An improved method to extract the capacitances of an integrated circuit

The layout of an integrated circuit divided into windows

14th June 1998

Willemijn van Til
Delft University of Technology
Faculty of Information Technology and Systems
Sections: Circuits and Sytems
Applied Mathematics
paper nr.: A98.06

Graduation commission:
N.P van der Meijs
P. Sonneveld
P. Wesseling

Contents

List of Figures

List of Tables

1 Introduction

Before the fabrication of a chip (an integrated circuit), the correctness of its layout should be verified. The verefication step ensures that no chip with an incorrect layout is taken into production. This verification could be done by simulating the equivalent circuit or by static analyses.

Due to the advancements of the integrated circuit technologies, the verification step becomes more and more important. Because of developments like the decrease in size of the features on the integrated circuits and reduction of the switching times, the electrical behavior of an integrated circuit is determined more and more by parasitic elements, like resistance, inductance and capacitance. Hence, to determine the behavior of an integrated circuit, the parasitic elements must be known.

The purpose of this research is to find an improved method to determine the capacitances of an integrated circuit, with a high accuracy. There are several methods to determine these capacitances. One of these methods is the Boundary Element Method. The main problem of this method is the necessary inversion of a dense matrix. The inversion of a dense $N \times N$ matrix requires $O(N^3)$ operations, where N is the number of elements. Since the number of elements increases, the size of the matrix increases and the computation time (the number of operations) increases. For a large number of elements the inversion becomes unmanageable. To reduce the computation time, the capacitances can be approximated. The methods to approximate the capacitances must be efficient and accurate. An algorithm is efficient if the computation time and the necessary amount of memory are as low as possible.

The size of the design that can be handled, is usually determined by the amount of available memory. Therefore the amount of memory is more important than the computation time, which is theoretical unbounded for non-real time problems

For this paper, two existing algorithms that more or less satisfy the above are described. These are the *Hierarchical Schur* algorithm and *FastCap* algorithm. The purpose is to find an algorithm that requires less time and memory than the traditional algorithms.

This paper is structured as follows: In Chapter 2, explanations of some electrical concepts are given. In Chapter 3, the extraction problem is explained. Furthermore the theoretical facts of the advancements of an integrated circuit are given. In Chapter 4, the mathematical background of the problem is discussed. This background is divided into two sections, the theoretical background and the numerical background. In the theoretical background the analytic equations that determine the capacitances is given. In the numerical background these equations are solved numerically with the Boundary Element Method. In Chapter 5, the two algorithms *Hierarchical Schur* and *FastCap* are described. In Chapter 6, the new algorithm is explained. In the first section a global description is given, the second section contains a detailed description and the high-order algorithms. In Chapter 7, the time complexities, space complexities and accuracies of the two algorithms described in Chapter 5 and the new algorithm described in Chapter 6 are given. In Chapter 8, the new algorithm is tested, and the results are discussed. In Chapter 9 the conclusion and an indication for further research is given.

2 Background

Some terms in this paper might not be common for the readers. Terms like capacitance, dielectric medium, ideal conductor, etc. Therefore an explanation of these terms is given in this chapter.

If the concepts are familiar, this chapter can be skipped.

2.1 Integrated Circuit

An integrated circuit can contain thousands of components like resistances, capacitors, inductors, transistors, etc. These components are created on a chip of semi-conducting material, the substrate, by using photographical and chemical techniques and connected by a network of conductors.

The network and the substrate are divided into several layers. The substrate is the bottom layer and the network is partitioned among the other layers. These layers are seperated by isolation layers which have holes at the intersection of two connecting conductors on different successive layers or at the point where the conductors are connected to the components.

When an integrated circuit is designed each layer has its own color. In Figure 2.2 a two-dimensional layout is made of a static RAM cell. Figure 2.1 shows the detail of a three-dimensional layout for the same static RAM. In the figures the layers are divided by the colors. The grey areas in Figure 2.1 are the connections between conductors of different layers. In Figure 2.2 these connections are given by the boxes □.

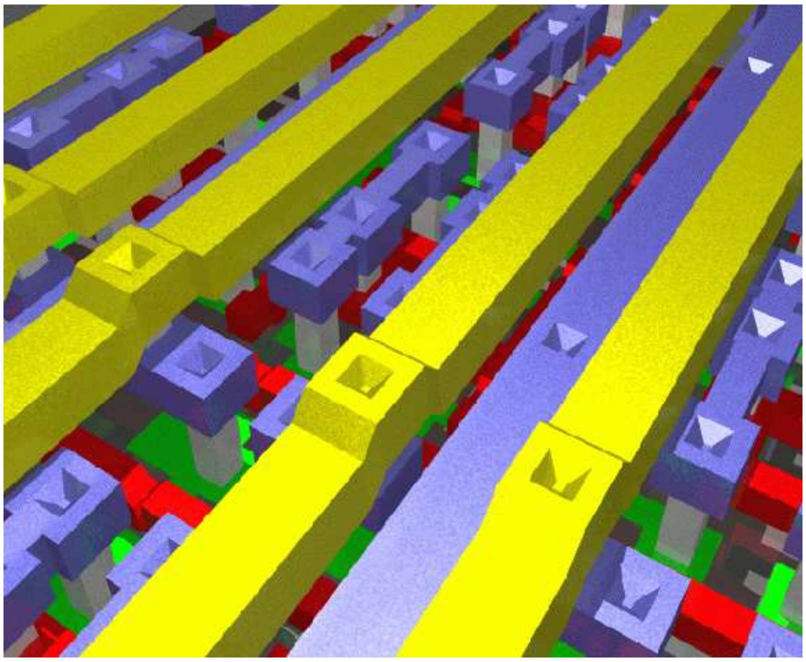

Figure 2.1: Detail of a 3D model for a static RAM array as generated by Space and subsequently rendered by povray.

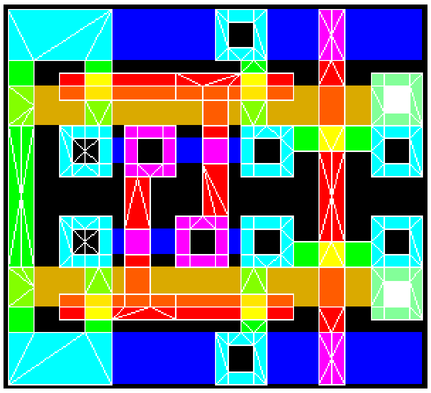

Figure 2.2: Detail of a 2D model for a static RAM array

2.2 Concepts

An **ideal conductor** is a conductor on which the potential is uniformally distributed. In other words, the potential of an ideal conductor is equal at two different points on the conductor.

To explain **capacitances** a plate-capacitor, two plates facing each other [Buijze (1992)], is used. On the plates there are equal but opposite charges q. Furthermore there is an electrical field $\vec{E}$ between the plates and an associated potential u, see Figure 2.3.

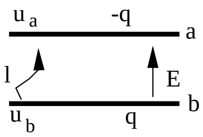

Figure 2.3: A plate-capacitor

The relation between the potential u and the electrical field $\vec{E}$ is given by

$$u = u_b - u_a = \int_a^b \vec{E} d\vec{l} \tag{2.1}$$

When the charge q on surface a is enlarged with a certain factor, the electrical field enlarges in the same proportion. Hence, the potential u enlarges. This relation between the potential u and the charge q is called the capacitance C and is given by

$$q = Cu \tag{2.2}$$

The capacitance of a plate-capacitor is approximately

$$C = \frac{q}{u} = \frac{\varepsilon_0 \varepsilon_r EA}{Ed} = \varepsilon_0 \varepsilon_r \frac{A}{d} \tag{2.3}$$

ε_0 = vacuum permittivity ($= 8.854 \cdot 10^{-12} \frac{C^2}{Nm^2}$)
ε_r = relative permittivity
A = size capacitor
d = thickness isolation layer

For more than two conductors a similar relationship among the conductors can be written in matrix notation as:

$$\hat{q} = \hat{C} \hat{u} \tag{2.4}$$

$\hat{q}$ = vector of conductor charges
$\hat{u}$ = vector of conductor potentials
$\hat{C}$ = capacitance matrix

The **dielectric medium** is a congestion of positive and negative charges which can not move freely between atoms and molecules [Buijze (1992)]. When a dielectric medium is placed between the plates of a capacitor, an electrical field $\vec{E}_p$ developed which is directed opposite of the electrical field $\vec{E}_v$ of the condenser, resulting into an electrical field $\vec{E}$ given by

$$\vec{E} = \vec{E}_p + \vec{E}_v$$

Since $E_p < E_v$ and the opposite direction of $\vec{E}_p$ and $\vec{E}_v$, the electrical field between the surfaces a and

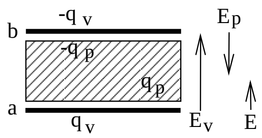

Figure 2.4: The dielectric medium

b, see Figure 2.4, is weakened. According to Equation (2.1) the potential u decreases, which results into an increased capacitance C, Equation (2.2), when the charge q stays the same.

3 The Capacitance Extraction Problem

In this chapter the extraction problem is described. First, the complexity of the extraction problem is explained by comparing an integrated circuit to the road system of the Netherlands. Second, the expectations of the development of an integrated circuit are given. These expectations will show the problem more clearly.

3.1 Example of the Extraction Problem

To be able to compare an integrated circuit with the road system of the Netherlands, this road system has the following characteristics:

- The roads are 3.5 m wide.
- The roads lie 3.5 m apart.
- There are 5 layers of roads above each other.

The roads of the road system have an influence on each other. Usually roads only influence each other when there is an intersection. The roads of the road system, described above, will not only influence each other at an intersection but everywhere. In other words, a road has an influence on a road above, beneath, at the side and even a block away of the initial road.

To determine the behavior of such a road system it is necessary to know the influence between two roads. The amount of influence depends on the distance between the roads. Two roads with a small distance between them will have a larger influence at each other, than two roads with a larger distance between them. For example, the road near The Hague has a larger influence on a road near Rotterdam than on a road near Groningen.

Since the road system has a large number of roads, it requires a lot of time to determine all influences between the roads. And the road system is still in development. The width of the roads decreases, the distance between the roads decreases and the number of layers increases. In other words, the number of roads increases in the course of time. Due to these developments the time required to determine all the influences increases, and the proces becomes more complex. Hence, the determination of all influences becomes a problem. This problem is called the extraction problem.

The size of the extraction problem is determined by the number of roads a road system contains. The number of roads depends on the width of the roads, the distance between the roads and the number of layers. As written above the number of roads increases in the course of time, which results in the increasing size of the extraction problem.

3.2 The Extraction Problem

The behavior of an integrated circuit can be compared with the behavior of the road system. Hence, the number of details on an integrated circuit determines the size of the extraction problem. In other words, when the number of details on an integrated circuit increases, the size of the extraction problem increases. The increasing of the number of details depend on the development of the integrated circuit. The organization SIA has predicted these developments until the year 2010.

Table 3.1: Expectation of the development of the integrated circuit by the SIA.

Year		1995	1998	2001	2004	2007	2010
Gate length	$[\mu m]$	0.35	0.25	0.18	0.13	0.10	0.07
Line width	$[\mu m]$	0.40	0.30	0.22	0.15	0.11	0.08
Chip size	$[mm^2]$	250	300	360	430	520	620
Transistors	$[10^6]$	8	17	37	85	210	450
Clock	$[MHz]$	300	450	600	800	1,000	1,100
Dissipation	$[Watt]$	80	100	120	140	160	180
Wire length	$[m]$	380	840	2,100	4,100	6,300	10,000
Wire levels		4-5	5	5-6	6	6-7	7-8

Table 3.1 shows that the line width decreases, the number of layers increases and the wire length increases in the course of time. In other words, the number of details on an integrated circuit increases and the feature size decreases. Due to the decrease in feature size, the electrical behavior of an integrated circuit is determined more and more by unintentional parasitic elements (capacitance, inductance, resistance, etc.).

4 The Mathematical Background

To extract the capacitances of an integrated circuit, it is necessary to determine the relation between the potential of the conductors and the charge on the conductors. In the first section of this chapter a description of this relation is given. In the second section the relation will be solved numerically.

4.1 Theoretical Background

Consider an integrated circuit of m ideal, charged conductors embedded in a uniform lossless dielectric medium with permitivity ε. The electrical field surrounding the conductors is governed by the Laplace equation

$$\nabla^2\psi(x) = 0 \tag{4.1}$$

A formal solution of the Laplace equation is

$$\psi(x) = \int_{\text{all charge}} \frac{1}{4\pi\varepsilon|x - x'|}\sigma(x')da' \tag{4.2}$$

$\psi(x)$	=	surface potential
$\sigma(x')$	=	surface charge density (unknown)
$x, x' \in I\!R^3$	=	positions in three-dimensional space
ε	=	permittivity dielectric medium
da'	=	incremental surface area

The factor $\frac{1}{4\pi\varepsilon|x-x'|}$ can be interpreted as the potential induced at point x due to an unit point charge at x'. With this interpretation it can be noted that the Laplace equation is linear and that the superposition principle is applied.

Equation (4.2) establishes a global relationship between potential and charge for a multiconductor system. A similar relationship exists among the conductors and is written in matrix notation as:

$$\hat{q} = C\hat{p} \tag{4.3}$$

$\hat{p} \in I\!R^m$	=	the vector of m conductor potentials
$\hat{q} \in I\!R^m$	=	the vector of m conductor charges
$C \in I\!R^{m*m}$	=	the short-circuit capacitance matrix (unknown)

An example of the possible capacitances between three conductors are depicted in Figure 4.1. The charge of each conductor induces potentials to the substrate (a ground plane), and the other conductors.

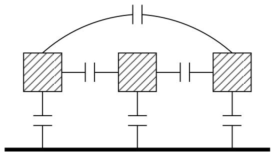

Figure 4.1: The capacitances for three conductors

Since the potential is uniformally distributed among the conductor (an ideal conductor), the potential depends on the charge distribution. The relation between the potential and the charge distribution is given by the following integral equation of the first kind as done in [Meijs (1992)] and [Nabors (1991)].

$$\psi(x) = \int_{\text{surfaces}} G(x, x')\sigma(x')\, da' \tag{4.4}$$

$\sigma(x')$	$=$	surface charge density (unknown)
$x, x' \in I\!R^3$	$=$	positions in three-dimensional space
da'	$=$	incremental surface area
$\psi(x)$	$=$	surface potential
$G(x, x')$	$=$	$\frac{1}{4\pi\varepsilon\lvert x-x'\rvert}$ the Green's function

There are several methods, analytic and numerical, to solve Equation (4.4). Since there is in general no analytical solution, a numerical approach is used to solve Equation (4.4).

4.2 The Numerical Methods

The commonly used numerical methods to solve Equation (4.4) are the Boundary Element mMthods (BE) and the Finite Element Methods (FE). The difference between these methods is related to the domain where the electrical field is solved. The Finite Element Method solves the electrical field for an infinite domain, which is usually truncated by introducing an artifical boundary. The Boundary Element Method solves the electrical field for a finite domain, which contains the problem.

Because of the above, the Boundary Element Method only discretizes the conductor surfaces, which results into a finite number of panels. The Finite Element Method discretizes the free space around the conductors, resulting into a theoretically infinite number of panels. The number of panels is reduced by the introduction of an artifical boundary. But by introducing an artifical boundary, an error is introduced. This error can be reduced by the conditions of the artifical boundary. Nevertheless, this research will focus on the Boundary Element Methods.

4.2.1 The Boundary Element Method

The Boundary Element Method partitions the domain of a function f, the conductor surfaces, into a number of sub-domains, the finite number of panels. On each sub-domain a local approximation of f is defined, such that their assembly is a piecewise approximation of f.

When the conductor surfaces are partitioned into the finite number of panels, N, the approximation of the charge density $\sigma(x)$ is

$$\sigma(x) \approx \tilde{\sigma}(x) = \sum_{i=1}^{N} \sigma_i f_i(x) \tag{4.5}$$

with $\int_{\text{panel } j} f_i(x) da = \begin{cases} 1 & i = j \\ 0 & i \neq j \end{cases}$

The approximation of Equation (4.4) is

$$\begin{aligned} \psi(x) &= \int_{\text{surfaces}} G(x, x') \sum_{i=1}^{N} \sigma_i f_i(x')\, da' && (4.6) \\ &= \sum_{i=1}^{N} \sigma_i \int_{S_i} G(x, x') f_i(x')\, da' && (4.7) \end{aligned}$$

The collocation method and the Galerkin method are two Boundary Element Methods that use an approximation for the charge density $\sigma(x)$. In both methods, the surfaces of m ideal conductors are discretized into a total of N two-dimensional panels and the potential is induced by the charge distribution. In the collocation method, the system of equations relates the total charge on panel a to the potential at the center of panel b. In the Galerkin method, the system of equations relates the total charge on panel a to the weighted averaged potential on panel b.

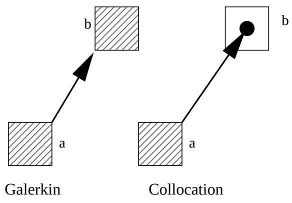

Figure 4.2: Methods of construction

The collocation method will be used in further research, because it is the simplest approach of the methods to contstruct a system of equations. The equation associated with the collocation method is

$$V_k = \sum_{l=1}^{N} \int_{\text{panel } l} \frac{\sigma_l(x')}{||x' - x_k||} \, da' \tag{4.8}$$

V_k	$=$	potential at the center of panel x_k
$\sigma_l(x')$	$=$	surface charge density panel l
x_k	$=$	center of panel k
x'	$=$	position on the surface of panel l
da'	$=$	incremental surface area
N	$=$	number of panels

This equation is the integration over the surface of panel l of the free-space Green's function multiplied by the charge density. The free-space Green's function is given by $\frac{1}{||x-x'||}$, with $||x - x'||$ the Euclidian distance between x and x'.

Assume the surface charge density $\sigma_l(x')$ constant in the first-order collocation method. Substituting $\sigma_l(x') = \frac{Q_l}{a_l}$ in Equation (4.8) gives

$$V_k = \sum_{l=1}^{n} \frac{Q_l}{a_l} \int_{\text{panel } l} \frac{1}{||x' - x_k||} \, da' \tag{4.9}$$

a_l	$=$	surface area of panel l
Q_l	$=$	surface charge of panel l
$\frac{Q_l}{a_l}$	$=$	surface charge density of panel l ($\sigma_l(x')$)

Equation (4.9) is one entry of the system of equations constructed by the collocation method, the system can be written as follows:

$$GQ = V \tag{4.10}$$

where

$$G_{kl} = \frac{1}{a_l} \int_{\text{panel } l} \frac{1}{||x' - x_k||} \, da' \tag{4.11}$$

$V \in I\!R^N$	$=$	panel potential vector
$Q \in I\!R^N$	$=$	panel charge vector
$G \in I\!R^{N*N}$	$=$	influence matrix, Green's function matrix

Since Equation (4.10) can be solved, the remaining problem is to relate the solution to the conductors. An incidence matrix A gives the relation between the panels and the conductors. The incidence matrix A is given by

$$A_{ij} = \begin{cases} 1 & \text{panel } i \text{ on conductor } j \\ 0 & \text{otherwise} \end{cases} \tag{4.12}$$

The charge on a conductor is the sum of all panel charges. Therefore, the relation between the conductor charge and the panel charges is given by

$$\hat{q} = A^T Q \tag{4.13}$$

The conductors are ideal, thus the potential is the same for all panels on a conductor and the relation between the conductor potential and the panel potential is given by

$$V = A\hat{p} \tag{4.14}$$

With the Equations (4.13) and (4.14), Equation (4.3) becomes

$$\hat{q} = A^T Q = A^T G^{-1} V = A^T G^{-1} A \hat{p} = C_s \hat{p} \tag{4.15}$$

where

$$C_s = A^T G^{-1} A \tag{4.16}$$

C_s is called the short-circuit capacitance matrix. The entry $C_{s_{ij}}$ is the charge on conductor i when the j-th conductor is held at unit potential and all other conductors are short-circuited to ground. The capacitance matrix C contains the capacitances between conductor i and the substrate, C_{ii}, and the coupling capacitances between conductor i and j, C_{ij}. The relation between the short-circuit capacitance matrix C_s and the capacitance matrix C is

$$C_{ii} = \sum_{j=1}^{m} C_{s_{ij}} \tag{4.17}$$

$$C_{ij} = -C_{s_{ij}} \tag{4.18}$$

The matrix C_s is constructed with the inverse of the dense matrix G. The matrix G is a $N \times N$-matrix with N the total number of panels. The time to construct G and its space complexity are $O(N^2)$, the time to compute the inverse of G is $O(N^3)$. An algorithm with such a time complexity rapidly becomes computationally unmanageable. Two methods to reduce the time complexity are the *Hierarchical Schur* and the *FastCap* method.

5 Commonly Used Methods for Solving the System

The short-circuit matrix is in the previous chapter given by Equation (4.16)

$$C_s = A^T G^{-1} A$$

with

C_s	=	the $m \times m$ short-circuit capacitance matrix
G	=	the $N \times N$ matrix of Green's function
A	=	the $m \times N$ incidence matrix

In order to find the $m \times m$ short-circuit capacitance matrix, the $N \times N$ matrix of Green's function G, Equation (4.10) must be factored once. The usual factorization is the LU-decomposition ($G = LU$), which requires $O(N^3)$ operations. A dense matrix G has a dense inverse G^{-1} which results in a dense short-circuit matrix C_s, that specifies a capacitance between every pair of conductors. Two algorithms to reduce the computation time of the short-circuit capacitance matrix C_s are *Hierarchical Schur* [Meijs (1992)] and *FastCap* [Nabor (1991)] [Nabor (1992)].

Most entries of C_s are small and have a negligible effect on the behavior of the circuit. *Hierarchical Schur* omits these negligible capacitances, resulting into a sparse approximation of the short-circuit matrix. *FastCap* does not omite the neglible capacitances, but avoids the factorization by using an iterative residual algorithm. Multipole expansions give a further reduction of the computation time.

In this chapter these algorithms are described, the complexities of them are given in Chapter 7.

5.1 The Hierarchical Schur Algorithm

A sparse approximation $\tilde{C}_s$ of C_s can model the behavior of the circuit accurately. The zeros in this approximation $\tilde{C}_s$ correspond to the negligible coupling between widely separated conductors and the non-zeros correspond to interactions between neighboring conductors. The approximation $\tilde{C}_s$ is sparse when an approximation $\tilde{G}^{-1}$ of G^{-1} is sparse.

5.1.1 Description

The *Generalized Schur* algorithm gives a sparse approximation $\tilde{G}^{-1}$ when the support[1] of $\tilde{G}^{-1}$ has a staircase band structure, see Section 5.1.2. The *Generalized Schur* algorithm produces an approximate positive definite inverse G_{ME}^{-1} for a positive definite matrix G that is specified on a staircase band S. This approximate inverse G_{ME}^{-1} is the inverse of the so-called maximum entropy extension of G. An extension of a partially specified positive matrix is any Hermitian matrix that matches the matrix on its specified entries. The maximum entropy extension is the unique positive definite extension whose inverse has zeros in the positions corresponding to the unspecified entries of the partially specified matrix. This unique extension has the maximum determinant of all possible extensions.

The *Hierarchical Schur* algorithm is based on the *Generalized Schur* algorithm. The *Hierarchical Schur* algorithm provides a sparse-inverse approximation G_{SI}^{-1} of G, where G is a positive definite matrix

[1] The support of a matrix S is the set of pairs of indices (i, j) such that $S_{ij} \neq 0$.

that is specified on a multiple band. The following notations are used to describe the sparse-inverse approximation.

$\mathbf{G}$ the matrix G partitioned as a block matrix

$\mathbf{G}(i,j)$ the principal block matrix that lies in the rows and columns of $\mathbf{G}$ indexed by $i,\ldots,j$

$\Box[\mathbf{G};(i,j)]$ the block matrix such that $(\Box[\mathbf{G};(i,j)])(i,j) = \mathbf{G}$ and is zero on the other parts of $\Box[\mathbf{G};(i,j)]$

$\mathbf{G}(.,.)$ takes a block out of a matrix and $\Box[\mathbf{G};(.,.)]$ embeds a matrix in a larger matrix such that it is surrounded by zeros. The sparse-inverse approximation is given by

$$G_{SI}^{-1} = \sum_{j=1}^{N-M} \Box[\mathbf{G}(j,j+M)_{ME}^{-1};(j,j+M)] - \sum_{j=1}^{N-M-1} \Box[\mathbf{G}(j+1,j+M)_{ME}^{-1};(j+1,j+M)] \quad (5.1)$$

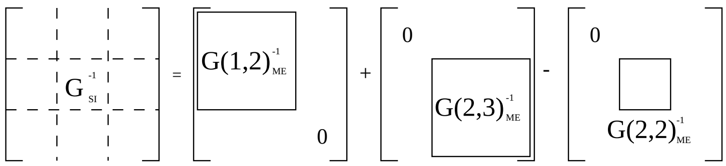

Figure 5.1: An example of the sparse-inverse approximation for N=3, M=1

5.1.2 Finite Element Numbering

Hierarchical Schur uses a staircase band S as support of G_{SI}^{-1}. This support S depends on the numbering of the panels. Let ω be a fixed distance over which coupling is considered significant. Partition the layout into non-overlapping vertical strips of width ω. It is possible that the right-most strip has a width $< \omega$. Number the strips from left to right $S_1, S_2, \ldots, S_n$. The strip S_i corresponds to block $\mathbf{G}(i,i)$ of $\mathbf{G}$ and the abutment of some successive strips $S_i \ldots S_j$ to the submatrices $\mathbf{G}(i,j)$ of $\mathbf{G}$, Figure 5.2. Within each

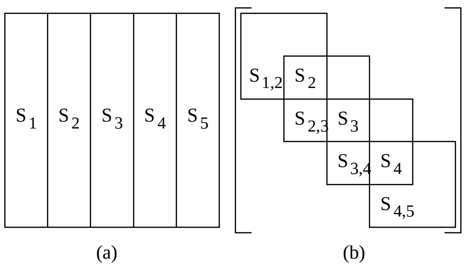

Figure 5.2: A strip partition (a), the corresponding blocks of the influence matrix (b)

strip, a staircase of the corresponding matrix is obtained by numbering the boundary elements in order of increasing y-coordinates and by neglecting the x-component and z-component of the distance between the boundary elements. In the y-direction all interactions between boundary elements at distance $\leq \omega$ are included and not all interactions between boundary elements at distance $> \omega$ are excluded. In fact, for $M = 1$ in Equation (5.1) there are at most two consecutive strips with a total width of 2ω. When the distance in the z-direction is neglected, the finite element numbering for the *Hierarchical Schur* algorithm is:

$$\begin{array}{lll} (i,j) \in S & \text{if} & dist(e_i,e_j) \leq \omega \\ (i,j) \notin S & \text{if} & dist(e_i,e_j) > 2\omega \end{array}$$

where some (i,j) with $\omega < dist(e_i,e_j) \leq 2\omega$ are in S and some are not. When the distance in the z-direction is taken into account, 2ω is replaced by $\sqrt{4\omega^2+t^2}$, where t is the thickness of the dielectric layer.

5.2 FastCap

An other method to reduce the time complexity of the inversion of the dense matrix G is the multipole approximation. The multipole method *FastCap* uses an iterative residual algorithm (GCR, GMRES, CGS) and expansions to solve the m charge distribution problem [Nabors (1991)] [Nabors (1992)].

5.2.1 Description

The factorization of the dense matrix G is avoided by using an iterative residual algorithm. Hence, the major costs of an iterative residual algorithm is initially forming the dense matrix G and computing the matrix-vector product Gw, with w the residual vector. The costs can be reduced by representing the potential arising from the charge distribution w by Gw.

In addition to this reduction it is possible to reduce the forming of G. If the distance between the center of panel i and panel j is large compared with the panel sizes, then

$$G_{ij} \approx \frac{1}{\|x_i - x_j\|} \tag{5.2}$$

The jth panel charge has the same effect on the potential at x_i as would a point charge of value w_j located at the center of panel j.
This approximation can be seen as the 1st coefficient of a multipole expansion. This implies that the costs for the computation Gw can be further reduced when expansions are used.

There are two expansions, the local and the multipole expansion. These expansions divide the panels into the point charges and the evaluation points, which are separated by two circles.

5.2.2 Expansions

Multipole expansions

Assume that the point charges lie in the inner circle and the evaluation points lie outside the outer circle, see Figure (5.3).

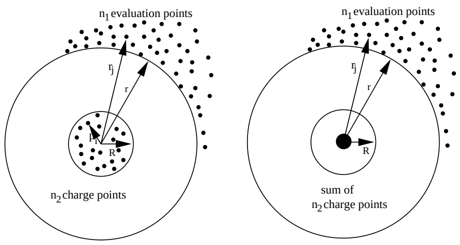

Figure 5.3: The evaluation points and the point charges for a multipole expansion

An approximation of the effect of the n_2 charges in the inner circle can be derived by replacing the charges with a single charge equal to the sum of the n_2 charges, placed at the inner circle's center. Hence, to compute the n_1 potentials requires $n_1 + n_2$ operations.

The potential arising from the charges is approximated by

$$\frac{1}{r_j} \sum_{i=1}^{n_2} q_i \tag{5.3}$$

q_i = the point charge i
r_j = the distance between the center of the charge circle and the jth evaluation point

The true potential arising from point charges inside a sphere at a location outside the radius of the sphere can be approximated by a truncated multipole expansion

$$\psi(r_j,\theta_j,\phi_j) \approx \sum_{n=0}^{l}\sum_{m=-n}^{n} \frac{M_n^m}{r_j^{n+1}} Y_n^m(\theta_j,\phi_j) \tag{5.4}$$

l = the order of the expansion
r_j, θ_j, ϕ_j = spherical coordinates of the jth evaluation point
$Y_n^m(\theta_j, \phi - j)$ = spherical harmonics
M_n^m = complex multipole coefficients (weights)

When the point charges lie outside the outer circle and the evaluation points lie in the inner circle, see Figure (5.4), the charge points are widely distributed and the multipole expansion can not be used. Therefore a local expansion is used to compute the potentials of the evaluation points.

Local expansions

An approximation of the potential at any of the n_1 evaluation points in the inner circle is the same as the potential evaluated at the center of the circle. Hence, to compute the potentials at the n_1 evaluation points takes $n_1 + n_2$ operations.

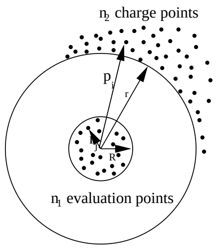

Figure 5.4: The evaluation points and the charge points for a local expansion

The potential at an evaluation point can be approximated by

$$\sum_{i=1}^{n_2} \frac{q_i}{\rho_i} \tag{5.5}$$

q_i = the point charge i
ρ_i = the distance from the ith charge to the center of the inner circle.

The true potential in a sphere caused by charges outside the radius of the sphere can be approximated with a truncated local expansion.

$$\psi(r_j,\theta_j,\phi_j) \approx \sum_{n=0}^{l}\sum_{m=-n}^{n} L_n^m r_j^n Y_n^m(\theta_j,\phi_j) \tag{5.6}$$

l = the order of the expansion
r_j, θ_j, ϕ_j = spherical coordinates of the jth evaluation point with respect to the sphere's center
$Y_n^m(\theta_j, \phi_j)$ = spherical harmonics
L_n^m = complex local coefficients

Partition

An integrated circuit is partitioned into cubes. The smallest cube that contains the entire circuit is determined. This cube is referred to as level 0 cube. Next, the volume of the level 0 cube is broken into eight child cubes of equal size, referred to as level 1 cubes. The panels are divided among the child cubes by associating the panel with a cube. Panels can be associated with a cube when a certain point of the panel is contained in the cube, for example the panel's center point. This process is repeated to produce L levels of cubes and L partitions of panels starting with an eight-way partition and ending with an 8^L-way partition. The number of levels, L, is chosen so that the maximum number of panels in the Lth level cube is less than some treshold.

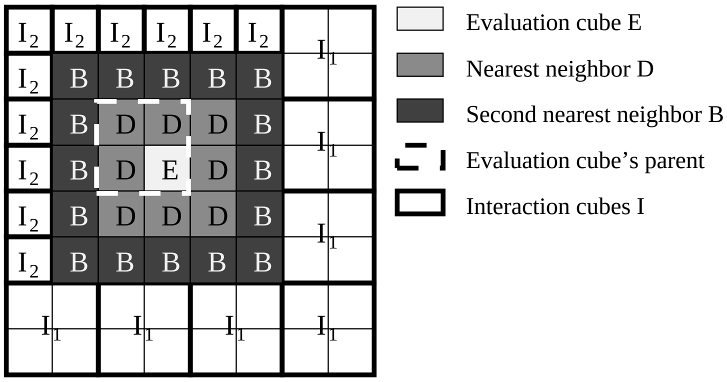

Figure 5.5: A two-dimensional representation of the division

In Figure 5.5 the cubes are grouped in evaluation cubes, the nearest neighbors, the second nearest neighbors, the parent of the evaluation cube and the interaction cubes. The nearest neighbors are those cubes on the same level which have at least one corner in common with the evaluation cube. The second nearest neighbors are those cubes on the same level which are not nearest neighbors, but have a corner in common with the nearest neighbors. The interaction cubes of the evaluation cubes are those cubes which are either the second nearest neighbors of the evaluation cube's parent, I_1, or children of the evaluation cube's parent's nearest neighbors, excluding nearest or second nearest neighbors of the evaluation cube, I_2.

The interaction cubes have two important properties:

1. Combined with the evaluation cube's nearest and second nearest neighbors, they cover the same volume as the evaluation cube's parent and the parent's nearest and second nearest neighbors.

2. The distance between a point in the evaluation cube and a point in the interaction cube is more than half the distance between the centers of the evaluation and the interaction cubes.

A multipole algorithm determines the potentials for each cube. The multipole expansions make it possible to determine the potentials for each cube of level i with the potentials of level $i+1$ cubes. The local expansions make it possible to determine the potentials for each cube of level i with the potentials of level $i-1$ cubes. These operations are called the conversion and shift operations. When the geometry is fixed, the conversion and shift operations are linear functions of the charges or the expansion coefficients. In other words, the complicated evaluations involved in conversion or shifting are computed once and stored as matrices, which operate on charges or coefficients.

6 A New Method to Solve the System

The purpose of this research is to develop an algorithm with a better time complexity and space complexity than the two algorithms described in the previous chapter. The algorithm that is developed is described in this chapter. In the first section a global description of the algorithm is given. In the second section a detailed description is given. In the next chapter the time complexity and space complexity of all three algorithms are given.

6.1 Global Description

The algorithm can be described in four steps. The first step is to divide the m conductors into a total of N panels. This step is usually done by the layout-design program. The second step is to partition the layout of the integrated circuit. *Hierarchical Schur* partitions the layout into non-overlapping strips. *FastCap* partitions the layout into a hierarchy of cubes, with each cube of the finest level containing less panels than a certain treshold. In this research a partition of non-overlapping windows is used. A detailed description is given in Section 6.2.1. The third step is creating a partition of blocks out of the window partition. The blocks contain adjacent windows, so that the coupling capacitances between two elements in different windows are taken into account. Hence, the blocks must be overlapping. A detailed description is given in Section 6.2.2. The last step is to extract the capacitances of the integrated circuit. For each block the capacitances are extracted with an extraction program like *Hierarchical Schur* and *FastCap*. When the capacitances for each block are extracted they are combined into a capacitance matrix for the integrated circuit. A detailed description is given in Section 6.2.3.

Algorithm 1: The algorithm to extract the capacitances.

```
/* Setup */
Divide the m conductors into a total of N panels.
Use algorithms 2 and 3 to divide the problem domain into windows.
Use algorithm 4 to construct the blockstructure.
/* Extract capacitances with methods like FastCap and Hierarchical Schur*/
Use algorithm 5 to combine the short-circuit capacitance matrix for the integrated
circuit.
```

6.2 Detailed Description

6.2.1 The partition of the integrated circuit

In *Hierarchical Schur* the advantage is the partition of the layout into strips. For each strip a staircase band matrix is determined. When the layout is partitioned twice, first into vertical strips and second a horizontal partition of the strips, the result is a set of non-overlapping windows, see Figure 6.1. Hence, the partition of the circuit is now in the x- and y-direction, the distance in the z-direction is neglected. This is allowed because the depth of an integrated circuit is negligible compared to its width and length.

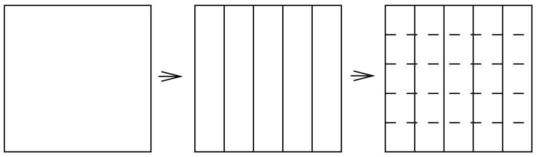

Figure 6.1: (a) the layout of an integrated circuit, (b) the layout partitioned into vertical strips, (c) the strips partitioned into windows.

Algorithm 2: Dividing the problem domain into windows

```
/* Setup */
/* m is the number of conductors
   N is the number of panels
   w is the fixed distance over which coupling is significant
   minx, miny, maxx and maxy give the size of the circuit
   Wy is the number of windows in the y-direction
   Wx is the number of windows in the x-direction */

/* Determine the size of the circuit */
minx=minimum x-coordinate of the circuit;
miny=minimum y-coordinate of the circuit;
maxx=maximum x-coordinate of the circuit;
maxy=maximum y-coordinate of the circuit;
/* Determine the lower left corner of the windows */
Wx=truncate((maxx-minx)/w); Wy=truncate((maxy-miny)/w);
for i=0 to Wx {
   for j=0 to Wy {
      set xlowerleft[i]=minx+w*i
      set ylowerleft[j]=miny+w*j
      window[i,j] is associated with the lower left corner
      [xlowerleft[i],ylowerleft[j]]
   }
}
```

When the circuit is divided into windows, the panels have to be associated with the windows. There are several ways to associate the panels with the windows. In this research a panel is associated with a window when a panel's center is contained in the window.

Algorithm 3: Associating the panels with the windows

```
/* N is number of panels
   w is the fixed distance over which coupling is significant
   px is the x-coordinate of the panel's center
   py is the y-coordinate of the panel's center */

for j=1 to N {
   set px=x-coordinate of the center of panel j
   set py=y-coordinate of the center of panel j
   /* Determine in which window the panel is to be placed */
   x=truncate(px/w); y=truncate(py/w);
   place panel j in window[x,y]
}
```

6.2.2 The construction of the blocks

Coupling capacitances are not limited to single windows. The coupling capacitances of adjacent windows are also significant when the behavior of an integrated circuit is determined. A combination of adjacent windows is called a block. These blocks are divided into large, horizontal, vertical and small blocks. All large blocks contain 2×2 adjacent windows, all horizontal blocks contain 2×1 adjacent windows, all vertical blocks contain two adjacent windows and all small blocks contain 1×1 window, see Figure 6.2.

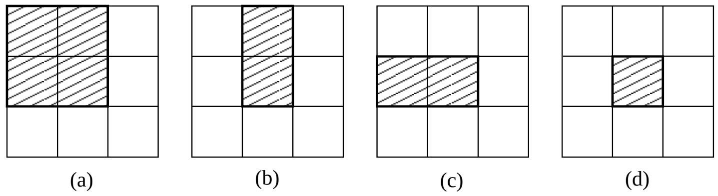

Figure 6.2: Examples of a large block (a), a vertical block (b), a horizontal block (c) and a small block (d)

A horizontal block and a vertical block are the overlaps of two large blocks. A small block is the overlap of two horizontal blocks or two vertical blocks. Due to the fact that a horizontal block is the overlap between one large block and the large block shifted down or up one window, it is not possible that a horizontal block occurs at the bottom or the top of the layout partition. It is also clear that the vertical blocks can not occur at the left and right border of the layout partition. Since the small blocks are the overlaps between two horizontal or vertical blocks, they can not occur at the border of the layout partition.

Algorithm 4: Constructing the blocks

```
/* Setup */
/* Wy is the number of windows in the y-direction
   Wx is the number of windows in the x-direction
   minx, miny, maxx and maxy give the circuitsize
   w is the distance over which coupling is significant */

/* Construct large blocks */
for i=0 to (Wx-1) {
   for j=0 to (Wy-1) {
      l_block[i,j] contains window[i,j], window[i+1,j], window[i,j+1] and
      window[i+1,j+1]
   }
}
/* Construct vertical blocks */
for i=1 to (Wx-1) {
   for j=0 to (Wy-1) {
      v_block[i,j] contains window[i,j] and window[i,j+1]
   }
}
/* Construct horizontal blocks */
for i=0 to (Wx-1) {
   for j=1 to (Wy-1) {
      h_block[i,j] contains window[i,j] and window[i+1,j]
   }
}
/* Construct small blocks */
for i=1 to (Wx-1) {
```

```
    for j=1 to (Wy-1) {
        s_block[i,j] contains window[i,j]
    }
}
```

6.2.3 The construction of the short-circuit capacitance matrix for the integrated circuit

The short-circuit capacitances are extracted for each block and stored in a blockmatrix. For each block the entries of the blockmatrix has to be placed into one matrix for the entire integrated circuit. The general principle of the placement of the matrices is given after an example which explains the adding/subtracting the entries of the blockmatrices to/of the capacitance matrix for the entire layout.

Imaging the layout of an integrated circuit that is partitioned into 3×3 windows. There are four large blocks, two vertical blocks, two horizontal blocks and one small block. The capacitances are extracted for the nine blocks. It takes four steps to construct the large matrix.

Step 1:

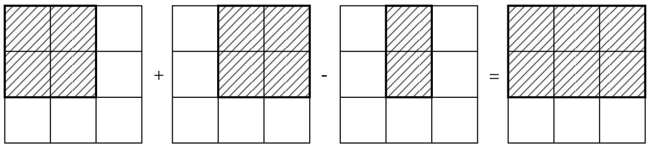

Figure 6.3: The first step of the construction of the large matrix

First the capacitances of the large blocks are extracted and stored in matrices. The entries of these matrices are added to the entries of a temporary matrix. Since the large blocks have a overlap (vertical block), some capacitances are added twice to the temporary matrix. To subtract these capacitances, it is necessary to extract the capacitances of the vertical block. In other words, the entries of the capacitance matrices of the large blocks are added to the temporary matrix and the entries of the capacitance matrix of the overlap (vertical block) are subtracted of the temporary matrix.

Step 2:

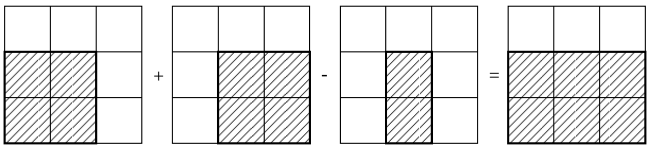

Figure 6.4: The second step of the construction of the large matrix

The second step is like the first step, except that the initial block is shifted down one window. Again a temporary large matrix can be constructed.

Step 3:

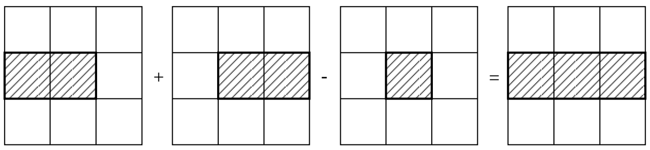

Figure 6.5: The third step of the construction of the large matrix

An overlap occurs between a large block and the large block that is shifted down or up one window. This overlap is not considered in the first step and second step and has to be dealt with seperately. The overlap can be divided into two horizontal blocks. When the capacitances of these two blocks are extracted and added to a temporary matrix for the entire overlap, there is again an overlap (small block) of which the capacitances must be extracted and subtracted of the temporary matrix.

Step 4:

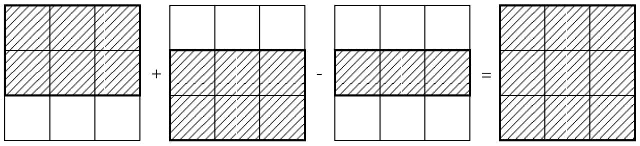

Figure 6.6: The fourth step of the construction of the large matrix

The temporary matrices are placed in the large short-circuit capacitance matrix.

When the concept is extended to a partition with any number of windows larger then 3×3, the following occurs each time:

- The entries of the capacitance matrices of the large blocks are added to a short-circuit capacitance matrix for the entire integrated circuit.
- The entries of the capacitance matrices of the horizontal blocks are subtracted of the short-circuit capacitance matrix.
- The entries of the capacitance matrices of the vertical blocks are subtracted of the short-circuit capacitance matrix.
- The entries of the capacitance matrices of the small blocks are added to the short-circuit capacitance matrix.

Hence the inverse approximation is given by

$$G_{blocks}^{-1} = \sum_{i=0}^{Wx-1} \sum_{j=0}^{Wy-1} G_{l_{ij}}^{-1} - \sum_{i=0}^{Wx-1} \sum_{j=1}^{Wy-1} G_{h_{ij}}^{-1} \quad (6.1)$$

$$- \sum_{i=1}^{Wx-1} \sum_{j=0}^{Wy-1} G_{v_{ij}}^{-1} + \sum_{i=1}^{Wx-1} \sum_{j=1}^{Wy-1} G_{s_{ij}}^{-1} \quad (6.2)$$

Wx = maximum number of windows in the x-direction
Wy = maximum number of windows in the y-direction

Algorithm 5: Constructing the short-circuit matrix for an integrated circuit

```
/* Setup phase */
/* Wx and Wy are the numbers of windows in
   respectively the x-direction and the y-direction */

Use an extraction algorithm to extract for the large, horizontal, vertical and small
blocks the short-circuit capacitance matrix.
/* Construct the matrix for the circuit */
for i=0 to (Wx-1) {
   for j=0 to (Wy-1) {
      Add the values of the short-circuit capacitance matrix of l_block[i,j] to
      the short-circuit capacitance matrix of the integrated circuit
```

```
   }
}
for i=1 to (Wx-1) {
   for j=0 to (Wy-1) {
      Subtract the values of the short-circuit capacitance matrix of v_block[i,j] of
      the short-circuit capacitance matrix of the integrated circuit
   }
}
for i=0 to (Wx-1) {
   for j=1 to (Wy-1) {
      Subtract the values of the short-circuit capacitance matrix of h_block[i,j] of
      the short-circuit capacitance matrix of the integrated circuit
   }
}
for i=1 to (Wx-1) {
   for j=1 to (Wy-1) {
      Add the values of the short-circuit capacitance matrix of s_block[i,j] to
      the short-circuit capacitance matrix of the integrated circuit
   }
}
```

7 Performance analyses

The purpose of this research is to find an algorithm with a better time complexity and space complexity than *Hierarchical Schur* and *FastCap*. To compare the three algorithms with a well-known algorithm, the time and space complexity of the exact inverse are also given. Beside the time and space complexity, the accuracy is also important. The developed algorithm should not make the accuracy of the extracted capacitances worse. In this chapter the complexities and the accuracy of the three algorithms are given. Furthermore the testcircuit and the used parameters are described.

7.1 Testcircuit

As a testcircuit a layout consisting of two crossing busses, which is also used in [Nabors (1991)] and is easily parameterized in the circuit size. The thickness and width of the conductors as well as the vertical seperation between the busses is in all cases 1μ, microns are more specific to the case of VLSI. Furthermore, the dielectric medium has a dielectric constant of 1.0 (vacuum) and the tests were made without a groundplane.

Figure 7.1 shows a 2×5 crossing bus and the used numbering scheme.

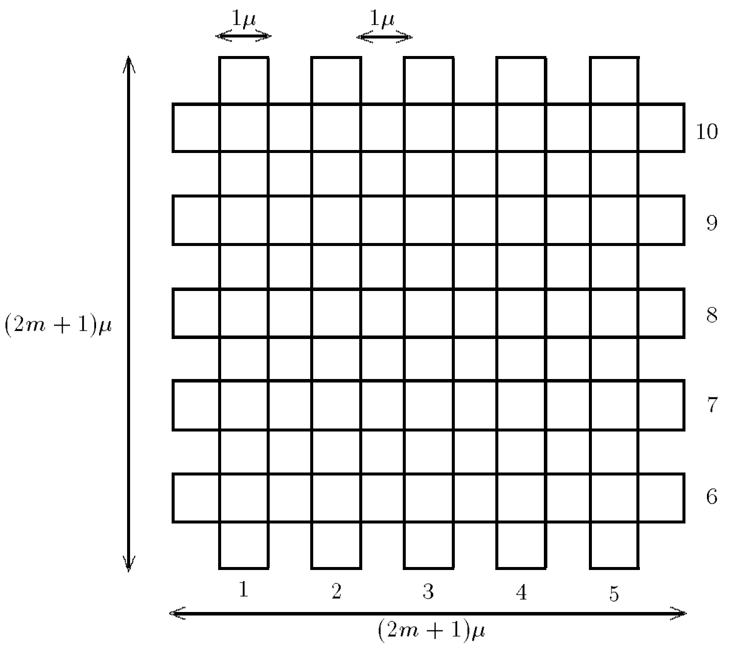

Figure 7.1: Layout of a 2×5 conductor crossing bus

FastCap uses the number of finite elements in the mesh as a measure of complexity. An example of a mesh is shown in Figure 7.2.

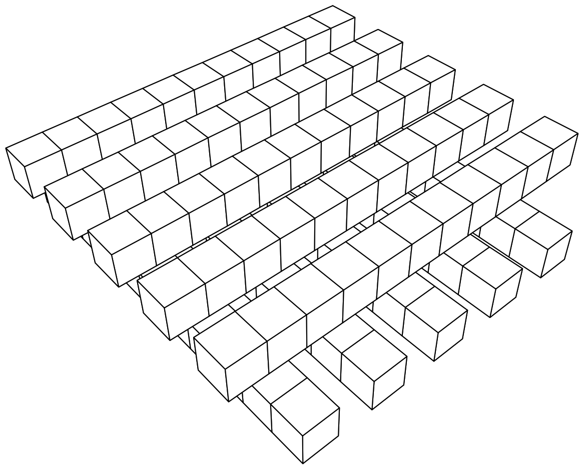

Figure 7.2: Finite element mesh for a 2×5 conductor crossing bus

7.2 Parameters

circuit size A is the surface covered by the layout of an integrated circuit.

window width ω is length of a side of a window. Since the length of the sides of a window are assumed to be equal, the window width is equal to the length of any side of the window.

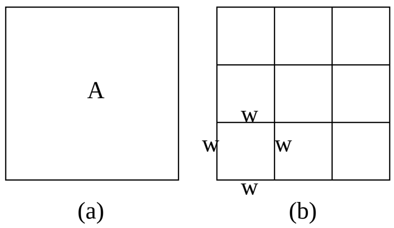

Figure 7.3: The circuit size A (a) and the window width ω (b)

number of conductors m is the number of conductors of a crossing bus. For a crossing bus it depends on the scaling parameter and the circuit size, $m = \frac{A}{\lambda^2}$

number of panels N is the number of panels in a mesh. For a crossing bus the number depends on the number of conductors and the relative panelsize, $N = \frac{m}{s^2} = \frac{A}{(s\lambda)^2}$

scaling parameter λ is the number of conductors per circuit size or the relation between the height and the width of the conductors. The conductors do not scale in height or distance to the ground plane. For a crossing bus the scale depends on the circuit size and the number of conductors, $\lambda^2 = \frac{A}{m}$.

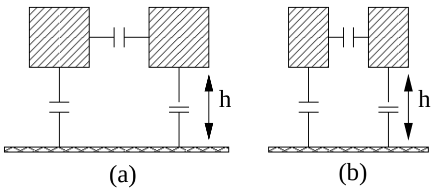

Figure 7.4: The scaling parameter λ, (a) $\lambda = 1$, (b) $\lambda = \frac{2}{3}$

relative panel size s^2 is the number of panels per conductors or the surface covered by one panel relative to the total size of the circuit. The relative panel size depends on the circuit size, the number of panels, and the scaling parameter, $s^2 = \frac{m}{N}$ or $s^2 = \frac{A}{N\lambda^2}$. The absolute panel size is given by $(s\lambda)^2$.

The circuit size A is the main parameter, because for a crossing bus all parameters depend on this parameter. Hence, the parameters s^2 (relative panel size), λ (scaling parameter) and A (circuit size) are used to describe the complexities.

7.3 Complexities

In this section the time complexities and the space complexities are given. Included in the time complexity of the new algorithm are the time complexities of the construction of the blocks, the short-circuit capacitance matrices per block and the combining these matrices into one short-circuit capacitance matrix of the integrated circuit.

7.3.1 The time complexities

In the literature the time complexities of the exact inverse, *Hierarchical Schur* and *FastCap* are found, the time complexities are given in Table 7.1.

Table 7.1: The time complexities of the three algorithms

Algorithm	Time complexity $O(..)$
Exact Inverse	$N^3 = \frac{A^3}{(s\lambda)^6}$
Hierarchical Schur	$\frac{N\omega^4}{(s\lambda)^4} = \frac{A\omega^4}{(s\lambda)^6}$
FastCap	$Nm = \frac{A^2}{s^2\lambda^4}$

For the new algorithm, described in Chapter 6, the time complexities are determined by the number of blocks, $O(\frac{A}{\omega^2})$, the number of panels per block, $O(\frac{\omega^2}{(s\lambda)^2})$, the bandwidth of the matrix (*Hierarchical Schur*), $O(\frac{\omega^2}{(s\lambda)^2})$, and the number of conductors per block (*FastCap*), $O(\frac{\omega^2}{\lambda^2})$.

Step 1:
The time complexity of the construction of the blocks is proportional to the number of panels, $O(\frac{A}{(s\lambda)^2})$, and the number of blocks, $O(\frac{A}{\omega^2})$, see Algorithms 2,3 and 4 in the previous chapter. Since there are more panels than blocks, the time complexity for the construction is $O(\frac{A}{(s\lambda)^2} + \frac{A}{\omega^2}) \approx O(\frac{A}{\omega^2})$.

Step 2:
The time complexity for constructing the short-circuit capacitance matrices for each block depends on the algorithm that is used. Examples of these algorithms are the exact inverse, *Hierarchical Schur* and *FastCap*. In other words, the time complexity to extract the capacitances of all the blocks is proportional to the time complexity of the used algorithm and the number of blocks, $O(\frac{A}{\omega^2})$.

Table 7.2: The time complexities to extract the capacitances.

Algorithm	Time complexity $O(..)$	
	Block	Total
Exact Inverse	$\frac{\omega^6}{(s\lambda)^6}$	$\frac{\omega^6}{(s\lambda)^6} \cdot \frac{A}{\omega^2} = \frac{A\omega^4}{(s\lambda)^6}$
Hierarchical Schur	$\frac{\omega^6}{(s\lambda)^6}$	$\frac{\omega^6}{(s\lambda)^6} \cdot \frac{A}{\omega^2} = \frac{A\omega^4}{(s\lambda)^6}$
FastCap	$\frac{\omega^4}{s^2\lambda^4}$	$\frac{\omega^4}{s^2\lambda^4} \cdot \frac{A}{\omega^2} = \frac{A\omega^2}{s^2\lambda^4}$

Step 3:
The combining of the matrices of each block into one short-circuit matrix is proportional to the number

of panels, $O(\frac{A}{(s\lambda)^2})$, see Algorithm 5 in the previous chapter. In other words, the time complexity of this step is $O(\frac{A}{(s\lambda)^2})$.

The time complexity of the new algorithm is equal to the summation of the time complexities of the three steps. In other words, the time complexity of the new algorithm is

$$O(\frac{A}{\omega^2}) + O(?) + O(\frac{A}{(s\lambda)^2})$$

with $O(?)$ the time complexity of the algorithm used to extract the capacitances.

Table 7.3: The time complexities of the new algorithm combined with the three algorithms

Algorithm	Time complexity $O(..)$
Exact Inverse	$\frac{A}{\omega^2} + \frac{A\omega^4}{(s\lambda)^6} + \frac{A}{(s\lambda)^2} \approx \frac{A\omega^4}{(s\lambda)^6}$
Hierarchical Schur	$\frac{A}{\omega^2} + \frac{A\omega^4}{(s\lambda)^6} + \frac{A}{(s\lambda)^2} \approx \frac{A\omega^4}{(s\lambda)^6}$
FastCap	$\frac{A}{\omega^2} + \frac{A\omega^2}{s^2\lambda^4} + \frac{A}{(s\lambda)^2} \approx \frac{A\omega^2}{s^2\lambda^4}$

Hence, the new algorithm in combination with *FastCap* has the best complexity of the time complexities given above.

7.3.2 The space complexity

The space complexity is important, because the memory of the computer gives the bounds of the extraction problem. Swapping data between working memory and background memory costs more time than when the problem fits completely in the working memory. Extraction problems are so complex that they usually do not fit in the working memory.

The space complexities for the exact inverse and *Hierarchical Schur* are again found in the literature. The space complexity of *FastCap* was not found in the articles [Nabors (1991)] [Nabors (1992)]. An estimate of the space complexity can be made by the algorithms given in these articles. Table 7.4 gives the space complexities.

Table 7.4: The space complexities of the three algorithms

Algorithm	Space complexity $O(..)$
Exact Inverse	$N^2 = \frac{A^2}{(s\lambda)^4}$
Hierarchical Schur	$\frac{\omega^4}{(s\lambda)^4}$
FastCap	$N = \frac{A}{(s\lambda)^2}$

With the window partition of the layout, the space complexities are equal to the space complexities of the large blocks, which is proportional to the number of panels per block, $O(\frac{\omega^2}{(s\lambda)^2})$. Table 7.5 gives the space complexities for a window partition of the layout.

Table 7.5: The space complexities of the new algorithm combined with the three algorithms

Algorithm	Space complexity $O(..)$
Exact Inverse	$\frac{\omega^4}{(s\lambda)^4}$
Hierarchical Schur	$\frac{\omega^4}{(s\lambda)^4}$
FastCap	$\frac{\omega^2}{(s\lambda)^2}$

Again the new algorithm in combination with *FastCap* has the best complexity of the space complexities given above.

7.4 Accuracy

It is necessary to determine the behavior accurate, because it is expensive and costs a lot of time to produce integrated circuits and it is difficult to make measurements of an integrated circuit. In other words, the approximated capacitances must be accurate enough to determine the behavior. Usually the designers are satisfied with accuracies in the range of 5-10%. Therefore, in most cases, a relatively coarse mesh is sufficent.

7.4.1 Hierarchical Schur

The error of *Hierarchical Schur* depends on the width of the strips. The accuracy increases, when the width of the strips increases. In other words, when the bandwidth of the influence matrix increases, the accuracy increases.

For the smaller window width the short-circuit capacitances are still accurate enough, because the “nearby” capacitances increase slightly an compensate for not computing the “far” capacitances.

7.4.2 FastCap

The error of *FastCap* depends on the separation between the evaluation points and the point charges and on the order of the expansions.

For the multipole expansion the accuracy improves as the separation, r, between the nearest evaluation point and the center of the inner circle containing the charges increases relative to the inner circle's radius, R, ($r \gg R$). For the local expansion the accuracy of the approximation improves as the separation, r, between the charges and the circle's center increases relative to the circle's radius, R, ($r \gg R$). But the accuracy can be improved by including the effect of the distance between the evaluation point and the enclosing sphere's center.

When the distance between an evaluation point and the charge point is more than half the distance between the centers of the evaluation and interaction cubes, $\frac{R}{r} < 0.5$, the accuracy increases when the order of the truncated expansions increases. The higher the order of the expansions, the smaller the error of the approximation is.

7.4.3 The new algorithm

The error of the new algorithm depends on the method to compute the capacitances of the blocks and the window width. For a window width more than half the lenght of the side of a circuit, one large block covers the entire circuit, Hence, the accuracu of the new algorithm is equal to the accuracy of the used method, here *FastCap* or *Hierarchical Schur*

The capacitances stayed within a range of 5% of the capacitances given by the used method (*FastCap*, *Hierarchical Schur*. In other words, the new algorithm is as accurate as the method, which extracts the capacitances per block.

8 Results

The new algorithm is tested in combination with *FastCap*. *FastCap* is used, because the time complexity and the space complexity are better than the complexities when *Hierarchical Schur* or the exact inverse are used. To compare the new algorithm, the capacitances of the entire integrated circuits are also extracted by *FastCap*. The complexities that are tested are given in Table 8.1. The testcircuits are $2 \times n$

Table 8.1: The complexities for the inverse algorithm *FastCap*

Layout	Time complexity $O(..)$	Space complexity $O(..)$
Entire	$\frac{A^2}{s^2\lambda^4}$	$\frac{A}{(s\lambda)^2}$
Blocks	$\frac{A\omega^2}{s^2\lambda^4}$	$\frac{\omega^2}{(s\lambda)^2}$

crossing busses, with a non-uniform discretisation. These circuits are constructed with Busgen, a layout program included by *FastCap*.

The tests were carried out on a Pentium 100 PC with a 22 Mbyte RAM memory and operating system Linux.

8.1 Discretisation

There are two possible discretisations of the conductors, uniform and non-uniform, Figure 8.1.

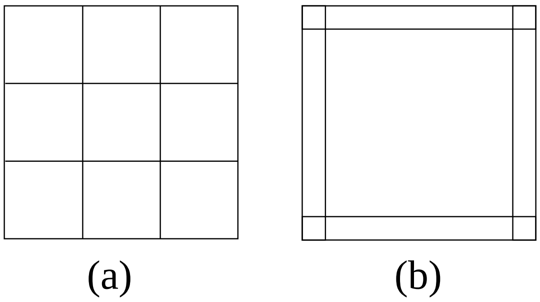

Figure 8.1: An uniform discretisation (a) and a non-uniform discretisation

All panels of the uniform discretisation are equal in size. The panels of the non-uniform discretisation differ in size. The edge panels have widths that are 10% of the inner panel widths. The non-uniform discretisation takes the expected increased charge density near conductor edges into account. In the tests a non-uniform discretisation is used.

8.2 The effects of the circuit size A

The effects of the circuit size A are investigated on the $2 \times n$ crossing bus, with $5 < n < 16$. The width of the wires is 1μ and the window width is 1μ. Table 8.2 shows the CPU time and the used memory.

Table 8.2: The effects of the circuit size A for window width $\omega = 1\mu$

bus	CPU time		Memory (M)	
	FastCap	Blocks	*FastCap*	Blocks
2x5	13.48	25.04	2.60	0.05
2x6	35.19	36.96	3.76	0.05
2x7	79.53	48.81	5.68	0.05
2x8	34.87	62.44	8.09	0.05
2x9	50.55	82.86	10.63	0.05
2x10	93.42	108.60	12.08	0.05
2x12	201.63	199.89	16.72	0.05
2x14	603.34	263.14	26.26	0.05
2x16	1104.62	428.51	36.83	0.05

Figure 8.2 shows the effect of the circuit size A on the CPU time. When the window width ω^2, the relative panel size s^2 and the scaling parameter λ are constant the time complexity is $O(A^2)$ for *FastCap* and $O(A)$ for the new algorithm. These theoretical amounts are confirmed by the testresults shown in Figure 8.2.

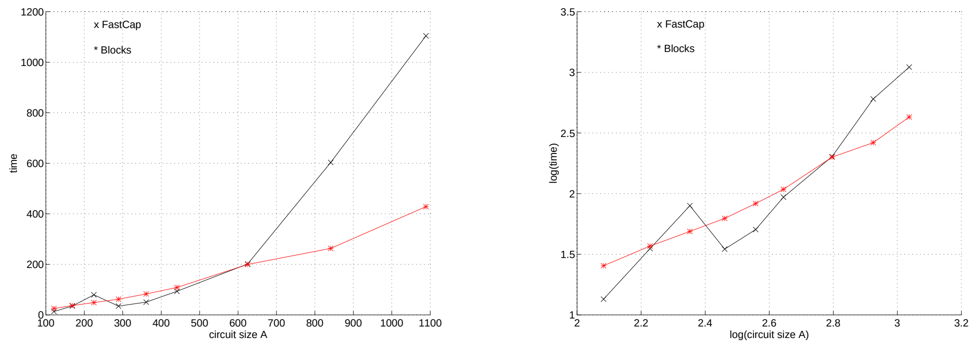

Figure 8.2: CPU time versus circuit size, normal scale and log-log scale

Figure 8.3 shows the effect of the circuit size A on the necessary amount of memory. When the window width ω, the relative panel size s^2 and the scaling parameter λ are constant the space complexity is $O(A)$ for *FastCap* and $O(1)$ for the new algorithm. These theoretical amounts are confirmed by the testresults shown in Figure 8.3.

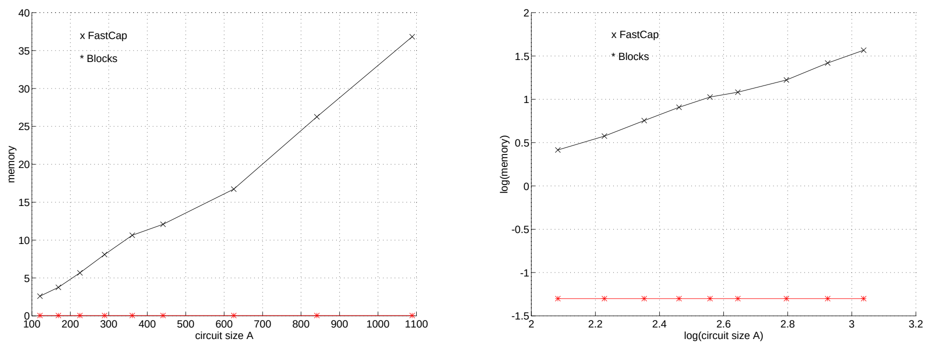

Figure 8.3: Memory use versus circuit size, normal scale and log-log scale

8.3 The effects of the window width ω

The effect of the window width ω is tested with a 2×20 crossing bus. *FastCap* has some trouble to extract the capacitances for the some layoutpartitions of the integrated circuit. *FastCap* gives non-negative off-diagonals in the output, resulting in a short-circuit capacitance that is incorrect and larger usertimes, which can not be explained.

Table 8.3 shows the short-circuit capacitance and some of the capacitances for this testcircuit, the CPU time and the used memory. Because of symmetry, $C_{1\,21} = C_{1\,40}$, $C_{1\,22} = C_{1\,39}$, $C_{1\,23} = C_{1\,38}$, $C_{1\,24} = C_{1\,37}$, $C_{1\,25} = C_{1\,36}$, $C_{1\,26} = C_{1\,35}$, $C_{1\,27} = C_{1\,34}$, $C_{1\,28} = C_{1\,33}$, $C_{1\,29} = C_{1\,32}$ and $C_{1\,30} = C_{1\,31}$.

Table 8.3: 20X20 bus crossing problem with 1e-06m wires (n=1 e=0.1)

program	w	time			memory	Capacitances ($10^{-18}F$)						
	(μ)	user	system	CPU	(M)	$C_{S\,1}$	$C_{1\,gnd}$	$C_{1\,2}$	$C_{1\,3}$	$C_{1\,4}$	$C_{1\,5}$	$C_{1\,6}$
fastcap		614.51	1825.60	2440.11	52.113	1261.00	91.43	449.40	35.76	14.87	11.35	5.326
blocks	32.00	617.51	1760.88	2378.39	52.114	1261.00	91.43	449.40	35.76	14.87	11.35	5.326
blocks	16.00	2702.78	685.27	3388.05	36.412	2218.00	187.83	779.60	60.01	27.61	16.76	11.290
blocks	8.00	3028.20	12.14	3040.34	7.635	1581.00	148.52	561.60	42.73	20.11	11.85	8.441
blocks	4.00	550.65	20.49	571.14	1.404	1589.00	192.60	559.50	42.84	28.88		
blocks	2.00	231.51	69.74	301.25	0.214	1610.00	245.29	562.40	35.44			
blocks	1.00	208.68	549.14	757.82	0.046	1539.00	243.29	472.80				

program	w	Capacitances ($10^{-18}F$)											
	(μ)	$C_{1\,7}$	$C_{1\,8}$	$C_{1\,9}$	$C_{1\,10}$	$C_{1\,11}$	$C_{1\,12}$	$C_{1\,13}$	$C_{1\,14}$	$C_{1\,15}$	$C_{1\,16}$	$C_{1\,17}$	$C_{1\,18}$
fastcap		4.934	3.591	2.942	3.565	0.088	1.658	1.585	1.369	1.396	0.843	0.974	1.04
blocks	32.00	4.934	3.591	2.942	3.565	0.088	1.658	1.585	1.369	1.396	0.843	0.974	1.04
blocks	16.00	8.277	6.414	3.899	4.235	3.582	3.194	2.354	2.599	2.660	5.153		
blocks	8.00	6.552	9.203										
blocks	4.00												
blocks	2.00												
blocks	1.00												

program	w	Capacitances ($10^{-18}F$)											
	(μ)	$C_{1\,19}$	$C_{1\,20}$	$C_{1\,21}$	$C_{1\,22}$	$C_{1\,23}$	$C_{1\,24}$	$C_{1\,25}$	$C_{1\,26}$	$C_{1\,27}$	$C_{1\,28}$	$C_{1\,29}$	$C_{1\,30}$
fastcap		10.64	20.99	38.56	30.85	30.58	30.57	30.30	30.37	30.32	30.61	30.34	30.35
blocks	32.00	10.64	20.99	38.56	30.85	30.58	30.57	30.30	30.37	30.32	30.61	30.34	30.35
blocks	16.00			47.48	38.30	37.81	37.77	37.66	37.78	37.62	54.38	76.21	75.31
blocks	8.00			45.91	37.70	37.28	37.92	37.69	37.72	37.55	38.84	37.56	37.62
blocks	4.00			44.37	37.55	37.41	38.19	37.13	37.88	37.13	37.96	37.17	37.54
blocks	2.00			43.76	38.40	37.61	37.55	37.27	37.80	38.28	37.61	37.55	37.27
blocks	1.00			46.65	38.48	39.94	39.50	44.45	44.26	38.58	40.03	38.32	38.51

Figure 8.4 shows the effect of the window width ω on the CPU time. When the circuit size A, the relative panel size s^2 and the scaling parameter λ are constant the time complexity is $O(1)$ for *FastCap* and $O(\omega^2)$ for the new algorithm. These theoretical amounts are confirmed by the testresults shown in Figure 8.4.

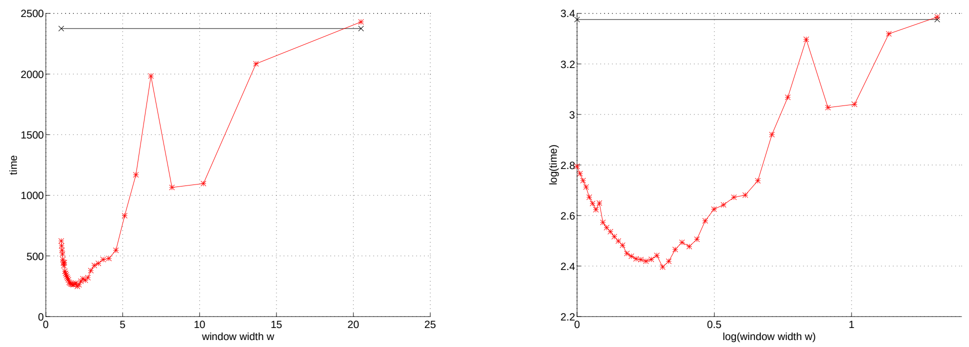

Figure 8.4: CPU time versus window width, normal scale and log-log scale

Due to the fixed starting time of *FastCap*, the CPU time for windows with a window width smaller than $\omega \approx 2.05$ is close to the starting time. Since the number of windows increase and the CPU time of two windows with $\omega < 2.05$ is the same, the total CPU time increases.

A explanation for the oscilation at the end of log-log figure, Figure 8.4, is not found, but the oscilation occures in all tests with $2 \times n$ crossing busses. Therefore the effect could be caused by *FastCap*.

Figure 8.5 shows the effect of the window width ω on the necessary amount of memory. When the circuit size A, the relative panel size s^2 and the scaling parameter λ are constant the space complexity is $O(1)$ for *FastCap* and $O(\omega^2)$ for the new algorithm. These theoretical amounts are confirmed by the testresults shown in Figure 8.5.

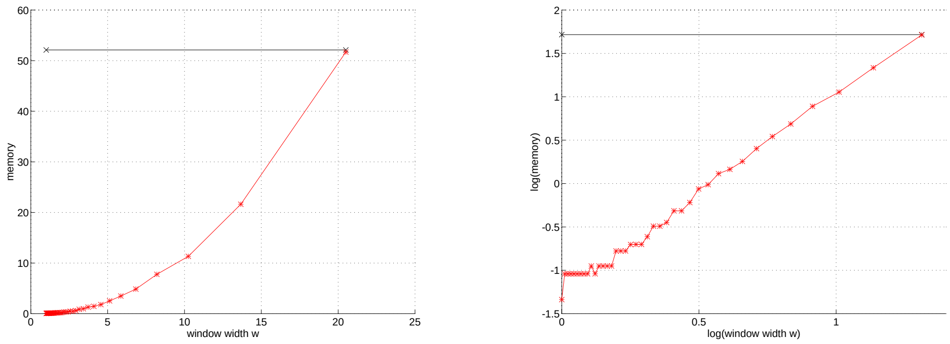

Figure 8.5: Memory use versus window width, normal scale and log-log scale

8.4 The effects of the relative panel size s^2

To test the relative panel size s^2 a 2×5 crossing bus is used. The number of conductors is constant, so that the relative panel size s^2 only depends on the number of panels N. The window width is $\omega = 1.0\mu$. Table 8.4 shows the CPU time and the used memory.

Table 8.4: The effects of the relative panel size s for a 2×5 crossing bus

s^2	N	CPU time		Memory (M)	
		FastCap	Blocks	*FastCap*	Blocks
$\frac{1}{46}$	460	14.05	23.81	2.60	0.05
$\frac{1}{92}$	920	76.50	58.20	16.56	0.25
$\frac{1}{138}$	1380	598.01	168.35	46.17	0.69
$\frac{1}{184}$	1840	1226.83	383.05	79.63	1.56

Figure 8.6 shows the effect of the relativ panel size s^2 on the CPU time. When the circuit size A, the window width ω and the scaling parameter λ are constant the time complexity is $O(s^{-2})$ for *FastCap* and $O(s^{-2})$ for the new algorithm. These theoretical amounts are confirmed by the testresults shown in Figure 8.6.

Figure 8.7 shows the effect of the relative panel size s^2 on the necessary amount of memory. When the circuit size A, the window width ω^2 and the scaling parameter λ are constant the space complexity is $O(s^{-2})$ for *FastCap* and $O(s^{-2})$ for the new algorithm. These theoretical amounts are confirmed by the testresults shown in Figure 8.7.

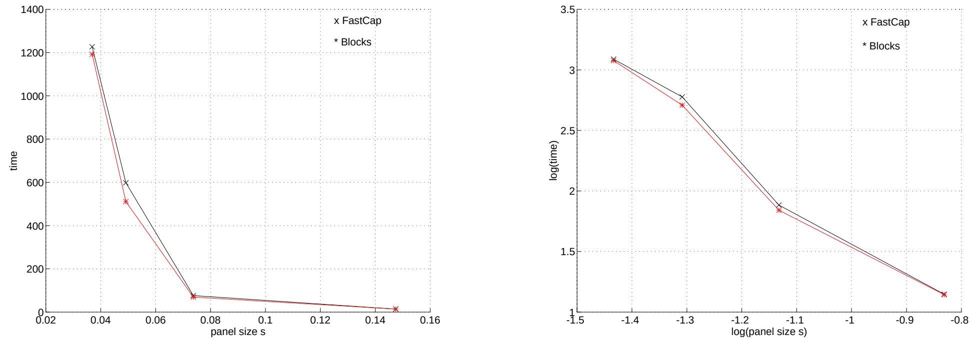

Figure 8.6: CPU time versus relative panel size, normal scale and log-log scale

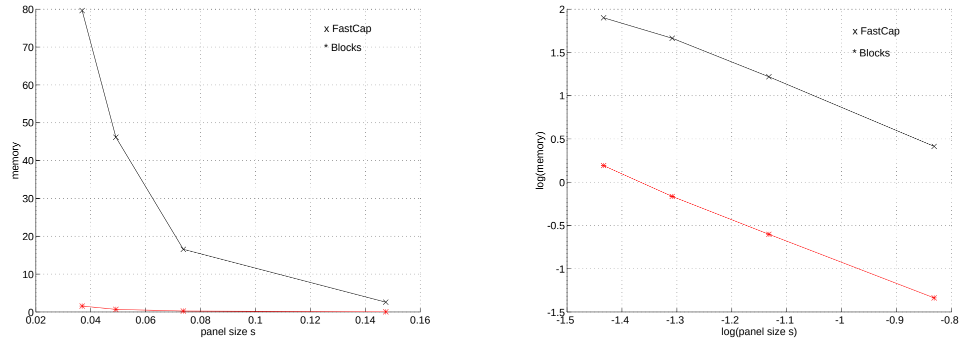

Figure 8.7: Memory use versus relative panel size, normal scale and log-log scale

8.5 The effects of the scaling parameter λ

The scaling parameter λ is tested for a 2×16 crossing bus and a window width $\omega \approx 2$. Since the scaling parameter λ effects the circuit size A, the number of conductors per window increase, when the scaling parameter decreases. Table 8.5 shows the CPU time and the used memory.

Table 8.5: The effects of the scaling parameter λ for a 2×16 crossing bus

λ	CPU time		Memory (M)	
	FastCap	Blocks	*FastCap*	Blocks
1.00	1293.06	190.47	36.832	0.244
0.50	1375.78	382.57	46.371	1.527
0.33	1243.71	466.81	51.110	4.026
0.25	1309.80	861.39	44.132	8.153
0.20	1854.97	1477.59	47.900	15.426
0.16	1669.23	1934.60	52.164	22.713
0.14	1881.87	2554.10	45.081	32.309
0.13	1789.19	3728.94	45.081	44.625
0.11	1087.72	1981.13	0.000	50.327
0.10	5.70	1785.83	0.000	46.355

Figure 8.8 shows the effect of the scaling parameter λ on the CPU time. Since the circuit size A is proportional to the scaling parameter λ, the theoretical CPU time for *FastCap* becomes $O(s^{-2}$ and for the new algorithm $O(\frac{\omega^2}{(s\lambda)^2})$. Hence, when the window width ω and the relative panel size s^2 are

constant, the CPU time is $O(1)$ for *FastCap* and $O(\lambda^{-2})$ for the new algorithm. The theoretical CPU time, as described above, is confirmed by the testresults as shown in Figure 8.8.

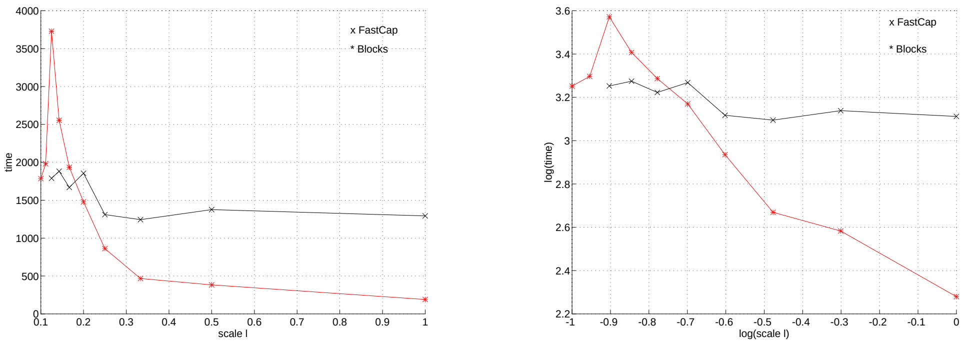

Figure 8.8: CPU time versus scaling parameter, normal scale and log-log scale

Figure 8.9 shows the effect of the scaling parameter λ on the necessary amount of memory. For the amount of memory the scaling parameter λ is proportional to the circuit size A, as for the CPU time, so that the theoretical necessary amount of memory is $O(s^{-2})$ for *FastCap* and $O(\frac{\omega^2}{(s\lambda)^2}$ for the new algorithm. Hence, when the relative panel size s^2 is constant, the necessary amount of memory is $O(1)$ for *FastCap* and $O(\lambda^{-2})$ for the new algorithm. These theoretical amounts are confirmed by the testresults shown in Figure 8.9.

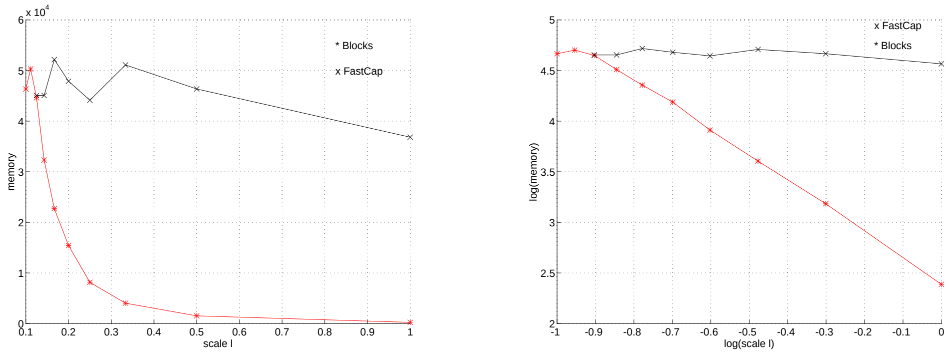

Figure 8.9: Memory use versus scaling parameter, normal scale and log-log scale

9 Conclusion

In this research, an improved method to extract the capacitances of an integrated circuit is found. This method partitions the layout of the circuit into non-overlapping windows, which are combined into overlapping blocks, see Chapter 6. The capacitances, extracted for each block, are combined to the capacitances for the entire integrated circuit. The improved method is only more efficient than the existing methods when the window width ω is less than half the width of the layout and more than twice the width of a conductor. In other words, the window width should satisfy $2\lambda \leq \omega \leq \frac{\sqrt{A}}{3}\lambda$, with λ the scaling parameter and A the circuit size.

The improved method is also more efficient then the existing methods *Hierarchical Schur* and *FastCap*. In other words, the time complexity and the space complexity of the new algorithm are lower than the time complexities and space complexities of *Hierarchical Schur* and *FastCap*. Table 9.1 gives the complexities of the methods.

Table 9.1: The time and space complexities

	Complexity $O(..)$	
	Time	Space
FastCap	$\frac{A^2}{s^2\lambda^4}$	$\frac{A}{(s\lambda)^2}$
Hierarchical Schur	$\frac{A\omega^4}{(s\lambda)^6}$	$\frac{\omega^4}{(s\lambda)^4}$
New algorithm combined with FastCap	$\frac{A\omega^2}{s^2\lambda^4}$	$\frac{\omega^2}{(s\lambda)^2}$

The testcircuits were $2 \times n$ crossing busses. The height of the conductors was $1\,\mu m$ and the width of the conductors and the distance between the conductors were $1\lambda\,\mu m$, where λ is the scaling parameter. The computation time of the 2×20 bus for window width $\omega \approx 2\lambda\,\mu m$ was approximately 10 times faster than *FastCap*. The necessary amount of memory for all partitions with window width $\omega \leq \frac{\sqrt{A}}{3}$ was less than the amount of memory *FastCap* needed.

For the window width $\omega \approx 2\lambda\,\mu m$ the accuracy of the new algorithm is within 2% of the accuracy of *FastCap*. For windows with a window width $1.0\,\mu m < \omega < 2\lambda\,\mu m$ the accuracy of the extracted capacitances becomes worse and the required time for the extraction increases, compared to the accuracy and time of a window width $\omega \approx 2\lambda\,\mu m$. This effect is due to the time that *FastCap* needs to start the extraction.

Further research can involve

1. the extension of the new algorithm for all parasitic elements.

2. recognition of patterns. When a pattern of features occures more than once, the pattern can be placed in one window, or block. Hence, the capacitances of this block/window have to be extracted once, instead of multiple times. The problem is that the partition of the layout will not be uniform.

3. adjustments of the layout. When a conductor is removed or placed in an existing layout and the capacitances of the layout are known, only the capacitances of the blocks/windows in which the adjustment is made, have to be extracted. These capacitances are extracted once with the adjustment and once without the adjustment, so that the capacitances of the block without the

adjustment can be subtracted of the known capacitances and the capacitances of the block with the adjustment can be added to the known capacitances.

4. extracting the capacitances on a parallel computer. The capacitances of the blocks, created by the new algorithm, can be extracted at the same time. When the capacitances are extracted they must be combined to find the capacitances for the entire layout. This could also be done parallel, see Section 6.2.3.

10 Samenvatting

Voordat een chip in productie genomen wordt, wordt de correctheid van de schakeling gecontroleerd. Deze controle kan door middel van simulatie of statische analyse van de schakeling worden gedaan.

De controle van de schakeling wordt steeds belangrijker, omdat de afmetingen van de elementen op een geïntegreerde schakeling kleiner worden en de schakeltijden nemen af. Door deze ontwikkelingen wordt het gedrag van een schakeling steeds meer bepaald door onbedoelde, parasitaire effecten, zoals de capaciteiten, weerstanden en inductanties van de bedrading op een chip.

In dit afstudeerverslag is gezocht naar een verbeterde techniek om de capaciteiten, met een bepaalde nauwkeurigheid, te bepalen. Een manier om de capaciteiten te bepalen is de Boundary Element Method. Bij deze methode wordt een volle matrix geïnverteerd. Zoals bekend kost het inverteren van een volle matrix veel tijd. Een mogelijkheid om deze tijd te reduceren is het maken van een benadering van de capaciteiten.

De methoden die de capaciteiten benaderen, moeten zo efficiënt en nauwkeurig mogelijk zijn. Met andere woorden deze methoden moeten eenvoudig en compact zijn, en alle belangrijke effecten beschrijven en de onbelangrijke weglaten. Oftewel de rekentijd moet zo laag mogelijk zijn.

Buiten de rekentijd is de benodigde geheugenruimte ook van belang. De grootte van een probleem wordt voornamelijk beperkt door de grootte van het werkgeheugen van de gebruikte computer. Dus net als de rekentijd moet de benodigde geheugenruimte zo laag mogelijk zijn.

Tijdens het afstudeeronderzoek zijn dan ook twee algoritmen, *Hierarchisch Schur* en *FastCap*, bekeken die momenteel worden gebruikt voor het bepalen van de capaciteiten. Deze twee algoritmen worden beschreven in hoofdstuk 5. De nieuwe techniek is beschreven in hoofdstuk 6.

Het nieuwe algoritme verdeelt de layout van de geïntegreerde schakeling in blokken. Per blok worden de capaciteiten bepaald met behulp van een algoritme als *FastCap* of *Hierarchisch Schur*. Daarna worden deze capaciteiten samengenomen tot de capaciteiten voor de gehele schakeling.

De complexiteiten van de verschillende methoden worden gegeven in Tabel 10.1 en bevestigd door de tests.

Table 10.1: De tijdcomplexiteit en geheugencomplexiteit

	Complexiteit $O(..)$	
	Tijd	Geheugen
FastCap	$\frac{A^2}{s^2\lambda^4}$	$\frac{A}{(s\lambda)^2}$
Hierarchisch Schur	$\frac{A\omega^4}{(s\lambda)^6}$	$\frac{\omega^4}{(s\lambda)^4}$
FastCap met blokken	$\frac{A\omega^2}{s^2\lambda^4}$	$\frac{\omega^2}{(s\lambda)^2}$

11 Bibliography

Buijze (1992) W.Buijze, 'Inleiding Electriciteit en Magnetisme', Delftse Uitgevers Maatschappij, 1992.

Meijs (1992) N.P. van der Meijs, 'Accurate and efficient layout extraction', Thesis Delft University of Technology, 1992.

Nabors (1991) K.Nabors and J.White, 'FastCap: A Multipole Accelerated 3-D Capacitance Extraction Program', *IEEE Trans. on Computer Aided Design of Integrated Circuits and Systems*, 10(11):1447-1459, November 1991.

Nabors (1992) K.Nabors, S.Kim and J.White, 'Fast Capacitance Extraction of General Three-Dimensional Structures', *IEEE Trans. on Microwave Theory and Techniques*, 40(7):1496-1506, July 1992.

Neerhoff (1992) F.L.Neerhoff, 'Elektrische circuits', Delftse Uitgevers Maatschappij, 1992.

A Software

The software is divided into two sections. The first section contains the perlscripts and the second section contains the C-source code. The perlscripts can be used for all kinds of integrated circuits. The C-source code is written for the input and output of *FastCap*, the origin is (0,0,0) and the corners of the panels must have positive coordinates. The input for *FastCap* is given in the third section.

A.1 Perlscripts

There are two perlscript. The first script is Calcap, which generates the output per block and combines the blockoutput into one matrix, containing the capacitances for the integrated circuit. The blocks are constructed, before Calcap is run, by "blocks".

The second script is Proef, which generates and runs the necessary tests to investigate the theoretical complexities.

A.1.1 Calcap

```
# A perlscript to generate the capacitance matrix with the extraction
# program FastCap

#!/usr/bin/perl

# determine the arguments
sub def_arguments {
  foreach $ARGV (@ARGV) {
    if ($ARGV =~ m/^(-c)/) {($conductor = $ARGV)=~ s/^(-c)//;}
    elsif ($ARGV =~ m/^(-o)/) {($output = $ARGV) =~ s/^(-o)//;}
    elsif ($ARGV !~ m/^(-)/)  {$input = $ARGV;}
  }
  ($file_name = $input) =~ s/[.]\w*//;
}

# combine the blockmatrices into one matrix
sub make_matrix {
  $command = "matrix @_[1].names -c@_[0]";
  print `$command`;
}

# determine the capacitances per block
sub make_output {
  $names = @_[0].".names";
  open(name,"$names");
  while ($line = <name>) {
    chop($line);
```

```
    $command = "fastcap -l$line.lst";
    open (file,">$line.out");
    print file `$command`;
    close file;
   }
  close name;
}

sub write_help {
  printf "\nUsage : calcap [<inputfile>] [-o<outputfile>] [-c<#conductors>] \n";
  exit 0;
}

if ($#ARGV < 2) {write_help;}
elsif ($#ARGV > 3) {write_help;}
def_arguments();
make_output($file_name);
make_matrix($conductor,$file_name,$output);
```

A.1.2 Proef

```
# A perlscript to generate tests with the different window widths

#!/usr/bin/perl

sub clean_dir {
  $command = "rm *_* *.out *.names *.aux *.log *.dvi";
  print `$command`;
}

# determine the arguments
sub def_arguments {
  foreach $ARGV (@ARGV) {
    if ($ARGV =~ m/^(-c)/) {($conductor = $ARGV) =~ s/^(-c)//;}
    elsif ($ARGV =~ m/^(-m)/) {($maxwin = $ARGV) =~ s/^(-m)//;}
    elsif ($ARGV =~ m/^(-u)/) {($unity = $ARGV) =~ s/(-u)//;}
    elsif ($ARGV =~ m/^(-o)/) {($output = $ARGV) =~ s/^(-o)//;}
    elsif ($ARGV =~ m/^(-t)/) {($texfile=$ARGV) =~ s/^(-t)//;}
    elsif ($ARGV =~ m/^(-d)/) {($delers=$ARGV) =~ s/^(-d)//;}
    elsif ($ARGV !~ m/^(-)/)  {($input = $ARGV);}
  }
  if ($output !~ m/^(\w+)/) {($file_name = $input) =~ s/[.]\w*//;}
  else {($file_name = $output)=~ s/[.]\w*//;}
  exponent($unity); $winsize=$symbol;
}

# determine the exponent of the capacitances
sub def_farad {
  $file=@_[0]."0.out";
  open(file,"$file");
  LINE : while ($line=<file>) {
    last LINE if ($line=~m/(%)/);
  }
```

```
  ($string=$line)=~s/^\s*(\w+\W\w+)\s+\d+\s+//;
  close file;
  ($value,$string)=split(' ',$string,2);
  exponent($value);
}

# determine the exponent of the used memory
sub def_byte {
  if (@_[0] < 1e+3) {$byte=1;$memsize=' ';}
  elsif (@_[0] < 1e+6) {$byte=1e+3;$memsize='$k$';}
  elsif (@_[0] < 1e+9) {$byte=1e+6;$memsize='$M$';}
  else  {$byte=1e+9;$memsize='$G$';}
}

sub exponent {
  if (@_[0] < 1e-15) {$farads=1e-18;$symbol='$a$';$name='$10^{-18}F$';}
  elsif (@_[0]<1e-12) {$farads=1e-15;$symbol='$f$';$name='$10^{-15}F$';}
  elsif (@_[0]<1e-9) {$farads=1e-12;$symbol='$p$';$name='$10^{-12}F$';}
  elsif (@_[0]<1e-6) {$farads=1e-9;$symbol='$n$';$name='$10^{-9}F$';}
  elsif (@_[0]<1e-3) {$farads=1e-6;$symbol='$\mu$';$name='$10^{-6}F$';}
  elsif (@_[0]<1e-0) {$farads=1e-3;$symbol='$m$';$name='$10^{-3}F$';}
  elsif (@_[0]<1e+3) {$farads=1;$symbol='$ $';$name='$F$';}
  else {$farads=1e+3;$symbol='$k$';$name='$10^3F$';}
}

# determine the maximum used memory
sub find_max {
  $max=0;
  open (file,"@_[0].names");
  while ($line=<file>) {
    $line=~ s/(\n)/.out/;
    find_memory($line);
    if ($memory>$max) {$max=$memory;}
  }
  close file;
}

sub find_memory {
  open (memfile,"@_[0]");
  LINE: while ($memory=<memfile>) {
    last LINE if ($memory =~ m/Total memory allocated:/);
  }
  close memfile;
  $memory =~ s/Total memory allocated: //;
  ($memory,$sign,$etc)=($memory =~ /^\s*(\S+)\s+(\S).*/);
  if ($sign eq "g") {$memory=$memory*1e+9}
  elsif ($sign eq"m") {$memory=$memory*1e+6}
  elsif ($sign eq "k") {$memory=$memory*1e+3}
  elsif ($sign eq "b") {$memory=$memory*1}
}

# find the values in the outputfiles
sub find_values {
```

```
    open (file,"@_[0]");
    LINE : while ($line=<file>) {
      last LINE if ($line=~m/(%)/);
    }
    close file;
    $line=~s/^\s*(\w+\W\w*)\s+\d+\s+//;

    $values[0]=0;
    for ($i=1;$i<=@_[1];$i++) {
      ($value,$line)=split(' ',$line,2);
      $values[$i]=$value/$farads;
      $values[0]+=$value/$farads;
    }
}

# determine the capacitances per block
sub make_blocks {
  $count=1;
  foreach $Windows (@Windows) {
    $command = "blocks @_[1] -w$Windows";
    print `$command`;
    $file="@_[0]$count.out";
    $command = "calcap @_[1] -o$file -c@_[2] -tblock@_[3].tex";
    @start = times;
    print `$command`;
    @end = times;
    $time="time".$count++.".out";
    find_max(@_[0]);
    write_time($time,$max/$byte);
  }
}

# determine the capacitances of the circuit
sub make_fastcap {
  if ($input =~ m/(.lst)/) {$command="fastcap -l@_[1]";}
  else {$command = "fastcap @_[1]";}
  $file="g_".@_[0].".out";
  open (file,">$file");
  @start=times;
  print file `$command`;
  @end = times;
  close file;

  open (file,">fast.names");
  print file "g_".@_[0]."\n";
  close file;
  $command = "matrix fast.names -c@_[2]";
  $file=@_[0]."0.out";
  open (file,">$file");
  print file `$command`;
  close file;

  $time="time0.out";
```

```
  find_max("fast");
  def_byte($max);
  write_time($time,$max/$byte);
}

# make windows with window width 1,2,3,4,...,maxwin
sub make_windows1 {
  for ($max=@_[0]/@_[1]-2;$max>-1;$max--) {
    $Windows[$max]=@_[2]/($max+2)*@_[1];
  }
}

## make windows with window width A/2,A/3,A/4,...,A/maxwin.
## A the width of the circuit
#sub make_windows1 {
#  for ($max=@_[0]/@_[1]-2;$max>-1;$max--) {
#    $Windows[$max]=@_[2]/($max+2)*@_[1];
#  }
#}

# write the output of the blocks
sub write_blocks {
  foreach $Windows (@Windows) {
    $file="time$Windows.out";
    open (file,"$file");
    if ($line=<file>) {chop($line);($time1,$time2,$mem)=split(' ',$line,3);}
    close file;

    $file=$file_name.$Windows.".out";
    find_values($file,@_[0]);
    $window=$Windows;

    open (file,">>@_[1]");
    printf file "blocks&%.2f&%.2f&%.2f&%.2f&%.3f",
                 $window,$time1,$time2,$time1+$time2,$mem;
    for $values (@values)
    {
      printf file "&%.3f",$values;
    }
    print file "\\\\ \n"
  }
  print file " \\hline \n\\end{tabular}\\end{table}\n\n\\end{document}";
  close file;
}

# write the output of fastcap
sub write_fastcap {
  $file="time0.out";
  open (file,"$file");
  if ($line=<file>) {chop($line);($time1,$time2,$mem)=split(' ',$line,3);}
  close file;

  $file=$file_name."0.out";
```

```
  find_values($file,@_[0]);

  open (file,">>@_[1]");
  printf file "fastcap&&%.2f&%.2f&%.2f&%.3f",$time1,$time2,$time1+$time2,$mem;
  for $values (@values)
  {
    printf file "&%.3f",$values;
  }
   print file "\\\\\n";
  close file;
}

sub write_help {
  printf "\nUsage : proef [<inputfile>] [-o<outputfile>] [-t<texfile>]\n";
  printf "              [-m<maximum windowsize>] [-u<exp windowsize>]\n";
  printf "              [-c<#conductors>]\n\n";
  exit 0;
}

# write a texfile of the output, generated by fastcap and blocks
sub write_texfile {
  open(file,"@_[2]");
  ($string=$line)=~s/^\S+// if ($line=<file>);
  chop($string);
  close file;
  open(file,">@_[1]");
  print file "\\documentclass[a4paper,landscape]{report}\n\n";
  print file "\\begin{document}\n\n\\begin{table}[h]\n\\centering\n";
  print file "\\caption{$string} \n";
  print file "\\begin{tabular}{|l|c|ccc|c|c";
  for ($i=1;$i<=$conductor;$i++) {
  print file "c";}
  print file "|} \\hline\n";
  $column=$conductor+1;
  print file "program&w&\\multicolumn{3}{c|} {time}&memory&\\multicolumn
             {$column}{c|} {Capacitances ($name)} \\\\\n";
  print file "&($winsize)&user&system&CPU&($memsize)&".
            '$C_{S\\,1}$&$C_{1\\, gnd}$';
  for ($i=2;$i<=$conductor;$i++)
  {
  print file '&$C_{1\\,'.$i.'}$';
  }
  print file "\\\\ \\hline\n";
  close file;

  write_fastcap(@_[0],@_[1]);
  write_blocks(@_[0],@_[1]);
  $command="rm *_*.qui *_*.lst *_*.out";
  print `$command`;
}

sub write_time {
  open(file,">@_[0]");
```

```
  printf file @end[2]-@start[2]."\t";
  printf file @end[3]-@start[3]."\t";
  printf file @_[1]."\n";
  close file;
}

if ($#ARGV < 5) {write_help;}
elsif ($#ARGV > 6) {write_help;}
def_arguments(@ARGV);

make_windows($maxwin,$unity,$conductor+1);
# $conductor+1 (the width of the circuit)
make_fastcap($file_name,$input,$conductor);
make_blocks($file_name,$input,$conductor,$delers);
def_farad($file_name);
write_texfile($conductor,$texfile,$input);
clean_dir();
```

A.2 C-source code

There are two programs written in C. The first is blocks, which generates the blocks of the layout partition. The second is matrix, which combines the capacitance matrices of each block into one capacitance matrix for the integrated circuit.

A.2.1 Blocks

The following source-codes help to generate the blocks:

size which determines the measures of the layout and the number of conductors.

fileread reads the input file.

createWindow generates the windows of the layout.

createBlock generates the blocks with the help of the windows.

Mainfile

```
/* BLOCKS.H */
#define LENGTHSTRING 10000

struct Coordinate
{
  float x;
  float y;
  float z;
};

extern float lengthx,lengthy,windowsize;
extern char file_name[];

/* BLOCKS.C */
#include <stdio.h>
#include <stdlib.h>
#include "createBlock.h"
```

```
#include "createWindow.h"
#include "size.h"

#define DEFNWIN 3
#define TRUE 1
#define FALSE 0

/* There are four global variables, used all over the program */
float windowsize,lengthx,lengthy;
char file_name[100];

/*
   blocks.c creates the block partition of the layout.
   input : The file containing the panels of the layout.
           The window width.
   output: Files containing the windowpanels (windowfiles).
           Files containing a list of windows (blockfiles).
           A file containing the names of the blockfiles.
*/
int main(int argc,char *argv[])
{
  char **chkp,*chk;
  FILE *file_descriptor;
  int index,cmderr,i,length;
  long strtol();

  /* defaults */
  cmderr=FALSE;
  lengthx=lengthy=windowsize=0.0;

  if (argc != 3) {
    fprintf(stderr,"%s: the number of arguments is incorrect.\n",argv[0]);
    cmderr=TRUE;
  }

  /* Determine the arguments */
  for (index=1;index<argc && cmderr==FALSE;index++)

    /* Read the window width */
    if (argv[index][0]=='-' && argv[index][1]=='w') {
      windowsize=atof(&argv[index][2]);
    }

    /* Read the name of the inputfile and open the inputfile */
    else {
      file_descriptor=fopen(&argv[index][0],"r");
      if (file_descriptor == NULL) {
        fprintf(stderr,"%s: inputfile does not exist or user is not permitted
                        to access.\n", &argv[index][0]);
        cmderr=TRUE;
        break;
      }
```

```
      /* remove the extension of the filename
         The filename is used for creating the filenames of the
         windowfiles and the blockfiles */
      strcpy(file_name,&argv[index][0]);
      length=strlen(file_name);
      for (i=0;((i<length) && (file_name[i]!='.'));i++);
      file_name[i]=0;
    }

  if (cmderr==TRUE)
  {
    fprintf(stderr,"Usage : %s [<inputfile>] [-w<windowsize>]\n",argv[0]);
    return 0;
  }

  /* Determine the length, the width and the number of conductors of the layout */
  determine_measures(&lengthx,&lengthy);
  if (lengthx<windowsize || lengthy<windowsize) {
    fprintf(stderr,"\n The width of the window exceeds
                       the width of the circuit.\n\n");
    return 0;
  }

  /* create the windows and place the panels in the associated windows */
  make_windows();

  /* create the blocks */
  generate_blocks();

  return 0;
}
```

size

```
/* SIZE.H */
extern void determine_measures(float *lengthx,float *lengthy);

/* SIZE.C */
#include <stdio.h>
#include <stdlib.h>
#include "blocks.h"
#include "fileRead.h"

/*
  Find the maximum value of the coordinates and the current length
  input : The coordinates of the panel and the current length of the layout.
  output: The maximum of the input variabels.
*/
float max(float *length,double x1,double x2,double x3,double x4)
{
  if ((x1>*length) || (x2>*length) || (x3>*length) || (x4>*length))
  {
    if ((x1>x2) && (x1>x3) && (x1>x4))
```

```
      *length=x1;
    else if ((x2>x3) && (x2>x4))
      *length=x2;
    else if (x3>x4)
      *length=x3;
    else
      *length=x4;
  }
  return *length;
}

/*
  Find the maximum of the coordinates of the panel and the size of the layout
  till now.
  input : coordinates panel, size of the layout up to now.
  output: The new size of the layout.
*/
void determine_Q_max(float *lengthx,float *lengthy,
                     struct Coordinate *coord1,struct Coordinate *coord2,
                     struct Coordinate *coord3,struct Coordinate *coord4)
{
  *lengthx=max(lengthx,coord1->x,coord2->x,coord3->x,coord4->x);
  *lengthy=max(lengthy,coord1->y,coord2->y,coord3->y,coord4->y);
}

/*
  Find the maximum of the coordinates of the panel and the size of the layout
  till now.
  input : coordinates panel, size of the layout up to now.
  output: The new size of the layout.
*/
void determine_T_max(float *lengthx,float *lengthy,struct Coordinate *coord1,
                     struct Coordinate *coord2,struct Coordinate *coord3)
{
  *lengthx=max(lengthx,coord1->x,coord2->x,coord3->x,0.0);
  *lengthy=max(lengthx,coord1->y,coord2->y,coord3->y,0.0);
}

/*
   verify that the windowwidth is no problem when the windows and blocks are created
   input : size of the layout and the windowsize.
   output: An adjusted windowsize (windowsize*1.001)
*/
void window_size()
{
  float lx,ly,window;
  int ix,iy;

  lx=lengthx/windowsize;ly=lengthy/windowsize;
  ix=lengthx/windowsize;iy=lengthy/windowsize;
  if ((lx-ix==0) || (ly-iy==0))
  {
    printf("With window width %.2e there are panels
```

```
            that can not be associated with any window. ",windowsize);
    printf("Therefore the window width will be multiplied by 1.001\n");
    window=windowsize;
    windowsize*=1.001;
    lx=lengthx/windowsize;ly=lengthy/windowsize;
    ix=lengthx/windowsize;iy=lengthy/windowsize;
  }
  if ((lx-ix==0) || (ly-iy==0))
  {
    fprintf(stderr,"Choose a different window width then %.2e",window);
    exit(0);
  }
}

/*
  Find the the size of the layout and the number of conductors.
  input : File containing all panels.
  output: The size of the layout and the number of panels.
*/
void determine_measures(float *lengthx,float *lengthy)
{
  char *string=malloc(LENGTHSTRING);
  char name[20],conductorname[20],temp_file_name[100];
  FILE *file_descriptor;
  int conductor=0;
  struct Coordinate coord1,coord2,coord3,coord4;

  sprintf(temp_file_name,"%s.qui",file_name);
  file_descriptor=fopen(temp_file_name,"r");
  while (!feof(file_descriptor))
  {
    fgets(string,LENGTHSTRING,file_descriptor);
    if (string[0]=='Q')
    {
      strcpy(name,conductorname);
      read_Q_coordinates(string,conductorname,&coord1,&coord2,&coord3,&coord4);
      if (strcmp(conductorname,name)!=0)
         conductor+=1;
      determine_Q_max(lengthx,lengthy,&coord1,&coord2,&coord3,&coord4);
    }
    else if (string[0]=='T')
    {
      strcpy(name,conductorname);
      read_T_coordinates(string,conductorname,&coord1,&coord2,&coord3);
      if (strcmp(conductorname,name)!=0)
         conductor+=1;
      determine_T_max(lengthx,lengthy,&coord1,&coord2,&coord3);
    }
  }
  fclose(file_descriptor);
  free(string);

  window_size();
```

```
  printf("width circuit  :%e\n",*lengthx);
  printf("length circuit :%e\n",*lengthy);
  printf("window width   :%e\n",windowsize);
  printf("# conductors   :%d\n",conductor);
}
```

fileread

```
/* FILEREAD.H */
extern void read_Q_coordinates(char *string,char *dummi,
                   struct Coordinate *coord1,struct Coordinate *coord2,
                   struct Coordinate *coord3,struct Coordinate *coord4);
extern void read_T_coordinates(char *string,char *dummi,
                   struct Coordinate *coord1,struct Coordinate *coord2,
                   struct Coordinate *coord3);

/* FILEREAD.C */
#include <stdio.h>
#include <stdlib.h>
#include "blocks.h"
#include "fileRead.h"

/*
   read the panel coordinates
   input : the string containing the panel
   output: the coordinates of the panel
*/
void read_Q_coordinates(char *string,char *dummi,
                        struct Coordinate *coord1,struct Coordinate *coord2,
                        struct Coordinate *coord3,struct Coordinate *coord4)
{
  char letter[10];

  sscanf(string,"%s %s %e %e %e %e %e %e %e %e %e %e %e %e",letter,dummi,
              &coord1->x,&coord1->y,&coord1->z,&coord2->x,&coord2->y,&coord2->z,
              &coord3->x,&coord3->y,&coord3->z,&coord4->x,&coord4->y,&coord4->z);
}

/*
   read the panel coordinates
   input : the string containing the panel
   output: the coordinates of the panel
*/
void read_T_coordinates(char *string,char *dummi,struct Coordinate *coord1,
                        struct Coordinate *coord2,struct Coordinate *coord3)
{
  char letter[10];

  sscanf(string,"%s %s %e %e %e %e %e %e %e %e %e %e %e",letter,dummi,
                &coord1->x,&coord1->y,&coord1->z,&coord2->x,&coord2->y,&coord2->z,
                &coord3->x,&coord3->y,&coord3->z);
}
```

createWindow

```
/* CREATEWINDOW.H */
extern void make_windows();

/* CREATEWINDOW.C */
#include <stdio.h>
#include <stdlib.h>
#include "blocks.h"
#include "fileRead.h"

/*
   place the panels into the associated windows
   input : The string containing the panel
   output: The panel is placed into the associated window
*/
void writequad(char *string)
{
  char temp_file_name[100],dummi[20];
  FILE *file_descriptor;
  float weigthx,weigthy;
  int x,y;
  struct Coordinate coord1,coord2,coord3,coord4;

  /* determine the weigth of the panel */
  read_Q_coordinates(string,dummi,&coord1,&coord2,&coord3,&coord4);
  weigthx=((coord1.x) + (coord2.x) + (coord3.x) + (coord4.x))/4.0;
  weigthy=((coord1.y) + (coord2.y) + (coord3.y) + (coord4.y))/4.0;

  /* determine the lower left corner of the window */
  x=weigthx/windowsize;
  y=weigthy/windowsize;

  /* place the panel in the window with lower left corner (x,y) */
  sprintf(temp_file_name,"%s_%d_%d.qui",file_name,x,y);
  file_descriptor=fopen(temp_file_name,"a+");
  fprintf(file_descriptor,"%s",string);
  fclose(file_descriptor);
}

/*
   place the panels into the associated windows
   input : The string containing the panel
   output: The panel is placed into the associated window
*/
void writetetra(char *string)
{
  char temp_file_name[100],dummi[20];
  FILE *file_descriptor;
  float weigthx,weigthy;
  int x,y;
  struct Coordinate coord1,coord2,coord3;

  /* determine the weigth of the panel */
```

```
  read_T_coordinates(string,dummi,&coord1,&coord2,&coord3);
  weigthx=((coord1.x) + (coord2.x) + (coord3.x))/3.0;
  weigthy=((coord1.y) + (coord2.y) + (coord3.y))/3.0;

  /* determine the lower left corner of the window */
  x=weigthx/windowsize;
  y=weigthy/windowsize;

  /* place the panel in the window with lower left corner (x,y) */
  sprintf(temp_file_name,"%s_%d_%d.qui",file_name,x,y);
  file_descriptor=fopen(temp_file_name,"a+");
  fprintf(file_descriptor,"%s\n",string);
  fclose(file_descriptor);
}

/*
   create and clean the windowfiles and place the title into the files
   input : The first line of the inputfile.
   output: All windowfiles are created and contain the title.
*/
void create_files(char *string)
{
  char temp_file_name[100];
  FILE *file_descriptor;
  int i,j;

  for(i=0;i<lengthx/windowsize;i++)
    for(j=0;j<lengthy/windowsize;j++)
    {
      sprintf(temp_file_name,"%s_%d_%d.qui",file_name,i,j);
      file_descriptor=fopen(temp_file_name,"w");
      fprintf(file_descriptor,"%s",string);
      fclose(file_descriptor);
    }
}

/*
   create the windowfiles and place the panels in the windowfiles.
   input : The file containing all panels
   output: The windowfiles containing the panels.
*/
void make_windows()
{
  char *string=malloc(LENGTHSTRING);
  char temp_file_name[100];
  FILE *file_descriptor;

  sprintf(temp_file_name,"%s.qui",file_name);
  file_descriptor=fopen(temp_file_name,"r");

  while (!feof(file_descriptor))
  {
    fgets(string,LENGTHSTRING,file_descriptor);
```

```
    if (string[0]=='O')
      create_files(string);
    else if (string[0]=='Q')
      writequad(string);
    else if (string[0]=='T')
      writetetra(string);
  }
  fclose(file_descriptor);
  free(string);
}
```

createBlock

```
/* CREATEBLOCK.H */
extern void generate_blocks();

/* CREATEBLOCKS.C */
#include <stdio.h>
#include <stdlib.h>
#include "blocks.h"

/*
   isempty checks if the file exists and is not empty
   input : The coordinates of the lower left corner of a window
   output: The window is empty (0) or the window is not empty (1)
*/
int isempty(int x,int y)
{
  char temp_file_name[100];
  FILE *file_descriptor1;
  char *string=malloc(LENGTHSTRING);

  /* Open the windowfile */
  sprintf(temp_file_name,"%s_%d_%d.qui",file_name,x,y);
  file_descriptor1=fopen(temp_file_name,"r");
  if (file_descriptor1==NULL)
     fprintf(stderr,"Problem opening file $s\n",temp_file_name);

  /* Read the lines of the file, stop by the end of the file or
     when the string contains a panel */
  fgets(string,LENGTHSTRING,file_descriptor1);
  while ((!feof(file_descriptor1)) && (string[0]!='Q') && (string[0]!='T'))
  {
    fgets(string,LENGTHSTRING,file_descriptor1);
  }
  fclose(file_descriptor1);
  free(string);

  if ((string[0]=='Q') || (string[0]=='T'))
     return 1;
  else
    return 0;
}
```

```
/*
   create_large_block create the large blocks, containing 2x2 windows,
   by placing the windows in a file
   input : the length of the layout = lengthx
           the width of the layout = lengthy
           the windowsize = windowsize
           the file containing the names of the blockfiles = names_fp
   output: all files that associate the windows with the large blocks.
*/
void create_large_block(FILE *names_fp)
{
  int x,y;
  int file1,file2,file3,file4;
  float nx,ny;
  FILE *file_descriptor;
  char temp_file_name[100];

  nx=lengthx/windowsize-1.0;
  ny=lengthy/windowsize-1.0;
  for(x=0;x<nx;x++)
    for(y=0;y<ny;y++)
      {
        /* determine if the files are not empty */
        file1=isempty(x,y);
        file2=isempty(x+1,y);
        file3=isempty(x,y+1);
        file4=isempty(x+1,y+1);
        if ((file1==1) || (file2==1) || (file3==1) || (file4==1))
        {
          sprintf(temp_file_name,"g_%s_%d_%d.lst",file_name,x,y);
          fprintf(names_fp,"g_%s_%d_%d\n",file_name,x,y);
          file_descriptor=fopen(temp_file_name,"w");
          if (file_descriptor==NULL)
            fprintf(stderr,"Problem opening file $s\n",temp_file_name);

          if (file1==1)
            if ((file2==1) || (file3==1) || (file4==1))
              fprintf(file_descriptor,"C %s_%d_%d.qui 1.0 0.0 0.0 0.0 +\n",
                      file_name,x,y);
            else
              fprintf(file_descriptor,"C %s_%d_%d.qui 1.0 0.0 0.0 0.0\n",
                      file_name,x,y);
          if (file2==1)
            if ((file3==1) || (file4==1))
              fprintf(file_descriptor,"C %s_%d_%d.qui 1.0 0.0 0.0 0.0 +\n",
                      file_name,x+1,y);
            else
              fprintf(file_descriptor,"C %s_%d_%d.qui 1.0 0.0 0.0 0.0\n",
                      file_name,x+1,y);
          if (file3==1)
            if (file4==1)
              fprintf(file_descriptor,"C %s_%d_%d.qui 1.0 0.0 0.0 0.0 +\n",
```

```
                    file_name,x,y+1);
              else
                fprintf(file_descriptor,"C %s_%d_%d.qui 1.0 0.0 0.0 0.0\n",
                        file_name,x,y+1);
            if (file4==1)
                fprintf(file_descriptor,"C %s_%d_%d.qui 1.0 0.0 0.0 0.0\n",
                        file_name,x+1,y+1);

            fclose(file_descriptor);
          }
        }
}

/*
   create_horizontal_block create the large blocks, containing 2x1 windows,
   by placing the windows in a file
   input : the length of the layout = lengthx
           the width of the layout = lengthy
           the windowsize = windowsize
           the file containing the names of the blockfiles = names_fp
   output: all files that associate the windows with the large blocks.
*/
void create_horizontal_block(FILE *names_fp)
{
  int x,y;
  int file1,file2;
  float nx,ny;
  FILE *file_descriptor;
  char temp_file_name[100];

  nx=(lengthx/windowsize)-1;
  ny=(lengthy/windowsize)-1;
  for(x=0;x<nx;x++)
    for(y=1;y<ny;y++)
       {
         /* determine if the files are not empty */
         file1=isempty(x,y);
         file2=isempty(x+1,y);
         if ((file1==1) || (file2==1))
         {
           sprintf(temp_file_name,"h_%s_%d_%d.lst",file_name,x,y);
           fprintf(names_fp,"h_%s_%d_%d\n",file_name,x,y);
           file_descriptor=fopen(temp_file_name,"w");
           if (file_descriptor==NULL)
             fprintf(stderr,"Problem opening file $s\n",temp_file_name);

           if (file1==1)
             if (file2==1)
               fprintf(file_descriptor,"C %s_%d_%d.qui 1.0 0.0 0.0 0.0 +\n",
                       file_name,x,y);
             else
               fprintf(file_descriptor,"C %s_%d_%d.qui 1.0 0.0 0.0 0.0\n",
                       file_name,x,y);
```

```
          if (file2==1)
            fprintf(file_descriptor,"C %s_%d_%d.qui 1.0 0.0 0.0 0.0\n",
                    file_name,x+1,y);

          fclose(file_descriptor);
        }
      }
}

/*
   create_vertical_block create the large blocks, containing 1x2 windows,
   by placing the windows in a file
   input : the length of the layout = lengthx
           the width of the layout = lengthy
           the windowsize = windowsize
           the file containing the names of the blockfiles = names_fp
   output: all files that associate the windows with the large blocks.
*/
void create_vertical_block(FILE *names_fp)
{
  int x,y;
  int file1,file2;
  float nx,ny;
  FILE *file_descriptor;
  char temp_file_name[100];

  nx=(lengthx/windowsize)-1;
  ny=(lengthy/windowsize)-1;
   for(x=1;x<nx;x++)
    for(y=0;y<ny;y++)
       {
         /* determine if the files are not empty */
         file1=isempty(x,y);
         file2=isempty(x,y+1);
         if ((file1==1) || (file2==1))
         {
           sprintf(temp_file_name,"v_%s_%d_%d.lst",file_name,x,y);
           fprintf(names_fp,"v_%s_%d_%d\n",file_name,x,y);
           file_descriptor=fopen(temp_file_name,"w");
           if (file_descriptor==NULL)
             fprintf(stderr,"Problem opening file $s\n",temp_file_name);

           if (file1==1)
             if (file2==1)
               fprintf(file_descriptor,"C %s_%d_%d.qui 1.0 0.0 0.0 0.0 +\n",
                       file_name,x,y);
             else
               fprintf(file_descriptor,"C %s_%d_%d.qui 1.0 0.0 0.0 0.0\n",
                       file_name,x,y);
          if (file2==1)
            fprintf(file_descriptor,"C %s_%d_%d.qui 1.0 0.0 0.0 0.0\n",
                    file_name,x,y+1);
```

```
          fclose(file_descriptor);
        }
      }
}

/*
   create_small_block create the large blocks, containing 1 window,
   by placing the window in a file
   input : the length of the layout = lengthx
           the width of the layout = lengthy
           the windowsize = windowsize
           the file containing the names of the blockfiles = names_fp
   output: all files that associate the windows with the large blocks.
*/
void create_small_block(FILE *names_fp)
{
  int x,y;
  float nx,ny;
  FILE *file_descriptor;
  char temp_file_name[100];

  nx=(lengthx/windowsize)-1;
  ny=(lengthy/windowsize)-1;
   for(x=1;x<nx;x++)
    for(y=1;y<ny;y++)
         if (isempty(x,y)==1)
         {
           sprintf(temp_file_name,"k_%s_%d_%d.lst",file_name,x,y);
           fprintf(names_fp,"k_%s_%d_%d\n",file_name,x,y);
           file_descriptor=fopen(temp_file_name,"w");
           if (file_descriptor==NULL)
             fprintf(stderr,"Problem opening file $s\n",temp_file_name);

           fprintf(file_descriptor,"C %s_%d_%d.qui 1.0 0.0 0.0 0.0\n",
                   file_name,x,y);
           fclose(file_descriptor);
         }
}


/*
   generate_blocks create the large, horizontal, vertical and small  blocks.
   input : the length of the layout = lengthx
           the width of the layout = lengthy
           the windowsize = windowsize
           the name of the inputfile = file_name
   output: all blockfiles that associate the windows with the blocks.
*/
void generate_blocks()
{
  char temp_file_name[100];
  FILE *names_fp;
```

```
  sprintf(temp_file_name,"%s.names",file_name);
  names_fp=fopen(temp_file_name,"w");

  create_large_block(names_fp);
  create_horizontal_block(names_fp);
  create_vertical_block(names_fp);
  create_small_block(names_fp);

  fclose(names_fp);
}
```

A.2.2 Matrix

The following source-code helps to generate the matrix:

readMatrix read the capacitance matrix of the output and places the matrices of each block into the general matrix for the integrated circuit.

Mainfile

```
/* MATRIX.H */

extern char file_name[30];
extern int conductor;

/* MATRIX.C */
#include <stdio.h>
#include <stdlib.h>
#include "readMatrix.h"
#include "matrix.h"

#define TRUE 1
#define FALSE 0

/* There are two global variables used all over the program */
char file_name[30];
int conductor;

/*
   matrix.c creates the short-circuit capacitance matrix of the entire layout
   input : The file containing the names of the blockfiles.
           Each blockfile contains a submatrix.
   output: the matrix of the layout
*/
void main(int argc,char *argv[])
{
  char **chkp,*chk;
  FILE *names_fp;
  int cmderr,index;
  long strtol();
  struct tmatrix amatrix,cmatrix;

  /* defaults */
  cmderr=FALSE;
```

```
conductor = 0;

/* parse command line */
chkp=&chk; /* pointers for error checking */

if (argc != 3) {
  fprintf(stderr,"%s : the number of arguments is incorrect.\n",argv[0]);
  cmderr=TRUE;
}

/* Determine the arguments */
for(index=1;index<argc && cmderr==FALSE;index++)

  /* Determine the number nof conductors */
  if (argv[index][0] == '-' && argv[index][1] == 'c') {
    conductor=(int) strtol(&(argv[index][2]),chkp,10);
    if (*chkp==&argv[index][2]) {
      cmderr=TRUE;
      break;
    }
  }

  /* Determine the filename of the file containing the names of the blockfiles */
  else {
    names_fp=fopen(&argv[index][0],"r");
    if (names_fp == NULL) {
      fprintf(stderr,"%s : inputfile %s does not exists or the user is not
                      permitted to access.\n",argv[0],&argv[index][0]);
      cmderr=TRUE;
      break;
    }
    strcpy(file_name,&argv[index][0]);
    create_file_name(file_name);
  }

if (cmderr==TRUE)
{
  fprintf(stderr,"Usage : %s [<inputfile>] [-c<#conductors>]\n\n",argv[0]);
  fprintf(stderr,"inputfile   = file containing the names of the blockfiles\n");
  fprintf(stderr,"#conductors = total number of conductors of the layout\n");
  exit(0);
}

/* create and clean the matrix */
create_matrix(&amatrix);

/* construct the matrix out the submatrices */
matrix(names_fp,&amatrix);

/* print the matrix into a file */
sort_matrix(&amatrix);
print_matrix(&amatrix);
```

```
  /* free the memory of the matrix */
  clean_matrix(&amatrix);

  fclose(names_fp);
  exit(0);
}
```

readMatrix

```
/* READMATRIX.H */
#define LENGTHSTRING 10000

struct tmatrix
{
  float *matrix;
  int length;
  char **names;
};

extern void create_file_name(char *file_name);
extern void create_matrix(struct tmatrix *matrix);
extern void clean_matrix(struct tmatrix *matrix);
extern void print_matrix(struct tmatrix *matrix);
extern void matrix(FILE *names_fp,struct tmatrix *amatrix);
extern void sort_matrix(struct tmatrix *amatrix);

/* READMATRIX.C */
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include "matrix.h"
#include "readMatrix.h"

char *expnames[]={"kilofarads","farads","millifarads", "microfarads",
  "nanofarads","picofarads","femtofarads","attofarads",NULL};

/*
   remove the extension of the filename
   input : filename = name.extension
   output: filename = name
*/
void create_file_name(char *file_name)
{
  int index,length;

  length=strlen(file_name);
  for (index=0;((index<length) && (file_name[index]!='.'));index++);
  file_name[index]=0;
}

/*
    To be able to associate rows of the submatrices with rows of the matrix
    each row is given a name.
```

```
    input : A conductorname.
            The row number.
            A matrix.
    output: The rowname = the conductorname
*/
void set_row_name(struct tmatrix *matrix,int row,char *name)
{
  free(matrix->names[row]);
  matrix->names[row]=malloc(strlen(name)+1);
  strcpy(matrix->names[row],name);
}

/*
  Allocate memory for the matrix
  input : The uninitialized matrix
  output: Memory is allocated for the matrix
*/
void clean_matrix(struct tmatrix *matrix)
{
  int row;

  for (row=0;row<matrix->length;row++)
    free(matrix->names[row]);
  free(matrix->matrix);
  free(matrix->names);
}

/*
   Allocate memory for the matrix and make the matrix a zero matrix.
   input : The matrix of the entire layout.
           The number of conductors.
   output: A zero matrix.
           Allocated memory for the rownames.
*/
void create_matrix(struct tmatrix *matrix)
{
  int column,row;

  /* Size matrix */
  matrix->matrix=malloc(conductor*conductor*sizeof(float));
  /* Length matrix */
  matrix->length=conductor;
  /* Names matrix */
  matrix->names=malloc(conductor*sizeof(char *));

  /* values matrix = 0 */
  for(column=0;column<conductor;column++)
    for(row=0;row<conductor;row++)
      matrix->matrix[row+column*conductor]=0.0;
  for(row=0;row<conductor;row++)
  {
    matrix->names[row]=malloc(1*sizeof(char));
    *(matrix->names[row])=0;
```

```
  }
}

/*
   Print the matrix to an outputfile (file_name.out)
   input : The matrix.
   output: A file containing the matrix.
*/
void print_matrix(struct tmatrix *matrix)
{
  int row,column;

  printf("CAPACITANCE MATRIX, farads\n");
  printf("          ");                    /* print ten spaces */
  for (row=0;row<matrix->length;row++)
    printf("%10d  ",row+1);
  printf("\n");
  for (row=0;row<matrix->length;row++)
  {
    printf("%10s  %2d  ",matrix->names[row],row+1);
    for (column=0;column<matrix->length;column++)
        printf("% 2.3e  ",matrix->matrix[row+column*(matrix->length)]);
    printf("\n");
  }
}

/*
   Read the submatrix of the outputfile of a block (file_name_out.out)
   input : the outputfile.
   output: the submatrix.
*/
void read_matrix(struct tmatrix *matrix,char *file_name_out,FILE *file_descriptor)
{
  char *string=malloc(LENGTHSTRING),farads[20];
  float capacitance,exponent;
  int expnum=0,length=0,dummi=0,row=0,column=0,conductor1=0;

  /* find the place of the shortcircuit matrix in the file */
  fgets (string,LENGTHSTRING,file_descriptor);
  while (strncmp(string,"CAPACITANCE MATRIX",18)) {
    fgets(string,LENGTHSTRING,file_descriptor);
  }

  /* determine the exponent of the entries of the shortcircuit matrix */
  sscanf(&string[20],"%s",farads);
  for (expnum=0;expnames[expnum]!=NULL;expnum++)
    if (strcmp(farads,expnames[expnum])==0)
      exponent=pow(1e3,(1-expnum));

  /* Read the last number on the line, the last number on the line is the
     number of conductors in the file */
  fgets(string,LENGTHSTRING,file_descriptor);
  length=strlen(string);
```

```
  for(;((length>=0) &&
       ((string[length]<'0') || (string[length]>'9')));length--);
  for(;((length>=0) &&
       ((string[length]>='0') && (string[length]<='9')));length--);

  length++;
  sscanf(string+length,"%d",&conductor1);

  /* free memory for the submatrix */
  matrix->matrix=malloc(conductor1*conductor1*sizeof(float));
  /* set length of the submatrix */
  matrix->length=conductor1;
  /* free memory for the row names */
  matrix->names=malloc(conductor1*sizeof(char *));

  /* read the values of the submatrix */
  for (row=0;row<conductor1;row++)
    {
       fscanf(file_descriptor,"%s",string);
       matrix->names[row]=malloc(strlen(string)+1);
       strcpy(matrix->names[row],string);

       fscanf(file_descriptor,"%d",&dummi);
       for (column=0;column<conductor1;column++)
       {
         fscanf(file_descriptor,"%f",&capacitance);
         matrix->matrix[row+column*conductor1]=capacitance*exponent;
       }
    }
  free(string);
}

/*
   place the submatrix in the  matrix
   input : the submatrix of a block
   output: the matrix of the layout
*/
void stamp_matrix(struct tmatrix *matrix, struct tmatrix *amatrix,float sign)
{
  int arow,row,column,*library;

  /* find for each row of the submatrix the place in the matrix of the layout
     library forms a lookup table */
  library=malloc(matrix->length*sizeof(int));
  for (row=0;row<matrix->length;row++)
  {
    for (arow=0;((arow<amatrix->length) && (strlen(amatrix->names[arow])!=0) &&
                 (strcmp(matrix->names[row],amatrix->names[arow])!=0));arow++);
    if (arow==amatrix->length)
    {
      printf("error: no matching names in matrices and now room to add new name.\n");
      exit(0);
    }
```

```
    else if (strlen(amatrix->names[arow])==0)
      set_row_name(amatrix,arow,matrix->names[row]);
    library[row]=arow;
  }

  /* place the submatrix in the matrix of the layout */
  for (row=0;row<matrix->length;row++)
    for (column=0;column<matrix->length;column++)
      amatrix->matrix[library[row]+library[column]*(amatrix->length)]+=
                           sign*matrix->matrix[row+column*(matrix->length)];

  free(library);
}

void add_to_matrix(struct tmatrix *matrix, struct tmatrix *amatrix)
{
  stamp_matrix(matrix,amatrix,1.0);
}

void subtract_off_matrix(struct tmatrix *matrix, struct tmatrix *amatrix)
{
  stamp_matrix(matrix,amatrix,-1.0);
}

/*
   Read the blockfiles out the inputfile and place the submatrices of the
   blockfiles into the matrix for the entire layout
   input : The inputfile
   output: The matrix of the layout.
*/
void matrix(FILE *names_fp,struct tmatrix *amatrix)
{
  FILE *file_descriptor;
  char temp_file_name[30];
  char file_name_out[30];
  struct tmatrix matrix;

  while (!feof(names_fp))
  {
    fscanf(names_fp,"%s",file_name_out);
    sprintf(temp_file_name,"%s.out",file_name_out);

  /* determine the outputfile of a block */
    file_descriptor=fopen(temp_file_name,"r");
    if (file_descriptor==NULL) {
      fprintf(stderr,"The inputfile %s does not exists or the user is not
                      permitted to access\n",temp_file_name);
      exit(0);
    }

    /* get the submatrix */
    read_matrix(&matrix,temp_file_name,file_descriptor);
```

```
    /* Determine if the submatrix must be added or subtracted */
    switch(temp_file_name[0])
    {
      case 'g':
      case 'k': add_to_matrix(&matrix,amatrix);
      break;

      case 'h':
      case 'v': subtract_off_matrix(&matrix,amatrix);
    }

    /* free the used memory of the submatrix */
    clean_matrix(&matrix);
    fclose(file_descriptor);
  }
}

void swap_row_column(struct tmatrix *amatrix,int normal,int swap)
{
  int row,column;
  float dummi;
  char *library;

  for (column=0;column<amatrix->length;column++)
  {
    dummi=amatrix->matrix[normal+column*(amatrix->length)];
    amatrix->matrix[normal+column*(amatrix->length)]=
                       amatrix->matrix[swap+column*(amatrix->length)];
    amatrix->matrix[swap+column*(amatrix->length)]=dummi;
  }
  for (row=0;row<amatrix->length;row++)
  {
    dummi=amatrix->matrix[row+normal*(amatrix->length)];
    amatrix->matrix[row+normal*(amatrix->length)]=
                    amatrix->matrix[row+swap*(amatrix->length)];
    amatrix->matrix[row+swap*(amatrix->length)]=dummi;
  }

  library=amatrix->names[normal];
  amatrix->names[normal]=amatrix->names[swap];
  amatrix->names[swap]=library;
}

void sort_matrix(struct tmatrix *amatrix)
{
  int row,search,compare;

  for (row=0;row<amatrix->length;row++)
  {
    compare=amatrix->length+1;
    for (search=row;((search<amatrix->length) && (compare!=(row+1)));search++)
      sscanf(amatrix->names[search],"%d",&compare);
    if (compare==(row+1))
```

```
      swap_row_column(amatrix,search-1,row);
  }
}
```

A.3 Inputfile for *FastCap*

To give an example of an inputfile, a 2×1 crossing bus is used. There are two possible inputfiles a qui-file or a list-file. A qui-file contains a title followed by the panels.

```
0 <title>
* <comment>
Q <cond.name> <x1> <y1> <z1> <x2> <y2> <z2> <x3> <y3> <z3> <x4> <y4> <z4>
T <cond.name> <x1> <y1> <z1> <x2> <y2> <z2> <x3> <y3> <z3>
```

A list-file contains the lines to include the panels of a file in the list-file.

```
* <comment>
G <group name>
C <file> <outperm> <xtran> <ytran> <ztran> [+]
D <file> <outperm> <inperm> <xtran> <ytran> <ztran> <xref> <yref> <zref> [-]
B <file> <outperm> <inperm> <xtran> <ytran> <ztran> <xref> <yref> <zref> [-] [+]
```

The C line is used to include the panels in file as parts of conductor surfaces surrounded by dielectric with relative permittivity outperm. The D and B lines specify dielectric interfaces along two regions with permittivities outperm and inperm. The B specifies that the dielectric interface is coincident with the infinitesimally thin conductor, while D surfaces are dielectric interfaces alone.

A.3.1 Qui-file

```
0 1X1 bus crossing problem with 1m wires (n=1 e=0.1)
* origin = (0 0 0)
*
* Bar corners: (1 0 1) (1 3 1) (2 0 1) (1 0 0)
* Reference side 0
Q 1 1.00000e+00 0.00000e+00 1.00000e+00 2.00000e+00 0.00000e+00 1.00000e+00
    2.00000e+00 1.00000e+00 1.00000e+00 1.00000e+00 1.00000e+00 1.00000e+00
*
* Opposite reference side 0
Q 1 1.00000e+00 0.00000e+00 0.00000e+00 2.00000e+00 0.00000e+00 0.00000e+00
    2.00000e+00 1.00000e+00 0.00000e+00 1.00000e+00 1.00000e+00 0.00000e+00
*
* Reference side 0
Q 1 1.00000e+00 1.00000e+00 1.00000e+00 2.00000e+00 1.00000e+00 1.00000e+00
    2.00000e+00 2.00000e+00 1.00000e+00 1.00000e+00 2.00000e+00 1.00000e+00
*
* Opposite reference side 0
Q 1 1.00000e+00 1.00000e+00 0.00000e+00 2.00000e+00 1.00000e+00 0.00000e+00
    2.00000e+00 2.00000e+00 0.00000e+00 1.00000e+00 2.00000e+00 0.00000e+00
*
* Reference side 0
Q 1 1.00000e+00 2.00000e+00 1.00000e+00 2.00000e+00 2.00000e+00 1.00000e+00
    2.00000e+00 3.00000e+00 1.00000e+00 1.00000e+00 3.00000e+00 1.00000e+00
*
* Opposite reference side 0
Q 1 1.00000e+00 2.00000e+00 0.00000e+00 2.00000e+00 2.00000e+00 0.00000e+00
```

```
    2.00000e+00 3.00000e+00 0.00000e+00 1.00000e+00 3.00000e+00 0.00000e+00
*
* Reference side 1
Q 1 1.00000e+00 0.00000e+00 0.00000e+00 1.00000e+00 0.00000e+00 1.00000e+00
    1.00000e+00 1.00000e+00 1.00000e+00 1.00000e+00 1.00000e+00 0.00000e+00
*
* Opposite reference side 1
Q 1 2.00000e+00 0.00000e+00 0.00000e+00 2.00000e+00 0.00000e+00 1.00000e+00
    2.00000e+00 1.00000e+00 1.00000e+00 2.00000e+00 1.00000e+00 0.00000e+00
*
* Reference side 1
Q 1 1.00000e+00 1.00000e+00 0.00000e+00 1.00000e+00 1.00000e+00 1.00000e+00
    1.00000e+00 2.00000e+00 1.00000e+00 1.00000e+00 2.00000e+00 0.00000e+00
*
* Opposite reference side 1
Q 1 2.00000e+00 1.00000e+00 0.00000e+00 2.00000e+00 1.00000e+00 1.00000e+00
    2.00000e+00 2.00000e+00 1.00000e+00 2.00000e+00 2.00000e+00 0.00000e+00
*
* Reference side 1
Q 1 1.00000e+00 2.00000e+00 0.00000e+00 1.00000e+00 2.00000e+00 1.00000e+00
    1.00000e+00 3.00000e+00 1.00000e+00 1.00000e+00 3.00000e+00 0.00000e+00
*
* Opposite reference side 1
Q 1 2.00000e+00 2.00000e+00 0.00000e+00 2.00000e+00 2.00000e+00 1.00000e+00
    2.00000e+00 3.00000e+00 1.00000e+00 2.00000e+00 3.00000e+00 0.00000e+00
*
* End faces
Q 1 2.00000e+00 0.00000e+00 0.00000e+00 2.00000e+00 0.00000e+00 1.00000e+00
    1.00000e+00 0.00000e+00 1.00000e+00 1.00000e+00 0.00000e+00 0.00000e+00
*
Q 1 2.00000e+00 3.00000e+00 0.00000e+00 2.00000e+00 3.00000e+00 1.00000e+00
    1.00000e+00 3.00000e+00 1.00000e+00 1.00000e+00 3.00000e+00 0.00000e+00
*
*
* Bar corners: (0 1 2) (0 2 2) (3 1 2) (0 1 3)
* Reference side 0
Q 2 0.00000e+00 1.00000e+00 2.00000e+00 1.00000e+00 1.00000e+00 2.00000e+00
    1.00000e+00 2.00000e+00 2.00000e+00 0.00000e+00 2.00000e+00 2.00000e+00
*
* Opposite reference side 0
Q 2 0.00000e+00 1.00000e+00 3.00000e+00 1.00000e+00 1.00000e+00 3.00000e+00
    1.00000e+00 2.00000e+00 3.00000e+00 0.00000e+00 2.00000e+00 3.00000e+00
*
* Reference side 0
Q 2 1.00000e+00 1.00000e+00 2.00000e+00 2.00000e+00 1.00000e+00 2.00000e+00
    2.00000e+00 2.00000e+00 2.00000e+00 1.00000e+00 2.00000e+00 2.00000e+00
*
* Opposite reference side 0
Q 2 1.00000e+00 1.00000e+00 3.00000e+00 2.00000e+00 1.00000e+00 3.00000e+00
    2.00000e+00 2.00000e+00 3.00000e+00 1.00000e+00 2.00000e+00 3.00000e+00
*
* Reference side 0
Q 2 2.00000e+00 1.00000e+00 2.00000e+00 3.00000e+00 1.00000e+00 2.00000e+00
```

```
    3.00000e+00 2.00000e+00 2.00000e+00 2.00000e+00 2.00000e+00 2.00000e+00
*
* Opposite reference side 0
Q 2 2.00000e+00 1.00000e+00 3.00000e+00 3.00000e+00 1.00000e+00 3.00000e+00
    3.00000e+00 2.00000e+00 3.00000e+00 2.00000e+00 2.00000e+00 3.00000e+00
*
* Reference side 1
Q 2 0.00000e+00 1.00000e+00 3.00000e+00 1.00000e+00 1.00000e+00 3.00000e+00
    1.00000e+00 1.00000e+00 2.00000e+00 0.00000e+00 1.00000e+00 2.00000e+00
*
* Opposite reference side 1
Q 2 0.00000e+00 2.00000e+00 3.00000e+00 1.00000e+00 2.00000e+00 3.00000e+00
    1.00000e+00 2.00000e+00 2.00000e+00 0.00000e+00 2.00000e+00 2.00000e+00
*
* Reference side 1
Q 2 1.00000e+00 1.00000e+00 3.00000e+00 2.00000e+00 1.00000e+00 3.00000e+00
    2.00000e+00 1.00000e+00 2.00000e+00 1.00000e+00 1.00000e+00 2.00000e+00
*
* Opposite reference side 1
Q 2 1.00000e+00 2.00000e+00 3.00000e+00 2.00000e+00 2.00000e+00 3.00000e+00
    2.00000e+00 2.00000e+00 2.00000e+00 1.00000e+00 2.00000e+00 2.00000e+00
*
* Reference side 1
Q 2 2.00000e+00 1.00000e+00 3.00000e+00 3.00000e+00 1.00000e+00 3.00000e+00
    3.00000e+00 1.00000e+00 2.00000e+00 2.00000e+00 1.00000e+00 2.00000e+00
*
* Opposite reference side 1
Q 2 2.00000e+00 2.00000e+00 3.00000e+00 3.00000e+00 2.00000e+00 3.00000e+00
    3.00000e+00 2.00000e+00 2.00000e+00 2.00000e+00 2.00000e+00 2.00000e+00
*
* End faces
Q 2 0.00000e+00 2.00000e+00 3.00000e+00 0.00000e+00 2.00000e+00 2.00000e+00
    0.00000e+00 1.00000e+00 2.00000e+00 0.00000e+00 1.00000e+00 3.00000e+00
*
Q 2 3.00000e+00 2.00000e+00 3.00000e+00 3.00000e+00 2.00000e+00 2.00000e+00
    3.00000e+00 1.00000e+00 2.00000e+00 3.00000e+00 1.00000e+00 3.00000e+00
```

A.3.2 Example of a List-file

The "+" causes all the panels to be grouped together and treated as a single conductor.

```
*
* 1x1 bus crossing problem with dielectric on lower conductors
*
* conductor to air interfaces
C cond_air_1x1.qui 1.0 0.0 0.0 0.0 +
*
* conductor to dielectric interfaces
C cond_dielec_1x1.qui 7.5 0.0 0.0 0.0
```