Placement for gate arrays by eigenvector decomposition

Schenning, J.H.M.

Award date:
1987

EINDHOVEN UNIVERSITY OF TECHNOLOGY
DEPARTMENT OF ELECTRICAL ENGINEERING
AUTOMATIC SYSTEM DESIGN GROUP

Placement for gate arrays by eigenvector decomposition

by J.H.M. Schenning

Master thesis
report on graduation work
performed from 15.12.86 to 22.10.87
by order of prof. dr.-ing. J.A.G. Jess
and supervised by ir. A.G.J. Slenter

Abstract

Starting from a netlist of a digital circuit the gate array system is able to generate a layout description on several types of gate arrays. Described in this paper is a placement which places the cells on the gate array. This placement is based on the eigenvector decomposition of a cost matrix. It tries to keep the total netlength short and so the routing chances high. One of the problems was to calculate promising eigenvectors, this was solved by using the method of lanczos as described in this paper. The goal of the placing algorithm is to place the cells close to the most promising eigenvectors.

CONTENTS

LIST OF FIGURES

1. Introduction

At the laboratory of the automatic system design group (ES) of the department of electronic engineering of the Eindhoven University of Technology effort is made on the construction of silicon compilers. In this environment there is a project which is concerned with the realization of digital circuits on gate arrays. The project is called GAS and is an "open" design system. The system is open with respect to the type of gate array and with respect to the tools for logic design, placement and routing.

The paper describes the newly developed placement algorithm, based on the eigenvector decomposition of a cost matrix.

2. Introduction to the gate array system used on the TUE

2.1 The gate array's

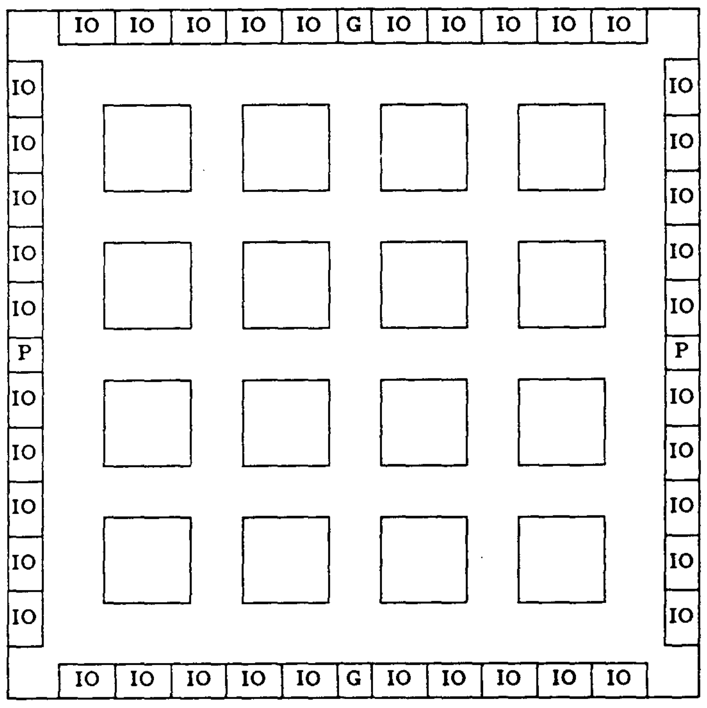
Gate arrays are used for the realization of digital systems. Their internal structure and the logic and interface components are fixed (fig 1). Only the wirering cost of the various components on the chip is subject of the design. Therefor they are known as 'semicustomized' chips. The advantages of the use of gate arrays are a quick turn around in time and the low cost of the fabrication process. This is due to the fact that only the wirering is different for every new design.

Because of the fixed structure of the gate arrays the wirering problem is fairly complex. In other technologies the channel width is adjusted according to the need, but in the gate array system the channel width is fixed. So in gate array design special reroute strategies are necessary if the number of tracks exceeds the number that is allowed.

There are many different types of gate arrays based on all kinds of logical families like I2L, Schottky TTL, static NMOS and CMOS. They all have there own so called image. The image of a gate array describes the possible locations of wires (polysilicon, silicide or metal) and the possible location of via holes connections between the various wiring layers. The image can be build up of one or more layers and different arrangements of gates (eg row or block arrangements), where a gate is the smallest logic port on that particular gate array.

2.2 The gate array system

The gate array system, of which the placement described later in this report is part of, covers all different types of gate arrays. In this system the structure of the gate array is described by a grid. The grid contains all the essential place and rout information at any stage of the



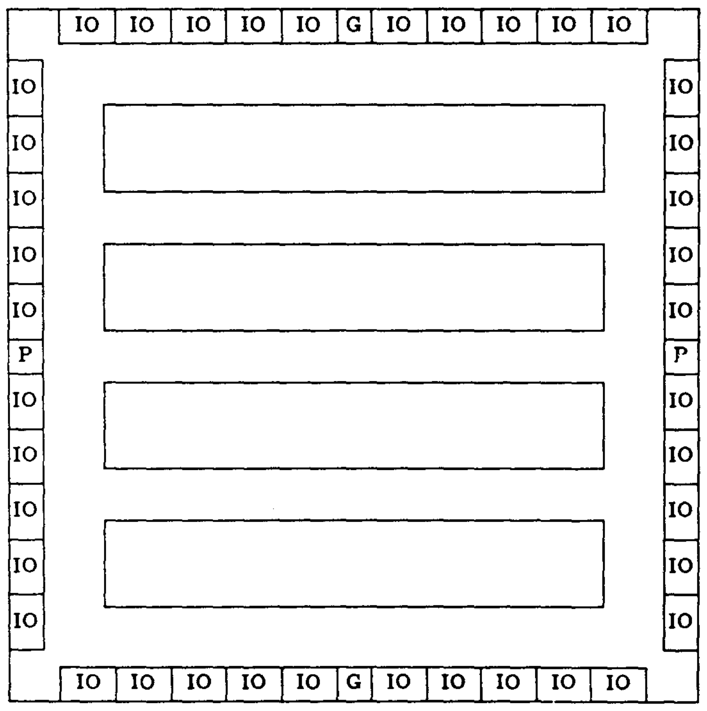


Figure 1. Example of the internal structure of a gate array

design process. If we want to fit a new type of gate array in the system, the only thing that has to be done is to describe the image, design rules, the router cost function and the macros. To do this a special language, the so called Gate Array Description Language [4], is provided. Once this is done for any type of gate array, a compiler translates the image, design rules and macro description into a grid and a macro library. Now any design can be handled by the gate array system. The great advantage of this gate array system is that there is no need to develop new placement and routing tools for a new type of gate array.

2.3 The macro library

By each type of gate array a macro library is generated. This macro library contains all the macros that can be realized on that particular gate array (e.g. inverters, nands, different types of flip-flops etc.). Each macro consists of different realizations (stamps) with different legal places, internal wirering, positions for input and output terminals etc.

2.4 The layout generation

The layout generation of a digital circuit is divided in several steps (fig 2). If it is a new type of gate array then an image description, a router cost function and a macro description must be generated with the Gate Array Description Language. Then a compiler generates a macro library, a grid description and the design rules, these last two are stored in the core library. If these descriptions are available all sort of circuits can be handled by the gate array system. For the generation of a netlist there is a netlist compiler, which can handle information given in by hand or from a schematic editor. To generate a layout from this net list, first a placer places all the modules that are in the netlist onto the grid, taken into consideration the legal places of the modules and a cost function. When all the modules are placed the nets are globally routed [5].

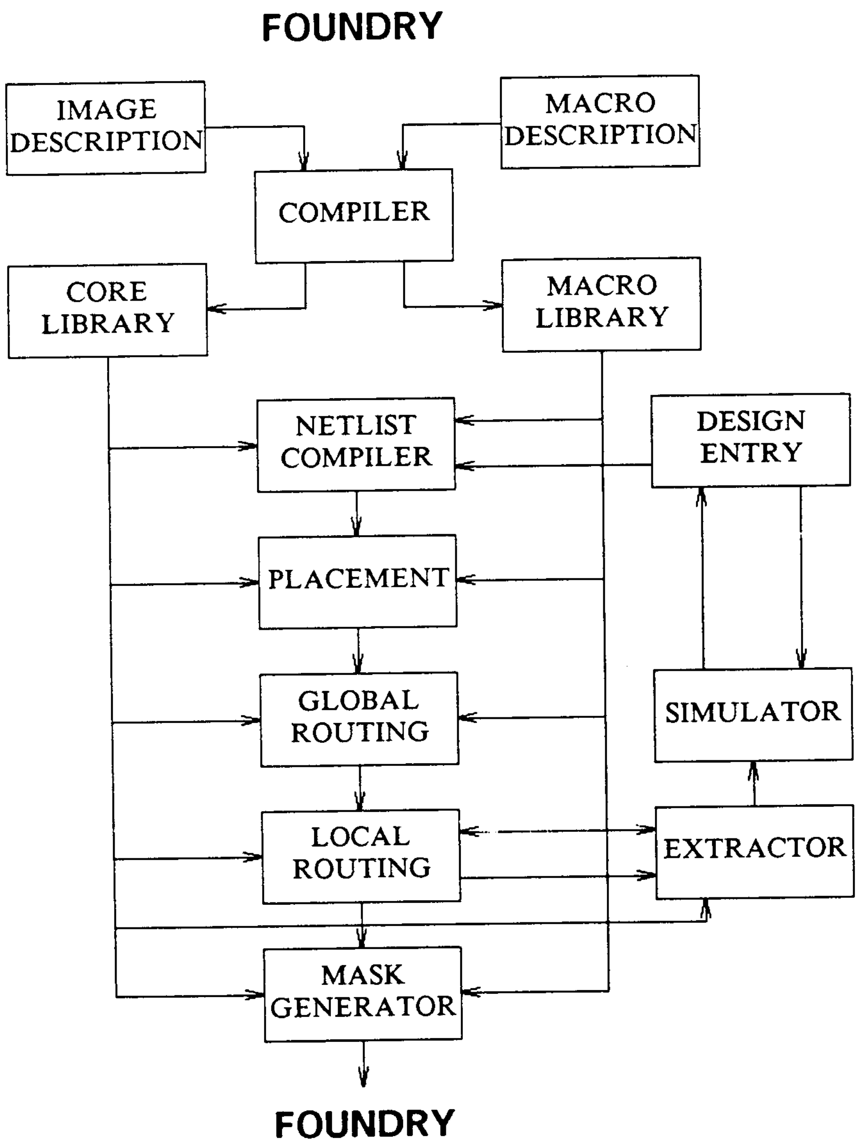


Figure 2. Overview Gate Array System

This means that the grid first is divided in a global grid with the smallest unit a grid cell of that global grid. Then the global router routes the nets hierarchical from grid cell to grid cell taken into consideration the the number of nets between these grid cells. The local router [6], then tries to rout each grid cell independently of the other grid cells given the information of the global router. The routing takes place wherever tracks are available and the modules (macros) are transparent for routing. In general not all pins of a net are connected by solving these local routing problems. To connect the remaining pins the router takes successively more grid cells in consideration until all pins are connected or the routing area is the complete chip. The extractor is able to get the information regarding critical paths. The place and rout tools are able to handle the nets in the critical path with special care.

3. Placement by eigenvector decomposition

3.1 Introduction

Another placement invoked in the gate array system is based on a simulated annealing approach [7]. Simulated annealing is based on interchanges of modules. The number of interchanges that take place at each step is an equivalent of the 'temperature' of the system. The interchanges are excepted if the cost function becomes smaller else the interchanges have a probabilistic chance that they are excepted. This probabilistic chance a function of the temperature, the lower the temperature the smaller the chance is that the interchanges are excepted. This temperature is cooled down during the process. The cost function is a measure of how hard it is to rout the modules. With simulated annealing a good placement can be obtained for long run time. It is because of this long run time that another method to place the modules is developed and that annealing is used as a 'back-up' placement. This new method is based on the eigenvector decomposition of a cost matrix as described by John Frankle and Richard M. Karp [2].

3.2 Problem definition

The input for the problem is :

- A netlist which describes which modules are connected to each other.
- A macro library. The macro library describes the macros (modules) in the terms of different realizations (stamps), where each stamp has a finite number of legal grid positions and a size.
- The size of the grid.

The constraints for the output are :

- Each module has to placed on a for that module legal position on the grid.
- There may be no overlap between the modules.
- The modules must be placed in such a way that the nets

can be routed.

3.3 The cost function

As stated before the cost function must be a measure of how hard it is to rout the modules. The cost function for the eigenvector decomposition is entirely based on keeping the total netlength as short as possible. This is done by adding to each net a certain weight w. Now a cost function can be build up where c_{ij} stands for the accumulated net weights, where both module i and module j belong to. The cost function becomes as following :

$$cost(x,y) = 1/2 \cdot \Sigma \Sigma c_{ij} \cdot \left[(x[i]-x[j])^2 + (y[i]-y[j])^2 \right]$$

Because $cost(x,y) = cost(x) + cost(y)$ the equation can be simplified by threating only the x-positions and for the y-positions will hold the same.

3.4 Eigenvectors

$cost(x) = 1/2 \cdot \Sigma \Sigma c_{ij} (x[i]-x[j])^2$, is expanded to get:

$cost(x) = -x^T Cx + \Sigma x^2[i] \cdot (i\text{-}th\ row\text{-}sum\ of\ C)$ or

$cost(x) = x^T Ax$

where the cost matrix $A = -C$ plus a diagonal matrix with entries equal to the *row-sums of C*.

Because A is symmetric it has n orthonormal eigenvectors u_r. Each eigenvector has an associated cost:

$cost(u_r) = u_r^T Au_r = \lambda_r$.

Any vector x has an unique expansion $x = \Sigma \alpha_r(x)u_r$ with coefficients $\alpha_r(x) = x^T u_r$. Since

$$cost(x) = \Sigma \alpha_r^2(x)\lambda_r , \qquad (1)$$

The problem restatement is :

Choose x to minimize $\Sigma \alpha_r^2(x)\lambda_r$, where x is a permutation of the legal positions and $\alpha_r = x^T u_r$.

The restatement allows a global approach of the placement vector, as a combination of eigenvectors that make independent contributions to solution cost. Here the observation that $cost(u_r+u_s) = cost(u_r)+cost(u_s)$, which is true because $u_r^T A u_s = 0$, is used.

The vector $(1,1,.....,1)^T$ is an eigenvector, because every row of A sums to 0. For the placing this vector is of no use, because it means that all the modules have to be placed on the same place. To make that the eigenvector (1,1,....,1) has no contribution to the placement, the legal positions are centered ($sum\ x[i] = 0$). Scaling so that $\Sigma\ x^2[i] = 1$ insures that $\Sigma\ \alpha_r^2\ (x) = 1$ for every placement.

Equation (1) shows that the cost of a vector is the weighted average of its constituent eigenvector costs λ_r. The bounds $\lambda_1 \le cost(x) \le \lambda_{n-1}$ follow. Thus if the cost of a heuristic approaches λ_1, a proof of near-optimality is immediate. Simply setting $x = u_1$ would be optimal, but the components of u_1 are unlikely to coincide with the x legal input positions required.

3.5 Transformation to the furthest away problem

For any H, minimizing $cost(x) = \Sigma\ \alpha_r^2(x)\lambda_r$ is equivalent to maximizing

$$H - cost(x) = \Sigma\ \alpha_r^2(x)(H-\lambda_r)$$

Pick $H > Max(\lambda_r)$ and define matrix V whose columns are the eigenvectors $v_r = u_r\ \mathrm{sqrt}(H-\lambda_r)$, scaled so that those with the least cost get the greatest length. Then the above expression to be maximized equals $\Sigma(x^T v_r)^2$, which is written as $|x^T V|^2$. By representing placements x as points $x^T V$ (with coordinates $x^T v_r$), the problem becomes a search for the furthest point from 0.

3.6 Probes for good points

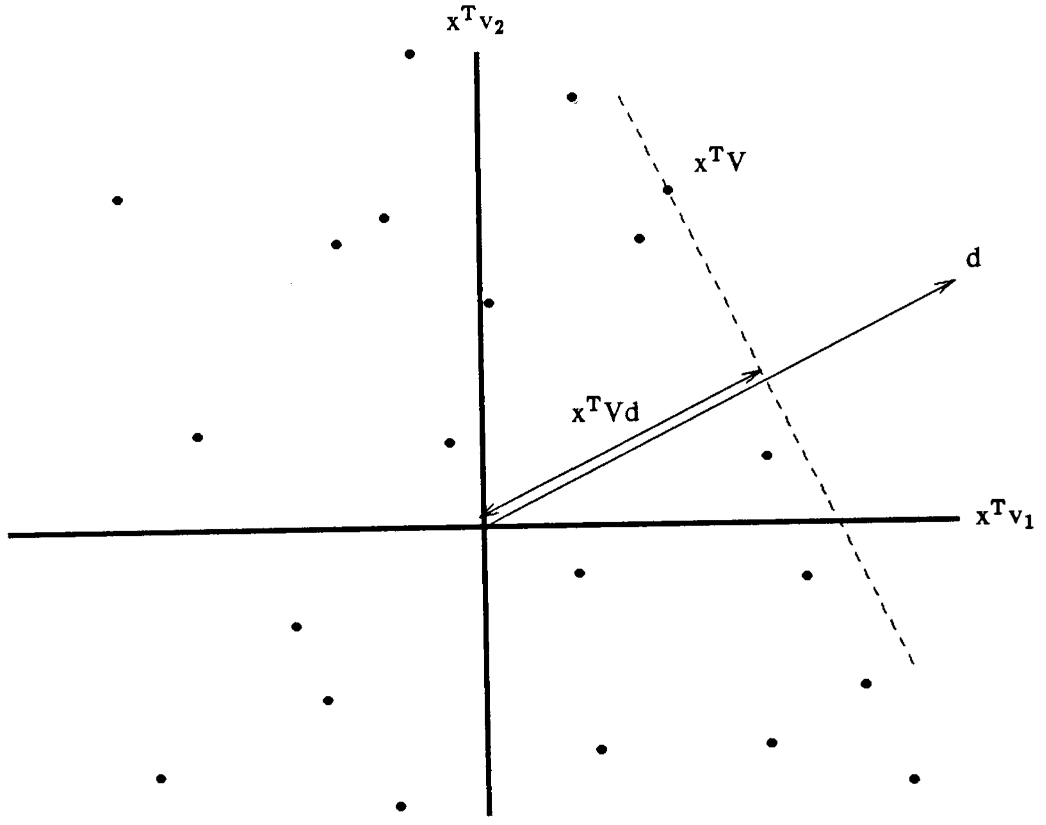
Now $H\text{-}cost(x)$ can be evaluated as $|x^T V|^2$. The payoff comes for any given direction $d \epsilon R^{n-1}$ (a "probe") a valuable x, one that maximizes $x^T V d$, can efficiently be produced. The

diagram (fig 3) illustrates the idea for the special case in which only d_1 and d_2 are nonzero. Each dot plots the first two components of a point $x^T V$ corresponding to some legal placement x.
Every probe yields both a low-cost placement and a proof that no point in the entire set of solutions has a greater projection in the probe direction.

To perform a probe, first Vd is computed, whose n elements $(Vd)[i]$ are the projections of the rows $V[i]$ onto the probe. The objective is then to compute the permutation of x positions that maximizes $x^T(Vd)$.
Probe directions with only a few nonzero components have several advantages in practice. If k components that correspond to promising eigenvectors (e.g., those with the lowest λ_r (is greatest $(H\text{-}\lambda_r)$) are selected then the range of possible contributions to $H\text{-}cost(x)$ from a particular set of eigenvectors can be explored by performing many probes. This artifice of constraining the probe directions effects a temporary reduction of the original search problem in R^{n-1} to a more manageable one in R^k. Each dimension in R^k may be identified with an "active" column-vector in V.

3.7 Placement by iterated probes

The problem is to identify particular good probe directions. This is done by taking the new probe direction towards the maximum-projection point discovered by the previous one. This makes sure that the contribution to $|x^T V|^2$ from the k active dimensions (in theory) never decreases. However in practise this is not the case, because the maximum-projection point is not found, but an approximation of it. As long as better results ($x^T V$ increases) are obtained it is useful to perform more iterations at the same stage. Later stages can use progressively more dimensions, with new eigenvectors entering in order of increasing λ_r (is decreasing $H\text{-}\lambda_r$). The current x determines the next probe: the $r\text{-}th$ coordinate of the new probe direction is $x^T v_r$ if the



The points are projections from the legal x vectors onto the sub space (v_1,v_2).

d is the probe direction.

x^TV is the maximum projection point. This maximum projection point is the next probe direction.

Figure 3. Probe direction

r-th vector is active 0 otherwise. At the beginning of each new stage, the probe is simply extended with the components of the newly activated eigenvectors.

3.8 Probes and (x,y) positions

The two dimensional problem can be stated in the following way:
Maximize $|x^T V|^2 + |y^T V|^2$, where the vectors x,y must be obtained by a permutation of the legal (x,y) positions.

The x and y vectors have to be handled jointly, because the legal positions in the gate array situation depend on each other. The analogue of a probe direction is a pair d,e, and the basic operation is to maximize $x^T Vd + y^T Ve$ over all possible placements (x,y).

By defining the vectors $a = Vd$ and $b = Ve$, the function of the probe can be concisely summarized :

Arrange the legal positions into vectors x,y so that

$\Sigma\ (x[i] \cdot a[i] + y[i] \cdot b[i])$ is maximized, or

$\Sigma\ (x[i] - a[i])^2 + (y[i] - b[i])^2$ is minimized

3.9 Net weighting

In the representation of a circuit as a connection matrix C, each net with s modules adds weight $w(s)$ to c_{ij} for every pair of its modules i,j ("the clique model"). If $w(s) = 1$ for all s, nets with many modules receive disproportionately large weights. Therefor other net weighting functions are considered.

definition st(i) = the minimum spanning tree of net i
definition n(i) = number of modules in net i.

The ideal situation is that if $\frac{st(i)}{n(i)-1} = \frac{st(j)}{n(j)-1}$ that then the contributions to the total cost from net i and net j are the same.

Assume that all the modules from the same net are lying in a

row on a equal distance d ($\frac{st(i)}{n(i)-1} = d$), then the net weights follows:
The contribution to the total cost for a net with s modules is :

$$1/2 \cdot w(s) \sum_{i=0}^{s-1} \sum_{j=0}^{s-1} (d \cdot (i-j))^2$$

Because the modules of a net do not lie in a row this is an over estimation. Assume that the modules are equally scattered in the x-direction and the y-direction then $((d \cdot (i-j))^2)$ could be roughly estimated by $(d^2 \cdot |i-j|)$. This would lead to the following formulas:

$$1/2 \cdot w(s) \sum_{i=0}^{s-1} \sum_{j=0}^{s-1} (d^2 \cdot |i-j|) =$$

$$d^2 \cdot w(s) \cdot \sum_{i=0}^{s-2} \sum_{j=i+1}^{s-1} (j-i) =$$

$$d^2 \cdot w(s) \cdot \sum_{i=1}^{s-1} ((s-i) \cdot i)$$

In the ideal situation the net cost is the same for all the nets if the distance d is the same. This leads to the following net weighting function $w(s)$.

$$w(s) = \frac{constant}{\sum_{i=1}^{s-1} ((s-i) \cdot i)}$$

In the literature [2] there are other net weighting functions proposed:

$$w(s) = \frac{1}{s-1} ,$$

$$w(s) = \frac{2}{s} ,$$

$$w(s) = \left(\frac{2}{s} \right)^{\frac{3}{2}}$$

Because the whole idea behind the eigenvector method merely depends a great deal on the net weighting function, further investigation is needed to find out which net weighting function is best suited in practical situations.

4. Probes and legal placements in gate arrays.

4.1 Introduction

In this chapter two different transformations are discussed.

- Transforming a legal placement into a probe direction.
- Transforming a probe direction into a legal placement.

The theory in chapter 3 describes this problems only for normalized vectors. Because the legal placement vector is not a normalized vector some particular problems arise.

4.2 From a legal placement to a probe.

A legal placement in the gate array system consists of a list of modules with their coordinates on the grid and a reference to the stamp which realizes this module. The modules may not have any overlap. From this information a placement vector with $\Sigma\, x_i = 0$ and $\Sigma\, x_i^2 = 1$ must be made. To do this the middle of each module is taken as the coordinates of the x vector. Then the following steps are taken :

$$offset = \frac{\Sigma\, x_i}{nr\_of\_modules}$$

$$x_i = x_i - offset \qquad \text{,centering } x_i$$

$$factor = sqrt(\Sigma\, x_i^2)$$

$$x_i = \frac{x_i}{factor} \qquad \text{,normalizing } x_i$$

Now the probe direction can be calculated as in chapter 3 with :

$$p_j = \sum_i x_i \cdot v_{ji}$$

where v_{ji} is the i_{th} element of eigenvector v_j, with eigenvector v_j belonging to the set of active eigenvectors.

For the *y* vector holds the same.

4.3 From a probe to a legal placement

If a probe direction is present, a normalized placement vector x can be calculated easily according to the theory presented in chapter 3 with :

$$x_i = \sum_j v_{ji} \cdot p_j$$

This normalized x vector is first converted to a global vector lying on the placement grid, with the following steps :

$$factor = \frac{max\ occupied\ x\ -\ min\ occupied\ x}{x\_max\ -\ x\_min}$$

$$x_i = x_i \cdot factor$$

$$offset = \frac{max\ occupied\ x\ -\ min\ occupied\ x}{2}$$

$$x_i = x_i\ -\ offset$$

max_occupied_x is the maximum occupied position on the grid

min_occupied_x is the minimum occupied position on the grid

With above steps a global x and *y* vector which lie between the min_occupied and max_occupied x and y boundaries is formed.

The legal placement must minimize :

$$\Sigma\ (legal\_x_i\ -\ x_i)^2 + (legal\_y_i\ -\ y_i)^2$$

The minimum of above formula is approximated by taking each module individual, and find the best legal place, taking in consideration the already placed modules (no overlap may occur), so that :

$$(legal\_x_i\ -\ x_i)^2 + (legal\_y_i\ -\ y_i)^2$$

is minimized. The reason to approximate the minimum is that this has to be done several times and to find the absolute minimum would take to much time. The only freedom is the order in which the modules are placed. To have better prospects for latter modules to be placed close to there optimal position, the greatest modules are chosen to be placed first. Another strategy could be to take the modules with the greatest contribution to solutions cost first.

Another problem that arises with this approach is that the normalized x and y vector tend to be dependent of each other. The scalar product $(x \cdot y)$ is not zero, but becomes greater after each step because both vectors are build up with a preference for the best eigenvectors. The global x and y vector are more and more growing to each other and the modules cluster around some diagonal of the gate array. To overcome this problem the x and y vector are orthogonalized to each other. This means that $\Sigma\ (x_i \cdot y_i)$ must be made zero. The x and y vector should be adjusted minimaly and each vector with the same proportions. To do this the following formulas are used :

$$(x \cdot x) = \Sigma\ x_i \cdot x_i$$

$$(y \cdot y) = \Sigma\ y_i \cdot y_i$$

$$(x \cdot y) = \Sigma\ x_i \cdot y_i$$

$$\hat{x} = \frac{x}{sqrt(x \cdot x)} + a \cdot \frac{y}{sqrt(y \cdot y)}$$

$$\hat{y} = \frac{y}{sqrt(y \cdot y)} + a \cdot \frac{x}{sqrt(x \cdot x)}$$

We want that $\Sigma\ \hat{x}_i \cdot \hat{y}_i$ becomes 0 so :

$$\Sigma\ \hat{x}_i \cdot \hat{y}_i = \frac{(x \cdot y)}{sqrt(\ (x \cdot x) \cdot (y \cdot y)\ )} \cdot (1+a^2) + 2 \cdot a = 0$$

chose the a which is the smallest :

$$a = \frac{-1 + sqrt(1 - cor\_xy \cdot cor\_xy)}{cor\_xy}$$

with $cor\_xy = \frac{(x \cdot y)}{sqrt(\ (x \cdot x) \cdot (y \cdot y)\ )}$

Remark :

If $cor\_xy \ll 1$ then $a = -\frac{1}{2} \cdot cor\_xy$

By making $x = \hat{x}$ and $y = \hat{y}$, the normalized x and y vector are orthogonal to each other.

5. The eigenvalue problem

5.1 Introduction

In the previous chapters a placement is described which is totally based on the eigenvectors of a symmetric matrix A. The problem is to calculate 'promising' eigenvectors. In this chapter a method is presented to calculate this eigenvectors in a fast way, making use of the fact that A is sparse and that not all the eigenvectors are needed. For a more detailed description of the theory behind the eigenvalue problem, the book from Parlett (The Symmetric Eigenvalue Problem) [3] is recommended.

5.2 Tridiagonalization of a matrix by Lanczos

definition T is *tridiagonal* if $t_{ij} = 0$ whenever $|i-j|>1$.

A tridiagonal matrix T and a transformation matrix Q are formed, with Q orthogonal, from the symmetric matrix A. So that

$$T = Q^T AQ$$

proof :
The transformation $T = Q^T AQ$. Can be rewritten as

$$QT = AQ \qquad , \textit{since } Q^T = Q^{-1}.$$

If the *j-th* column on each side is equated, and the terms are rearranged, then the important relation is found.

$$q_{j+1}\beta_j = Aq_j - q_j\alpha_j - q_{j-1}\beta_{j-1} \equiv r_j$$

By defining $\beta_0 = \beta_n = 0$ this will hold for $j = 1,\ldots,n$. Thus r_n and q_0, q_{n+1} are undefined. Next the orthogonality of Q is used to obtain.

1. $0 = q_j^T(q_{j+1}\beta_j) = q_j^T Aq_j - 1\cdot\alpha_j - 0\cdot\beta_{j-1}$,
2. $\beta_j = |q_{j+1}\beta_j| = |r_j|$,
3. $q_{j+1} = r_j/\beta_j$, $\beta_j > 0$ by the previous equation

From the equations above the 'simple' Lanczos algorithm is derived.

r_0 is given, $\beta_0 = |r_0| \neq 0$. For $j = 1,2,\ldots n$ repeat

1. $q_j = r_{j-1}/\beta_{j-1}$
2. $u_j = Aq_j$
3. $r_j = u_{j-1}\beta_{j-1}$ $(q_0 = 0)$
4. $\alpha_j = q_j^T r_j$
5. $r_j = r_j\alpha_j$
6. $\beta_j = |r_j|$

Remark 1 :
If β_j becomes 0 at step 6, then we chose at step 6 a new r_j which is orthogonal to all the previous found r_j's. The fact that $\beta_j = 0$ means that the chosen r_0 was orthogonal to some eigenvectors of A.

Remark 2 :
Because there is a subspace $Q_j = (q_1 \cdots q_j)$ build, by multiplying A with a random vector, convergence will occur for the outher eigenvalues of A [3] This feature will be used to find the most 'promising' eigenvectors of A without calculating all eigenvectors.

Remark 3 :
To keep all found q's orthogonal to each other it is necessary to perform a full orthogonalization after step 5.

5.3 Eigenvalue and eigenvector estimation from a Tridiagonal matrix T

This can be done with the "QL" algorithm.

$$T-\sigma = QL$$

$$T_1 \equiv LQ+\sigma = Q^T(T-\sigma)Q+\sigma = Q^T TQ$$

σ is the origin shift. L is lower triangular (L_{ij} is 0 for

$j > i$) $Q^T = Q^{-1}$

The tridiagonal form is preserved, because the shape of Q and L simply preserve the zero elements above the diagonal. The elements below the diagonal must be zero through cancellation, since T_1 is like T symmetric. See also fig 4. QL (with $T_0 = T$) is as follows:

$$T_{k+1} = Q_k^T T_k Q_k$$

$$T_k - \sigma = Q_k L_k$$

If the eigenvectors are not needed, but only the eigenvalues then this calculation could be done in approximately 1.7 QL transforms [3,p. 162] per eigenvalue when Wilkinson's shift is used.

Wilkinson's shift : if $\alpha_1 = \alpha_2$ then $\sigma = \alpha_1 - |\beta_1|$

$$\text{else } \sigma = \alpha_1 - \frac{sign(\delta)\beta_1^2}{\left(|\delta| + sqrt(\delta^2 + \beta_1^2)\right)}$$

$$\text{with } \delta = (\alpha_2 - \alpha_1)/2$$

The QL algorithm using Wilkingson's shift will always converge in exact arithmetic. The eigenvalues will appear on the diagonal of T_k for $k \to \infty$ And the eigenvectors can be calculated as follows :

$$P_k = Q_1 \cdot Q_2 \cdot \cdots \cdot Q_k$$

And now each column of P contains an eigenvector of T if $k \to \infty$. To perform a QL transformations Givens transformation is used [3].

5.4 Accuracy of eigenvalues and eigenvectors

A good measure for the accuracy, of the eigenvectors u and eigenvalues λ, is the residual norm $|Au - u\lambda|$. Fortunately this norm can be computed without computing u. After j steps in the Lanczos algorithm :

$$|Au - u\lambda| = |AQs - Qs\lambda| \qquad , \text{ since } u = Qs$$

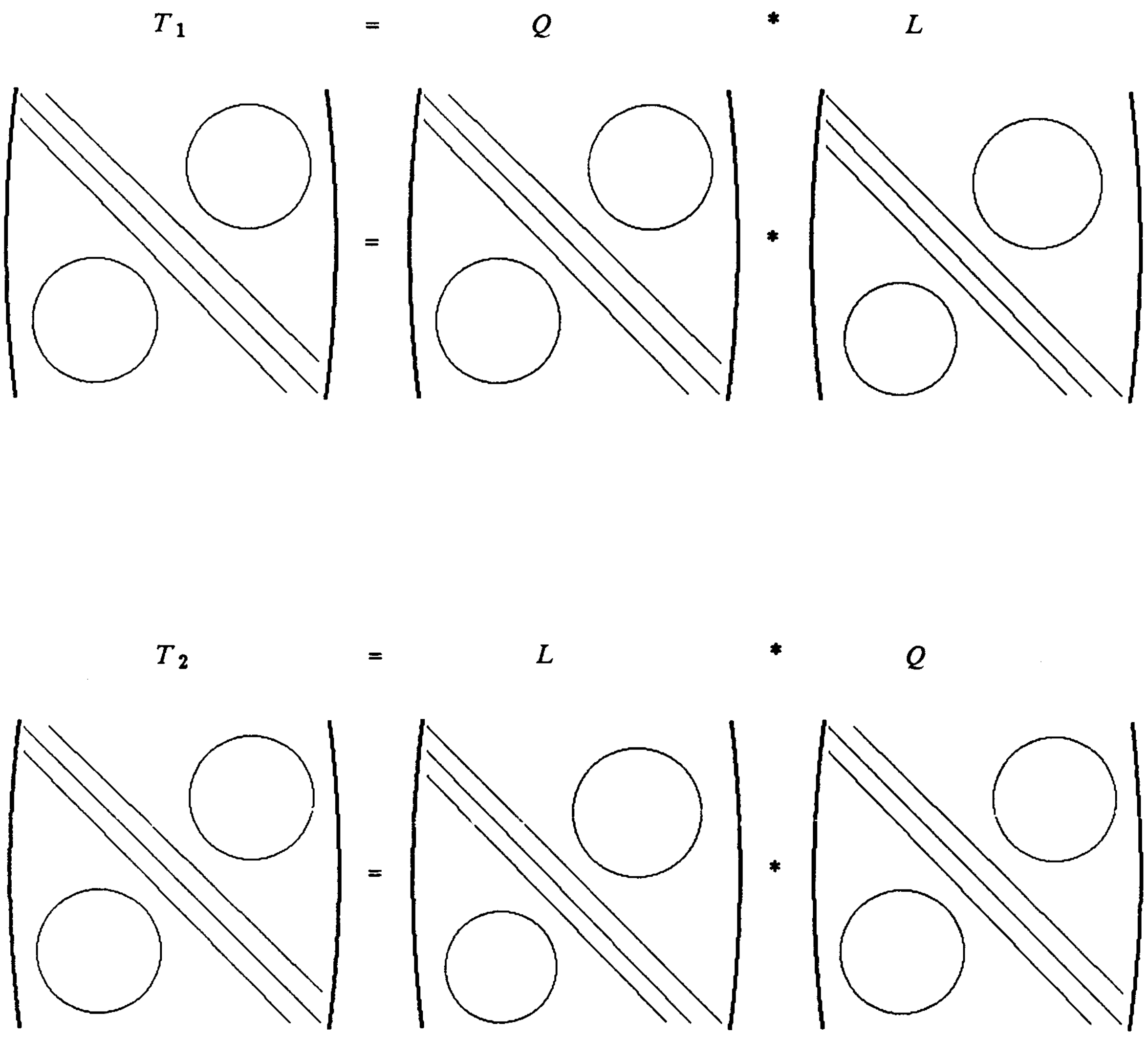


Figure 4. Preservation of the tridiagonal form

$= |(AQ-QT)s|$, since $s\lambda = Ts$

$= |(r_j e_j^T)s|$, since $AQ_j - Q_j T_j = r_j e_j^T$

$= \beta_j |e_j^T s|$, since $|r_j| = \beta_j$

So the bottom elements of the normalized eigenvectors (s) of T_j signal convergence and there is no need to form u until its accuracy is satisfactory.

6. Short overview of the program and the data structures

6.1 Used data structure

The main data structures that are used in the program can be divided in two different types.
First there are the data structures generated by the routines to read the net list and the macro library.
Second there are the data structures that are constructed from the first type.

There are two different data structures of the first type:
One generated from the net list (fig 5) consists of a net list. Each net in the net list contains a pointer to a list which contains the modules connected to that net. The modules are also contained in a list. From the structure drawn in fig 5 a sparse matrix (fig 6) is constructed, this sparse matrix is the cost matrix mentioned in paragraph 2.3.
An other data structure is generated from the macro library (fig 7). Because not all the information given by the structure off the macro library is needed, a new structure (fig 8) is made. There are two main differences between the old structure and the new structure, the new structure holds no information over terminals and wires, and the new structure has a new variable called gridlegals. These gridlegals contain in a compact form the legal grid positions of a macro or stamp.

6.2 Program description

The program can be roughly diveded in the following steps.

```
read_macro_library;
read_netlist;
read_size_gatearray;
define_gridlegals;
make_gridlegals;
calculate_net_weights;
make_Sparse_cost_matrix;
```

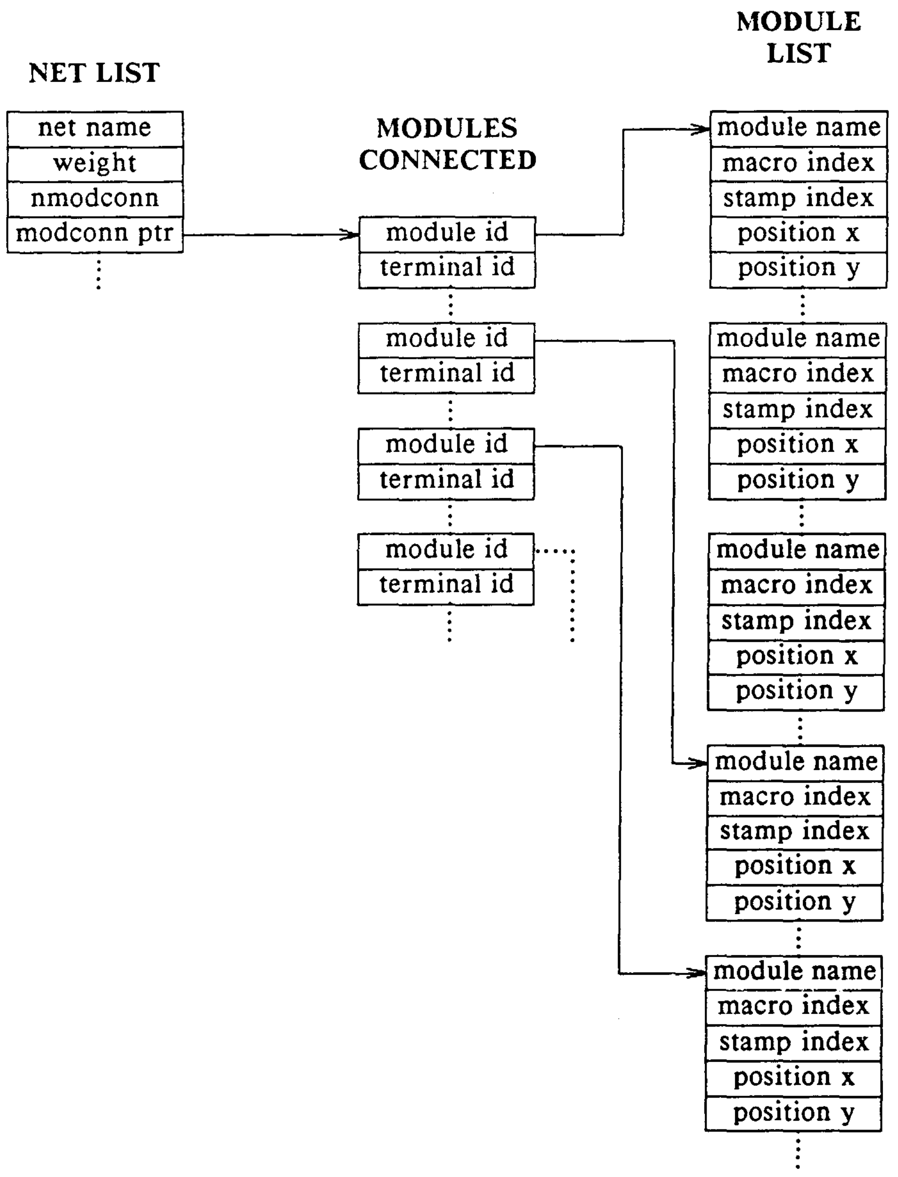


Figure 5. Structure net list

```
    calculate_promising_eigenvectors;
    evaluate_placement;
    write_netlist;
```

read_macro_lib :
Read the macros from the macro library of this type of gate array.

read_net_list :
Reads the net list which represents the circuit description.

read_size_gatearray :
Reads the size of the gate array. The X_SIZE, Y_SIZE and the NLAYERS (number of layers) is read.

define_gridlegals :
Determines all the possible X positions and Y positions that may lead to legal positions for the modules contained in the net list. Also generated is a new grid called PL_GRID which consists of as many legal X and Y positions then there are legal X and Y positions.

make_macro_gridlegals :
Defines on which grid positions each macro and stamp may lie on PL_GRID.

calculate_net_weights :
Calculates the weights of the nets taken into account the amount of modules of the net and the user specified importance of the net.

$$net\_weight = i(n) \cdot \frac{constant}{\sum_{i=1}^{s-1} ((s-i) \cdot i)}$$

where $i(n)$ is the user specified importance of the net, and s the number of modules in the net.

make_Sparse_cost_matrix :

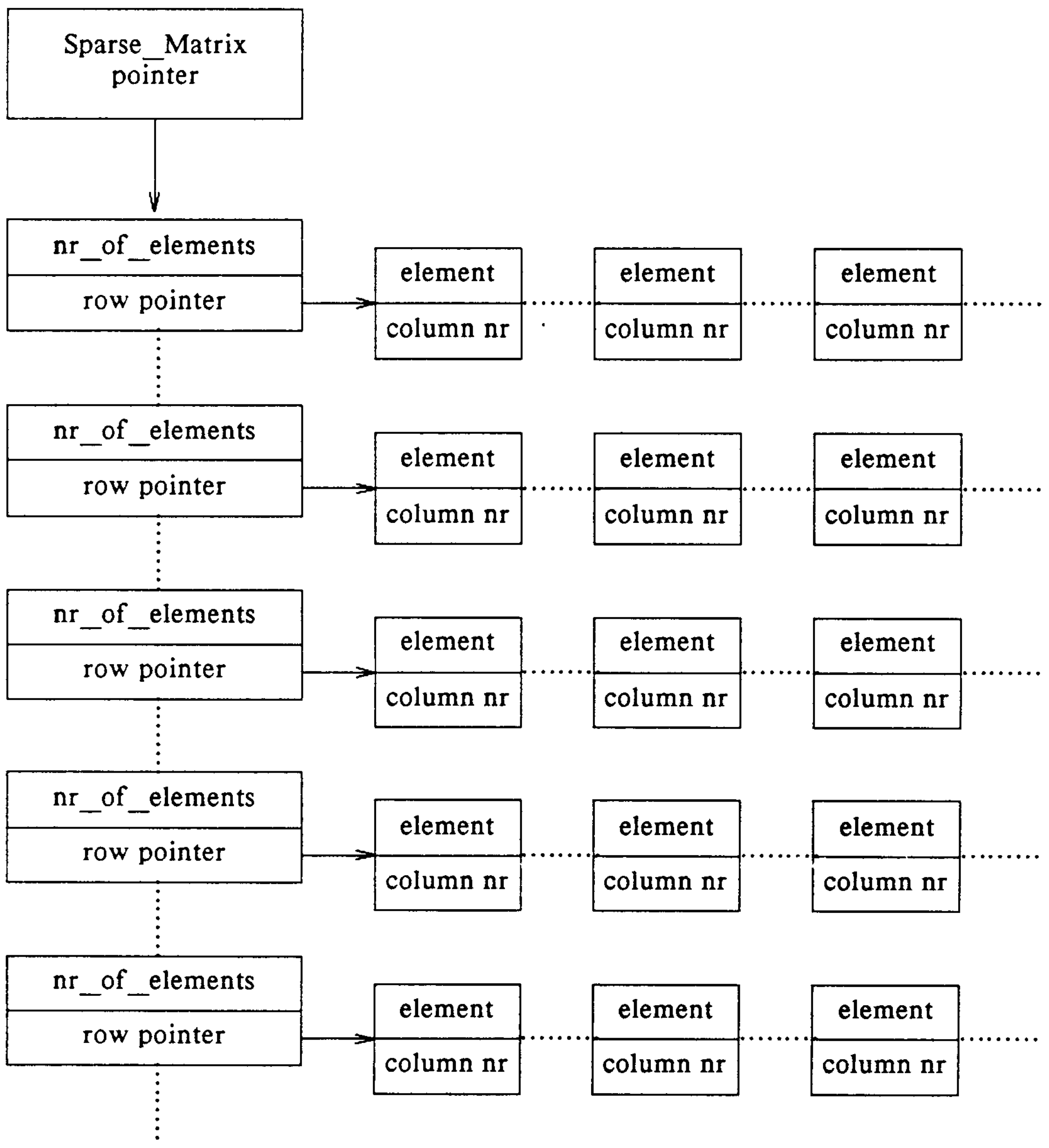


Figure 6. Structure Sparse Matrix

First a NxN cost matrix is made as described in chapter 2 (N is the number of modules). Because we need the matrix only if we multiply the matrix by a vector we made the matrix sparse as shown in fig 6. The sparse matrix contains only the nonzero components.

calculate_promising_eigenvectors :
To calculate the most promising eigenvectors the matrix must be adjusted in the right way, first the eigenvector with the greatest lambda (H) is calculated. Then the sparse matrix S undergoes a transformation $S = (H \cdot I - S)$. Now the most promising eigenvectors consists of the eigenvectors with the greatest lambdas with this transformed matrix.

evaluate_placement :
This routine tries to find a low cost placement with the help of the calculated eigenvectors. It can be divided in an initialization step and a repetition.

```
   make_random_global_placement;
   find_legal_placement;
   nr_of_probes = sqrt( nr_of_modules );
   do {
      do {              find_new_probe_direction;
find_legal_placement;
      }
      while (improvement);
      nr_of_probes = nr_of_probes * 2;
   }
   while ( nr_of_probes < ( 2 * nr_of_eigenvectors ) );
```

This repetition leads to a local convergence, so if we repeat above routines with another random global placement then another local convergence arises which can be better or worse then the first one. To find the global minimum one would have to repeat this steps infinitely. Because the aim of this program is a good placement in a fast running time these steps are only taken once.

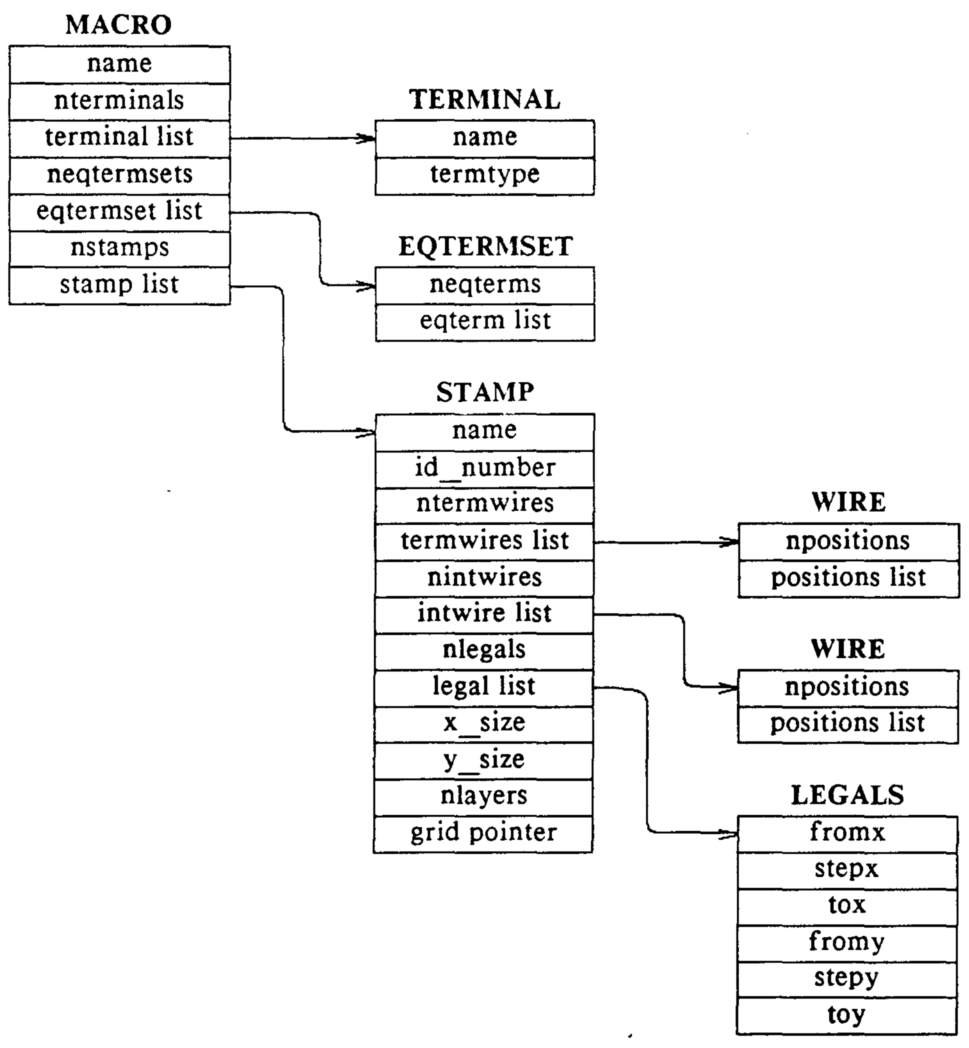


Figure 7. Structure macro library

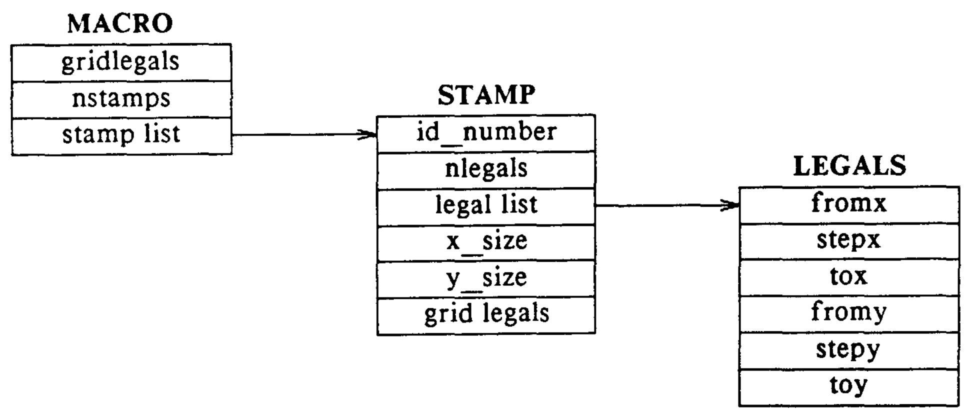


Figure 8. Data needed from the library

find_new_probe_direction :

Before we can calculate the new probe direction we must determine the x and y vector of the best found placement so far. The mid of every module is taken as the real position of the module, and we make that $\sum_i x_i = 0$. Now we can calculate for every eigenvector we want to use $probe_j = \sum_i x_i \cdot v_{ji}$. The new x vectors is then calculated as follows :

$$x_i = \sum_j v_{ji} \cdot probe_j \cdot \lambda_j$$

λ_j is the eigenvalue of the eigenvector from the matrix $(H \cdot I - S)$. For this x vector $\sum_i x_i$ is still 0 and the maximum x_i and minimum x_i are not the maximum and minimum of the possible grid coordinates. To map the x vector onto the grid we must multiply the x vector by a factor and then add an offset to make sure that,

$$\frac{\sum x_i}{nr\_of\_modules} = mid\ of\ the\ grid$$

find_legal_placement :

We want that the legal placement vectors x and y

maximizes : $\sum (x_i \cdot a_i) + (y_i \cdot b_i)$

minimizes : $\sum (x_i - a_i)^2 + (y_i - b_i)^2$

(a_i, b_i) is the global place of module i

To accomplish this we first order the modules on their sizes, greatest module first. Then we minimize the second formula for each module separately. By taking the greatest module first there is more change that the last modules will fit nearby their global place, because they are small.

write_netlist :

Writes away the best resulting placement into a file.

7. Conclusions

- A program which handles the placement of modules on gate arrays with the eigenvector method [2], is succesfully developed. The practical tests justify the choice of the new netweighting function above the netweighting funtions given in literature.
- The order of the algorithm depends, for netlists with more then a hundred modules, merely on the calculation of the eigenvectors. If all eigenvectors have to be calculated then the order of the program is $O(n^3)$. The order of the algorithm depends on the convergence of the wanted eigenvectors, if not all eigenvectors have to be calculated.
- The program is able to handle a net with special care, so that the netlength of that net decreases.
- The disadvantage of the placement algorithm is that the costfunction is limited to the total netlength. This can lead to the forming of clusters of modules which are heavily connected to each other.

References

[1] Dewilde, P.
The Integrated Circuit Design Book,
Delft University Press, 1986.

[2] Frankle, J. and Karp, R.M.
Circuit Placement and Cost Bounds by Eigenvector Decomposition,
IEEE, p 414,417 1986.

[3] Parlett, B.N.
The Symmetric Eigenvalue Problem,
Prentice Hall Inc. , 1980.

[4] Lippens, P.E.R.
GADL - a gate array description language
M.Sc. Thesis. Automatic System Design Group, Department of Electrical Engineering, Eindhoven University of Technology, 1984.

[5] Nuijten, P.A.C.M.
Hierarchical wire routing of gate arrays
M.Sc. Thesis. Automatic System Design Group, Department of Electrical Engineering, Eindhoven University of Technology, 1985.

[6] Slenter, A.G.J.
Local routing of gate arrays
M.Sc. Thesis. Automatic System Design Group, Department of Electrical Engineering, Eindhoven University of Technology, 1985.

[7] Jongen, R.J.
Gate array placement by simulated annealing
M.Sc. Thesis. Automatic System Design Group, Department of Electrical Engineering, Eindhoven University of Technology, 1984.