MASTER

Floorplan optimization by means of module orientation, channel width estimation and pin position calculation

Brans, A.G.C.T.

Award date:
1989

EINDHOVEN UNIVERSITY OF TECHNOLOGY
DEPARTMENT OF ELECTRICAL ENGINEERING
DESIGN AUTOMATION GROUP

FLOORPLAN OPTIMIZATION BY MEANS OF MODULE ORIENTATION, CHANNEL WIDTH ESTIMATION AND PIN POSITION CALCULATION

A.G.C.T. Brans

Master thesis
reporting on graduation work
performed from Februari 1988 to Februari 1989
by order of prof. dr. ing. J.A.G. Jess
and supervised by ir. L.P.P.P. van Ginniken.

Summary.

In Integrated Circuit design, a number of stages can be distinguished. After dividing the initial description into several functional blocks, the layouts for these blocks are generated using cell generators or cell libraries. In a floorplanning stage, the blocks are positioned within the entire layout area.

In the floorplanning programs developed so far, optimal floorplans are found by grouping heavily interconnected modules close together. Others are positioned at greater distance. An even more optimal floorplan can be found if these modules are mirrored in one or two directions. This way, wiring space required to connect all modules, can be minimized. A way to find optimal module orientations has been developed for slicing type floorplans. However, the method used is floorplan type independent.

After optimizing the module orientations, still more size improvement can be established if channel widths are known approximately. During floorplan adjustment stages, this information can be used to recalculate optimal module sizes. If piece-wise linear shape constraints are available, more optimal module dimensions can often be found without changing the module positions. A program to find these approximate channel widths has been written and tested.

A final point of interest is the pin position calculation. If module descriptions allow variable pin positions, optimal positions must be found to minimize the overall wire length. This minimization can lead to overall area improvement. Programs to find these optimal pin positions have been written.

Tests of the programs developed, showed area improvements up to 30 per cent. The average improvement of the best method was about 10 to 12 per cent.

Contents.

Introduction.

When an Integrated Circuit is being developed, a number of steps are performed. First of all, a behaviour description of the circuit must be drawn up, using one of several possible description languages. This description, or algorithm, is translated into network and controller descriptions using Petri-nets and Data-Flow-Graphs. The latter descriptions are optimized by means of state encoding, logic simplification, decomposition and technology mapping.

The overall circuit will now be divided into several separate functional blocks. The next step will be the generation of cells performing the functions represented by these blocks. The cells produced can either be fixed cells available in a library or custom made cells generated in cell generators. These latter cells can have fixed dimensions, but a number of cell generators allows several different aspect ratios. They produce piece-wise linear shape constraints as described in [1]. A possible cell generator, able to produce piece-wise linear shape constraints is described in [2]. Several cell generators produce cells having fixed pin positions, others allow the user to specify the pin positions. Sometimes a certain interval is given along the cells edge within which the pin position must be specified. A possible cell description from a cell generator having both piece-wise linear shape constraints and variable pin positions is given in appendix A. In the rest of this report, these cells will be refered to as modules.

After producing the modules, they must be placed in a floorplan. To perform this floorplanning step, several algorithms have been developed. A number of them are described in [1], [3] and [4]. At the Eindhoven University of Technology (TUE), a slicing algorithm is used to perform the floorplanning stage. The method used is described in [5].

The floorplan obtained after the slicing, is calculated with a number of uncertainties in mind. No information on channel widths was available at the beginning of the floorplanning stage. The floorplan found, is therefore calculated assuming zero-width channels. According to these channel widths, the optimal module dimensions have been derived from the shape constraints.

A number of optimizations can be carried out starting from this type of floorplan. First of all, changing the modules orientation might lead to more optimal floorplans. Each module may be mirrored in one or two directions without conflicting with it's reserved area. Beside mirroring the modules, optimal pin position calculation may lead to improvements of the floorplan. A final improvement can be made by making an estimation of the channel widths. A module resizing step can be performed to alter the module sizes according to the new channel widths.

1. *Some initial considerations.*

Before any steps can be taken, some initial problems must be looked at. One has to know what an optimal floorplan is. Also, how can the problem of optimizing the floorplan be divided into separate blocks. In this chapter some considerations regarding these problems are elaborated more deeply.

1.1 What is an optimal floorplan ?

Before being able to develop algorithms to orient modules optimally, one has to consider what optimal floorplanning really means. In the initial floorplanning stages, the modules are positioned in places in the floorplan according to their mutual interconnection. "Long-wire-penalties" or net-weights may have been taken into consideration. This type of floorplanning normally results in short interconnections and probably small layouts. It is most likely, that even more optimal floorplans can be found. Module mirroring and pin position calculation can lead to this. The question that remains is: " What is an optimal floorplan ? ". Three objectives are possible:

1) An optimal floorplan is one in which the overall speed of the circuit is the highest.
2) A minimal overall layout size implies an optimal floorplan.
3) Minimization of the overall wire length leads to finding an optimal floorplan.

It must be stated here that the objectives mentioned above do not imply or exclude each other. To obtain a maximum circuit speed, the nets associated with the critical path must be the shortest. Other less critical connections may be longer at cost of the critical one. On the other hand, minimal overall wire length does not imply minimal overall layout size. An example of this is given in figure 1.

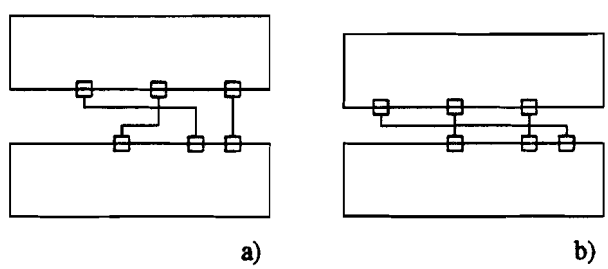


Figure 1. a) Module orientation with minimal overall wire length, b) module orientation with minimal overall layout size.

It will be clear that one of the objectives above must be chosen to serve as an objective function. The first objective will be very difficult to achieve. In the initial information, extra data must be stored to indicate nets belonging to the critical path. At this moment such data is not available. The second and third

objective on the other hand will not conflict with each other most of the time. Although examples can be given where they do conflict (figure 1), often this will not be the case. The main goal of optimization is to achieve fast and small floorplans. Checking the overall layout size will involve a great amount of computation time. It will be almost impossible to achieve this goal for extensive floorplans in a reasonable time. Small floorplans can often be found by finding minimal overall wire length.

For the reasons mentioned above, it seams reasonable to try to achieve the third objective. Minimizing the overall wire length is thus a derived objective resulting from the second objective. The algorithms described in the following chapters will therefore be based on this objective.

1.2 Problem decomposition.

It will be clear that the entire problem is to big to solve at once. It must be divided into several separate blocks. These blocks can then be solved one by one. Although the possible optimizations like module orientation and pin position calculation have mutual influences, it must be possible to find a sequence of handling them.

First, we must find the separate subproblems. They have been mentioned in the Introduction: channel width estimation, module orientation and pin position calculation. A final block representing the module resizing must be added at the end.

When the subproblems are found they must be placed in a certain order. It will be clear that the channel width estimation step must be done before the resizing step. At this stage, all module orientations and pin positions must be known. Looking at the two subproblems left, it appears logical to solve the module orientation problem first. Optimal pin positions can only be found after finding the module orientations. Module orientation algorithms may assume optimal pin positions, along the pin position interval, for each orientation.

Based on the sequence described above, the subproblem solving sequence is given in figure 2.

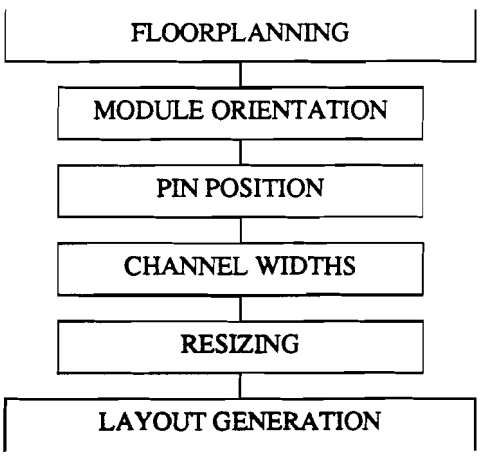


Figure 2. Subproblem solving sequence.

The first and the last block in figure 2 are added to indicate steps performed before and after floorplan optimization. These steps will not be described in this report. The steps in between, will be described in the same order.

2. *Module orientation.*

As shown in the previous chapter, the first problem to cope with is the module orientation. In this chapter, a method to find suboptimal orientations is described. First, a description of the problem is given. Then a data structure is described in detail, to store the information necessary for a (sub) optimal solution. Some calculation and possible solution methods are given next. A description of the program implementation will be given at the end of the chapter.

2.1 The module orientation problem.

The module orientation problem can be described as a translation from one state-space to another. The first state-space contains all possible orientation combinations. The second is determined by the goal we try to achieve. In chapter 1, a number of possible goals have been discussed. Minimal floorplan dimensions was a possibility. Minimal overall wire length could be another. It was shown, that the objective function to be satisfied is minimal overall wire length. The objective function is therefore the translation which is graphically represented in figure 3.

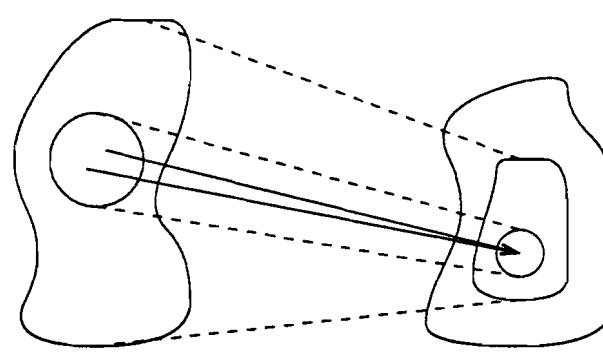


Figure 3. Graphic representation of the module orientation problem.

For every possible module orientation combination, an overall wire length can be found. The destination state-space contains all positive numbers. The numbers reached by the transformation are a subset of the destination state-space. Every subset of the destination state-space contains at least one optimal state. Since the goal we try to achieve is to minimize the overall wire length, the optimal state is the one with the smallest number. At least one orientation combination is translated to this smallest number. Although, in the destination space it has been represented by a circle, the optimal destination set will only contain one state.

From figure 3 we can see that the back-transformation is non-unique. Two different orientation combinations may both have the same optimal destination state. Although this looks problematic, finding one of the original orientation combinations is sufficient. All orientation combinations resulting in the optimal destination state will satisfy the criterion set: minimal overall wire length.

The total number of orientation combinations depends on the number of orientations allowed for each module. In the floorplan to be optimized, a certain space is reserved for each module. A limited length and width is available. Mirroring the module in x- or y-direction will never violate this length or width. Rotating a module over 90 or 270 degrees however, may violate them. A low, wide module being rotated over 90 degrees will result in a high, narrow module. Rotation over 180 degrees is allowed since it is equal to mirroring in two directions. This knowledge shows that each module can take four different orientations. The number of states is therefore 4^N, where N is the number of modules in the floorplan.

Instead of handling both mirroring directions at the same time, one might decide to handle both separately. This way the number of states is reduced to 2^N while the final solution is still near optimal. Handling the mirroring directions separately will reduce the computing time with a factor $(4^N / (2 * 2^N)) = 2^{N-1}$.

From the problem description given above, it will be clear that the entire problem is NP-complete.

2.2 The connection matrix.

As described, the first operation to be carried out, is the transformation from the module orientation-space to the wire length-space. To store the wire lengths belonging to each connection in each orientation combination, a matrix can be constructed. Both rows and columns represent modules and the matrix will look like:

$$L = \begin{bmatrix} L_{aa} & L_{ab} & L_{ac} & L_{ad} & L_{ae} \\ L_{ba} & L_{bb} & L_{bc} & L_{bd} & L_{be} \\ L_{ca} & L_{cb} & L_{cc} & L_{cd} & L_{ce} \\ L_{da} & L_{db} & L_{dc} & L_{dd} & L_{de} \\ L_{ea} & L_{eb} & L_{ec} & L_{ed} & L_{ee} \end{bmatrix} \tag{1}$$

Every element from this matrix must contain information on the connections between the two corresponding modules. Therefore, the elements L_{bc} and L_{cb} contain information on all connections between module B and module C. The elements must also contain information showing the influence of turning either module B or module C or both. It may be clear that this information can't be put in only two elements (L_{bc} and L_{cb}) and therefore the elements from the matrix shown in equation (1) are really submatrices containing several separate elements themselves.

The size of the submatrices is determined by the number of possible orientations of each module. In the final implementation, both mirroring directions are handled separately. Therefore the number of possible orientations is limited to two. The number of elements in each submatrix is equal to four. The elements of the submatrix L_{xy} will be:

$$L_{xy} = \begin{bmatrix} l_{xy} & l_{x\bar{y}} \\ l_{\bar{x}y} & l_{\bar{x}\bar{y}} \end{bmatrix} \tag{2}$$

All elements of the submatrix must contain information on the connections from module X to module Y in the orientation given. So, $l_{x\bar{y}}$ contains information on the connections between module X in its original orientation and the turned module Y. Further details on these submatrix elements will be discussed in paragraph 2.3..

Looking at the matrix two things attract attention. First, the nondiagonal elements in equation (1) are connected in pairs. It will be clear that elements of the submatrix L_{ab} are equal to elements of the submatrix L_{ba}. These elements will not be in the same position in both submatrices because of the exchange of row and column numbers. Or mathematically: $L_{ab}=(L_{ba})^T$. However, almost half of the matrix does not have to be allocated in program implementation. This results in considerable memory savings when optimizing large floorplans.

A second thing that attracts attention is the diagonal elements of the matrix in equation (1). The submatrices belonging to these diagonal elements are given in equation (3). From this configuration two things can be said. First, the nondiagonal elements of this submatrix $l_{\bar{x}x}$ and $l_{x\bar{x}}$; these elements should describe connections from module X in the original orientation to the same module in mirrored orientation (or the other way around). Since a module can never be in two different orientations at the same time, these elements will be equal to zero. Secondly, the diagonal submatrix elements l_{xx} and $l_{\bar{x}\bar{x}}$; these elements describe connections from the current module to itself in respectively the original or the mirrored orientation. It will be clear that such connections will always have the same length irrespective of the orientation. Therefore, these elements can be made zero also or given their initial value. The elements can not be left out however since they may be of some use later on (see paragraph 2.4.1).

$$L_{xx} = \begin{bmatrix} l_{xx} & l_{x\bar{x}} \\ l_{\bar{x}x} & l_{\bar{x}\bar{x}} \end{bmatrix} = \begin{bmatrix} l_{xx} & 0 \\ 0 & l_{\bar{x}\bar{x}} \end{bmatrix} \tag{3}$$

Our objective was to find a minimal overall wire length. From the matrix L in equation (1) we are able to derive the information needed. The objective function ξ is the addition of all matrix elements belonging to the orientation combination examined. Mathematically this will look like:

$$\xi = \sum_i \sum_j l_{O(i)O(j)} \tag{4}$$

where $O(i)$ resp. $O(j)$ represent the orientation of module i resp. j in the orientation combination examined. So, $O(i)$ is either i or $\bar{i}$ and $O(j)$ either j or $\bar{j}$.

2.3 Several element calculation methods.

After describing the connection matrix, let us take a closer look at the exact calculation of the submatrix elements. These elements must contain information on the wire length of all connections between the corresponding modules. This can be described as:

$$l_{ij} = \sum_c C_{ij}(c,X,Y) * F(c,X,Y) \tag{5}$$

in which $C_{ij}(c,X,Y)$ is the connection length of the connection c running from module X to module Y. This connection length depends on the orientation of the modules X resp. Y. Index i must therefore be x or $\bar{x}$ and index j must be either y or $\bar{y}$. $F(c,X,Y)$ is a scaling factor used to give each connection an equal influence

in the total wire length. It may be used when a connection connects three or more modules. If it is not used, its value must be a non-zero constant.

Both multiplication terms can depend on several factors. A number of these factors will be discussed below.

2.3.1 Length.

Most important of all factors will be the connection length. It is more or less represented by the first multiplication term. It will be clear that this connection length must be the shortest connection possible between begin- and end-point of the connection. An optimal situation can only be reached if the true wire length, found after using a global router, would be used. This however will mostly be impossible, so other values must be found to approximate this true wire length. Before discussing these approximations, a description must be given of the pins connected by them. In the Introduction, a cell generator was mentioned being able to produce variable pin positions. A module description of a variable pin position cell is given in Appendix A.

Pins described by an interval may be located at any position on the interval. The interval can be seen as a piece-wise linear description. The description is determined by the modules corners within the interval and the begin- resp. end-point of the interval. To determine the minimal distance possible between two piece-wise linear descriptions, it is sufficient to determine the minimal distance between all combinations of point-pairs. The point-pairs consist of an interval-point of each pin. Interval-points are either corners or begin- or end-points. A calculation error can be made using this method if part of the intervals run parallel to each other.

A few approximations of the true wire length are shown in figure 4. In this figure it is assumed that all connections interconnect two pins. Three or more pin interconnections can be dealt with by handling each pin-pair separately. Adjustments to be made when using this method are discussed in section 2.3.2.

Described in words these approximations are:

1) X-distance (D_x) or Y-distance (D_y): A minimal distance in x- resp. y-direction is used to be a measure of the minimal true wire length. Mathematically:

$$D_x(c,X,Y) = \min_{k,l} (| x_{ak} - x_{bl} |) \qquad \forall_{1 \le k \le \#_a} \forall_{1 \le l \le \#_b} \tag{6}$$

where x_{ak} stands for the x-coordinate of the interval-point k belonging to pin a. $\#_a$ resp. $\#_b$ indicate the number of interval-points on the interval describing the pins a resp. b. A similar description stands for $D_y(c,X,Y)$:

$$D_y(c,X,Y) = \min_{k,l} (| y_{ak} - y_{bl} |) \qquad \forall_{1 \le k \le \#_a} \forall_{1 \le l \le \#_b} \tag{7}$$

2) Euclidian distance (D_e): The minimal eulidian distance between the piece-wise linear intervals. A mathematical description is:

$$D_e(c,X,Y) = \min_{k,l} (\sqrt{(x_{ak} - x_{bl})^2 + (y_{ak} - y_{bl})^2}) \qquad \forall_{1 \le k \le \#_a} \forall_{1 \le l \le \#_b} \tag{8}$$

The euclidian distance is not necessarily the square root of ($D_x^2 + D_y^2$), as is shown in figure 4.

3) Manhattan distance (D_m): The minimal Manhattan distance between two piece-wise linear intervals.

$$D_m(c,X,Y) = \min_{k,l} (| x_{ak} - x_{bl} | + | y_{ak} - y_{bl} |) \qquad \forall_{1 \le k \le \#_a} \forall_{1 \le l \le \#_b} \tag{9}$$

The Manhattan distance is not necessarily equal to ($D_x + D_y$). This is illustrated in the example in figure 4.

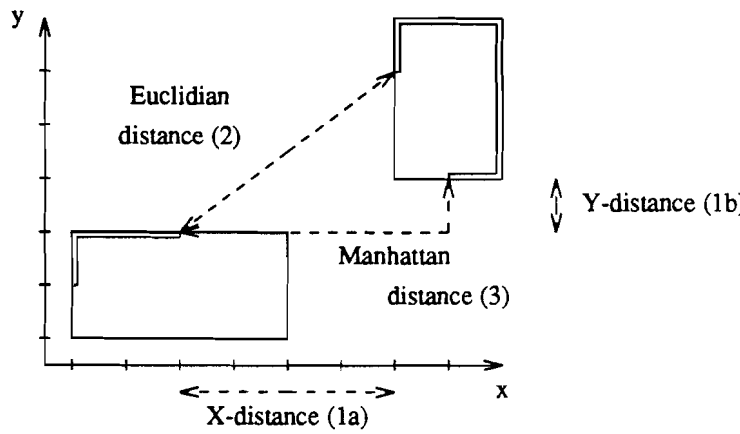


Figure 4. Distances. The calculated minimal values are: 1a) 4 units; 1b) 1 unit; 2) 5 units; 3) 6 units.

In equation (5), the first multiplication term, $C_{ij}(c,X,Y)$, can be replaced by one of the distances described above. From figure 4, it can be seen that this term depends on the orientation of the two modules involved indicated by the subscripts i and j. So:

$$C_{ij}(c,X,Y) = D_x(c,X,Y) \qquad \text{or} \tag{10}$$
$$C_{ij}(c,X,Y) = D_y(c,X,Y) \qquad \text{or} \tag{11}$$
$$C_{ij}(c,X,Y) = D_e(c,X,Y) \qquad \text{or} \tag{12}$$
$$C_{ij}(c,X,Y) = D_m(c,X,Y) \tag{13}$$

2.3.2 Number of modules per connection.

A second factor being important in the calculation of the matrix elements, is the number of elements being interconnected by each connection. In many cases three or more modules are linked together by one connection, e.g. system clocks and power- and ground nets.

When all elements in the matrix are calculated one by one, dividing these nets up into pin-pairs, they are likely to have a disproportionate influence on the final result. If for instance all modules to be interconnected with a particular module, are on the left side of it, the connection is most likely to be established by only one wire running from the source on the current module to the left. To balance the influence of all connections, their contribution to the matrix elements is divided by $(N-1)$, where N is the total number of modules being interconnected by the current connection (including the current module).

If all modules are on the same side, relative to the current one, their total contribution to the matrix elements will be about the same size in compared to the case where there is only one module (on the same side) being interconnected to the current module. If the "destination-modules" are spread out over the entire floorplan, their contributions to both possible orientations will be almost equal, thus having only little influence on the final result. Therefore:

$$F(c,X,Y) = \frac{1}{(\#_c - 1)} \tag{14}$$

A second reason to involve the number of modules per connection into the calculation, is found in the fact that if a connection is connected to a big number of modules, it will be widely spread over the floorplan (eg. power nets). Therefore, allowing the module to be easily connected to any part of the connection, independent to the module orientation. Mathematically, this influence can be written as:

$$F(c,X,Y) = (\#_c < (\#_{floor} / n)) \tag{15}$$

where $F(c,X,Y)$ is equal to 1 if $\#_c$ is smaller than $\#_{floor}/n$ or 0 if not.

In both equation (14) and (15), $\#_c$ stands for the number of pins being interconnected by connection c. $\#_{floor}$ is the number of modules in the floorplan. The constant n determines the maximum number of interconnected modules, still to have any influence on the result. In the final implementation, n was taken to be equal to 2.

A combination of equations (14) and (15) can also be used.

2.3.3 Module distance.

Since module orientation can only influence certain parts of the total wire length, a third point of interest is the distance between modules. If for instance the x-distance between two connected modules is very low, this doesn't mean these modules are close together. Several slices or modules can be placed between them.

In most floorplanners, modules are placed next to each other for several reasons. Quite often it will not be very important to minimize long connections at the cost of more important, short connections. The reason for this is that the modules positioned near each other are situated there because of the connections between them. In floorplanning stages, the longer connections are considered less important. The influence of each connection could therefore be divided by the distance between the modules concerned or a factor derived from this distance, to favor short connections. Or:

$$F(c,X,Y) = \frac{1}{\Delta(X,Y)} \tag{16}$$

where $\Delta(X,Y)$ represents the distance between the modules X and Y.

2.3.4 Other influences.

Other factors of influence can be thought of, but only a few will be mentioned here:

- A way to remove the influence of long connections could be the removal of the non-influential part of a connection; only count the connection length up to one of the modules corners. The part running from corner (of the current module) to corner (of the destination module) will most likely not be changed

by turning either module.

$$C_{ij}(c,X,Y) = (D_x(c,X,Y) - \Delta_x(X,Y)) \tag{17}$$

where $\Delta_x(X,Y)$ represents the x-distance between the two nearest sides of the modules X and Y. Similar equations can be found using D_y, D_e and D_m. Although this method has no influence if we just try to minimize this new value, it does show it's influence if it is used in combination with the factor $F(c,X,Y)$ discussed in the previous section.

- Modules that are placed within the same slice, are likely to be stronger tied together than modules not in the same slice.

$$F(c,X,Y) = S(X,Y) \tag{18}$$
$$S(X,Y) = \begin{cases} A & \text{if X and Y are in the same slice.} \\ B & \text{if X and Y are not in the same slice.} \end{cases}$$

where A and B are constants indicating the influence of in-slice and out-slice connections. Normally, $A > B$.

- Finally, in a number of floorplanners it is possible to attach weights to a connection. These weight can also be used in this orientation optimization part.

$$F(c,X,Y) = W(c) \tag{19}$$

Combinations of several factors are possible. If the euclidian distance is chosen together with the influence factors described in equations (14), (15) and (19), equation (5) can be written as:

$$l_{ij} = \sum_c \left\{ D_e(c,X,Y) * \frac{W(c)}{(\#_c - 1) * (\#_c < (\#_{floor} / n))} \right\} \tag{20}$$

2.4 Minimal wire length calculation.

After calculating the matrix elements, ways must be found to calculate the overall wire length. Since the modules can take two different orientations each, this means calculating 2^N possible configurations. This number can get quite large if N increases. Therefore, to find the minimal overall wire length without trying all configurations, several methods have been looked at. They are described below.

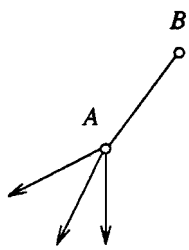
2.4.1 Trying all possibilities.

Despite what is said above, in some cases it could be useful to try all possible configurations. Since, in the implementation used, both directions are handled separately, the total number of configurations is limited to 2^N. Examples up to 10 or 12 modules (1000 - 5000 possibilities) can therefore be checked in a reasonable amount of time.

2.4.2 Matrix sweep method.

A second method to find the minimal overall wire length, is the matrix sweep method. In this method, we remove the modules from the matrix one by one, by adding their contribution to the wire length to other matrix elements.

To explain the way to do this, let us take a look at an example where one module in the floorplan is connected to only one other module. In figure 5 this has been illustrated by drawing a part of the graph giving the interconnections between the modules and a part of the corresponding matrix.



$$
\begin{array}{cccccc}
l_{aa} & 0 & l_{ab} & l_{a\bar{b}} & l_{ac} & l_{a\bar{c}} \\
0 & l_{\bar{a}\bar{a}} & l_{\bar{a}b} & l_{\bar{a}\bar{b}} & l_{\bar{a}c} & l_{\bar{a}\bar{c}} \\
l_{ba} & l_{b\bar{a}} & l_{bb} & 0 & 0 & 0 \\
l_{\bar{b}a} & l_{\bar{b}\bar{a}} & 0 & l_{\bar{b}\bar{b}} & 0 & 0 \\
l_{ca} & l_{c\bar{a}} & 0 & 0 & l_{cc} & 0 \\
l_{\bar{c}a} & l_{\bar{c}\bar{a}} & 0 & 0 & 0 & l_{\bar{c}\bar{c}}
\end{array}
$$

Figure 5. Removal of a 1-tree node (module B).

If module B is only interconnected to module A, it is called a 1-tree node. In this case, only four situations can be thought of: leave both modules in their original orientation, turn either A or B and leave the other, or turn both modules. However, for each orientation of module A (a or $\bar{a}$), an optimal orientation of module B can be found because of a shorter wire length. The total wire length belonging to one of the four possible orientation combinations, can be found in the matrix and are given below:

orientation ab is determined by $l_{aa} + l_{ab} + l_{bb}$ (21)
orientation $a\bar{b}$ is determined by $l_{aa} + l_{a\bar{b}} + l_{\bar{b}\bar{b}}$ (22)
orientation $\bar{a}b$ is determined by $l_{\bar{a}\bar{a}} + l_{\bar{a}b} + l_{bb}$ (23)
orientation $\bar{a}\bar{b}$ is determined by $l_{\bar{a}\bar{a}} + l_{\bar{a}\bar{b}} + l_{\bar{b}\bar{b}}$ (24)

So, for orientation a we must find the minimal value of $l_{aa} + l_{ab} + l_{bb}$ and $l_{aa} + l_{a\bar{b}} + l_{\bar{b}\bar{b}}$. Of course, a similar construction is available for orientation $\bar{a}$. Adding the diagonal elements l_{aa}, $l_{\bar{a}\bar{a}}$, l_{bb} and $l_{\bar{b}\bar{b}}$ may seem strange, since these elements are initially zero, but since the wire length of connections between A and B contribute to the overall wire length, they are added to the elements l_{aa} and $l_{\bar{a}\bar{a}}$ respectively. This introduces some kind of penalty for choosing a certain orientation of module A.

After performing this sweep step, the elements l_{aa} and $l_{\bar{a}\bar{a}}$ will be changed to:

$$l_{aa_{new}} = l_{aa_{old}} + \min((l_{ab} + l_{bb}),(l_{a\bar{b}} + l_{\bar{b}\bar{b}})), \tag{25}$$
$$l_{\bar{a}\bar{a}_{new}} = l_{\bar{a}\bar{a}_{old}} + \min((l_{\bar{a}b} + l_{bb}),(l_{\bar{a}\bar{b}} + l_{\bar{b}\bar{b}})), \tag{26}$$

Apart from this wire length addition, information on the orientation of module B must be saved also. For each orientation of the remaining module, an optimal orientation can be found for the swept one. This is done in separate data structures.

These structures contain an array representing all possible configurations of the modules being interconnected with the current one. For each of these orientations, the most optimal orientation of the current module is stored (eg. a implies $\bar{b}$ and $\bar{a}$ implies b). If the current module is interconnected with more than one module, this can result in an array having 2^N elements, where N is the number of modules, the current one is interconnected with.

This was sweeping a 1-tree node; now lets take a look at a 2-tree example. In figure 6 this has been illustrated. Analogous to the preceding description, an optimal orientation can be found, belonging to a certain orientation combination of the two modules involved. Since the optimal orientation of the module to be swept now depends on two other modules, the contribution belonging to this orientation must be added to non-diagonal elements, as indicated in figure 6.

The new elements l_{ac}, $l_{\bar{a}c}$, $l_{a\bar{c}}$ and $l_{\bar{a}\bar{c}}$ will now get the new values:

$$l_{ac_{new}} = l_{ac_{old}} + \min(\,(\,l_{ab} + l_{bc} + l_{bb}\,),(\,l_{a\bar{b}} + l_{\bar{b}c} + l_{\bar{b}\bar{b}}\,)\,), \tag{27}$$

$$l_{\bar{a}c_{new}} = l_{\bar{a}c_{old}} + \min(\,(\,l_{\bar{a}b} + l_{bc} + l_{bb}\,),(\,l_{\bar{a}\bar{b}} + l_{\bar{b}c} + l_{\bar{b}\bar{b}}\,)\,), \tag{28}$$

$$l_{a\bar{c}_{new}} = l_{a\bar{c}_{old}} + \min(\,(\,l_{ab} + l_{b\bar{c}} + l_{bb}\,),(\,l_{a\bar{b}} + l_{\bar{b}\bar{c}} + l_{\bar{b}\bar{b}}\,)\,), \tag{29}$$

$$l_{\bar{a}\bar{c}_{new}} = l_{\bar{a}\bar{c}_{old}} + \min(\,(\,l_{\bar{a}b} + l_{b\bar{c}} + l_{bb}\,),(\,l_{\bar{a}\bar{b}} + l_{\bar{b}\bar{c}} + l_{\bar{b}\bar{b}}\,)\,), \tag{30}$$

The module orientation information added is similar to the 1-tree node removal.

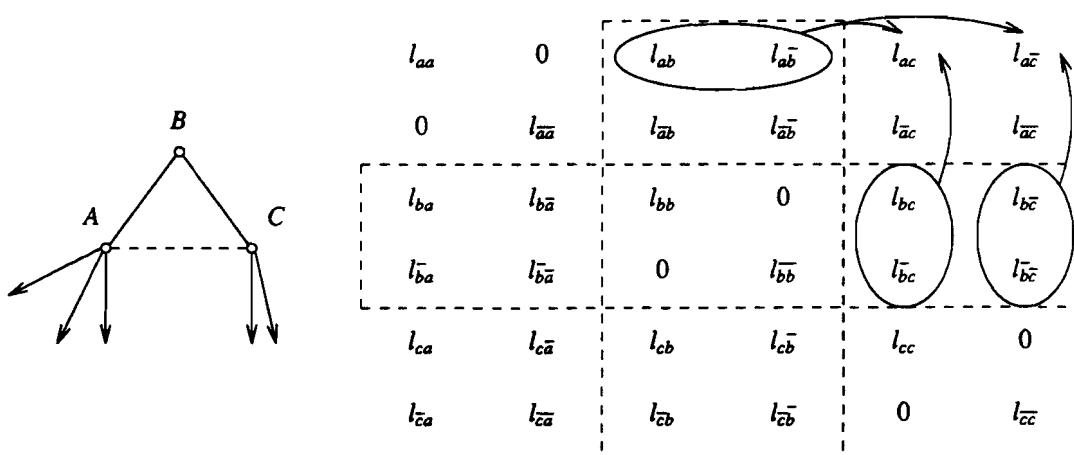


Figure 6. Removal of a 2-tree node (module B).

For 1-tree node and 2-tree node removal, this method works excellent. Problems arise however when three or more modules are interconnected to the one to be swept. In these cases, the wire length contribution should be added to elements containing information on a three or more module orientation combination (eg. $a\bar{b}\bar{c}$ or $\bar{b}d\bar{f}$ etc.). This implies three or more dimensional matrices. Since these type of matrices consume enormous amounts of memory space, the implementation used tries to solve the problem by adding the wire length contribution divided by the number of modules to be connected to the current

one, to elements contributing to the configuration given. If the module orientation combination given is $a\bar{b}\bar{c}d$, the contribution

$$\min\left(\left(l_{ae} + l_{\bar{b}e} + l_{\bar{c}e} + l_{de} + l_{ee} \right), \left(l_{a\bar{e}} + l_{\bar{b}\bar{e}} + l_{\bar{c}\bar{e}} + l_{d\bar{e}} + l_{\bar{e}\bar{e}} \right) \right) / \left(N - 1 \right), \tag{31}$$

is added to the elements $l_{a\bar{b}}$, $l_{a\bar{c}}$, l_{ad}, $l_{\bar{b}\bar{c}}$, $l_{\bar{b}d}$ and $l_{\bar{c}d}$, where N is the number of modules to be interconnected (here $N = 5$). The optimal orientation e or $\bar{e}$ is again stored in a temporary data structure as described earlier.

The selection criterion determining the module to be swept is obvious. The 1- and 2-tree node removals introduce no error in the final result. Sweeping a 3- or higher-order node does. Modules being interconnected to one or two others are therefore to be swept first. Sweeping one of these latter modules reduces the order of the module(s) it is interconnected with. In some cases, the entire matrix can be swept with only 1- and 2-tree node removals. This strategy reduces the overall error introduced by higher order node removal.

After sweeping all modules, except the last one, an optimal orientation can be found for this last module. Having found this orientation, it is possible to determine the optimal orientation of the module being swept last since it only depends on the last module. Continuing this "back substitution" optimal orientations can be found for all modules.

2.4.3 Scan line method.

A third method which has only briefly been looked into, is the scan line method. Since no serious research into this method has been done, it will only be discussed briefly here.

The main idea is to let an imaginary scan line divide the floorplan into two pieces. If this line is positioned at the left (or bottom) of the floorplan, it will not divide any connections in two. While it is moved to the right (or top) of the floorplan, a number of connections and modules will be divided. For each position of the line, the total wire length of connections starting on the left of (or below) the scan line, from their starting point up to the line is calculated. When the line reaches the end of a module, an optimal orientation can be found for this module calculating the minimal wire length as described above. While the line moves further, the optimal orientation of the modules on the left of (or below) the cut line is maintained.

It can be shown that the wire length only depends on the modules currently being divided in two by the cut line. After moving the line over the entire floorplan, all modules will be appointed their optimal orientation.

This dynamic programming method will result in an exact solution. The time necessary to solve the problem is exponential with the number of modules on the scan line. Although this method does not use the earlier described connection matrix, it can be used to solve the optimal orientation problem.

2.4.4 Average values.

Another way to find optimal module orientations using the connection matrix, is to calculate average values. While looking at a certain module, the wire lengths are calculated for both possible destination orientations. For instance, while looking at module A (having connections with module B), the contribution to the orientation a will be $(l_{ab} + l_{a\bar{b}}) / 2$. On the other hand, the contribution to orientation $\bar{a}$ will be $(l_{\bar{a}b} + l_{\bar{a}\bar{b}}) / 2$. To get the contribution of all connections, connecting module A to other modules, all elements on the matrix rows a resp. $\bar{a}$ must be added to each other, or mathematically:

$$A_x(X) = \sum_i (l_{xi} + l_{xi}^-) / 2 \tag{32}$$

$$A_{\bar{x}}(X) = \sum_i (l_{\bar{x}i} + l_{\bar{x}i}^-) / 2 \tag{33}$$

where $A_x(X)$ and $A_{\bar{x}}(X)$ represent the total wire length of the current module in orientation a and $\bar{a}$ respectively. Index i represents all modules in the floorplan. The optimal orientation for the current module is found by finding the minimal value of $A_x(X)$ and $A_{\bar{x}}(X)$. This way, optimal orientations for all modules can be found.

A safety-band can be used by allowing to link a certain orientation to a module only if one of the values $A_x(X)$ or $A_{\bar{x}}(X)$ is bigger than the other multipied by a constant bigger than 1. A problem which now rises is the fact that after one run, not all modules get an optimal orientation appointed. For these modules, new runs are performed until all modules have an optimal orientation. During these next runs, modules already having been appointed an orientation, only contribute the wire length corresponding to this orientation. Or in mathematical form:

$$A_x(X) = \sum_{i \in umod} ((l_{xi} + l_{xi}^-) / 2) + \sum_{j \in omod} l_{xO(j)} \tag{34}$$

$$A_{\bar{x}}(X) = \sum_{i \in umod} ((l_{\bar{x}i} + l_{\bar{x}i}^-) / 2) + \sum_{j \in omod} l_{\bar{x}O(j)} \tag{35}$$

where *umod* represents the set of unoriented modules and *omod* the set of oriented modules. $O(j)$ represents the orientation of module $j \in omod$: either j or $\bar{j}$.

It will be clear that this method tries to disconnect modules from each other by using average wire lengths.

2.5 Methods used.

A number of element calculation methods and minimal wire length calculation methods have been tested. The combinations tried out are given below. The results of a number of test runs using these combinations are given in chapter 6. The names used in the mathematical descriptions are equal to those described earlier in this chapter.

2.5.1 Method 1.

In the first method used, the matrix elements are calculated using the X- resp. Y-distance, the module distance and the number of interconnected modules. The factors $C_{ij}(c,X,Y)$ and $F(c,X,Y)$ used in equation (5) can be written as:

$$C_{ij}(c,X,Y) = D_x(c,X,Y) \qquad \text{or}$$

$$C_{ij}(c,X,Y) = D_y(c,X,Y) \tag{36}$$

$$F(c,X,Y) = \frac{(\#_c < (\#_{floor} / 2))}{(\#_c - 1) * \Delta_e(X,Y)} \tag{37}$$

To calculate the optimal orientation, the average value method is used. If the wire length belonging to orientation i is bigger than (1.1 * $A_{\bar{i}}(I)$), orientation i is appointed. If ($A_{\bar{i}}(I)$ > (1.1 * $A_i(I)$)),orientation $\bar{i}$ is appointed. For modules not having an orientation appointed after one run, new runs are done. If after a while, no new orientations are found while there are still unoriented modules, the module having the best ($A_{\bar{i}}(I) / A_i(I)$)- or ($A_i(I) / A_{\bar{i}}(I)$)-ratio will be oriented.

2.5.2 Method 2.

This method uses exactly the same matrix element calculation and optimal orientation method as used in method 1. The difference however is found in the fact that each orientation run only appoints an orientation to the module having the best ($A_i^{\bar{}}(I) / A_i(I)$)- or ($A_i(I) / A_i^{\bar{}}(I)$)-ratio. The number of runs is therefore equal to the number of modules.

2.5.3 Method 3.

The matrix elements used in this method depend on two factors: the X- resp. Y-distance and the number of interconnected modules.

$$C_{ij}(c,X,Y) = D_x(c,X,Y) \qquad \text{or}$$

$$C_{ij}(c,X,Y) = D_y(c,X,Y) \tag{38}$$

$$F(c,X,Y) = (\#_c < (\#_{floor} / 2)) \tag{39}$$

To find the minimal overall wire length, the matrix sweep method described in paragraph 2.4.2. is used.

2.5.4 Method 4.

Although this method also uses the matrix sweep method to find the minimal overall wire length, the matrix elements in this case depend on the euclidian distance and the number of interconnected modules. The number of interconnected modules has a larger influence on the final result than in the previous method.

$$C_{ij}(c,X,Y) = D_e(c,X,Y) \tag{40}$$

$$F(c,X,Y) = \frac{(\#_c < (\#_{floor} / 2))}{(\#_c - 1)} \tag{41}$$

2.5.5 Method 5.

In method 5, the matrix element calculation method is equal to the one used in method 4, but the orientation calculation method is different. If the number of modules in the floorplan is sufficiently small, all possible configurations are checked on minimal wire length.

2.6 Program implementation.

To perform the module orientation step, two programs have been written. Using these programs, the output files generated by the floorplanner programs are transformed to a suitable format to be used for orientation calculation. Below, the separate programs are discussed briefly.

2.6.1 JOIN.

In the **join**-program, the floorplanner output files are read and the information is stored, grouping all module resp. pin information together. The files read are: **intfaces**, **geometry.mod**, **modules**, **pinptrs.mod**, **plan.ldm** and **modtonet.pin**. Module rotation information is updated and slice dimension information is being calculated. Also, channels are added to the floorplan, having an initial width zero and an initial height equal to the slice's height. Information on the modules (sizes, orientation, slice information and names) are than stored in the file **join.mod**. Pin information is being written to the file

join.pin.

2.6.2 ORIENT

The real orientation routine is embedded in the **orient**-program. First, the **join.mod** and **join.pin** files are read. After some initial calculations and constructing a connection list, physical pin positions are derived from the relative values. Using these physical positions, orientation calculation is done in two steps, using the routine MIRROR() in which both matrix element calculation and module orientation is done. After this, the new, relative positions are calculated and written to the **orient.mod** and **orient.pin** files. These files use the same format as the **join.mod** and **join.pin** files.

3. *Pin position calculation.*

Not only module orientation can influence wire lengths and floorplan dimensions. Pin positions can also lead to more optimal situations. After finding a certain orientation, many module descriptions allow multiple pin positions (see Appendix A). The next step to optimize floorplans must therefore be the calculation of optimal pin positions. The calculated pin positions must be within the boundaries set in the description. In this chapter, a method will be described, together with a short description of the program implementation.

3.1 Optimal pin positions.

After calculating optimal module orientations, the physical pin positions are known. Sometimes, these positions consist of one coordinate only, but it is also possible that the position is represented by an interval, described by boundary positions along the modules edge. An example of this latter possibility is shown in the figure below.

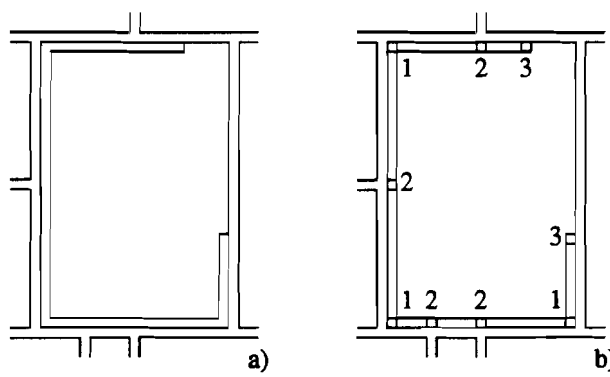


Figure 7. Optimal pin position appointment. a) pin position interval. b) possible pin positions.

Before trying to find one optimal pin position, it is important to determine which positions along the "pin-interval" can be considered for this optimal position. There are three kind of possible positions:

1) Module corners; At these places, channels to other parts of the floorplan "leave" the module, so it is likely for the wires to be routed through these channels. This type of position is comparative with the next one,
2) Channel intersections; Here channels intersect with module edges.
3) Area boundaries; These positions are useful if no channel intersection or module corner is within the pin-interval, or if the pin-interval is opposite to the destination position. For example if the destination is to the left of the current module while the pin-interval is on the right.

In figure 7 b), these positions are given for the pin-interval and channel positions shown in figure 7 a).

In cases of adjacent modules this method is however less effective. If part of the corresponding pin-intervals are opposite to each other, this method only finds possible opposite pin positions in a limited number of cases. If interval boundaries of both pin-intervals are at the same physical position, or if one of the module corners within the pin-interval is also contained by the opposite pin-interval, correct positions are found. A module corner causes a channel intersection on a adjacent module.

In all other cases, the possible pin positions described before represent a collection of the best possible positions.

3.2 Using a global router.

Finding the possible pin positions using the method described above, doesn't solve the problem, but it simplifies the solution considerably. Now, it is possible to use a global router to route all connections as optimal as possible.

The global router used [6] recognizes several pin types. They may be one of the following types:

- Internally interconnected pins: A number of pins on the module are grouped together. Normally these pins carry the same name apart from a different appendix (eg. CLOCKa, CLOCKb and CLOCKc). Pins belonging to the same group are supposed to be interconnected inside the module. Nets containing these pins (eg. CLOCK) may be connected to any of the pins within the group. In this case: CLOCKa, CLOCKb or CLOCKc. Other nets connected to the same pin may be connected to the same or other pins within the group.
- Individual pins; Pins not being internally interconnected with any of the other pins on the module.

If the first type of pins is used, pin-intervals can be described by a series of pin positions as described in paragraph 3.1. Running the global router will reveal certain pin positions to be used to realize the connections. So, after using the global router, it will be possible to determine which pin positions are used most frequently. These pin positions are supposed to be the most optimal pin positions. The pin positions that are not (or less frequently) used by the global router, can now be removed and the optimal pin positions remain.

3.3 Program implementation.

The process described above has been implemented. A number of programs have been developed to perform the different steps. Also, some minor changes have been made to the global router to allow error-free execution.

3.3.1 FILES.

In the **files**-program, the input files **orient.mod** and **orient.pin** are read, and possible pin positions are calculated. The result is written to the files **florplan**, **netlist** and **mod2net**, containing information on the floorplan, nets to be routed and pin descriptions respectively. These three latter files are formated in a way, the global router is able to read.

3.3.2 REMOVE.

After running the global router, its output files **mod2net.app** and **channels** and the input file **mod2net** are read by the **remove**-program, in order to remove the less frequently used pins. The output file **mod2net.tmp** contains the optimal pin positions using the same format as used in the files **mod2net** and **mod2net.app**.

4. *Channel width estimation.*

A final problem to be solved before resizing is the channel width estimation. In this chapter a general description will be given of the method developed. After this, some special properties of the estimation method used will be described. At the end of the chapter an overview of the programs written to perform the estimation will be given.

4.1 General description.

After finding exact pin positions, like described in chapter 3, the channel widths have to be found. To get accurate channel width estimation, one has to know how the router used in stages still to be performed, will place the connections. At this stage, the floorplan layout will be slightly different from the one that will be found after resizing. Since it is the only one available and since the final floorplan with channel space will look something like a blown up version of the current one, it will do as an approximation of this final floorplan. If we use the global router again to route all connections, we will get a fairly accurate estimation on how the connections will be routed in the final floorplan.

From the output of the global router, information can be obtained on how the connections are routed. The **mod2net.app** file contains extra produced pin positions located at channel intersections and used by the global router. The file uses the same format as the input file **mod2net.** The **channels** file contains information on which pin positions are interconnected. These pin positions are named in either **mod2net** or **mod2net.app**. This information is ordered by channel. For every channel available, the interconnections between pin positions are given. The pin positions are bounded by or are part of the current channel.

Having obtained this information, the next step must be to find the maximum amount of connections crossing any section of the channel. Since the channels all have an initial width equal to zero, all pin positions leaving or entering the current channel at a channel intersection will have the same physical position. In figure 8 b. this means that pin positions A1 to A5 all have the same physical position.

To obtain the maximum width, an array is used, with each element representing a micrometer part of the channel. For every connection running through the channel, a contribution is added to the array on the interval it is occupying. The connection A3-E adds one unit to the array elements x_{A3} up to x_E; where x_{A3} and x_E are the x-coordinates of point A3 resp. E. For a vertical channel, y-coordinates must be used. The unit added depends on the connection width given in the file **channels** and eventually altered using an available program option.

A small problem rises since connections using channels C and D will have the same physical position. For instance connections A5-C2 and C1-B2 can be routed in one track. To prevent the program from adding two contributions to the coordinate x_{C1} (= x_{C2}), the contribution adding starts at the coordinate next to the starting point and ends one coordinate before the end point. This way, the most probable channel width is estimated.

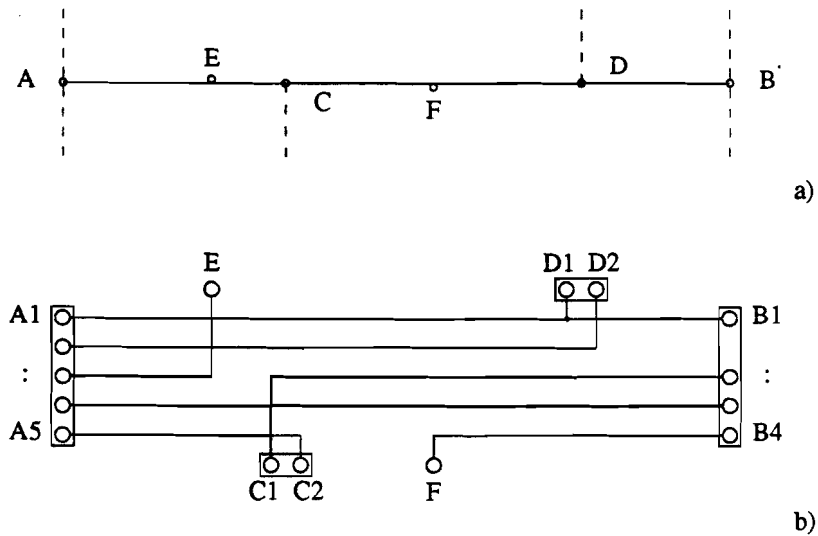


Figure 8. Channel routing example. A, B, C and D are channel intersections, E and F are pin positions on modules. a) Schematic channel representation. b) Possible routing diagram

There is a cheaper and faster way to calculate the maximum channel widths. First all connection points are sorted on increasing x-coordinate for horizontal channels. For vertical channels y-coordinates must be used. Then the connection points are given a number according to this ordering. In the example in figure 11 this means that pin A1 gets number 1, A2 to A5 also get 1, E gets 2, C1 and C2 get 3 etc. Now an array is created in which the number of array elements is limited to the number of different coordinates minus one. The contribution of a connection running from a pin with number x to a pin with number y will be added to elements x to $y-1$, if x smaller then y. This calculation method in much cheaper then the one mentioned earlier. In the program used for testing, the first method is used. The description given below will therefore be based on this method. Alterations to allow the second method to be used are minimal.

4.2 Special features.

The description given above is a general one. The developed program has some special features. Three of these features will be glanced at below.

4.2.1 Power and Ground connections.

Although the global router routes all connections, it treats the power and ground connections in a different way. This quite often leads to strange connection patterns. To prevent faulty contributions to the array mentioned in paragraph 4.1., power and ground connections do not contribute to the array elements. In stead, every channel is widened with a default width which is partly controllable by the program option mentioned earlier. Since the global router requires power and ground pins to be available on two opposite sides of each module, it will be possible to route all power and ground connections using this default connection space.

4.2.2 Equal-connection contribution.

In some cases, the **channels** file contains rather remarkable connection combinations. If for instance the connection A1-D1-B1 must be routed, this is sometimes done by routing connection A1-B1 and connection B1-D1. This combination however, causes a double contribution for this connection on the

interval D1-B1. The estimator locates all connections belonging together, finds the overall connection interval and adds only one unit to the array elements intersecting with the interval.

Adding only one unit is most likely allowed since the final router will probably route one connection from A1 to B1 with a via near D1 to connect it with channel D.

4.2.3 Program options.

Beside a verbose option -v, also a default connection width alteration option is supported by the program. Default width value for every connection is 15 micrometer. If the option -s is used, all minimal connection widths are set to 5 micrometer, the -m option causes a minimal width of 10 micrometers. Although only three values are possible now, this could easily be altered to allow the user to enter a number to be used as width value.

The connection width set by the options -s or -m, or the default value, is a minimal value. If the **channels** file indicates that is needs a wider width for a certain connection, this latter width is used.

4.3 Program implementation.

To perform the estimation, the **estimate**-program has been written. In this program, the files **florplan**, **mod2net**, **mod2net.app**, **channels** and **Breaks.mod** are read. After this the minimal channel widths are calculated using the method described in paragraph 4.1. The channel widths are than stored in the **breaks.mod** output file. The channels in the **breaks.mod** output file have zero length but estimated width, to allow the floorplanner to make arbitrary long channels.

The input file read carries the name **Breaks.mod**, which is the original **breaks.mod** file generated by the floorplanner. For reasons of testability, the name has been altered to **Breaks.mod** using the "**mv breaks.mod Breaks.mod**"-command under UNIX. The name of the output file is **breaks.mod** since the **geometry** program uses this name for the input file. The **geometry**-program will be used in chapter 5 for resizing purposes.

5. *Resizing.*

After finding optimal module orientations, calculation of the pin positions and estimation of the channel widths, a number of floorplanning steps must be repeated to review the initial floorplan.

Initially, the channels in the floorplan are assumed to have zero-width. After performing the module orientation step, the pin position calculation step and finally the channel width estimation step, the channel widths are known approximately. When these channels are added to the floorplan, a new calculation step can be performed to establish new module dimensions. Better fitting in the reviewed floorplan is only possible if adjusted module dimensions are allowed in the module description.

This resizing step can be performed by using the **geometry**-program from the initial floorplanner [1] with expanded input files. The expansion of the input files will be caused by adding the channels and their dimension information. Adding this extra information to the input files is established by writing a subroutine to the **estimate**-program, producing a new **breaks.mod**-file and a new **tree.mod**-file.

To the original **geometry**-program, some changes had to be made. Since the original floorplanner did not calculate initial orientations, the program assumed all possible orientations to be allowed. To prevent the program to change the orientation found in the module orientation step, extra code was added to several floorplanner programs to allow them to find out if the modules may be rotated or not. This resulted in an extra column in the **breaks.mod**-file and therefore the **Breaks.mod**-file. The added column indicates weather the module being read may be rotated (1) or not (0).

6. *Results and conclusions.*

After implementing all algorithms described before and adding a number of changes to the original floorplanner programs, a number of test runs have been made using the orientation methods as described at the end of chapter 2. In this chapter, a survey is given of the programs to be run to get more optimal floorplans, together with the results of the test runs.

6.1 Program sequence.

In order to get more optimal floorplans, the programs developed together with those already available, must be run in a fixed sequence. In the figure below, this sequence is shown together with a description of their meaning in the total process.

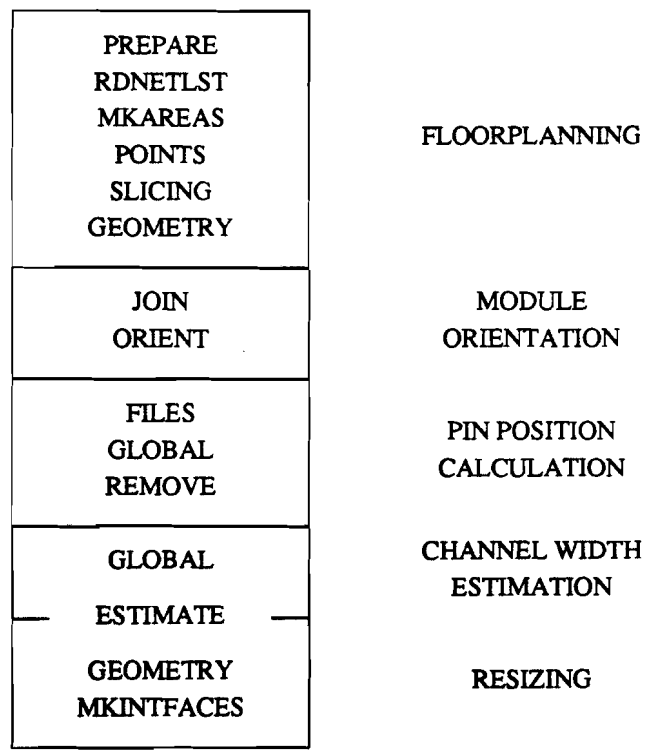


Figure 9. Final program sequence.

Since a number of files are produced by these programs, but also used and altered in other programs in the sequence, a number of these files must be copied to temporary ones. The complete sequence of running programs plus performing the copying can be executed using a command file.

If a visual representation of the **geometry.mod**-file is wanted, the **show**-program can be used. This program uses the **geometry.mod**-file and the **plan.ldm**-file as input and produces a **view.mod**-file containing PIC-information that can be printed using the command: **esmmt -p -rN4 view.mod**.

The file names mentioned are default names which can be altered using one of the program options. All program options are given when executing the command: **show -h**.

6.2 Test results.

In order to test the program sequence described above and to see its effect, a number of examples have been developed. The seven examples contain both small (ex. 3 and 7) and big floorplans (ex. 5 and 6). Some examples contain only a few modules (ex. 5 and 7), others like ex. 1 contain a considerably larger number.

One example contains pin intervals rather than single coordinate pin positions (ex. 3) while two examples uses module descriptions with piece-wise linear shape constraints (ex. 3 and 5).

The program sequence has been run for these examples using the module orientation methods as described in chapter 2. The results of these test runs are given in the table below. The orientation methods are numbered in the same way as has been done in chapter 2. Initial and final floorplans are given in Appendix C.

	original	method						
nr	floorplan size	-	1	2	3	4	5	#
1	593 * 552	100.0	89.5	91.7	93.2	91.3		64
2	261 * 159	100.0	92.4	87.7	102.5	105.2	94.5	12
3	183 * 147	100.0	89.3	91.1	106.9	108.7	112.1	12
4	224 * 244	100.0	86.0	86.0	79.5	93.4		17
5	2151 * 1400	100.0	86.9	87.7	84.3	84.3	93.9	10
6	8913 * 10734	100.0	89.5	89.5	100.3	101.2		16
7	80 * 70	100.0	69.5	69.5	100.0	100.0	95.0	4

TABLE 1. Results from 7 test examples. The sizes in the second column are floorplan sizes before module orientation, pin position calculation and channel width estimation. The numbers shown are percentages relative to the floorplan size without module orientation indicated by -. The last column contains the number of modules in each example.

Although the object function used in chapter 2 was the minimization of the overall wire length, our main goal was to find minimal floorplan dimensions. Information on the overall wire length is not available, because of the fact that during the entire process, only little information is available on true wire length. The differences in floorplan size among the methods used are quite big. A considerable difference in overall wire length may therefore be expected.

6.3 Conclusions.

From the results shown above, the following conclusions can be drawn:

- The number of elements per connection only plays a small role in the final result. The difference between method 3 and method 4 is rather small.
- The distance between modules seems to have an important influence on the final result. In methods 1 and 2 the module distance is used to affect the scaling factor used to reduce to wire length contribution; while in methods 3, 4 and 5 this type of scaling is omitted.
- "Disconnecting" modules seems to have a positive influence on the result of the test runs. Again methods 1 and 2 which try to disconnect modules give much better results than the methods 3, 4 and 5 which try to reduce wire lengths depending from all module orientations.

The results given in table 1. do not show enough. Looking at the layouts made using the SHOW program reveal something else. Often an enormous improvement in floorplan size can be made if modules are located on more optimal positions in the initial floorplan. An example can be seen in Example 5 (Appendix C) where block 10 was heavily interconnected with blocks 1 and 3. If, in the initial floorplan, block 10 was located under either block 1 or 3, this would have lead to a much smaller floorplan.

The overall result of the research done during the research period is that a considerable size improvement can be established using floorplan optimization. In the best case up to 30 per cent smaller floorplans were found, while the average improvement was about 11 per cent.

7. *Recommendations for future work.*

Since the time available to perform the work is limited, a number of options found during the research could not be checked. In this chapter a few of these points are mentioned for future research.

While working on methods to solve the matrix problem it was found that solving this problem exactly would consume a lot of time. The matrix sweep method greatly improves the computer time used, but the solution found will may be suboptimal. The sub-optimal treatment of 3- or higher-order node removals indicates this. The scan line method mentioned in paragraph 2.4.3. will lead to optimal results but it has not yet been implemented. This method however will use an exponential amount of time.

A second point of interest is the number of possible combinations of both matrix element calculation and matrix problem solving methods. Only five possible combinations have been tried. If a method has been found to solve the matrix in a reasonable time, it might be possible to try out all possible matrix element calculation methods one by one to find the best.

Appendix A.

MODULE DESCRIPTION.

In the floorplanner output files the modules are described in a way similar to the one given below.

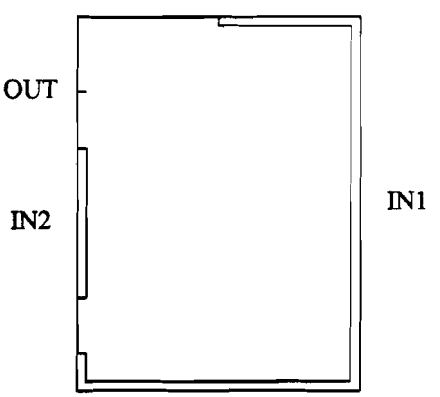
After the keyword MODULE, the module's name is given. A keyword SHAPE is next, followed by a sequence of number pairs. These numbers describe the piece-wise linear shape constraint. Now the pin descriptions will follow.

Pin descriptions contain the pin's name, and a relative start and end point. The lower left corner of the module is position 0.00, lower right is 1.00, upper right corner is 2.00 and the upper left corner is indicated by 3.00. If a pin description gives 3.35 to 3.75 as possible positions, this means that the corresponding pin can be located at any position along the interval 3.35 to 3.75. The first number indicates the starting point, the second the end-point. The interval is found in the counter clockwise direction starting at the starting point. If a pin can be located at any position along the modules perimeter, this is given by the numbers 0.00 to 4.00. The numbers 0.00 to 0.00 describe a point at the lower left corner of the module.

The module description is closed by the keyword END.

Below, a module description is given together with a graphical representation of it. The first three pins have been drawn. The POWER- and GND-pin have been omitted.

```
MODULE EXAMPLE
SHAPE 30 80 40 50 45 42
PIN IN1 3.90 2.50
PIN IN2 3.35 3.75
PIN OUT 3.20 3.20
PIN POWER 0.00 4.00
PIN GND 0.00 4.00
END
```



Appendix B.

COMPLETE COMMAND SEQUENCE.

To perform the entire process of floorplanning, optimization and resizing, a number of programs have to be run in a certain sequence. Below, an example of a command file is given, that can be used to run the entire sequence.

```
prepare lib.ldm modules
rdnetlst modules terminal
mkareas modules intfaces
points pinptrs.mod modtonet.pin weights.mod
slicing breaks.mod coord.mod
geometry breaks.mod tree.mod
instance geometry.mod
show -a -m
cp breaks.mod Breaks.mod
cp tree.mod Tree.mod
cp geometry.mod Geometry.mod
cp plan.ldm Plan.ldm
cp view.mod View.mod
cp intfaces Intfaces
join -v
orient -v
files -v
global -v
remove -v
mv mod2net.tmp mod2net
global -v
estimate -v
geometry breaks.mod tree.mod
mkintfaces -v
show -m -o -a
```

The copy and move commands (cp and mv) are used to save and rename a number of files for comparison purposes. The show program has been used to output floorplan layouts.

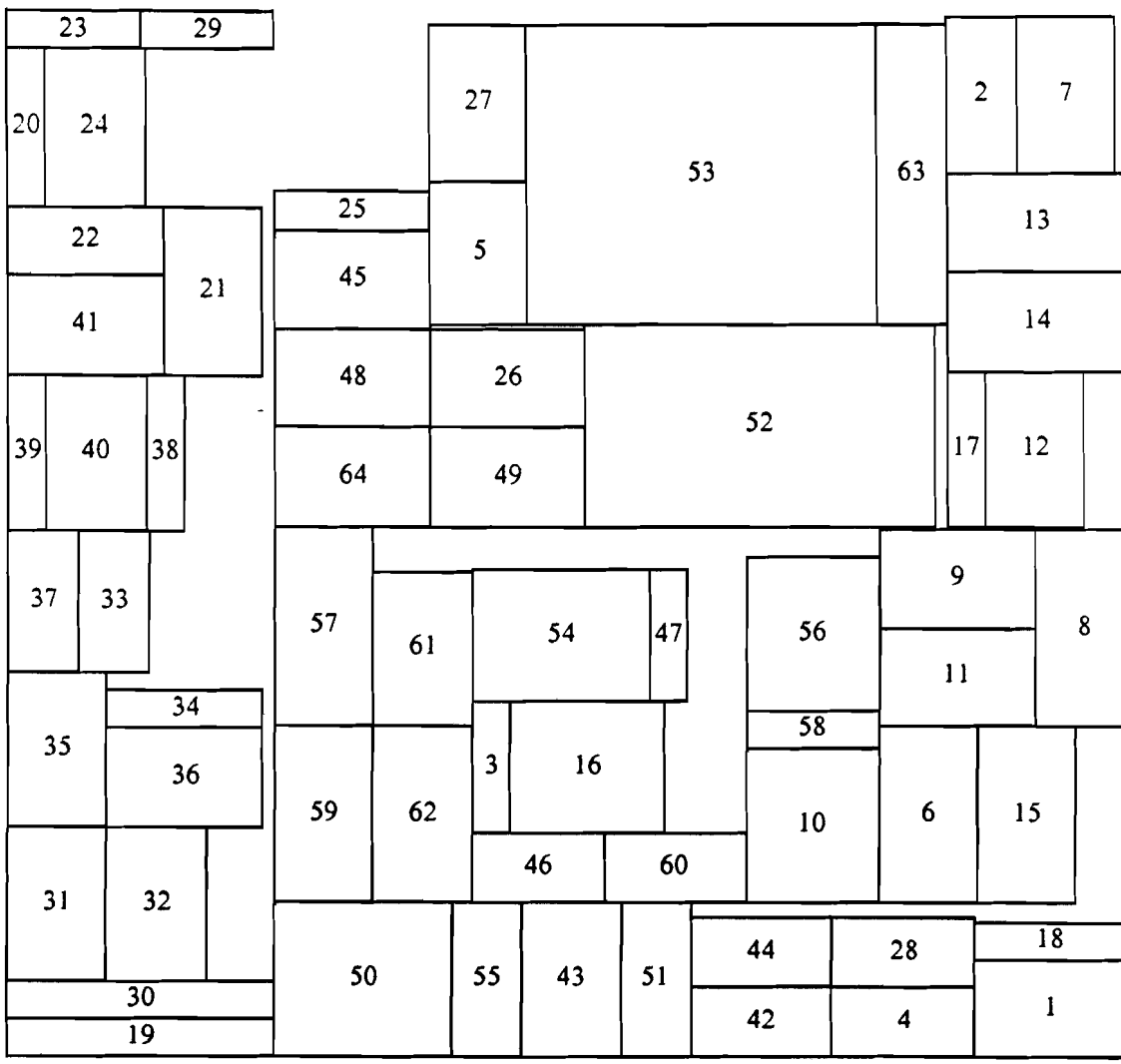
Appendix C.

FLOORPLAN LAYOUTS.

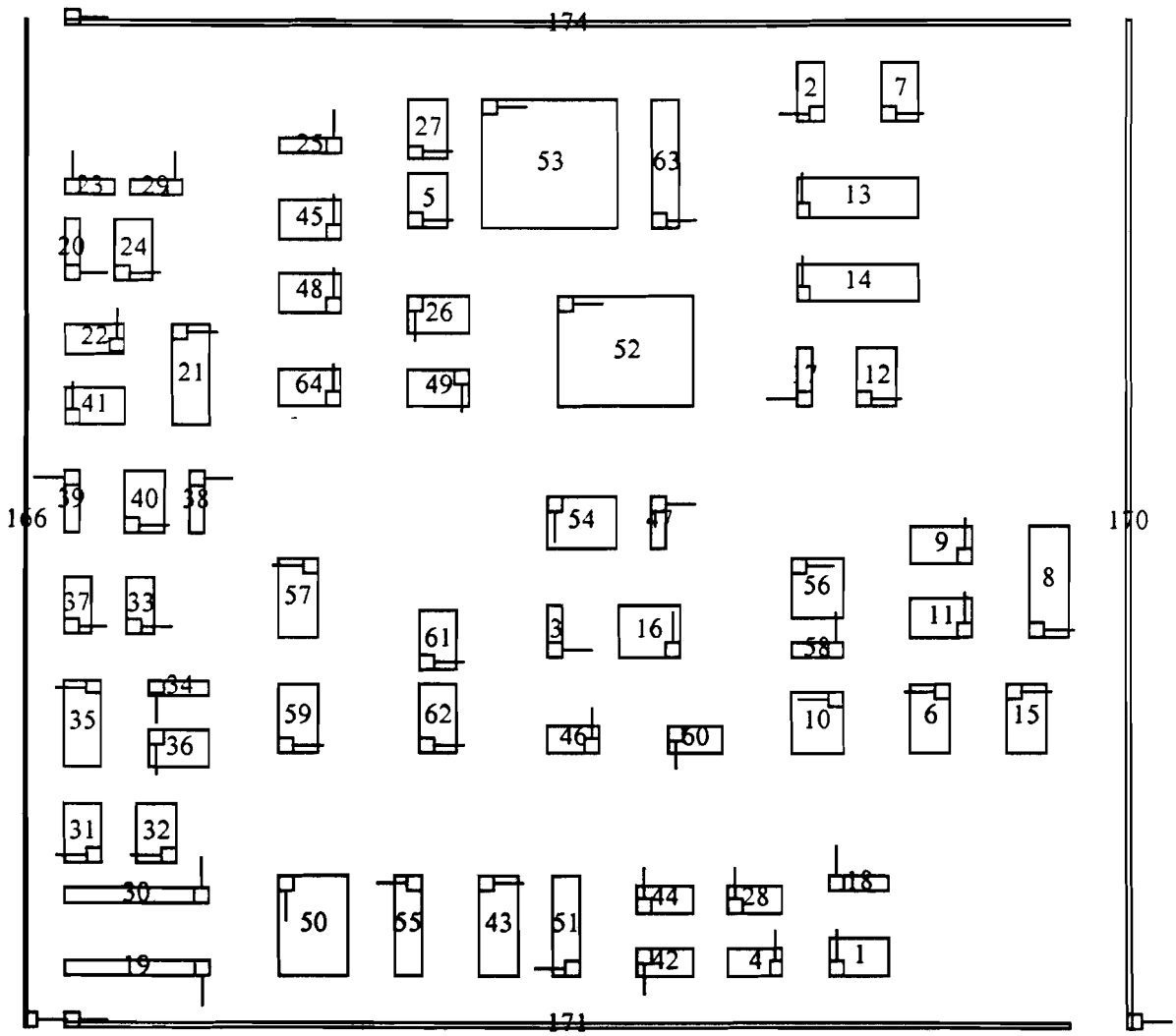
In this Appendix, the layouts are given of the examples mentioned in chapter 6, which were used to perform the test runs. The numbers of the layout correspond with those given in Table 1, chapter 6.

The first layout shown is the original, non-channel layout; the second layout is the layout found using method 1. The ticks in one of the modules corners represent the original origin. The little line points along the original base edge of the module (interval 0.00 to 1.00, see Appendix A).

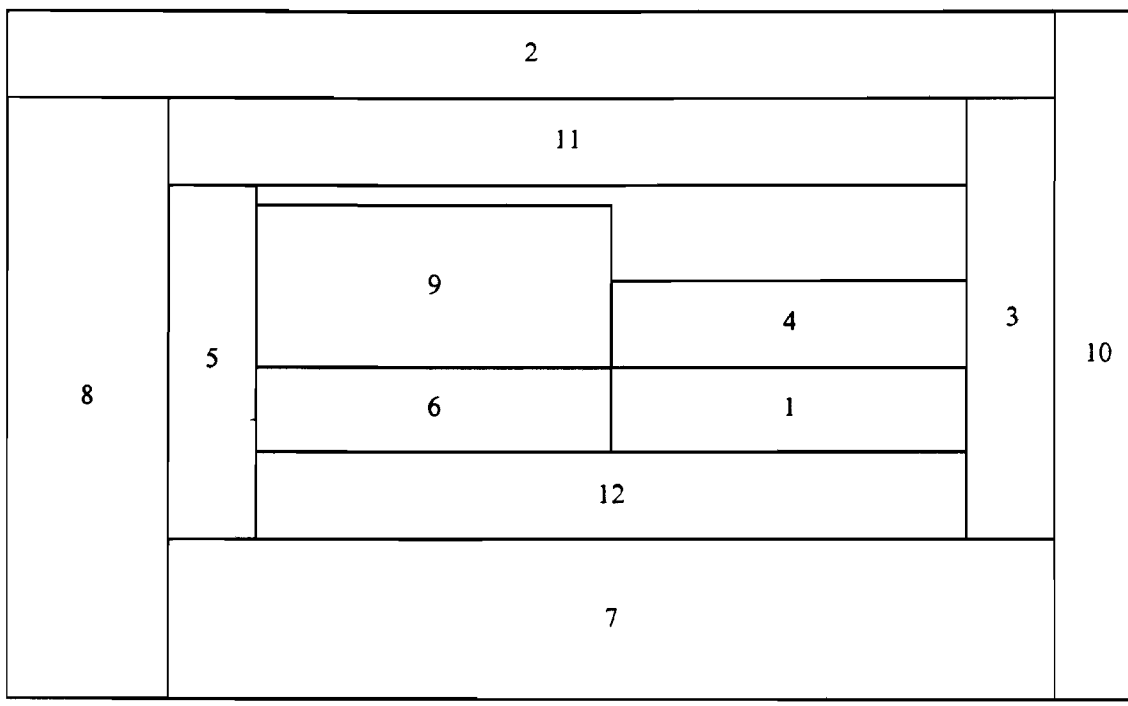
Example 1, no channels (size 593 * 552).
No module orientation.



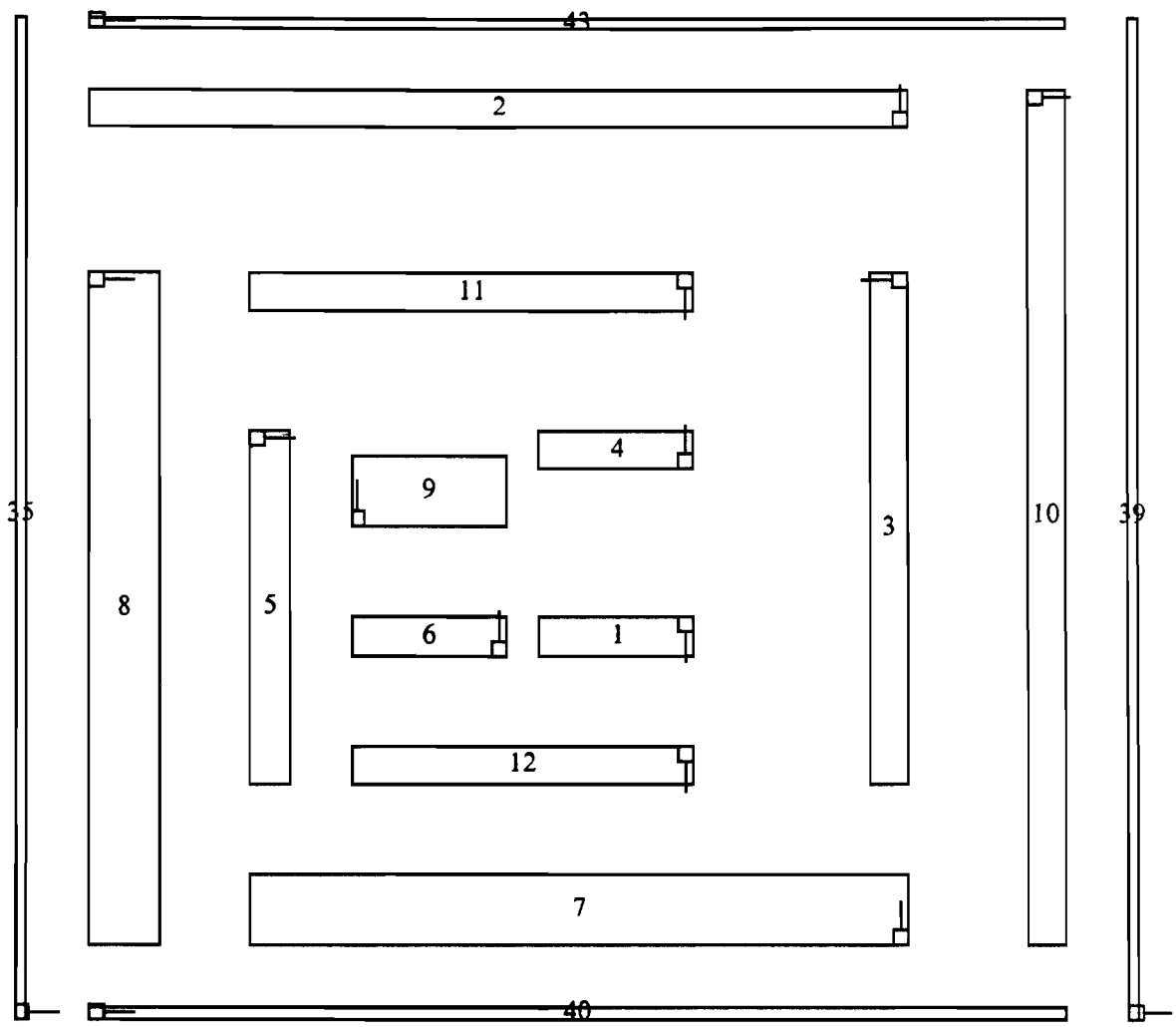
Example 1, channels (size 1499 * 1369).
Module orientation method 1.



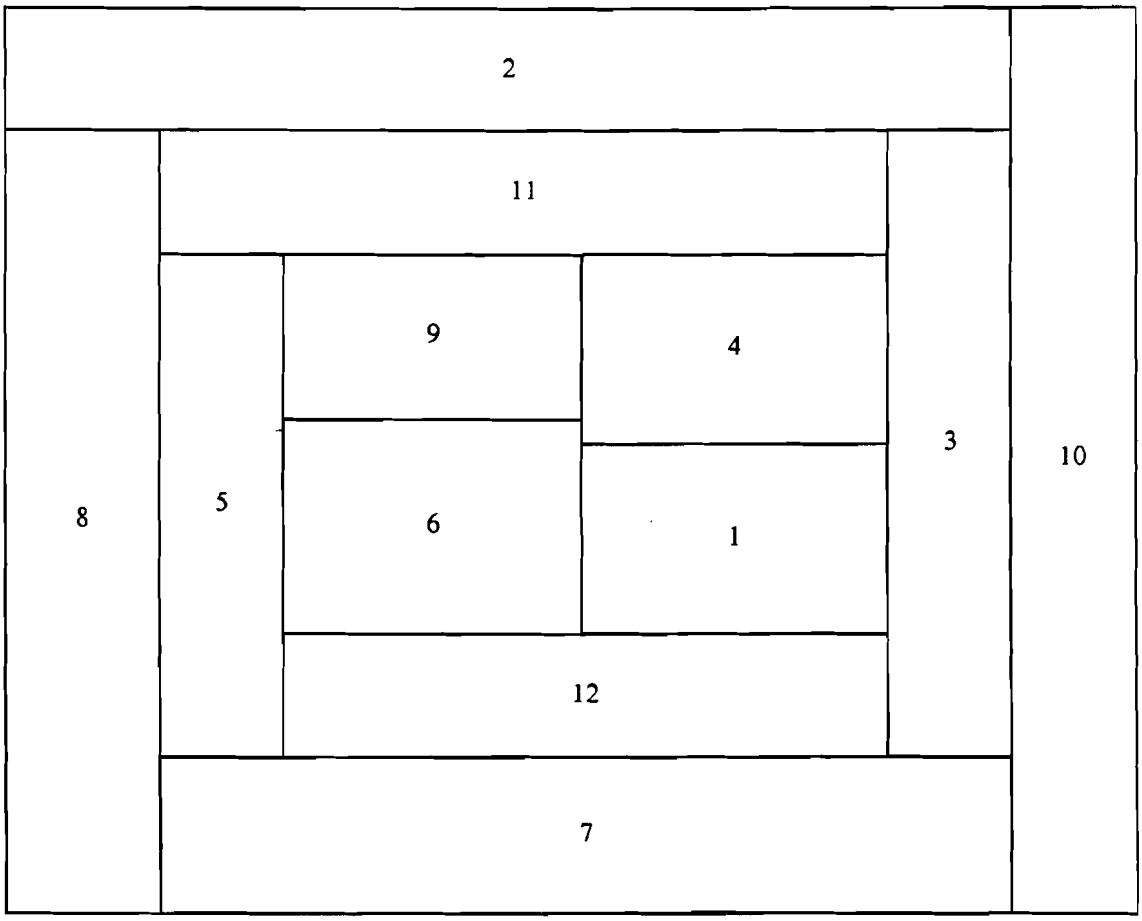
Example 2, no channels (size 261 * 159).
No module orientation.



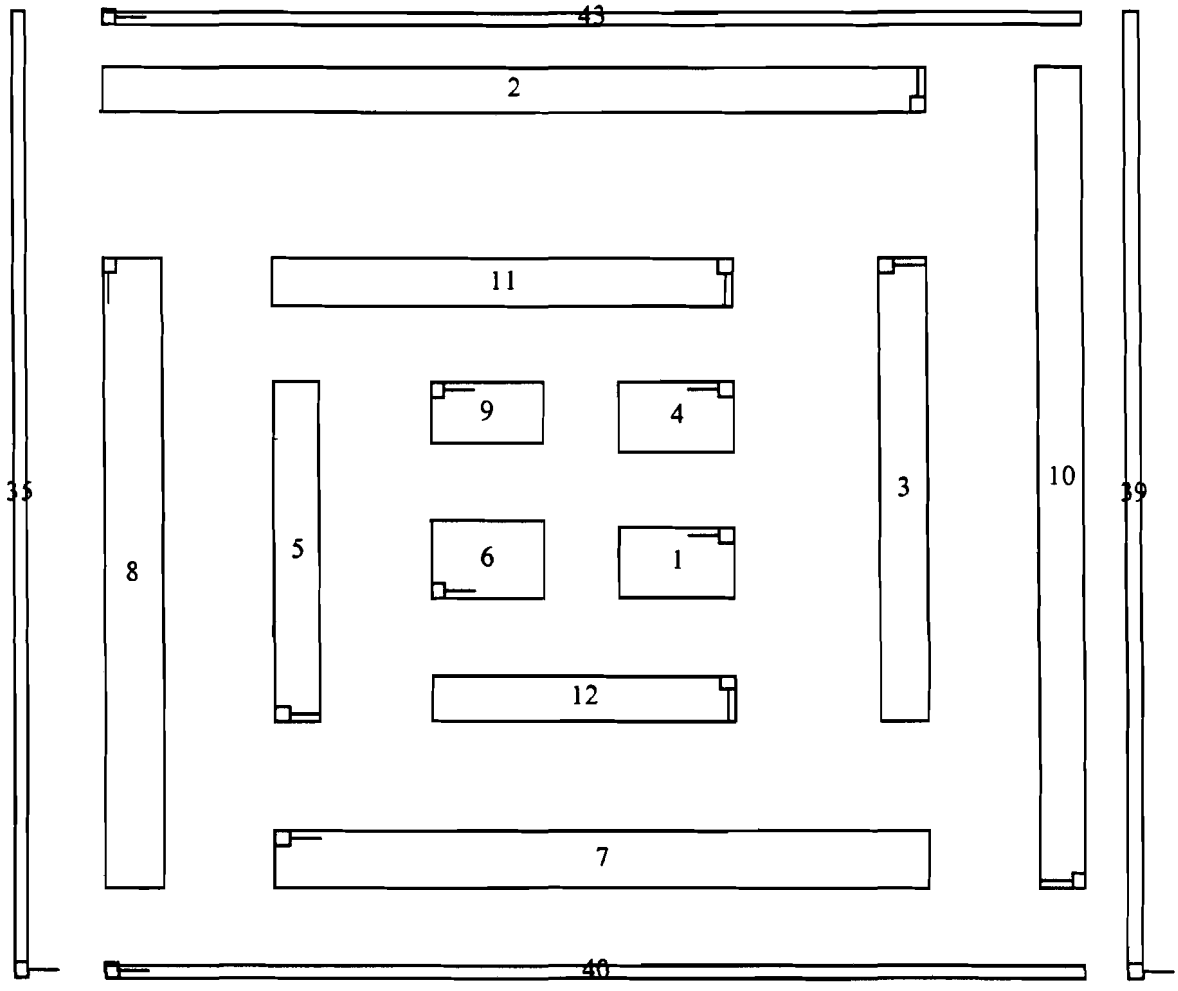
Example 2, channels (size 594 * 530).
Module orientation method 1.



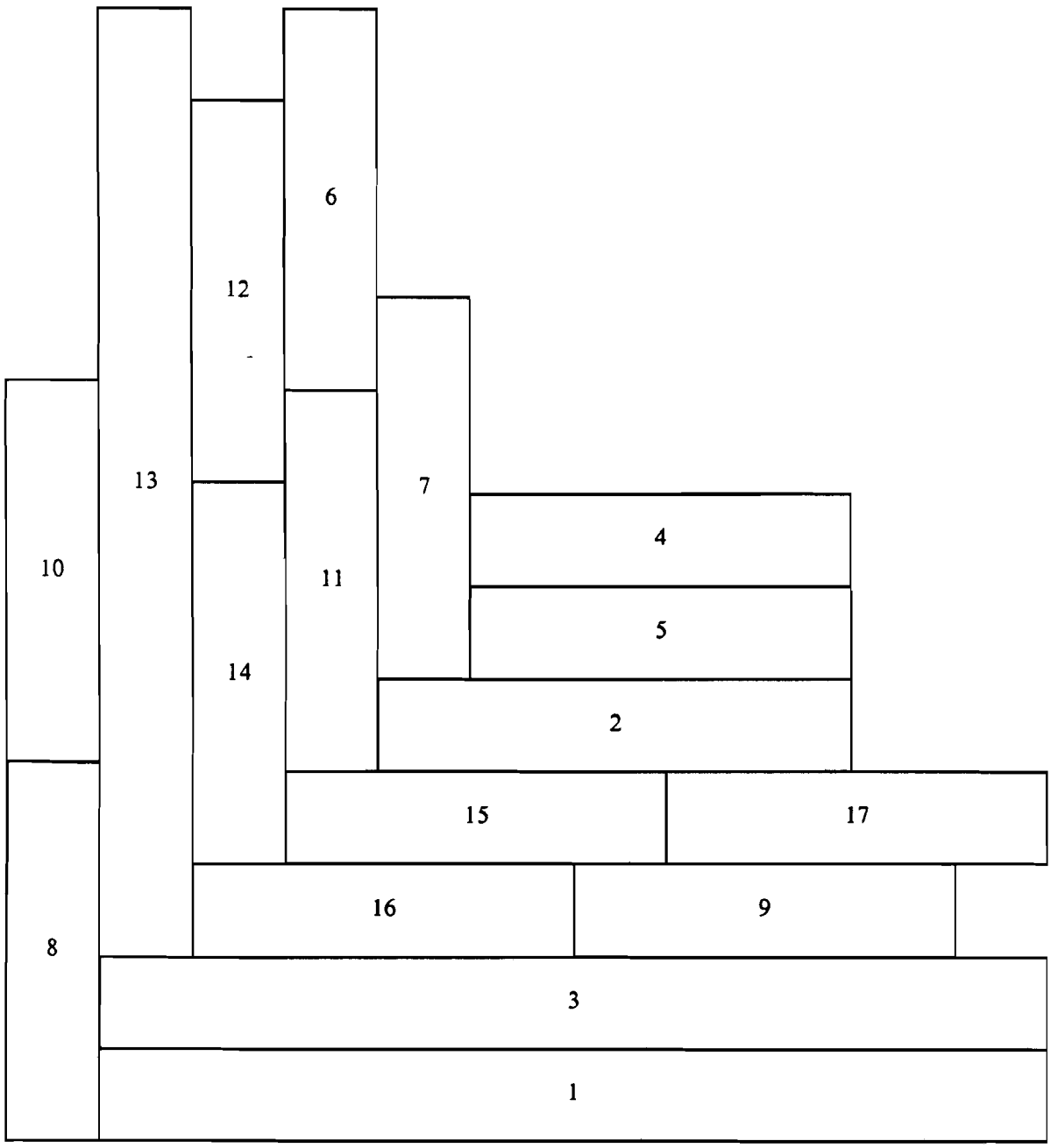
Example 3, no channels (size 183 * 147).
No module orientation.



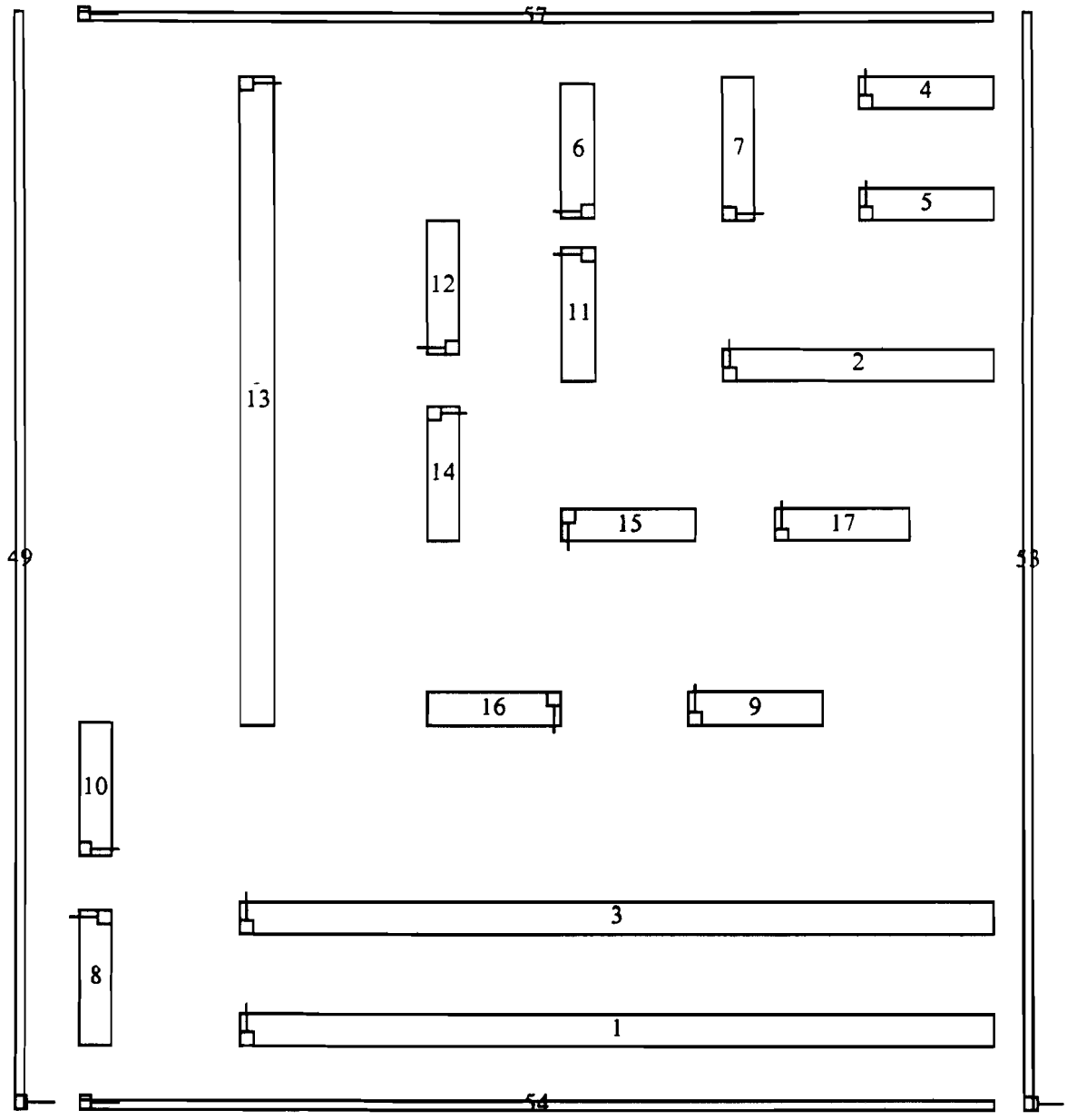
Example 3, channels (size 484 * 413).
Module orientation method 1.



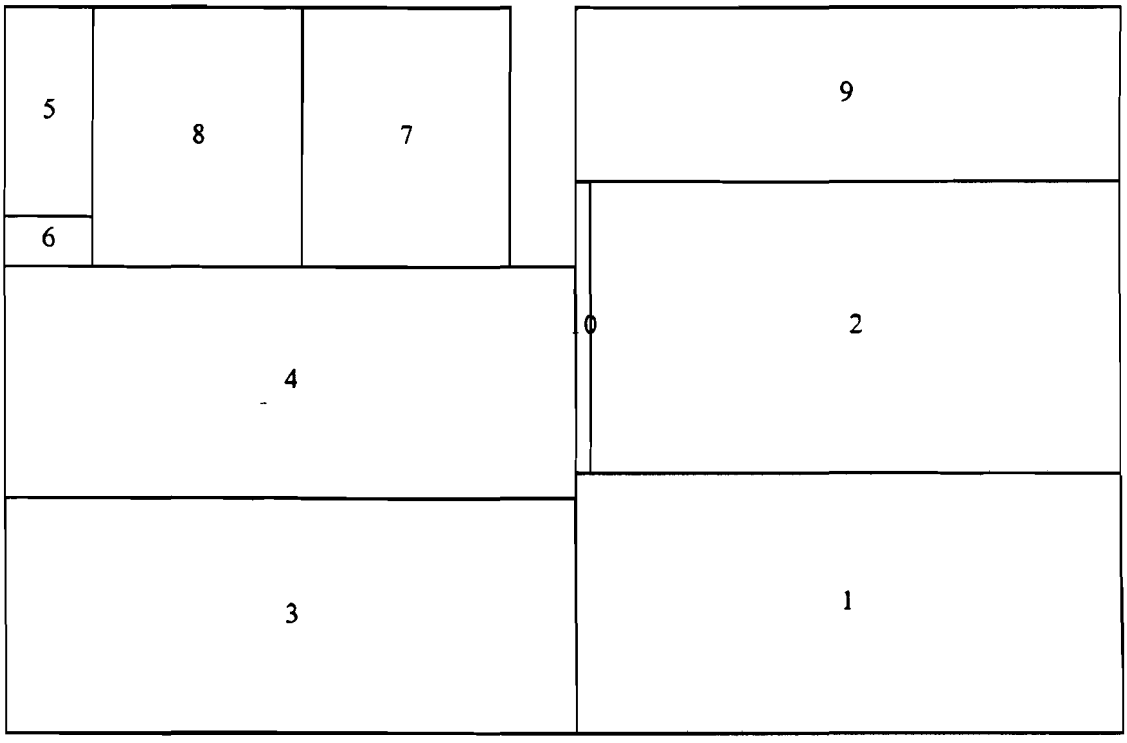
Example 4, no channels (size 224 * 244).
No module orientation.



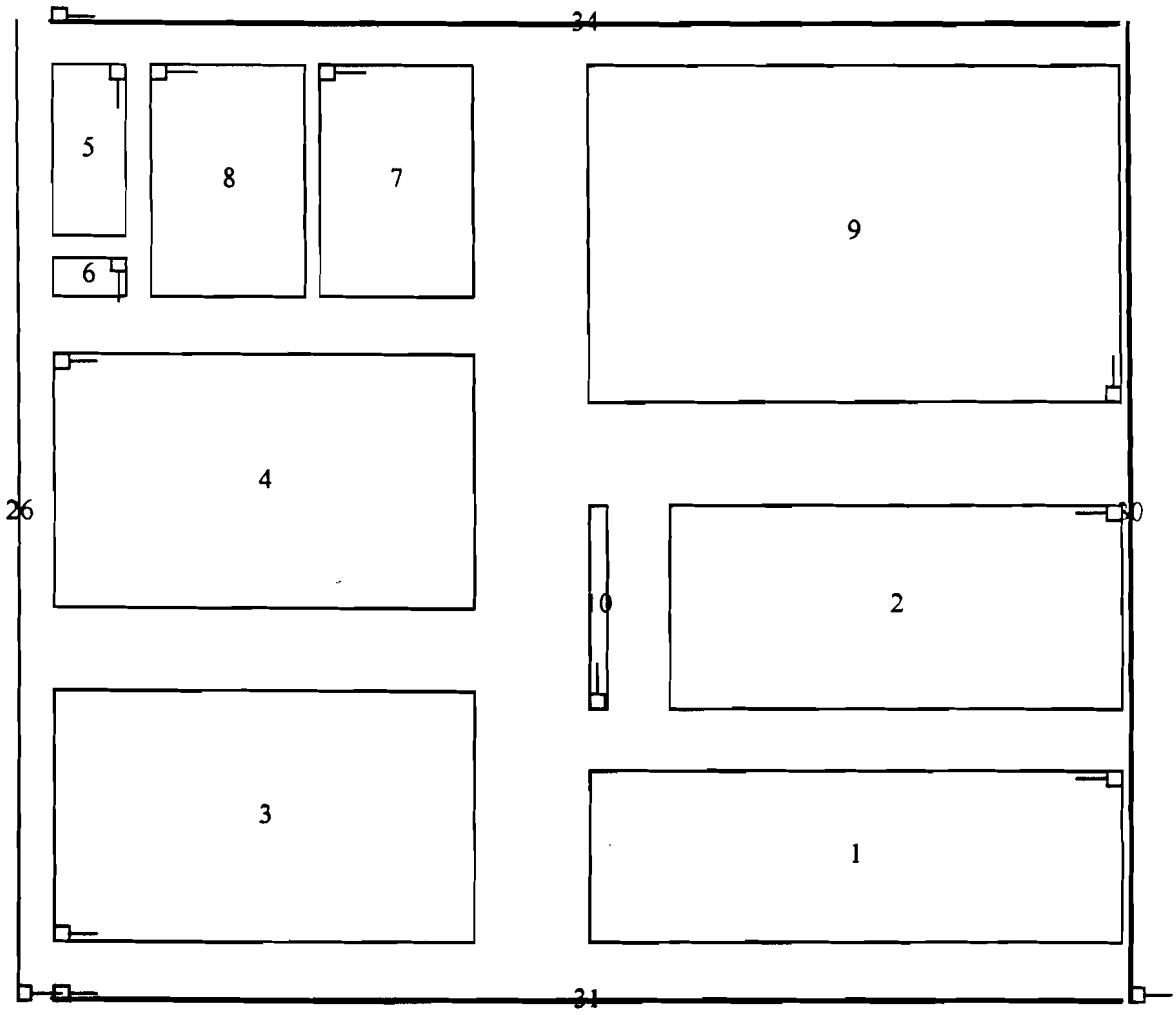
Example 4, channels (size 620 * 671).
Module orientation method 1.



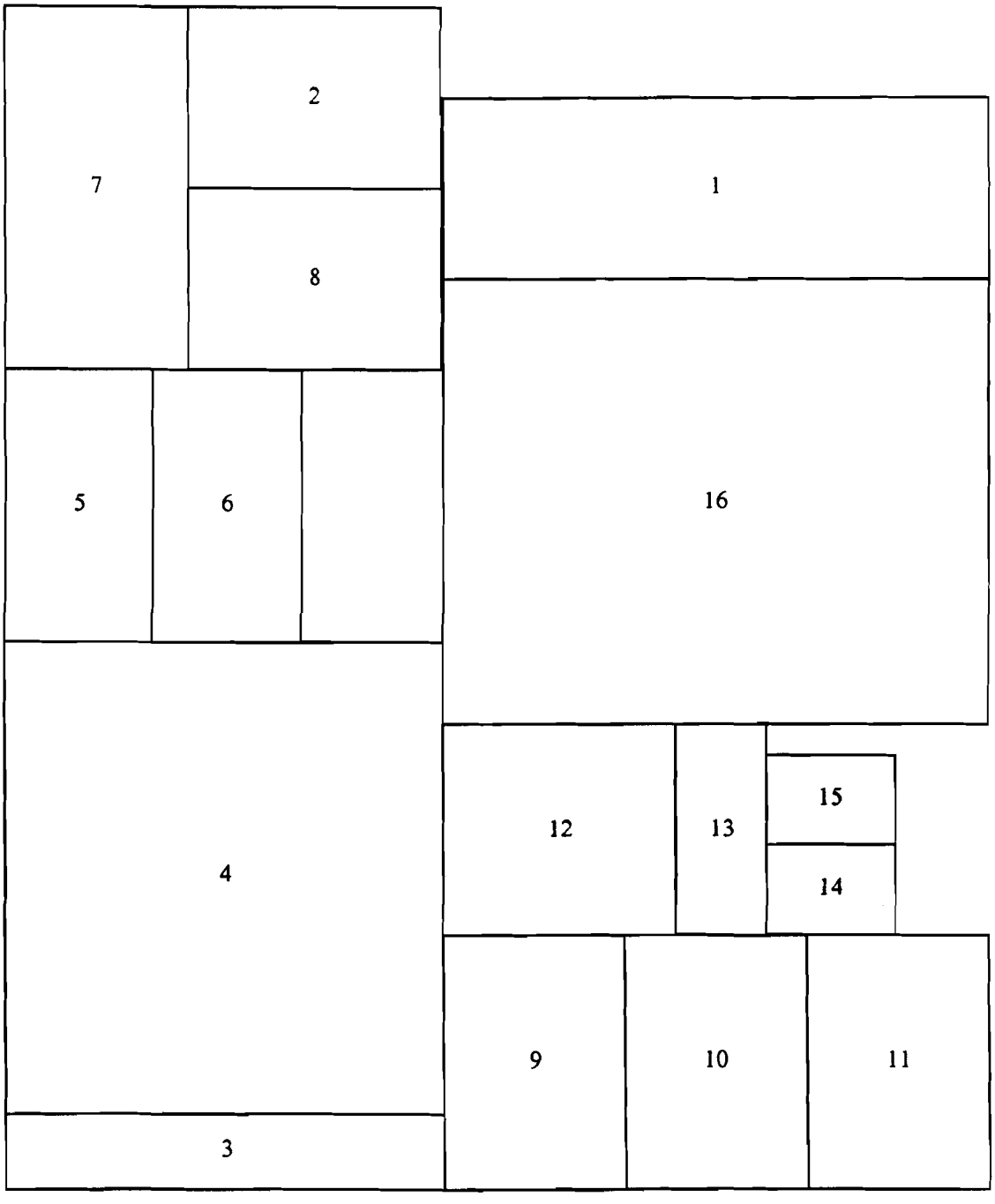
Example 5, no channels (size 2151 * 1400).
No module orientation.



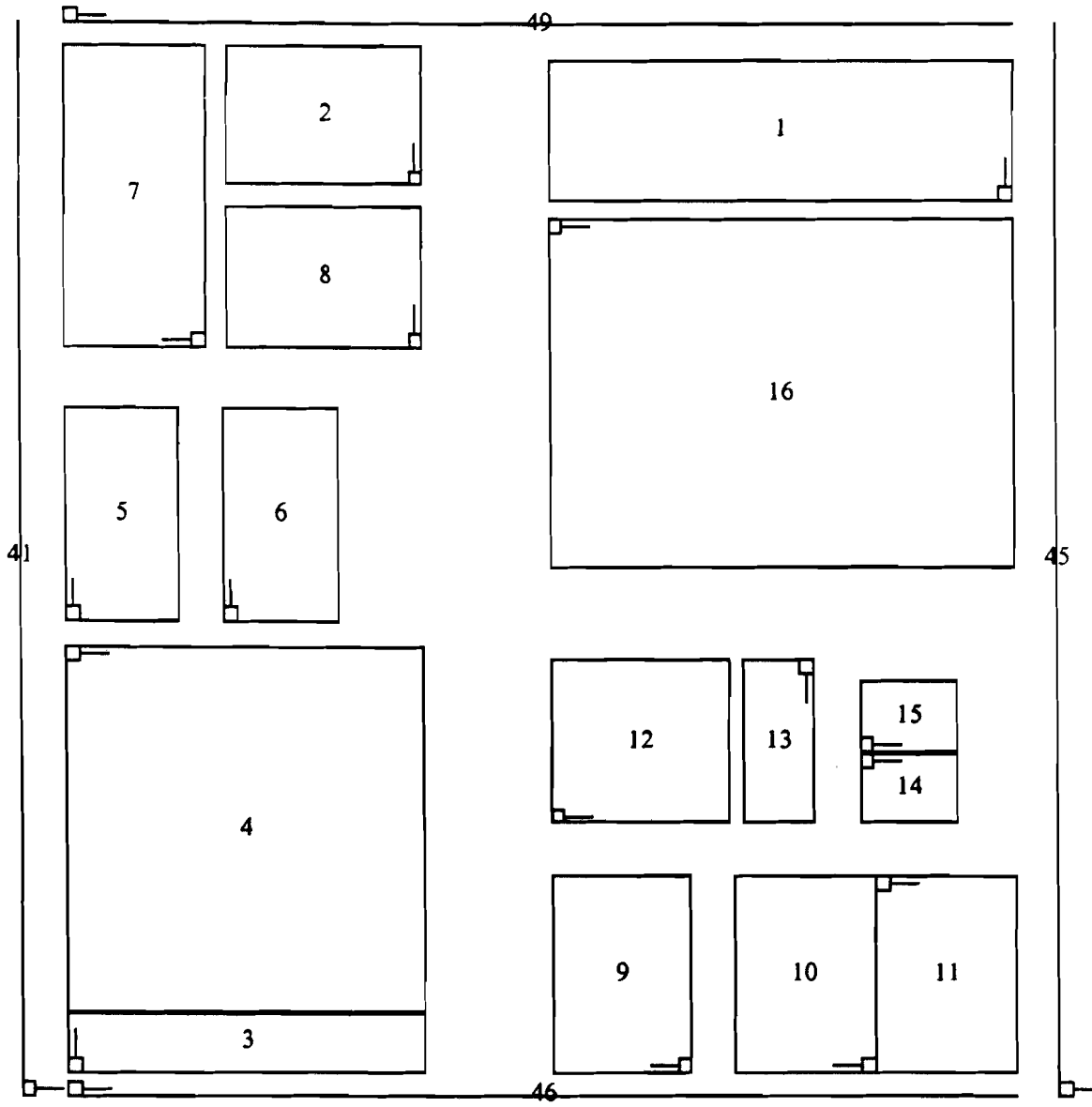
Example 5, channels (size 2622 * 2328).
Module orientation method 1.



Example 6, no channels (size 8913 * 10734).
No module orientation.



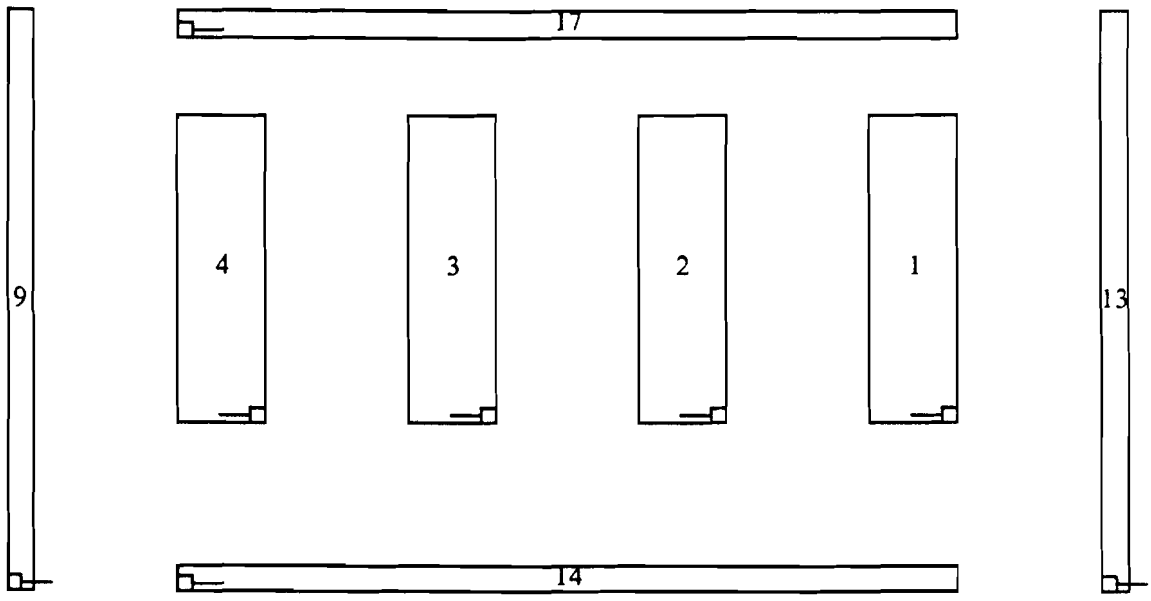
Example 6, channels (size 12180 * 12564).
Module orientation method 1.



Example 7, no channels (size 80 * 70).
No module orientation.

4	3	2	1

Example 7, channels (size 257 * 133).
Module orientation method 1.



References.

[1] Ralph H.J.M. Otten, "**Efficient Floorplan Optimization.**", Proceedings of IEEE International Conference on Computer Design, Port Chester, 1983, pp. 499 - 502.

[2] Lukas P.P.P. van Ginneken, Jos T.J. van Eijndhoven, Paul R.M. van Teeffelen and Theo J. Deckers, "**Soft Macro Cell Generation by Two Dimensional Folding.**", still to be published.

[3] B.T. Preas and W.M. van Cleemput, "**Placement Algorithms for Arbitrary Shaped Blocks.**", Proceedings of the 16th Design Automation Conference, San Diego, 1979.

[4] R. Malladi, G. Serrero and A. Verdillon, "**Automatic Placement of Rectangular Blocks with the Interconnection Channels.**", Proceedings of the 18th Design Automation Conference, 1981, pp. 419 - 425.

[5] Ralph H.J.M. Otten, "**Automatic Floorplan Design.**", Proceedings of the 19th Design Automation Conference, 1982, pp. 261 - 267.

[6] L.P.P.P. van Ginneken and J.A.G. Jess, "**Gridless Routing of General Floor Plans.**", Proceedings of the International Conference on Computer-Aided Design '87, 1987, pp. 30 - 33.