce the design-time and to minimize the total layout area.
The most important issue is to improve the placement program. Another algorithm must be used to calculate the two heighest eigenvalues of the Schoenberg matrix. This will considerably reduce the design-time because the placement program that is currently used calculates all eigenvalues.
Also the placement-algorithm must take account of the positions of the IO-pins. In the present system the user has to fix the IO-positions. So the user decides where the IO-pins are positioned. When this approach is used the results of the placement-algorithm must depend on the positions of the IO-pins.
Another approach is to give the user less freedom and more freedom to the system in positioning the IO-pins. When the system is able to position the IO-pins a further reduction of the total area can be accomplished.
Another issue is further reduction of the accomplished layout. When some attention is paid to the several channels in the layout then it is evident that these channels can be compacted. The main cause for this compaction possibility is the dimension of the routing grid. the dimension of the grid is choosen in such a manner that the greedy router has complete freedom for introducing vias. But the introduction of vias next to each other is occurring not very often. Compaction can, in most cases, reduce the width of a channel considerably.
The most important conclusion is that the macro-cell generator should only be used for the generation of combinatorial logic that has a lot of input- and outputvariables and a few productterms. If combinatorial logic doesnot have this characteristic then it is better to use another realisationform, for example a PLA approach.

10. REFERENCES

[1] A.J. Smits,
"Manipulation on Boolean Functions",
Thesis Report, Eindhoven University of Technology,
Eindhoven, october 1984.

[2] P.T.H.M. Van Paassen,
"Optimisation of Combinatorial Logic",
Thesis Report, Eindhoven University of Technology,
Eindhoven, february 1985.

[3] R.H.J.M. Otten,
"Automatic Floorplan Design",
Proceedings 19th Automatic Design Conference,
Las Vegas, june 1982, pp. 261-267.

[4] R.H.J.M. Otten,
"Eigensolutions in Top-Down Layout-Design",
15th International Symposium on Circuits and Systems,
Rome, may 1982, pp. 1017-1020.

[5] L.P.P.P. Van Ginneken, R.H.J.M. Otten,
"Global Wiring for Custom Layout Design",
18th International Symposium on Circuits and Systems,
Kyoto, june 1985, pp. 207-208.

[6] A.V. Aho, J.E. Hopcroft, J.D. Ullmann,
"Data Structures and Algorithms",
Addison-Wesley, june 1983.

[7] R.L. Rivest, C.M. Fiduccia,
"A 'Greedy' Channel Router",
Proceedings 19th Design Automation Conference,
Las Vegas, june 1982, pp. 418-424.

[8] T.G. Szymanski,
"Dogleg Channel Routing is NP-Complete",
IEEE Transactions on Computer-Aided Design,
Vol. CAD-4, no. 1, january 1985, pp. 31-41.

[9] T. Yoshimura, E.S. Kuh,
"Efficient Algorithms for Channel Routing"
IEEE Transactions on Computer-Aided Design
of Integrated Circuits and Systems,
Vol. CAD-1, no.1, january 1982, pp. 25-35.

[10] J. Soukup,
"Circuit Layout",
Proceedings of the IEEE,
Vol. 69, no.10, october 1981, pp. 1281-1304.

[11] D.N. Deutsch,
"A 'Dogleg' Channel Router",
Proceedings 13th Design Automation Conference,
San-Fransisco, june 1976, pp. 425-433.

[12] G. Persky, D.N. Deutsch, D.G. Schweikert,
"LTX - A System for the Directed Automated
Design of LSI Circuits",
Proceedings 13th Design Automation Conference,
San Fransisco, june 1976, pp. 399-407.

[13] "Cooperative Development of an Integrated, Hierarchical and Multiview VLSI-Design System with Distributed Management on Workstations",
A proposal submitted to the Comission of the European Communities, Information Technologies Task Force "ESPRIT", 1985.

[14] A.T.H.A.J. De Groot,
"Cellgeneration for Combinatorial Logic using NMOS Standard-Cell Technology",
Training Report, Eindhoven University of Technology, Eindhoven, january 1986.

[15] C.H.H. Janssen,
"Database for Macro-Cell Generation",
Training Report, Eindhoven University of Technology, Eindhoven, july 1985.

11. EXAMPLE

INPUT LOGIC EDITOR:

CONTROLLER FOR A 16 BIT BUS

```
F1 = BCDEHI'JKL'+BCDFHI'JKL'+DEHIJKL'O+DFHIJKL'O+D'EHIJKL'MN+D'FHIJKL'MN+
     ABCDE+ABCDF+AD'G+AE'F'G+D'GI'JKL'+E'F'GI'JKL'+GH'IKL'+GH'JKL'+
     GH'I'J'K'+H'I'JKLP'+DE'F'HIJKL'M
F2 = D'EH'I'JKLMNP'+D'FH'I'JKLMNP'+DEH'I'JKLOP'+DFH'I'JKLOP'+DEHI'JKL'O+
     DFHI'JKL'O+AD'EMN+AD'FMN+ADEO+ADFO+D'I'JKL'Q+E'F'I'JKL'Q+
     DE'F'H'I'JKLMP'+H'I'JKQ+H'IKL'Q+ADE'F'M
F3 = AB'P'R+AC'P'R+B'HJKL'P'R+C'HJKL'P'R+AD'R+AE'F'R+D'HJKL'R+E'F'HJKL'R+
     H'I'J'K'R+H'I'JKR+H'IKL'R+APS+HJKL'PS
F4 = AB'P'T+AC'P'T+B'HJKL'P'T+C'HJKL'P'T+AD'T+AE'F'T+D'HJKL'T+E'F'HJKL'T+
     H'I'J'K'T+H'I'JKT+H'IKL'T+APU+HJKL'PU
F5 = AB'P'V+AC'P'V+B'HJKL'P'V+C'HJKL'P'V+AD'V+AE'F'V+D'HJKL'V+E'F'HJKL'V+
     H'I'J'K'V+H'I'JKV+H'IKL'V+APW+HJKL'PW
F6 = AB'P'X+AC'P'X+B'HJKL'P'X+C'HJKL'P'X+AD'X+AE'F'X+D'HJKL'X+E'F'HJKL'X+
     H'I'J'K'X+H'I'JKX+H'IKL'X+APY+HJKL'PY
F7 = AB'P'Z+AC'P'Z+B'HJKL'P'Z+C'HJKL'P'Z+AD'Z+AE'F'Z+D'HJKL'Z+E'F'HJKL'Z+
     H'I'J'K'Z+H'I'JKZ+H'IKL'Z+APa+HJKL'Pa
F8 = AB'P'b+AC'P'b+B'HJKL'P'b+C'HJKL'P'b+AD'b+AE'F'b+D'HJKL'b+E'F'HJKL'b+
     H'I'J'K'b+H'I'JKb+H'IKL'b+APc+HJKL'Pc
F9 = AP'x+HJKL'P'x+APd+HJKL'Pd+H'I'J'K'x+H'I'JKx+H'IKL'x
F10= AP'e+HJKL'P'e+APy+HJKL'Py+H'I'J'K'e+H'I'JKe+H'IKL'e
F11= AP'g+HJKL'P'g+APf+HJKL'Pf+H'I'J'K'g+H'I'JKg+H'IKL'g
F12= AP'i+HJKL'P'i+APh+HJKL'Ph+H'I'J'K'i+H'I'JKi+H'IKL'i
F13= AP'k+HJKL'P'k+APj+HJKL'Pj+H'I'J'K'k+H'I'JKk+H'IKL'k
F14= AP'm+HJKL'P'm+APl+HJKL'Pl+H'I'J'K'm+H'I'JKm+H'IKL'm
F15= AP'o+HJKL'P'o+APn+HJKL'Pn+H'I'J'K'o+H'I'JKo+H'IKL'o
F16= AP'q+HJKL'P'q+APp+HJKL'Pp+H'I'J'K'q+H'I'JKq+H'IKL'q
F17= AP'w+HJKL'P'w+APv+HJKL'Pv+H'I'J'K'w+H'I'JKw+H'IKL'w
```

OUTPUT LOGIC EDITOR:

```
F 1 = HIL'#J#M+A#D'#F'+#F'#V+AG#D+G#W+GIL'#N+GH'L'#M+GJ'K'#P+#U
F 2 = A#J+#J#U+O#V+Q#B#N+Q#W
F 3 = R#K+S#L
F 4 = T#K+U#L
F 5 = V#K+W#L
F 6 = X#K+Y#L
F 7 = Z#K+a#L
F 8 = b#K+c#L
F 9 = x#H+d#L
F10 = e#H+y#L
F11 = g#H+f#L
F12 = i#H+h#L
F13 = k#H+j#L
F14 = m#H+l#L
F15 = o#H+n#L
F16 = q#H+p#L
F17 = w#H+v#L
```

The new literals are:

```
#A = A+HL'#M
#B = I'J+IL'
#C = E+F
#D = D'+#C'
#E = K#B+I'J'K'
#F = B'+C'
#H = H'#E+P'#A
#J = DM#C'+D'MN#C+O#D'
#K = P'#A#F+#A#C'+D'#A+H'#E
#L = P#A
#M = JK
#N = H'K
#P = H'I'
#S = I'L'#M
#T = #M#P
#U = LP'#T
#V = H#D'#S
#W = #D#S
```

OUTPUT NETDECOMPOSITION

```
     1     (J+K)'                    43    (A+H.78.61)'        74    (D)'
     2     (72+A.48+63.1)'      #A :44    (43)'               75    (I)'
     3     (I.62)'              #B :45    ((75+L).(I+76))'    76    (J)'
     4     (2.(H+85).(78+3))'        46    (F+E)'              77    (K)'
     5     (70+A.82)'           #C :47    (46)'               78    (L)'
     6     (5)'                 #D :48    (D.47)'             79    (O)'
     7     (68+84.6+G.4)'            49    (K.45)'             80    (P)'
F1 : 8     (7.(L+85+73))'       #E :50    (49.(I+J+K))'       81    (44)'
     9     (72+45.62)'          #F :51    (B.C)'              82    (48)'
    10     (68+A)'              #H :52    ((P+81).(H+83))'    83    (50)'
    11     (O.70+55.87+Q.86)         53    (N.47)'             84    (51)'
F2 :12     (11)'                     54    ((74+53).(74+47))' 85    (61)'
    13     (S.59+R.57)'         #J :55    ((79+48).(M+54))'   86    (9)'
F3 :14     (13)'                     56    ((P+84).47.D)'      87    (10)'
    15     (U.59+T.57)'         #K :57    ((H+83).(44+56))'
F4 :16     (15)'                     58    (P.44)'
    17     (W.59+V.57)'         #L :59    (58)'
F5 :18     (17)'                     60    (J.K)'
    19     (Y.59+X.57)'         #M :61    (60)'
F6 :20     (19)'                #N :62    (H+77)'
    21     (a.59+Z.57)'         #P :63    (H+I)'
F7 :22     (21)'                #S :64    (I+L+85)'
    23     (c.59+b.57)'              65    (61.63)'
F8 :24     (23)'                #T :66    (65)'
    25     (d.59+x.52)'              67    (L.66)'
F9 :26     (25)'                #U :68    (80+67)'
    27     (y.59+e.52)'              69    (H.64)'
F10:28     (27)'                #V :70    (82+69)'
    29     (f.59+g.52)'              71    (48.64)'
F11:30     (29)'                #W :72    (71)'
    31     (h.59+i.52)'              73    (H.I.55)'
F12:32     (31)'
    33     (j.59+k.52)'
F13:34     (33)'
    35     (1.59+m.52)'
F14:36     (35)'
    37     (n.59+o.52)'
F15:38     (37)'
    39     (p.59+q.52)'
F16:40     (39)'
    41     (v.59+w.52)'
F17:42     (41)'
```

OUTPUT MACRO-CELL GENERATOR

48 PRIMARY INPUTS
17 PRIMARY OUTPUTS
6 COLUMNS
ASPECT-RATIO : 2358μ x 2352μ