TR diss 2021 S

Stellingen
behorende bij het proefschrift
Accurate and Efficient Layout Extraction
door
N.P. van der Meijs

1. De in dit proefschrift beschreven technieken voor het modelleren en bepalen van capaciteiten in VLSI layouts kunnen ook hun nut bewijzen bij het ontwikkelen en optimaliseren van IC fabricageprocessen, daar ze toelaten direct het effect van schaling en andere geometrische veranderingen op de prestatie van een schakeling te kwantificeren.

2. Vanwege de fantastische snelheid en geheugencapaciteit van moderne computers, kan bijna ieder probleem uit de lineaire analyse met een redelijke mate van nauwkeurigheid opgelost worden.
[R.F. Harrington, Field Computation by Moment Methods, Macmillan, 1968.]

3. Bij het versnellen van computers wordt de nieuwe maximale grootte van een probleem bepaald door de tijd-complexiteit van het gebruikte algoritme.

4. Lazy deletion is een praktische, efficiënte heuristiek voor het union-find-delete probleem.
[N.P. van der Meijs and A.J. van Genderen, Space-Efficient Extraction Algorithms, accepted for publication: Proc. IEEE 3rd European Design Automation Conference, Brussels, Belgium, March 1992.]

5. Gegeven een IC interconnectie die geen andere interconnecties kruist, met een dikte t, hoogte boven het substraat h, totale omtrek P en bodem-oppervlakte A, dan wordt zijn totale kortsluitcapaciteit C_s goed benaderd door

$$C_s = A \times C_a + P \times C_e,$$

met

$$C_a = \varepsilon/h \qquad \text{de bodem-capaciteit,}$$

$$C_e = \varepsilon\{1.08 + 0.53\sqrt{t/h}\} \qquad \text{de rand-capaciteit.}$$

[N.P. van der Meijs and J.T. Fokkema, VLSI Circuit Reconstruction from Mask Topology, INTEGRATION, the VLSI Journal 2 (1984), pp. 85-119.]

6. Het feit dat software zo gemakkelijk verbeterd kan worden, in vergelijking met bijvoorbeeld geïntegreerde hardware, vormt een belangrijke oorzaak voor de slechte kwaliteit van veel softwaresystemen.

7. Zogenaamde "profilers" zijn noodzakelijke hulpmiddelen om programmatuur te optimaliseren, omdat keer op keer blijkt dat de rekentijd anders besteed wordt dan in eerste instantie wordt aangenomen.

8. Een belangrijke oorzaak van het slecht functioneren van veel administratieve en bestuurlijke systemen, is een fenomeen dat in de regeltechniek bekend staat als "dode tijd".

9. Door het elimineren van de drempel opgeworpen door hoge zetkosten, is de laserprinter een van de belangrijkste oorzaken van de explosieve toename van het aantal technisch-wetenschappelijke boeken.

10. Een college "Datastructuren en Algoritmen voor CAD van IC's" is nuttig voor een grote groep studenten als het de nadruk legt op de probleem-aanpak, daar deze een voorbeeld kan zijn voor het oplossen van een veelheid van andere technisch-wetenschappelijke analyse-, synthese- en modelleringsproblemen.

544603
3172554
TR diss 2021

TR diss
2021

Accurate and Efficient Layout Extraction

Accurate and Efficient Layout Extraction

Proefschrift

ter verkrijging van de graad van doctor
aan de Technische Universiteit Delft, op gezag van de
Rector Magnificus, prof. drs. P.A. Schenck,
in het openbaar te verdedigen ten overstaan van een
commissie aangewezen door het College van Dekanen
op maandag 27 januari 1992 te 14.00 uur

door

Nicolaas Petrus van der Meijs

geboren te Maasland
elektrotechnisch ingenieur

Dit proefschrift is goedgekeurd door de promotor
prof. dr. ir. P.M. Dewilde

CIP-DATA KONINKLIJKE BIBLIOTHEEK, DEN HAAG

Meijs, Nicolaas Petrus van der

Accurate and efficient layout extraction / Nicolaas Petrus van der
Meijs. - [S.l. : s.n.]. - Ill.
Thesis Delft University of Technology. - With ref. - With summary in
Dutch.
ISBN 90-9004806-5
Subject heading: computer aided circuit design / physical IC design
verification / IC interconnection modeling.

CONTENTS

1. Introduction

1.1 The Extraction Problem

Extraction is defined in this dissertation as ''the modeling and determination of electrical characteristics of integrated circuits, given their layout and relevant data concerning the fabrication process''. The resulting model is in the form of an equivalent circuit, consisting of active devices and passive circuit elements. The correctness of the layout can then be verified before fabrication, for example by simulating the equivalent circuit or by static (timing) analysis.

This verification step is becoming increasingly important. With the on-going decrease in feature size and reduction of switching times, the electrical behavior of advanced integrated circuits is being determined more and more by unintentional, parasitic elements. These elements include parasitic active devices such as thyristor structures in CMOS that cause latchup problems, capacitances, resistances and inductances associated with the wires on the chip and substrate resistance.

In this dissertation, we focus on obtaining accurate models that reliably predict the electrical behavior of integrated circuits: new techniques are required to capture effects that were previously not important or for which standard techniques cannot obtain sufficient accuracy.

Not only should the resulting models be accurate—they should also be efficient. That is, they must be as simple and compact as possible and capture all relevant effects while omitting irrelevant detail. For example, the models should not contain small coupling capacitances between distant features. However, the total of all these small couplings may have a non-negligible influence on the delay of the circuit. Therefore, small coupling capacitances should not be neglected altogether, but should be collected with the other capacitances instead. This point will be made precise in the course of the work.

Like the resulting models, the algorithms used to determine them must also be efficient. The CPU time needed by the algorithms as a function of the problem size (their time complexity), must be as low as possible. In particular, the suitability of algorithms requiring computation times greater than linear in the size of the problem is easily defeated by the exponential growth of the complexity of IC's—the number of features on a chip doubles approximately every two years. The increased speed of computers is generally not much help: if we account for the increased speed by assuming a constant ratio of computer speed and problem size, the computation time of a polynomial-time algorithm continues to grow exponentially! Therefore, in this dissertation, we emphasize linear-time algorithms.

Moreover, the amount of core memory needed by the algorithms (their space complexity) must also be as low as possible. This is perhaps even more important than their time complexity. For non-real time problems, computation time is theoretically unbounded, despite the foregoing discussion on time complexity. In practice, the size of the largest design that can be handled is often hard-limited by available memory. This is also true in a virtual memory environment, because even virtual memory is bounded and because *thrashing* can severely degrade performance up to the point where the effective throughput becomes (nearly) zero. When the exponential growth of the number of features on an IC is considered with respect to time, it is essential to have a sublinear space complexity of extraction algorithms.

Layout verification is most effective when it is part of the *design loop*. This is the only way to avoid costly redesigns because of problems that are detected late when relying on difficult-to-use, standalone verification tools. Therefore, the algorithms must be integrated in a layout-to-circuit extractor that is readily available on the designer's workstation and fits well into the design flow as a user-friendly tool. It must be easy to use even by designers having no knowledge of the theory behind the program or how it works internally.

An important consideration is also the compatibility of the various algorithms that determine specific components of the resulting equivalent circuit. For example, interconnect resistance and capacitance are determined by two completely different methods, yet these methods must be coupled so as to create a consistent lumped model that accurately describes the distributed RC characteristics of the interconnects.

1.2 Overview and Summary

This dissertation is structured and can be summarized as follows:

In Chapter 2, we study the electrical behavior of IC interconnections. This behavior can generally be described in terms of distributed resistance, capacitance and inductance, and we analyze which simplifications are possible under which circumstances. We also project these results onto future needs by considering the scaling behavior. One conclusion of Chapter 2 is that while interconnect capacitance is important, the classical estimation techniques for it are not sufficiently accurate for state-of-the-art technologies and critical designs.

In Chapter 3, we consider the problem of efficiently handling and manipulating VLSI/ULSI layout data, specifically with a view towards layout-to-circuit extraction. We develop a combination of the so-called corner stitching data structure and the scanline technique, which achieves an expected-case linear time and sublinear space complexity.

In Chapter 4, we discuss the mathematics and, briefly, the theoretical background of a boundary element method for accurate computation of the interconnect capacitances of (critical parts of) integrated circuits. Using a new algorithm for approximating the inverse of a matrix, we realize a linear time complexity and a constant space complexity. The resulting capacitance network accurately reflects the total capacitive load of all signals, without containing irrelevant, small capacitances between distant features.

In Chapter 5, we take a look at how the mathematical concepts and tools developed in Chapter 4 can be implemented in a practical and efficient program that puts their modeling power under the fingertips of the designer. We develop detailed algorithms and computational procedures necessary to build a full-fledged layout-to-circuit extractor, of which the finite element based capacitance extraction is an integrated part.

We conclude the work in Chapter 6, and give some indication of further research which may be carried out on the subject.

2. Interconnect Parasitics and Scaling

2.1 Introduction

Integrated circuits consist of (localized) active devices and a (distributed) interconnection network. The properties of the interconnections are increasingly important factors affecting the performance and operation of the circuit as a whole. In this chapter, we study the behavior of such interconnections, their electrical significance, and what constitutes an effective model for them. This is primarily intended as an overview of the subject and as motivation for the work that follows in subsequent chapters. For additional information, see e.g. [Ling (1987)] and [Bakoglu (1990)].

2.2 IC Interconnection Modeling

The electromagnetic behavior of IC interconnections is governed by Maxwell's equations. Because of the complexity of solving these equations, approximations are needed. A very important approximation is that of a set of lossy coupled transmission lines. We only consider otherwise ideal lines, i.e. those that are linear, frequency-independent and have no skin effect, etc. Except for diffused conductors, this is usually a valid approximation of IC interconnections when the operating frequencies are below 1 Ghz.

Thus, the transmission line model is an accurate model for IC interconnections. However, in many cases (depending on e.g. the source, load, line length and frequency), much simpler approximations are also accurate. In this section, we will discuss the applicability of such approximations, in particular, that of lumped equivalent circuits that model the distributed nature of the system of interconnects as a discrete network consisting of ideal resistances and/or capacitances, without inductances.

For that purpose, let us first introduce the following notation:

l	Length of an interconnection line.
r, c, h	Resistance, capacitance, inductance of a line, per unit length.
R, C, L	Total resistance, capacitance, inductance of a line.

The following equations are useful for calculating the delay caused by a line. The RC delay of a line is given by:

$$t_{RC} = \tfrac{1}{2}rcl^2 = \tfrac{1}{2}RC \tag{2.1}$$

The transmission line (LC) delay of a line is given by:

$$t_{LC} = l\sqrt{hc} = \sqrt{LC} \tag{2.2}$$

The characteristic impedance of a lossless transmission line is given by

$$Z_0 = \sqrt{\frac{L}{C}} \tag{2.3}$$

The value of r is strongly dependent on the type of the material. For example, typical sheet resistances of different wires (of typical thickness) are as follows:

polysilicon: 40 Ω/□ silicided poly: 4 Ω/□ aluminum: 40 mΩ/□

Josephson Junction IC's operate at superconducting temperatures, and their interconnection wires have no resistance.

The value of c exhibits much smaller variations. In Section 2.4, we will see that a value of 150 pF/m is a useful estimate. Of course, this estimate is only valid for standard technologies and not for technologies such as Silicon on Insulator.

In the case of lossless, homogeneous transmission lines, L and C are related as follows:

$$L = \frac{\varepsilon\mu}{C} \tag{2.4}$$

where ε and μ are the permitivity and permeability of the medium, respectively. Although IC interconnections are neither lossless nor homogeneous, an effective ε can often be determined so that Equation (2.4) approximately holds. Consequently, $h = 0.5\mu$H/m is a reasonable value for the analysis in the remainder of this section.

Some typical interconnect parameter values, assuming 2 μm wide lines, are summarized in Table 2.1. This table specifies non-zero values for all interconnect parameters. However, not all three types of parasitics are always equally important. This depends, for example, on the following.

Table 2.1. Typical interconnect parameters

parameter	value
r_{poly}	20 MΩ/m
r_{metal}	20 kΩ/m
c	150 pF/m
h	0.5 μH/m

Length of interconnection. In general, the longer the line, the more important all three types of parasitic elements will be. This does not necessarily mean that at high speeds all effects are always important. Rather, it means that if an effect is marginally important at a certain speed, it is generally more important at a higher speed.

More quantitatively, by comparing Equations (2.1) and (2.2), we find that we can neglect the transmission line behavior of signal lines when

$$t_{RC} \gg t_{LC}$$

or

$$RC \gg \sqrt{LC}$$

or

$$R = rl \gg \sqrt{\frac{L}{C}} = Z_0$$

We can immediately conclude that for polysilicon lines, inductance is never important: Even for short lines, the resistance will already be much larger than Z_0.

For power lines, interconnect inductance can be important because it introduces so-called dI/dT noise [Bakoglu (1990)].

Signal rise time and driver impedance. To determine their significance, t_{RC} and t_{LC} must be compared with the signal rise time t_r. The signal rise time depends on the circuit characteristics and speed of the signal that drives the line driver, but also on the ratio between driver impedance Z_s and line impedance Z_0 [Bakoglu (1990)]. It turns out that if $Z_s \gg Z_0$, the line inductance can be ignored. This condition is usually true for MOS technologies, but not for bipolar technologies.

An attempt to summarize (and generalize) which circuit elements are required to model on-chip interconnections is presented in Table 2.2, which is taken from [Ling (1987)].

Table 2.2. Dominant circuit components for interconnections in typical technologies

Technology	transistor impedance	level of current	wire length	dominant circuit elements
FET	high	low	short	C
			long	C, R
		high	short	C, R, (L)
			long	C, R, L
Bipolar	intermediate	low	short	C
			long	C, R, (L)
		high	short	C, L, (R)
			long	C, R, L
Josephson Junction	low	Determined by Z_0	short	L
			long	L, C

2.3 Interconnection Scaling

The continuous improvement in the characteristics of integrated circuits (their functionality, performance, device dimensions etc.) is the result of *scaling*. With scaling, physical dimensions are reduced. In order to avoid problems caused by high electrical field strengths, voltages are also reduced while doping levels are increased. As a result, the speed of the devices and the circuits improve.

A scaling theory, relating speed improvement and the amount of scaling, was first developed for MOS devices [Dennard (1974)]. Using this approach to scaling, a scaling factor S ($S > 1$) is defined, expressing the proportional reduction of the physical dimensions and voltage levels, and increase in the doping level. The results are summarized in Table 2.3. Uniform scaling as defined above is often called *ideal scaling*, because it ignores many second-order effects, making it easy to determine and analyze the effects of scaling. However, in practice, the multitude of second-order effects make scaling more problematic than predicted by ideal scaling theory. These effects include the non-scalability of material parameters (the silicon bandgap, junction potentials etc.), resulting in so-called short-channel effects.

Another problem of ideal scaling is related to the RC delay of the interconnections. To describe this effect, we must distinguish between local and global interconnections.

Table 2.3. Ideal scaling of devices.

parameter	factor
physical dimensions	$1/S$
substrate doping	S
voltages	$1/S$
intrinsic device delay	$1/S$
device area	$1/S^2$

Local interconnections are between individual devices, forming higher-level building blocks such as gates, flip-flops, counters, and so on. The length of these interconnects is proportional to $1/S$. *Global interconnections* are between modules on a chip. Their length is proportional to the size of the chip, which increases with every new technology generation. When S_c is used to denote this chip scaling factor, the length of the global interconnections is thus proportional to S_c.

With respect to an interconnect as shown in Figure 2.1, the delay caused by such interconnections is proportional to RC, where R is the total resistance and C is the total capacitance of the line. (Source resistance and load capacitance are ignored.)

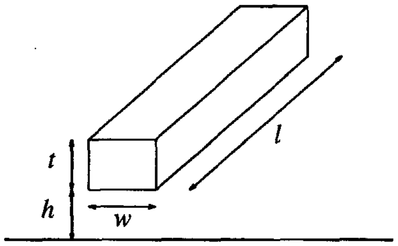


Figure 2.1. Piece of interconnect to illustrate scaling.

The resistance of a piece of interconnect equals

$$R = \frac{\rho\, l}{w\, t} \tag{2.5}$$

where ρ is the specific resistance (material constant), l is the length of the line, w the width and t the thickness of the line. This interconnect resistance can be written as rl, where $r = \rho/wt \propto S^2$ is the resistance per unit length.

The capacitance of such interconnects is given by cl, where c is the capacitance per unit length. This value depends on the cross-sectional dimensions h, w and t, and on the dielectric constant ε of the material. In the case of ideal scaling, c is a constant.

Together with the scaling behavior of l, the RC delay of interconnections is thus described by $RC = rcl^2 \propto l^2S^2$. Since the intrinsic gate delay scales as $1/S$, the ratio between the local interconnect delay and the gate delay increases by a factor of S and the ratio of the global interconnect delay and the gate delay increases by a factor of $S^3S_c^2$. These results are summarized in Table 2.4 in the column labeled "ideal scaling". (The column labeled "lateral scaling" is explained below.)

Table 2.4. Scaling of interconnections.

parameter	ideal scaling	lateral scaling
unit length interconnect resistance (r)	S^2	S
unit length interconnect capacitance (c)	1	1
length of local interconnects	$1/S$	$1/S$
RC delay of local interconnects	1	$1/S$
ratio of local interconnect delay and device delay	S	1
length of global interconnects	S_c	S_c
RC delay of global interconnects	$S^2S_c^2$	SS_c^2
ratio of global interconnect delay and device delay	$S^3S_c^2$	$S^2S_c^2$

So, with ideal scaling, the performance of a chip is determined to an increasing extent by the interconnections. To diminish this effect, circuit techniques such as repeater circuits are being used, and IC technology development is being concentrated on low resistance interconnects. Nevertheless, the problem is sufficiently severe so that ideal scaling is not strictly applied. Instead, the lateral dimensions of the interconnections are scaled more than the vertical dimensions. The thickness of the conductors is kept approximately constant.

For purposes of illustration, we will consider the case in which vertical dimensions are not scaled at all, which we will call *lateral scaling*. The interconnection delay resulting from lateral scaling is also indicated in Table 2.4, assuming that the unit length interconnect capacitance c is still constant. While this is not strictly the case, we will see in Section 2.4 that this assumption is justified up to line widths around $0.5 - 1.0\ \mu$.

Another reason to maintain the thickness of the interconnections is the occurrence of *electromigration*. When the current density in a metal line exceeds a certain material-dependent value, material of the line is displaced and the line can break, resulting in an open circuit. Although some technological measures to decrease electromigration can be taken, they usually have the adverse effect of increasing the resistance. While lines are not readily made wider because it decreases packing density, their current conduction capability must be maintained by (relatively) large vertical dimensions. Large vertical dimensions imply lateral scaling.

2.4 Interconnection Capacitance

With lateral scaling, the cross-sectional dimensions of the interconnects change. As a result, the coupling capacitance between interconnects is becoming far more important. In this section, we investigate these effects and we illustrate their relevance to the electrical behavior of the chip through some examples.

For the sake of concreteness, we assume in this section that we are dealing with a hypothetical but realistic double metal CMOS process with perfect planarization and layer thicknesses as given in Table 2.5. The dielectric (assumed to be of infinite thickness) is SiO_2.

Table 2.5. Layer thicknesses

gate oxide	250Å
inter-wire oxides	0.75μ
poly	0.5μ
metal 1	0.75μ
metal 2	1.0μ

2.4.1 Case of Parallel Lines

Consider the parallel metal 2 interconnects (cf. Table 2.5) as illustrated in Figure 2.2.

Figure 2.3(a) presents several characteristic capacitance values per unit length for the middle conductor, as a function of the lateral dimensions. Figure 2.3(b) presents the same capacitances normalized to the parallel-plate capacitance to show the relative magnitudes at small dimensions more clearly. In Figure 2.3, C_{pp} denotes the ground capacitance computed using the parallel-plate formula, C_{gnd} the total ground capacitance including the effects of the fringe field and C_s the so-called short-circuit capacitance of conductor 2. This is its total capacitance when all other conductors have a low impedance (see Chapter 4).

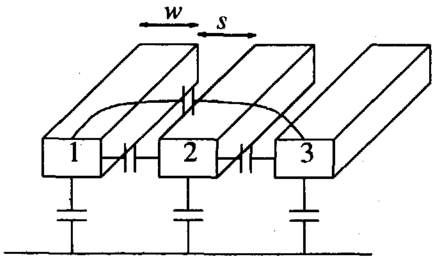


Figure 2.2. Three parallel metal 2 interconnects.

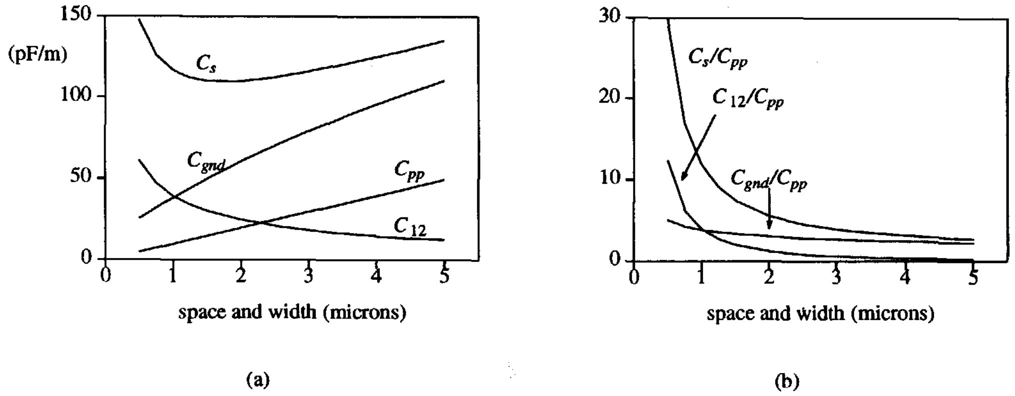


Figure 2.3. Absolute (a) and normalized (b) capacitances for the middle conductor of Figure 2.2 as a function of the lateral dimensions.

It is clear that C_s increases with smaller dimensions, thereby increasing the delay caused by the interconnect. However, as this effect is not so strong until $w = s = 0.5\mu$, we are justified in assuming a constant capacitance with lateral scaling, as we did in Section 2.3.

It is also clear that the total coupling capacitance $2 \times C_{12}$ dominates C_{gnd} for $w,s \leq 2\mu$. In Section 2.4.2, we will see that such strong coupling can easily lead to incorrect circuit behavior.

2.4.2 Electrical Analysis of Capacitive Coupling

The strong capacitive coupling between parallel lines that readily occurs in state-of-the-art VLSI circuits can cause intolerable crosstalk as well as increased delay because of a phenomenon similar to the Miller effect: signal delay can depend on the signal waveforms on neighboring lines. This introduces a certain unpredictability that, in effect, worsens the problem of predicting electrical behavior. If the electrical behavior cannot be predicted accurately enough, large safety margins that reduce performance and/or increase area, must be maintained during the design phase. By way of illustration, consider the case of five parallel lines as shown in Figure 2.4.

In Figure 2.4, each line is driven by an inverter and different input patterns are applied as shown. Each input pattern (or excitation) has a label that indicates what type of electrical behavior will be observed at the end of the middle line, when the input pattern is applied. The result of SPICE simulations of this network for each of these excitations is presented in Figure 2.5. The waveforms show the output of the middle line in Figure 2.4, labeled "response". For each excitation, the correspondence between waveform and excitation is indicated by the labels in front of the waveforms. The timing and shape of the excitations is indicated by the signal "in_1" in Figure 2.5. This analysis clearly demonstrates the significance of capacitive coupling on the electrical behavior. While it can be argued that the present case is somewhat extreme, it can also be argued that many circuits cannot tolerate even a fraction of the coupling capacitance assumed in this analysis. Such circuit examples are easily found in analog circuits, digital circuits employing precharge schemes and/or tri-state buffers, memory arrays employing sense amplifiers and so on. Indeed, a discussion of the capacitive coupling problem for bit lines in DRAM's can be found in [Konishi (1989)] and [Hidaka (1989)].

2.4.3 Case of Crossing Lines

While the previous section considered a 2-dimensional situation, we now consider a 3-dimensional situation. One of the most simple 3-dimensional configurations is that of two crossing metal 1 and metal 2 lines, as shown in Figure 2.6. An analysis of the coupling capacitance between the lines as a function of line width shows the necessity of a 3-dimensional treatment for state-of-the-art technologies. For the purposes of this section, we can assume the length of both lines to be infinite.

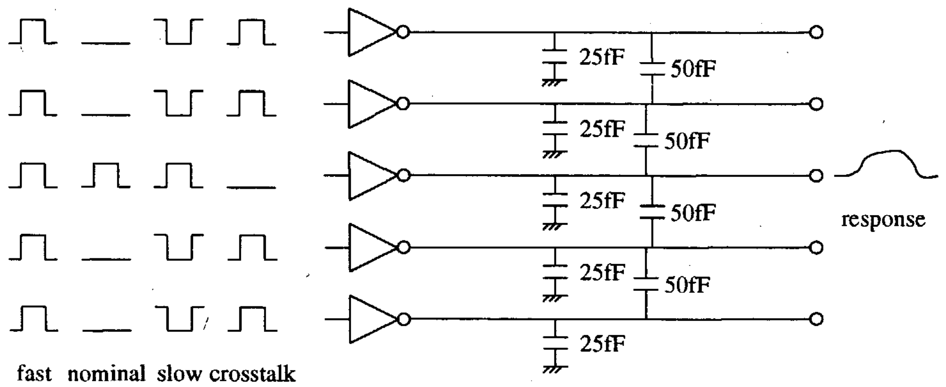


Figure 2.4. Five-conductor bus illustrating the effects of coupling capacitance.

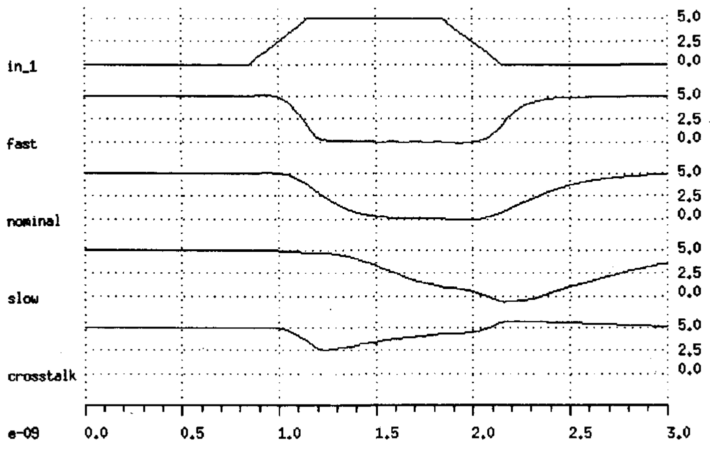


Figure 2.5. SPICE analysis of the network and excitations in Figure 2.4.

Figure 2.7 presents the coupling capacitance computed in three ways:

C_{pp}	Parallel-plate calculation.
C_{2d}	Parallel-plate calculation with first-order correction for fringe fields by summing 2-dimensional components (see e.g. [Meijs (1984)]).
C_{3d}	Fully 3-dimensional computation.

Absolute values of the computed capacitance are shown in Figure 2.7(a). In order to show the relative magnitude of the values computed for small line widths more clearly, Figure 2.7(b) shows the same capacitances normalized to the parallel-plate value.

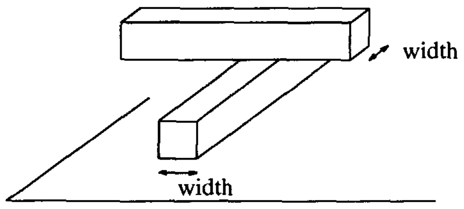


Figure 2.6. Crossing metal 1 and metal 2 conductors.

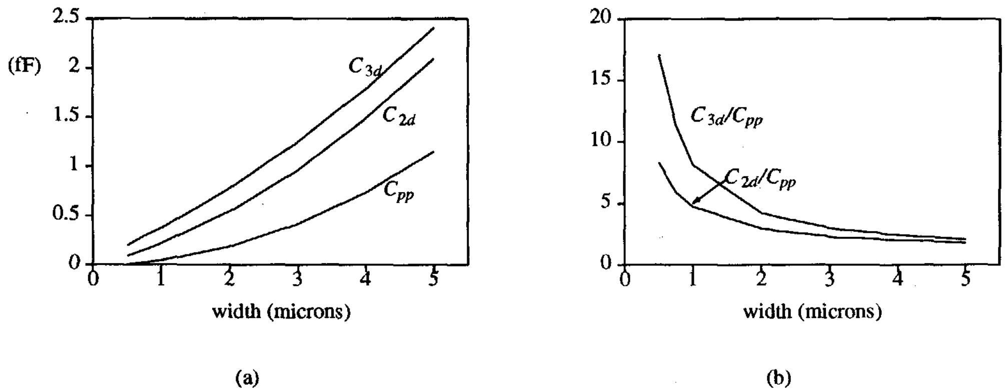


Figure 2.7. Absolute (a) and normalized (b) capacitances for Figure 2.6 as a function of line width.

From this analysis we can conclude that parallel-plate calculations are totally inadequate, even for conservative technologies. For state-of-the-art technologies and critical designs, however, 2-dimensional computations are also shown to be inadequate.

To overcome this problem, we can attempt to devise a heuristic approach that improves the 2-dimensional approximations to better reflect the 3-dimensional reality. For example, the coupling capacitance between two crossing lines as shown in Figure 2.6, can be estimated by adding a constant correction term to the capacitance obtained from the 2-dimensional approximation (or one correction term for each corner of the overlap region). However, there is a very large number of different interconnect configurations all needing their own calibrated correction terms that strongly depend on the distance to and geometry of neighboring wires. This is especially true with an increased number of interconnects (double poly and 3 or even 4 metal interconnect layers), with 45 degree or unconstrained geometry, and with gridless (or with a very fine grid) coordinate systems. Therefore, such heuristics will not work in practice.

We thus conclude that for state-of-the-art technologies and critical designs, rigorous mathematical techniques that model the 3-dimensional nature of the electric field explicitly, are required to obtain sufficiently accurate estimates of interconnect capacitance.

2.5 Conclusion

In this chapter, we have illustrated the significance of interconnection parasitics and shown that they become even more significant with scaling. Consequently, these parasitics must be considered in the *design loop*. A failure to do so may result in lower than expected performance, higher than expected dissipation and/or unreliable or incorrect circuit behavior.

The parasitic behavior exhibited by interconnections depends on many factors. For MOS technologies, however, RC behavior (as opposed to LC or RLC behavior) is most important and great priority must be given to accurate modeling. While they become more important, these parasitic effects also become harder to determine. Verification tools that accurately model 3-dimensional field effects are needed. In Chapters 4 and 5, we will therefore develop a finite element technique for capacitance extraction that is integrated into a layout-to-circuit extractor. This extractor indeed enables the parasitics to be predicted and evaluated during the layout design phase.

References

Bakoglu (1990) H.B. Bakoglu, *Circuits, Interconnections and Packaging for VLSI,* Addison-Wesley, Reading, MA (1990).

Dennard (1974) R.H. Dennard, F.H. Gaensslen, H.N. Yu, V.L. Rideout, E. Bassous, and A.R. LeBlanc, "Design of ion Implanted MOSFET's with very small Physical Dimensions," *IEEE Journal of Solid State Circuits* **SC-9** pp. 256-268 (Oct 1974).

Hidaka (1989) H. Hidaka, K. Fujishima, Y. Matsuda, M. Asakura, and T. Yoshinara, "Twisted Bit-Line Architectures for Multi-Megabit DRAM's," *IEEE Journal of Solid-State Circuits* **SC-24**(1) pp. 21-27 (Feb. 1989).

Konishi (1989) Y. Konishi, M. Kumanoya, H. Yamasaki, K. Dosaka, and T. Yoshihara, "Analysis of Coupling Noise Between Adjacent Bit Lines in Megabit DRAMS," *IEEE Journal of Solid-State Circuits* **SC-24**(1) pp. 35-42 (Feb. 1989).

Ling (1987) D.D. Ling and A.E. Ruehli, "Interconnection Modeling," in *Circuit Analysis, Simulation and Design, 2*, ed. A.E. Ruehli, Elsevier Science Publishers (North Holland), Amsterdam, the Netherlands (1987).

Meijs (1984) N.P. van der Meijs and J.T. Fokkema, "VLSI circuit reconstruction from mask topology," *INTEGRATION, the VLSI Journal* **2**(2) pp. 85-119 (1984).

3. Geometric Algorithms for Extraction

3.1 Introduction

One of the problems in developing an accurate and efficient layout-to-circuit extractor is that of geometric algorithms to handle the layout data. These algorithms must implement geometrical operations that support, for example, transistor recognition, connectivity analysis, and calculation of parasitic (or intended) capacitance values, all being important components in an extraction system. The main requirements and conditions that the geometry module of the extractor must fulfill can be stated as follows:

1. The module, and the resulting extractor, must perform well on conventional workstations and minicomputers, even in the case of very large and flat layouts. This requires a (near) linear time complexity and a sublinear space complexity, since we must assume that the available main memory is not enough to contain the complete layout at one time, and we run the risk of extensive thrashing if we completely rely on virtual memory.

2. The module must provide natural and efficient operations for handling *contextual* or neighborhood information. These operations must support, for example, the calculation of capacitances between neighboring geometries.

3. The module must support arbitrary polygonal geometries. Restricting the module to orthogonal geometries would be inappropriate since the extractor is intended to be used for advanced designs aiming at getting the most from the fabrication technology—especially these designs employ non-orthogonal geometries. A solution in which non-orthogonal geometries are approximated by orthogonal "staircases" is unacceptable since this would impair the accuracy of the resistance and capacitance calculation. It would also impair the efficiency because of the increased amount of data to be processed.

Many geometric data structures and algorithms have been proposed and extractors have been described that use the bitmap approach [Losleben (1979), Willigen (1986)], one of many quad-tree variations, see e.g. [Kedem (1982), Berger (1988)], the kd-tree method [Su (1987)], the bucket method [Nahar (1986)] and the corner stitching method [Scott (1985)]. In fact, most extractors use a scanline algorithm as first presented in [Baird (1977)], see also e.g. [Szymanski (1983)]. However, none of these data structures completely satisfies the criteria stated above.

Scanline algorithms operate by sweeping a vertical line from left to right over the plane containing the layout data to be processed, and all operations take place on the objects intersected by the scanline at each of its successive positions. (Alternatively, a horizontal scanline or a right to left sweep is also possible.) They are ideally suited for solving problems based on the intersections of objects (see e.g. [Bentley (1979)]), since these can be found by analyzing the cross-sections of the plane at each scanline position. Scanline algorithms can be efficient; they often achieve linear or even sublinear space complexities and time complexities of $(ON\log N)$.

However, scanline algorithms are generally weak at manipulating contextual information as delineated in the second requirement above. For *decision problems*—i.e. problems whose answer is chosen from a fixed number of possible answers—a solution can be obtained by *growing* and/or *shrinking* the individual shapes. For example, with design rule checking, the set of answers is violation or no violation. The problem of verifying a minimum distance between geometric features is solved by growing. This operation transforms the original problem into an intersection problem: features that are not separated enough in the original problem intersect in the transformed problem.

For *measuring problems*—i.e. problems requiring numerical calculation, e.g. the calculation of the distance between two features—these solutions are not applicable. An example of this type of problem is the extraction of the capacitance between parallel wires. In response, we have developed a scanline algorithm that enables neighborhood operations by combining it with the corner stitching method. The algorithm represents the geometry in a (narrow) band of adjustable width immediately to the left of the scanline by using a corner stitching data structure. This data structure is created at the front of the (moving) scanline, and is destroyed a fixed distance to the left of the scanline. The algorithm thus combines the advantages of the scanline algorithm (low storage requirement) with those of the corner stitching method (powerful neighborhood operations). To meet the third requirement stated above, the corner stitching method has been modified for non-rectangular geometries. Furthermore, by implementing the scanline data structure as a doubly linked list, we have improved the expected-case time bounds of the scanline algorithm to $O(N)$.

The rest of this chapter is structured as follows. After a review of the corner stitching method (Section 3.2) and the scanline method (Section 3.3), we develop a general scanline algorithm in Section 3.4. Subsequently, we use that algorithm in Section 3.5 to develop an algorithm for the contour of a union of polygons. That algorithm provides the input data for the combined scanline-corner stitching algorithm that we describe in Section 3.6. In Section 3.7, we describe the Space layout-to-circuit extractor that implements the algorithms developed in this chapter, together with measurements confirming good performance. We conclude in Section 3.8.

3.2 Corner Stitching

A data structuring technique that has been shown to be advantageous for representing and analyzing VLSI layout data is *corner stitching* [Ousterhout (1984)]. This method basically provides a form of 2-dimensional sorting of the layout data (in a sense, it may be viewed as a 2-dimensional extension of a linked list data structure) and hence provides a *notion of proximity.* That is, neighboring features are stored logically close together in the data structure. Consequently, it can efficiently support neighborhood search operations. The corner stitching data structure was employed successfully in a VLSI layout design system called *Magic* [Ousterhout (1984a), Ousterhout (1985)].

As originally presented, corner stitching was limited to layout data containing only *isothetic* (or *orthogonal* or *manhattan*) features (i.e. polygonal features with edges parallel to one of the coordinate axes). However, at least two extensions for non-orthogonal polygonal geometries have been presented [Marple (1988), Meijs (1989)]. In this section, we first present the corner stitching method for orthogonal layout and subsequently our non-orthogonal extension.

3.2.1 Data Structure

For purposes of explanation, we first consider a system with only one mask layer. The orthogonal layout polygons need not be simple, but may have holes in them.

The corner stitching data structure represents the layout according to the *paint* paradigm [Ousterhout (1984b)], i.e. for each point in the plane it represents the presence and absence of the mask layer. This has to be contrasted with the *object* paradigm, which explicitly represents individual rectangles or other features that can overlap each other.

The paint paradigm is more natural and efficient for many layout applications, including extraction and editing.

With corner stitching, the 2-dimensional plane containing the layout pattern is partitioned into a set of disjoint (non-overlapping) rectangles, called *tiles*. Tiles are either totally opaque *(solid tiles)* or totally transparent (*space tiles*). Every point in the plane is contained in one and only one tile; tiles contain their lower and left edges and their lower-left corner. Conceptually, the 2-dimensional plane extends to infinity on all sides. (In a practical implementation, the largest and smallest possible values of the data type used to store the coordinates can be used.)

Tiles are made first as tall as possible, then as wide as possible[1]. This particular subdivision provides a unique canonical form, which prevents the generation of many small tiles which would cost storage space and slow down the algorithms working on the data structure. An example of a corner-stitched plane is given in Figure 3.1.

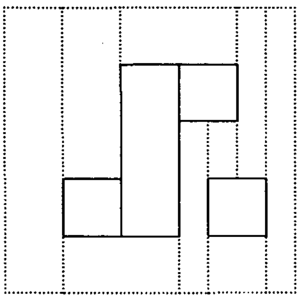


Figure 3.1. Illustration of the canonical tile subdivision of a layout. Solid (space) tiles have a solid (dotted) outline.

The dissection is represented in a data structure in which tiles are linked to their neighbors at the bottom-left and top-right corners. These links are called *corner stitches*. Each tile has four stitches, labeled *bl*, *lb*, *tr* and *rt*. For each tile *t*, they are

1. In Magic, tiles are made first as wide as possible, then as tall as possible. This change does not affect the functionality of the data structure, and only requires some trivial changes to be made to the corner stitching algorithms. The reason for the change here will become clear later on in this chapter.

defined as follows:

bl points to the bottom-most tile sharing a finite segment of the left edge of *t*.
lb points to the left-most tile sharing a finite segment of the bottom edge of *t*.
tr points to the top-most tile sharing a finite segment of the right edge of *t*.
rt points to the right-most tile sharing a finite segment of the top edge of *t*.

This is illustrated in Figure 3.2. The stitches point in each of the four directions, and enable various algorithms to efficiently search for and enumerate neighboring features, as will be discussed in Section 3.2.2.

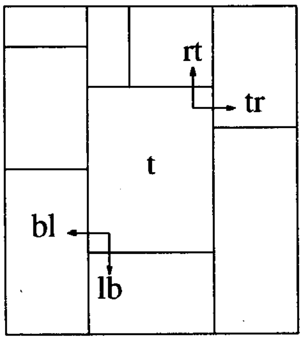


Figure 3.2. Illustration of the stitches of tile *t*.

When more than one mask layer is present, as in practice, the corner stitching method described above must be generalized to provide for multiple mask layers. There are basically two ways to accomplish this:

1. Use multiple tile types, one type for each combination of mask layers (2^n different tile types for n mask layers). This will result in a fragmented corner stitched plane with many small tiles.

2. Use multiple tile planes, one plane for each mask layer. This will require more work for the detection and registration of relevant overlaps.

In Magic, the first solution would take up too much memory, since in that system the complete layout must fit in the computer's memory. It would also take up too much disk space, since the database is also stored in tile format. The second solution is not efficient with respect to CPU time, since the algorithms then involve many complicated shape computations. For example, in MOS circuits, the transistor shapes are defined by the overlap of polysilicon and diffusion masks, but in the case of the multiple tile plane, they are only available implicitly.

As a solution, Magic implements a combination of both approaches. By conveniently combining strongly interacting mask layers into a small number of tile planes, this approach results in efficient algorithms for applications such as design rule checking and circuit extraction. For example, with a MOS process, the polysilicon mask layer and the diffusion layer are stored together in one plane and metal is stored in another plane. Contact holes are duplicated in each of the planes they connect. This combination is convenient in several ways. For example, it facilitates circuit extraction because the corner stitching representation is almost a direct representation of the electrical circuit and it facilitates design rule checking because most planes can be checked almost independently.

3.2.2 Corner Stitching Algorithms

The most attractive features of corner stitching are that it provides fast geometrical searches and permits fast incremental modification. This latter aspect is extremely important in interactive layout synthesis systems such as Magic. Here, the term ''synthesis'' includes activities such as layout editing, compaction and routing. However, this feature is irrelevant in batch verification (analysis) tools, such as used for extraction. As many concrete descriptions of corner stitching algorithms can be found in, for example, [Ousterhout (1984)] and [Marple (1990)], we will only touch on this topic here. A summary of corner-stitching operations and their running times is given in [Ousterhout (1984)].

Fast geometric searches are facilitated by the corner stitches—to reach neighboring tiles only a few well-defined stitches must be traversed. For example, all direct neighbors of a tile are visited by *edge walks* along the tiles' edges. To find all tiles along the right edge of tile *t*, one starts with the *tr* stitch of tile *t*, and then traverses *lb* stitches until the bottom of tile *t* has been reached, as illustrated in Figure 3.3.

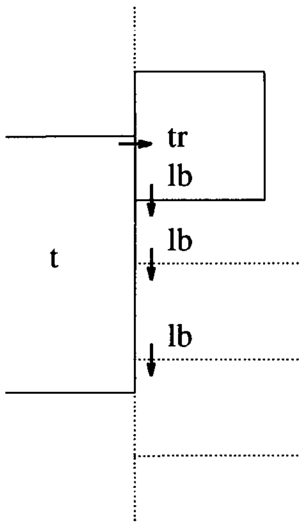


Figure 3.3. An edge walk along the right edge of tile *t*.

3.2.3 Trapezoidal Corner Stitching

Although the restriction to orthogonal layout is often acceptable, it is in conflict with our objectives. Thus, we need a generalization of the corner stitching principles to accommodate non-orthogonal polygonal layout. Such a generalization has been presented in [Marple (1988)] and [Meijs (1989)].

In this type of generalization, tiles are trapezoids instead of rectangles and we refer to *trapezoidal corner stitching* instead of *rectangular corner stitching.* While trapezoidal tiles have vertical left and right boundaries as rectangular tiles do, their upper and lower boundaries can have any slope (except vertical). Trapezoids may degenerate into triangles when the length of either the left or the right boundary becomes zero.

With trapezoidal corner stitching, the upward and downward pointing stitches (*rt* and *lb*) can be defined the same as for rectangular corner stitching. However, the definition of the left and right pointing stitches (*bl* and *tr*) is no longer applicable, because a shared segment need not exist in the case of a triangular tile, see Figure 3.4. Therefore, the definition of these stitches is adapted as follows. When "contains" is defined in such a way that the end points of the edges are considered part of the edges, then for each tile *t*:

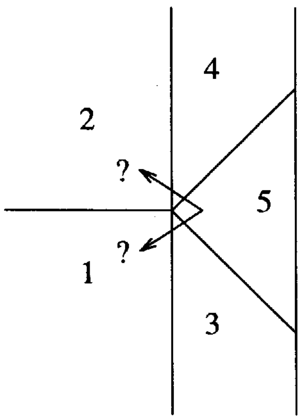


Figure 3.4. Illustration of the ambiguity of the left (and right) pointing stitch definition.

bl points to the bottom-most tile whose right edge contains the lower-left corner of *t*.
tr points to the top-most tile whose left edge contains the upper-right corner of *t*.

This definition means that in Figure 3.4 the left pointing stitch of tile 5 would point to tile 1. This stitch could also be defined as pointing to tile 2. This definition, which would be consistent with the definition for rectangular corner stitching, was originally considered. However, it would result in more complicated algorithms, for example, for *shadow searching*. Shadow searching [Ousterhout (1984)] involves the detection of parallel edges in, for example, design rule checking and capacitance extraction.

3.3 Introduction to Scanline Algorithms

Scanline algorithms form an important class of algorithms for geometric problems in CAD. These algorithms offer efficient solutions to many geometrical and topological problems, such as the detection, reporting and processing of intersections.

The method can best be introduced using an example, which is taken from [Bentley (1979)]. The problem is to report all intersecting pairs among a set of horizontal and vertical line segments. For purposes of explanation, degenerate cases are neglected. For example, it is assumed that no two line segments overlap.

Imagine a straight line (the *scanline*) that, by convention, is vertical and sweeps the layout from left to right. During the sweep operation, the scanline encounters the input line segments. Projected onto the scanline, the horizontal segments are points and the vertical segments are intervals. The algorithm maintains a data structure S that captures the projections of the horizontal edges, and is queried using projections of the vertical edges. When the scanline reaches the left end point of a horizontal edge h, its projection is inserted in S and when the scanline reaches the right end point of h, its projection is deleted from S. When the scanline encounters a vertical segment v, S is searched for all points in the interval defined by v. For each point p found in this way, an intersection between v and the horizontal segment represented by p is reported.

It is of course only necessary to update or query S (to "stop the scanline") at the abscissas defined by the projections of the end points of all segments on the x-axis. The sweeping of the scanline is then implemented by first sorting these abscissas in increasing x-order and then processing them in that order. The abscissas are called *event points*.

The data structure S must support the insertion and deletion operations, as well as interval queries that ask for all points in the interval. A suitable and efficient implementation is by using a height balanced binary tree sorting the points with their ordinate as the key, in which the leaves are linked together in a sorted doubly linked list. The interval queries then involve locating the lowest point in the tree covered by the interval, and from there traversing the linked list.

The performance of the algorithm can be established as follows. Let there be N line segments and thus at most $2N$ event points. The time needed to sort these points is bounded by $O(N\log N)$. For each event point, we need to perform an insertion in the tree, a deletion from the tree or a range query. The first two can be completed in $O(\log N)$ time per operation if height balanced binary trees are used. The range query can be performed in time $O(\log N+k_q)$, where k_q is the number of intersections found by the query q. Thus, the total time needed is bound by $O(N\log N+k)$, where k is the total number of all intersections. The worst-case space complexity is trivially $O(N)$.

Generalizing from this example, we can make the following remarks:

1. There are two data structures common to all scanline algorithms. The first records the relevant information at each cross-section of the plane defined by a particular position of the scanline. This data structure, in essence, maintains the *state* of the algorithm, and is called the *state ruler* in [Fokkema (1983)] or the

sweep-line status in [Preparata (1985)]. In the above algorithm, the state ruler is implemented by the data structure *S*.

For specific applications, specific state ruler data structures are needed to achieve optimal space and time complexities. The state ruler is often implemented as some kind of balanced binary tree, as in the example above. Other typical state ruler implementations are the segment tree [Bentley (1980)] and the interval tree [McCreight (1980), Edelsbrunner (1981)]. See [Preparata (1985)] for a discussion of these data structures. Combinations of these data structures are also useful [Edelsbrunner (1981)].

The second data structure is the *event schedule*. This schedule controls the operation of the algorithm. It is often the input data itself, suitably sorted, but it can also be derived from the input data as in the above example. Sometimes, the scanning operation involves inserting new events in this schedule, see e.g. [Nievergelt (1982)]. In that case, a priority queue is usually used. If new events do not need to be inserted, the events can be presorted and stored in a disk file.

2. The suitability of scanline algorithms for many applications stems from its ability to transform a 2-dimensional, static problem into a 1-dimensional, dynamic problem. A *static problem* is a problem in which all input data are fixed before the algorithm is executed. A *dynamic problem* is a problem in which the relevant data are continuously updated. In the above example, the 2-dimensional static problem is defined by the set of horizontal and vertical line segments, and the 1-dimensional dynamic problem by the continuously updated state ruler. Although the scanline technique generalizes to higher dimensions and then transforms static N-dimensional problems into dynamic $N-1$-dimensional problems, it is particularly efficient in the 2-dimensional case because the resulting 1-dimensional problems can often be solved efficiently.

3. With scanline algorithms, we can usually distinguish between scanline maintenance and scanline processing. *Scanline maintenance* involves all operations for updating the state ruler when the scanline is moved from scanline stop to scanline stop. *Scanline processing* involves the analysis of the state ruler in order to realize specific applications. In the above example, scanline maintenance is the insertion and deletion of points, and scanline processing is formed by the interval queries.

4. At each scanline position, the state ruler usually contains only the objects intersected by the scanline. This is also the case in the above example. The worst-case space complexity is $O(N)$, since pathological cases in which it is possible to draw a vertical line intersecting all objects do trivially exist. However, the expected-case space complexity is much better because most layouts show a fairly regular distribution of features. For practical applications, this low expected-case space complexity is one of the most attractive features of the scanline method.

3.4 The Scanline Maintenance Algorithm

We have distinguished between scanline maintenance and scanline processing. As we will see, this distinction allows the scanline maintenance to be a generic step for a common set of applications that mainly differ in their scanline processing algorithm. In this section we will develop a scanline maintenance algorithm that we will indeed use (as described in subsequent sections) for different applications, namely, contour edge generation and region enumeration.

3.4.1 Algorithm

The input of the algorithm is formed by the edges of the polygons making up the mask geometry. Only the non-vertical edges are explicitly present; the vertical edges are implicit in this description [Lauther (1981)]. The edges are described by their end points (x_l,y_l) and (x_r,y_r) with $x_l < x_r$, and data that identify the side of the opaque region. The edges are sorted lexicographically, first on increasing x_l, then on increasing y_l and finally on their slope.

The state ruler contains the edges that intersect the scanline at its current x-position, in the order of the increasing y-coordinate of their intersection with the scanline. Edges with the same scanline intercept are ordered according to the slope of the edge. Such a lexicographic ordering on $<y_{int}, slope>$ is called the $<_x$ (*smaller at* x) ordering (see also [Preparata (1985)]) and edges intersecting a vertical line at x are said to be *comparable at* x. Other edge relations can be defined analogously.

The state ruler is implemented as a doubly-linked list of edges. At every scanline stop, this list is traversed from head to tail. Because of the $<_x$ ordering, this corresponds to an upward motion in the slice of the plane represented by the state ruler: the y-coordinate never decreases.

The $<_x$ ordering is the scanline invariant; scanline maintenance corresponds to maintaining its truth. This involves the following actions: (1) inserting edges starting at the current scanline position, (2) deleting edges ending at the current scanline position and (3) updating the order of intersecting edges in the state ruler when the scanline reaches the x-position of their intersection point.

Edges in the state ruler may have an application-dependent type or, in general, may have application dependent attributes associated with it. Edge attributes are data, such as topological data or data indicating the origin of an edge, facilitating the specific application. An example of a topological attribute is a bit that identifies the side of the opaque region, i.e. the region above or below the edge. An example of data indicating the origin of an edge is, in layout verification, a proper identification of the mask layer of which the edge is part.

When edges can overlap each other, there must be a rule or a set of rules that defines how attributes combine when two (or possibly more) edges overlap. For example, in the case of a bit denoting opacity, there can be a rule stating that two overlapping edges, one with and one without the bit set, cancel each other's effect. In what follows, we will distinguish between *simple edges* and *manifold edges*.

Furthermore, there must usually be a rule or a set of rules to compute certain properties of the regions in the plane. We will refer to these properties as the *plane state* and we will consider the plane state to be a function of the position in the plane defined incrementally as follows: The plane state above an edge is a combinatorial function of the plane state below that edge and the attributes of the edge. The plane state at $y = -\infty$ has, by definition, a value that we shall denote as the **null** value. An example of a plane state is opacity. Opacity above an edge follows from opacity below the edge and the opacity bit of the edge. More complex plane states for example allow Boolean mask operations to be formulated elegantly.

Clearly, the encoding of the attributes can have a pronounced effect on the implementation of the rules maintaining the edge attributes and, consequently, on the efficiency of the scanline algorithm. For example, consider using an integer polarity instead of a bit to indicate opacity. The polarity is $+1$ for an edge with the opaque side above it and -1 for an edge with the opaque side below it. The combined polarity of manifold edges is then simply given by the arithmetic sum of the polarities of each component edge.

Thus, developing an application that uses the scanline maintenance algorithm described in this section generally also means defining the attributes to be associated with an edge, including the encoding of the attributes and the rules for combining them, as well as defining the plane state function. We therefore describe these aspects along with the specific applications in subsequent sections.

The scanline algorithm that we have developed is based on the principles described above and is shown in Algorithm 3.1. It is structured such that all steps of the algorithm, including the determination of the next scanline stop and the scanline processing operations, are executed during a single traversal of the state ruler at each scanline stop. Some of the features of the algorithm are elucidated below.

Head and tail. The state ruler is implemented as a doubly-linked list of edges, with a head and a tail sentinel edge of infinite length respectively below and above every input edge. More formally, $head <_x e <_x tail$ for every input edge e. The head and tail sentinels are, by definition, the initial and final contents of the state ruler. These initial contents satisfy the scanline invariant and therefore form a valid starting configuration for the scanline maintenance algorithm. The final contents also satisfy the scanline invariant.

Input. Given a certain position and state of the scanline, the next input edge is returned by the *fetch* operation. The head of the input queue, i.e. the edge returned by the next execution of *fetch*, is available in the global variable *nextEdge*. If the input has been exhausted, *nextEdge* contains a sentinel edge with xl = ∞ so that the algorithm terminates.

Manifold edges. We can describe the proper scanline maintenance by using abstract attributes as follows: Edges from different polygons can overlap each other. In the state ruler, such edges are represented as a single edge with composite attributes, where attributes are composed according to an application-specific rule. When edges only partially overlap, the attributes of an edge change with the scanline's x-position. Therefore, a list is maintained, containing ''partial edges'' according to the attributes of the parts. The *bundle* operation adds edges to the list and *unbundle* deletes the first element when the scanline has reached a position where the attributes of the edge changes. The next position where an edge changes attributes is always in the *xc* field of the edge. If an edge does not change attributes, this field is greater than the *xr* field of the edge. Insofar as they implement the rules for combining edge attributes, the *bundle*

```
x := nextEdge.xl                                    # current scanline abscissa
next_x := ∞                                         # next scanline abscissa

while x < ∞                                         # state ruler advancing loop
    edge := head.fwd                                # state ruler traversal pointer
    while edge $<_x$ tail or nextEdge.xl = x        # state ruler traversal loop
        if edge.xi = x                              # edge intersection
            edge := split (edge)

        if edge.xc = x                              # edge changes attributes
            unbundle (edge)                         # partly application-specific

        if nextEdge $<_x$ edge
            insert (fetch (), edge)                 # insert new edge below edge
            edge := edge.bwd                        # new edge becomes current

        if nextEdge $=_x$ edge                      # manifold edges
            do
                bundle (fetch (), edge)             # partly application-specific
            while nextEdge $=_x$ edge
            intersect (edge, edge.fwd)
            intersect (edge, edge.bwd)

        if edge.xr = x                              # edge should be deleted
            handle (x, edge)                        # application-specific
            edge := edge.fwd
            delete (edge.bwd)
            intersect (edge, edge.bwd)

        else
            if edge.xl = x or edge.bwd.xl = x
                intersect (edge, edge.bwd)          # edge has new neighbors
            handle (x, edge)                        # application-specific
            next_x := Min (next_x, edge.xi, edge.xc, edge.xr)
            edge := edge.fwd

    x := Min (next_x, nextEdge.xl)                  # set next scanline abscissa
    next_x := ∞
```

Algorithm 3.1. The Scanline Algorithm

and *unbundle* operations are application-specific.

Intersection handling. All edge intersections are correctly detected by checking all pairs of edges that are new neighbors of each other [Bentley (1979), Lauther (1981)]. This is done by the *intersect* operation, when edges are inserted or deleted. In our algorithm, the *bundle* operation may make the edges longer and these edges are also checked to see if they intersect with their upper and lower neighbors. If two edges intersect, and if neither of them has another intersection to the right of the scanline but to the left of their mutual intersection, the intersection abscissa is recorded with the edges in their *xi* field.

Once the scanline reaches the intersection point of an edge e, i.e. when $x = e.xi$, the parts of the edges to the right of the scanline are split off and inserted in the state ruler such that the $<_x$ ordering of the edges is satisfied again. Thus, if there are K intersecting edges, they are replaced by $2K$ edges: K ending and K starting at the current x-position. The lowest of these $2K$ edges is returned by the *split* operation and becomes the ''current'' edge in the scanline algorithm.

As far as scanline maintenance is concerned, there is a certain freedom allowed in defining the $<_x$ ordering. The two intersecting edges shown in Figure 3.5(a) can be replaced by the four edges shown in Figure 3.5(b) or in Figure 3.5(c). In fact, any order in which the ending edges are ordered according to decreasing slope and the starting edges are ordered according to increasing slope, is valid for scanline maintenance. Ending and starting edges can otherwise be intermixed. On the other hand, there is often a strong preference for a particular ordering rule from the point of view of scanline processing. Thus, the $<_x$ order can be refined to optimize or simplify the scanline processing.

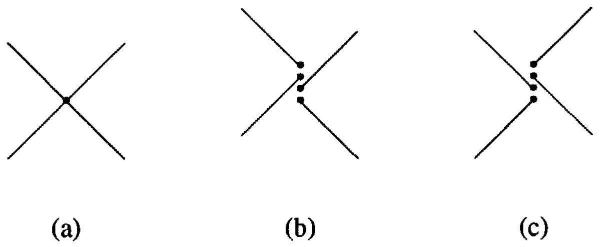


Figure 3.5. Splitting edges and inserting the new pieces in the state ruler.

Only the left-most intersection to the right of the scanline of an edge is remembered and stored in the *xi* field of an edge. This means that intersections may be discovered more than once. For example, in Figure 3.6, intersection point I_2 is discovered when the scanline is at x_1 and e_2 becomes a new neighbor of e_1, and recorded with e_1 and e_2. However, when the scanline reaches x_2, the intersection between e_2 and e_3 is discovered and recorded with e_2 and e_3. The intersection of e_2 at I_1 obscures the intersection of e_2 at I_2. Only when the scanline advances to x_3 is I_2 rediscovered as an intersection between e_1 and e'_2, where e'_2 is the part of e_2 to the right of I_1.

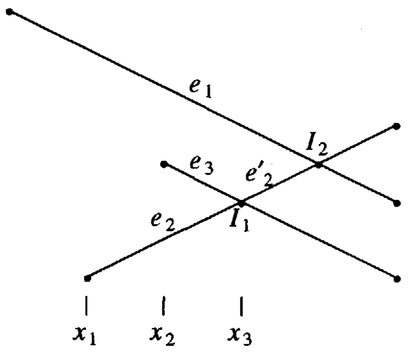


Figure 3.6. Some edge intersections are discovered more than once.

We note that while remembering only the left-most intersection to the right of the scanline and storing the intersection abscissa with the edge differs from the usual practice, it is vital to the performance of the algorithm. For example, in [Lauther (1981)], edges are split into parts to the left and to the right of an intersection as soon as the intersection has been discovered. The right-hand parts are then merged with the input edges, to automatically show up again when the scanline has reached the intersection abscissa. This involves maintaining a priority queue for the input edges. With N being the number of edges in the queue, which are initially all input edges, this takes $O(\log N)$ time per intersection operation [Aho (1974)]. The method presented here only requires a constant time per intersection operation. In the section on complexity, we will use this fact to show that the algorithm runs in linear expected time.

Historically, this technique was proposed in [Brown (1981)] as an improvement to the algorithm of [Bentley (1979)]. There, it resulted in lower memory requirements without altering the time complexity.

Scanline processing. Scanline processing is accomplished via the *handle* operation. It is executed for every edge present in the state ruler, at every scanline position. The semantics of this operation are application-specific—the operation must be implemented separately for each application. It can maintain state information internally, or state information can be attached to the edges in the state ruler. In other words, the edge data structure can contain data that are manipulated by the scanline processing only, and not by the scanline maintenance. In the sections on scanline processing, we will return to this subject.

Next scanline position. During the traversal of the state ruler, a variable *next_x* is maintained. This variable contains the minimum of the intersection abscissa of *e* (*e.xi*), the change attribute abscissa (*e.xc*), and the right abscissa (*e.xr*), of all edges *e* in the state ruler. The next scanline position is then the minimum of this value and the left abscissa of the next input edge (*nextEdge.xl*).

3.4.2 Complexity

In this section, we discuss the time and space complexity of the scanline algorithm with an empty implementation of the *handle* operation used in Algorithm 3.1. This means that we discuss scanline maintenance only, without considering scanline processing. This is useful since the same scanline algorithm has several applications. For the same reason, we ignore the cost of an initial sorting step to prepare the input edges, since this has to be done only once and since this cost can (partly) be credited to other applications of the scanline algorithm. Throughout this section, we let N denote the number of input edges and K the number of all intersections among these edges. In the worst case, $K = O(N^2)$ since N edges can generate $\binom{N}{2} = O(N^2)$ intersections.

Scanline maintenance requires a complete traversal of the state ruler at each scanline abscissa. During this traversal, fetching, insertion, deletion, splitting, intersection and overlap operations are performed. The *fetch*, *insert* and *delete* operations take a constant time per edge. However, the time for *split*, *intersect* and *overlap* operations depends on the number of edges involved. In practice, if we exclude pathological cases, this number is bounded by a small constant. Hence, we assume that all operations to be performed during state ruler traversal, including the latter three, take a constant time per operation and thus that the time for one state ruler traversal is proportional to its length.

In the worst case, all edge end points and intersections have different x-abscissas that require the $<_x$ order to be updated. Thus, the number of scanline positions is $O(N + K)$ in the worst case. At the same time, the length of the state ruler may be as high as $O(N)$. Hence, the worst-case time complexity of the algorithm is $O(N^2 + KN)$ or $O(N^3)$. The worst-case space complexity of the algorithm is $O(N)$.

The worst-case time complexity is clearly suboptimal. For example, the algorithm in [Bentley (1979)] achieves a $((N+K)\log N)$ worst-case time complexity. Because that algorithm involves a balanced tree, this is also the expected-case time complexity. Our algorithm, however, achieves a better expected-case time complexity. This is due to the linked-list implementation of the state ruler, which in fact results in a trade-off between a good worst-case and a good expected-case time complexity.

An expected-case analysis requires the characteristics of the expected-case, or average, input to be analyzed. For that purpose, we assume that we have a discrete coordinate system, i.e., all edge end points as well as intersections are assumed to lie on an integer grid. This is usually the case in design systems in which only orthogonal and 45-degree angles are allowed. Under this assumption, the expected-case number of scanline stops as well as the scanline length are assumed to be $O(\sqrt{N})$ [Bentley (1980a), Lauther (1981)]. Consequently, the expected-case time and space complexities of the algorithm are $O(N)$ and $O(\sqrt{N})$, respectively.

This assumption can be made plausible by considering the case of array-structured repetitions of a basic cell [Lauther (1981)]: Starting with a single cell containing N edges and placing them in a 2×2 array quadruples the number of edges, but only doubles the length of the scanlines and the number of scanline stops.

3.4.3 Measurements

Some experimental support for the preceding analysis was obtained by analyzing a large number of layouts designed in our environment. Figure 3.7 plots the number of input edges, which is taken as a measure of the complexity of the layout, versus the accumulated state ruler length. This accumulated state ruler length, denoted by A, is the sum of all state ruler lengths over all scanline stops.

We investigate the accumulated state ruler length instead of the length of the scanlines and the number of scanline stops directly, in order to eliminate the influence of the aspect ratios of the layouts. Let the theoretically expected length of the scanline and the

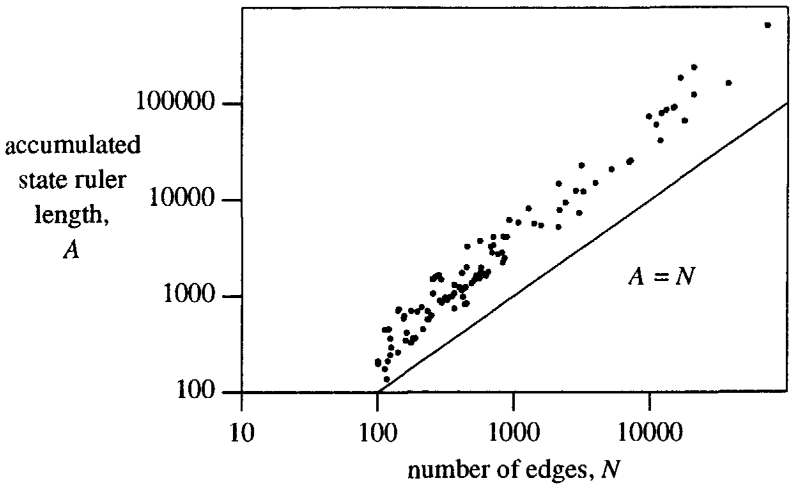


Figure 3.7. Scanline algorithm performance data.

number of scanline stops be denoted by $E(L)$ and $E(S)$ respectively. With $E(L) = E(S) = O(\sqrt{N})$, the theoretically expected value of A, $E(A)$, is given by

$$E(A) = E(L) \times E(S) = O(N) \tag{3.1}$$

In Figure 3.7, this means that the trend indicated by the data points must be a straight line parallel to the line $A = N$. Clearly, this is not exactly the case. When a straight line is fitted through all data points (a least-squares fit on the log-log data), the accumulated state ruler length is given by

$$A = 1.53N^{1.11}$$

However, when the same fitting is performed for all data points with $N \geq 1000$, we obtain

$$A = 1.53N^{1.05}$$

which is much closer to the theoretical analysis. This result can (partly) be explained by the fact that with larger input sizes, the number of scanline stops approaches an upper limit imposed by the discrete coordinate system.

3.4.4 Sorting Input Edges

Before they can serve as input for the scanline algorithm, the edges must be sorted in lexicographical order. Several issues are involved in this sorting step:

1. If we want to be able to deal with very large, flat layouts, we cannot use a standard in-core (or "internal") sorting algorithm [Knuth (1973)]. This would render the $(O\sqrt{N})$ space complexity of the scanline algorithm useless. Instead, we should revert to external sorting methods.

2. Although straightforward, comparison-based sorting techniques result in a time complexity of $O(N\log N)$, we can exploit the properties of the layout data to improve on this. For example, in an environment supporting hierarchical design, the edges from different instances can be sorted separately and efficiently merged together.

3. In addition, we can exploit radix sorting techniques [Knuth (1973)] to exploit the fact that the coordinates are discrete values, thereby reducing the time complexity even more.

3.5 Contour Edge Generation

3.5.1 Algorithm

In this section, we investigate an algorithm that determines the contour of the union of a set of polygonal regions. This algorithm operates on each mask of the layout individually, and each mask is transformed into a mask without overlapping features. The algorithm is sometimes also called an *overlap removal* algorithm. It forms the first step in our extraction program; the actual extraction algorithms work on a layout description of which the individual masks are overlap-free. The algorithm is a variation of the algorithm proposed in [Lauther (1981)].

The input of the algorithm is formed by the non-vertical edges of the input polygons, as discussed in Section 3.4, sorted in lexicographic order. The output is in the same format and order, since it is the input of subsequent scanline algorithms. However, the output edges describe the boundary of the polygonal regions formed by the union of all the input polygonal regions.

We define a polarity as the attribute of an edge. The polarity is an integer number, the polarity of a simple edge is +1 if the opaque side is above the edge and −1 if it is below the edge. The polarity of a manifold edge is defined as the arithmetic sum of the polarities of the component edges.

A plane state above an edge is defined by an integer number that is the arithmetic sum of the plane state below that edge and the polarity of the edge, whereby the plane state at $y = -\infty$ is 0. We refer to the plane state of a region (or a point) as the polarity of the region (or the point). A polarity > 0 indicates an opaque region and a polarity $= 0$ indicates a transparent region. When the polarity of a region changes from 0 to > 0 or vice versa, we have crossed a contour edge. Begin and end points of such contour edges follow from the polarity in each of the four quadrants defined by all the intersections of an edge with the scanline, as detailed below.

The algorithm can be connected to the scanline algorithm in Algorithm 3.1 by replacing the *handle* operation in that algorithm by the *contour* operation of Algorithm 3.2. In this algorithm, the *sign* operation is defined as returning 1 for positive numbers, -1 for negative numbers and 0 otherwise. Also, *code* is an encoding of the 16 different possible ways to intersect an edge in the state ruler with the scanline. Such an intersection naturally defines four quadrants, and *code* is a binary four-tuple with each bit representing the transparency/opacity of a particular quadrant. This is depicted in Figure 3.8(a).

For example, an intersection encoded as "2" is an intersection between an edge and the scanline, whereby only the north-east quadrant is opaque. In most cases, this is not a true intersection, but merely a touching of the left end point of an edge with the scanline. An intersection encoded as "5" is an intersection with all four quadrants being opaque. An example of an input geometry that shows all 16 encodings is given in Figure 3.8(b). Using this figure, we can easily verify the correctness of the actions (the execution of the *beginContourEdge* and *endContourEdge* operations) that are taken for a particular type of intersection.

The state of the *contour* operation is partially maintained in persistent variables (*sw, nw, se* and *ne*) that store the polarity (the plane state) in each quadrant, and partially attached to the edges in the state ruler. Each edge has a previous polarity (*edge.polarityLeft*) and a pointer to an output edge (*edge.output*).

```
procedure contour (x, edge)
begin
    # sw, nw, se and ne are persistent variables to maintain the plane state
    if edge.xl < x                          # edge extends to left of scanline
        nw := nw + edge.polarityLeft
    if edge.xr > x                          # edge extends to right of scanline
        ne := ne + edge.polarity

    y := Y (edge, x)                        # ordinate of intersection with scanline

    code := sign (se) + 2 × sign (ne) + 4 × sign (sw) + 8 × sign (nw)

    if code ∈ {1, 2, 13, 14}
        edge.output = beginContourEdge (x, y)
    else if code ∈ {4, 7, 8, 11}
        endContourEdge (edge.output, x, y)
    else if code ∈ {6, 9}
        endContourEdge (edge.output, x, y)
        edge.output = beginContourEdge (x, y)

    sw := nw, se := ne
    edge.polarityLeft := edge.polarity
end
```

Algorithm 3.2. The contour algorithm

3.5.2 Sorting Output Edges

Since the contour edges being produced serve as the input for other scanline programs, they must be in lexicographic order. However, this is not the natural order in which they are produced by the scanline algorithm. In fact, they are produced in <*xr*, *yr*, *slope*> lexicographic order. One obvious way to sort the edges in the desired order is to use an external sort algorithm like the one used to sort the original input edges in the first place. However, a more efficient and elegant method was proposed in [Szymanski (1983)][2]. There, a scanline algorithm was presented that exploits the order in which the edges are actually produced by the scanline. The algorithm does a reverse

2. However, in a later version of their layout verification program [Chiang (1988)], this algorithm was replaced by a bucket sort technique, which reportedly improved performance.

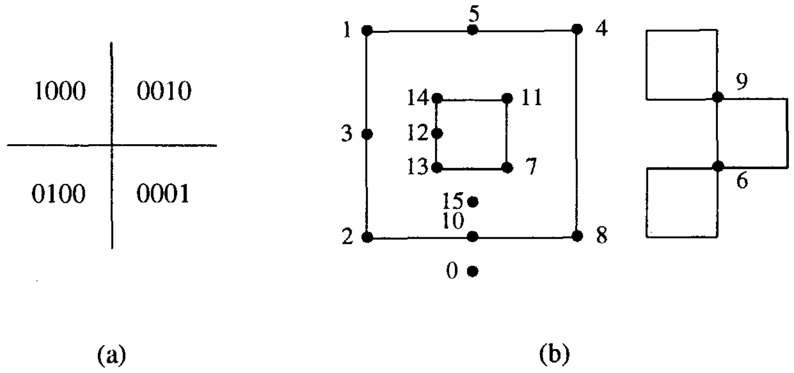


Figure 3.8. Illustration of the encoding of the intersections between an edge in the state ruler and the scanline.

(right to left) scan of the edges to produce the contour edges in reverse-sorted order. The expected-case time and space complexities are $O(N\log N)$ and $O(\sqrt{N})$, respectively, although for fairly large designs the running time appears to be I/O bounded.

As with the scanline algorithm in Section 3.4, we can improve the $O(N\log N)$ time complexity to $O(N)$ by implementing the scanline as a linked list. However, this would still produce the contour edges in reverse-sorted order. Therefore, we propose a different solution.

Our solution is based on the observation that although the scanline produces the edges in $<x_r, y_r, slope>$ order, we can tell whether an edge will become an output edge at its left end point. So, as soon as we discover the left end point of a contour edge, we place the edge into a *queue* of unfinished edges and mark it as not ready. Then, for every right end point of a contour edge, we mark the edge as being ready, and we inspect the front of the queue to see which edges are ready and can be written out.

This method is implemented by the *beginContourEdge* and *endContourEdge* operations executed by the *contour* operation of Algorithm 3.2. These operations are presented in Algorithm 3.3. Here, Q implements the queue of unfinished edges, the *head* operation returns the head of the queue and the operations *inject* and *eject* insert and delete edges into (and from) this queue, respectively. The *eject* operation returns the element that it deletes.

```
function beginContourEdge (x, y)
begin
    edge := create contour_edge
    edge.xl := x
    edge.yl := y
    edge.ready := false

    inject (Q, edge)

    return edge
end

procedure endContourEdge (edge, x, y)
begin
    edge.xr := x
    edge.yr := y
    edge.ready := true

    while head(Q).ready = true
        output (eject (Q))
end
```

Algorithm 3.3. The beginContourEdge and endContourEdge operations

This method, however, increases the space complexity of the algorithm to $O(N)$. This is not just a theoretical inconvenience, but of practical importance too. For example, it is very possible that the layout contains very long horizontal edges from, say, supply or clock lines. As long as these edges remain in the queue, the other edges will remain there too.

As a solution, when the number of finished edges in the queue exceeds a certain threshold, we write these edges to a temporary file and reclaim their core memory. Afterwards, these individually sorted blocks of contour edges are merged using a k-way merge algorithm, where k is the number of partitions created. This can be done in $O(N \log k)$ time, which is almost linear in practice since k will be small. At the cost of one extra I/O step, we have re-established an $O(\sqrt{N})$ space complexity.

3.6 Region Enumeration.

In the introduction to this chapter, we announced a combined scanline and corner stitching technique as a powerful tool to handle the geometry in layout verification programs. In this section, we describe how this can be accomplished given the scanline maintenance algorithm of Section 3.4 and the trapezoidal corner stitching algorithm of Section 3.2.3. While building the corner stitching data structure, the algorithm enumerates all tiles and pairs of abutting tiles in a canonical order. Since the tiles form (trapezoidal) regions, we call this algorithm a *region enumeration algorithm.* The order of enumeration appears to be eminently suited to extraction purposes. The input of the algorithm is in the form of non-vertical edges of the contour of each mask. In other words, the output of the contour edge algorithm described in the previous section serves as this algorithm's input.

In this section, we refer to the attributes of an edge as its color. For a simple edge, the color defines its mask layer. For a manifold edge, the color likewise defines its component mask layers. The color of an edge is defined as a binary n-tuple, or bit mask, with one bit for each mask layer. The combination rule for manifold edges is a bitwise Boolean OR operation. Since the input edges form the contour of each layer (i.e. they are overlap-free) we do not need a polarity of an edge as defined in the previous section. The polarity is implicit in the $<_x$ order of edges when they intersect the scanline.

Furthermore, in this section we refer to the plane-state of a region (or a point) as the color of the region (or the point), which is defined by using the same bit masks as for the color of an edge. Since the edges are contour edges, the color in a region above an edge can be found (the definition of the plane-state function) by a bitwise exclusive-OR operation on the color below the edge and the color of the edge. The color at $y = -\infty$ is 0, a bit mask with all bits cleared.

While the original corner stitching method [Ousterhout (1984)] uses multiple linked tile planes to achieve a higher storage efficiency, we can use one single tile plane because the corner stitching data structure will never exist for the entire layout at the same time. We naturally let the type of a tile (defining the set of masks present at its position) coincide with the plane-state or color of a region. In what follows, we refer to the type of a tile as its color. Space tiles have a color bit mask with all bits cleared.

Since the edges are contour edges, the color above an edge must be different from the color below an edge, and since tiles are regions of one color, all edges must coincide with upper and lower tile boundaries. It is then natural to avoid further upper and lower tile boundaries and to make the tiles as tall as possible, so that if two tiles abut vertically, they are of different color. Vertical tile boundaries are then created so as to make the tiles as wide as possible. Unlike vertically abutting tiles, horizontally abutting tiles can be of the same color. The result will be a trapezoidal tile dissection as defined in Section 3.2.

We first describe the principle of the region enumeration algorithm, momentarily disregarding how it is implemented with the scanline algorithm. Afterwards, we describe this implementation.

At the start of the algorithm, there is one space tile conceptually covering an infinite plane. In the state ruler, tiles are bordered by the edges present in the state ruler. The initial infinite tile is bordered by the head and tail sentinel edges of the state ruler, which is schematically illustrated in Figure 3.9(a). In Figure 3.9, solid lines denote the non-vertical tile boundaries, which coincide with the edges in the state ruler, while the dotted vertical lines denote the vertical tile boundaries.

This infinite plane is then traversed in scanline order, from left to right and from bottom to top. During this traversal, the subdivision of the plane in a set of covering but non-overlapping tiles is refined. For example, when the left end point of an edge is encountered, we have discovered the lower-left corner of a new tile above the edge and the upper-left corner of the tile below the edge. This is illustrated in Figure 3.9(b).

When a right end point of an edge is encountered, we have discovered the lower-right corner of the tile above the edge and the upper-right corner of the tile below the edge, as depicted in Figure 3.9(c). Note from Figure 3.9(d) that an edge can have an arbitrary number of horizontally abutting tiles above and below it.

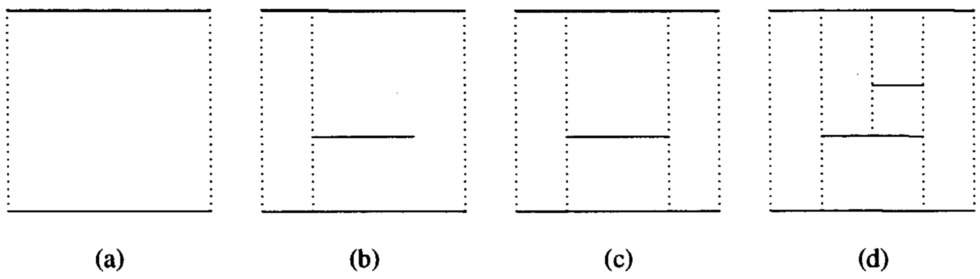


Figure 3.9. Refinement of tile subdivision.

The traversal of the plane is performed by the previously introduced scanline algorithm, in which the *handle* operation is replaced by the *updateTiles* operation defined below. The *updateTiles* operation is executed for every edge in the state ruler during the bottom-to-top traversal at every x-position. This operation then refines the tile subdivision. The *updateTiles* operation evaluates the plane-state function for all edges incident to the scanline. Based on the colors of the regions around the intersection of an edge with the scanline, tiles are completed and new tiles are begun.

A pseudocode description of the *updateTiles* operation appears in Algorithm 3.4. All edges contain a pointer to the tile directly above it in the slice of the plane represented by the state ruler. As the state ruler is traversed, a tile immediately to left of the current x,y position whose upper-right corner has not yet been determined, might be present. This tile is denoted by *freeTile.*

Tiles are created in stages. The updateTiles operation executes four operations, each one fixing one of the corners of a tile. These operation are *newTile, fixTL, fixBR* and *fixTR,* and for each individual tile they are executed in the sequence indicated. When a tile is completed, i.e. as a result of executing fixTR, it can be passed to the (application specific) visit operation. This operation, *enumTile,* is actually executed from within fixTR. Likewise, *enumPair* is executed from within fixTL, fixBR, and fixTR. For that purpose, these operations require some extra arguments specifying neighboring tiles. For the sake of conciseness, these arguments are omitted in Algorithm 3.4.

The operation of the algorithm is elucidated in Figure 3.10, where the numbered hooks refer to the line numbers in Algorithm 3.4. For an NMOS inverter, Figure 3.11 illustrates the layout (a), the tile subdivision, where the tile's color is represented by an octal number (b), and the tile subdivision for a rotated inverter (c). Figure 3.11(c) does not present a typical case, but clearly shows the trapezoidal and triangular shapes of the tiles.

3.6.1 Maintenance of a Corner Stitching Band

The *enumPair* operation already provides a certain amount of contextual information fully enabling non-trivial applications such as connectivity extraction, MOS device recognition, and overlap and sidewall capacitance computation [Meijs (1989), Meijs (1992)]. However, for other applications, for example those needing to measure the distance between neighboring features (like design rule checking or extraction of the capacitance between neighboring wires), the corner stitches that link the tiles

```
    procedure updateTiles (x, edge)
    begin
        y := Y (edge, x)                          # y ordinate at current abscissa
        prev_y := Y (edge.bwd, x)                 # y ordinate of lower neighbor edge
        if x = edge.xl                            # left end point
            color := color xor edge.color
            if freeTile = null
                freeTile := edge.bwd.tile
1               fixBR (freeTile, x, prev_y)
2               edge.bwd.tile := newTile (x, prev_y, freeTile.color)
3           fixTL (edge.bwd.tile, y)
4           edge.tile := newTile (x, y, color)
        else if x = edge.xr                       # right end point
            if freeTile = null
                freeTile := edge.bwd.tile
5               fixBR (freeTile, x, prev_y)
6               edge.bwd.tile := newTile (x, prev_y, freeTile.color)
7           fixTR (freeTile, y)
            freeTile := edge.tile
8           fixBR (freeTile, x, y)
        else                                      # edge straddling scanline
            color := color xor edge.color
            if freeTile ≠ null                    # two tiles below edge
9               fixTR (freeTile, y)
                freeTile := null
10              fixTL (edge.bwd.tile, y)
            if edge.tile.color ≠ color            # two tiles above edge
                freeTile := edge.tile             # because edge changes color
11              fixBR (freeTile, x, y)
12              edge.tile := newTile (x, y, color)
    end
```

Algorithm 3.4. The updateTiles algorithm

together can easily be maintained in the *updateTiles* operation. For conciseness, however, we have omitted these details from Algorithm 3.4.

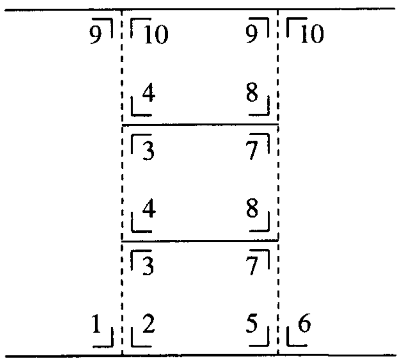


Figure 3.10. Illustration of the operation of updateTiles. The numbers correspond to the line numbers in Algorithm 3.4.

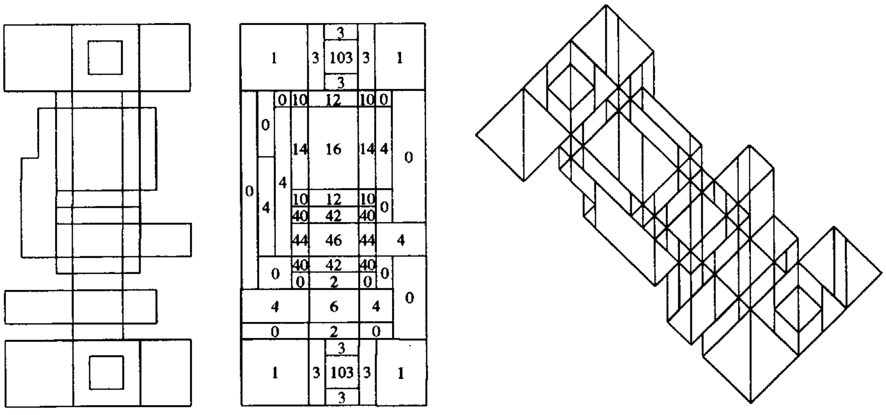


Figure 3.11. Layout of an inverter (a), its tile subdivision (b) and tiles for a rotated inverter (c).

A moving band of corner-stitched tiles can be realized by maintaining a FIFO-queue of completed tiles. A tile is put into the queue as soon as it is completed. Thus, the queue contains the tiles ordered according to their right abscissa, since this is the order in which the tiles are completed. (They are also ordered according to the ordinate of their upper-right corner, but this is irrelevant here.) Each time the scanline advances, the front of the queue is inspected to see which tiles leave the band so that their storage space can be reclaimed.

When the storage space of a tile is reclaimed, any stitches pointing to the tile become invalid. For most applications performing a neighborhood search, this will not appear to be a problem because invalid stitches are never traversed since they point outside the region of interest. Alternatively, it is possible to let the invalid stitches point to a special space tile or to explicitly check the left abscissa of a tile before traversing its left-pointing or down-pointing stitch.

Just prior to actually reclaiming storage, an application-specific *clearTile* operation is executed, enabling the application to take appropriate action. When there is no corner stitching band, the execution of enumTile is always immediately followed by the execution of clearTile and the actual deletion of the tile.

3.7 The Space Layout-to-Circuit Extractor

3.7.1 Description

The algorithms developed in this chapter have been implemented in an efficient layout-to-circuit extractor called *Space*. Space is a full-featured layout-to-circuit extractor that was developed with both accuracy and efficiency in mind. All extraction operations are performed in one scanline pass over the layout. These operations include device recognition, connectivity extraction, capacitance extraction and resistance extraction. While we will not describe the actual extraction algorithms (they are briefly described in [Meijs (1989)] and [Meijs (1992)]), we make the following remarks:

1. It must be noted that the operation of the circuit extractor differs fundamentally from that of the Magic circuit extractor [Scott (1985)]. Basically, the Magic extractor works *net-wise*. That is, the algorithm starts with one tile, marks it as belonging to net *n* if it has not yet been marked, and recursively marks all tiles connected to the first tile. Our algorithm works *tile-wise* and connectivity is resolved via a set-merging algorithm. There is a standard algorithm for efficiently performing set merging: a so-called *union-find* algorithm [Aho (1974), Tarjan (1983)].

2. Adopting a strategy to output all network elements as soon as possible, instead of keeping them in the computer's core memory, makes the (measured) space complexity almost proportional to the square root of the size of the input. To

implement this strategy efficiently, however, the standard union-find algorithm mentioned above has been extended to also implement deletions of elements from sets, as presented in [Meijs (1992)]. The efficiency of this latter operation appears to be paramount to the overall efficiency of the extractor.

The following is, in no particular order, an enumeration of some characteristics of Space.

Integrated in NELSIS. Space is an integrated part of the NELSIS IC Design System, a modular and open system consisting of a framework and a set of tools for chip design and verification [Wolf (1990)]. It works comfortably together with other tools in the system such as a layout editor and a network-comparison (graph isomorphism) tool. Tight integration of tools in NELSIS is achieved by adhering to common data formats and by using a standard *Data Management Interface* [Meijs (1987)] to obtain access to the design data and to register the actions of a tool with the framework. In this way, Space can intelligently deal with framework-specific aspects such as design versions, concurrency control and design flow management.

Capacitance Extraction. Besides a finite element method for capacitance extraction, to be described in Chapter 5, Space implements the capacitance model that was described in [Meijs (1984)]. The model implements first-order corrections to conventional area-perimeter models, resulting in reasonably accurate values of the ground and coupling capacitances. This model includes the coupling capacitance between neighboring conductors.

Resistance Extraction. Space implements a finite element method for resistance extraction as presented in [Genderen (1988), Genderen (1991)]. This method is much more accurate than polygon-partitioning-based heuristics, while nearly as efficient.

Lumped Approximation of RC Lines. Space employs a new, Elmore time-constant-preserving algorithm to transform a detailed finite element RC mesh into a low complexity lumped network that accurately models the distributed nature of the parasitic resistance and capacitance of IC interconnects [Genderen (1988), Genderen (1991)].

Hierarchical or Flat Mode of Operation. Space can operate in a hierarchical or a flat mode.

External Interfaces. The NELSIS system provides interfaces to several external formats, including the GDSII and CIF layout formats and the SPICE and EDIF net-list formats.

Technology Compiler. Crucial to the efficiency of Space is an *element lookup table.* Based on the color of a tile and the differences in color between two abutting tiles, this table can quickly be searched to find out which circuit elements are present in a tile. Since the construction and optimization of the lookup table from a user-defined technology file may take a long time (several CPU minutes) this task is separated into a dedicated technology compiler. Since Space then only has to read the pre-constructed lookup table, it can provide instant response for small designs.

45 Degree Geometries. The current version of Space can handle 45° polygonal geometry rather than general polygonal geometry. This allows the program to use integer calculations, which improves the efficiency.

MOS Devices only. The current version of Space can only recognize MOS devices. For complex MOS devices, such as those found in off-chip drivers, Space implements an algorithm to approximate the effective width and length of the device.

3.7.2 Performance

In this section, we will briefly present some measurements concerning the performance of Space. These results were obtained by a flat extraction of a number of randomly selected layouts of varying sizes (measured by the number of tiles and transistors) on an HP 9000/720 workstation. In the figures that follow, each data point corresponds to another design.

Figure 3.12 displays the maximum number of tiles in the core memory of the computer versus the total number of tiles. The straight line denotes a $O(\sqrt{N})$ behavior, for visual comparison. Of course, the deviations between observed and $O(\sqrt{N})$ behavior can partly be attributed to the fact that the aspect ratios of the layout are not unity.

Figure 3.13 displays the actual memory requirements. From the amount of memory used, we have subtracted the size of an internal table for element recognition, in order not to clutter the design size - memory relationship. The size of this table, typically 100-350 kbyte, depends only on the technology file—not on the size of the design. Although the results shown in Figure 3.13 clearly demonstrate a sub-linear space complexity, it is actually worse than $O(\sqrt{N})$. This can (partly) be explained by the fact that the average interconnection length increases with increased chip dimensions.

Finally, Figure 3.14 displays the extraction time versus the transistor count. This figure demonstrates that reducing space complexity need not be at the cost of increased computation times. The solid line represents a $O(N)$ complexity of 100 transistors/second.

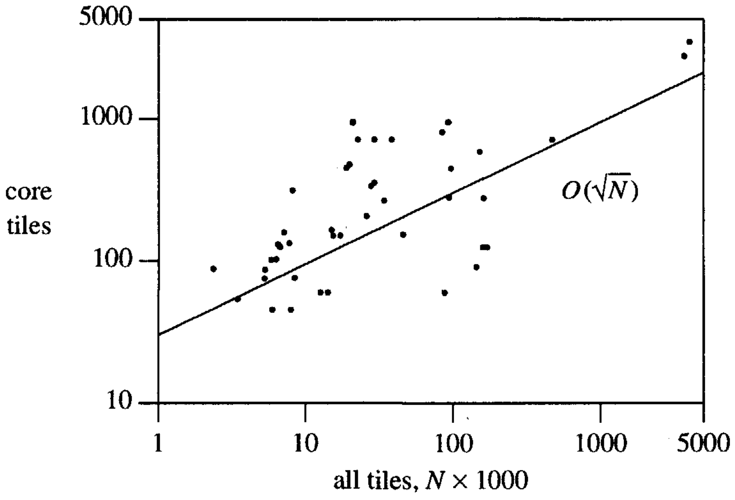


Figure 3.12. Tiles in core vs. all tiles.

3.8 Conclusion

We have presented an algorithm that combines the corner stitching method with the scanline technique. It performs a directed enumeration of all tiles and pairs of abutting tiles in linear time. This enumeration appears to be a valuable extension to the repertoire of corner stitching algorithms as originally presented in [Ousterhout (1984)]. Since the tiles only need to be retained in a narrow band sweeping over the layout, the memory requirements are reduced from $O(N)$ to almost $O(\sqrt{N})$, yielding a practical solution for batch-mode analysis of very large and relatively flat layouts. The tiles can have a trapezoidal (or triangular) shape to accommodate oblique geometry.

A key factor in the linear time complexity of the algorithm is the implementation of the scanline part, i.e. the linked list representation of the state ruler in combination with the method of handling edge intersections. It is worthwhile in this context to note that the

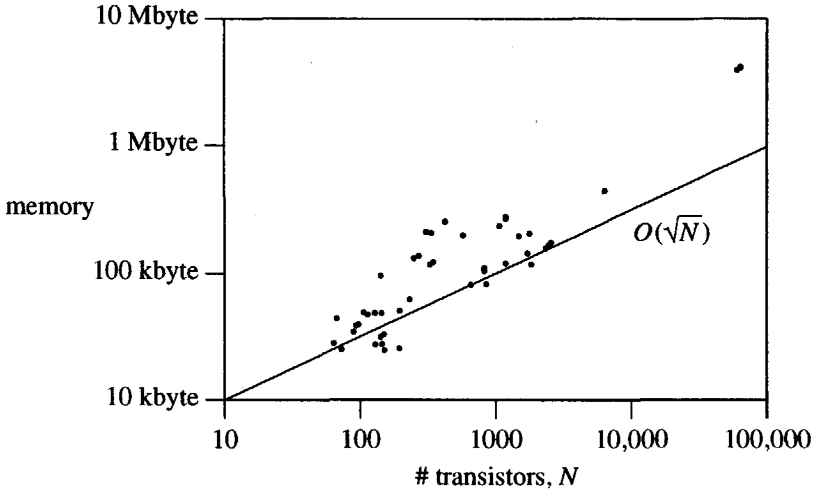


Figure 3.13. Memory vs. number of transistors.

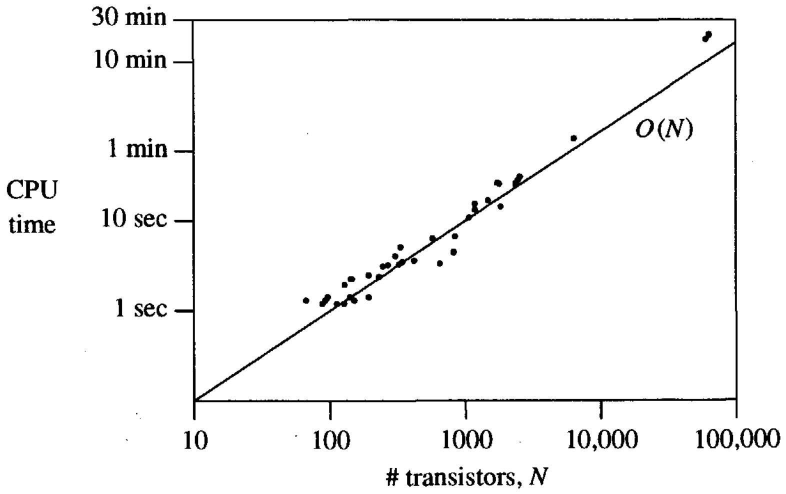


Figure 3.14. Extraction time vs. number of transistors, on an HP 9000/720.

edge intersection strategy presented in this chapter would allow a significant simplification of the scanline processor proposed in [Carlson (1986)] and

[Carlson (1987)]. In particular, it would be possible to omit the ''temporary queue'' used to store the results of edge intersections and all the associated hardware (the bus) and software (to sort the edges in this queue).

Many other applications of the region enumeration algorithm can be envisaged. Design rule checking is one obvious example: a moving corner stitching band whose width is slightly greater than the largest design rule distance enables all design rule violations to be detected. Other notable examples include raster plotting, fault extraction [Shen (1985)], pattern generator tape generation, and bipolar device recognition.

The reduced memory requirements eliminate the need to employ multiple corner stitched planes as in Magic [Ousterhout (1984), Ousterhout (1984a)], thereby avoiding plane cross-registering overhead. This is especially convenient in the case of strong interactions between different masks. For example, the fragmentation of the tile structure simplifies the computation of coupling capacitances: The exact overlap region (for parallel plate capacitance) as well as the edges bordering the region (for fringing capacitance) are explicitly represented in a fragmented single plane tile structure, while they must be computed in a multiple plane tile structure.

On the other hand, for applications such as design rule checking, the fragmentation resulting from the combination of all masks might be unprofitable, since in that case many mask interactions are really irrelevant. This problem can be overcome by separately checking unrelated rules. For pattern generator tape generation, a mask-wise operation seems to be most appropriate.

The geometrical algorithms developed in this chapter are used in Chapter 5, where we develop an algorithm for generating a 3-dimensional boundary element mesh from a 2-dimensional layout and a suitable description of the fabrication process. Moreover, we will implement a complex finite element method for capacitance extraction in a scanline fashion—using the enumPair/enumTile interface defined in Section 3.6.

References

Aho (1974)	A.V. Aho, J.E. Hopcroft, and J.D. Ullman, *The Design and Analysis of Computer Algorithms,* Addison Wesley, Reading, MA (1974).
Baird (1977)	H.S. Baird, "Fast algorithms for LSI artwork analysis," *Proc. 14th Design Automation Conference*, New Orleans, pp. 303-311 (June 1977).
Bentley (1979)	J.L. Bentley and T.A. Ottman, "Algorithms for reporting and counting geometric intersections," *IEEE Trans. on Computers* **C-28**(9) pp. 643-647 (Sept. 1979).
Bentley (1980)	J.L. Bentley and D. Wood, "A Optimal Worst Case Algorithm for Reporting Intersections of Rectangles," *IEEE Trans. Comp.* **C-29**(7) pp. 571-577 (Jul. 1980).
Bentley (1980a)	J.L. Bentley, D. Haken, and R. Hon, "Statistics on VLSI designs," CMU-CS-80-111, Carnegie-Mellon University, Pittsburgh, Pa. (1980).
Berger (1988)	J. Berger, "Un Sous-Systeme de Recherche Geometrique et d'Equivalence pour la CAO de Circuits Integres VLSI," Ph.D. Dissertation, Institut National Polytechnique de Grenoble, Grenoble, France (1988).
Brown (1981)	K.Q. Brown, "Comments on 'algorithms for reporting and counting geometric intersections'," *IEEE Trans. on Computers* **C-30**(2) pp. 147-148 (Feb. 1981).
Carlson (1986)	E.C. Carlson and R.A. Rutenbar, "A Data Structure Processor for VLSI Geometry Checking," *Proc. IEEE ICCAD-86*, Santa Clara, CA, pp. 404-407 (Nov. 11-13, 1986).
Carlson (1987)	E.C. Carlson and R.A. Rutenbar, "A Scanline Data Structure Processor for VLSI Geometry Checking," *IEEE Trans. on CAD* **CAD-6**(5) pp. 780-794 (Sept. 1987).
Chiang (1988)	K.W. Chiang and S. Nahar, "Time Efficient VLSI Artwork Analysis Algorithms in GOALIE2," *Proc. 25th Design Automation Conference*, Anaheim, CA, pp. 471-475 (June 1988).

Edelsbrunner (1981) H. Edelsbrunner and H.A. Maurer, ''On the intersection of orthogonal objects,'' *Inform. Processing Lett.* **13** pp. 177-181 (1981).

Fokkema (1983) J.T. Fokkema and T.G.R. van Leuken, ''An efficient datastructure and algorithm for VLSI artwork verification,'' *Proc. IEEE ICCD-83*, New York, pp. 350-353 (Oct. 1983).

Genderen (1988) A.J. van Genderen and N.P. van der Meijs, ''Extracting Simple but Accurate RC Models for VLSI Interconnect,'' *Proc. ISCAS-88*, Helsinki, Finland, pp. 2351-2354 (June 7-9, 1988).

Genderen (1991) A.J. van Genderen, ''Reduced Models for the Behavior of VLSI Circuits,'' Ph.D. Dissertation, Delft University of Technology, Delft, the Netherlands (1991).

Kedem (1982) G. Kedem, ''The quad-CIF tree: a data structure for hierarchical on-line algorithms,'' *Proc. 19th Design Automation Conference*, pp. 352-357 (June 1982).

Knuth (1973) D.E. Knuth, *The Art of Computer Programming, Volume 3, Sorting and Searching,* Addison Wesley, Reading, Mass. (1973).

Lauther (1981) U. Lauther, ''An O(N log N) algorithm for boolean mask operations,'' *Proc. 18th Design Automation Conference*, pp. 555-562 (1981).

Losleben (1979) Paul Losleben and Kathryn Thompson, ''Topological Analysis for VLSI Circuits,'' *Proc. 16th Design Automation Conference*, San Diego, CA, pp. 461-473 (Jun. 23-27, 1979).

Marple (1988) D. Marple, M. Smulders, and H. Hegen, ''An Efficient Compactor for 45° Layout,'' *Proc. 25th Design Automation Conference*, pp. 396-402 (June 1988).

Marple (1990) D. Marple, M. Smulders, and H. Hegen, "Tailor: A Layout System Based on Trapezoidal Corner Stitching,'' *IEEE Trans. CAD* **CAD-9**(1) pp. 66-90 (Jan. 1990).

McCreight (1980) E.M. McCreight, ''Efficient Algoritms for enumerating intersecting intervals and rectangles,'' Tech. Rep. PARC CSL-80-9, Xerox Palo Alto Res. Cent., Palo Alto, CA (1980).

Meijs (1984) N.P. van der Meijs and J.T. Fokkema, "VLSI circuit reconstruction from mask topology," *INTEGRATION, the VLSI Journal* **2**(2) pp. 85-119 (1984).

Meijs (1987) N.P. van der Meijs, T.G.R. van Leuken, P. van der Wolf, I. Widya, and P. Dewilde, "A Data Management Interface to Facilitate CAD/IC Software Exchanges," *Proc. IEEE ICCD '87*, pp. 403-406 (1987).

Meijs (1989) N.P. van der Meijs and A.J. van Genderen, "An Efficient Algorithm for Analysis of Non-Orthogonal Layout," *Proc. ISCAS-89*, pp. 47-52 (May 1989).

Meijs (1992) N.P. van der Meijs and A.J. van Genderen, "Space-Efficient Extraction Algorithms," *Proc. IEEE 3rd European Design Automation Conference*, Brussels, Belgium,(March 1992). (Accepted for publication.)

Nahar (1986) Surendra Nahar and Sartaj Sahni, "A Time and Space Efficient Net Extractor," *Proc. 23rd Design Automation Conference*, pp. 411-417 (Jun. 29 - Jul. 2, 1986).

Nievergelt (1982) J. Nievergelt and F.P. Preparata, "Plane-sweep algorithms for intersecting geometric figures," *Comm. of the ACM* **25**(10) pp. 739-747 (Oct. 1982).

Ousterhout (1984) J.K. Ousterhout, "Corner stitching: A data-structuring technique for VLSI layout tools," *IEEE Trans. on CAD* **CAD-3**(1) pp. 87-100 (Jan. 1984).

Ousterhout (1984a) J.K. Ousterhout, G.T. Hamachi, R.N. Mayo, W.S. Scott, and G.S. Taylor, "Magic: A VLSI Layout System," *Proc. 21st Design Automation Conference*, Albuquerque, New Mexico, pp. 152-159 (Jun. 25-27, 1984).

Ousterhout (1984b) J.K. Ousterhout, "The User Interface and Implementation of an IC Layout Editor," *IEEE Trans. on CAD* **CAD-3**(3) pp. 242-249 (Jul. 1984).

Ousterhout (1985) J.K. Ousterhout, G.T. Hamachi, R.N. Mayo, W.S. Scott, and G.S. Taylor, "The Magic VLSI Layout System," *IEEE Design and Test* *of Computers*, pp. 19-30 (Feb. 1985).

Preparata (1985) F.P. Preparata and M.I. Shamos, *Computational Geometry, an Introduction,* Springer-Verlag, New York (1985).

Scott (1985) W.S. Scott and J.K. Ousterhout, ''Magic's circuit extractor,'' *Proc. 22nd Design Automation Conference*, pp. 286-292 (1985).

Shen (1985) J.P. Shen and W. Maly, and F.J. Ferguson, ''Inductive Fault Analysis of MOS Integrated Circuits,'' *IEEE Design and Test of Computers* 2(6) pp. 13-26 (Dec. 1985).

Su (1987) S.L. Su, V.B. Rao, and T.N. Trick, ''HPEX: A Hierarchical Parasitic Circuit Extractor,'' *Proc. 24th Design Automation Conference*, pp. 566-569 (Nov. 1987).

Szymanski (1983) T.G. Szymanski and C.J. Van Wyk, ''Space efficient algorithms for VLSI artwork analysis,'' *Proc. 20st Design Automation Conference*, pp. 734-739 (1983).

Tarjan (1983) R.E. Tarjan, *Data Structures and Network Algorithms,* Society for Industrial and Applied Mathematics, Philadelphia (1983).

Willigen (1986) E. van Willigen and R. Nouta, ''A VLSI Artwork Analysis System using Dedicated Hardware and Syntactic Pattern Recognition,'' *Proc. IEEE ICCAD-86,* Santa Clara, CA, pp. 400-403 (1986).

Wolf (1990) P. van der Wolf, P. Bingley, and P. Dewilde, ''On the Architecture of a CAD Framework: The NELSIS Approach,'' *Proc. IEEE 1st European Design Automation Conference*, Glasgow, UK,(1990).

4. A Boundary Element Method for Capacitance Extraction

4.1 Introduction

As indicated in Chapter 2, there is an increasing need for accurate prediction of interconnect capacitances, to verify the correct functionality of a chip before fabrication. At the same time, however, this problem becomes more difficult because the lateral dimensions are decreasing more rapidly than the vertical dimensions. In fact, traditional approaches based on parallel-plate computations and heuristic, calibrated formulas to estimate the capacitances are becoming inaccurate. Instead, rigorous, mathematically sound techniques are required to model and determine the interconnection capacitance.

Many of these techniques are known, including the finite difference method [Dang (1981), Dierking (1982), Guerrieri (1987), Seidl (1988)], the finite element method in which the finite elements model the electrical field [Cottrell (1985)] and the boundary or Green's function finite element method [Ruehli (1975), Ruehli (1979), McCormick (1984), Ning (1987)], in which the finite elements model the conductor charges.

In this chapter, we present the mathematics and give some theoretical background of one of these methods, which we have implemented in our layout verification system. Chapter 5 describes experimental results as well as aspects involved in the method's actual implementation. Before proceeding, we first list the main requirements that must be satisfied by a successful mathematical technique for accurate extraction of IC interconnect capacitances.

First, the numerical procedures involved generally require a great deal of computer time and memory. Long computation times cannot be tolerated when these techniques are applied inside the *IC design loop*. Thus, it is imperative to choose the most efficient technique possible.

Second, the resulting capacitance model must contain all relevant capacitive couplings. However, it is not useful to determine each individual component with high precision. Because of tolerances in the fabrication process, the capacitance values present on the actual silicon display relatively large statistical variations. Hence, an accuracy on the order of 10% in each component is sufficient.

Third, the final result must be a reduced, lumped model of the distributed capacitance network formed by the interconnections. In other words, it must be as simple and compact as accuracy allows. The model must not contain small capacitances that only add irrelevant detail and, for example, unnecessarily complicate simulation of the circuit.

Fourth and last, the method must be suitable for implementation in a layout-to-circuit extractor, together with a (finite element) method for resistance extraction so that the distributed RC networks formed by the interconnections will be modeled accurately.

Based on these criteria, we have adopted a boundary finite element method that can be described briefly as follows:

1. For the purpose of modeling IC interconnections, it is sufficient to suppose that the chip is a stratified medium in which the conductors are floating. For such a medium, the potential in a point p can be written as

   $$\Phi(p) = \int_{\text{all charge}} G(p, q)\, \rho(q)\, dq$$

   where the so-called Green's function $G(p, q)$ can be interpreted as the potential induced at point $p = (x_p, y_p, z_p)$, due to a unit point charge at point $q = (x_q, y_q, z_q)$.

2. The above equation can be transformed into a matrix equation by discretizing the charge that is present on the conductors as a piecewise linear and continuous distribution on a set of *finite elements.*

3. This matrix equation can be written as:

   $$\phi = G\,\sigma$$

   where $\phi = [\phi_1\ \phi_2\ \cdots\ \phi_N]^T$ and $\sigma = [\sigma_1\ \sigma_2\ \cdots\ \sigma_N]^T$ collect the finite element potentials and the finite element charges respectively, and G_{ij} is the potential induced at node i by the charge at finite element j.

4. Using this equation, we can compute the conductor capacitances as follows: Let A be an incidence matrix relating finite elements to conductors, i.e. A_{ij} is 1 if element i lies on conductor j, and 0 otherwise. Also, let $V = [V_1\ V_2\ \cdots\ V_M]^T$ be the vector of conductor potentials and $Q = [Q_1\ Q_2\ \cdots\ Q_M]^T$ the vector of charges on the conductors. Then:

$$Q = A^T \sigma = A^T G^{-1} \phi = A^T G^{-1} A V = C_s V$$

Here,

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

is the capacitance matrix to be obtained.

Consequently, we need to invert a matrix G. G can be very large: for a circuit with 500 conductors and only 50 finite elements per conductor, G is 25000×25000 in size. The computation of G^{-1} requires $O(N^3)$ time and $O(N^2)$ storage. These values are clearly prohibitive and restrict the applicability of the method to small test cases without any practical significance. Moreover, C_s is a full matrix. That is, it specifies a capacitance between every pair of conductors—clearly not a reduced model.

However, [Nelis (1989)] defines an algorithm that avoids the complete inversion of G and with much greater efficiency delivers a sparse approximation of G^{-1}. The result is a reduced capacitance model which ignores small capacitances between conductors that are physically ''far'' from each other. The time complexity of the approximate matrix inversion technique is optimal: proportional to the square of the number of pairs of nearby finite elements. Consequently, since the average density of finite elements in actual layouts can be considered constant, the running time is proportional to the size of the layout.

Before detailing the technique in subsequent subsections, we make the following assumptions/restrictions:

First, we consider the electrostatic case only. In the majority of cases, this assumption is a valid approximation of the physical situation, it only becomes invalid with signals approaching the Giga-hertz spectrum. A very important practical consequence for the electromagnetic behavior of macroscopic matter is that Maxwell's equations are reduced to electrostatic equations, such as the Laplace equation.

Second, we assume that the silicon substrate forms a ground plane. In effect, the field implant functions as a highly conducting electrical plate. At very high operating frequencies, however, the entire substrate plays a role [Hasegawa (1971)].

Third, we shall assume that the medium is a stratified medium with each layer i having a known and constant permitivity ε_i. In practice, these layers may be SiO_2, Si_3N_4 or air. This assumption corresponds to the case in which the surface of the IC is perfectly planarized between each interconnect deposition step. Planarization is becoming standard, in order to decrease step coverage problems and depth-of-focus problems that would otherwise become more severe when the dimensions are reduced. However, the method can be extended to the case of no planarization.

Fourth and last, for purposes of this discussion we assume that the conductors are perfect and that each conductor forms an equipotential. Resistivity of the conductors is brought in later, by subdividing them into parts for which the equipotential assumption is approximately valid.

The rest of this chapter is structured as follows. In Section 4.2, we give some relevant definitions for multiconductor capacitance networks. Subsequently (Section 4.3), we present the Green's function formulation as a solution to the electrostatic field equations and explain how it can be solved using the boundary finite element method (Section 4.4). Section 4.5 discusses the resulting matrix inversion problem, for which an effective algorithm is described in Sections 4.6 and 4.7.

4.2 Multiconductor Capacitances

4.2.1 Electrostatics

Consider a system of charged conducting bodies embedded in a homogeneous dielectric medium with permitivity ε and of infinite dimensions, as illustrated in Figure 4.1. The electric field surrounding the conductors is governed by the Laplace equation

$$\nabla^2 \Phi(p) = 0 \tag{4.1}$$

with suitable boundary conditions, where ∇^2 denotes the Laplacian

$$\nabla^2 = \frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2} + \frac{\partial^2}{\partial z^2} \tag{4.2}$$

for a three-dimensional space, and $\Phi(p)$ is the electrostatic potential in a point $p = (x_p, y_p, z_p)$.

A formal solution of this equation can be written (see e.g. [Stakgold(1968)]) as follows:

$$\Phi(p) = \int_{all\ charge} \frac{1}{4\pi\varepsilon\,|p - q|}\,\rho(q)\,dq \tag{4.3}$$

Here, $\rho(q)$ is the charge density at point $q = (x_q, y_q, z_q)$, and $|p-q|$ denotes the

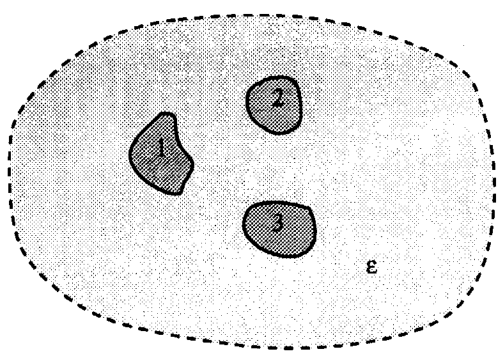


Figure 4.1. Charged conducting bodies in an infinite dielectric medium.

Euclidian distance between two points p and q. The factor $1/4\pi\varepsilon|p-q|$ can be interpreted as the potential induced at point p due to a unit point charge at q. Given this interpretation, the integral over all charge in Equation (4.3) can be understood intuitively by noting that the Laplace equation is linear and applying the superposition principle. Thus, given $\rho(q)$, the electrostatic potential $\Phi(p)$ follows from Equation (4.3).

While Equation (4.3) establishes a global relationship between potential and charge for a multiconductor system, there exists a similar relationship among the conductors. This is written in matrix notation as:

$$V = G\,Q \tag{4.4}$$

where $V = [V_1\ V_2 \cdots V_N]^T$ and $Q = [Q_1\ Q_2 \cdots Q_N]^T$ collect the conductor voltages and charges, respectively, and G is an $N \times N$ matrix whose entries G_{ij} are called the "coefficients of potential" [Ruehli (1975)]. It can be shown that G is symmetrical and positive definite.

4.2.2 Short Circuit Capacitance Matrix

The inverse C_s of G is the so-called short-circuit capacitance matrix of the system of conductors. It expresses the charge Q_i on conductor i as a function of the voltages $V_1, V_2, \cdots V_n$:

$$Q = C_s\,V \tag{4.5}$$

The short-circuit capacitance matrix derives its name from the fact that 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. Like G, C_s is also symmetrical and positive definite.

4.2.3 Network or two-terminal capacitances

In an equivalent circuit, the value of a capacitor is the ratio of the free charge associated with a voltage difference between two conductors or between a conductor and the reference (e.g. the ground plane or the point at infinity), and that voltage difference. The values of these capacitors are known as the two-terminal or network capacitances. We are usually interested in these values instead of in the short-circuit capacitance matrix because an equivalent circuit is the most common input for circuit simulators and other timing verification tools. As an example, the equivalent circuit of Figure 4.1 is given in Figure 4.2.

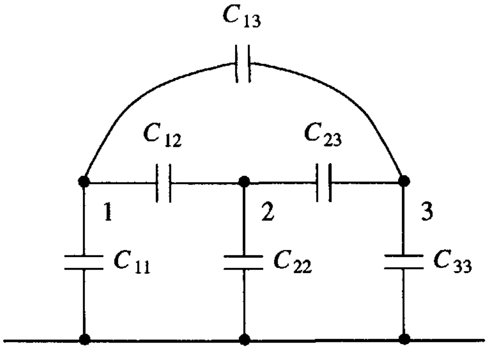


Figure 4.2. Equivalent circuit of the interconnections of Figure 4.1.

From Equation (4.5), we have with $V_{ij} \equiv V_i - V_j$, $i \neq j$:

$$\begin{aligned} Q_1 &= (C_{s_{11}} + C_{s_{12}} + \cdots + C_{s_{1N}})V_1 - C_{s_{12}}V_{12} - C_{s_{13}}V_{13} - \cdots - C_{s_{1N}}V_{1N} \\ Q_2 &= (C_{s_{21}} + C_{s_{22}} + \cdots + C_{s_{2N}})V_2 - C_{s_{21}}V_{21} - C_{s_{23}}V_{23} - \cdots - C_{s_{2N}}V_{2N} \\ &\vdots \\ Q_N &= (C_{s_{N1}} + C_{s_{N2}} + \cdots + C_{s_{NN}})V_N - C_{s_{N2}}V_{N2} - C_{s_{N3}}V_{N3} - \cdots - C_{s_{NN}}V_{NN} \end{aligned} \tag{4.6}$$

This can be written as

$$Q_i = C_{ii}\,V_i + \sum_{j=1}^{N} C_{ij}(V_i - V_j) \qquad i = 1, 2, \cdots N \tag{4.7}$$

where the two-terminal capacitances are given by

$$C_{ii} = \sum_{j=1}^{N} C_{s_{ij}} \qquad i = 1, 2, \cdots, N \tag{4.8a}$$

$$C_{ij} = -C_{s_{ij}} \qquad i \neq j \tag{4.8b}$$

C_{ii} is the ground capacitance between conductor i and the reference and C_{ij} is the coupling capacitance between conductors i and j.

The inverse relationship of Equation (4.8) specifies the short-circuit capacitances in terms of the network capacitances:

$$C_{s_{ii}} = \sum_{j=1}^{N} C_{ij} \qquad i = 1, 2, \cdots, N \tag{4.9a}$$

$$C_{s_{ij}} = -C_{ij} \qquad i \neq j \tag{4.9b}$$

and we note that the diagonal entries $C_{s_{ii}}$ of C_s are important characteristic figures of a multiconductor system. They are the sums of the ground and coupling capacitances for each conductor i. As such, they represent the total capacitive load a conductor forms for its driving stages when all other conductors are shorted to ground or driven by low impedance sources. Therefore, they are frequently used for delay calculations in digital circuits, see e.g. [Genderen (1989)].

4.2.4 Total Capacitance

Alternatively, the total capacitance $C_{t_{ii}}$ of conductor i is the load it forms for its driving stage when all other conductors are floating:

$$C_{t_{ii}} = \frac{\Delta Q_i}{\Delta V_i} \qquad \Delta Q_j = 0, \quad j \neq i \tag{4.10}$$

For a system of conductors as in Figure 4.1, we can determine $C_{t_{ii}}$ as follows: Assume an initial distribution of charges over the conductors. Changing the potential of conductor i by an amount ΔV_i, changes its charge by an amount ΔQ_i. When the other conductors are floating, their potential will change by an amount ΔV_j, while their charges will not change: $\Delta Q_j = 0$. Since the system is linear, we get from Equation

(4.4):

$$\Delta V = G\,\Delta Q \tag{4.11}$$

where $\Delta V = [\Delta V_1\ \Delta V_2 \cdots \Delta V_i \cdots \Delta V_N]^T$ and $\Delta Q = [0\ 0 \cdots \Delta Q_i \cdots 0]^T$ and we conclude that $\Delta V_i = G_{ii}\,\Delta Q_i$. Hence:

$$C_{t_{ii}} = \frac{1}{G_{ii}} \tag{4.12}$$

The total capacitance between conductors i and j is similarly defined as the amount of charge moved from conductor i to j when a voltage is applied to these conductors, with all other conductors floating. Using Equation (4.11), where $\Delta Q = [0\ 0 \cdots\ \Delta Q_i \cdots \Delta Q_j \cdots 0]^T$ with $\Delta Q_j = -\Delta Q_i$, we find $\Delta V_i - \Delta V_j = (G_{ii} - 2G_{ij} + G_{jj})\Delta Q_i$. Hence:

$$C_{t_{ij}} = \frac{1}{G_{ii} - 2G_{ij} + G_{jj}} \tag{4.13}$$

4.2.5 Crosstalk

Crosstalk is the noise on an interconnection caused by the switching of other interconnections. Capacitive coupling is frequently the primary cause of crosstalk. This manifests itself most strongly when the non-driven conductors have a high impedance. In this case, the crosstalk amplitude depends on the total capacitance values. In particular, if there is a step ΔV_i on conductor i, the resulting spike ΔV_j on conductor j can be computed by using Equation (4.11):

$$\Delta V_i = G_{ii}\,\Delta Q_i \tag{4.14a}$$

$$\Delta V_j = G_{ij}\,\Delta Q_i \tag{4.14b}$$

By eliminating ΔQ_i, we obtain

$$\frac{\Delta V_j}{\Delta V_i} = \frac{G_{ij}}{G_{ii}} \tag{4.15}$$

This relation can be used in circuit analysis tools to select, for example, the k most strongly coupled pairs of conductors for further analysis.

4.3 The Green's Function

Equation (4.3) gives the electrostatic potential due to a number of conducting bodies in a homogeneous dielectric medium of infinite dimensions. In this section, we wish to find this potential for the case of on-chip integrated circuit interconnections. We shall not perform a formal derivation of the equations dealt with here, but only justify them intuitively. For a formal derivation, see e.g. [Dewilde (1990)].

First, reconsider Equation (4.3). We may rewrite this equation as

$$\Phi(p) = \int_{all\ charge} G(p, q)\, \rho(q)\, dq \tag{4.16}$$

where

$$G(p, q) = \frac{1}{4\pi\varepsilon |p - q|} \tag{4.17}$$

is the so-called Green's function for a uniform dielectric of infinite dimensions and with permitivity ε. The Green's function can be interpreted as the potential induced at point p, due to a unit point charge at q. Point p is called the *observation point* and q is called the *source point.* The Green's function is symmetrical: $G(p, q) = G(q, p)$.

The Green's function is characteristic of and dependent on the geometry of the dielectric medium. In other words, Equation (4.16) can be used for cases with different dielectric properties if a suitable Green's function is available. The problem is then to derive the Green's function for the dielectric configurations that are encountered in integrated circuits, as schematically illustrated in Figure 4.3. Under the simplifying—but not unreasonable—assumptions made in Section 4.1, this will indeed appear to be possible.

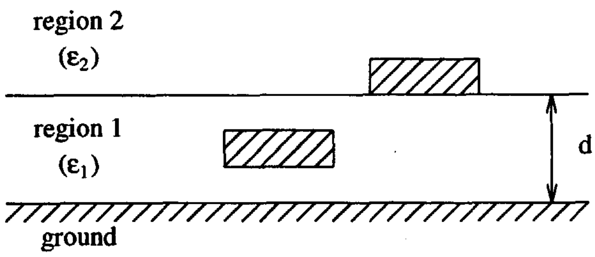


Figure 4.3. A 2-layer stratified dielectric.

Starting from Equation (4.17), the Green's function for the case with a ground plane added to a uniform halfspace can be developed as follows, see e.g. [Silvester(1968)]. Consider a semi-infinite dielectric half space with a conducting ground plane at $z = 0$, as in Figure 4.4(a). A system of point charges above this plane is equivalent to a system without the ground plane, but with a "reflection" of each charge about the ground plane, where the charge is equal in magnitude but opposite in sign (Figure 4.4(b)). These reflected charges are called "images".

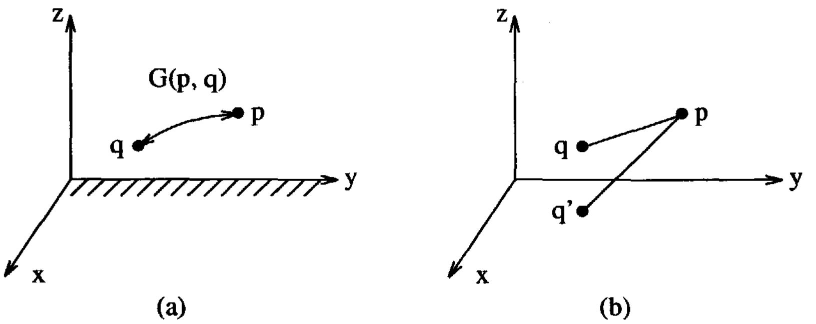


Figure 4.4. Illustration of the method of images.

With the coordinates of p and q denoted by (x_p, y_p, z_p) and (x_q, y_q, z_q), respectively, and with r_l the lateral distance between p and q, $r_l = \sqrt{(x_p-x_q)^2 + (y_p-y_q)^2}$, we have

$$G(p, q) = \frac{1}{4\pi\varepsilon}\left\{\frac{1}{\sqrt{(z_p-z_q)^2 + r_l^2}} - \frac{1}{\sqrt{(z_p+z_q)^2 + r_l^2}}\right\} \tag{4.18}$$

The second term can be seen as a correction to Equation (4.17) necessitated by the presence of a ground plane. If the ground plane is moved to $z = -\infty$, this term vanishes, leaving Equation (4.17).

This way of developing the Green's function, i.e. by transforming the domain to a homogeneous domain while adding image charges, is called the *method of images.* It can also be used for the case of a stratified dielectric medium as applicable to VLSI interconnect capacitance. This was indeed done in, for example, [Weeks (1970)] for the 2-dimensional case. For the 3-dimensional case, however, it was found to be easier to use a Fourier integral method [Ning (1987), Ning (1989)].

For illustrating the resulting Green's functions, we consider a 2-layer dielectric as shown in Figure 4.3. When both the source point and the observation point are in dielectric layer 1, the Green's function may be written as

$$G_{11}(p, q) = \frac{1}{4\pi\varepsilon_1}\left\{\frac{1}{\sqrt{(z_p - z_q)^2 + r_l^2}} - \frac{1}{\sqrt{(z_p + z_q)^2 + r_l^2}}\right.$$
$$+ \sum_{n=0}^{\infty} (-1)^n K^{(n+1)} \left[\frac{1}{\sqrt{[2(n+1)d - (z_p + z_q)]^2 + r_l^2}} - \frac{1}{\sqrt{[2(n+1)d + (z_p - z_q)]^2 + r_l^2}}\right.$$
$$\left.\left. + \frac{1}{\sqrt{[2(n+1)d + (z_p + z_q)]^2 + r_l^2}} - \frac{1}{\sqrt{[2(n+1)d - (z_p - z_q)]^2 + r_l^2}}\right]\right\} \qquad (4.19)$$

where ε_1 and ε_2 denote the dielectrical constant of region 1 and region 2, respectively, d denotes the thickness of SiO_2 and $K = (\varepsilon_1 - \varepsilon_2)/(\varepsilon_1 + \varepsilon_2)$. Similar formulas have been developed for other locations of source and/or observation points and for more complex dielectric configurations [Ning (1987), Ning (1989)].

This Green's function also has a clear physical interpretation. In fact, the first two terms of $G_{11}(p, q)$ are identical to Equation (4.18). The third term now presents a correction to that equation, due to the presence of an extra dielectric. It can easily be verified that if $\varepsilon_1 = \varepsilon_2$, Equation (4.18) remains.

4.4 Solution of the Green's Function Formulation

4.4.1 Boundary Element Method

A formal solution of the Laplace equation was given by Equations (4.3) and (4.16). Since the charge is only present on the surface S of the conductors, we can replace the integration over all charge by an integration over S only. Equation (4.16) can then be written as follows:

$$\Phi(p) = \int_S G(p, q)\, \rho(q)\, dq \qquad (4.20)$$

In this equation, p and q are 3-dimensional variables but the integration is a double integration. We now investigate how, given a known Green's function (see Section 4.3), we can actually solve this equation numerically on a computer.

The computational tool that we shall use belongs to the class of *finite element methods*. It is specifically known as a *boundary element method* [Brebbia (1989)] or a *method of moments* [Harrington (1968)]. With this method, the domain of a function f to be approximated is partitioned into a number of sub-domains (finite elements). A local approximation of f is defined on each sub-domain, such that their assembly is a piecewise approximation of f.

In our case, the function $\rho(q)$ is approximated as

$$\rho(q) \approx \tilde{\rho}(q) = \sum_{i=1}^{N} \sigma_i f_i(q) \tag{4.21}$$

Here, the f_i's are N independent *shape functions* (also called *basis* or *expansion* functions), which are non-zero only on a part S_i of S and which have unity weight. In particular:

$$\int_{S_j} f_i(q)\,dq = \begin{cases} 1 & \text{if } i = j \\ 0 & \text{if } i \neq j \end{cases} \tag{4.22}$$

We then obtain an approximation $\tilde{\Phi}(p)$ of $\Phi(p)$ as follows:

$$\begin{aligned} \tilde{\Phi}(p) &= \int_S G(p,q) \sum_{i=1}^{N} \sigma_i f_i(q)\,dq \\ &= \sum_{i=1}^{N} \sigma_i \int_{S_i} G(p,q) f_i(q)\,dq \end{aligned} \tag{4.23}$$

The σ_i's are then N independent constants to be determined so as to obtain a good approximation of $\Phi(p)$. For N unknown constants, we need N independent equations. These can be obtained by using the theory of the *method of moments* [Harrington (1968)]. For that purpose, Equation (4.23) can be written as

$$\tilde{\Phi}(p) = \sum_{i=1}^{N} \sigma_i g_i(p) \tag{4.24}$$

where $g_i(p) = \int_{S_i} G(p,q) f_i(q)\,dq$. The general method of moments formulation of Equation (4.24) would be written as:

$$\begin{aligned} <\Phi(p), w_j(p)> &= <\tilde{\Phi}(p), w_j(p)> \\ &= <\sum_{i=1}^{N} \sigma_i g_i(p), w_j(p)> \\ &= \sum_{i=1}^{N} \sigma_i <g_i(p), w_j(p)> \qquad j = 1 \cdots N \end{aligned} \tag{4.25}$$

where $<.,.>$ is a suitable bi-linear product and $w_j(p)$, $j = 1 \cdots N$, is a set of weighting functions aimed at "averaging out" the approximation error[1]. The weighting functions are defined on the domain of $\Phi(p)$, the surfaces of the conductors. The solution of the simultaneous equations results in the desired linear combination of the N independent basis functions f_i.

A suitable bi-linear product for our problem is [Harrington (1968)]:

$$< \Phi(p),\ w_j(p) > \ = \int_S \Phi(p) w_j(p)\, dp$$

where the (two-dimensional) integral extends over the conductor surfaces.

Suitable weighting functions are defined below, but in the case of the finite element method they are also defined in a piecewise fashion, like the shape functions. In that case, $w_j(p)$ is non-zero only on the part S_j of S. Hence, Equation (4.25) can be written in matrix notation as:

$$\phi = G\, \sigma \tag{4.26}$$

where $\phi = [\phi_1, \phi_2, \cdots, \phi_N]^T$, $\sigma = [\sigma_1, \sigma_2, \cdots, \sigma_N]^T$ and

$$\phi_j = \ < \Phi(p),\ w_j(p) > \ = \int_{S_j} \Phi(p)\, w_j(p)\, dp \tag{4.27}$$

$$G_{ji} = \ < g_i(p),\ w_j(p) > \ = \iint_{S_j S_i} G(p,\ q)\, w_j(p)\, f_i(q)\, dq\, dp \tag{4.28}$$

We call G the *influence matrix.*

A convenient choice for the weighting functions is $w_j = f_j$, i.e. the weighting functions are made equal to the shape functions. This is known as *Galerkin's method.* A Galerkin solution leads to symmetric matrices, which is attractive from a computational point of view as well as from a physical one, since the physical system is symmetrical. We should avoid non-symmetrical models because they do not conserve energy and, under certain circumstances, give rise to numerical instabilities in circuit simulators. This may be manifested as non-convergence of a simulation of the network.

1. By the linearity of the bi-linear product, this is equivalent to the *weighted residual formulation*, see e.g. [Zienkiewicz (1983)]: $< \Phi(p) - \tilde{\Phi}(p),\ w_j(p) > \ = 0,\ j = 1 \cdots N$.

With Galerkin's method, Equation (4.27) reduces to:

$$\phi_j = \Phi(p_j) \tag{4.29}$$

where p_j is some point on S_j, because the potential is constant over a finite element and because $\int_{S_j} w_j(p)\,dp = 1$ since the weighting functions are normalized. Equation (4.28) reduces in the case of Galerkin's method to

$$G_{ij} = G_{ji} = \int_{S_j}\int_{S_i} G(p, q) f_j(p) f_i(q)\,dq\,dp \tag{4.30}$$

We may also select $w_j(p) = \delta(p - p_j)$ where δ is the Dirac delta function and p_j is some point on S_j. This particular choice is called a *point collocation* (or *point matching*) solution, since it matches the superposition of the basis functions to the function to be approximated at N points p_j. This choice make evaluating the bi-linear product trivial and obtaining G as simple as possible. The resulting matrix is not symmetrical, but can be made so by averaging the entries in symmetrical positions. This is acceptable because, with a sufficiently regular finite element mesh, the entries G_{ij} and G_{ji} are close.

With the collocation method, Equation (4.27) also reduces to Equation (4.29). Equation (4.28) reduces to:

$$G_{ji} = \int_{S_i} G(p_j, q) f_i(q)\,dq \tag{4.31}$$

because

$$\int_{S_j} G(p, q)\,\delta(p - p_j) = G(p_j, q)$$

A point collocation solution effectively amounts to evaluating $\tilde{\Phi}(p)$ at N different positions p_j, $j = 1 \cdots N$, with $p_j \in S_j$ and requiring equality with $\Phi(p)$ at those positions.

4.4.2 Incidence Between Finite Elements and Conductors

After having established a computational procedure for solving Equation (4.20), the remaining problem is to relate the solution to the multiconductor capacitances. In order to do that, we define an incidence matrix A with N rows (N is the total number of finite elements in the system) and M columns (M is the number of conductors). Since, in general, each conductor contains many finite elements, A is non-square with $N \gg M$.

A is given a value as follows:

$$A_{ij} = \begin{cases} 1 & \textit{if finite element } i \textit{ is on conductor } j \\ 0 & \textit{otherwise} \end{cases} \tag{4.32}$$

First, bear in mind that Q is the vector of conductor charges. The charge on a conductor is the sum of all elemental charges, which can be expressed as

$$Q = A^T \sigma \tag{4.33}$$

Second, bear in mind that V is the vector of conductor potentials. In the electrostatic case, the element voltages are the same for all elements on a conductor. Hence, V and ϕ are related as follows:

$$\phi = A\, V \tag{4.34}$$

Then, using Equations (4.26), (4.33) and (4.34), we have

$$Q = A^T \sigma = A^T G^{-1} \phi = A^T G^{-1} A\, V \tag{4.35}$$

By comparing this with Equation (4.5), we find

$$C_s = A^T G^{-1} A \tag{4.36}$$

which is the desired result.

4.4.3 Shape Functions

The discretization of the charge as required by Equation (4.21) can be accomplished in various ways, with different results. For example, the surface S of all conductors can be partitioned into N finite elements of uniform [Patel (1971)] or non-uniform area S_k, $k = 1 \cdots N$ [Benedek (1972), Balaban (1973), Ruehli (1973)]. On each element, the charge can be approximated by a constant [Patel (1971), Ruehli (1973)] or a non-constant function [Benedek (1972), Balaban (1973)]. In all methods, the error is reduced when the number of finite elements is increased, but this is obviously at the cost of CPU time.

Choosing the finite elements and the shape functions in accordance with the form of the exact solution, improves the convergence characteristics of the method. This means that for a given number of finite elements, the computation time needed to achieve a certain accuracy is minimized. For example, [Ruehli (1973)] exploited the fact that the charge distribution on a rectangular conductor in free space is much larger on the edges than in the center, by making the finite elements near the edges smaller than those near the center.

However, such heuristics are expected to be less useful with the irregular geometries of crossing and neighboring interconnects found in actual IC's. For this reason, and because of the fact that a moderate accuracy is satisfactory, we did not experiment with such heuristics. In our prototype implementation of the capacitance extraction method, to be described in Chapter 5, we have implemented constant and linear elements of non-uniform area.

When constant elements are used, the global charge distribution becomes a piecewise constant function defined over the surfaces of the conductors, as illustrated in Figure 4.5(a). The geometrical shape of a constant element is really arbitrary, although numerical procedures are greatly simplified when the elements have some simple and regular shape. Triangles and quadrilaterals are often used. Figure 4.5(a) illustrates the case of rectangular elements.

When linear elements are used, the global charge distribution becomes a piecewise linear function, as shown in Figure 4.5(b). For this case, the finite element discretization must be a triangulation. The global charge distribution becomes continuous when individual elements have a form and shape function as illustrated in Figure 4.5(c) and when they overlap each other such that every vertex of the triangulation is a center node of a finite element.

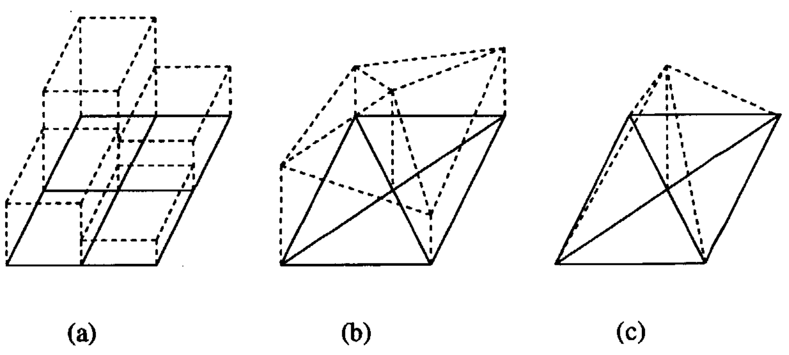


Figure 4.5. Illustration of different shape functions.

4.4.4 Discussion.

We assumed in Section 4.1 that the surface of an IC has a stratified structure. With this assumption, a Green's function could be developed (Section 4.3). Actually, this assumption is only true when the surface of an IC is *planarized* between successive patterning steps. Without planarization, the IC surface may actually be very irregular and a Green's function that assumes planarization will be inaccurate. In that case, the Green's function for one dielectric above a ground plane (Equation (4.18)) can be used if the integration domain is extended to include the bound charge on the dielectric interfaces as well. This approach was, for example, followed in [Ruehli (1979)] and for the 2-dimensional case in [Wei (1984)]. In [Janak (1989)], an approach is described that combines the use of a Green's function for a stratified dielectric with explicit treatment of the bound charge at finite and irregular dielectric interfaces.

4.5 Matrix Inversion

In Section 4.4 we obtained the following expression for the short-circuit capacitance matrix:

$$C_s = A^T G^{-1} A \tag{4.37}$$

With N the number of finite elements, G is an $N \times N$ matrix of Green's function influences. As G is a full matrix, the time needed to construct G (the total time to evaluate the Green's functions) and its storage complexity are both $O(N^2)$. Even worse, the inversion of G requires $O(N^3)$ time. An algorithm with such a time complexity quickly becomes computationally intractable on even the fastest computer.

For example, consider a problem with M conductors containing 50 finite elements each. Figure 4.6 illustrates the time needed for matrix inversion (double precision floating point arithmetic) as a function of the number of conductors for 2 different types of hardware. The first is an HP 9000/840 computer and the second is a hypothetical one with 100 times the floating point performance.

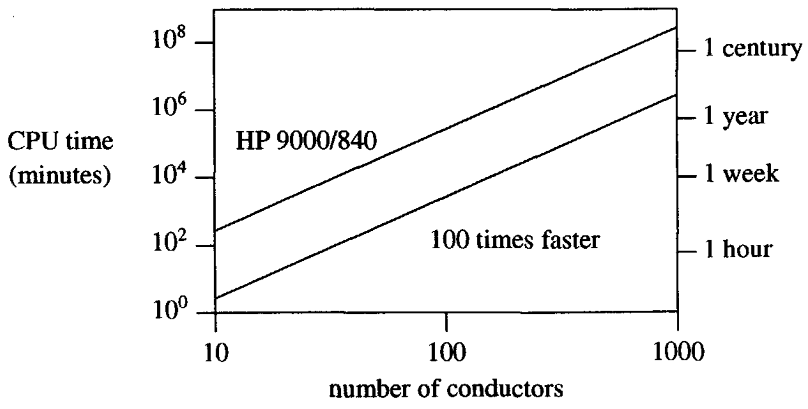


Figure 4.6. Illustration of the time needed for matrix inversion.

These results clearly indicate that matrix inversion is a costly step in the computation of capacitances, and the Green's function method seems therefore to be limited to small problems of little practical significance. Indeed, results reported for 3-dimensional capacitance computations are often for simple configurations with only a small number of conductors.

This matrix inversion bottleneck was also realized in [Nabors (1989)]. Their iterative algorithm for inversion of the matrix computes the iterates via a so-called *multipole approximation*. Basically, that method replaces the charge in a cluster of closely spaced finite elements by only a few charges located around their electrical center of gravity. This approximation is then used to compute the aggregate contribution of all the finite elements in the cluster to the potential at other points, at some distance from the cluster. The accuracy of this approximation improves as the distance between the cluster and the points in which the potential is computed increases.

This method is similar to efficient solutions of the classical N-body problem of computing interactions between objects in a gravitational field. These solutions approximate the force operating on individual objects by the force exerted by large clusters.

With respect to the speedup, it is obvious that the time needed to compute the influence matrix is decreased, since it does not need to be evaluated completely. The time for matrix inversion also decreases. In fact, the multipole approximation reduces the time complexity from $O(N^3)$ to $O(mN)$, where m is the number of conductors.

Although this is a significant speedup, we will see in what follows that it is possible to achieve even better results. However, there is another problem associated with Equation (4.37), which is not solved by the multipole approximation: The inversion of G gives a full matrix G^{-1} and, consequently, a full C_s. That is, C_s specifies a capacitance between every pair of conductors. However, most of the elements of C_s are small and have only a negligible effect on the circuit's electrical characteristics such as timing and crosstalk.

In addition, the large number of capacitances will require an excessively large amount of disk space for the database storage of the circuit and will severely strain the consumers of the circuit description, whether they are design tools that can break down because of internal limits being exceeded or designers who cannot comprehend the level of detail involved.

Thus, we should look for an approximation $\tilde{C}_s$ of C_s, which can be computed efficiently, which contains as many zeros as possible and yet accurately models the electrical behavior of the circuit. This can be accomplished if the non-zeros in $\tilde{C}_s$ correspond to interactions between neighboring conductors and the zeros correspond to the negligible coupling between widely separated conductors.

To find $\tilde{C}_s$, we might (erroneously) be tempted to neglect the influence between finite elements that are far apart. That is, we wish not to compute the Green's function between these elements and to make the corresponding entry in the influence matrix equal to zero. It might be hoped that it would then be more efficient to compute $\tilde{C}_s$, since we could use sparse matrix techniques [Duff(1986)] to invert the matrix with a lower time complexity. However, making G sparse would not make $\tilde{C}_s$ sparse: the inverse of a sparse matrix only is sparse in special cases. Thus, the capacitance matrix would still correspond to a network containing capacitances between conductors that are far apart. Moreover, $\tilde{C}_s$ may not even be positive definite, thus corresponding to a non-physical situation. This is clearly not an accurate approximation of C_s.

What happens can be explained physically as follows. By zeroing entries in the influence matrix we indeed make the mutual influence between two elements zero in such a way that varying the potential of one conductor would not perturb the amount of charge present on the other conductor. This can only be accomplished if the network contains negative, non-physical, capacitances. These are the capacitances that make the C_s matrix non-sparse and non-positive.

For example, consider a (hypothetical) four-conductor circuit whose Green's function matrix and its exact inverse are as shown below.

$$G = \begin{bmatrix} 2.420 & 0.645 & 0.223 & 0.092 \\ 0.645 & 2.380 & 0.635 & 0.223 \\ 0.223 & 0.635 & 2.380 & 0.645 \\ 0.092 & 0.223 & 0.645 & 2.420 \end{bmatrix} \qquad G^{-1} = \begin{bmatrix} 0.446 & -0.118 & -0.009 & -0.004 \\ -0.118 & 0.484 & -0.116 & -0.009 \\ -0.009 & -0.116 & 0.484 & -0.118 \\ -0.004 & -0.009 & -0.118 & 0.446 \end{bmatrix}$$

If the G matrix is approximated by G_1 or G_2 as below, its inverse will contain positive off-diagonal entries that correspond to negative coupling capacitances—see the Equations (4.6) - (4.8):

$$G_1 = \begin{bmatrix} 2.420 & 0.645 & 0.223 & 0.000 \\ 0.645 & 2.380 & 0.635 & 0.223 \\ 0.223 & 0.635 & 2.380 & 0.645 \\ 0.000 & 0.223 & 0.645 & 2.420 \end{bmatrix} \qquad G_1^{-1} = \begin{bmatrix} 0.446 & -0.118 & -0.014 & 0.015 \\ -0.118 & 0.484 & -0.114 & -0.014 \\ -0.014 & -0.114 & 0.484 & -0.118 \\ 0.015 & -0.014 & -0.118 & 0.446 \end{bmatrix}$$

$$G_2 = \begin{bmatrix} 2.420 & 0.645 & 0.000 & 0.000 \\ 0.645 & 2.380 & 0.635 & 0.000 \\ 0.000 & 0.635 & 2.380 & 0.645 \\ 0.000 & 0.000 & 0.645 & 2.420 \end{bmatrix} \qquad G_2^{-1} = \begin{bmatrix} 0.448 & -0.132 & 0.038 & -0.010 \\ -0.132 & 0.494 & -0.142 & 0.038 \\ 0.038 & -0.142 & 0.494 & -0.132 \\ -0.010 & 0.038 & -0.132 & 0.448 \end{bmatrix}$$

4.6 The Generalized Schur Algorithm

We are looking for an approximation $\tilde{C}_s$ of C_s, and we saw that making G sparse does not help. Recall that we really need a method that makes $\tilde{C}_s$ sparse. Because of the structure of A, this is equivalent to searching for an approximation $\tilde{G}^{-1}$ of G^{-1} that is sparse. Such an approximation makes only the direct coupling between finite elements zero, but not the indirect coupling.

4.6.1 The Algorithm

For the case in which the support[2] of $\tilde{G}^{-1}$ has a staircase band structure, the problem is solved by the so-called generalized Schur algorithm, as presented in [Dewilde (1987)]. A band is said to be a staircase band if its support S is such that for all $i \leq k \leq l \leq j$, $(i, j) \in$ S implies $(k, l) \in$ S. See Figure 4.7 for an example of a staircase band. The algorithm only utilizes entries in those positions (i, j) of G for which there are non-zero entries in $\tilde{G}^{-1}$. Thus, only a subset of the entries of G need to be specified. For a partly specified matrix, we will refer to the set of pairs of indices of specified entries as the *specification support* of the matrix.

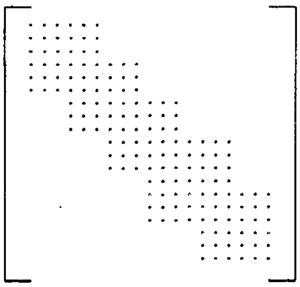


Figure 4.7. An example of a matrix that is specified on a staircase band.

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 is called G_{ME}^{-1} since it actually is the inverse of the so-called maximum entropy extension of G [Dym (1981)]. An extension of a partially specified positive definite matrix M is any Hermitian matrix that matches M on its specified entries. In [Dym (1981)], it has been shown that 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. Of all possible extensions, it is also the one with the maximum determinant.

2. The support of a matrix X is the set of pairs of indices (i, j) such that $X_{ij} \neq 0$.

The algorithm is given below, without proof, for which we refer to [Dewilde (1987)] and also to [Nelis (1988)].

Algorithm 4.1. The generalized Schur algorithm

Input: a positive definite symmetrical matrix G of size $N \times N$ that is specified on a staircase band S.

Output: the maximum entropy inverse G_{ME}^{-1} of G.

Method:

1. Let D be the diagonal matrix $[G_{11}\ G_{22}\ \cdots\ G_{NN}]$. Let G_n be the normalization $D^{-1/2}\, G\, D^{-1/2}$ of G, such that $[G_{n_{ii}}] = 1$, $i = 1 \cdots N$.

2. Let V be a strictly upper triangular matrix, such that $G_n = V + I + V^*$, and let $U = V + I$.

3. Let Θ be a $2N \times 2N$ identity matrix.

4. **[Eliminate]**
   For $(i, j) \in \{S \mid V_{ij} \neq 0\}$ in diagonal-major order **do**

   $$\begin{aligned} [U\ V] &= [U\ V]\,\theta(i, j) \\ \Theta &= \Theta\theta(i, j) \end{aligned}$$

   Here, $\theta(i, j)$ is an elementary hyperbolic rotation matrix as defined below, such that V_{ij} is eliminated. The diagonal-major order of entries to be eliminated is as follows: first the first non-zero entry of the first diagonal, then the second non-zero entry, etc. If the first diagonal of V has been eliminated, proceed with the second diagonal, and so on.

   After $m = |S|$ steps, V is zero on the staircase band S.

5. Compute the triangular factors L^{-*} and M^{-1} of the maximum entropy inverse of G_n:

   $$[L^{-*}\ M^{-1}] = [I\ I]\,\Theta$$

6. Evaluate $G_n^{-1}{}_{ME}$ and denormalize, resulting in G_{ME}^{-1}:

   $$\begin{aligned} G_{ME}^{-1} &= D^{-1/2} L^{-*} L^{-1} D^{-1/2} \\ &= D^{-1/2} M^{-1} M^{-*} D^{-1/2} \end{aligned}$$

■

In the above algorithm, $\theta(i, j)$ is a $2N \times 2N$ identity matrix, except for the following entries:

$$\begin{aligned}
\theta(i,j)_{i,i} &= (1-|\rho|^2)^{-1/2} \\
\theta(i,j)_{i,j+N} &= \rho(1-|\rho|^2)^{-1/2} \\
\theta(i,j)_{j+N,i} &= \rho^*(1-|\rho|^2)^{-1/2} \\
\theta(i,j)_{j+N,j+N} &= (1-|\rho|^2)^{-1/2}
\end{aligned}$$

where the reflection coefficient ρ ($|\rho| < 1$) is given by $\rho = -V_{ij}/U_{ii}$.

Now, we consider the case in which G is a positive definite matrix that is not specified on a staircase band, but for which there exists a permutation matrix P so that $P\,G\,P^*$ is a positive definite matrix that is specified on such a band. We will say that such matrices are *staircase permutable*. The maximum-entropy inverse of a staircase permutable matrix G can be computed as

$$G_{ME}^{-1} = P^*\,(P\,G\,P^*)_{ME}^{-1}\,P \tag{4.38}$$

When N denotes the size of the matrix and b the maximum width of the staircase band of G_{ME}^{-1}, the generalized Schur algorithm can be implemented to run in $O(Nb^2)$ time and $O(b^2)$ space [Genderen (1991)]. To achieve the low space complexity, the algorithm is implemented as a pipeline, with the matrix being read row-by-row. As soon as $O(b^2)$ rows have been read, the result matrix begins to be produced, row-by-row.

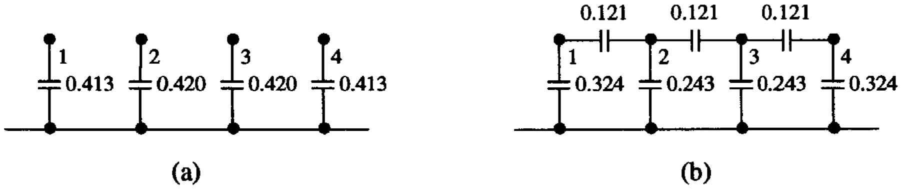
4.6.2 Approximation

To illustrate the approximation properties of the Schur algorithm, consider the 4-conductor example in Section 4.5. The influence matrix G of that problem is completely specified. Define the first-order maximum entropy approximation of G^{-1} as the inverse of the maximum entropy extension of the diagonal of G, the second-order maximum entropy approximation of G^{-1} as the inverse of the maximum entropy extension of the diagonal and the first upper and lower diagonals of G, and so on. The n-th order maximum entropy approximation of G^{-1} is denoted as $G_{(n)ME}^{-1}$. For an $N\times N$ matrix G, we have $G_{(N)ME}^{-1} = G^{-1}$.

The successive maximum entropy approximations of G^{-1} are given below, and Figure 4.8 shows the corresponding capacitance networks. It can be seen that when the approximation order is increased, the networks obtained converge to the network corresponding to the exact inverse.

$$G^{-1}_{(1)ME} = \begin{bmatrix} 0.413 & 0.000 & 0.000 & 0.000 \\ 0.000 & 0.420 & 0.000 & 0.000 \\ 0.000 & 0.000 & 0.420 & 0.000 \\ 0.000 & 0.000 & 0.000 & 0.413 \end{bmatrix} \quad G^{-1}_{(2)ME} = \begin{bmatrix} 0.445 & -0.121 & 0.000 & 0.000 \\ -0.121 & 0.485 & -0.121 & 0.000 \\ 0.000 & -0.121 & 0.485 & -0.121 \\ 0.000 & 0.000 & -0.121 & 0.445 \end{bmatrix}$$

$$G^{-1}_{(3)ME} = \begin{bmatrix} 0.446 & -0.118 & -0.010 & 0.000 \\ -0.118 & 0.484 & -0.115 & -0.010 \\ -0.010 & -0.115 & 0.484 & -0.118 \\ 0.000 & -0.010 & -0.118 & 0.446 \end{bmatrix} \quad G^{-1} = \begin{bmatrix} 0.446 & -0.118 & -0.009 & -0.004 \\ -0.118 & 0.484 & -0.116 & -0.009 \\ -0.009 & -0.116 & 0.484 & -0.118 \\ -0.004 & -0.009 & -0.118 & 0.446 \end{bmatrix}$$



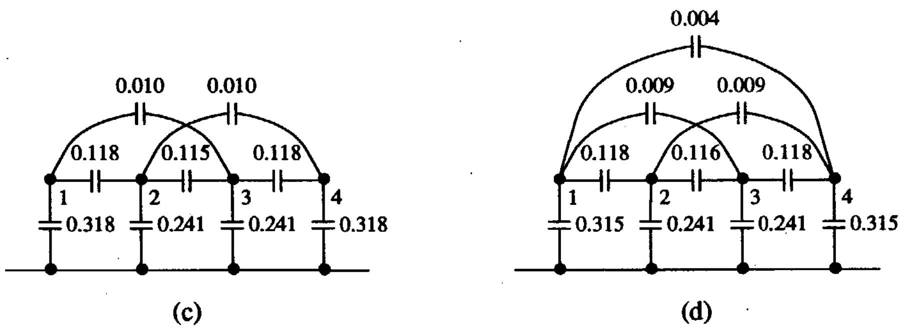


Figure 4.8. Capacitance network corresponding to the first order (a), second order (b), third order (c) and exact Schur inversion (d).

A property of all these approximations is that the total capacitance (see Section 4.2) of any node *i* (i.e. between node *i* and the reference) is exact. Here, exact means equal to the value in the network resulting from the exact inversion of the completely specified influence matrix. For example, in Figure 4.8(a) the total capacitance of node 2 is 0.420. When the series connection of 2 capacitances is denoted as $\otimes$, and has a higher precedence than +, which denotes parallel connection, the total capacitance of node 2 in Figure 4.8(b) can be computed as $0.121 \otimes 0.324 + 0.243 + 0.121 \otimes (0.243 + 0.121 \otimes$

0.324) = 0.420, clearly the same value as above.

This property directly follows from the observation that the capacitance network corresponding to any extension (i.e. not just the maximum entropy extension) of G, where G is specified on at least the main diagonal, has exact total capacitance values for any node i. This observation follows immediately from the expression for the total capacitance,

$$C_{t_{ii}} = \frac{1}{G_{ii}} \tag{4.39}$$

and the definition of the extension of a partially specified matrix.

By a similar reasoning, and using the expression

$$C_{t_{ij}} = \frac{1}{G_{ii} - 2G_{ij} + G_{jj}} \tag{4.40}$$

if the specification support of G contains the pair (i, j), the capacitance network resulting from any extension of G has exact total capacitance values between node i and j. For example, with the above expression, the total capacitance between nodes 1 and 2 is found to be 0.285. The same value results whe the capacitances in Figure 4.8(b) are used for the computation: $C_{t_{12}} = 0.121 + 0.324 \otimes (0.243 + 0.121 \otimes (0.243 + 0.121 \otimes 0.324)) = 0.285$.

These results can be extended for capacitance matrices resulting from the elimination of any number of floating nodes from a system of conductors.

4.6.3 Finite Element Numbering

For circuit extraction purposes, the support S of G_{ME}^{-1} depends on the numbering of the finite elements. Ideally, the finite elements must be numbered such that S only contains entries corresponding to pairs of finite elements that are closer to each other than some fixed distance w over which we consider coupling to be significant, and such that S does not contain entries corresponding to pairs of boundary elements that are further apart—this is the numbering that will lead to the desired, sparse capacitance model. The distance w (window size), should then be a parameter of the method trading accuracy for computer time. Thus, we seek a numbering scheme in which, for any two boundary elements numbered i and j,

$$(i, j) \in \mathrm{S} \text{ iff } dist\,(e_i,\ e_j) \leq w \tag{4.41}$$

where $dist\,(e_i,\ e_j)$ denotes the distance between elements i and j. For constant elements, this is the distance between the centers of the elements. For linear elements, this is the distance between the center nodes of the (possibly overlapping) elements.

Such a numbering is trivially possible when the elements are located in a 1-dimensional row, but is generally impossible when they are located in 2-dimensional or 3-dimensional space.

To investigate the impact of this on the efficiency of the extraction method, we consider a 2-dimensional grid and assume that there is a finite element located at each grid point. This is an idealized model of the distribution of finite elements in real layouts, in which the z-component of the finite element locations is neglected. This is acceptable because the z-component is limited in magnitude by virtue of the dielectric structure of an IC, and we will neglect it for finite element ordering purposes. We will take this grid model as a reference model for computing asymptotic complexity measures.

Let us now assume that we only wish to compute influences between finite elements that are located within a distance of 2 grid units of each other. If we choose a numbering as in Figure 4.9(a)[3], the entries of the influence matrix corresponding to nearby elements are shown in Figure 4.9(b). This is a sparse band, which is however not staircase-shaped. To make it staircase, the holes must be filled in by computing some influences between pairs of boundary elements that are far apart. The Schur algorithm can then be applied to a matrix with a structure as shown in 4.9(c).

Under this model, and with the finite element numbering as presented above, Equation (4.41) becomes for large N

$$(i, j) \in \mathrm{S} \text{ if } dist\,(e_i,\ e_j) \leq w \tag{4.42a}$$

$$(i, j) \notin \mathrm{S} \text{ if } dist\,(e_i,\ e_j) > \sqrt{w^2 + N} \tag{4.42b}$$

where some (i, j) with $w < dist\,(e_i,\ e_j) \leq \sqrt{w^2 + N}$ are in S and some are not. Because S contains influences between elements that are separated more than a distance

3. This numbering is convenient in our scanline based extractor, but it can be shown that no asymptotically better numbering exists.

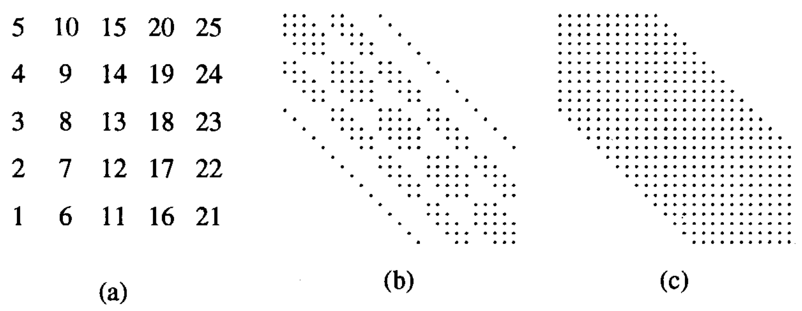


Figure 4.9. Finite element numbering scheme (a), entries of the influence matrix corresponding to pairs of nearby elements (b), and a staircase band structure showing the fill-ins (c).

w, the asymptotic complexity is sub-optimal. Specifically, the width of the staircase band can be seen to be $O(w\sqrt{N})$ and the time complexity $O(Nb^2)$ of the Schur algorithm can be written more precisely as $O(N^2w^2)$. The space complexity $O(b^2)$ then becomes $O(Nw^2)$.

4.7 Approximate Inversion of Multiple Band Matrices

A reduction in the time complexity for the matrix inversion step from $O(N^3)$ to $O(N^2w^2)$, with w a constant depending on the desired accuracy, is significant. Even so, it is not satisfactory because a quadratic time complexity is still computationally demanding, and the resulting capacitance network still contains direct capacitances between distant conductors. We conclude that the limitation to matrices with a staircase specification support is too stringent. However, there are no known efficient algorithms to obtain G_{ME}^{-1} in the general case, in which the diagonal elements are given but the specification support is arbitrary otherwise. For an exact solution of G_{ME}^{-1} in the general case, we can only use iterative algorithms. These are very inefficient—they require a complete inversion of a matrix of the same size as G in each iteration step. Therefore, iterative algorithms for maximum entropy matrix extension are not useful in practice [Nelis (1989)]. Although there are indications, see also [Nelis (1989)], it has in fact not even been shown that in the general case, the maximum entropy extension is a close approximation of the original matrix.

However, in [Nelis (1988)] an algorithm is developed that, given certain assumptions, efficiently computes a close approximation of G^{-1} for a more general class of matrices (to be defined below). As the algorithm is in fact a hierarchical extension of the Schur algorithm, we will call it the "hierarchical Schur algorithm" for short. The resulting approximation is the inverse of the maximum entropy extension of a matrix that is close to G and, like G_{ME}^{-1}, it vanishes on the complement of S.

4.7.1 The Hierarchical Schur Algorithm

In order to describe the algorithm, we introduce the following notation. Block matrices and entries of block matrices are denoted by bold uppercase letters. If we partition a matrix G as a block matrix, we denote it by **G**. Conversely, if we interpret **G** as a matrix with scalar entries, we denote it by G. For a block matrix **G**, the notation $\mathbf{G}(i,j)$ denotes the principal block matrix that lies in the rows and columns of **G** indexed by $i,\ldots,j$. The symbol $\square[\mathbf{G};(i,j)]$ denotes the block matrix such that $(\square[\mathbf{G};(i,j)])(i,j)=\mathbf{G}$ and is zero on the other parts of $\square[\mathbf{G};(i,j)]$. The size of $\square[\mathbf{G};(i,j)]$ will be defined by its context. $\mathbf{G}(.,.)$ takes a block out of a matrix and $\square[\mathbf{G};(.,.)]$ embeds a matrix in a larger matrix such that it is surrounded by zeros.

Now, let G be a positive definite matrix that is specified on a multiple band. Let it be partitioned as $\mathbf{G}=[\mathbf{G}_{ij}]$, $i,j=1,\ldots,N$, where the blocks $\mathbf{G}_{ij}$ are of size $n_i \times n_j$, such that, for some band with support $\{\ (i,j)\ |\ |i-j| \le M\ \}$, the partially specified principal sub-matrices $\mathbf{G}(j,j+M)$, $j=1,\ldots,N-M$, have a staircase support and such that the blocks outside the band are unspecified.

The sparse-inverse approximation of G, denoted by G_{SI}^{-1}, is defined as [Nelis (1988)]

$$G_{SI}^{-1} = \sum_{j=1}^{N-M} \square[\mathbf{G}(j,j+M)_{ME}^{-1};(j,j+M)] \tag{4.43a}$$

$$- \sum_{j=1}^{N-M-1} \square[\mathbf{G}(j+1,j+M)_{ME}^{-1};(j+1,j+M)] \tag{4.43b}$$

It is shown there that G_{SI}^{-1} is close to G^{-1} and that it has the desired sparsity pattern, provided that G is not close to singular. Although G_{SI}^{-1} cannot be guaranteed to be positive definite, it is conjectured in [Nelis (1988)] that it will fail to be so only if the completely specified matrix G is close to singular. In practice, the algorithm performs well; results are presented in Section 5.5.

An example of how Equation (4.43) would produce the sparse-inverse of a block matrix for $N = 3$ and $M = 1$ is given in Figure 4.10.

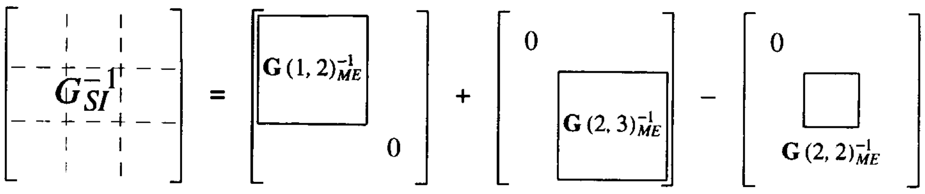


Figure 4.10. An example of Equation (4.43) for N = 3 and M = 1.

4.7.2 Finite Element Numbering

For the Schur algorithm, it is impossible to devise a finite element numbering scheme that satisfies Equation (4.41), i.e. such that the specification support S of G only contains entries corresponding to pairs of nearby finite elements. We will now show that the hierarchical Schur algorithm allows a numbering for which Equation (4.41) is almost satisfied.

Let w again be the distance over which capacitive coupling is significant. Then, partition the layout into non-overlapping vertical strips of width w. (The right-most strip possibly has a width $< w$.) Number the strips from left to right as $S_1, S_2, \ldots, S_n$, and let strip S_i correspond to block $\mathbf{G}(i,i)$ of $\mathbf{G}$, and the abutment of some consecutive strips $S_i \cdots S_j$ to the sub-matrices $\mathbf{G}(i,j)$ of $\mathbf{G}$. Viewing the consecutive strips $S_i \cdots S_j$, $j = i + M$, as one wider strip $S_{i,j}$, gives a one-to-one correspondence between the terms in Equation 4.43 and the strips in the layout. This is illustrated in Figure 4.11 for $N = 5$ and $M = 1$.

Each strip then corresponds to a matrix that can be approximately inverted with the Schur algorithm. Within each strip, a staircase support of the corresponding matrix is obtained by numbering the boundary element nodes in order of increasing y-coordinates and by neglecting the x- and z- components of the distance between the boundary elements. (In Figure 4.11(a), the positive x-, y- and z-axes are respectively to the right, up and out of the paper.) An illustration of this ordering and the resulting support of the matrix is given in Figure 4.12.

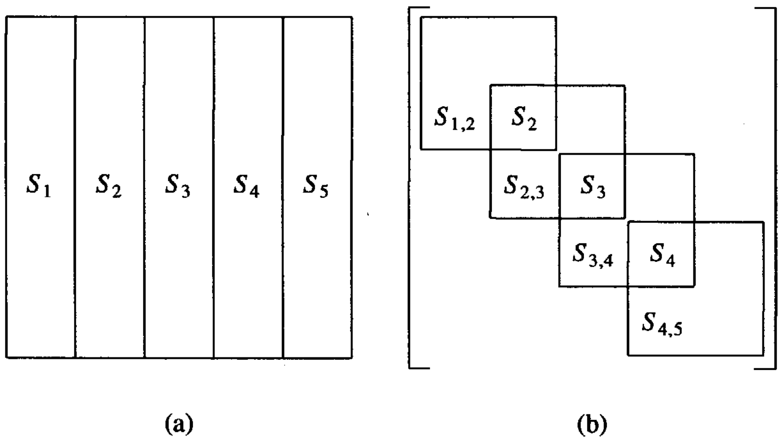


Figure 4.11. A strip partitioning of a layout (a) and the corresponding blocks of the influence matrix (b).

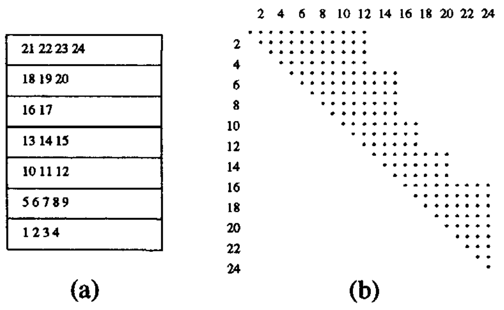


Figure 4.12. An example of a finite element ordering (a) and the resulting matrix support (b).

While this one-dimensional distance measure clearly includes all interactions between boundary elements at distances $\leq w$, it still does not exclude all interactions between elements at distances $> w$. However, it is acceptable since the elements must be close in the x-direction as well as in the z-direction, because they are all located in the same strip and because of the dielectric structure of an IC, respectively. In fact, for $M = 1$ in Equation (4.43). we have at most 2 consecutive strips with a total width of $2w$. Thus, when we neglect the distance in the z-direction, we have for the hierarchical Schur algorithm:

$$(i,\ j) \in \text{S if } dist\ (e_i,\ e_j) \leq w \tag{4.44a}$$

$$(i,\ j) \notin \text{S if } dist\ (e_i,\ e_j) > 2w \tag{4.44b}$$

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

In comparing Equations (4.42) and (4.44), it must be noted that in the latter there is no dependence on the total size of the layout. This is reflected in the time complexity of the algorithm. Assuming a square layout with the finite elements uniformly distributed, the dimension of each block $G(i, j)$ of G is proportional to the number of finite elements in a strip, or $O(w\sqrt{N})$. With the width of the staircase band being $O(w^2)$, the Schur algorithm takes $O(Nb^2) = O(w\sqrt{N}) \times O(w^4) = O(w^5\sqrt{N})$ time per block. Since there are $O(\frac{\sqrt{N}}{w})$ blocks, the total time is $O(Nw^4)$, which is linear in the size of the layout. Also, the space complexity $O(b^2)$ becomes $O(w^4)$, which is independent of the size of the layout. These figures must be considered the main result of this section.

4.8 Conclusion

This chapter has dealt with the theory and mathematics of an efficient method for accurately computing the capacitances of integrated circuit interconnections. The method is a so-called boundary element method (also an integral equation method), in which the equations for a static electric field are solved by discretizing the surfaces of the conductors. An amount of charge is assumed on each of the discrete elements and the charge on each individual element induces a potential in all other elements. The capacitances are obtained by inverting an influence matrix which expresses the charge-potential relation between elements.

In the past, the matrix inversion formed the computational bottleneck in this capacitance extraction method, limiting its applicability to small problems. However, the hierarchical Schur algorithm described in this chapter can be used to quickly approximate the inverse, yielding a reduced capacitance network that exclusively contains all significant capacitances.

The efficiency of the hierarchical Schur algorithm is summarized in Table 4.1. For contrast, the table also shows the computational cost of exact inversion as well the Schur algorithm, of which the hierarchical Schur algorithm is an extension. These results were obtained for the case of a square design with a uniform distribution of N boundary elements of identical size. Thus, N is linear in the size of the layout. The parameter w denotes the distance over which capacitive coupling is significant. It is a constant that depends only on the desired accuracy of the capacitance network extracted—not on the size of the layout.

Table 4.1. Time and space complexities

algorithm	CPU time	memory
exact inversion	N^3	N^2
Schur	N^2w^2	Nw^2
hierarchical Schur	Nw^4	w^4

We conclude that the hierarchical Schur algorithm is an utterly efficient (optimal) algorithm for inverting the influence matrix in the boundary element method for capacitance extraction.

How to integrate the boundary element method presented in this chapter into a layout-to-circuit extractor running on an IC designer's workstation is the subject of Chapter 5. For that purpose, we must solve many other issues, such as that of generating the boundary element mesh. The result will be a practical software tool for 3-dimensional IC capacitance extraction. Chapter 5 will also present data (such as measured computation times and memory use, and capacitance data), obtained from an actual implementation of the method. These data indeed confirm the efficiency and accuracy of the method.

References

Balaban (1973)	P. Balaban, "Calculation of the capacitance coefficients of planar conductors on a dielectric surface," *IEEE Trans. on Circuit Theory* **CT-20**(6) pp. 725-731 (Nov. 1973).
Benedek (1972)	P. Benedek and P. Silvester, "Capacitance of parallel rectangular plates separated by a dielectric sheet," *IEEE Trans. on Microwave Theory and Techniques* **MTT-20**(8) pp. 504-510 (Aug. 1972).
Brebbia (1989)	C.A. Brebbia and J. Dominguez, *Boundary Elements, an Introductory Course,* Computational Mechanics Publications, Southampton (1989).
Cottrell (1985)	P.E. Cottrell and E.M. Buturla, "VLSI Wiring Capacitance," *IBM J. Res. Develop.* **29**(3) pp. 277-288 (May 1985).
Dang (1981)	R.L.M. Dang and N. Shigyo, "Coupling capacitances for two-dimensional wires," *IEEE Electron Device Letters* **EDL-2**(8) pp. 196-197 (Aug. 1981).
Dewilde (1990)	P. Dewilde and Z.Q. Ning, *Models for Large Integrated Circuits,* Kluwer Academic Publishers, Dordrecht, the Netherlands (1990).
Dewilde (1987)	P.M. Dewilde and E. Deprettere, "Approximate Inversion of Positive Matrices with Applications to Modelling," *Nato ASI Series, Modelling, Robustness and Sensitivity Reduction in Control Systems,* (1987).
Dierking (1982)	W.H. Dierking and J.D. Bastian, "VLSI parasitic capacitance determination by flux tubes," *IEEE Circuits and Systems Magazine,* pp. 11-18 (Mar. 1982).
Duff (1986)	I.S. Duff, A.M. Erisman, and J.K. Reid, *Direct Methods for Sparse Matrices,* Oxford Science Publications, Oxford, UK (1986).
Dym (1981)	H. Dym and I. Gohberg, "Extensions of band matrices with band inverses," *Linear algebra and its applications,* (36)(1981).
Fletcher (1984)	C.A.J. Fletcher, *Computational Galerkin Methods,* Springer Verlag (1984).

Genderen (1989) A.J. van Genderen, "SLS: An Efficient Switch-Level Timing Simulator Using Min-Max Voltage Waveforms," *Proc. VLSI 89*, Munich, pp. 79-88 (Aug. 16-18, 1989).

Genderen (1991) A.J. van Genderen, "Reduced Models for the Behavior of VLSI Circuits," Ph.D. Dissertation, Delft University of Technology, Delft, the Netherlands (1991).

Guerrieri (1987) R. Guerrieri and A.L. Sangiovanni-Vincentelli, "Three Dimensional Capacitance Evaluation on a Connection Machine," *Proc. IEEE ICCAD-87*, Santa Clara, CA, pp. 446-449 (Nov. 9-11, 1987).

Harrington (1968) R.F. Harrington, *Field Computation by Moment Methods,* The Macmillan Company, New York (1968).

Hasegawa (1971) H. Hasegawa, M. Furukawa, and H. Yanai, "Properties of microstrip-lines on Si-SiO2 system," *IEEE Trans. on Microwave Theory and Techniques* **MTT-19** pp. 869-881 (Nov. 1971).

Janak (1989) J.F. Janak, D.D. Ling, and H.M Huang, "C3DSTAR: A 3D Wiring Capacitance Calculator," *Proc. IEEE ICCAD-89*, pp. 530-533 (1989).

McCormick (1984) S.P. McCormick, "EXCL: A circuit extractor for IC designs," *Proc. 21st Design Automation Conference*, pp. 616-623 (1984).

Nabors (1989) K. Nabors and J. White, "A Fast Multipole Algorithm for Capacitance Extraction of Complex 3-D Geometries," *Proc. IEEE 1989 Custom Integrated Circuits Conference*, pp. 21.7.1-21.7.4 (1989).

Nelis (1988) H. Nelis, E. Deprettere, and P. Dewilde, "Approximate Inversion of Positive Definite Matrices, specified on a Multiple Band," *Proc. SPIE 88*, San Diego, CA,(Aug. 1988).

Nelis (1989) H. Nelis, "Sparse Approximations of Inverse Matrices," Ph.D. Dissertation, Delft University of Technology, Delft, the Netherlands (1989).

Ning (1987) Z.Q. Ning, P.M. Dewilde, and F.L. Neerhoff, "Capacitance Coefficients for VLSI Multilevel Metallization Lines," *IEEE Trans.*

on Electron Devices **ED-34**(3) pp. 644-649 (March 1987).

Ning (1989) Z.Q. Ning, ''Accurate and Efficient Modeling of Global Circuit Behavior in VLSI Layouts,'' Ph.D. Dissertation, Delft University of Technology, Delft, the Netherlands (1989).

Patel (1971) P.D. Patel, ''Calculation of capacitance coefficients for a system of irregular finite conductors on a dielectric sheet,'' *IEEE Trans. on Microwave Theory and Techniques* **MTT-19**(11) pp. 862-868 (Nov. 1971).

Ruehli (1973) A.E. Ruehli and P.A. Brennan, ''Efficient capacitance calculations for three-dimensional multiconductor systems,'' *IEEE Trans. on Microwave Theory and Techniques* **MTT-21**(2) pp. 76-82 (Feb. 1973).

Ruehli (1975) A.E. Ruehli and P.A. Brennan, ''Capacitance models for integrated circuit metallization wires,'' *IEEE Journal of Solid-State Circuits* **SC-10**(6) pp. 530-536 (Dec. 1975).

Ruehli (1979) A.E. Ruehli, ''Survey of computer-aided electrical analysis of integrated circuit interconnections,'' *IBM J. Res. Develop.* **23**(6) pp. 626-639 (Nov. 1979).

Seidl (1988) A. Seidl, M. Svoboda, J. Oberndorfer, and W. Rosner, ''CAPCAL - A 3-D Capacitance Solver for Support of CAD Systems,'' *IEEE Trans. on CAD* **CAD-7**(5) pp. 549-556 (May 1988).

Silvester (1968) P. Silvester, ''TEM wave properties of microstrip transmission lines,'' *Proc. Inst. Elec. Eng.* **115** pp. 43-48 (Jan. 1968).

Silvester (1983) P.P. Silvester and R.L. Ferrari, *Finite Elements for Electrical Engineers,* Cambridge University Press, Cambridge, UK (1983).

Stakgold (1968) I. Stakgold, *Boundary Value Problems of Mathematical Physics, volume II,* Macmillan, New York (1968).

Weeks (1970) W.T. Weeks, ''Calculation of Coefficients of Capacitance of Multiconductor Transmission Lines in the Presence of Dielectric Interface,'' *IEEE Trans. on Microwave Theory and Techniques* **MTT-18**(1) pp. 35-43 (Jan. 1970).

Wei (1984) C. Wei, R.F. Harrington, J.R. Mautz, and T.K. Sarkar, ''Multiconductor Transmission Lines in Multilayered Dielectric Media,'' *IEEE Trans. on Microwave Theory and Techniques* **MTT-32**(4) pp. 439-450 (Apr. 1984).

Zienkiewicz (1983) O.C. Zienkiewicz and K. Morgan, *Finite Elements and Approximation,* Wiley, New York (1983).

5. 3-Dimensional Capacitance Extraction

5.1 Introduction

In this chapter, we study how the mathematical concepts and tools developed in Chapter 4 can be implemented in a practical and efficient program that puts their modeling power under the fingertips of the designer. Our objective is to develop the detailed algorithms and computational procedures necessary to build a full-fledged layout-to-circuit extractor having the the finite element based capacitance extraction method of Chapter 4 as an integrated part.

The following are the main issues to be resolved:

1. How to fully automatically generate an accurate 3-dimensional boundary element mesh from a 2-dimensional layout and a suitable description of the fabrication process, in an efficient way.
2. How to efficiently implement the arithmetic operations and algorithms to compute and invert the influence matrix and how to convert the inverse of the influence matrix into a capacitance network.
3. How to combine all these operations with the basic device, connectivity and resistance extraction steps.

By way of introduction, we begin with a conceptual description of the overall finite element based capacitance extraction algorithm. Individual steps of this algorithm are then refined in subsequent sections.

With w the distance over which coupling capacitances are considered significant, the algorithm defines a window of width $2w$ and height w that is swept over the layout in scanline order (to be defined below). During the sweep, a finite element mesh is created. Within the window, finite elements are assumed to be coupled to all other finite elements in the window. The Green's function is evaluated for all pairs of coupled finite elements and the resulting influence matrix is inverted on the fly.

Algorithm 5.1. 3-Dimensional Capacitance Extraction

Input: A layout of a VLSI (sub) circuit, in the form of a canonical edge description as described in Chapter 3.

A parameter w, which specifies the distance over which coupling capacitances are considered significant.

Output: An equivalent circuit, in a form ready to be simulated by a program such as Spice.

Method:

1. **[Scan Layout]** A strip of width $2w$ is moved from left to right over the layout in steps of size w, as a result of which neighboring strips overlap half of their width. For each strip of width $2w$ and for each overlap of strips of width w, do steps 2-6. For strips of width $2w$ these steps correspond to one term in the summation in Equation (4.43a) and for overlaps of strips of width w, these steps correspond to one term in the summation in Equation (4.43b).

   During this step, circuit extraction (e.g. transistor identification, resistance computation, connectivity resolution) takes place.

2. **[Generate Mesh]** A finite element mesh is created for the layout in the current strip. The mesh is made fine enough to accurately model the differences in height of the conductors above the substrate and to satisfy criteria regarding its coarseness.

3. **[Compute Influence Matrix]** For every pair of nearby finite elements in the current strip, their mutual influence is computed and inserted in the influence matrix. The influence matrix is left unspecified for pairs of finite elements which are not close together.

4. **[Invert Influence Matrix]** The influence matrix is inverted using the Schur algorithm. The result is a short-circuit capacitance matrix C_s, with non-zero entries only on the positions for which the Green's function has been specified.

5. **[Update Circuit]** The non-zero entries of the inverted influence matrix, i.e. the short-circuit capacitance matrix, are converted to two-terminal capacitances and inserted in the extracted circuit. In case the current strip is of width w, i.e. it is an overlap of two strips, the capacitances are made negative before being inserted in the circuit. See also the last remark made under step 1.

6. **[Clean Up]** As the strip moves, the finite elements to the left of the strip are destroyed. Thus, the finite element mesh is only present in a sliding strip of width $2w$. ■

The mesh generation, step 2, is worked out in detail in Section 5.2. The above algorithm shows steps 3-5 as sequential steps, to be executed one after the other. However, in Section 5.3 it is shown that they are actually executed together in an interleaved fashion. This interleaving results in less storage space. Subsequently, the actual software implementation of our 3-dimensional capacitance extraction method in the Space layout-to-circuit extractor is described in Section 5.4. Section 5.5 contains an experimental analysis of the accuracy and performance of the program. The results appear to be in agreement with the theoretical figures obtained in Chapter 4. Finally, a conclusion follows in Section 5.6.

5.2 Geometric IC Modeling and Mesh generation

5.2.1 Introduction

In this section, we study the problem of *mesh generation*. In other words, we are looking for methods and algorithms that enable us to obtain a finite element mesh on the surfaces of the conductors (see Chapter 4). Although user input of the actual 3-dimensional geometry is common, it is not acceptable here since the method must be integrated in a layout to circuit extractor. Thus, given a suitable description of the chip fabrication technology, the mesh must be derived from the layout automatically.

To understand and discuss the problem, it is convenient to consider two subproblems:

1. The problem of automatically deriving a 3-dimensional model of the IC structures from a 2-dimensional specification in the form a a layout description.

2. The problem of generating a finite element mesh on the surfaces of the model.

Some work related to both of these problems has already been done. For example, the problem of determining the geometry of IC structures (and other characteristics, such as doping profiles) is handled in so-called *process simulators*. These tools actually simulate the IC processing steps like implantation, deposition and heat treatment with the aim of developing and optimizing the processing modules. This is done via an analysis (also using simulation) of the performance of the resulting devices.

Although most of the process simulators only deal with the processing steps needed to fabricate the active devices, some can handle lithography, etching, deposition and other processing steps to model the geometry of interconnections. For example, 2-dimensional interconnection modeling capabilities are provided by SAMPLE [Oldham (1979), Oldham (1980)] and SIMPL [Grimm (1983), Lee (1985)]. Given a layout description of a part of an IC and a sequence of parameterized processing steps, these tools produce cross-sectional views of the IC. These cross-sections somewhat resemble those obtained by SEM photography. They can subsequently be used to determine interconnection parasitics. In fact, a postprocessor for SAMPLE which enables studying resistance and capacitance of interconnections in relation to the step coverage of crossing wires is presented in [Lee (1983)]. However, this program uses a heuristic method for capacitance determination instead of a finite element method.

In [Koppelman (1983)], a comprehensive prototype system for the simulation and analysis of IC processing steps, called OYSTER, is presented. This system is based on a general-purpose 3-dimensional geometric modeling system [Wesley (1983)]. Its input is an IC layout description and a description of the sequence of processing steps. Each step of the sequence is "applied" to a database containing 3-dimensional geometric models of the IC device structures. After the last step, a 3-dimensional geometric model of the IC structure is complete. This model may then be used to automatically determine various geometric, mechanical, thermal and electrical properties. In particular, the authors have applied it to determine interconnect capacitances. For that purpose, the system was extended to perform automatic mesh generation for a finite element based capacitance modeling system. This is shown in Figures 5.1 and 5.2, which are reproduced from [Koppelman (1983)].

The process simulators described above obtain the final geometry by means of simulation. Although accurate geometrical models can then be obtained, this method has some disadvantages when applied to the problem of capacitance extraction:

1. A detailed knowledge of the processing steps is needed, i.e. for every step, nearly all the parameters (temperature, duration, etc.) must be available.

2. The problem solved is in fact far more general than needed for capacitance extraction. For that purpose, only the final 3-dimensional geometry is needed. The physical shapes during the intermediate stages are irrelevant.

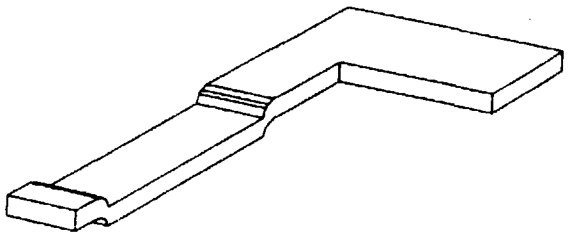


Figure 5.1. Model of a polysilicon structure created by OYSTER. (Reproduced with permission, IBM J. Res. Develop.)

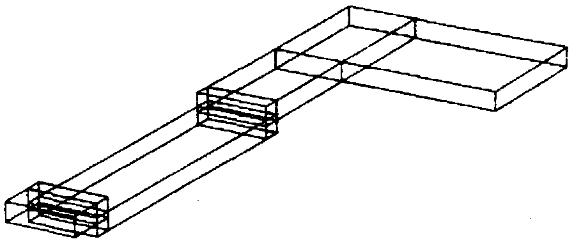


Figure 5.2. Finite element decomposition of a polysilicon structure. (Reproduced with permission, IBM J. Res. Develop.)

Therefore, we are looking for an algorithm that directly produces a geometric model of the finished IC, and avoids the intermediate results. This also benefits the efficiency of the algorithm.

The model that is produced should accurately describe the shape of the conductors. For example, in the case of non-perfect planarization, the height differences of the conductors above the substrate, which are a result from the presence or absence of other conductors, should be modeled. The model should also account for the systematic deviations of the dimensions on silicon from the designed dimensions. These deviations may result in oversizing or undersizing the physical conductors relative to the dimensions in the design database. Although this is often partly compensated before mask making, by applying appropriate grow and/or shrink operations, some deviations remain. For example, the cross-section of a conductor often has a trapezoidal shape.

5.2.2 Geometric IC Modeling

General Considerations

Obtaining a 3-dimensional geometric model of an IC is an application of the field of *geometric modeling*. Geometric modeling is concerned with efficient and robust representation of geometric information for the design and analysis of physical structures, see, e.g. [Miller (1989)]. We define *geometric IC modeling* as the geometric modeling of the 3-dimensional structures on an IC.

In a simple form of geometric modeling, called wire frame modeling, physical shapes are represented by lines and points which lie on the surfaces of the objects. Surface modeling is a more elaborate form of geometric modeling that explicitly describes the surfaces of the objects. Solid modeling, finally, also provides information on the closure and connectivity of the physical structures.

Solid modeling is the most appropriate technique to solve our capacitance extraction problem. For example, it is clear that a data structure that merely describes the surfaces of the conductors (surface modeling) will have difficulty handling such aspects as (electrical) connectedness.

Various solid modeling techniques have been developed, but two of them are most important because of the power and generality of the algorithms that have been developed for them. The first is called *constructive solid geometry* (CSG) and the second is called *boundary representation* (B-rep) geometric modeling.

In the CSG approach, objects are defined by and represented as sequences of Boolean operations and geometric transformations on simpler objects. Common operations are the union and intersection of two objects. Due to its stepwise way of construction, CSG is ideally suited to the simulation approach to geometric IC modeling. For example, the geometric modeler [Wesley (1983)] that was used in the OYSTER system is in fact a CSG system. Since the boundary of the objects is only represented implicitly, CSG is less suited to mesh generation for the boundary element method that we use.

Instead, we employ the B-rep approach. In the B-rep approach, a graph data structure is used to describe the objects by their boundary. Conceptually, a graph is mapped (drawn such that no two edges cross) onto the surfaces of the objects. The vertices of the graph then correspond to particular points on the objects and the edges of the graph define regions on the surfaces of the objects. In the B-rep approach, the bounding surfaces of the objects are represented explicitly. If the graph data structure is suitably organized, all the necessary information about the boundary of the objects can be obtained by graph traversal algorithms.

The regions on the surfaces defined and bounded by the edges and vertices of the graph are called *faces.* The boundary of a face may consist of a single, closed *loop* of edges and vertices. Such faces are said to be *simply connected.* In the general case, the boundary of a face may also consist of two or more loops, as in the case of surfaces containing holes. Such faces are said to be *multiply connected.* In our case, we use only B-reps with simply connected faces.

The graph data structure captures the *topology* of the objects. The geometry is normally added to the graph data structure via the coordinates of the vertices and/or a parametric description of the faces. Therefore, the faces should be chosen in such a way that their form can be described by the parametric function in use. In our case, as the faces will be planar polygons, the parametric function will not be needed—the faces are described implicitly by the vertex coordinates. When the boundary of a face consists of more than three vertices, there will be some redundancy in this description and—because of floating point imprecision—some inconsistency. In our case, this does not create problems, although the problem of fixed resolution solid modeling deserves specific attention, see e.g. [Hoffmann (1989)].

As an example, Figure 5.3 gives a B-rep model of an L-shaped conductor.

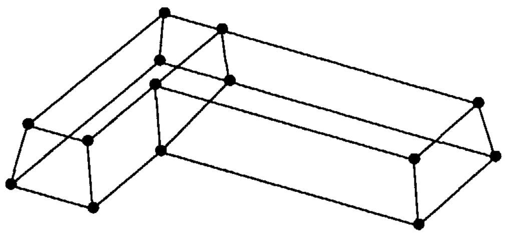


Figure 5.3. A B-rep model of an L-shaped conductor.

Algorithm

The solid modeling algorithm is implemented on top of the geometrical algorithms and data structures described in Chapter 3. Recall from that chapter that the layout is represented as a tile dissection, where a *tile* is a trapezoidal region of one *color*. The color of a tile is defined as a binary n-tuple or bit mask indicating the presence and absence of masks. Space tiles have a color bit mask with all bits cleared.

The mesh is constructed while processing all pairs of abutting tiles, using the *enumpair* operation described in Chapter 3. This is illustrated in Figure 5.4. Figure 5.4(a) shows three tiles; the shaded tiles contain an interconnect layer, say, metal. If the interconnect is present in one and only one tile of two abutting tiles, as in the case of tiles 1 and 2 or 2 and 3 in Figure 5.4(a), (a part of) a conductor side wall will be found. Then, the mesh along that part will be generated. If the conductor is present in both tiles, the mesh will only be created along the top and bottom surfaces of the conductor. The lateral position of the mesh vertices will be adjusted a proper amount to model the actual shape of the conductors more accurately.

For example, a side view of the mesh along the boundary between tiles 1 and 2 is given in Figure 5.4(b). Note how the vertices are displaced laterally. The mesh along the common boundary of tile 1 and 3 is shown in Figure 5.4(c). Thus, the mesh is explicitly created along all tile boundaries. This implicitly creates the mesh at the top and bottom sides of the conductors—a face for the top and bottom sides of each tile. The solid model then becomes a model as shown in Figure 5.3. In the mesh refinement algorithm that follows, each face of this model is individually refined into faces that are sufficiently small.

The above idea is implemented using *mesh recipes*. A mesh recipe specifies the topology of a part of the solid model, corresponding to the common boundary of two abutting tiles. Basically, there are two types of mesh topology to be specified by the mesh recipes—these correspond to the cases in Figures 5.4(b) and 5.4(c), respectively. Mesh recipes also specify part of the mesh geometry. In particular, they specify the z-position of the vertices and indicate how to adjust the positions of the vertices to model D.O.S.[1] deviations.

1. D.O.S. = Dimensions On Silicon.

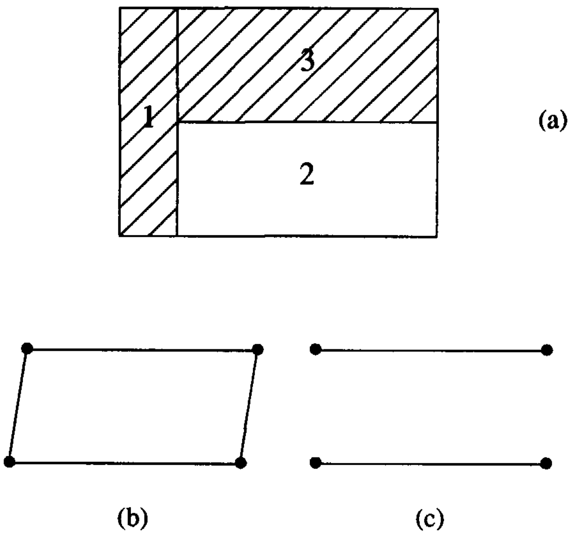


Figure 5.4. A layout (a), side view of the mesh on the boundary between tiles 1 and 2 (b) and between 1 and 3 (c).

Mesh recipes depend on the presence and absence of particular masks in both tiles, or stated otherwise, on the color transitions between two abutting tiles. Then, for a given pair of tiles, a list of vertex prototypes can be found by means of hashing with a key constructed from the color of the tiles. (The hash table itself is compiled beforehand from a suitable description of the fabrication process.) The principles described above are formalized in Algorithm 5.2.

Algorithm 5.2. Solid Model Generation

Input: A 2-dimensional layout in the form of a colored tile dissection.
Result: A 3-dimensional solid model of the resulting interconnections.

Method:
For each pair of abutting tiles, do steps 1 and 2 below. The pairs of abutting tiles are found using the algorithms given in Chapter 3 and enumerated by the *enumpair* operation defined in that chapter.

1. **[Look up mesh recipes]** Use the color of both tiles to construct a hash key, and use this key to look up a list of mesh recipes. The list contains one recipe for each of the interconnect layers present in one or both tiles. The look up table has

been compiled beforehand from a suitable description of the fabrication process.

2. **[Process recipes]** For each recipe, do the following steps *a-g*.

   a. **[Conditionally create side wall face]** If the recipe is for a conductor side wall, create a face for it. The face will model the part of the conductor side wall along the common boundary of both tiles.

   b. **[Find or create top and bottom faces]** For each of the two tiles, see if the tile contains interconnect. If so, look up the top and bottom faces. The faces can be found efficiently when maintaining appropriate connections between tiles and faces. If a tile is encountered for the first time, its top and bottom faces do not exist yet and must be created now.

   c. **[Find or create vertices]** Look up the already existing vertices, if any, required for the recipe. This can be accomplished efficiently when maintaining appropriate connections between the tiles and vertices. Create any required but not yet existing vertex.

   d. **[Conditionally find or create vertical edges]** If the recipe is for a conductor side wall, look up the already existing edges, if any, required for the recipe. Create any required but not yet existing edge. The edges are linked to the vertices as appropriate.

   e. **[Create lateral edges]** Create the two edges, along the bottom and top sides of the conductor, linking the vertices at both end points of the common boundary between the two tiles.

   f. **[Update face pointers]** For each of the two or four edges required for the recipe, determine the faces bordered by the edge and update the face pointers of the edge. The faces bordering the edges have already been found in steps a and/or b, and connecting the faces and edges appropriately is a straightforward process.

   g. **[Conditionally adjust position]** To account for the systematic deviations between layout dimensions and the dimensions-on-silicon, adjust the position of the vertices on a conductor side wall by shifting them in the direction orthogonal to the boundary between the tiles. The amount of shifting may be different for the top and bottom of a conductor to enable the modeling of non-rectangular cross-sections. ■

5.2.3 Mesh Generation

General Considerations

Although we distinguish between the 3-dimensional modeling of the structures on an IC and mesh generation, it turns out that these two problems can in fact share the same data structure. That is, the finite element mesh can be represented by a graph data structure identical to the one used for the B-rep model, although it contains more vertices and has shorter edges and smaller faces than strictly needed for the B-rep alone. Thus, the mesh generation algorithm that we have developed is the second stage (a mesh refinement stage) in a combined solid modeling and mesh generation algorithm.

The required topological structure of the mesh depends on the type of boundary element method used. For a method employing linear elements, generating a piecewise linear charge distribution, a triangular mesh is required. In such a mesh, elements are bordered by exactly three edges. There is no such requirement in the case of constant elements, generating a constant charge distribution. Any subdivision suffices. In this case, we may even employ a mixture of n-sided elements, in which n can assume any value ≥ 3.

Although a strict triangular mesh could be employed in the case of constant elements, which would be preferable for reasons of programming simplicity, it is better to use a combination of n-sided elements to improve the efficiency. Depending on the required accuracy, many of the faces in the unrefined solid model may already be small enough. If the final triangulation step is omitted for these faces, there will be fewer elements. Although the per-entry computation time of the influence matrix is increased because of the more complex shape of the elements, the total time for influence matrix computation and inversion is reduced, because the matrix will be smaller.

A regular mesh is preferred for reasons of numerical accuracy. Experiments have shown that the influence matrix can become ill-conditioned (or even non-positive) with (very) irregular meshes. Thus, areas as well as side lengths and internal angles of all elements should be comparable.

Mesh refinement requires a *measure* of the elements to be defined; mesh refinement stops when there are no elements left of which the measure exceeds a user-defined threshold. An obvious measure is the area of the element. However, this measure is not without problems since it can lead to meshes containing elements with large aspect ratios that violate the requirement of regularity.

Therefore, the stopping criterion of the mesh refinement must somehow include the aspect ratio of the elements. A convenient way to do this is by also using the *diameter* of a face, defined as the maximum of the distance between each pair of vertices of the face. Mesh refinement then continues as long as there are elements of which the area and/or the diameter exceeds the allowed maximum. Since some irregularity can be tolerated, refinement based on the diameter alone is not appropriate. A good default for the diameter is around 4 times the square root of the maximum area.

Algorithm

The solid model forms a coarse initial mesh that must still be refined according to a prescribed granularity criterion. Moreover, in the case of first-order finite elements, it must be transformed into a triangular mesh in which each face is bordered by exactly three edges. As the mesh refinement algorithm for linear elements is obviously more complex than that for constant elements, we only describe the former here. If desired, an algorithm for the latter can be obtained by a trivial simplification.

It appears to be better in general, to not let the algorithm immediately begin to triangulate a given face and then refine the triangles that are too large. For faces of the solid model having a large aspect ratio, this would result in a relatively irregular mesh. For such faces, it is better to subdivide them until their aspect ratio is around unity, and then to perform the triangulation.

A recursive algorithm can elegantly perform the initial refinement. This is illustrated in Figure 5.5(a), where the refinement steps are numbered (2) and (3). The final triangulation step is numbered (4). For contrast, Figure 5.5(b) illustrates the case of immediate triangulation, step (2), followed by a recursive refinement in which each triangle is split into 4 similar triangles. While the element areas of the final mesh in 5.5(a) and 5.5(b) are equal, the first mesh is more regular.

During the mesh refinement, extra edges must be added that split an element so that the aspect ratios of the two resulting elements are optimized. For that purpose, two edges of the original element can be split to create two new vertices that become the end points of the new edge. The locations of the new vertices can be chosen to optimize the aspect ratio of the new elements. However, new vertices must not be created very close to existing vertices, because this leads to a bad triangulation. Thus, instead of always creating two new vertices, we must use any existing vertices that are close to the intended location of the new vertices. This is illustrated in Figure 5.6. Figure 5.6(a) displays the initial element, two refinement steps and a good triangulation. Figure

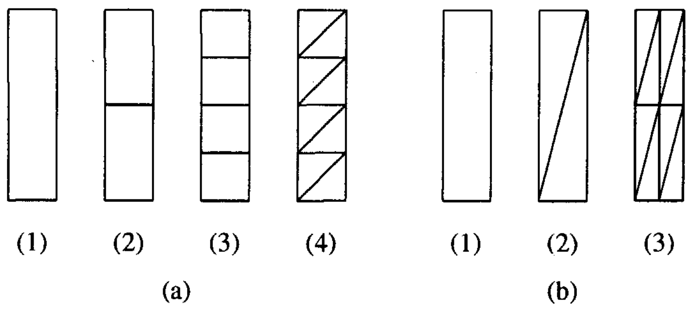


Figure 5.5. Illustration of mesh generation.

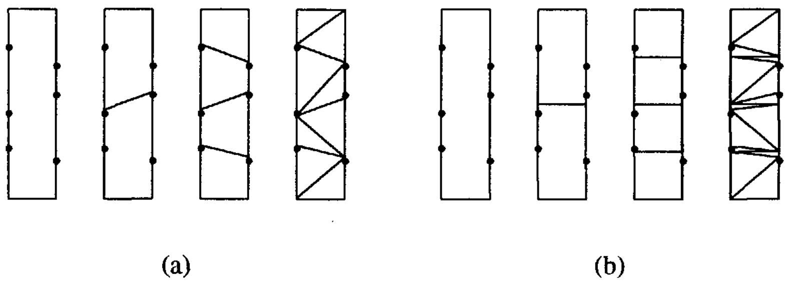


Figure 5.6. Good and bad triangulations.

5.6(b) displays non-optimal refinement steps followed by a bad triangulation.

An algorithm that implements these ideas is given as Algorithm 5.3. The result of the algorithm for the L-shaped conductor of Figure 5.4 is displayed in Figure 5.7.

Algorithm 5.3. Mesh Refinement on Tile Surfaces

Input: A solid model of a VLSI (sub) circuit, in the form of a B-rep as produced by Algorithm 5.2.

A parameter *maxarea*, specifying the maximum size of the finite element triangles.

A parameter *maxdiameter*, specifying the maximum diameter of the finite element faces before triangulation.

Output: A boundary element triangulation.

Method:

1. **[Initialize Priority Queue]** For each face *f* in the initial finite element mesh of the current strip, insert *f* in *Q*. *Q* is a priority queue data structure ordered according to the area of the faces.

2. **[Refine]**

```
while Q ≠ ∅
    f := eject (Q)
    if area (f) > 2 × maxarea or diameter (f) > maxdiameter
            $f_1, f_2$ := splitface (f)
            inject (Q, $f_1$)
            inject (Q, $f_2$)
```

   The splitface operation takes a face as input and returns two smaller faces, each of which have a smaller area and diameter. The *inject* operation inserts a face in *Q*. The *eject* operation returns and deletes the largest element from *Q*.

   The area of a face is compared with 2 times the maximum area, since a face will be split into at least two triangles, usually approximately equal in area.

3. **[Triangulate]** For each face *f* in the refined finite element mesh, triangulate *f*. In many cases, *f* is a quadrilateral face with 4 vertices and can be triangulated in a trivial way. For the general case of a face consisting of a number of vertices, a suitable algorithm is that of [Garey (1978)]. ■

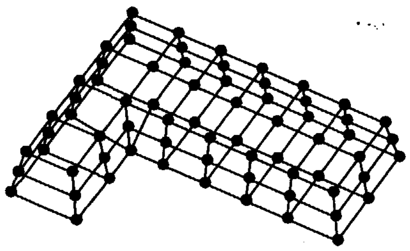


Figure 5.7. Result of Algorithm 5.3.

5.3 Finite Element Arithmetic

5.3.1 Introduction

The central equation that must be evaluated is Equation (4.36) as reproduced below:

$$C_s = A^T G^{-1} A \tag{5.1}$$

where

C_s is the so-called short-circuit capacitance matrix
A is an incidence matrix relating finite elements and conductors
G is the influence matrix.

In this equation, G^{-1} is approximated as G_{SI}^{-1} in Equation (4.43), which is also reproduced below:

$$G_{SI}^{-1} = \sum_{j=1}^{N-M} \square[\mathrm{G}(j,j+M)_{ME}^{-1};(j,j+M)] \tag{5.2a}$$

$$-\sum_{j=1}^{N-M-1} \square[\mathrm{G}(j+1,j+M)_{ME}^{-1};(j+1,j+M)] \tag{5.2b}$$

In this section we will develop efficient procedures to evaluate Equation (5.1) by using the approximation in Equation (5.2). Each of the issues involved can have a great impact on the overall efficiency:

1. Computation of the entries of G.

2. The implementation and use of the hierarchical Schur algorithm to evaluate Equation (5.2). For example, since the storage cost of the complete influence matrix—even when it is only partly specified—and its inverse is clearly prohibitive, we should look for more efficient methods.

3. The exact implementation of the pre-multiplication (post-multiplication) of G^{-1} with A^T (A). A straightforward implementation using a matrix multiplication algorithm is impractical. Instead, a graph-based algorithm is more efficient.

4. The computation of the circuit capacitances from the values of C_s.

In this section, we will focus on each of the issues in turn.

5.3.2 Computation of the Entries of G.

In the case of a Galerkin solution, the influence matrix G is defined by Equation (4.30), as reproduced below:

$$G_{ij} = \int_{S_j}\int_{S_i} G(p, q)\, f_j(p)\, f_i(q)\, dq\, dp \tag{5.3}$$

In the case of a collocation solution, the influence matrix G is defined by Equation (4.31), likewise reproduced below:

$$G_{ji} = \int_{S_i} G(p_j, q)\, f_i(q)\, dq \tag{5.4}$$

To evaluate Equation (5.4) and the inner integral over S_i in Equation (5.3), we use an (exact) analytical formula that was presented in [Wilton (1984)]. To evaluate the integral over S_j in Equation (5.3), we use 2-dimensional numerical quadrature formulas, as presented in [Stroud (1971)].

5.3.3 Matrix Construction and Inversion

The Schur algorithm can be implemented so as to operate in a pipeline fashion. That is, with the input being consumed in a certain order—in our case, this order will be row-by-row—and the output being produced in the same order, partial output is already produced before the matrix is completely known [Genderen (1991)]. With w the (maximum) width of the staircase band, the length of the pipeline and, consequently, the amount of memory needed for matrix inversion is $O(w^2)$. This is actually a $O(1)$ complexity with respect to the size of the layout.

We do not consider the implementation of the Schur algorithm here—see [Genderen (1991)] instead—but we present the interface of the software module that implements the Schur algorithm below. These operations are considered primitives in the algorithm for the evaluation of Equation (5.2) that we will develop.

schurIn (v)	Specify a new element v of the upper triangular part of the influence matrix.
schurNewRow ()	Signal the beginning of a new (possibly the first) row of the influence matrix.

schurStop ()	Signal the exhaustion of the input data.
schurOut (v)	Called by the Schur module to pass back result values in v. This operation has to be implemented by the client of the Schur module.

The Schur module works on the upper triangular part of the influence matrix only. This is appropriate since the matrices must be symmetrical.

We have already indicated that a graph-based implementation of the incidence matrix multiplications is the most efficient. In such a graph-based algorithm, the finite elements are not numbered and the elements of G and G^{-1} do not carry indices. (They can carry indices inside the Schur module, but this is not part of the external interface of the module). This implies the need for maintaining a correspondence between the entries of the influence matrix G or its inverse G^{-1} and the pair of finite elements that produced the entry in G. This can be implemented by a second pipeline (actually, a queue or FIFO data structure) operating in parallel with and synchronous to the matrix inversion pipeline. When at time t an entry of G, say, G_{ij}, is injected in the Schur pipeline, the corresponding pair of finite elements (e_i, e_j) is injected in the finite element queue. At time $t + \delta t$ the entry G_{ij}^{-1} is ejected from the Schur pipeline, and the corresponding pair of finite elements is ejected from the finite element queue. The pair of finite elements ejected from the finite element queue then correspond to the entry of G^{-1} that is ejected from the Schur pipeline. This is illustrated in Figure 5.8.

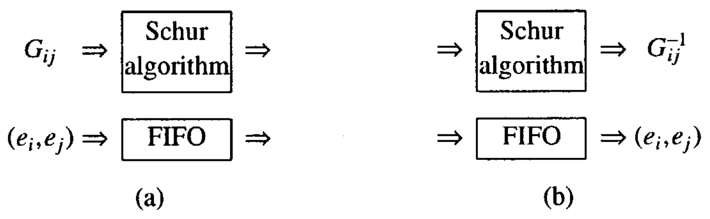


Figure 5.8. Illustration of the Schur pipeline and finite element queue operating in parallel, at time t (a) and at time $t + \delta t$ (b).

These concepts are formalized as follows. We recall from Section 4.7 and Algorithm 5.1 that the layout is partitioned into strips and that for each strip a matrix has to be inverted with the Schur algorithm. Within each strip, the finite elements are ordered according to increasing y-coordinate. The *doStrip* operation in Algorithm 5.4 corresponds to step 1.3 of Algorithm 5.1, and the operation is executed for every strip.

It takes two parameters: the head of a list of finite elements e_1 and a window size w. The list of finite elements has been sorted in order of increasing y-coordinate, and the specific traversal of the list conducted by this algorithm results in a staircase specification support as discussed in Section 4.6. This algorithm produces the upper triangular part of the the matrix only, as required by the Schur module. The operations *schurNewRow, schurIn,* and *schurStop* are implemented by the Schur module, as discussed above. The *inject* operation inserts a pair of finite elements in the queue identified by *feq*. The operation *green* (e_1, e_2) evaluates the entry in the influence matrix for the two finite elements e_1 and e_2.

```
procedure doStrip ($e_1$, w)
begin
    while $e_1 \neq$ null
        schurNewRow ()
        $e_2 := e_1$

        while $e_2 \neq$ null and $e_2.y - e_1.y <= w$
            inject (feq, $e_1$, $e_2$)
            schurIn (green ($e_1$, $e_2$))
            $e_2 := e_2$.next

        $e_1 := e_1$.next

    schurStop ()
end
```

Algorithm 5.4. Construction of the influence matrix, step 1.3 of Algorithm 5.1

5.3.4 Processing the entries of G^{-1}

The entries of G^{-1} are reported back from the Schur module by the *schurOut* operation. This operation is implemented by the client of the Schur module, as presented below. The circuit capacitances have to be updated from the entries of G^{-1}. There are two issues involved:

1. The incidence between finite elements and conductors.
2. The conversion of short-circuit capacitances to two-terminal capacitances.

In Equation (5.1), post-multiplying G^{-1} by A results in a matrix in which the columns of G^{-1} that correspond to finite elements lying on the same conductor are replaced by one column containing the vector sum of the original columns. Similarly, pre-multiplying this result by A^T gives a matrix in which the rows that correspond to finite elements lying on the same conductor are replaced by one row containing their row-vector sum. Together, this results in a matrix in which the entries of G^{-1} that correspond to the same finite elements are taken together in a summation. More precisely, when there are N finite elements and M conductors, and when the sets I_k of indices of finite elements incident to conductor k, $k = 1 \cdots M$, are defined by

$$I_k = \{ i \mid i \in [1 \cdots N] \text{ and element } i \text{ incident to conductor } k \},\ k = 1 \cdots M \tag{5.5}$$

the entries of C_s can be written as[2]

$$C_{s_{kl}} = (A^T G^{-1} A)_{kl} = \sum_{i \in I_k} \sum_{j \in I_l} G_{ij}^{-1} \tag{5.6}$$

This is illustrated in Equation (5.7) for an example with 3 finite elements and 2 conductors such that $I_1 = \{1,2\}$ and $I_2 = \{3\}$. Thus,

$$\begin{gathered}
\begin{bmatrix} 1 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}
\begin{bmatrix} G_{11}^{-1} & G_{12}^{-1} & G_{13}^{-1} \\ G_{21}^{-1} & G_{22}^{-1} & G_{23}^{-1} \\ G_{31}^{-1} & G_{32}^{-1} & G_{33}^{-1} \end{bmatrix}
\begin{bmatrix} 1 & 0 \\ 1 & 0 \\ 0 & 1 \end{bmatrix} = \\
\begin{bmatrix} 1 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}
\begin{bmatrix} G_{11}^{-1}+G_{12}^{-1} & G_{13}^{-1} \\ G_{21}^{-1}+G_{22}^{-1} & G_{23}^{-1} \\ G_{31}^{-1}+G_{32}^{-1} & G_{33}^{-1} \end{bmatrix}
= \begin{bmatrix} G_{11}^{-1}+G_{12}^{-1}+G_{21}^{-1}+G_{22}^{-1} & G_{13}^{-1}+G_{23}^{-1} \\ G_{31}^{-1}+G_{32}^{-1} & G_{33}^{-1} \end{bmatrix}
\end{gathered} \tag{5.7}$$

This means that, after the initialization of all C_{skl} to 0, the multiplications with A^T and A in Equation (5.1) can be implemented as in Algorithm 5.5. This algorithm is a first, incomplete, version of the *schurOut* operation toward which we are aiming.

2. In this and subsequent equations, the notation G_{ij}^{-1} must be read as $(G^{-1})_{ij}$ and not as $(G_{ij})^{-1}$.

```
**for** all $G_{ij}^{-1}$
    $k = x \mid i \in I_x$
    $l = x \mid j \in I_x$
    $C_{skl} = C_{skl} + G_{ij}^{-1}$
```

Algorithm 5.5. The schurOut operation, version 1.

This algorithm does not account for the fact that the *schurOut* operation only produces the upper triangular part of G^{-1}. Besides a modification for this, we need another one to convert the short-circuit capacitances into two-terminal capacitances. These modifications can be combined, and the new algorithm is shown in Algorithm 5.6. In this algorithm, C_{kk} is the ground capacitance of conductor k, and C_{kl}, $k \neq l$, is the coupling capacitance between conductors k and l.

```
**for** all $G_{ij}^{-1}$, $j \geq i$
    $k = x \mid i \in I_x$
    $l = x \mid j \in I_x$
    **if** $i = j$                        # *diagonal entry*
        $C_{kk} = C_{kk} + G_{ij}^{-1}$
    **else if** $k = l$                   # *same conductor*
        $C_{kk} = C_{kk} + 2 \times G_{ij}^{-1}$
    **else**                              # *coupling capacitance*
        $C_{kk} = C_{kk} + G_{ij}^{-1}$
        $C_{ll} = C_{ll} + G_{ij}^{-1}$
        $C_{kl} = C_{kl} - G_{ij}^{-1}$
```

Algorithm 5.6. The schurOut operation, version 2.

One issue remains, namely, when the current strip is an overlap of 2 strips and is of width w, the capacitances have to be subtracted from the capacitances in the circuit, since they correspond to the second summation in Equation (5.2). This can be implemented by first negating the value of G_{ij}^{-1}.

The first two lines in the body of Algorithm 5.6 implement the incidence relationship of finite elements and conductors. In the actual program, this incidence relationship is implemented with pointers from the finite elements to the conductors, together with the queue of finite element pairs. Taking this and the negation of G_{ij}^{-1} as discussed above into account, Algorithm 5.6 can finally be rewritten as Algorithm 5.7.

In this algorithm, the *eject* operation removes the tail of the finite element queue and returns it. The variable *OverlapStrip* serves as a flag for negating or not negating G_{ij}^{-1}, and is controlled outside Algorithm 5.7. The *addCapacitance* operation is the interface with the network module of the program. This operation accumulates the capacitance values between distinct nodes.

```
procedure schurOut (val)
begin
    $e_1, e_2$ := eject (feq)

    if OverlapStrip
        val = −val

    if $e_1 = e_2$                          # diagonal entry in matrix
        addCapacitance ($e_1$.node, gndNode, val)

    else if $e_1$.node = $e_2$.node          # finite elements are on same conductor
        addCapacitance ($e_1$.node, gndNode, $2 \times$ val)

    else                                    # finite elements are on different conductors
        addCapacitance ($e_1$.node, gndNode, val)
        addCapacitance ($e_2$.node, gndNode, val)
        addCapacitance ($e_1$.node, $e_2$.node, −val)
end
```

Algorithm 5.7. The schurOut operation, final version, step 1.5 of Algorithm 5.1.

5.4 The Space Layout-to-Circuit Extractor

In Section 3.7, we have described a layout-to-circuit extractor called *Space*. Space is an efficient program that performs, in one scanline pass over the layout, all basic extraction tasks such as device and connectivity extraction, interconnect resistance calculation and calculation of interconnect capacitance based on area/perimeter calculations.

However, the 3-dimensional capacitance model that was developed in Chapter 4 and the present chapter, has also been implemented in Space as an integrated part running together with all other extraction operations. All the steps of the 3-dimensional capacitance computation method (such as boundary element mesh generation and matrix operations) are executed while scanning the layout. The result is an efficient program that performs one single pass over the input data in such a way that complete intermediate results (e.g. the complete finite element mesh or the complete influence matrix) never exist, neither in the memory of the computer nor in temporary files on the disk. Since the basic extraction operations take place as usual, the input is a layout description and the output is a circuit description, ready to be simulated by, for example, SPICE.

With respect to the 3-dimensional capacitance model, the implemented version of Space includes the following characteristics:

Conductors in first dielectric layer. All conductors must be in the first dielectric layer (region 1 of Figure 4.3). The thickness of this layer as well as its relative dielectric constant can assume any value >0.

Also infinite or semi-infinite dielectric layers. In the evaluation of the Green's function, all terms but the first can be omitted, in which case the results are for a uniform dielectric medium of infinite dimensions (vacuum when $\varepsilon_r = 1$). All terms but the first two can be omitted, in which case the results are for a uniform dielectric half space.

Planarization. Perfect planarization of the ground plane, dielectric interface and conductors is assumed. This is becoming more true with modern IC technologies [Small (1990)].

Orthogonal layout. The layout must be orthogonal with sides parallel to the x- and y-axes.

Trapezoidal conductor cross-section. To more accurately model their cross-sectional shapes, conductors are modeled as trapezoids. More complex shapes (e.g. shapes with rounded edges [Zemanian (1989)]) are not implemented.

Diffused conductors. Diffused conductors are handled as described in Appendix 5.1.

Different element types. The program can employ either constant (zeroth order) or linear (first order) elements.

Galerkin and collocation methods. Space implements both the Galerkin and the collocation method.

Adjustable parameters. The granularity of the finite element mesh and the size of the influence window can be adjusted. With a very large window, the program does a complete and exact computation and inversion of the influence matrix. The sizes of the influence window in the x-direction and the y-direction are independent. With a very large value in the x-direction (the direction of scanline movement), the program effectively implements the non-hierarchical Schur algorithm for approximate matrix inversion.

Parameter file. All important parameters such as layer thickness, dielectric constants and D.O.S. corrections can conveniently be specified in a parameter file that is read at program start-up.

Resistance extraction. The 3-dimensional capacitance extraction works together with resistance extraction using a method explained in Appendix 5.2.

Computational optimizations. The program can optionally perform various heuristic computational optimizations and short-cuts for improving the computation speed. An example is mixed Galerkin (for nearby interactions) and collocation (for remote interactions) or even point-point computations.

Although the computation time increases linearly with the size of the layout, there remain numerous ways to further improve the speed. Some of these are discussed in Appendix 5.3.

5.5 Numerical Results

5.5.1 Introduction

In this section, we present some experimental results that were obtained using Space. The main issues involved are:

1. Comparison and assessment of the relative merits (accuracy and efficiency) of each the four methods implemented (zeroth-order collocation, zeroth-order Galerkin, first-order collocation, first-order Galerkin). Our objective is to identify the method that requires the least amount of computation time to obtain sufficient accuracy.

2. The effects of program parameters (such as the size of the window and granularity of the finite element mesh) on the accuracy and efficiency.

3. Evaluation of the overall accuracy and efficiency.

4. Comparison with other results.

This analysis confirms the usefulness and practicality of the method.

In the rest of this section, we often use a layout consisting of two crossing buses as a benchmark. This is the same benchmark as the one used in [Nabors (1991)] and is easily parameterized in the number of conductors. The thickness and width of the conductors as well as the vertical separation between both buses is in all cases 1μ. In [Nabors (1991)], the dimensions were all given in meters and the conductors were embedded in a uniform dielectric, with a dielectric constant of 1 (vacuum). To be more specific to the case of VLSI, we use microns. To compare with the results of [Nabors (1991)], we scale their capacitance values by a factor of 10^{-6}. In addition, we use a ground plane in most cases. In these cases, the height of the conductors above the ground plane is also 1μ and the dielectric constant of the medium is 3.9 (SiO_2) instead of 1.0.

As an example, a 2×5 conductor crossing bus is shown in Figure 5.9. This figure also illustrates the numbering scheme used. When a ground plane is present, conductors 1-5 are in the top layer and conductors 6-10 are in the bottom layer closest to the ground plane.

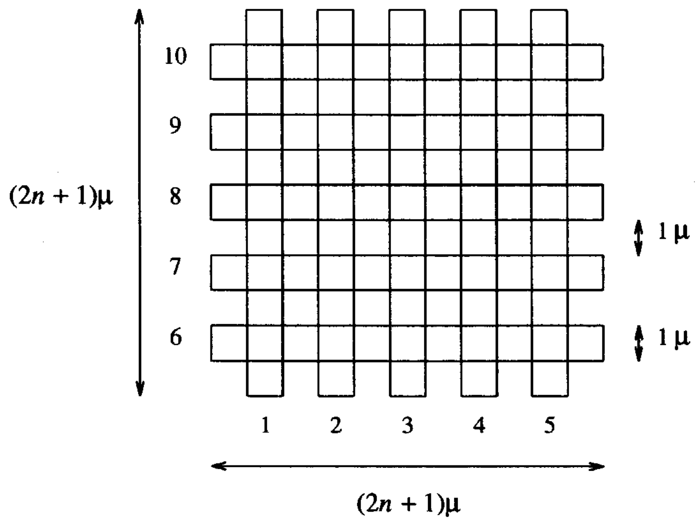


Figure 5.9. Layout of a 2×5 conductor crossing bus.

As a measure of complexity, in most cases we use the number of finite elements in the mesh that is created for it, instead of the number of conductors. An example of such a mesh is shown in Figure 5.10.

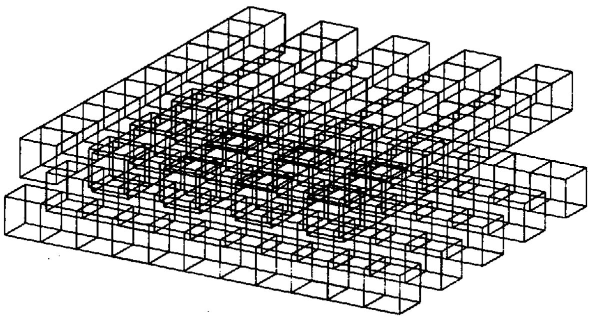


Figure 5.10. Finite element mesh for a 2×5 conductor crossing bus.

In this section, we use the following notation:

0C	Zeroth-order collocation method.
1C	First-order collocation method.
0G	Zeroth-order Galerkin method.
1G	First-order Galerkin method.
n	Number of conductors.
N	Number of finite element nodes.
w	Size of the window for the hierarchical Schur algorithm as defined in Algorithm 5.1, in μm.
s	Maximum area of finite elements, in μm^2.

Unless otherwise noted:

- These experiments were carried out on a HP 9000/720 computer with a 32 Mbyte main memory.
- CPU times are total values, including the reading of input and technology data, mesh generation, output of results, etc.
- Memory figures are net amounts, excluding memory allocation overhead. Typically, the memory is allocated in big chunks. Excluding the overhead gives smoother curves.

5.5.2 Accuracy vs. Element Size

This section discusses the accuracy and convergence properties of Space in relation to the granularity of the finite element mesh. The following remarks must be made first:

1. The accuracy of Space should preferably be checked by comparing with known results. The literature provides many results for many different geometries. Although comparatively few of them are obtained or confirmed by measurements, we can consider (nearly) identical results obtained via two or more different methods as sufficiently reliable reference data.

   However, out of the many results available in the literature, we must however select those for configurations that match the capabilities of our prototype extractor, as listed in the previous section. On the other hand, they must not be trivial either. For the purposes of this section, we have selected two useful comparisons.

2. For the intended application area, we are usually satisfied with accuracies in the range of 5-10%. Therefore, in most cases, we can suffice with relatively coarse meshes. However, since the purpose of this section is to confirm the consistency of the finite element method used (and the correctness of the implementation), we also use very small finite elements that result in large numbers of nodes and high accuracies. Moreover, the results in this section use fully populated matrices and exact matrix inversions, as well as high-order numerical integration formulas.

Cube in Vacuum
A frequently used benchmark is that of a perfectly conducting cube in vacuum, whose dimensions are $1\text{m} \times 1\text{m} \times 1\text{m}$. For example, in [Ruehli (1973)] this benchmark is also used to compare different solution methods in order to demonstrate, for example, the accuracy of the Galerkin method when compared to the collocation method. Our results are consistent with those of [Ruehli (1973)], as confirmed in Figure 5.11. This figure gives the value of the capacitance found by each of the 4 methods that have been implemented, in relation to the number of nodes (the size of the matrix to be inverted).

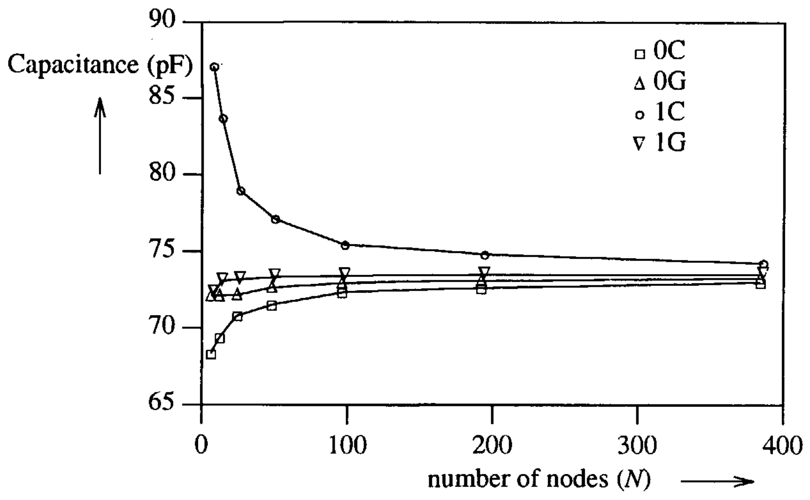


Figure 5.11. Computed capacitance results for a cube in vacuum for 4 different methods, in relation to the number of finite elements.

While the exact value of the capacitance is not known, useful upper and lower bounds are given in [Ruehli (1973)]:

$$73.3 \text{ pF} < C < 74.3 \text{ pF}$$

These bounds can be used to define a maximum approximation error as follows:

$$error = max \left[\left| \frac{C_{approx} - 73.3}{73.3} \right| , \left| \frac{C_{approx} - 74.3}{74.3} \right| \right] \times 100\% \tag{5.8}$$

This error is displayed in Figure 5.12, which displays the relative accuracy of each method more clearly than Figure 5.11.

We make the following remarks:

1. As expected, the collocation method is less accurate than the Galerkin method and a zeroth-order method is less accurate than the corresponding first-order method.

2. With a coarse mesh, the relative accuracy of first-order collocation compared to both zeroth-order solutions is surprisingly low. A possible explanation for this is that when the collocation method is viewed as an approximation of the outer 2

integrations in the Galerkin method, the first-order shape functions apparently require more accurate integration than the zeroth-order shape functions do.

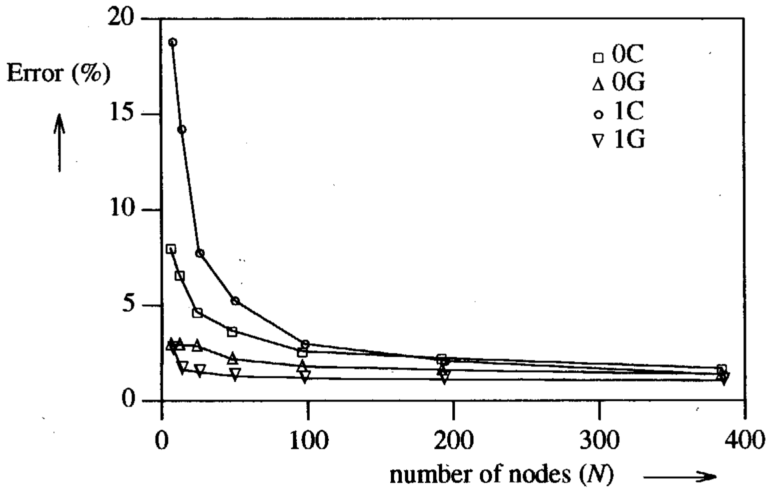


Figure 5.12. The relative error, as defined in Equation (5.8), for each of the 4 methods.

Two Crossing Lines

Figure 5.13 shows two crossing lines. With l denoting the conductor length, t_i the thickness of conductor i, and h_i the height of conductor i above the substrate, the geometry is as follows: $l = 25\mu$, $t_1 = 0.5\mu$, $t_2 = 1\mu$, $h_1 = 0.8\mu$, $h_2 = 2\mu$.

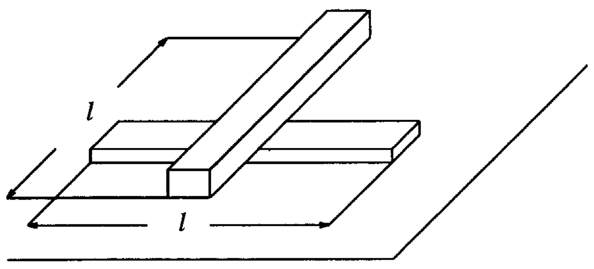


Figure 5.13. Two crossing lines, with the dimensions as in given in the text.

Table 5.1 presents the capacitances found for this structure in [Ruehli(1975)] (also using a boundary finite element method) and by Space, using 4 different methods. Except for the bad performance of the first-order collocation method, as was already observed in the case of the cube in vacuum, these results appear to be in agreement with each other. In fact, the differences lie clearly within the variations resulting from the inaccuracies of the fabrication process and certainly are small enough to be used to analyze parasitic capacitances in VLSI circuits.

Table 5.1. Comparison of Ruehli/Brennan and Space. (Capacitances in fF.)

		Space ($w = \infty$)			
	R/B	0C, s=5	0G, s=5	1C, s=4	1G, s=4
C_{12}	2.25	2.22	2.29	2.77	2.29
$C_{1\,gnd}$	8.38	8.25	8.38	9.55	8.35
$C_{2\,gnd}$	4.30	4.22	4.29	4.61	4.31

5.5.3 Computation Time

It is expected that linear elements will yield a better accuracy than constant elements and that a Galerkin will method yield a better accuracy than a point collocation method. However, the accuracy is improved at the cost of computation time. It is essential to minimize the computation time needed to obtain a sufficient accuracy.

It is very hard to accurately predict, for each method, the computational resources necessary to obtain a certain accuracy. Therefore, the methods must be compared on the basis of computation time measurements. For these measurements, we have selected the 2×2 crossing bus example, with a ground plane. Compared to, say, the cube in vacuum, the crossing bus is more typical of the intended application area.

The two main components of the total CPU time are the time needed for matrix computation and that needed for matrix inversion. Their relative magnitudes depend on the method used and on the size of the window, since they have an asymptotic complexity of $O(Nw^2)$ and $O(Nw^4)$, respectively (see Chapter 4). Thus, in order to make a fair comparison of the methods, we must use a reasonable window size. For the analysis in this section, we use a window size of $w = 5\mu$. For the present case of a 2×2 bus, this is in fact equivalent to using an infinite window, since the dimensions of the layout itself are $5\mu \times 5\mu$.

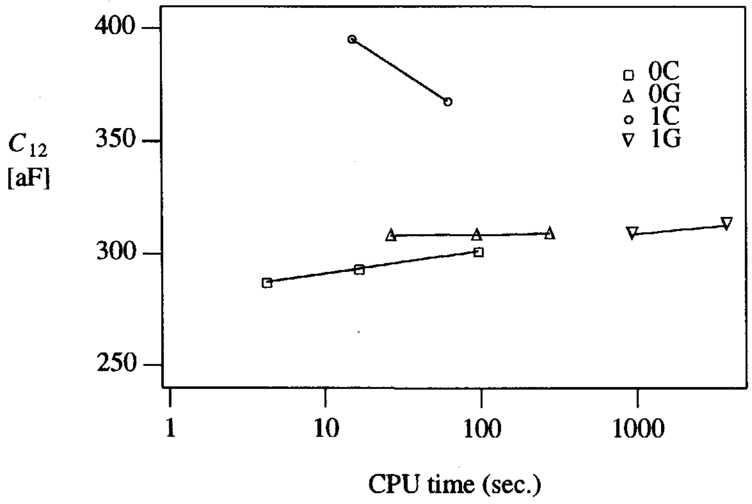


Figure 5.14. C_{12} as a function of the CPU time.

Figure 5.14 shows the capacitance between conductors 1 and 2, the coupling between the two upper conductors. While these results confirm the convergence properties found for the case of the cube in vacuum (cf. Figures 5.11 and 5.12), they actually show that the improved accuracies of the zeroth-order and first-order Galerkin methods are achieved at the cost of long computation times.

That the extra CPU time is spent in the time for evaluating the Green's functions is confirmed in Figure 5.15. This figure shows the CPU time as a function of the number of elements, N, for each of the 4 methods, together with the time needed for matrix inversion.

Although this section presents only the results for the coupling between conductors 1 and 2 in the 2×2 crossing bus example, further experiments have revealed that the indicated tendency is valid in general. Therefore, we can draw the following conclusions:

1. The zeroth-order collocation method is the fastest method and can be used for a quick, rough analysis.
2. The bad accuracy of the first-order collocation method is not compensated for by decreased CPU times, and this method does not appear to be very useful.

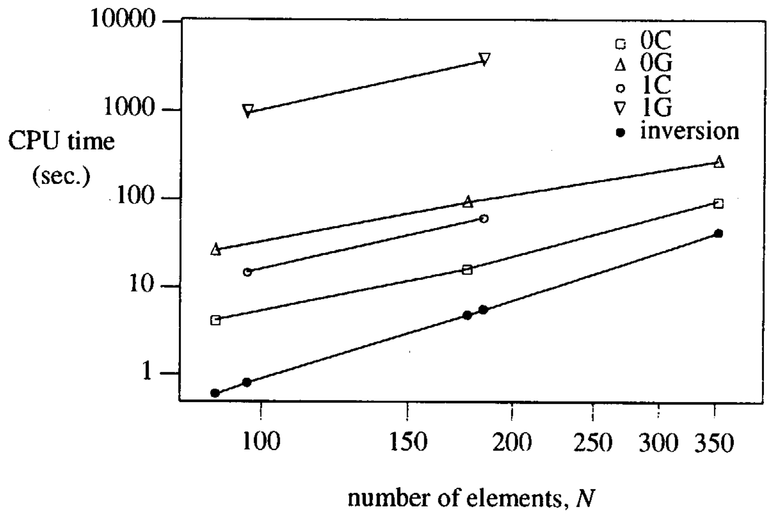


Figure 5.15. CPU time as a function of the number of elements, log-log scale.

3. The zeroth-order Galerkin method is accurate and does not take too much CPU time. It will often be the preferred method.

4. The accuracy of the first-order Galerkin method is improved at a high cost in CPU time, and its use can only be justified when absolutely the highest accuracy is necessary.

5.5.4 Effects of Window Size

To investigate the effects of the size of the window on the efficiency (the run time and memory use) and the accuracy of the result, we consider the 2×5 crossing bus benchmark with a ground plane as defined in Section 5.5.1. For this benchmark, we extract the capacitance using the zeroth-order Galerkin method and a variable window size. The finite element mesh that is created is the same in all cases. It consists of 460 equally sized elements ($1\mu \times 1\mu$ large) located on the top, bottom and sides of the conductors as shown in Figure 5.10.

Table 5.2 shows some of the capacitances for this structure. Because of symmetry, $C_{1\,10} = C_{1\,6}$, $C_{1\,9} = C_{1\,7}$ and $C_{1\,8} \approx C_{1\,7}$. The case "window = 11μ" corresponds to an exact inversion of the influence matrix, and the capacitance values depend on the accuracy of the finite element method (determined by, for example, the granularity of the finite element mesh) only. Note that smaller window sizes result in considerably lower memory use and CPU times, and that a window size of 3μ already gives reasonably accurate approximations lying within the variations resulting from fabrication tolerances.

Table 5.2. 2 × 5 crossing bus benchmark, s = 1, method = 0G, with a ground plane.

w	time	mem	capacitances (10^{-18} F)							
(μ)	(sec)	(Mbyte)	C_{1gnd}	C_{12}	C_{13}	C_{14}	C_{15}	C_{16}	C_{17}	C_{s1}
11.0	308.4	20.71	458.4	638.1	43.1	18.5	12.9	157.8	141.0	1909.2
10.0	306.9	20.72	458.5	638.1	43.1	18.5	12.9	157.8	141.0	1909.2
9.0	299.3	20.72	458.8	638.0	43.1	18.4	12.8	157.8	141.0	1909.4
8.0	284.4	16.72	459.6	637.9	43.0	18.3	12.7	157.7	141.0	1909.7
7.0	262.2	10.72	460.9	637.7	42.8	18.2	12.6	157.6	141.0	1910.3
6.0	234.5	10.72	463.3	637.3	42.5	18.0	12.4	157.4	141.2	1911.3
5.0	202.9	6.46	467.6	636.6	42.1	17.8	12.4	157.0	141.4	1913.2
4.0	139.9	3.58	481.2	635.8	41.9	22.2		156.6	140.9	1916.7
3.0	91.7	1.97	502.2	633.7	49.2			156.1	140.6	1919.6
2.0	53.1	0.97	565.2	655.9				155.6	140.3	1953.1
1.0	21.3	0.65	581.1	527.9				151.0	137.8	1824.9

Note that when the window size is decreased, the extracted coupling among conductors that are far apart becomes zero. Also note that the short-circuit capacitance C_s—which is the sum of a conductor's ground and coupling capacitances, giving the total load formed by a conductor for its driver assuming that other conductors have a low impedance, and therefore largely determining the timing of the circuit—is already accurate for very small window sizes. In fact, if the window size decreases, the "nearby" capacitances increase slightly (especially the capacitances to the ground plane), thereby compensating for not computing the "far" capacitances.

These data show that a reduction of the window size increases the efficiency. They also confirm an optimal trade-off between the level of detail of the model obtained—which is in a sense a more appropriate denotation than the "accuracy" of the model—and the requisite computer resources.

5.5.5 Performance vs. Layout Size

To investigate the effects of the size of the layout on the extraction performance (the CPU time and memory use), we again consider the parameterized crossing bus benchmark. To compare our results with the results obtained by FastCap [Nabors (1991)], we do not use a ground plane in this case. Some capacitance values obtained by Space and FastCap for the 2×4 bus are compared in Table 5.3. This table shows results for both the zeroth and first-order Galerkin methods, the latter mainly as a reference for estimating the accuracy of the 0G and FastCap results. However, for Space to obtain an accuracy comparable to that of FastCap, it is sufficient to use the zeroth-order Galerkin method, a 3μ or a 5μ window and 1μ × 1μ large finite elements.

Table 5.3. Comparison of results for 2×4 crossing bus, without a ground plane.

method	C_{1gnd}	C_{12}	C_{13}	C_{14}	C_{15}	C_{16}	C_{17}	C_{18}	C_{s1}
FastCap (direct)	70.63	137.0	12.04	7.910	48.42	40.09	40.09	48.42	404.6
FastCap (MGCR, l=2)	70.50	137.8	11.91	8.079	48.36	40.09	40.01	48.45	405.2
Space (0G, w=3, s=1)	79.63	137.4	15.11	0.000	46.82	40.08	40.10	46.99	406.2
Space (0G, w=5, s=1)	72.01	137.6	11.24	7.668	47.91	40.34	40.34	48.07	405.2
Space (0G, w=∞, s=1)	70.71	137.9	11.41	7.859	48.31	39.95	39.95	48.46	404.5
Space (1G, w=∞, s=0.50)	70.54	135.3	12.82	7.839	47.76	39.50	39.49	47.77	401.0
Space (1G, w=∞, s=0.25)	70.70	137.0	12.20	7.919	48.46	40.08	40.08	48.46	404.9

Figure 5.16 shows the amount of CPU time needed in relation to the problem size, for Space with 3 different window sizes and for FastCap. FastCap results were obtained on an IBM R6000 computer[3]. To make this comparison, we cannot use the number of finite elements (N) as the common variable, since the multipole method employed by FastCap apparently needs more elements than Space does to obtain comparable accuracy. Therefore, we use the number of conductors as the common variable. However, the number of finite elements is a better measure of problem size than the number of conductors. Thus, we use N as the scale of the x-axis. For convenience, however, the number of conductors is indicated at the top of the graph.

3. Depending on the exact model, an IBM R6000 achieves a performance that is approximately between 0.5 and 1.0 times the performance of an HP 9000/720 computer, which we have used. In comparing the CPU times, we have ignored this performance difference.

The results confirm the efficiency of Space. One can see that the CPU time increases linearly with the size of the layout when the size of the window is kept constant. We should not conclude from this figure, that Space using an exact matrix inversion (w=∞) is faster than FastCap: the time complexity of exact matrix inversion is $O(N^3)$, while the time complexity of FastCap is $O(nN)$. We can, however, conclude that the associated constant in the asymptotic time-complexity of FastCap is rather high.

Figure 5.17 illustrates the memory use of Space in relation to the problem size. (FastCap memory data are not known to us.) As expected, with exact matrix inversion ($w = \infty$), the amount of memory needed is quadratic in the problem size. With a bounded window, the amount of memory needed for matrix inversion is also bounded. Only a small, non-constant but sub-linear, term for the rest of the program remains. The total amount of memory needed is thus practically constant.

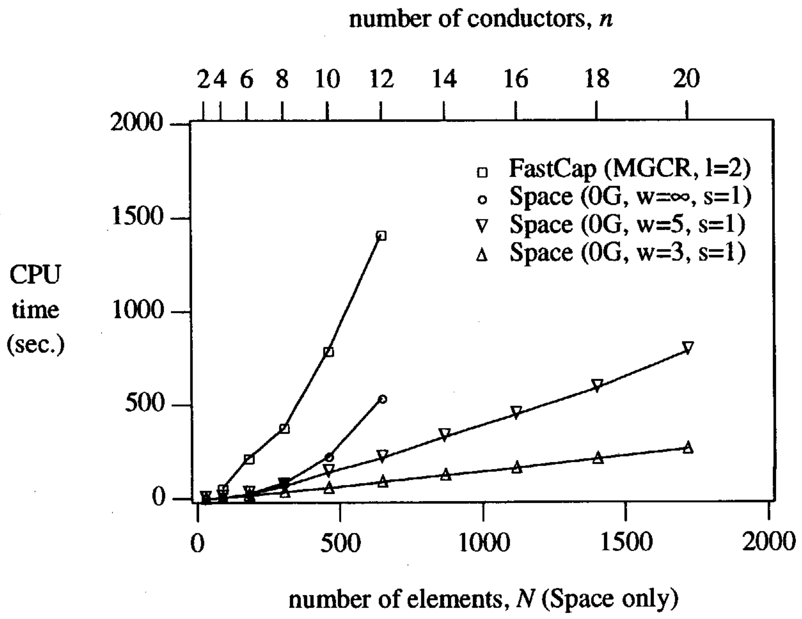


Figure 5.16. CPU time versus problem size.

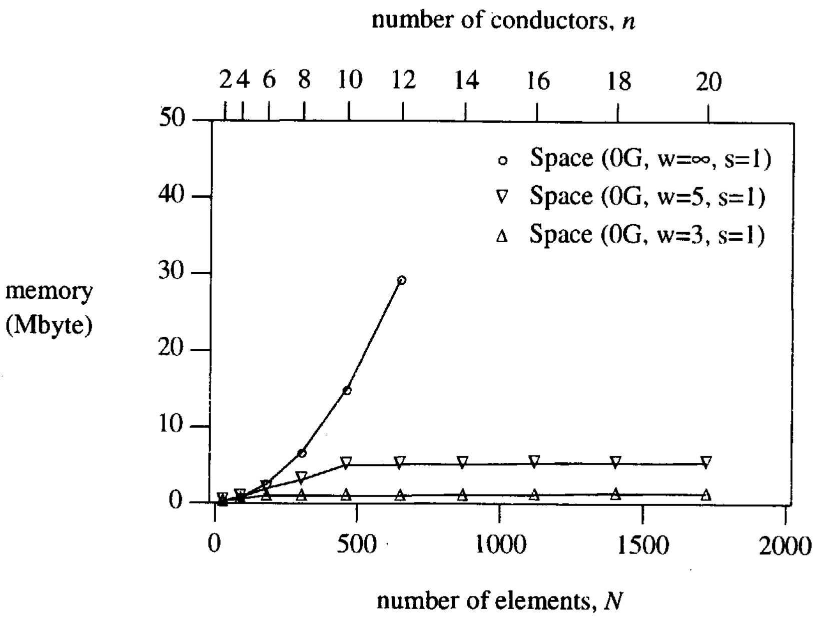


Figure 5.17. Memory use versus problem size.

From these results, we may conclude that Space is an efficient program. The low memory needs of the hierarchical Schur algorithm combined with the method's linearity, make comprehensive finite element based capacitance extraction feasible on minicomputers and even on workstations.

5.5.6 Ram Cell Example

As an example of a practical application of the program, we now present the results of an extraction of a 6 transistor SRAM cell in a double metal CMOS technology. The technology assumed is a hypothetical but realistic 1μ CMOS process ($\lambda = 0.5\mu$), with a perfect planarization and the following layer thicknesses, which are are the same as assumed in Section 2.4:

gate oxide	250 Å
inter wire oxides	0.75 μ
poly	0.5 μ
metal 1	0.75 μ
metal 2	1.0 μ

The finite element model employed is based on metallic conductors embedded in a stratified dielectric above a ground plane. Diffused conductors, which appear in real layouts, do not fit in this model. However, Appendix 5.1 describes a heuristic method so that diffused conductors can be handled anyway.

The layout is shown in Figure 5.18 and the finite element mesh that was created, from the layout and a technology description implementing the values from the above table, is shown in Figure 5.19.

The results, using the zeroth-order collocation method, are summarized in Figure 5.20. In this figure, the height of the boxes is proportional to the capacitances. The boxes labeled "gate" denote intrinsic device capacitances, those labeled "ground" denote capacitances to the substrate and those labeled "coupling" denote inter-wire capacitances. Below each bar, the name (or number) of the node and its short-circuit capacitance are given.

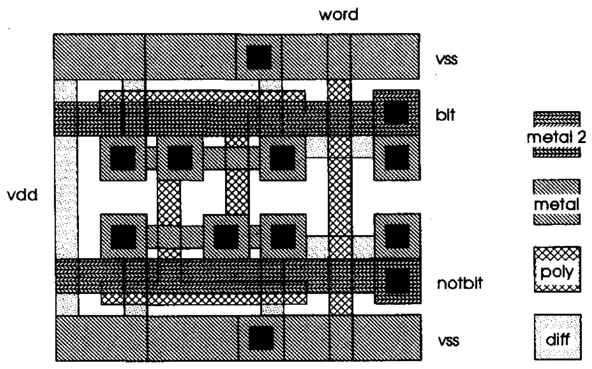


Figure 5.18. Layout of the SRAM cell.

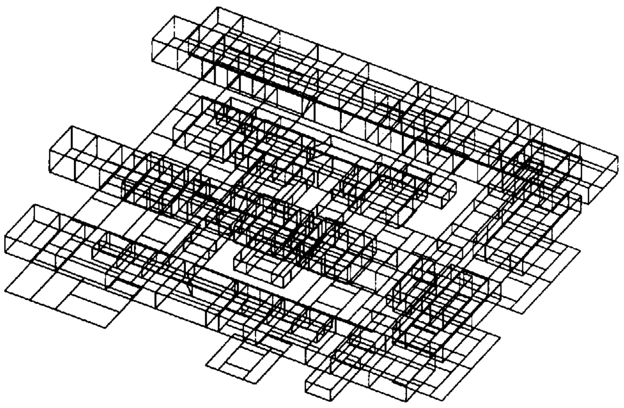


Figure 5.19. Finite element mesh of the SRAM cell.

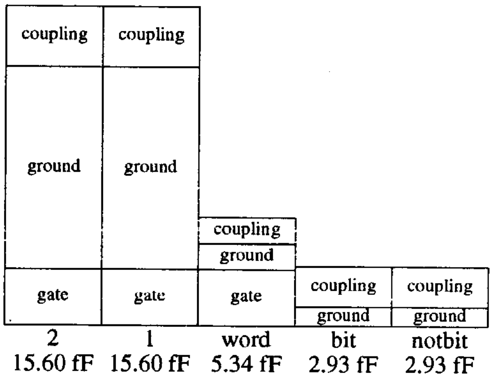


Figure 5.20. Results of the SRAM cell extraction.

With respect to these results, we make the following remarks:

1. Nodes 1 and 2 are diffused conductors (transistor source and drain regions). This explains the relatively large ground capacitance for these nodes.

2. The bit line and inverted bit line have relatively large coupling capacitances. This coupling can cause severe signal degradation, and must be taken into account in the design of the memory chip. See also [Konishi (1989)], who used a three-dimensional capacitance extractor to analyze bit line coupling noise in

DRAM circuits.

3. Coupling capacitances determined from the extraction of a single cell, as done here, actually underestimate the coupling compared to the same cell embedded in an array. This can be explained by noting that, for e.g. the bit lines, there is now a significant contribution to the ground capacitance from both ends of the bit line. If the cell were embedded in an array, this would not be the case.

Finally, some statistics of the program are shown in Table 5.4.

Table 5.4. SRAM Extraction Results

number of finite elements	839
number of Green evaluations	232663
window size (μ)	4
number of strips	5
maximum matrix dimension	466
maximum width of band	177
memory needed (Mbyte)	3.33
CPU time (min:sec)	2:42

5.6 Conclusion

In this chapter, we have developed algorithms and data structures for finite element based capacitance extraction, implementing the mathematical techniques described in Chapter 4. The algorithms have been implemented in the Space layout-to-circuit extractor, and experimentally verified. In particular, the results obtained with Space demonstrate the efficiency, accuracy and general practicality of the new algorithms. They do indeed satisfy their main goal: accurate layout-to-circuit extraction on the designer's workstation as a one-step transformation from a layout to an equivalent circuit.

Of course, the program as implemented, and described in Section 5.4, is only a prototype. On the one hand, it will have to be extended, for example, to handle more dielectric layers and to relieve the requirement of perfect planarization. Preferably, some optimizations as discussed in Appendix 5.3 must also be included. Most of these extensions and optimizations, however, are straightforward to implement and do not present scientific challenges. Chapter 6 discusses some extensions that do present scientific challenges.

On the other hand, the program can be simplified: the results obtained with our prototype indicate that, for the purpose of layout verification, zeroth-order elements provide sufficient accuracy.

Appendix 5.1 Diffused Conductors

The finite element model employed is based on metallic conductors embedded in a stratified dielectric above a good conducting ground plane. In practice, diffused conductors (which implement the transistor source and drain regions) do not fit into this model. However, to be able to evaluate our finite element method on real layouts, we used a heuristic approach to incorporate these diffusion paths in the capacitance extraction method. This approach is illustrated in Figure 5.21.

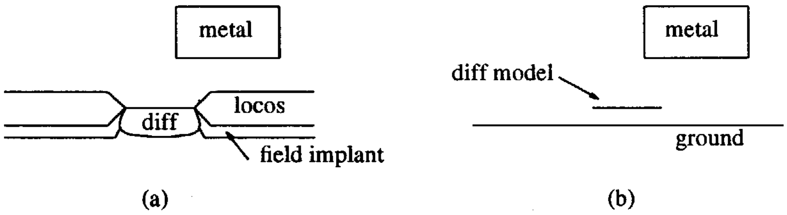


Figure 5.21. Illustration of the heuristic approach to incorporate diffusion capacitances, physical structure (a) and finite element model (b).

Figure 5.21(a) shows a cross-sectional view of a diffused conductor. The finite element model employed for such a conductor is shown in Figure 5.21(b), where the diffused interconnect is replaced by a thin sheet conductor. The sheet conductor is positioned half the thickness of the field oxide above the ground plane, which is flat and continuous, and must be thought of as modeling the top side of the diffused conductors.

The program replaces all the diffusion conductors by such sheet conductors and, using the finite element method, computes the coupling capacitances between these sheet conductors and the other conductors, and mutually between sheet conductors. These capacitances are inserted in the extracted circuit. For the sheet conductors, the finite element method also yields the capacitances to the substrate. These are then discarded by the program and the values that result from conventional area/perimeter calculations are inserted in the circuit instead.

Although this approach is purely heuristic, its results are satisfactory when the width of the diffusion paths is significantly larger than the height of the sheet conductors above the ground plane. However, in Chapter 6 we propose further research to incorporate the diffused conductors in the finite element model.

Appendix 5.2 Combination with Resistance Extraction

The Space program also implements a finite element method for resistance extraction, as described in [Genderen (1988)] and [Genderen (1991)]. With this method, the conductors are modeled as thin sheet resistances on which a finite element mesh is formed, consisting of triangles and rectangles as shown in Figure 5.22.

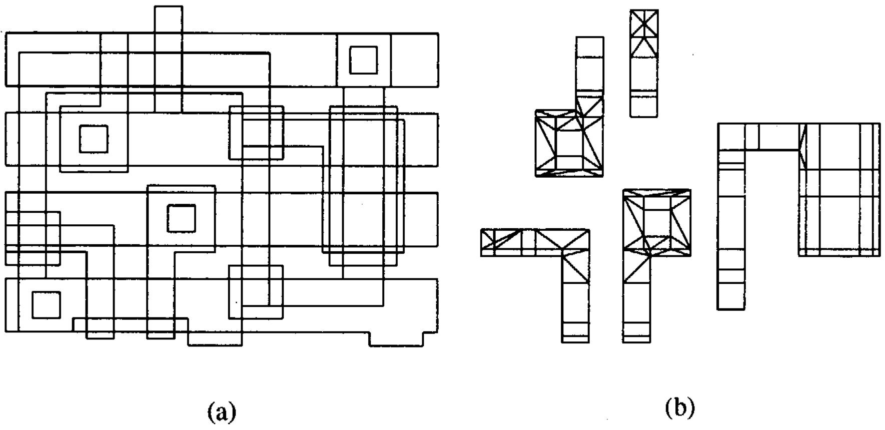


Figure 5.22. A layout example (a) and the resistance finite element mesh for the polysilicon mask (b).

The edges of this mesh can be viewed as resistances in a resistance network, of which the nodes are formed by the vertices of the finite element mesh. Capacitances are added to this mesh using geometric calculations. Capacitances associated with a conductor edge are equally divided over the two vertices in the resistance mesh that delimit the edge, and capacitances associated with a conductor surface are divided according to the areas that are bounded by the lines of gravity of the boxes and triangles. The latter is illustrated in Figure 5.23.

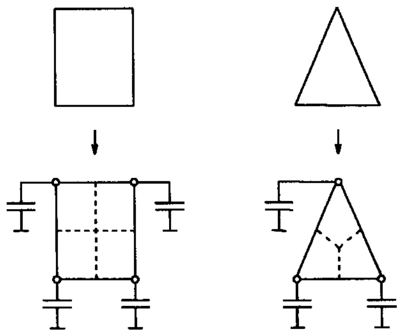


Figure 5.23. Assignment of capacitances to vertices.

The resulting detailed RC model is reduced (i.e. simplified) by an optimal node elimination algorithm that preserves the first-order or Elmore [Elmore (1948)] time constants.

The two techniques, finite element based resistance extraction and finite element based capacitance extraction, can be combined by setting up a suitable incidence relationship between the vertices in the resistance mesh (*r-vertices*) and the finite elements in the capacitance mesh (*c-elements*). This should be based on proximity: a c-element must be incident to the closest r-vertex.

Appendix 5.3 Discussion on Optimization

Although the algorithms for capacitance extraction developed in this dissertation result in a $O(N)$ time complexity, with N representing the size of the layout, the associated constant factor is relatively high. There are two bottlenecks that mainly determine its magnitude: the computation of the entries of the influence matrix and the inversion of the matrix[4]. In this appendix, we present a number of possibilities for optimizing the speed of these steps. Some of these have indeed been implemented in Space.

1. Most of the finite elements are in three strips: one of width $2w$, one of width $2w$ displaced over a distance w to the right and one of width w corresponding to an overlap of two $2w$ strips. Thus, many finite element interactions are evaluated 2 or 3 times. As a result, the number of such computations is 5/3 times the minimum possible. Compared to recomputation, the storage and reuse of previous results is faster. For that purpose, 2 FIFO data structures are sufficient. Instead of relying on the virtual memory performance of the system, these queues are conveniently maintained in temporary files.

2. Even when the number of Green's function computations is reduced by using the technique described above, it remains desirable to further reduce the computer time associated with this step. A particular heuristic approach avoids the exact integration of the finite element shape functions for finite elements that are far apart. Instead, a multipole approximation of the integral can be carried out much faster [Newman (1986)].

3. In the case of constant boundary elements, we have seen that it is advantageous to apply a mixture of n-sided elements with $n \geq 3$. In such a boundary element mesh, many of the elements will have $n \geq 5$. In general, the 2-dimensional integrations over n-sided elements require the integration boundaries to consist of n parts. Numerical integration can then be performed by decomposing the integration domain into $n-2$ triangles and using a 2-dimensional integration formula for a triangular domain [Stroud (1971), Hammer (1956)]. In practice, this integration can be accelerated significantly by disregarding, for the triangulation, vertices

4. Of course, since the complexity of evaluating the influence matrix is $O(w^2)$ and of inverting it is $O(w^4)$, the inversion will dominate for large window sizes w.

common to two co-linear edges. It can be accelerated even further when elements with a rectangular shape are not subdivided at all, but are integrated directly using a 2-dimensional integration formula for a rectangular domain [Stroud (1971)].

4. Influence matrix computation can also be accelerated by using a table lookup mechanism. Probably the best way to implement this lookup table is to turn it into a hash table indexed by a key constructed from the parameters describing the shape and (relative) location of the finite elements. The size of the hash table should be limited, and it should operate like a cache. The hit/miss ratio of this cache can be optimized if the finite element mesh generated is as regular and repetitive as possible.

5. For the hierarchical Schur algorithm, coarse-grain parallelism is trivially possible by allocating different blocks of the matrix to different processors. The communication and scheduling overhead will be low enough to justify this method even when the processors are individual (UNIX) computers or workstations coupled together in a local area network.

6. The Schur algorithm can be implemented to run efficiently on a vector computer [Lossie (1988)].

7. Alternatively, special purpose hardware can be developed. A CORDIC chip [Lange (1988)], for example, can be a processor in a systolic architecture for the Schur algorithm [Bu (1990), Dewilde (1988)].

References

Bu (1990)	J. Bu, "Systematic Design of Regular VLSI Processor Arrays," Ph.D. Dissertation, Delft University of Technology, Delft, the Netherlands (May 1990).
Dewilde (1988)	P. Dewilde, "New Algebraic Methods for Modelling Large-Scale Integrated Circuits," *International Journal of Circuit Theory and Applications* **16** pp. 473-503 (1988).
Elmore (1948)	W.C. Elmore, "The Transient Response of Damped Linear Networks with Particular Regard to Wideband Amplifiers," *J. Applied Physics* **19** pp. 55-63 (Jan. 1948).
Garey (1978)	M. Garey, D.S. Johnson, F.P. Preparata, and R.E. Tarjan, "Triangulating a simple polygon," *Inform. Processing Lett.* **7**(4) pp. 175-180 (1978).
Genderen (1988)	A.J. van Genderen and N.P. van der Meijs, "Extracting Simple but Accurate RC Models for VLSI Interconnect," *Proc. ISCAS-88*, Helsinki, Finland, pp. 2351-2354 (June 7-9, 1988).
Genderen (1991)	A.J. van Genderen, "Reduced Models for the Behavior of VLSI Circuits," Ph.D. Dissertation, Delft University of Technology, Delft, the Netherlands (1991).
Grimm (1983)	M.A. Grimm, K.Lee, and A.R. Neureuther, "SIMPL-1 (Simulated Profiles from the Layout—Version 1)," *IEDM Technical Digest*, pp. 255-258 (1983).
Hammer (1956)	P.C. Hammer, O.J. Marlowe, and A.H. Stroud, "Numerical Integration over Simplexes and Cones," *Math. Tables Aids Comput.* **10** pp. 130-137 (1956).
Hoffmann (1989)	C.M. Hoffmann, "The Problems of Accuracy and Robustness in Geometric Computation," *IEEE Computer Magazine* **22**(3) pp. 31-41 (March 1989).
Konishi (1989)	Y. Konishi, M. Kumanoya, H. Yamasaki, K. Dosaka, and T. Yoshihara, "Analysis of Coupling Noise Between Adjacent Bit Lines in Megabit DRAMS," *IEEE Journal of Solid-State Circuits* **SC-24**(1) pp. 35-42 (Feb. 1989).

Koppelman (1983) George M. Koppelman and Michael A. Wesley, "OYSTER: A Study of Integrated Circuits as Three-Dimensional Structures," *IBM J. Res. Develop.* **27**(2) pp. 149-163 (Mar. 1983).

Lange (1988) A.A.J. de Lange, A.J. van der Hoeven, E.F Deprettere, and J. Bu, "An Optimal Floating Point Pipelined CMOS CORDIC Processor," *Proc. ISCAS 1988*, Helsinki, Finland, pp. 2043-2047 (June, 1988).

Lee (1983) K. Lee, Y. Sakai, and A.R. Neureuther, "Topography-Dependent Electrical Parameter Simulation for VLSI Design," *IEEE Trans. Electron Devices* **ED-30**(11) pp. 1469-1474 (Nov. 1983).

Lee (1985) K. Lee and A.R. Neureuther, "SIMPL-2 (SIMulated Profiles from the Layout—Version 2)," *1985 Symposium on VLSI Technology, Digest of Technical Papers*, Kobe, Japan, pp. 64-65 (1985).

Lossie (1988) M.L.F. Lossie, "The Generalized Schur Algorithm: Roundoff Analysis, Vectorization, and an Application to VLSI Modelling," Internal Report, Delft University of Technology, Delft, the Netherlands (Feb. 1988).

Miller (1989) J.R. Miller, "Architectural Issues in Solid Modelers," *IEEE Computer Graphics and Applications*, pp. 72-87 (Sept. 1989).

Nabors (1991) K. Nabors and J. White, "FastCap: A Multipole Accelerated 3-D Capacitance Extraction Program," *IEEE Trans. on CAD* **CAD-10**(10) pp. 1447-1459 (Nov. 1991).

Newman (1986) J.N Newman, "Distributions of Sources and Normal Dipoles over a Quadrilateral Panel," *Journal of Engineering Mathematics* **20** pp. 113-126 (1986).

Oldham (1979) W.G. Oldham, S.N. Nandgaonkar, M.M O'Toole, and A.R. Neureuther, "A General Simulator for VLSI Lithography and Etching Processes: Part 1—Application to Projection Lithography," *IEEE Trans. on Electron Devices* **ED-26**(4) pp. 717-722 (Apr. 1979).

Oldham (1980) W.G. Oldham, A.R. Neureuther, C. Sung, J.L. Reynolds, and S.N. Nandgaonkar, "A General Simulator for VLSI Lithography and Etching Processes: Part 2—Application to Deposition and Etching," *IEEE Trans. on Electron Devices* **ED-27**(8) pp. 1455-1459 (Aug.

1980).

Ruehli (1973) A.E. Ruehli and P.A. Brennan, "Efficient capacitance calculations for three-dimensional multiconductor systems," *IEEE Trans. on Microwave Theory and Techniques* **MTT-21**(2) pp. 76-82 (Feb. 1973).

Ruehli (1975) A.E. Ruehli and P.A. Brennan, "Capacitance models for integrated circuit metallization wires," *IEEE Journal of Solid-State Circuits* **SC-10**(6) pp. 530-536 (Dec. 1975).

Small (1990) M.B. Small and D.J. Pearson, "On-chip Wiring for VLSI: Status and Directions," *IBM J. Res. Develop.* **34**(6) pp. 858-867 (Nov. 1990).

Stroud (1971) A.H. Stroud, *Approximate Calculation of Multiple Integrals,* Prentice Hall, New Jersey (1971).

Wesley (1983) M.A. Wesley, T. Lozano Perez, L.I. Liebermann, M.A. Lavin, and D.D. Grossman, "A Geometric Modeling System for Automated Mechanical Assembly," *IBM J. Res. Develop.* **27**(2) pp. 149-163 (Mar. 1983).

Wilton (1984) D.R. Wilton, S.M. Rao, A.W. Glisson, D.H. Schaubert, O.M. Al-Bundak, and C.M Butler, "Potential Integrals for Uniform and Linear Source Distributions on Polygonal and Polyhedral Domains," *IEEE Trans. on Antennas and Propagation* **AP-32**(3) pp. 276-281 (Mar. 1984).

Zemanian (1989) A.H. Zemanian, R.P. Tewarson, C.P. Ju, and J.F. Jen, "Three-Dimensional Capacitance Computations for VLSI/ULSI Interconnections.," *IEEE Trans. on CAD* **CAD-8**(12) pp. 1319-1326 (Dec. 1989).

6. Conclusion

In this dissertation, we have developed new techniques for accurate and efficient layout-to-circuit extraction. Compared to traditional approaches, the increased accuracy and efficiency are needed because of the continuous increase in integration density.

The algorithms and data structures developed have been implemented in a layout-to-circuit extractor called Space, which is briefly described in Sections 3.7 and 5.4. The success and practicality of the new algorithms were verified by experimental results obtained with the program. These results where evaluated and compared to other results available in the literature.

Despite the great step forward in accuracy offered by the finite element technique described in Chapter 4, further improvements are necessary. In particular, we have treated the problem of modeling IC interconnections assuming that there is a perfect planarization of the dielectric layers and that the substrate is a perfect ground plane, as shown in Figure 6.1.

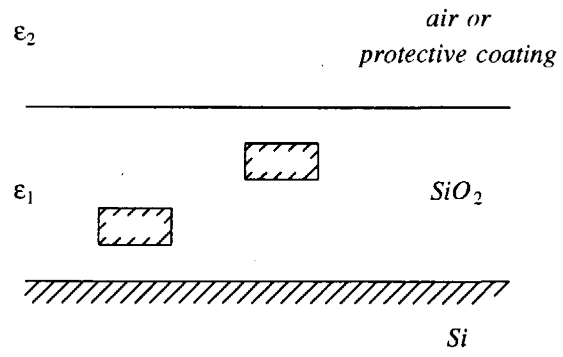


Figure 6.1. The silicon substrate as a ground plane with stratified dielectrics.

Further research must be aimed at relaxing these assumptions and including new effects. In particular, the method must be extended to:

1. Model and determine interconnect capacitances in non-planar technologies accurately and efficiently.
2. Include the bulk interconnects (the so called "diffusion paths") in the resulting capacitance model.
3. Model and determine the substrate resistance and its effects. These include substrate current flow, feedback, crosstalk, noise and electromagnetic losses.

As we have seen in Chapter 2, parasitic inductance is also becoming important. Future research must therefore be aimed at accurate extraction of inductances, or transmission line effects in general, as well. This is especially important because of the proliferation of bipolar (or BICMOS) integrated circuits.

The determination of parasitics (i.e. the values of the parasitic elements) is but one step in the verification process—it results in the electrical model of the chip. This model must subsequently be analyzed, thereby extrapolating the behavior of the integrated circuit. This analysis can be accomplished through simulation or static (timing) analysis. However, when the extracted models become more complex (i.e. when their size or their number of components increase), the effectiveness of traditional analysis tools is often unsatisfactory. This subject will therefore also require additional research.

Samenvatting

Nauwkeurige en Efficiënte Layout-extractie

Extractie wordt in dit proefschrift gedefinieerd als ''het modelleren en bepalen van de elektrische eigenschappen van geïntegreerde schakelingen (IC's), uitgaande van de layout en relevante gegevens over het fabricageproces''. Als resultaat ontstaat een *equivalent circuit*, bestaande uit actieve en passieve elementen zoals transistoren, weerstanden en capaciteiten. De correctheid van de schakeling kan dan gecontroleerd worden, vóórdat de schakeling gefabriceerd wordt, door middel van simulatie of statische analyse van het equivalente circuit.

Deze verificatie stap wordt steeds belangrijker. Met de steeds maar verdergaande verkleining van de afmetingen van de elementen op een IC, en de afname van hun schakeltijden, wordt het elektrische gedrag van geavanceerde IC's steeds sterker bepaald door onbedoelde, parasitaire effecten. Voorbeelden van zulke effecten zijn thyristor structuren in CMOS schakelingen die latch-up kunnen veroorzaken, de capaciteiten, weerstanden en inductanties van de bedrading op een chip, en de weerstand van het substraat.

In dit proefschrift zoeken we naar modelleringstechnieken die nauwkeurig en betrouwbaar het elektrische gedrag van geïntegreerde schakelingen voorspellen: nieuwe technieken zijn nodig om effecten te beschrijven die vroeger niet belangrijk waren of waarvoor de nauwkeurigheid van standaardtechnieken te wensen overlaat.

De zo verkregen modellen moeten niet alleen nauwkeurig zijn, maar ook efficiënt. Ze moeten zo eenvoudig en compact als mogelijk zijn, en alle belangrijke effecten beschrijven en de onbelangrijke weglaten. De modellen moeten, bijvoorbeeld, geen kleine capaciteiten tussen ver uit elkaar liggende elementen bevatten. Echter, het totaal van al deze kleine capaciteiten kan toch een niet-verwaarloosbare invloed hebben op de vertraging van de schakeling, dus zomaar weglaten van deze capaciteiten is vaak ontoelaatbaar. Ze kunnen vaak wel verdisconteerd worden in de andere capaciteiten.

Net als de modellen, moeten ook de algoritmen (programma's) om deze modellen te bepalen efficiënt zijn. De rekentijd, benodigd door de algoritmen als functie van de grootte van de schakeling (hun tijd-complexiteit), moet zo laag mogelijk zijn. In dit proefschrift leggen we de nadruk op lineaire-tijd algoritmen. Zulke algoritmen zijn belangrijk in het licht van de steeds verder toenemende complexiteit van IC's.

Ook het geheugenbeslag van de algoritmen (hun geheugen-complexiteit) moet zo laag mogelijk zijn. Ondanks het belang van snelle algoritmen is de rekentijd in theorie onbegrensd. In de praktijk wordt de maximale grootte van een probleem dat opgelost kan worden veelal onbarmhartig beperkt door de grootte van het werkgeheugen van de gebruikte computer. Ook virtueel geheugen vormt geen oplossing, omdat dat óók begrensd is en omdat veelvuldig wisselen van de data tussen werkgeheugen en achtergrondgeheugen te veel tijd kan kosten. De geheugen-complexiteit van de algoritmen is dus eigenlijk nog belangrijker als hun tijd-complexiteit. In dit proefschrift streven we naar algoritmen met een sub-lineaire geheugen-complexiteit.

De technieken uit dit proefschrift zijn het meest effectief wanneer ze toegepast worden in de *layout ontwerp-lus*. Alleen dan is het mogelijk om kostbare *redesigns*, als gevolg van het te laat ontdekken van een probleem, te voorkomen. De modelleringstechnieken uit dit proefschrift moeten dus ingebouwd (kunnen) worden in de verificatie-programmatuur zoals een IC ontwerper die op zijn werkstation ter beschikking heeft. Het resulterende programma moet gebruikersvriendelijk zijn, en moet van de IC ontwerper geen kennis vereisen van de achterliggende wiskundige theorieën. Dit stelt eisen aan de opbouw van de algoritmen.

Gegeven bovengenoemde uitgangspunten, is dit proefschrift als volgt ingedeeld:

In hoofdstuk 2 wordt eerst het elektrische gedrag van de bedrading op een IC bestudeerd. Dit gedrag kan in de regel worden beschreven in termen van verdeelde weerstanden, capaciteiten en inductanties. We gaan na hoe, en onder welke omstandigheden, zo'n beschrijving vereenvoudigd kan worden. Ook gaan we na wat het effect is van de verkleining van de afmetingen op IC's op het elektrische gedrag van de bedrading. Een van de conclusies die in dit hoofdstuk getrokken wordt is dat bedradingscapaciteit steeds belangrijker wordt, maar dat de gangbare technieken om deze te bepalen niet voldoen voor geavanceerde IC technologieën en kritische ontwerpen.

In hoofdstuk 3 behandelen we het probleem efficiënt om te gaan met de enorme hoeveelheid geometrische gegevens die de layout van een VLSI (Very Large Scale Integration) schakeling beschrijven, met name toegespitst op het layout-extractie-probleem. We ontwikkelen onder andere een combinatie van de zogenaamde *corner-stitching* en *scanline* methoden, welke een lineaire tijd-complexiteit en een sub-lineaire geheugen-complexiteit realiseert. De algoritmen uit dit hoofdstuk zijn geïmplementeerd in een layout-extractieprogramma, genaamd Space, en de resultaten verkregen met dit programma bevestigen hun goede prestaties. Space is opgenomen in het NELSIS IC ontwerpsysteem.

In hoofdstuk 4 bespreken we de mathematiek en, in het kort, de theoretische achtergrond van een zogenaamde rand-elementenmethode voor de nauwkeurige berekening van de bedradingscapaciteiten van geïntegreerde schakelingen. Dankzij een nieuw algoritme voor het benaderen van de inverse van een matrix, onstaat een lineaire tijd-complexiteit en een constante geheugen-complexiteit. Het resulterende capaciteits-netwerk beschrijft nauwkeurig de totale belasting van alle signalen, maar bevat geen kleine, irrelevante, capaciteiten tussen afgelegen elementen.

In hoofdstuk 5 bekijken we hoe de mathematische concepten en technieken uit hoofdstuk 4 omgezet kunnen worden in praktische, efficiënte en krachtige algoritmen, die in een layout-extractieprogramma ingebouwd kunnen worden. Zaken die hierbij aan de orde komen zijn, onder andere, de generatie van een 3-dimensionaal rand-elementenmodel van de schakeling en de koppeling met de eigenlijke netwerk-herkenningsalgoritmen van het extractieprogramma. De algoritmen zijn inderdaad geïmplementeerd in het programma Space, en de resultaten verkregen met het programma tonen aan dat de ontwikkelde technieken het mogelijk maken nauwkeurige capaciteits-extractie op te nemen in de layout ontwerp-lus.

Het proefschrift eindigt in hoofdstuk 6 met conclusies en suggesties voor verder onderzoek.

Acknowledgements

The research presented in this dissertation was supported in part by the Commission of the EC under Esprit contract 991 and by the Dutch Foundation for Fundamental Research on Matter (FOM), under IOP-IC contract 45.009. The contributions of all participants and facilitators are hereby gratefully acknowledged. The Network Theory Section of the Department of Electrical Engineering at the Delft University of Technology has provided a pleasant and stimulating environment in which to work. I wish to express my gratitude to all those working on the 16th floor, especially to Prof. P.M. Dewilde for being an inspiring organizer and leader, the Secretary Corrie Boers-Boonekamp for assisting where needed, and Arjan van Genderen for having worked toghether so pleasantly. For their work, I am, in particular, indebted to Prof. P.M. Dewilde, Arjan van Genderen, Zhen-Qiu Ning and Harry Nelis—it was instrumental in achieving the results of the Space project. I wish to thank Theo Smedes for proofreading the draft and Susan Massotty for eliminating many linguistic and stylistic errors from the English text. Lastly, my most special thanks go to my wife, Antoinette, for her love and acceptance of my occupation in the course of this work.

Biography

Nick van der Meijs was born in Maasland, the Netherlands, on November 1, 1959. In 1978, he received a VWO diploma from the St. Stanislas College in Delft and in 1985 he received an M.Sc. Degree in Electrical Engineering (cum laude) from the Delft University of Technology in Delft, the Netherlands. In August 1985, he joined the Network Theory Section of the Department of Electrical Engineering at the Delft University of Technology, to work as a research assistant under the supervision of Prof. P.M. Dewilde. Since March 1, 1990, he has been an assistant professor. He has taught classes on algorithms and data structures for the computer-aided design of integrated circuits and has conducted research into various topics, including module generation, frameworks and physical design verification. He has contributed extensively to the design and implementation of the Nelsis IC design system. At present, his main research interests are in the field of practical algorithms for modeling parasitic effects in advanced integrated circuits.