TU/e EINDHOVEN UNIVERSITY OF TECHNOLOGY

A generalised approach to gate array layout design automation

Citation for published version (APA):
Slenter, A. G. J. (1990). *A generalised approach to gate array layout design automation*. [Phd Thesis 1 (Research TU/e / Graduation TU/e), Electrical Engineering]. Technische Universiteit Eindhoven. https://doi.org/10.6100/IR342622

DOI:
10.6100/IR342622

Document status and date:
Published: 01/01/1990

Document Version:
Publisher's PDF, also known as Version of Record (includes final page, issue and volume numbers)

Please check the document version of this publication:

• A submitted manuscript is the version of the article upon submission and before peer-review. There can be important differences between the submitted version and the official published version of record. People interested in the research are advised to contact the author for the final version of the publication, or visit the DOI to the publisher's website.
• The final author version and the galley proof are versions of the publication after peer review.
• The final published version features the final layout of the paper including the volume, issue and page numbers.

Link to publication

General rights
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

• Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
• You may not further distribute the material or use it for any profit-making activity or commercial gain
• You may freely distribute the URL identifying the publication in the public portal.

If the publication is distributed under the terms of Article 25fa of the Dutch Copyright Act, indicated by the "Taverne" license above, please follow below link for the End User Agreement:

www.tue.nl/taverne

Take down policy
If you believe that this document breaches copyright please contact us at:

openaccess@tue.nl

providing details and we will investigate your claim.

Download date: 04. Oct. 2023

A GENERALISED APPROACH TO GATE ARRAY LAYOUT DESIGN AUTOMATION

André G.J. Slenter

A
GENERALISED APPROACH
TO
GATE ARRAY LAYOUT DESIGN AUTOMATION

Dit proefschrift is goedgekeurd door de promotoren

Prof.dr.-ing. J.A.G. Jess en Prof.dr.ir. R.H.J.M. Otten

© Copyright 1990 André G.J. Slenter
All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, photocopying, recording, or otherwise, without the prior written permission from the copyright owner.

Druk: Dissertatiedrukkerij Wibro, Helmond

CIP-GEGEVENS KONINKLIJKE BIBLIOTHEEK, DEN HAAG

Slenter, André Guillaume Joseph

A generalised approach to gate array layout design automation/
André Guillaume Joseph Slenter. - [S.l. : s.n]. - Fig., tab.
Proefschrift Eindhoven. - Met index, lit.opg., reg.
ISBN 90–9003442–0
SISO 663.43 UDC 621.382:681.3.06 (043.3) NUGI 832
Trefw.: geïntegreerde schakelingen; computer aided design / algoritmen.

A
GENERALISED APPROACH
TO
GATE ARRAY LAYOUT DESIGN AUTOMATION

PROEFSCHRIFT

ter verkrijging van de graad van doctor aan de Technische Universiteit Eindhoven, op gezag van de Rector Magnificus, prof. ir. M. Tels, voor een commissie aangewezen door het College van Dekanen in het openbaar te verdedigen op dinsdag 11 december 1990 om 14.00 uur

door

André Guillaume Joseph Slenter

geboren te Kessel

Contents

Abstract

The topic of this thesis concerns layout design automation for gate array IC's. It presents a conceptual framework that provides a generalised approach to gate array design automation. The presented concepts are new because the intended generality is defined as the center point around which a consistent framework is developed.

Up to now, generality of gate array layout design systems is achieved as a, sometimes unexpected, side effect. The implication of a general approach is that all aspects concerning the customisation of gate array IC's must be reconsidered. These aspects can be summarised by the modeling of the layout patterns, the characterisation of the gate array families and designs to be implemented on a chosen gate array IC, and the applied placement and routing strategies. This indicates the elementary problem of gate array layout design: most algorithms developed do not provide solutions in the general case and can therefore be applied in a very limited range of layout design problems.

The solution, presented in this thesis, gives the user the capability to equip his design environment with a layout design system of an enormous generality. Virtually any layout style can be supported in almost any technology. The adjustments are achieved by a standardised user interface and don't require any intrusion into the software code of the system. The presented solution has two faces. The first face concerns the formulation of the gate array and design data, based on a generalisation of gate array design by inclusion of details. Further, only one abstraction is defined to eliminate the direct link to a specific technology. The space-graph is introduced as a new way to describe layout patterns and related information. Based on this space-graph, a precise, yet compact and general description of gate array and

design data is defined. The second face concerns the embedding of layout design algorithms in the general context at hand. The placement problem is divided into two phases, referred to as global and detailed placement. It is shown that by a simplification of the global placement problem, known placement algorithms can be used with minimal modifications. A new algorithm is presented for detailed placement. The routing problem is also solved in two stages, generally referred to as global and detailed routing. Global routing is done hierarchically in successively smaller but more detailed areas. Detailed routing consists of a maze runner principle tuned to the description of the design area by the space-graph. An important aspect of the detailed routing is that placed modules are transparent for routing. The outline of an overall routing strategy, incorporating both global and detailed routing, is presented. It is shown that this strategy can solve some of the problems occurring with separate global and detailed routing. Those problems are associated with the requirement of an accurate definition of the global routing boundary capacities on one hand, and with the cpu-time and memory requirements for detailed routing on the other hand.

In order to evaluate the concepts presented in this thesis, a prototype Gate Array design System (GAS) has been implemented. The basics of this system are that the gate arrays to be used are described in a specially developed language (GADL) and are next compiled into a common data base. The second part of the system consists of a small selection of coherent placement and routing algorithms.

Up to now, five different gate arrays have been used for evaluation. These experiments reveal that tuning GAS to a new gate array requires about one month. Further, the amount of data required by GAS for gate array and design descriptions is in the order of 10 Kbytes to 1 Mbytes, which is acceptable throughout the industrial world. These experiments also show that the design algorithms need further consideration. Although the incorporated placement strategy has proven to be very flexible, better algorithms are required in order to achieve higher area utilisations within acceptable design time. The current routing strategy is capable to deal with large and complex designs occupying more than 90% of the available design space. Further improvements can be achieved by a combined global and detailed routing strategy.

Samenvatting

Het onderwerp van dit proefschrift betreft het automatisch ontwerpen van bedrading voor gate array IC's. Er wordt een raamwerk gepresenteerd dat een generalisering van dit ontwerpprobleem realiseert. De gepresenteerde concepten zijn nieuw omdat de flexibiliteit het uitgangspunt is geweest bij de ontwikkeling van een consistent raamwerk.

Tot op heden wordt flexibiliteit van gate array ontwerpsystemen bereikt als een, soms onverwacht, neveneffect. De implicatie van een gegeneraliseerde aanpak is dat alle aspecten betreffende de personalisering van een gate array IC heroverwogen moeten worden. Samengevat zijn deze aspecten: de modellering van de layout patronen, de karakterisatie van de gate array families en de te realiseren ontwerpen, en de toegepaste plaatsings- en bedradings-algoritmen. Dit geeft tevens een indicatie van het gegeneraliseerde gate array layout ontwerp-probleem: de meeste algoritmen geven geen oplossingen voor de gegeneraliseerde situatie en zijn daarom slechts beperkt toepasbaar.

De oplossing, zoals gepresenteerd in dit proefschrift, biedt de gebruiker de mogelijkheid om zijn ontwerpomgeving uit te rusten met een layoutontwerpsysteem met vrijwel ongelimiteerde mogelijkheden. Bijna elke layoutstijl in iedere technologie kan ondersteund worden. Aanpassingen worden bereikt door een gestandariseerde user-interface, en vergen géén ingrijpen in de software-code van het systeem. De gepresenteerde oplossing is tweeledig. Het eerste deel betreft de formulering van gate array en ontwerp data, gebaseerd op een generalisering van gate array ontwerpen door detaillering. Verder is er slechts één abstractie gedefiniëerd om de directe link met een specifieke technologie te elimineren. De space-graaf wordt geïntroduceerd als een nieuwe manier om layout patronen en gerelateerde informatie te beschrijven. Gebaseerd op deze space-graaf wordt een precieze, doch compacte en algemene beschrij-

ving van gate array en ontwerp data gedefiniëerd. Het tweede deel betreft de inpassing van layout ontwerp-algoritmen in de gegeneraliseerde context. Het plaatsings-probleem is gesplitst in twee fasen, genaamd globale en gedetailleerde plaatsing. Door een simplificatie van het globale plaatsings-probleem kunnen bekende algoritmen worden ingepast met minimale modificaties. Voor gedetailleerde plaatsing wordt een nieuw algoritme gepresenteerd. Het bedradingsprobleem wordt ook in twee fasen opgelost, genaamd globale en gedetailleerde bedrading. De globale bedrading wordt hierarchisch gedaan in successievelijk kleinere, meer gedetailleerde regio's. De gedetailleerde bedrading is gebaseerd op een "maze-runner" principe, toegespitst op de beschrijving van de ontwerp-ruimte door de space-graaf. Een belangrijk aspect hierbij is dat de geplaatste modules transparant zijn. Een laatste aspect betreft de integratie van zowel globale als gedetailleerde bedrading in één, consistente, bedradings-strategie. Deze strategie is in staat om een aantal problemen op te lossen die zich voordoen bij separaat globale en gedetailleerde bedrading. Deze problemen zijn geassociëerd met de noodzaak van een accurate definitie van de globale bedradings-capaciteiten enerzijds, en met de rekentijd en geheugen-behoefte van de gedetailleerde bedrading anderzijds.

Ter evaluatie van de in dit proefschrift gepresenteerde concepten is een prototype Gate Array ontwerp Systeem (GAS) geïmplementeerd. Het basis concept van dit systeem is dat gate arrays worden beschreven in een daartoe speciaal ontwikkelde taal (GADL) en vervolgens gecompileerd in een algemeen data bestand. Het systeem bestaat verder uit een kleine selectie van coherente plaatsings- en bedradings-algoritmen.

Tot op heden zijn een vijftal verschillende gate arrays geëvalueerd. Hieruit blijkt dat het aanpassen van GAS aan een nieuw gate array ongeveer één maand in beslag neemt. Verder is de hoeveelheid data, nodig voor de beschrijvingen van gate arrays en ontwerpen, in de orde van 10 Kbytes tot 1 Mbytes, wat acceptabel is voor industriële toepassingen. Tevens blijkt dat de plaatsings-algoritmen een nadere beschouwing behoeven. Alhoewel de gebruikte plaatsings-strategie zeer flexibel is, zijn betere algoritmen nodig om een hogere benuttingsgraad te bereiken. De bedradings-strategie is in staat om oplossingen te vinden voor ontwerpen die meer dan 90% van de beschikbare oppervlakte in beslag nemen. Verdere verbeteringen zijn mogelijk door de gecombineerde globale en gedetailleerde bedradings-strategie.

1. Introduction

This thesis is about layout design automation for gate array IC's. Gate arrays are application specific IC's (ASICS) with preprocessed layout patterns, so-called "master slices". Customisation of a gate array is achieved by designing the application specific interconnections of the preprocessed layout patterns. Gate arrays differ in many aspects. The differences are in the logic family they implement, the geometry of the preprocessed patterns and the technology. All these aspects must be taken into account while customising a gate array IC.

The great variety in gate arrays reflects the trade offs that can be made between technical and economic aspects. The two most important technical aspects concern the performance required of the designs [Beresford84] and the relation between the preprocessed patterns and the designs that can be implemented on a gate array IC [Gagliardi84]. An economic aspect concerns the number of interconnect layers available for customisation, which determines the processing cost of the IC and thus the break even point.

This thesis presents a conceptual framework that provides a generalised approach to gate array design automation. The presented concepts are new because the intended generality is defined as the center point around which a consistent framework is developed. Up to now, generality of gate array layout design systems is achieved as a, sometimes unexpected, side effect. The implication of this general approach is that all aspects, concerning the customisation of gate array IC's, must be reconsidered. These aspects can be summarised by the

modeling of the layout patterns, the characterisation of the of gate array families and designs to be implemented on a chosen gate array IC, and the applied placement and routing strategies. The relation between these four aspects is shown in figure 1.1.

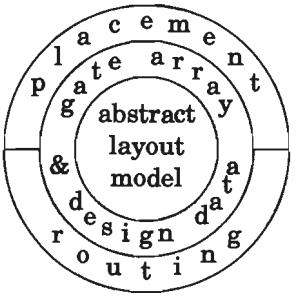


Figure 1.1. Conceptual overview of a generalised gate array layout design automation approach.

The generality provided by layout modeling is in the definition of the transformation of abstract layout pattern descriptions to a specific processing technology. This eliminates the necessity of technology specific information in the remaining customisation stages. Gate array and design characterisation provides a uniform description that captures all essential information required for customisation of gate array IC's. The definition of placement and routing strategies is essential in order to provide correct design solutions without introducing unnecessary constraints by ignoring the available information.

1.1 Layout design automation

Although customisation of gate array IC's only concerns the application specific interconnection patterns, it is still a complex task. Without claiming to be complete, four aspects can be mentioned that are essential for gate array layout design. A first aspect concerns the design space. Due to the preprocessed master slice, the available space in which the application specific interconnection patterns must be designed, is fixed. This implies that design decisions that determine the distribution of the interconnect patterns determine also whether a

design can be implemented in the given design space or not. A second aspect is to maximise the utilisation of the available design space. This is in principle an economic aspect. The cost of a gate array IC is, amongst others, determined by the size of the IC, in that smaller sized chips result in lower production cost. This implies that the smallest sized chip on which the design can be implemented is the optimal choice. A third aspect is the library of wiring patterns usually provided with every gate array master slice. When properly mapped onto the master slice, these wiring patterns perform a desired function at their terminal pins. More than one pattern may be given that implement the same function, yet differ in shape or in the positions where it may be mapped onto the master slice. The wiring patterns depend on the preprocessed structure of the master slice, as illustrated in figure 1.2.

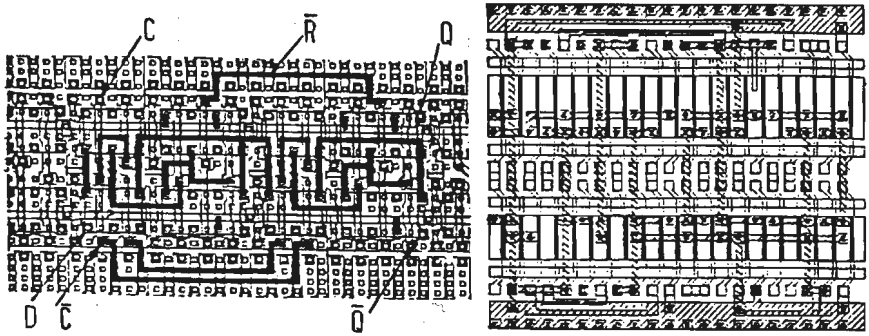


Figure 1.2. Example of two wiring patterns performing the same function for two different master slices.

The fourth and last aspect concerns the organisation of the routing area. Given a master slice, the routing area is fixed. Gate arrays differ in the organisation of routing area. In principle three different organisations can be recognised:

1) A row oriented structure, where routing channels are defined between other rows hosting the predefined wiring patterns.

2) An island oriented organisation, where routing areas are around islands containing the predefined patterns.
3) Sea-Of-Gates organisation, where no a priori routing areas are defined, but are determined by the unoccupied area after the predefined wiring patterns are placed.

Other important aspects are whether the predefined wiring patterns are regarded as transparent for routing, or routing may occur over the module area, or must be placed around the module areas. All these aspects determine the exact proceeding of the actual interconnect design.

The actual layout design problem is denoted by placement and routing. The placement and routing problems to be solved are known to be NP-complete [Garey79]. The restrictions mentioned above concerning the limited number of positions available for placement and the fixed routing areas, introduce additional complications. Under very special conditions those restrictions and the resulting limitation of the solution space can be used to develop algorithms that perform reasonably well under these circumstances. For placement, the most frequently used restriction is the assumption that modules to be placed have uniform shapes, resulting in a row oriented placement. For routing, the most frequently used assumption is that routing areas are organised in channels with no routing obstacles. In this situation efficient algorithms can be used that provide fast solutions, provided that the channels are not saturated.

Obviously, the major advantage of the placement and routing algorithms mentioned above is that they provide solutions for special cases within short time. The disadvantage of these algorithms is that they only can be applied if all the assumptions are valid. This indicates the elementary problem of gate array layout design: most algorithms developed do not provide solutions in the general case and can therefore be applied in a very limited range of layout design problems.

1.2 Generalisation versus specialisation

The complexity of the gate array layout design automation is reflected by the commercial gate array design systems available [VLSI87]. An overview of these systems clearly shows that they are dedicated to a

very small range of different gate array families. The major advantage of these systems is that they produce high quality design solutions for the gate array IC's they are intended for. This is achieved by development of design algorithms incorporating many assumptions of the gate array IC's on which designs must be implemented. Obviously, the incorporation of constraints in the design algorithms that reduce the solution space and indicate preferred solutions, accounts for fast design time and high quality solutions. But it also indicates the limited problem range for which these algorithms provide good solutions. This is shown by the restrictions on the range of possible applications of these systems. The most important restrictions are the number of interconnect layers, the presence of interconnect channels and the pin positions of the predefined wiring patterns. The most severe restrictions are in the routing capability of these systems. This was clearly shown with the introduction of the Sea-Of-Gates gate arrays around 1982. The switch to a channelless architecture, together with the application of gate isolation, marked the start of these second generation gate array IC's. The lack of good design software was clearly shown in some publications [Hsu86, Kubosawa87], revealing that channel routing algorithms were used for interconnect design. It took several years to develop software that was capable to deal with the second generation gate arrays. With a more generalised approach to gate array layout design these problems never would have existed.

In principle two approaches are possible to achieve a general approach to gate array layout design. The first approach is by abstraction of the problem. This implies that the eliminated details must be replaced by abstract constructs. The success of this approach is completely determined, and at the same time limited, by the number of constructs introduced. This also indicates the possible failures of the approach. The first failure is that with the introduction of new gate array types, new abstractions must be defined in order to capture the essential features of the new gate array types. This implies a software redesign because algorithms must be extended in order to interpret the newly introduced construct. A second failure occurs if an abstract construct can not discriminate a newly introduced design feature, which may result in wrong design solutions. The conclusion of this discussion is that abstraction not always provides the intended generality. The second approach to achieve generality is by inclusion of all the details. In this way, no additional restrictions are introduced that may limit

the application range. The first disadvantage of this method is that special care must be taken to limit the amount of data required. A second disadvantage is that relations are maintained that may imply unnecessary constraints, eg. the technology descriptions.

The approach presented in this thesis is, in principle, based on a generalisation of gate array layout design by inclusion of details. Further, only one abstraction is defined to eliminate the direct link to a specific technology. As will be shown in the rest of this thesis, this approach results in the intended generality of the gate array layout design.

1.3 A system concept

In order to evaluate the concepts presented in this thesis and to demonstrate the feasibility of these concepts, a prototype Gate Array design System (GAS) has been implemented. The architecture of GAS is outlined in figure 1.3.

The basics of this system are that the gate arrays types are described in a specially developed Gate Array Description Language (GADL) and these descriptions are compiled into a common data base. The structure of GADL reflects the organisation of the data within the data base and provides a flexible and natural way to describe gate arrays. The resulting data base is unique for every gate array family, and is built only once for every gate array description. The second part of the system consists of a design street, which is a small selection of coherent placement and routing algorithms. The data base contains sufficient information for the design algorithms to assure that the provided flexibility is used and that constraints are satisfied such that correct designs are obtained. Another important fact of the current implementation is the flexibility with respect to the design algorithms. Whenever possible, design algorithms are implemented as single programs, thus constructing a modular system that is easy to maintain from a software view, but is also easy to modify because a change of design strategy is achieved by a run-time-selection of design programs. GAS provides thus a gate array design framework, as shown in figure 1.3.

In the current implementation, the placement shell consists of an

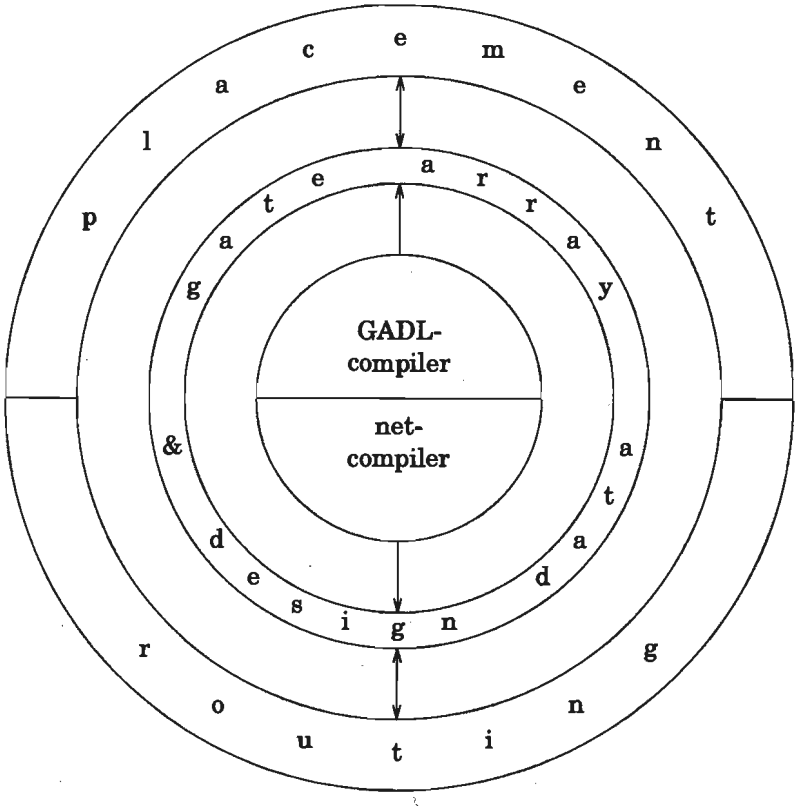


Figure 1.3. Overview of GAS.

interface to global placement tools. The two sample global placement algorithms implemented are based on simulated annealing [Otten84] and eigenvalue decomposition [Frankle86]. The routing shell is currently established by three routing tools, defining two routing strategies. The first strategy consists of a separate global routing tool, based on [Burstein83] and a maze runner [Lee61] for detailed routing. The second routing strategy is the combination of the two algorithms, mentioned above, into one consistent routing approach.

1.4 Relevance of the research

The relevance of the research presented in this thesis is indicated by an evaluation of the possible users of a commercial version of GAS. An evaluation of the IC market shows that there are in principle three groups: the customer, the broker and the foundry.

The customer, or client, specifies the design to be implemented in an IC. The specification concerns, amongst others, the functional specification of the design, the performance requirements, and the final product cost in order to make the IC profitable. The interface level between the customer and the broker is relatively high in that the functional description will not be more detailed than a net-list with elementary functions. The actual layout design is normally of no concern to the customer, as long as the specified requirements are met.

The broker acts as a representative between the customer and the foundry. At the moment, the most important task of the broker is seen as making customers less dependent of foundries. The expected effect will be that low budget customers are more willing to use the possibility of circuit integration. The advantage of GAS in the broker's situation is that a consistent customer interface can be maintained, while on the other side the flexibility towards the foundries is guaranteed. A broker will typically use GADL descriptions of gate arrays from different foundries, which can then be offered to customers for selection.

The foundry is the actual plant where gate arrays are developed and processed. The value of GAS in this environment is that newly developed master slice structures can easily be evaluated. This eliminates the necessary development of new design software and replaces this development with the formulation of a GADL description of the master slice, which is easier and faster.

The current implementation of GAS is already used at several places. At the Twente University of Technology, GAS is used for student training courses but also for the development of new Sea-Of-Gates structures. Another installation of GAS is at Philips Research Labs in Eindhoven, where also Sea-Of-Gates master slices are developed. At both sites, designs made with GAS have been processed successfully. Recently, GAS is installed at Sagantek, a CAD development and IC

design house in Eindhoven. To conclude, the extreme flexibility of GAS is used at the Design Automation Section, where GAS was developed, to implement an arithmetic chip in a standard cell design. The routing capabilities of GAS to deal with three interconnect layers, together with transparent routing through modules were the main reasons for outdating the other available design software.

2. A Gate Array Layout Model

A first step towards general layout design is the definition of an abstract layout description model. The essence of such a model is that it eliminates any direct relation between the described layout patterns and the silicon integration technology, with minimal restrictions imposed on the possible layout patterns that can be captured. Further, a solution for the technology related design rules must be provided, such that the application specific layout patterns are correct by design. The layout pattern descriptions must be compact to minimise the amount of data stored, while the overhead for data retrieval, during the actual layout design, is kept to a minimum.

The layout patterns that must be captured for gate array layout design are the preprocessed and the application specific layout patterns. Because gate array layout design is only concerned with interconnect design, the preprocessed patterns need not to be described exactly. The essential information of the preprocessed patterns for a correct layout design, is in the connectivity relations introduced by these patterns. This implies that all layout patterns can be regarded as connectivity relations and described accordingly. The description of the preprocessed patterns implies that the complete design space must be formulated in terms of this model.

In the next section the grid representation of a design space is introduced which defines an abstract coordinate system and eliminates most of the technology related information. In section 2.2 the space-graph is introduced as the basic layout description model of the design

space. The modeling of elementary wiring patterns and constraints in terms of the space-graph is presented in section 2.3. The description of the application specific interconnections is defined in section 2.4. Section 2.5 describes the abstract modeling of the design rules. The last section of this chapter deals with some implementation aspects of the presented layout model.

2.1 Grid representation

Assume the wiring space of a gate array is defined by a rectangle, in which the application specific interconnections must be designed in an arbitrary number of wiring layers. The positions of these interconnections can be formulated in terms of a grid if the following two conditions are satisfied:

- The preprocessed patterns are regular [Jess84].
- The wire widths and via sizes are not subject of design and are uniform for every single wiring track.

The grid with origin (0,0) and dimensions d_x x d_y is defined by a tuple of horizontal and vertical grid lines:

$$\text{Grid}(d_x, d_y) = (\text{HGL}(d_x), \text{VGL}(d_y)) \quad d_x, d_y \in \mathbb{N} \tag{2.1}$$

with:

$\text{HGL}(d_x) = l_0, l_1, .., l_{d_x-1}$, the sequence of horizontal grid lines, and

$\text{VGL}(d_y) = l_0, l_1, .., l_{d_y-1}$, the sequence of vertical grid lines.

A grid line l_i is associated with a set of grid points, where a grid point is defined as the intersection of a horizontal and a vertical grid line.

Wires are required to run along grid lines and bend at grid points and the center points of vias are required to coincide with grid points, as denoted in figure 2.1.

The link with a specific technology is defined by associating a layout coordinate $c_i \in \mathbb{N}$ and a width-set with every grid line. The layout coordinate denotes the vertical (y)-coordinate for a horizontal grid line l_i

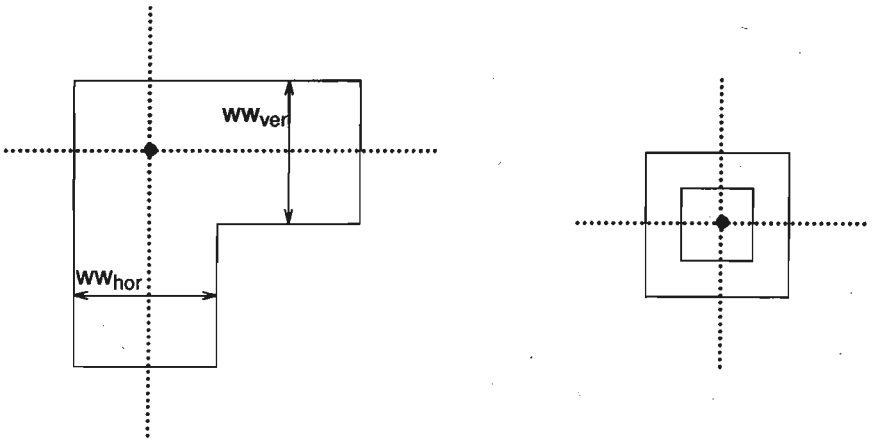


Figure 2.1. Wire segment and via with corresponding grid line definitions.

and the horizontal (x)-coordinate for a vertical grid line.

The width-set associates for every wiring layer the wire widths, denoted by ww, of wires that may run along the grid line and the widths of via overlaps and contact holes, denoted by vo and ch respectively, that may placed at grid points along the grid line:

$$\text{WidthSet}_i = (ww_{i0}, vo_{i0}, ch_{i0}) , .., (ww_{id_z-1}, vo_{id_z-1}, ch_{id_z-1}) \quad (2.2)$$

with d_z the number of wiring layers. As denoted in figure 2.1, ww_{hor} is the wire width associated with a wire running in the y-direction, as is ww_{ver} for a wire running in the x-direction. The same applies to via overlaps and contact hole widths.

The resulting tuple associated with a grid line l_i is denoted by:

$$l_i = (c_i, \text{WidthSet}_i) \quad (2.3)$$

The important complexity reducing choice made is that the WidthSet does not depend on the x-coordinate of the segment of a horizontal grid

line, or the y-coordinate of the segment of a vertical grid line. Another complexity reducing choice is that the grid definition is identical for the different wiring layers, which implies that the layer with the highest grid resolution dictates the grid resolution in the other wiring layers.

It should be stressed that these choices are a priori difficult to motivate. However, these decisions have a posteriori proven to viable after investigating a number of gate array structures. This is one of the major results of this thesis: namely that the complexity reducing choices did not affect the applicability of the concept. To find the right compromise was one of the more heuristic parts of the research.

Note that by associating a layout coordinate c_i with every grid line l_i, the grid lines need not to be equidistant, but are determined by the condition that every possible via and terminal position must be covered by a grid point. This condition defines a lower limit on the distance between two adjacent grid lines. Higher grid resolutions are allowed at the cost of introducing design rules between grid lines. These design rules are necessary to avoid the situation that designed interconnections on adjacent grid lines, which are regarded as unrelated, overlap if the actual grid line distances and interconnect widths are taken into account. Higher grid resolutions also imply that not every grid line is capable to represent a possible wiring track in every wiring layer and not every grid point may represent a possible via or terminal position.

The technology link is not considered during the construction of the application specific interconnections. All relevant information, such as design rules, is formulated in terms of the grid. The technology link is only maintained to provide a consistent transformation from the abstract interconnect descriptions to a correct mask layout definition.

2.2 Wiring space description

Given a grid to represent positions in the wiring space, let the wiring space be described by a space-graph with origin (0,0,0) and dimensions $d_x \times d_y \times d_z$:

$$G(d_x, d_y, d_z) = (V, E) \qquad d_x, d_y, d_z \in \mathbf{N} \tag{2.4}$$

The first two dimensions of G represent the wiring area. The third dimension is determined by the condition that every available wiring layer is represented by a vertex plane, and an extra vertex plane is added for a description of the preprocessed layout patterns.

The vertices of the space-graph coincide with the grid points defined before, and are labeled accordingly:

$$V = \left\{ v_{xyz} \mid 0 \le x < d_x,\ 0 \le y < d_y,\ 0 \le z < d_z \right\} \qquad x, y, z \in \mathbf{N} \tag{2.5}$$

The origin (0,0,0) is defined as the lower left corner of the top wiring plane, where a wiring plane p is defined by:

$$V(p) = \left\{ v_{xyz} \mid 0 \le x < d_x,\ 0 \le y < d_y,\ z = p \right\} \tag{2.6}$$

A set of edges is incident with every vertex, representing the possible wiring directions from a vertex to its adjacent vertices.

For a so-called manhattan-style wiring, six wiring directions exist from every vertex, as shown in figure 2.2. For a 45 degree wiring style, four additional wiring directions, and thus edges, must be incorporated. These edges can be denoted as the north-east, south-east, south-west and north-west edge. In the following the manhattan-style wiring is assumed.

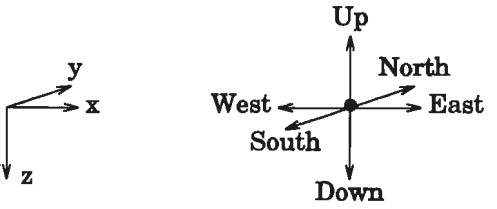


Figure 2.2. Wiring directions associated with every vertex for a manhattan-style wiring.

Let v_i denote the vertex $v_{x_i y_i z_i} \in V$, then the distance between two vertices $v_i, v_j \in V$ is defined as the manhattan distance :

$$d(v_i, v_j) = |x_i - x_j| + |y_i - y_j| + |z_i - z_j| \tag{2.7}$$

If v_i, v_j are adjacent $d(v_i,v_j) = 1$.

Given the wiring style, the edge set E of G is defined by:

$$E = \left\{ \{ v_i, v_j \} \mid v_i, v_j \in V \wedge d(v_i, v_j) = 1 \right\} \tag{2.8}$$

An edge $\{v_i,v_j\}$ of E will be denoted as e.

2.3 Wiring patterns and wiring constraints

Given an edge set E, wiring information is provided by assigning status labels to every edge. The possible edge status labels and interpretations are listed in table 2.1.

edge-status	interpretation
INITIAL	not connected, free wiring direction
INHIBIT	not connected, illegal wiring direction
IMAGE	predefined connection
ROUTER	router made connection

Table 2.1. Enumeration of edge status labels and interpretations.

The assignment of status labels to every edge of E is denoted by an edge labeling function f_s, which is defined as:

$$f_s : E \rightarrow ES, \quad ES = \{ \text{INITIAL, INHIBIT, IMAGE, ROUTER} \} \tag{2.9}$$

The edge labeling function is used to describe wiring patterns and constraints along the principal wiring directions.

A different approach is used for the description of the preprocessed patterns in the bottom plane. Because these patterns are preprocessed, an exact modeling in terms of wires and vias is not required and would impose unnecessary constraints on the possible geometries that can be described.

Further, the number of positions at which these patterns are accessible for wiring purposes, are restricted by preprocessed vias or predefined positions at which programmable vias may be placed.

By definition of the grid, these access positions are covered by grid points. The observation that these grid points can be electrically equivalent, leads to the introduction of an equivalence relation between vertices. Let v_i and v_j denote two vertices in the space-graph. Then v_i and v_j are defined to be electrically equivalent if they represent two positions in the wiring space that are unconditionally on the same potential level in the original grid, that is, this relation is not established in the wiring space by some settings of the edge status labels.

The equivalence relation is denoted as: $v_i \, R_{eq} \, v_j$. An equivalence set of vertices Eq with respect to a vertex v_i, is defined as:

$$Eq(v_i) = \left\{ v \mid v R_{eq} v_i \right\} \tag{2.10}$$

The equivalence relation is symmetrical. Thus every vertex is equivalent with itself. This implies that this relation also is also an equivalence relation in the mathematical sense. An equivalence set with exactly one vertex is called a singleton equivalence set. In general the singleton equivalence sets are omitted for convenience.

Although the equivalence relation is introduced for the description of the preprocessed layout patterns, this is not a limitation. In general, in any wiring plane the equivalence relation is used to describe fixed connections that are not described by an appropriate labeling function.

For a space-graph G, the set of equivalence sets is defined as:

$$EQ = \{ Eq_1, Eq_2, .., Eq_n \} \tag{2.11}$$

assuming that n such sets exist.

2.4 Net modeling

Electrical connections in the space-graph are described by WiringSets. For a given space-graph G_s, a WiringSet is defined as a sub-graph:

$$\text{WiringSet} = G_w = (V_w, E_w) \tag{2.12}$$

which is the union of components called "wiring patterns". Any wiring pattern is a maximal connected sub-graph of G_w. A graph G is connected if there is a path of edges labeled IMAGE (in the case of a predefined connection, not made by the router) or ROUTER (in the case of connections made by the router) between any pair of vertices in V_w. A path P is defined as a sub-graph of G_w:

$$P = (X, U) \tag{2.13}$$

with

$$X = \{ v_0, v_1, .., v_n \} , \tag{2.14}$$

an ordered set of pairwise distinct vertices of V_w, and

$$U = \{ e_1, e_2, .., e_n \} , \tag{2.15}$$

an ordered set of pairwise distinct edges of E_w, satisfying the following two conditions:

$$\forall_{i \in \{1,2,..,n\}} \left[e_i = \{ v_{i-1}, v_i \} \right] \tag{2.16}$$

$$\forall_{v_i, v_j \in V_w} \left[v_i \neq v_j \Leftrightarrow \neg (v_i R_{eq} v_j) \right] \tag{2.17}$$

The last condition implies that a path contains no loops due to some equivalence relations between the vertices of the path.

The wiring patterns of G_w can be linked together by equivalences.

A "net" is defined as a set of unconnected WiringSets. A net with one wiring set is defined as "complete". Before any routing procedure has been applied, a net is assumed to have more than one wiring set. These wiring sets are then connected by router made connections.

In order to distinguish the vertices of the space-graph belonging to different nets, a net-number is associated with every vertex.

Two special values are used to indicate that a vertex does not belong to any net and is available for future use, or to indicate that a vertex is blocked for some reason.

This is denoted by the labeling function f_{net}:

$$f_{net} = V \rightarrow NN, \quad NN = \left\{ FREE, BLOCKED, 1, 2, .., \#nets \right\} \tag{2.18}$$

The vertices of the different wiring sets of the same net are also distinguished by associating a wiring-set-number with every vertex.

In this way, the creation of loops by a routing procedure can be detected by inspection of the wiring-set-number of a vertex in the space-graph.

The labeling function is denoted by f_{wset} and defined by:

$$f_{wset} = V \rightarrow \mathbb{N} \tag{2.19}$$

The initial wiring sets of a net are called the terminal wiring sets. The task of a routing procedure is to construct paths in the space-graph that connect these terminal wiring sets.

In order to distinguish the terminal wiring sets from the constructed wiring paths, the vertices of a net are marked, which is denoted by the labeling function f_{term}:

$$f_{term} = V \rightarrow B, \quad B = \left\{ 0, 1 \right\} \tag{2.20}$$

The value "1" is used for the vertices of the terminal wiring sets, the value "0" for vertices of constructed wiring paths.

Concluding, a net is completely defined by the following 4-tuple:

$$Net = (WiringSets, f_{net}, f_{wset}, f_{term}) \tag{2.21}$$

with:

$$\text{WiringSets} = \left\{ \text{WiringSet}_1, .., \text{WiringSet}_n \right\} \tag{2.22}$$

the set of unconnected wiring sets associated with the net.

2.5 Design rule modeling

For a description of design rules in terms of the layout model, the term design rule is interpreted rather literally as defining a set of guidance rules for a routing procedure. The rules indicate preferences or limitations.

The preference rules often indicate preferred wiring directions and are described by means of a cost function f_{cost}.

The cost labels are associated with edges:

$$f_{cost} = E \rightarrow \mathbb{N} \tag{2.23}$$

The so-called "critical" design rules may never be violated during wiring construction. These design rules define illegal wiring patterns. Given the fact that wiring patterns and vias are both denoted by edges, any illegal wiring pattern can be described by sets of edges exhibiting a connection status. All design rules can thus be formulated as sets of edges sets, where every edge set denotes an illegal wiring pattern.

However, for the router, a more convenient way is to formulate design rules with respect to a reference edge e_a, where a basic wiring action is taken. A wiring action on e_a changes the status label of e_a from INITIAL to ROUTER. Now, a design rule is said to "shadow" the wiring action on e_a by a "shadowing" set of edges, denoted as $Sh(e_a)$, if $\{e_a\} \cup Sh(e_a)$ denotes an illegal wiring pattern if all edges exhibit a connection status.

In this concept, the router can determine whether or not a wiring action on an edge e_a is allowed by inspection of the edge status labels of the edges in the shadowing set $Sh(e_a)$ as follows:

$\forall_{e \in Sh(e_a)} \left[f_s(e) = IMAGE \vee f_s(e) = ROUTER \right] \Rightarrow$ wiring not allowed

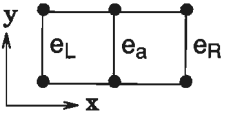
and:

$\exists_{e \in Sh(e_a)} \left[f_s(e) \neq IMAGE \wedge f_s(e) \neq ROUTER \right] \Rightarrow$ wiring allowed

In general, one design rule may introduce multiple illegal wiring patterns. Two example design rules are given below.

Example 2.1 :

Assume a design rule, specifying that wires may not be placed at adjacent grid lines. The typical situation where this rule applies to is denoted below:



With respect to edge e_a, the design rule specifies two wiring patterns that are illegal. One pattern is defined by the two edges e_a and e_L exhibiting both a connection status, the other analogous for e_a and e_R. This design rule leads to the following sets $Sh(e_a)$:

$Sh_1 = \{ e_L \}$

$Sh_2 = \{ e_R \}$

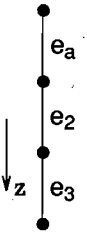
and the complete design rule for e_a is given by:

$$DR(e_a) = \left\{ Sh_1, Sh_2 \right\}$$

Example 2.2 :

The second example is a design rule, specifying that a combination of three stacked vias is not allowed. Any

combination of two vias above each other is correct. A typical situation is denoted below:



This design rule leads to one set $Sh(e_a)$:

$$Sh(e_a) = \{ e_2, e_3 \}$$

and the complete design rule for e_a is given by:

$$DR(e_a) = \left\{ Sh(e_a) \right\}$$

An analogous design rule can be formulated with respect to e_2 and e_3.

As shown above, together with the reference edge e_a, every set $Sh(e_a)$ enumerates an illegal wiring pattern. This is convenient in case more than one design rule is formulated for the same edge. All the sets $Sh(e_a)$ of all the design rules for an edge e_a can be stored in a single set, thus defining the design rules associated with the edge e_a:

$$Dr(e_a) = \left\{ Sh_1, Sh_2, .., Sh_n \right\} \tag{2.24}$$

The set of all design rules for a space-graph is denoted by:

$$DR = \left\{ Dr_1, Dr_2, .., Dr_n \right\} \tag{2.25}$$

Given the definition of the design rule sets, a design rule function f_{dr}, that operates on these sets, is defined to indicate for every edge if a

wiring action on that edge is allowed or not:

$$f_{dr} : E \rightarrow B, \quad B = \{ 0, 1 \} \tag{2.26}$$

and consequently:

$$f_{dr}(e_w) = 0 \Leftrightarrow \exists_{Sh \in Dr(e_w)} \left[\forall_{e \in Sh} \left[f_s(e) = \text{IMAGE} \vee f_s(e) = \text{ROUTER} \right] \right]$$

$$f_{dr}(e_w) = 1 \Leftrightarrow \forall_{Sh \in Dr(e_w)} \left[\exists_{e \in Sh} \left[f_s(e) \neq \text{IMAGE} \wedge f_s(e) \neq \text{ROUTER} \right] \right]$$

2.6 Implementation aspects

The implementation of the modeling concepts, presented in the previous sections, is critical in the sense that a trade off must be found between minimising both the amount of data required and the overhead for data access. The most important structure in this context is the space-graph together with the design rule and equivalence sets. In principle, the space-graph and the sets describe the complete design space and should therefore require a minimal amount of data. On the other hand, the description of the design space is used during the detailed interconnect design and should therefore be accessed with a minimal overhead. Only the manhattan-style layout design is considered in the current implementation.

For a given master slice, the maximal design space is fixed. This implies that also the space-graph description of the design space is fixed and therefore the vertex and edges sets of the space-graph.

The overhead to access the vertex set of a space-graph is minimised by implementing the vertex set as a 3-dimensional array of vertices with dimensions d_x x d_y x d_z. Every array element, or vertex, consists of a data record in which the labels associated with a vertex are stored. Vertices are thus identified by the three carthesian coordinates:

$$v = (x, y, z)\ ,\ x, y, ,z \in \mathbb{N}\ , \quad \begin{array}{l} 0 \le x < d_x, \\ 0 \le y < d_y, \\ 0 \le z < d_z \end{array} \tag{2.27}$$

The presentation of the outline of the data record is deferred until the implementation of the edge cost labels, the equivalence sets and the design rule sets is discussed.

Because the edges associated with every vertex are fixed for a given layout design style, these edges are not implemented explicitly, only the edge labels are stored. In the current context, it is sufficient to associate the labels of three edges with every vertex, which is shown by:

$$\begin{aligned} |E| &= (d_x - 1)d_y d_z + (d_y - 1)d_x d_z + (d_z - 1)d_x d_y \\ &< 3d_x d_y d_z = 3|V| \end{aligned}$$

The three edge status labels stored with every vertex are the North, East and Down edge labels, and are identified by:

$$f_s(e) = f_s(x,y,z,d)\ ,\ d \in \{\text{North, East, Down}\} \tag{2.28}$$

As an example for the other three directions, the edge status label for the South-direction is defined as follows:

$$f_s(\text{South}) = \begin{cases} f_s(x,y-1,z,\text{North}) & \text{if } (x,y-1,z) \in V \\ \text{INHIBIT} & \text{otherwise} \end{cases} \tag{2.29}$$

The analogous holds for the West- and Up-directions.

In the following the short notation e_v is used to denote an arbitrary edge associated with a vertex v, and e_N, e_E and e_D are used to denote respectively the North, East and Down edges, if the edges or edge labels are regarded independently of the vertices.

For implementation, a wiring pattern is decomposed into a set of paths. The paths are further decomposed until all edges in any path have the edge status label IMAGE or ROUTER. For every wiring pattern, two distinct lists are maintained, one for paths with the IMAGE label, and one for paths with the ROUTER label. In this way, these labels need not to be stored explicitly. The paths are implemented as an ordered list of vertices of the path. Data reduction is straightforward by storing only the start end end points, as well as the positions where the path changes direction. With this implementation, the distinction between wires in a plane and vias is made implicitly by two consecutive points in the vertex list.

Note that in this way paths with the other two edge status labels can also be stored, which is used in the case of adjustment of edge status labels in "stamps", as discussed in the next chapter.

Efficient implementation of the edge cost labels, the equivalence sets and design rule sets is obtained by using an indirect addressing scheme. The essential observation leading to significant savings in store is that all three data items appear in the form of a few characteristic patterns that are repeated many times over the total area of the gate array. So those patterns are stored in a table of relatively small size. At the vertices of the grid only the identifiers of those patterns are stored. The method requires an offset mechanism for all three data items. For instance while dealing with the edge cost labels, triples of values of the north, east and down edge are stored, assuming that only very few of those triples exist. At any vertex a pointer to that particular triple of edge cost values valid for that vertex is stored. The first reduction is that only one value is stored at a vertex compared to three values if all cost labels are stored. The second reduction is obtained by elimination of table entries having equal cost labels for the three edges. The access is efficient by implementing the table as a one dimensional array, as denoted in figure 2.3.

A similar approach is used to implement the equivalence sets. Because vertices are identified by tuples (x,y,z), an equivalence set is implemented as a circular list of these tuples. This allows to store these sets as offsets between the equivalent vertices, rather then using absolute coordinates. The details are explained with the aid of the following example:

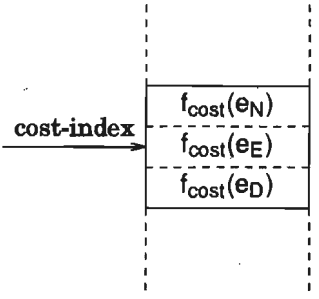
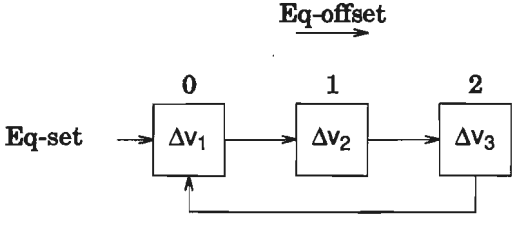


Figure 2.3. Implementation of cost table entry.

Example 2.3 :

Assume an equivalence set of three vertices: v_1, v_2 and v_3. The three vertices are given as vectors of the three principal carthesian coordinates x, y and z. This set is stored as:



with Δv_1, Δv_2 and Δv_3 defined by:

$$\Delta v_1 = (v_2 - v_1)$$

$$\Delta v_2 = (v_3 - v_2)$$

$$\Delta v_3 = (v_1 - v_3)$$

The two equivalent vertices ev_1 and ev_2 with respect to v_1 are given by:

$$ev_1 = v_2 = v_1 + \Delta v_1$$
$$ev_2 = v_3 = ev_1 + \Delta v_2$$

The analogous equivalence set with respect to v_2 starts at offset 1:

$$ev_1 = v_3 = v_2 + \Delta v_2$$
$$ev_2 = v_1 = ev_1 + \Delta v_3$$

From the example follows that for a minimal overhead two entry points must be provided with every equivalence set: one to access the complete equivalence set and one to access the correct offset position within the set. The table implementation is thus a 2-dimensional array, with the first dimension determined by the total amount of equivalence sets defined and the second dimension determined by the size of the corresponding equivalence set.

From the example described above it will be clear that the order of the stored vertex offsets is essential. Data reduction is obtained by elimination of sets exhibiting equal vertex offsets. An equivalence set is associated with a vertex by storing the corresponding table index and set-offset at the vertex.

Note that by this implementation only one equivalence set may be associated with any single vertex.

To conclude the discussion of the implementation of the equivalence sets, the following example is given.

Example 2.4 :

Consider the example layout pattern of figure 2.4 and the corresponding space-graph representation denoted in figure 2.5.

It can easily be verified that 49 vertices have an associated equivalence set following from definition 2.10. The number of equivalence sets if absolute coordinates where used is 22. In the current implementation only 5 equivalence sets need to be

stored, shown in figure 2.6.

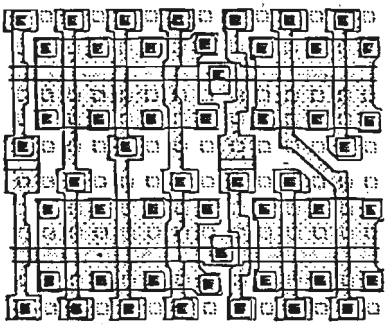


Figure 2.4. Example layout pattern.

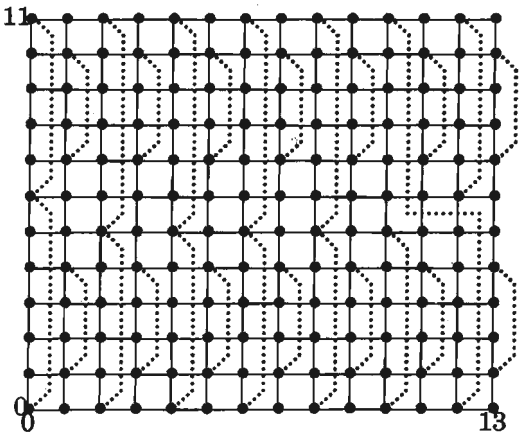


Figure 2.5. Wiring graph representation of the layout pattern shown above. The equivalence relations between the vertices are denoted by dotted lines.

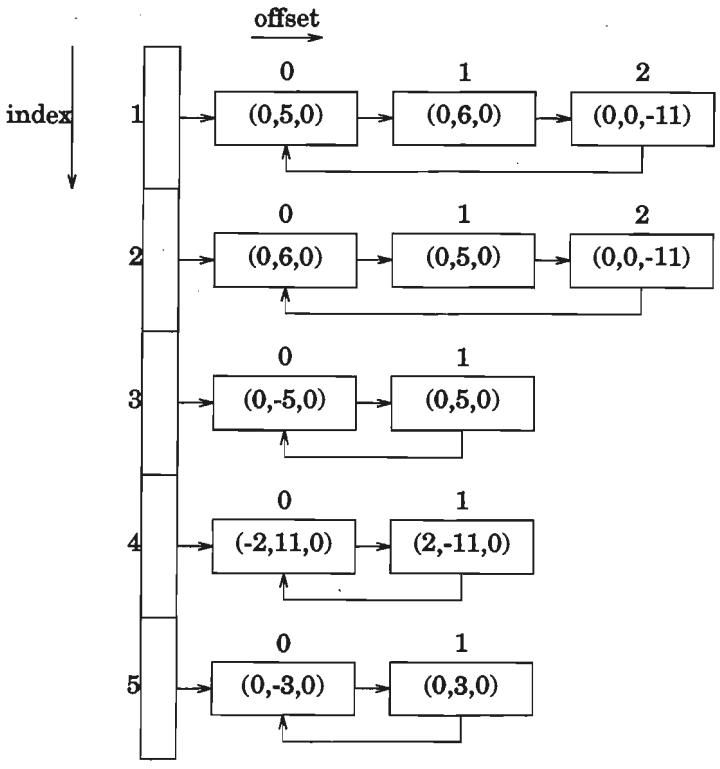


Figure 2.6. Equivalence table resulting for the space-graph in figure 2.5.

The design rule sets are also implemented by using a table. Every table entry denotes the design rule sets $Dr(e_N)$, $Dr(e_E)$ and $Dr(e_D)$, as indicated in figure 2.7. The design rule sets $Dr(e_v)$ are simply a list of the edge sets $Sh(e_v)$. The implementation of an edge set Sh(e) is further similar to the equivalence sets in that again relative coordinates are used. This has the same advantages as for the equivalence sets, namely that the number of edge sets stored can be minimised.

Figure 2.8 shows how the design rules of examples 2.1 and 2.2 are stored in a table entry.

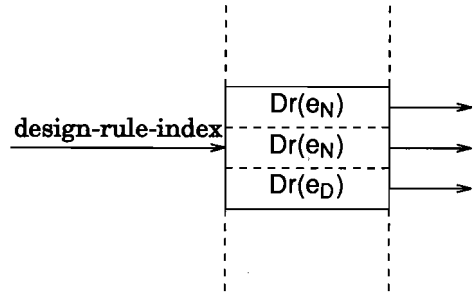


Figure 2.7. Implementation of design rule table entry.

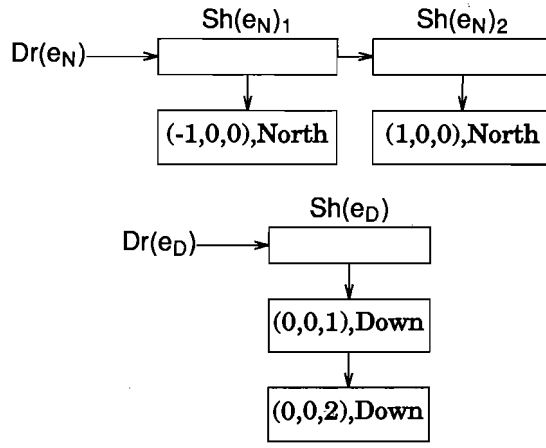


Figure 2.8. Table entries for design rules of example 2.1 (upper) and example 2.2 (lower).

As with the edge cost labels, a table index is stored at a vertex to associate the corresponding design rule with the three principal edges associated with the vertex.

$f_s(e_N)$	2 bits
$f_s(e_E)$	2 bits
$f_s(e_D)$	2 bits
$f_{net}(v)$	14 bits
$f_{wset}(v)$	8 bits
$f_{term}(v)$	1 bit
cost-index	8 bits
eq-index	8 bits
eq-offset	6 bits
dr-index	8 bits
router-data	5 bits

Figure 2.9. Implementation of compacted vertex data record.

With the discussion of the implementation of the design rules sets, the principal values stored in a vertex data record are presented. The size of this record is reduced by a combination of limited value ranges and the application of a compacted data record. The final result is shown in figure 2.9. The field "router-data" is reserved for values that might be used by a routing procedure.

3. Gate Array & Design Characterisation

In the previous chapter the gate array layout model was introduced as the first step towards general layout design. The second step is to formalise descriptions of gate arrays and designs in terms of the layout model.

The formal descriptions, introduced in this chapter, provide maximal flexibility by capturing all relevant data at the most detailed level. In this way no restrictions are introduced limiting the application to gate array types. Further, no preference is incorporated for specific design algorithms.

The amount of data required for descriptions of gate array types and designs is controlled by using hierarchy and repetition. The application of a 2-level hierarchical description of the master slice, accounts for an amount of data that depends on the complexity of the preprocessed structures, rather than the actual size of the master slice. Further, predefined and application specific patterns are also stored hierarchically.

The advantage of a formal design description is in the uniformity of the stored data. The design is regarded as the top hierarchical level, the master slice as the bottom. Intermediate hierarchical levels are provided by the predefined wiring patterns stored in the library, usually provided with every gate array. An additional advantage is that completed designs can easily be added to a library, thus allowing hierarchical design.

The most intensive data access is defined at the detailed routing level, where the most detailed information is required for the interconnect design. This requires that the space-graph representation must be accessible at the top hierarchical level. The overhead in the space-graph access is minimised by providing a space-graph expansion procedure that constructs the complete space-graph at the top level for a requested region on the master slice.

The formal description of a gate array is presented in the following two sections. In section 3.1 the description of the master slice is presented. The formulation of the gate array data is completed by the description of the gate array macro library in section 3.2. The formal descriptions are concluded with the characterisation of the design data in section 3.3. The last section of this chapter discusses some implementation aspects, amongst others the space-graph expansion procedure mentioned above.

3.1 The master slice

The master slice defines the initial wiring space. This wiring space is usually partitioned into an internal area and a peripheral area. The internal area, also referred to as core area, is highly regular, and can easily be described in terms of the space-graph. The peripheral area consists of bonding pads, for off-chip connections, and input/output buffers to protect the chip's circuitry. By its nature, this area is not easy to describe by a space-graph. If connections to the peripheral circuits are subject of a design, a minimal requirement is that the terminals of the peripheral circuits are placed at grid points. The space-graph description of the core area must be extended to incorporate these grid points.

Besides the initial wiring space, the master slice description may also involve predefined nets. These nets are not subject of the layout design, but must be taken into account by every design made on the master slice.

By convention, predefined nets are complete, independent whether this is reflected in the net description or not. This allows to take into account connections that are realised outside the wiring space described by the master slice space-graph. The edge status label for

the edges that belong to the predefined nets has the value IMAGE.

Typical predefined nets on the master slice are the power nets, but also global clock and reset nets may be predefined.

A straightforward approach to describe the master slice wiring space is by a space-graph together with the required labeling functions f_s, f_{cost} and sets EQ and DR. The predefined nets can be inserted in the space-graph by setting the appropriate vertex and edge labels.

The major disadvantage of this approach is that the size of the space-graph is directly related to the wiring space dimensions. The following example illustrates this.

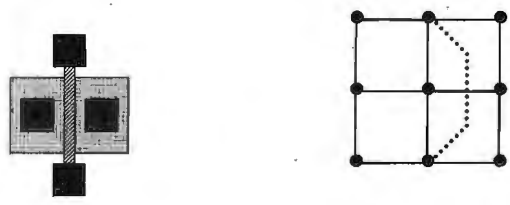


Figure 3.1. Transistor layout and corresponding space-graph representation for the bottom vertex plane. The dotted line indicates the equivalence relation between the two vertices representing the gate connections of the transistor.

Example 3.1 :

Consider the simple transistor as shown in figure 3.1. The space-graph description of this transistor requires 9 vertices for every wiring plane. With the increasing integration scale, gate arrays containing over one million transistors are possible. This leads to 9 million vertices per vertex plane, and 27 million vertices in case of two wiring layers, including the extra plane for the description of the preprocessed patterns.

Obviously, the data requirements can not be kept within acceptable limits by using this straightforward approach. A more intelligent way

to describe the master slice, by using its structure, is now presented.

Inspection of a wide range of gate arrays leads to the observation that repetition is the key notion for most gate array master slices. Usually, the core area of the master slice consists of an arrangement of a very small set of different layout patterns, the core cells. The core cells represent areas that are small compared to the complete core area.

This observation leads to the description of the master slice by a floorplan. The floorplan provides a space-graph description of the master slice that is partitioned into the space-graphs of the core cells.

The core cells are thus described by their space-graph representation, together with the labeling functions f_s, f_{cost} and the sets EQ and DR, forming the following 5-tuple:

$$cc = (G, f_s, f_{cost}, EQ, DR) \tag{3.1}$$

Every different pattern, defining a part of the wiring space, is defined as a separate core cell, and described accordingly.

This holds especially for the peripheral areas, which are also regarded as core cell areas.

Now, an arrangement of a specific core cell on the master slice is described by a repetition of instances of that core cell.

An easy way to describe a repetition is by means of a translation in a plane. A translation in two dimensions is completely specified by the following 6-tuple:

$$f_T = (from_x, step_x, to_x, from_y, step_y, to_y) \tag{3.2}$$

$$from_x, step_x, to_x, from_y, step_y, to_y \in \mathbb{N}$$

Application of this translation function to a core cell, gives the (x, y) positions of the instances of that core cell in the plane:

$$x = from_x + i_x step_x, \quad i_x = 0, 1, .., (to_x - from_x) / step_x \tag{3.3}$$
$$y = from_y + i_y step_y, \quad i_y = 0, 1, .., (to_y - from_y) / step_y \tag{3.4}$$

In general, more than one translation function may be associated with a core cell. These functions define the translation set for the core cell:

$$FT = \left\{ f_{T_0}, f_{T_1}, ..., f_{T_n} \right\} \tag{3.5}$$

With the definitions given above, the description of the complete floorplan of the gate array master slice is defined by:

$$FloorPlan = \left\{ (cc_0, FT_0), (cc_1, FT_1), ..., (cc_n, FT_n) \right\} \tag{3.6}$$

It is important to note that, in this concept, the floorplan description of the wiring space is required to be a complete cover of the wiring space. This implies that every vertex of the explicit space-graph description of the complete wiring space, belongs to exactly one instantiation of a core cell. Stated in other words, the floorplan description leaves no "holes" in the wiring space description, and the core cells do not overlap at any position in the wiring space. So, no vertex is left undefined and no vertex is defined more than once.

Returning to the previous example, the same master slice is now described by one core cell, requiring 27 vertices, and one translation function, requiring 6 values. The reduction in the amount of data is obvious. Note that, by definition of the translation function, this amount of data in independent of the size of the master slice.

An important consequence of the decomposition of the space-graph into its floorplan is that not all edge labels are covered by the core cell descriptions. More specific, all edge labels between two adjacent core cells are undefined in the floorplan description, while they would be defined in the explicit space-graph representation of the wiring space. Therefore, the floorplan description of the wiring space can not be complete with respect to the edge set E.

The consequence of this observation is that the wiring space description of the master slice must be completed by the inclusion of the overall labeling functions f_s and f_{cost}. Further, the sets EQ and DR must be taken into account and the predefined nets, which gives the complete space-graph description of the master slice:

$$\text{MasterSlice} = (\text{Nets}, \text{FloorPlan}, f_s, f_{cost}, \text{EQ}, \text{DR}) \tag{3.7}$$

3.2 The macro library

Usually, together with a gate array, a library of elementary, predesigned macros is provided. The functions performed by the macros can range in complexity from a simple inverter to a master-slave flipflop, or even more complex functions such as multiplexers. The macro library is denoted as a set of macros:

$$\text{MacroLib} = \left\{ \text{Macro}_1, \text{Macro}_2, .., \text{Macro}_n \right\} \tag{3.8}$$

Given a master slice pattern, the function of a macro is defined by its interconnection pattern. When mapped onto the master slice at an appropriate location, this pattern creates a circuit that performs the desired function at its terminal pins. The positions where a pattern may be placed, are called the "legal positions" of that pattern.
More than one wiring pattern can be given, performing the same function, yet with different terminal positions, aspect ratios, orientations or legal positions on the master slice. The different wiring patterns that implement the same macro-function are called the "stamps" of the macro. A macro is thus denoted by the following tuple:

$$\text{Macro} = (\text{Pins}, \text{Stamps}) \tag{3.9}$$

The essential information of a stamp, for placement and routing, is in the positions and the signals assigned to the pins, the wiring patterns of the stamp, and the legal positions of the stamp.

The wiring space occupied by a stamp is limited by a bounding box, denoting the shape of the stamp. This wiring space can be described by a space-graph G, with x- and y-dimensions equal to the shape of the stamp. The z-dimension is given by the number of wiring layers available. In general, the amount of data required for the stamp can be reduced by taking into account that the internal area of the stamp is strongly related to the master slice layout patterns. If the underlying master slice patterns are identical for all the legal positions of the stamp, the stamp can be described by a sub-graph G_S of G.

The wiring patterns of a stamp are described by nets. The nets can be partitioned into terminal nets and internal nets. The terminal nets of the stamp are related to the pins of the macro, and are used for interconnections at higher hierarchical levels. The internal nets of the stamp define those nets required to achieve its functionality.

A stamp is allowed to contain modules. These modules are instances of other functions in the library. Every library is required to have macros with stamps containing no modules. A module is thus specified by the selection of a macro, a stamp associated with that macro, the nets connected to the pins of the macro, and the coordinates of the stamp within the area of the calling stamp.

The nets (or signals) assigned to the pins of the macro are required to be a sub-set of the nets of the calling stamp. Because modules refer to functions contained in the library, a module is described by references to these functions, rather then a full function description:

$$\text{Module} = (\text{ Macro–call, Stamp–call, (x, y), PinSignals }) \tag{3.10}$$

A very compact description of the legal positions of a stamp can be given by using the translation functions introduced for the definition of the floorplan. The set of legal positions of a stamp is thus denoted in the same way as the translation set for a core cell:

$$\text{Legals} = \left\{ f_{T_0}, f_{T_1}, \ldots, f_{T_n} \right\} \tag{3.11}$$

As mentioned before, the space-graph description of the stamp consists of a sub-graph of the space-graph for the internal area of the stamp. This sub-graph is already partly introduced by the definition of the nets of the stamp, which are in fact sub-graphs.

The main importance of using sub-graphs, is that labeling values of edges at any hierarchical level are always overruled by higher hierarchical levels. This increases the flexibility of the description, where labels may be adjusted to take into account the effects of wiring patterns at higher hierarchical levels.

An example of this effect concerns the cost labels assigned to edges. Due to some specific wiring patterns of a stamp, it can be advantageous to change the cost labels, defined at the master slice level, of some edges internal to the stamp, to give a router procedure specific guidance within the area.

This concept can be further extended to the status function in general, but also to the sets EQ and DR.

For example, if a stamp contains connections that are not described by wires, these connections can be described by equivalence sets. This situation occurred for the stamps of the HDGA gate array, which is described in more detail in chapter 6, were internal stamp connections are realised with straps. Because for this gate array the width of the straps is half the width of an ordinary wire, the choice was to describe the straps as equivalence sets rather than using a higher grid resolution, at the cost of more complicated gate array description, without gaining additional wiring space. A requirement concerning the EQ and DR sets is, that they present extensions to, or merge, the already given sets for the master slice. Reduction on previously specified sets is not allowed. A reduction of an equivalence set would imply that a previously defined connection between vertices is eliminated by placement of a stamp. Because equivalences are normally used to describe preprocessed patterns, elimination of these patterns will never occur in practice. The reduction of a design rule set at a higher hierarchical level would imply that, due to a specific wiring pattern in a stamp, an illegal pattern defined at a lower level would not be illegal any more, which is also assumed to have no practical meaning.

The complete definition of a stamp is denoted below, by the following 8-tuple:

$$\text{Stamp} = (\text{Shape, Nets, Modules, Legals, } f_s, f_{cost}, \text{EQ, DR}) \quad (3.12)$$

A last issue concerns the level of detail of the stamp description. It is by no means required to describe stamps as detailed as above.

The idea of the stamp definition is to provide a stamp description model containing all essential information for placement and routing. The minimal description of a stamp is given by the shape and the legal positions for placement, and, for routing, the position of the stamp's terminals. This indicates a choice, concerning the level of detail used, to describe a stamp.

If details, concerning the stamp's internal structure, are described, a routing procedure can use this information to construct wiring patterns through the stamp's internal area. A stamp specified in this way is thus transparent for routing.

However, a stamp may also be described by a black box with terminals on the boundaries. In this case, routing through the stamp is prevented by an appropriate labeling of the internal edges of the stamp.

Any form of detailed description between the two extrema mentioned above is also possible.

3.3 Design characterisation

Together with a master slice, and a macro library, a design is regarded as the instance of a layout problem to be solved. It consists at least of a set of modules and a net-list. Additional information, such as net weights, can be supplied to guide the layout design. The design specification is very closely related to the macro and stamp descriptions. A design is regarded as a macro definition, and the design implementation reflects the nature of a stamp:

Design = (Pins, Net–weights, Implementation) (3.13)

A rectangular window is associated with every design implementation that bounds the area in which the design must fit. This window defines the shape of the design implementation.

The nets of a design are associated with the pins of the modules of the design. The main difference between design nets and the nets of the stamps is, that the design nets are not complete. Before a routing procedure is applied, the wiring sets of the nets only consists of their terminal wire sets. These sets are the wiring patterns of the terminal wires of the stamp to which the net is connected. The wiring patterns constructed by a router, to connect the terminal wires, are labeled with the value ROUTER, in order to distinguish them from the predefined wiring patterns.

The modules of the design are defined according to the definition given previously. Before a placement procedure is applied, the module consists only of the macro-call. It is the task of the placement procedure to determine, for every module, an optimal stamp and legal position of the stamp.

This results in the following implementation definition:

Implementation = (Window, Nets, Modules) (3.14)

A given macro library can be extended with a completed design in the following way. A function must be associated with a design, and the pins of the design define the pins of the associated macro. The design implementation defines a stamp definition for the macro.

The design implementation must be transformed into a stamp definition. Therefore, the net labels are changed from ROUTER to IMAGE. The labeling functions f_s and f_{cost} are taken to be trivial functions, and the sets EQ and DR are empty.

In this way, hierarchical levels are added by incorporating completed designs as macros in the library, and use these macros in subsequent

designs.

3.4 Implementation aspects

The implementation of the gate array and design descriptions presented in the previous sections is based on the stamp definition 3.12. The importance of the stamp structure is determined by the fact that part of the master slice and design data can be arranged in terms of a stamp.

The master slice structure is partioned into its floorplan and the remaining data. The implementation of the floorplan is straightforward according to definition 3.6. The core cells are implemented as 3-dimensional arrays. Note that, with the chosen implementation of the vertices, the edge labels at the boundaries can be stored within the core cells, and are not required to be kept explicit at the master slice level. The remaining data is stored in an additional stamp structure, the so-called master stamp. The size of this stamp equals the size of the master slice.

The design data is partioned into the incomplete nets, which are stored separately, and the fixed data, which consists of the placed modules of the design and the completed nets. The incomplete nets must be stored separately, in order to keep track of the different wiring sets. Completed nets on the contrary, can be stored more efficiently because they are complete like the nets associated with a stamp. This fixed design data can thus be stored like a stamp. The shape of the stamp is the supplied design window, the modules of the design stamp are the instances of the library macros used in the design.

The uniformity in data structure obtained in this way accounts for an efficient implementation of the complete data structure and limits the number of procedures required to handle the stored data.

The implemented stamp structure is shown in figure 3.2.

The order of nets stored in the stamp is determined by the terminal definition of the macro to stamps belongs to. In this way the difference between terminal nets and internal nets of the stamp is made implicit. The data field "Default–labels" represents edge labels that are uniform

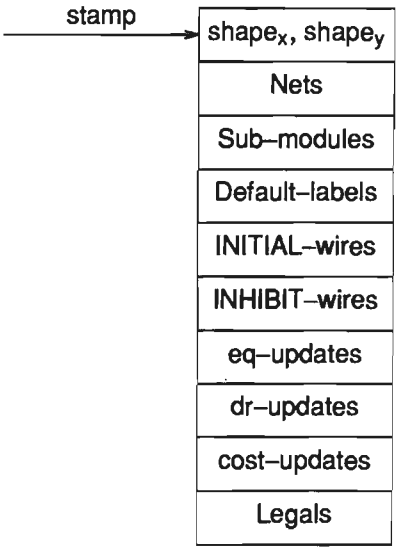


Figure 3.2. Implementation of the stamp data structure.

in a single vertex plane, thus reducing the amount of data required to store the edge labeling function f_s. The description of f_s is completed by definition of the INITIAL- and INHIBIT-wires. The updates for equivalences, design rules and cost represent the union of these sets for the first two, and the overruling of the cost labels in the last case. An update is identified by the coordinates of the vertex and the new index and/or offset to be associated with that vertex.

The most important action with a stamp is the placing of the stamp into the space-graph. A stamp is placed at a correct legal position and the labels f_{net} and f_{wset} are provided for a correct labeling of the vertices of the terminal nets of the stamp. The order in which the stamp's data is mapped in the graph is not trivial. Assume a space-graph and a stamp. The actual placing of the stamp is defined by five actions:

1) Setting the default labels for all edges in the stamp's area. This action affects all the edge labels in the stamp's internal area and must therefore be done before interconnections are inserted.
2) Placing of the sub-modules of the stamp.
3) Placing of the INITIAL- and INHIBIT-wires. Additional edge labels,

not representing interconnections are also mapped before the nets are inserted. This allows for efficient storage of these wires, and allow overruling of the labels by the subsequent insertion of the nets. By convention, the edge labels may not destroy the interconnection patterns of the sub-modules mapped before.

4) Placing of the updates. Essential is that the new equivalence relations are inserted in the space-graph before the nets are inserted. The reason is that it is assumed that the wiring patterns of the nets are defined according to the newly defined equivalence relations, which are thus essential for a correct propagation of the net signals. The placing of the design rule and cost updates at this point is not essential.
5) Inserting the nets. The nets of the stamp are inserted last. The net and wiring set labels assigned to the vertices of the nets are provided externally. Propagation of the nets accounts for the correct vertex labels of the nets in the previously placed submodules. In this way, all the interconnections of the terminal nets of the stamp get the provided net and wiring set labels and are recognised as belonging to the same net and wiring set. All the net labels of the vertices of the internal nets of the stamp are marked as BLOCKED.

Given the data organisation and placing procedure of the stamp, the most essential actions of the space-graph construction are presented. Given a rectangular region for which the space-graph must be constructed, the first action is to copy the data of the core cells into the requested space. Next the design stamp, which defines the top hierarchical level is placed. By recursion, this completes in principle the construction of the space-graph, only the incomplete nets must be inserted afterwards.

A final remark concerns the relation between the shape of the constructed space-graph and the shape of the requested region. Assume a rectangular region, defined by two corner coordinates as shown in figure 3.3. Due to the vertex offsets used for equivalence relations and design rules, vertices and edges outside the window may be referenced. In order to minimise the overhead in access for these external references, the actually constructed region corresponds with the expanded window, as indicated by the dotted lines. The expansions in the x- and y-directions are easily determined by inspection of the offsets stored in the equivalence and design rule tables.

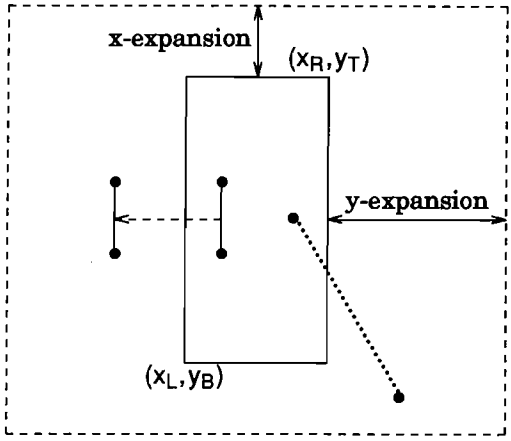


Figure 3.3. Region expansion required to solve external region references for equivalence relations (denoted by the dotted line) and design rule evaluation (denoted by the dashed arrow).

The importance of the space-graph construction is in the relation with the detailed routing, which is discussed in section 5.2. If the detailed routing solves routing problems in small areas, only a small portion of the space-graph needs to be present to minimise the overhead in data access. This portion can be constructed once, before the actual routing procedure is applied for the first time on the area. After the routing within the area is completed, the newly added routing patterns within the routing area are extracted and stored in the wiring list of the respective net. The current space-graph is not referred to any more, and can be eliminated. In this way, the space-graphs for the subsequent areas are constructed one after another, so that, at any time during the routing process, exactly one, fully expanded, portion of the space-graph is present. The amount of memory used for the space-graph is thus controlled by the maximum size of the routing region used by the detailed router. The complete space-graph is never built during the detailed routing.

4. A Placement Strategy

In this chapter a placement strategy, based on the gate array and design descriptions defined in the previous chapter, is presented. The claim is that the presented placement strategy is general in that it applies to all placement problems defined by appropriate gate array and design descriptions. This claim is justified by the fact that no restrictions are introduced that limit the generality provided by the gate array and design descriptions.

Although the placement problem has gained a lot of attention in literature, most strategies proposed are not appropriate to solve the problem in a general context. The main reason for this shortcoming is that the restrictions that occur in a general gate array environment, are not taken into account. These restrictions are the limited area in which a design must fit, in relation with the limited number of different stamp shapes that can be used to implement a module, and the limited set of legal positions associated with a stamp.

Incorporation of the restrictions, mentioned above, in a placement algorithm results in a significant performance loss due to the increased problem size. The solution, presented in this chapter, is to compute a placement in two stages. In the first stage, based on a simplified module description, well known strategies are applied, requiring a small modification. A placement solution, found in this stage, is called a global placement. In the second stage, called detailed placement, the global placement is adapted by taking into account the different stamps that can be chosen to implement a module, and the legal positions for

every stamp.

The next section presents a discussion of the general gate array placement problem. In section 4.2 the simplified module description and global placement are discussed. The last section of this chapter presents a detailed placement algorithm.

4.1 The placement problem

For placement, a design specification is given as a set of modules and a net-list. The modules of a design are instances of macros contained in the macro library. The placement problem is now defined as follows.

Assume a design, a 2-dimensional space in which the design must fit, and the macro library, used for the modules, be given. The problem is now to assign to every module the tuple (stamp–id, (x,y)), with (x,y) a legal position of the stamp, while optimising a given object function. Modules are not allowed to overlap, implying that different modules are not allowed to use the same vertices or edges of the space-graph.

The size of a placement problem instance is given by the number of modules to be placed, the number of different stamp choices for every module, and the number of legal positions associated with every stamp.

The restrictions that must be considered are already mentioned and concern the limited and fixed design space, the stamps available to implement a module, and the legal positions for every stamp.

The limitations on the design space lead to two global criteria that determine the quality of an actual placement. Firstly, the placement must lead to short nets for timing optimisation. Secondly, the routability of a design must be guaranteed. This means that, given a placement of the modules, the interconnect design must be completed in the available design space. A good measure of the routability of a design is given by the wire distribution, or wire densities, in the given design window. Irregular wire distributions lead to congested wiring areas, implying that certain nets can not be completely wired.

Modifying a known placement strategy, to incorporate all the above mentioned restrictions and taking the wire distribution into account in

the objective function, seems not practical, due to the increase of the problem size as defined above.

The solution presented in the following sections is to compute a complete placement in two stages. The first stage determines a global placement, based on a simplified module description. In this stage, the different implementations for every module are not considered. Further, a simplified set of legal positions is associated with every module. These two assumptions give a considerable reduction of the problem size. The objective for global placement is to optimise net length and wire distribution. In the second stage, a detailed placement is computed, taking into account the different stamp options for every module, and the legal positions for every stamp. The objective of the detailed placement is to choose for every module an optimal shape by selection of a stamp, and to place that stamp at the optimal legal position, while preserving the positions of the modules relative to each other.

4.2 Global placement

The first step of the global placement procedure is to derive a simplified module description, given a design instance. Because all modules are instances of macros contained in the macro library, the approach chosen is to derive simplified macro descriptions. Macros with no associated instances in the design are not considered.

The basic idea of the simplified macros is to create a placement problem instance that can be used by a wide range of placement strategies. The derived problem instance approaches an ideal situation [Frankle86] if all stamps have equal shapes and legal positions.

This ideal situation is defined by a set of n nodes to be placed on a set of n given legal (x,y) positions and a net-list. The objective is to assign a unique legal position to every node, while optimising a given object function incorporating net length and wire distribution.

The reasons that this problem instance is regarded as ideal are:

1) Modules are regarded as nodes, implying that shapes may be omitted, or are uniform.
2) Nodes placed at adjacent legal positions have no overlap. Thus

overlap needs no consideration during placement.
3) Any of the legal positions assigned to a node is correct. Further consideration of other aspects is not needed.
4) By having not more than the exact number of legal positions available, the problem size is minimal.

Now, the topic of the remainder of this section is to present a simplified macro description, that approaches the ideal problem instance as close as possible, starting from a general problem instance.

The first step is to provide uniform module shapes. Therefore, a simplified macro has no associated stamps, but has a fixed shape. This shape is defined as the average shape of the stamps associated with the original macro. It is trivial that in this way, the macro description approaches the ideal situation if all the stamps of all the macros have the same shape.

The second step concerns the legal positions. This is a more difficult problem, because the number of simplified legal positions must be minimised, and the legal positions must also approach the ideal problem instance if the distribution of the legal positions is more uniform. Note that the ideal situation, having n legal positions for n nodes, can also be regarded as having n legal positions for every single node, or at every legal position one of the n nodes may be placed. However, in the general case, different sets of legal positions are associated with nodes, and these sets may have a non-empty intersection.

In the following, an algorithm is presented to compute these sets for every simplified macro. The objective is to compute a minimal set of legal positions, represented by grid cells, while satisfying the condition that every module can be placed.

For every simplified macro m', the number of instances of the associated, original macro m in the design is determined. This number is called the "demand" for this simplified macro m', and is denoted by:

$$\text{Demand}(m') = \#\text{instances of associated macro } m \in \text{MacroLib} \qquad (4.1)$$

The window in which the design must fit is described by a rectangular placement grid, denoted as PG. Every grid cell represents a small area of the window. The shapes of these grid cells are of the same order of magnitude as the shapes of the stamps in a design. The simplified legal positions will be denoted in terms of the placement grid, where one grid cell represents one unit in an associated carthesian coordinate system. If a stamp of the original macro m has a legal position in the area associated with the grid cell, the grid cell represents a legal position for the simplified macro m′. A grid cell is denoted by the tuple (x,y), where x and y are the carthesian coordinates of the respective location in PG.

Two crucial assumptions are made at this point. The first is that, whether or not a macro may be placed at a grid cell, is only determined by inspection of legal positions of the macro. The second assumption is that at most one module may be placed at one grid cell at a time.

The first assumption implies that the actual shape of the simplified macro and the area represented by a grid cell where the macro is placed, are not considered. Consequently, the area of a macro m′ may be larger then the area represented by the grid cell (x,y), where the macro is placed. By associating a shape with every simplified macro and grid cell, the relation between a simplified macro m′ and the grid cell (x,y) where the macro is placed, can be incorporated into the object function for global placement. The crucial point is that the actual shape of a simplified macro and the area represented by a grid cell are not used to determine whether or not a simplified macro may be placed at a certain grid cell (x,y). This is essential for the simplification of the legal positions of the original macros presented in this the remainder of this section.

The second assumption avoids the situation that more modules are placed in the same grid cell, resulting in, for example, shorter net length. If more modules could be placed in one grid cell, a detailed placement procedure must determine whether there exists a choice of stamps for every module such that all stamps can be placed at one of their legal positions in the area, represented by the core cell, without overlap. If this is not the case, one module, or more modules, must be selected that are removed from the grid cell and placed in other grid cells. The selection of these modules is not trivial, and will cause

unwanted disturbance of the global placement. Further, it facilitates the evaluation of the "supply" of grid cells of a given placement grid, as will be shown below.

With every grid cell, the simplified macros that can be placed at that grid cell are associated. This is defined as the legal set associated with that grid cell, and is denoted by:

$$\text{Legals}(x, y) = \left\{ m'_1, m'_2, .., m'_n \right\} \tag{4.2}$$

The coordinates of the grid cell in the associated coordinate system thus represent legal positions for the simplified macros.

Now assume that a placement grid is proposed. Then the number of legal positions associated with every simplified macro provide a supply of possible positions where that macro can be placed. However, in general, more than one macro can be placed at the same legal position. The possibility that a specific grid cell remains available for a specific module thus depends on the number of different macros that can be placed at the same grid cell, but also on the number of instances of that macro in the design. So, the supply of grid cells, or legal positions, for the simplified macro is denoted by the following formula:

$$\text{Supply}(m') = \sum_{(x,y) \in PG} \left\lfloor \frac{\text{Demand}(m')}{\sum_{m'_i \in \text{Legals}(x,y)} \text{Demand}(m'_i)} \right\rfloor \tag{4.3}$$

The claim is, that the supply of grid cells approaches the ideal situation, if the distribution of the legal positions of the original macros is more uniform. This can be seen as follows. In case of an equal distribution of the legal position of the original macros, assume that all macros can be placed in every grid cell. By definition 4.1 the total demand equals the total number of modules in a design:

$$\sum_{m'} \text{Demand}(m') = \#\text{modules} = |\text{Modules}|$$

Assume the dimension of the placement grid PG be given by p_x and p_y. Then the total supply for a single simplified macro m′ is given by:

$$\text{Supply}(m') = \sum_{(x,y) \in PG} \left[\frac{\text{Demand}(m')}{|\text{Modules}|} \right] = \frac{p_x p_y \text{Demand}(m')}{|\text{Modules}|}$$

The supply for all simplified macros together is now given by:

$$\sum_{m'} \left[\frac{p_x p_y \text{Demand}(m')}{|\text{Modules}|} \right] = \frac{p_x p_y \sum_{m'} \text{Demand}(m')}{|\text{Modules}|} = p_x p_y$$

So, the total number of legal positions supplied equals the number of grid cells of the proposed placement grid. The minimal number of grid cells needed is given by the demand for all simplified macros, and is, of course, equal to the number of modules in the design. Thus if the dimensions of the placement grid are chosen such that $p_x p_y = |\text{Modules}|$, the minimal number of legal positions is given.

The demand and supply for every macro, given a placement grid, determine whether or not a placement grid proposal is accepted. A placement grid is accepted if the total supply of grid cells for every macro exceeds the demand of grid cells for that macro. The refinement of a not accepted placement grid is done by a straightforward bisection procedure that is applied in both x- and y-direction. For every orientation, a grid cell is split into two equal parts. This implies that both dimensions of the placement grid are increased with a factor 2, thus after k bisections both dimensions of PG equal 2^k.

In the following, the differences between the x and y directions are omitted for convenience, and it is assumed that p_x equals p_y. The lower bound, on the number of bisections required, is determined by the number of modules used in the design. It is trivial that after k bisections, the number of grid cells must be greater than or equal to the

number of modules to be placed:

$$2^k 2^k = 2^{2k} \geq |\,\text{Modules}\,|$$

This defines the lower bound on the number of bisections required:

$$k_{lower} \geq \frac{1}{2}\,{}^2\log|\,\text{Modules}\,| \tag{4.4}$$

An upper bound is determined by the minimal shape of the placement grid cells. If the grid cells become too small, further bisections of the grid cells can not increase the supply for any simplified macro. This situation is reached if, for every stamp, at most one legal position occurs in the grid cell, or if the shape of the grid cells becomes smaller than the shapes of all the stamps. Assuming equal shapes for all grid cells, this is denoted for both orientations by:

$$\text{CellSize}_k = \frac{\text{WindowSize}}{2^k} \geq \text{CellSize}_{min}$$

This defines the upper bound on the number of bisections that are applied for one orientation:

$$k_{upper} \leq {}^2\log\left\lfloor \frac{\text{WindowSize}}{\text{CellSize}_{min}} \right\rfloor \tag{4.5}$$

The algorithm fails if the upper bound on the number of bisections for both orientations is exceeded and the supply of grid cells is not sufficient for all the simplified macros. This indicates the possibility that the design can not be placed in the given design window.

A last remark concerns the final placement grid PG. By applying the bisection procedure straightforward for both orientations, rows and columns of grid cells may be obtained that contain no legal positions for any simplified macro. These rows and columns need not to be considered for subsequent bisections and are removed after the placement grid computation is completed, by merging them with

adjacent grid cells. Consequently, the final x and y dimensions of the placement grid may be smaller than what would result from the number of bisections applied.

The outline of the algorithm is given below. The differences in the x- and y-directions are omitted for convenience.

1) PG = DesignWindow ;

2) $\forall_{m'}$ [compute Demand(m')] ;

3) compute k_{min}, k_{max} ;

4) **for** k = 0 **to** k_{min} −1 **do** bisect (PG) ;

5) **for** k = k_{min} **to** k_{max} **do**

6) **begin**

7) bisect (PG) ;

8) **if** $\forall_{m'}$ [Supply(m') ≥ Demand(m')] **then ready** ;

9) **end** ;

10) cleanup PG ;

Algorithm 4.1. Placement grid computation.

The complexity of the algorithm is determined by the number of bisections used, and the computation of the supply of grid cells for every simplified macro. The computation of the supply, after k bisections, takes $O(2^{2k} | Macros |^2)$. Let p denote the maximum size of the placement grid for both orientations, given by: $\frac{WindowSize}{CellSize_{min}}$, then

incorporating the maximum number of bisections, as given above, leads to a worst time complexity of $O(p^2 \mid Macros \mid^2)$.

The performance of the algorithm is measured by comparing the actual supply of legal positions against the ideal number of legal positions, denoted by the demand. The optimal supply values are denoted by the demand values for the macros in a design, which implies that the total supply of grid cells and the distribution of the supply values must equal the distribution of the demand values. These distributions are denoted by the average value (μ) and the standard deviation (σ). The experimental results for two designs, Random200 and 9Sym, mapped onto three different gate array master slices, are summarised in tables 4.1 and 4.2. The master slices are discussed in detail in chapter 6.

Supply	Required	HDGA	UA6	TAL004
dff	50.00	79.75	59.75	72.50
inv	48.00	87.12	62.80	194.95
nand2	49.00	88.94	64.11	147.97
nand3	53.00	96.19	69.34	154.58
Total	200.00	352.00	256.00	570.00
μ	50.00	88.00	64.00	142.50
σ	1.87	5.85	3.47	44.23

Table 4.1. Supply values obtained for the macros used in design Random200 mapped onto three different gate array master slices with constant occupation rate.

The results denoted in table 4.1 show that the macro "dff" bounds the supply values for all gate arrays used. The reason is that "dff" is the largest macro with the least number of legal positions for all three gate array master slices. Because the refinement of the placement grid continues until all supply values exceed the demand values, the supply values for the other macros are larger then strictly necessary. The results for UA6 are the best because the differences in the shapes of the four macros are the least and dominate the differences in the distributions of the legal positions. In case of UA6, part of the wiring pattern of "dff" is "hidden" in the channel, which does not influence the legal position sets of the other macros and reduces the difference in the

shapes. For the other two gate array master slices, the channel area is not used or available, resulting in larger macros that directly influence the available legal positions. The smoother distribution of legals for the Sea-Of-Gates gate array HDGA, compared to the channel gate array TAL004 are clearly shown.

Supply	Required	HDGA	UA6	TAL004
inv	46.00	47.48	98.83	82.13
nand2	120.00	123.87	257.82	162.07
nand3	51.00	52.65	49.35	65.81
Total	217.00	224.00	406.00	310.00
μ	72.33	74.67	135.33	103.33
σ	33.77	34.86	88.93	42.06

Table 4.2. Supply values obtained for the macros used in design 9Sym mapped onto three different gate array master slices with constant occupation rate.

The supply values for design 9Sym are dominated by the distribution of the legal positions. The macro "dff" that dominates the supply in the previous example is not used in this design. The results summarised in table 4.2 show the superiority of the HDGA for the distribution of the legal positions. The disturbance of the legal positions is mostly shown for UA6, where the macro "nand3" has the least number of legal positions which must all be shared with the other two macros. The same situation holds for TAL004, but the structure of this master slice is more regular than for UA6, resulting in an acceptable solution.

With the computation of the placement grid, the simplified macro description is completed. Together with the net-list, a problem instance for placement is given that can be used by a large range of placement algorithms. No essential assumptions concerning placement style have been made. This means, in the gate array context, that no preference is incorporated for a row-based, island-based or Sea-of-Gates style placement.

Examples of placement algorithms that can be applied are simulated annealing, eigenvalue decomposition and min-cut placement, provided

that an appropriate object function is formulated. Two example algorithms have been implemented, one based on simulated annealing [Otten84], the other based on an eigenvalue decomposition algorithm [Frankle86]. The quality of the solutions of both algorithms are compared by evaluation of the net length and wire densities obtained. These results are summarised in table 4.3 and table 4.4.

gate array	net length		wire densities			
			x-axis		y-axis	
	anneal	eigen	anneal	eigen	anneal	eigen
HDGA	25154	33044	91	129	125	131
UA6	42154	54486	113	140	106	150
TAL004	26989	32365	127	158	103	138

Table 4.3. Evaluation of global placement by simulated annealing and eigenvalue decomposition for design Random200.

The number of eigenvalues computed for Random200 are 14, leading to a run time of about 16 minutes for all three gate array master slices. The run time required for annealing ranged from 21 to 37 minutes. All runs are timed on an Alliant FX4 computer.

gate array	net length		wire densities			
			x-axis		y-axis	
	anneal	eigen	anneal	eigen	anneal	eigen
HDGA	21567	28289	74	141	144	176
UA6	32431	49111	94	148	114	163
TAL004	24785	31707	124	157	85	153

Table 4.4. Evaluation of global placement by simulated annealing and eigenvalue decomposition for design 9Sym.

The number of eigenvalues computed for design 9Sym equals that for Random200, namely 14, which leads in this case to a run time in the order of 28 minutes. The annealing required in the order of 26 minutes cpu time for all the three master slices.

Both designs clearly show the inferior results obtained by the eigenvalue decomposition. The principal reasons for this shortcoming are in the object functions used in both algorithms and the evaluation of the object functions during the execution of the algorithms. The object function for the eigenvalue decomposition is fixed by the weighted entries in the connectivity matrix used by the algorithm. A further limitation is that the object function is required to be formulated in a quadratic form in the carthesian coordinates of the modules to be placed. This limits the flexibility of the object function with respect to the effects that can be incorporated. During the execution of the algorithm the values stored in the connectivity matrix remain fixed and determine the ultimate result. The main difference with the annealing procedure is that the object function is more flexible, by allowing the incorporation of non-linear forms. Further, the object function is constantly evaluated for many different placements, thus the link between the objective and a placement is more directly, compared to the eigenvalue decomposition. This results in the better global placements obtained by simulated annealing compared to eigenvalue decomposition. In the remainder of this thesis, simulated annealing is used for global placement.

4.3 Detailed placement

In the following it is assumed that a global placement has been found. Thus every module is associated with a placement grid cell. By convention, there exists a stamp in the macro library that has a legal position within the area represented by the placement grid cell.

The goal of detailed placement is to determine the optimal stamp for the module and to place this stamp at the optimal legal position close to a reference position. The reference position is the lower left corner of the area represented by a placement grid cell. The size of the placement problem is reduced by the fact that the area in which every module must be placed is bounded by the global placement procedure.

A straightforward approach to solve the detailed placement problem is to solve an independent problem for every grid cell. The disadvantage of this approach is that the global effects of a local solution are completely ignored. A better approach is to regard the problem in a more global context. The strategy proposed here is based on a one

dimensional placement approach. The placement grid is regarded as a number of strips, where a strip is a single row or column of placement grid cells. For every strip, the optimal stamp size and legal position in the direction of the strip are determined. For a row, the optimal width and x-position and for a column the optimal height and y-position are determined.

The algorithm proceeds by strip. The order in which the strips are inspected is determined by the "saturation" of the strips. The saturation is determined by the minimal size in the direction of the strip of all modules that are placed in the strip, compared to the size of the strip. The strips are processed in decreasing value of saturation.

As mentioned before, for every module placed in the strip, the optimal shape and position must be determined, this is called "module fitting". The area in which the module must fit is limited by an interval. The lower bound is determined by the first free coordinate in the strip. The upper bound is determined by the size of the strip minus the total, minimal, size of the modules that must be placed in the strip to the right or above the current module. Given this interval, the stamps, in conjunction with the legal positions, are examined to find a stamp with a legal position that fits into the given interval. If the module is already partially fixed, only the corresponding stamps and legal positions need to be examined.

If the module fitting fails, two escape procedures are at hand. The first escape is to move the module to an adjacent, parallel strip that is not already inspected. The module is placed in the optimal, available grid cell in the strip. The optimal grid cell is defined as the grid cell at minimal distance from the grid cell where the module was originally placed. A grid cell is available if no module is placed at this grid cell, and it has not been previously inspected. In this way, the displacement of the module is limited, which, in turn, bounds the disturbance of the placement.

The second escape procedure is used if the module can not be moved to an adjacent strip. In this case, the module is removed from the placement grid and put in an overflow list. After all strips have been inspected, the modules in the overflow list are re-inspected and placed on a best fit basis. The order in which the modules are placed is based

on their shapes, the largest are tried first. The position where the module will be placed is has a minimal distance to the originally proposed grid cell position.

The global effect of the algorithm described above is that instead of fixing the complete module at once, the module is fixed in two steps. A complete determination of the exact stamp and legal position only occurs if the module is fixed in two orthogonal strips. This leads to better results than a complete fix of the module in one step. The resulting algorithm is shown at the next page.

The complexity of the algorithm is determined as follows. Let M denote the number of modules used in the design, S the maximum number of stamps associated with a single macro, L the maximum number of legal positions of a stamp in the given design window, and p the maximum strip size for the strips of PG. If the legal positions are uniformly distributed over the placement grid, a good estimation of the number of legal positions for one placement grid cell and in one direction is given by: $\frac{\sqrt{L}}{p}$.

The complexity of the first phase of the algorithm, the processing of the strips, is determined by the number of times one module is inspected, the number of stamps inspected for every module and the number of legal positions inspected for every stamp. This leads to a complexity of $O(MS\sqrt{L})$.

In the second phase, the modules put in the overflow-list are inspected. For every module all possible stamps and legal positions for every stamp are inspected to determine the optimal stamp and legal position. This takes worst case O(MSL) time.

The practical performance of the detailed placement algorithm is measured according to placement preserving characteristics. This means that the quality of a placement computed by the global placement procedures must be preserved by the detailed placement algorithm. The quality of a placement is defined by the overall net length and the maximal wire densities along the x- and y-axis. Evaluation of these quantities before and after the detailed placement algorithm is applied, gives a good indication of the placement

```
1) ∀strip ∈ PG [ compute Saturation(strip) ] ;
2) Sort strips according to decreasing value of saturation ;
3) for each strip ∈ PG do
4)     for each module ∈ strip do
5)     begin
6)         compute interval [ low, high ] ;
7)         if module partially fixed then
8)             find optimal matching stamp and legal position ;
9)         else
10)            find optimal stamp and legal position ;
11)        if no solution found then
12)            move module to adjacent strip ;
13)        if module not moved then
14)            put module in overflow–list ;
15)        adjust strip saturation and strip order ;
16)    end ;
17) end ;
18) fit modules in overflow–list ;
```

Algorithm 4.2. Detailed placement.

preserving characteristics of the detailed placement. Some practical results are denoted in table 4.5 and table 4.6. The designs used are the Random200 and 9Sym, which are introduced earlier in this chapter, and are globally placed by the simulated annealing procedure discussed in the previous section.

gate array	net length		wire densities			
			x-axis		y-axis	
	before	after	before	after	before	after
HDGA	25154	27677	91	96	125	112
UA6	42154	43247	113	114	106	106
TAL004	26989	30338	127	123	103	99

Table 4.5. Practical results showing placement quality preserving by detailed placement for design Random200 mapped onto three different gate arrays.

The results summarised in table 4.5 show that for design Random200 the detailed placement the net length increases, while the wire densities show both improvements and distortions.

gate array	net length		wire densities			
			x-axis		y-axis	
	before	after	before	after	before	after
HDGA	21567	20561	74	72	144	144
UA6	32431	31932	94	88	114	113
TAL004	25730	24499	125	125	90	84

Table 4.6. Practical results showing placement quality preserving by detailed placement for design 9Sym mapped onto three different gate arrays.

The results shown in table 4.6 reveal that for design 9Sym improvements are obtained in all evaluated cases. This is achieved by the fact that the macros used in the design are more uniform then in the former design.

Concluding, the detailed placement procedure presented in this section provides a fast and constructive method for the shape fitting problem and is capable to achieve placement solutions with a quality close to that obtained by the global placement procedure.

5. A Routing Strategy

In this chapter a general routing strategy for gate arrays is presented, based on the gate array and design descriptions as defined in chapter 3. The generality of the approach is justified by the fact that no restrictions are introduced that may limit the range of application. By consequence, this strategy applies to all routing problems defined by appropriate gate array and design descriptions. The essence of the presented strategy is to solve the routing problem by a divide and conquer strategy. A first step in this approach is to solve the routing problem in two stages, generally referred to as global and local (detailed) routing.

For global routing, the design space is modeled by a simplified description, based on a partition of the design space into small sub-areas, and taking into account the predesigned wiring patterns of the placed stamps. An important aspect of this model is that it allows to apply various global routing algorithms with minor modifications.

A global routing solution determines for every sub-area which net must be routed through this area and which boundaries must be crossed. This interpretation allows to extend the divide and conquer approach to the detailed routing stage. In this stage, every sub-area defines a switchbox routing problem with possible obstacles within the represented area. The terminals of the nets, crossing the area's boundaries, may be fixed or floating, depending on the predesigned wiring patterns of the placed stamps overlapping the area, and the routing of nets in adjacent sub-areas. In this way, a global routing

solution defines a sequence of switchbox routing problems to be solved in the detailed routing stage. The detailed routing is based on a maze runner principle which is tuned to the general description of the design space. Two important aspects determining the choice were the presence of obstacles within the switchbox area, and the capability to deal with any number of wiring layers.

Although the strategy outlined above provides a general routing concept, it has also some disadvantages. The two most important are the accuracy of the global router model used and the memory requirements during the detailed routing stage. A possible solution of both problems is to combine the two stages into one routing strategy.

A last topic of this chapter concerns the implementation of a parallel detailed routing strategy. With the development of modern computer architectures, the possibility to exploit inherent parallelism in the various design processes comes within reach. Because in the automated design process, much time is spent in the detailed routing stage, it is the most evident stage to exploit the benefits of parallelism.

The next section presents a discussion of the global routing stage. Main topic of this section concerns the modeling of the design space used for global routing. The detailed routing is discussed in sections 5.2 and 5.3. The former section presents the general outline of the detailed routing strategy, the latter outlines the tuned version of the maze runner algorithm. In section 5.4 the combined routing strategy is discussed. The last section, 5.5, of this chapter concerns the implementation of the parallel routing scheme.

5.1 Global routing

A problem instance for global routing is in principle defined by a design after placement and a netlist. However, for global routing, a more suitable problem instance is defined by a rectangular grid of cells, representing the design space. This grid is called the "global router grid" and is denoted by GG with:

$$GG = GG(x,y) \text{ with: } x = 1, 2, .., n_x \text{ and } y = 1, 2, .., n_y \qquad (5.1)$$

The global router grid is "uniform", meaning that no distinction is made between the principal wiring area and the area occupied by placed stamps. The portion of the design space represented by a grid cell GG(x,y) is described by the space-graph $G_s(x,y)$ of the area.

The four boundaries of a grid cell are called the North, East, South and West boundary. On any boundary a "wiring capacity" is defined that denote the number of nets allowed to cross that boundary. The wiring capacity of a grid cell GG(x,y) is denoted by:

$$\text{WiringCap}(x,y) = (\text{NorthCap}, \text{EastCap}, \text{SouthCap}, \text{WestCap}) \quad (5.2)$$

A set of nets is defined for every grid cell, denoted by GlobalNets. For global routing, a net is defined by a net identification, denoted by Netid, and a GlobalRoute which is a sub-set of:

$$\text{GlobalWiring} = \left\{ \text{Terminal, NWired, EWired, SWired, WWired} \right\} \quad (5.3)$$

The GlobalRoute of a net denotes if the net has a terminal in the area represented by a the grid cell, and the cell boundaries crossed. A net has a terminal in the area represented by the grid cell, if at least one vertex of the space-graph $G_s(x,y)$ has a corresponding net label f_{net}. A global net thus consists of the following tuple:

$$\text{Net} = (\text{Netid}, \text{GlobalRoute}) \quad (5.4)$$

And a grid cell GG(x,y) is described by the following triple:

$$\text{GG}(x,y) = (G_s(x,y), \text{WiringCap}, \text{GlobalNets}) \quad (5.5)$$

Initially, GlobalNets contains only those nets that have a terminal in grid cell. The task of the global router is to find "global routes" that connect all terminals of the nets, while satisfying the capacities on the global grid cells. A "global route" of a net is defined by a set of grid cells and the cell boundaries crossed. In case of gate array layout,

special attention must be paid to find solutions that provide an acceptable balance between short net length and a smooth distribution of the wires over the design space, to avoid wire congestion.

The advantages of the global router grid description of the design space are the simplified problem description, the uniform description of the design space occupied by placed stamps and the principal wiring area, and the algorithms that can be applied with this problem description.

The simplification of the problem description is evident by the association of small areas with every grid cell and the boundary capacities which take into account various aspects of the detailed routing. Some examples of these aspects are the number of wiring layers available and the number of routing tracks available for wires crossing a cell's boundary.

The uniform description of the design space is important to achieve a transparent routing through placed stamps. By incorporation of these stamps into the global router grid, stamp areas are treated in the same way as the principal wiring area. The predesigned wiring patterns of the placed stamps are captured by an appropriate setting of the cell boundary capacities, which is not simple. In this way, transparent routing through stamps is achieved naturally. It also implies that at this stage no restrictions are introduced that are based on any assumption of the organisation of the principal routing areas in the master slices used, as well as the wiring patterns of the placed stamps.

The generality of the problem description makes it easy to incorporate various global routing algorithms with minor modifications [Ting83, Li84, Parng89].

Although the global router grid has important advantages, there are also a number of important disadvantages. The most important is that the final interconnect result of the subsequent detailed routing stages depends on the resolution of the global grid. The second is that the capacity definitions are not trivial, as well as the updates on these capacities as a net is routed across a boundary. The third disadvantage is that, to exploit the transparency of a predesigned stamp, not only the terminals of the stamp must be incorporated in the global grid, but also the predesigned wires.

A profound study concerning the definition of the global router grid is presented in [Saeijs86]. The general conclusion resulting from this study is that routing obstacles should preferably be placed at the boundaries of the global grid cells, leaving the center of the cell regions free for routing. Obstacles in this context concern any limitation of the wiring possibilities. In the general context at hand, these limitations may be caused by predefined wires, such as predefined power wires, predesigned wiring patterns of placed stamps, design rules and local routing cost functions. This indicates that the global grid definition can not be defined by considering the master slice structure only. It is design dependent. An almost trivial example of this situation occurs in Sea-Of-Gates design, where no predefined, dedicated routing area is provided.

A last issue in relation to the global routing grid concerns the capacity definitions. A good estimate of the capacity values is essential for the validity of the global routing result for the detailed routing stage. Too high capacity values will result in local wire congestion and thus non routable situations. On the other hand, too low capacity values will result in unnecessary detours of nets and an increase of the overall net length. The capacity values are determined by accounting for the following four aspects. Firstly, the internal cell blockages may limit the number of wires that can be successfully routed from one boundary to another. Secondly, design rules may also cause limitations on the number of wires that can be routed within a predefined region. This effect is hard to capture in a fixed capacity definition, because wiring limitations are determined by routes taken by other nets and this information is not available until the detailed routing within a grid cell is done. The same holds for the third effect on the capacities, which is caused by the cost function. The cost function is defined to guide the detailed routing, and determines more or less the wiring patterns that are generated by the maze runner. It is obvious that different wiring patterns have different effects on the overall number of wires that can be routed within a region and is therefore also difficult to determine before the actual routing is done. The fourth effect that determines a good capacity estimation is the performance of the detailed router itself. If, for example, the detailed router proceeds on a "one-net-at-a-time" basis, the total wiring result in the region represented by a global cell depends on the chosen net ordering. Further, if the router exploits rip-up and re-route techniques, the number of nets that can be

completed within a predefined region can be increased. This implies that in this situation higher capacity values can be chosen. In general, the conclusion is that capacity values should also take into account the wiring performance of the subsequent detailed routing stage. More about this topic is presented in section 5.4. The general idea of the discussion presented above is that a good and still generally applicable estimation of global grid cell capacities is not as straightforward as suggested in, for example [Burstein83]. In the approach presented here, capacities are estimated by inspecting the cell's region for positions where nets are allowed to cross a specific boundary. Although this still does not take into account all of the effects outlined above, experiments indicate that these capacity estimates are sufficiently accurate.

An important consequence of the global grid definition concerns the general net modeling for global routing. In general, nets have multiple terminals and every terminal is represented by a single global router grid cell. However, in the general approach outlined above, a single net terminal may appear in more than one grid cell, due to the predesigned, internal wiring patterns of the placed stamp in relation to the global grid definition. This situation occurs if a stamp occupies an area which is represented by more than one global grid cell, and a terminal wire of this stamp is situated in more than one global grid cell. In order to model this situation, a distinction is made between fixed global wires and routed global wires. It is assumed that net terminals appearing in more than one grid cell are connected by fixed global wires which are allowed to form "loops" in the global net representation. Because, by definition, terminal wires of a stamp are complete, the global representation of these wires concerns a terminal in every grid cell in which a part of the terminal wire is situated, and these terminals are already connected by a fixed global wire. By consequence, if, for example, a terminal wire segment crosses the boundaries of four global grid cells mutually abutting, a global wire loop is defined. A global router must take these situations into account. The advantage of the fixed global wiring is in the reduction of the net length of the wired connections. To account for the fixed global wiring, the GlobalWiring as denoted in definition 5.3 is extended with the following set:

$$\text{FixedWiring} = \left\{ \text{NFixed, EFixed, SFixed, WFixed} \right\} \tag{5.6}$$

Given a design after placement, a global grid satisfying the requirements as presented above, is determined by a straightforward bipartition of the available wiring space. It is assumed that the bipartition produces a uniform global router grid, which is obtained by defining subsequent cut lines for the complete wiring space. The cut lines are determined by using a point configuration of the placed modules. A vertical cut line for a region is determined by the average x-coordinate of all the points contained in the region. The analogous criterion holds for a horizontal cut line. The partitioning terminates if appropriately sized grid cell are obtained. The appropriate size of the grid cells is partly determined by the placed stamps, because regions that do not contain center points of placed stamps are not further partitioned, and partly user controlled. The importance of a user controlled cell size is in the time performance and memory usage of the subsequent detailed routing stage. Because the detailed routing algorithm has a linear time complexity, with respect to the available wiring space, smaller routing areas increase the speed of the detailed routing process. This aspect is discussed in more detail in the next section. Further, the memory requirements are also directly related to the global grid cell sizes by the wiring graph representation of the routing area in a global grid cell. These features are discussed in more detail in the following sections.

After the global router grid is determined, the capacity values are determined as follows. For every global router grid cell GG(x,y) the space-graph $G_s(x,y)$ of the design space is built, according to the procedure outlined in section 3.4. Next, the vertices of the space-graph are partitioned into two disjoint groups, based on the net labels. Those vertices that are occupied by nets indicate that the global grid cell is a terminal cell for those nets. If a net has also connections crossing the boundary of the cell, a fixed global wiring exists. As already outlined in section 3.4, this situation occurs if an edge from a vertex on the boundary has a connection status and refers to a vertex outside the cell's area. Outgoing connections also occur if vertices inside the grid cell have equivalent vertices outside this area. The remaining vertices of G are not occupied by a net and thus free for routing. Those vertices, which are accessible and have also a free edge or equivalent vertex

outside the cell area, indicate the possibility to route across the cell's boundary and therefore add wiring capacity to the boundary crossed. The total wiring capacity of a cell boundary is taken as the number of vertices allowing to cross the boundary. This completes the global router grid definition.

Given this problem instance, general global routing algorithms can be used. As an example, the hierarchical wire router of Burstein [Burstein83, Nuijten85] is implemented. Some experimental results are shown in the table given below. In order to demonstrate the effect of the transparency of the placed stamps, the estimates for net length and wire densities after placement of the design are incorporated.

gate array	net length		wire densities			
			x-axis		y-axis	
	placed	routed	placed	routed	placed	routed
HDGA	4251	1794	28	24	40	12
UA6	5377	2874	29	20	36	18
TAL004	5056	4466	32	27	42	46

Table 5.1. Practical results comparing net length and wire density estimates after placement and after global routing for design Rdc50 mapped onto three different gate arrays.

The global routing results show the effect of the transparency of the placed stamps, which results in a decrease of both net length and wire densities. For placement the net length is estimated by half the perimeter of the bounding box of a net, whereas for global routing the net length is estimated by the global route of the net and the size of the corresponding global grid cells. The wire densities along the x-axis are estimated by the maximal horizontal span of the nets, and the maximal number of nets crossing any (imaginary) vertical line. The analogous holds for the wire densities along the y-axis. The differences in the wire density definition are that after placement the terminals of the nets are assumed at the center of the placed stamps, and after global routing at the center of the corresponding global grid cell. An increase of the wire density along the y-axis for gate array TAL004 indicates that the horizontal transparency of the stamps is very low. This, in

turn, results in a smaller decrease of netlength, due to the detouring of the nets around the placed stamps.

In addition to the net characteristics the routability of the design is also indicated by the resulting global grid capacities. Ideally, the completion of the subsequent detailed interconnect design is guaranteed if no boundary capacity shows an overflow by the global routes. Unfortunately, this guarantee can not be given, due to the fact that the capacity values are estimates not accounting for all phenomena given earlier. By consequence, the completion of the design may fail, contrary to the predictions indicated by the capacities. An example of the situation is shown in the table below, which compares the completion of the design documented in table 5.1, against the predicted routability supplied by the global routing.

gate array	boundary capacities		nets completed
	min	max	
HDGA	1	11	100 %
UA6	1	19	97 %
TAL004	-7	13	76 %

Table 5.2. Practical results showing routability prediction after global routing compared to net completion rate for design Rdc50 mapped onto three different gate arrays.

A possible solution for this problem is presented in section 5.3 and consists of an iteration between global and detailed routing, where the results of the detailed routing are used to update the global grid capacity values.

5.2 Detailed routing

A problem instance for detailed routing is called a "routing task", and is defined by a global router grid cell GG(x,y). The "wiring space", represented by the grid cell, is described by a space-graph $G_S(x,y)$, as defined in chapter 2. The space-graph is built from the master slice and the patterns of the placed stamps that overlap the wiring space. The global net wiring specifies which nets must be routed within the

limited wiring space. Further, the boundaries, at which the nets must have a terminal, are defined. These boundary terminal positions may be undefined or fixed due to previously wiring in adjacent grid cells. If the terminal positions are undefined, the optimal terminal positions must be determined by the detailed router.

Proceeding by global router grid cell, the detailed wiring of a chip is completed if all wiring problems for every grid cell are solved. The order in which the grid cells are processed, is determined by the boundary capacity values of the grid cells. These capacity values indicate the difficulty of the detailed wiring problem to solve. The larger the overflows, the more difficult the wiring problems to solve. The most difficult wiring problem is solved first, having the maximum degree of freedom, concerning the boundary terminal positions. The easiest problems are solved last, having most of the boundary terminal positions fixed, due to the previous wiring in the adjacent grid cells.

The most important advantages of this detailed routing scheme are: Firstly, the search space for detailed routing is kept small. The size of the search space is independent of the space occupied by a design and can be user controlled. Compare for example, a strict net-oriented approach with the search space defined by the bounding box of the net, or taking rows or columns of global router grid cells. In these situations, the wiring space depends on the area occupied by the design, and is difficult to control. Secondly, the freedom, to choose the order of the grid cells, has a positive effect on the success of the detailed routing process.

A disadvantage of the routing scheme is that the boundaries introduce side effects that influence the success of the detailed routing within a small area. In practice, this means that a wiring solution can not always be found within the restricted wiring space of a global router grid cell. One solution that solves this problem is to enlarge the wiring space, or "routing scope", by redefining the global router grid. This, so-called "scope-enlarging" defines a larger search space by combining sets of global router grid cells into one super grid cell. The global net wiring is updated accordingly. The super grid cell defines a new routing problem in the same way as the original global grid cells. The detailed router can now solve the remaining wiring problems in the larger wiring space. The disadvantage of the larger wiring space is partly

compensated by the wires constructed at earlier stages of the detailed routing process. These wires introduce more obstacles and thus reduce the search space.

The sequence of detailed routing problems to be solved are switchbox or switchbox-like wiring problems, depending whether the boundary terminal positions are fixed or undefined. Unfortunately, most switchbox routing algorithms are not capable to solve this problem. The main shortcomings of these algorithms are in the number of wiring layers required and the possibility to handle obstacles in the wiring space. The same reasons hold for channel routing algorithms. These algorithms have an extra shortcoming, namely that they do not guarantee to find a solution, even if one exists, if the wiring space is fixed. This holds also for line search algorithms. Other difficulties are introduced by the cost functions and design rules that must guide the wiring design. A further difficulty is introduced by the equivalence relations between vertices in the space-graph, which can lead to wiring jumps.

These considerations lead to the maze runner or Lee-router as the most appropriate algorithm in this context. Before the algorithm is explained in more detail in the next section, a further refinement of a single router task is given.

Given a space-graph $G_S(x,y)$ and the global net wiring, the wiring problems to be solved by a detailed router can be partitioned into two types. The first type is determined by the internal wiring problems. These problems are defined by two or more wiring sets that must be connected, and that are already present in the window, because of the fixed wiring of different stamps placed within the window, or previously routed connections. The second type of wiring problem is the boundary problem. In this case a wire must be constructed that connects a terminal at a specified boundary. Given the space-graph $G_S(x,y)$, a set of vertices in $G_S(x,y)$ can be identified that solve this type of problem. This means that if a wire is constructed to one of these vertices, the corresponding boundary problem is regarded as solved. For convenience, a separate set for every boundary is maintained. The boundary vertices of the space-graph are called the "trivial boundary solvers". Not trivial boundary solvers are vertices of $G_S(x,y)$ having equivalent vertices outside the wiring space. Because the equivalence

relation defines a connection of some sort between the equivalent vertices, a connection, from the vertex within $G_s(x,y)$ to one outside $G_s(x,y)$, must cross a boundary, and thus is the vertex in $G_s(x,y)$ a boundary solver.

Before a wiring algorithm is applied, the vertices of $G_s(x,y)$ are partitioned, based on the net labels. For every net, a separate set of vertices is maintained. Vertices that are free for use are inspected for boundary solving and updated in the according set for every boundary. The next step is to inspect the vertices of the nets to determine if boundary problems given by the global net wiring are already solved, due to wiring in the adjacent grid cells. The remaining problems are those wiring problems to be solved by the detailed router.

5.3 The modified Lee-algorithm

The original Lee-algorithm was introduced in 1961 [Lee61] and has gained a much attention since. Suggested improvements concern the memory requirements and certain accelerator schemes.

In the present context, the memory requirements are not the primary concern, due to the limited wiring space in combination with the selective space-graph expansion presented in section 3.4. The space-graph is chosen as the main data structure on which the algorithm operates.

The problem addressed in this section concerns the selection of start and target points. The original algorithm finds an optimal cost path from a given start point to a given target point. However, practical routing problems concern multi-terminal nets and the terminals of the nets are not single points or vertices, but wire segments. This holds especially for situations with transparent routing through predesigned modules. In this situation, the choice of start and target points is not trivial and determine the success of the path search algorithm. The situation becomes even more difficult if design rules must be taken into account. If the algorithm fails to find a path, some guarantee must be given that no path can be found for any combination of start and target points, which is not trivial.

The solution to the problems outlines above, is based on the

acceleration schemes proposed in the literature. The main idea of these schemes is to speed up the path search by preferring path extensions towards the target, or limiting the search space by starting the path search from both start and target point [Rubin74, Soukup78, Korn82].

The first type of acceleration involves a modification of the cost function. This is not preferred in situations where the cost function is user defined, because the relation between the user supplied cost and the behaviour of the algorithm becomes unclear. The approach chosen is to initiate the path search from both start and target point. In the literature, the distinction between start and target vertices is made explicitly, by using different lists for start and target vertices. The reason for this explicit distinction is to retain the minimal amount of data required. However, in the present context, this limitation is not necessary. The main difference between the approach presented in this section and those presented in literature, is that the distinction of vertices according to wavefront is made implicitly. In fact, the labels needed to distinguish between the different wavefronts are already introduced in chapter 2, with the definition of the nets. Recall that for every net the vertices have an associated wiring set label, indicating whether or not they belong to the same wiring set. All vertices with the same wiring set label are connected, independent whether this is reflected in the current portion of the space-graph. By consequence, the task of the routing algorithm is redefined to find the optimal cost path that connects two vertices of the same net but with different wiring set labels.

The necessary modification of the algorithm is very small in the sense that the elementary actions taken by the algorithm must inspect the wiring set labels. In this way, the main behaviour of the algorithm remains unaltered. The main loop of the algorithm is outlined in algorithm 5.1.

Note that only one list is used for possible extensions from both start and target vertex. The basic actions taken by the algorithm are captured by the routing primitives expand(v) and extend(v,e_w). The primitive expand(v) updates all possible extensions from a vertex v and Eq(v), together with their cost, in the candidate list. The outline of this primitive is given by algorithm 5.2. An actual path extension is done by extend(v,e_w), which extends a path from vertex v in the direction

```
1)  Input: S ; /* set of vertices */
2)  CandidateList = ∅ ; Cost = 0 ;
3)  ∀v∈S [ expand(v) ] ;
4)  while CandidateList ≠ ∅ do
5)      begin
6)          ( v, ew, Cost ) = cheapest( CandidateList ) ;
7)          if extend(v,ew) then
8)          begin
9)              trace back path ;
10)             goto line 14 ;
11)         end
12)         expand( (n = v + ew) ) ;
13)     end
14)     remove unused paths ;
```

Algorithm 5.1. Outline modified Lee-algorithm.

denoted by e_w to a neighbour vertex n, denoted by: $n = v + e_w$. This primitive also indicates if two different wiring sets are merged. The outline of this primitive is given by algorithm 5.3.

The proof that an optimal path is found by the algorithm is trivial because at any time a cheapest expansion is taken from the list of possible extensions.

Given the routing algorithm as presented by algorithm 5.1, the key notion is that the set S, containing the initial vertices, is not required to contain exactly one start vertex and one target vertex to be connected. The generality of the algorithm is achieved by putting all vertices already assigned to the net in the set S. In this way, S may contain vertices of any number of wiring sets.

This observation is crucial for the solution of the boundary problems, as defined in the previous section. If a net must cross a boundary, all the

```
1)  V = { v } ∪ Eq(v) ;
2)  for each v ∈ V do
3)      if v in routing window then
4)          for all edges e_w ∈ v do
5)              if f_s(e_w) = INITIAL then
6)                  if f_net(v) = FREE ∨
7)                      [ f_net(v) = f_net(n) ∧ f_wset(v) ≠ f_wset(n) ] then
8)                      if f_d(e_w) = 0 then
9)                          if f_cost(e_w) < INFINITE then
10)                             CandidateList = CandidateList ∪
11)                                 ( v, e_w, Cost + f_cost(e_w) ) ;
```

Algorithm 5.2. Outline routing primitive: expand(v).

vertices in the corresponding boundary set are assigned the net label of the net under construction. Further, a wiring set label, not used by the net, is assigned to these vertices. All the vertices of one boundary get the same wiring set label, in order to prevent unwanted connections between different vertices of one boundary. Vertices of different boundary sets obtain different wiring set labels, in order to allow boundary to boundary connections. Next, these vertices are incorporated in the start set S, and the routing procedure is entered. If a boundary problem is solved, the vertices of the path found are eliminated from the corresponding boundary set and the unused vertices are blocked to prevent additional paths to the boundary, and for future use of boundary problems for other nets. The vertices of boundary sets, not used by a net, are also blocked, to prevent them of being used for connections while eliminating the possibility of solving boundary problems for other nets.

By its very nature, the algorithm finds exactly one optimal path, connecting two wiring sets represented by the vertices in S. By updating the vertices of the found path in the set S, subsequent reentering of the procedure will give additional paths for unconnected wiring sets until all wiring sets are connected or no path can be found.

```
1)  f_s(e_w) = ROUTER ;
2)  if f_net( (n = v + e_w ) ) = FREE then
3)      begin
4)          V = {n} ∪ Eq(n) ;
5)          for ∀_{v_i ∈ V} do
6)              begin
7)                  f_net(v_i) = f_net(v) ;
8)                  f_wset(v_i) = f_wset(v) ;
9)                  return ( FALSE ) ;
10)             end
11)     end
12) else
13)     return (TRUE ) ;
```

Algorithm 5.3. Outline routing primitive extend(v, e_w).

Because all vertices of S are used for path searching, the proof that no path exists if no path has been found, independent of the choice of start and target vertices, is trivial.

5.4 Combined global and local routing

In this section, a combined global and local strategy is presented. The problems that are solved by this approach are the definition of the global wiring capacities at the global grid cell boundaries, the scope-enlarging needed by the local routing to complete unfinished nets, and the global rerouting of incomplete nets.

With respect to the global wiring capacities, recall that these capacities should indicate the number of wires that can be routed across the associated boundary. It will be clear that these capacity values are, besides others, determined by the capability or "intelligence" of the detailed router to solve the presented routing problems. Including a feedback of the actually solved problems has the advantage that a fair

indication of the capacity values is obtained, incorporating the wiring capability of the detailed router.

A second advantage of the feedback relates to the limitation of the wiring space. The basic idea of the detailed routing was to enlarge the wiring space to solve unfinished nets. However, the larger the wiring space, the larger the portion of the space-graph that must be expanded and the advantage of the hierarchical data structure is lost. Further, the detailed router has a complexity proportional to the wiring space, which means a performance drop.

Incorporating both global and detailed routing into one common strategy solves the above mentioned problems. The approach is to embed both algorithms into an overall routing framework. In this way, the algorithms can be replaced by others, without altering the essence of the strategy.

The initial global wiring is done as before, with a preferably good guess of the wiring capacities. At the subsequent detailed routing phase, the global grid cells together with the associated global net wiring and the resulting boundary capacity values define the detailed routing tasks to be solved. Now, after the detailed routing of one global grid cell is finished, the wiring result is inspected for solved and unsolved routing problems to update the associated boundary capacities. An outline of the capacity update is presented in algorithm 5.4.

When all tasks have been processed by the detailed router, the remaining nets are returned to the global router, which determines a new global route for these nets with the updated boundary capacity values. This process continues until all nets are completed or no net is completed during the last iteration through the detailed routing phase.

Table 5.3 shows some results obtained with an experimental implementation of the combined routing strategy.

As shown in the table, the combination of both routers provided a complete solution for design Rdc50, where the detailed router alone was not able to complete the interconnect design. This result also shows that the solution is achieved with less memory usage and shorter design time than the detailed router. The performance improvement is

```
1) Input: #problems ; /* routing problems solve */
2) Input: #solved ;   /* routing problems solved */
3) Input: capacity ;  /* original capacity guess */
4) Output: capacity ; /* adjusted capacity guess */
5) if #solved = #problems then
6)       capacity = max ( capacity, 0 ) ;
7) else if #solved = 0 then
8)       capacity = min ( capacity, -#solved ) ;
9) else
10)      capacity = #solved - #problems ;
```

Algorithm 5.4. Outline capacity update for a single cell boundary.

router	nets completed	run time (sec.)	memory usage (Kbytes)
detailed	97 %	1358	3514.1
combined	100 %	850	117.5

Table 5.3. Practical results showing performance of detailed and combined routing strategies for design Rdc50 mapped onto gate array UA6.

the direct result of the limited scope enlarging during detailed routing which limits the size of the space graphs built (memory) and the search space (time).

5.5 Concurrent detailed routing

With the introduction of modern computer architectures, the possibility to exploit the inherent parallelism in the layout design process becomes interesting. Because most time is spent in the actual wiring design, the detailed routing is the most evident process to exploit the benefits of parallelism.

Before a parallel execution scheme of the detailed routing process can be presented, the major data streams within the detailed routing process must be examined. Due to the organisation of routing tasks within the detailed routing process, the obvious choice is to execute these routing tasks parallel. The detailed router as presented in this chapter, operates on a set of global data and private router data, as shown in figure 5.1.

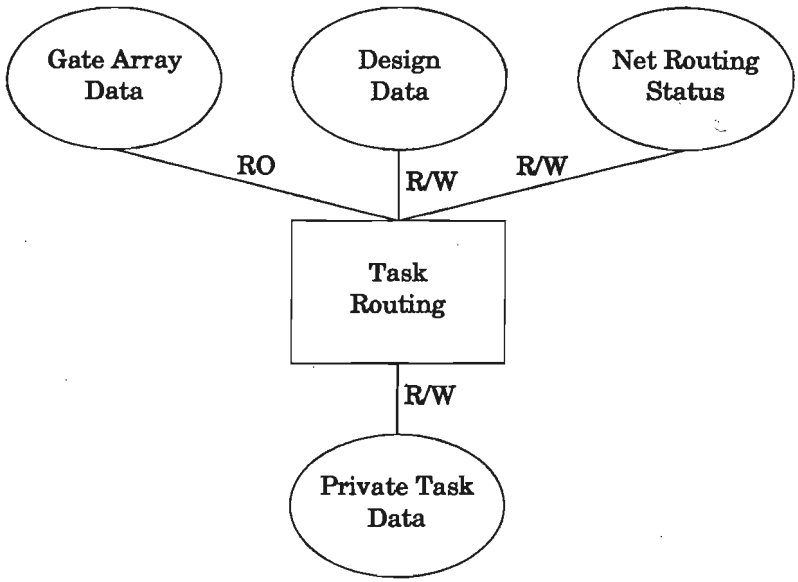


Figure 5.1. Main data streams within the detailed routing process. RO stands for read only and R/W for read and write data access. The Design Data can be further partitioned into placement data, which is accessed on a read only basis, and net list data, which is accessed on a read and write basis.

The global data is partitioned into gate array, design and net routing status data. The gate array data represents the basic gate array master slice structure together with the macro library, containing the predesigned wiring patterns. The router accesses this data on a read only basis, no information is changed during the routing process. The design data describes the placement of the stamps used in the design

and the net-list definition. The placement data is also accessed on a read only basis, especially for the mapping of the correct wiring patterns onto the correct positions in the section of the space-graph that is built for the detailed routing. The net-list data is accessed on a read/write basis. The routing status of every net is maintained globally, and concerns the already wired connections and the wiring sets to be connected. If the completed nets are stored in the design data base at the end of the routing procedure, the net-list data can be regarded as read only during the wiring design. However, it is preferred to maintain the completed nets in a database separated from the incomplete nets. A good solution is to store the completed nets in the global design database. The last data structure concerns the private router data. The most important structure in this data concerns the space-graph used by the detailed router for the actual wiring design.

The available machine architectures can be partitioned into two major classes based on the memory available for every single processor. The two classes are called "shared memory" processors and "distributed memory" processors. The type dictates how an application must be implemented. In case of shared memory, the global data is available at all times for every processor. If the data is accessed on a read only basis, no special care must be taken. The write access on global data must be controlled by a semaphore mechanism to guarantee data consistency. In a distributed memory organisation, every processor operates on a private memory. All data required by a process must be stored in the respective processor's memory. This is independent whether the data is accessed for read actions only, or for read and write actions. The major drawback of this structure, in case of the detailed routing procedure as presented in this chapter, is that the most important data, the net routing status, must be distributed over all the available processors and the different processors must communicate in order to keep the data consistent. The conclusion is that the shared memory architecture has a great advantage over the distributed memory architecture if processes need large global data structures which are accessed intensively. In the remainder of this chapter, the parallel execution of the detailed routing tasks is therefore explained for a shared memory computer architecture.

Given the organisation of data, as presented in figure 5.1, the parallel

execution of routing tasks requires that every task running on a processor has its own private router data. This implies that for every processor a separate space-graph must be built for subsequent routing. In general, routing tasks are data dependent by the definition of a routing area and a set of nets that must be routed within the area. These data dependencies must be eliminated or controlled for parallel processing. The first data dependency is created by the routing area. If space-graphs for different routing tasks have a non empty intersection, label changes on vertices and edges may influence corresponding labels in other space-graphs. The graph intersections are caused by the equivalence relations between vertices, design rule sets for edges, and boundary intersections. An extra constraint is added by the linear order in which the routing tasks must be processed according to their degree of difficulty, as mentioned in section 5.2. If this order is strictly maintained, the parallelism that can be exploited is limited by the number of tasks having the same degree of difficulty. The second data dependency between different routing tasks is given by the nets that must be routed per task. In order to avoid the creation of loops, a net may not be routed in different tasks at the same time.

The data dependencies between different tasks are used to define an execution schedule. A schedule consists of a set of "time-slots", which are sequentially processed. A time-slot is defined by a set of routing tasks that can be processed in parallel. The data dependencies introduced by the nets are not taken into account for the schedule definition. The reason is that if nets are taken into account, the parallelism that can be exploited depends on the net length in the design under construction. This can lead to a purely sequential schedule in case of very long nets such as clock or reset nets. Therefore, exclusive routing of nets is controlled by semaphores and the net data dependencies are not considered for the schedule definition. The schedule is defined according to the linear ordering of the routing tasks to degree of difficulty.

Design	9Sym	Random200a	Random200b	Logop
routing tasks	60	143	151	757
scheduled time	19	38	43	193
speed increase	3.16	3.76	3.51	3.92
run time (seq)	1684	2881	2831	29515
run time (par)	668	977	993	10533
timed increase	2.52	2.95	2.85	2.80
overhead	0.64	0.81	0.66	1.12

Table 5.4. Evaluation of overhead during schedule execution on a 4-processor computer with shared memory.

In order to evaluate the performance increase, some examples are shown in table 5.4. The table summarises the actual execution of a single schedule compared to the theoretical execution time of the schedule. The execution time of a single routing task is set to unit time. The results indicate that the amount of overhead during the execution of a schedule is in the order of one processor.

6. GAS: an evaluation

This chapter presents an evaluation of the most important aspect of GAS, which is its capability to adapt to various gate array families. Because this capability is the direct result of the generalised approach to layout modeling incorporated in the system, the success of this approach is directly related to the performance of GAS with respect to this adaptation procedure. The performance in this context is measured in terms of the time required to derive a GADL description, the amount of data required to capture the essence of a gate array family and design mapped onto a gate array, and the performance of the layout design phase.

Up to now, six gate array families have been evaluated to demonstrate the extreme flexibility and performance of GAS. These gate arrays differ in many aspects, such as the number of interconnect layers available for customisation, the organisation of the wiring area and the logic family.

In the next section a brief outline of the six gate arrays is presented. The following two sections concern a discussion of the experimental results obtained by adapting GAS to these gate arrays.

6.1 Evaluated gate arrays

The gate arrays presented in this section reflect the rapid changes in gate array layout design during the last years, which has lead to the introduction of the Sea-Of-Gates gate arrays. Due to these developments, the presented gate arrays were not at an equally mature

Gate Array	status	#macros	#gates	year
UA6	commercial	20	1260	82
GA440	educational	52	440	83
TAL004	commercial	22	500	83
GATE FOREST	experimental	10	> 50K	86
HDGA	experimental	5	> 20K	89
OCTAGON	experimental	0	> 50K	89

Table 6.1. Global overview of the evaluated gate arrays.

level at the time they were evaluated. A global overview of the used gate arrays is given in table 6.1. The mature status of the older gate arrays is clearly shown by the size of the macro library. For the three Sea-Of-Gates examples, listed at the bottom, only a very small macro library was used for evaluation. The reason for this is that these gate arrays were still under development at the time they were evaluated. Also shown in table 6.1 is the effect of the maturing of the silicon integration technology over the last years by the increase of the number of available gates. In the remainder of this section the core patterns of the six gate array families are discussed in the same order as they are listed in table 6.1. The macro libraries are not shown, the reason is that the wiring patterns of the stamps are mostly regarded as confidential information.

UA6

Documentation: see [AMI82].
Customisation: 1 metal layer, contacts to polysilicon and diffusion are preprocessed.
Core pattern: see figure 6.1.

Note the use of oxide isolation that creates the transistor islands. In the right structure the gates of the p- and n-transistors are connected. Therefore the left structure is provided for an efficient transmission gate implementation. Routing in channels is preferably horizontal. For vertical connections the polysilicon underpasses are provided to avoid channel blocking. Channel switching can be done by using the polysilicon feedthroughs in the transistor areas. The master architecture consists of alternating transistor and channel rows. The

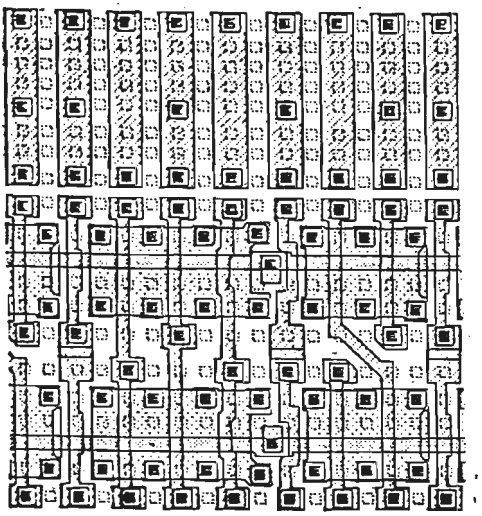


Figure 6.1. Basic core cell layout pattern of UA6.

functions in the macro library range in complexity from an inverter to a master-slave d-type flip-flop with set/reset. The macros are designed for manual layout design, which is reflected by the terminal connections that are available at the top and bottom of the stamp, thus at the channel boundaries. Special stamp wiring patterns are provided such that adjacent stamps may have overlapping power connections.

GA440

Documentation: see [Leenstra83].
Customisation: 1 metal layer, contacts to polysilicon and diffusion are preprocessed.
Core pattern: see figure 6.2.

Compared to UA6, note the use of gate isolation that allows the use of transistor rows rather then transistor islands as for UA6. An other important difference is that the gates of the p- and n-transistors are not connected. The required gate connections must be made in metal, which causes saturation of the internal area and a reduction of the routing transparency. The master architecture is again row oriented, with alternating channel and transistor rows. The functions in the macro library range from a simple inverter to a master-slave d-type flip-flop with set/reset and some complex logic functions. As with the

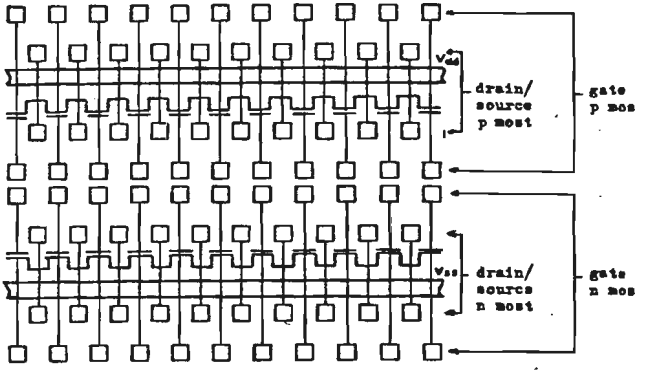


Figure 6.2. Basic core cell layout pattern of GA440.

previous gate array, the macros are designed for manual layout design. The terminals of most macros are available at the top and bottom of the macros.

TAL004

Documentation: see [TI83].
Customisation: 2 metal layers, contacts to gates are in the fixed predesigned stamps.
Core pattern: see figure 6.3.

Compared to the previous gate arrays, the most important difference is in the logic family implemented. The basic core cells are 4-input nand gates for ECL. This makes this gate array difficult to compare to the previous two. The master architecture is in principle column oriented, but, due to the power definition, an island like structure is created. It is not a true island structure in that islands of gates are surrounded by wiring area. An island consists of two columns of ten gates, mirrored around the y-axis. The functions in the macro library are more complex compared to the previous two gate arrays, due to the 4-input nand gate as basic core cell. The design of simple gates such as 4-input nands is left to the designer. Also in this case, the terminals of the macros are available at the channel boundaries.

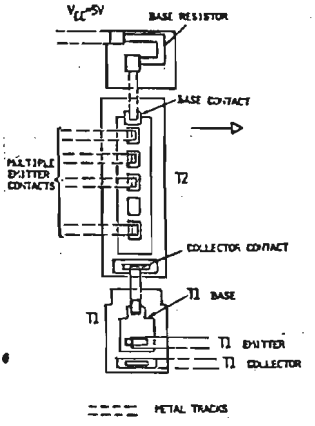


Figure 6.3. Basic core cell layout pattern of TAL004.

GATE FOREST

Documentation: see [Beunder87].
Customisation: 2 metal layers, programmable vias and contacts.
Core pattern: see figure 6.4.

This is the first of the Sea-Of-Gates gate arrays presented. The core cell is developed for routing transparency and is not optimised for area utilisation. For example, the track changing capability is seen as an important aspect of the core cell layout. In principle the core cells overlap, which makes it difficult to describe the master by a partitioned floorplan. The master architecture is typical for a Sea-Of-Gates gate array. Alternating rows of p- and n-transistors are used. Adjacent rows are mirrored for area reduction by abutment of two p- or n-transistor rows. The intention was to provide a design specific power routing, but for the evaluation fixed power tracks were defined. The macro library is, as for the other CMOS gate arrays, limited to functions of medium complexity, ranging from inverter to flip-flop. The library reflects the fact that automated layout design is done by the restriction to a uniform height for all wiring patterns. In the early version used, no attention was given to routing transparency of the macros. All internal wiring of the macro was done in first metal and the power connections were large wired loops, which did not seem to be

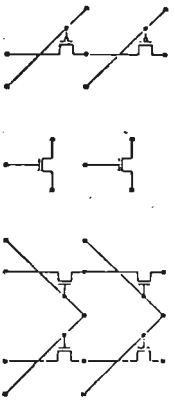


Figure 6.4. Basic core cell layout pattern of GATE FOREST.

necessary with respect to the required current.

HDGA

Documentation: see [Veendrick90].
Customisation: 2 metal layers, programmable vias and contacts.
Core pattern: see figure 6.5.

This very recently developed pattern is designed for efficient implementation of both memory structures and logic. The non-orthogonal gate wiring allows for very dense transistor rows. The master architecture is that of a common Sea-Of-Gates gate array with the same characteristics as GATE FOREST. As with GATE FOREST, the adjacent rows are mirrored for space saving by abutment of p- and n-transistor areas. The power connections occupy predefined tracks in the bottom metal layer. The advantage of the power in the first metal layer is that the necessary connections to the transistors do not introduce blockings in intermediate interconnect layers as for the GATE FOREST. The macro library is of the same complexity as the GATE FOREST, although less extensive. The wiring patterns have again a uniform height to facilitate the automatic placement. Special care has been take to provide sufficient transparency of the predesigned wiring patterns. The approach chosen here is different

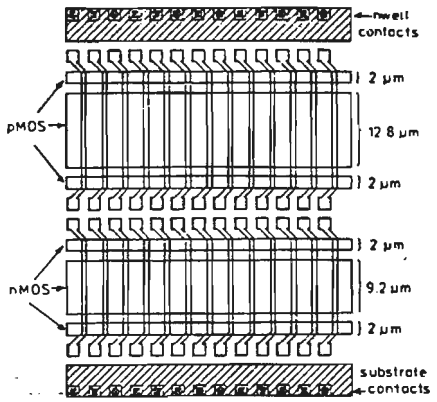


Figure 6.5. Basic core cell layout pattern of HDGA.

from the other gate arrays, by the use of straps for internal macro connections.

OCTAGON

Documentation: see [Koopman90].
Customisation: 3 metal layers, programmable vias and contacts.
Core pattern: see figure 6.6.

A last Sea-Of-Gates pattern is presented by the OCTAGON gate array. It differs from the previous two Sea-Of-Gates patterns by an alternative master and core cell architecture. The core cells are islands of both p- and n-transistors, separated by oxide isolation. Gate isolation is used for individual p- and n-transistors. The master architecture is that of a chess-board by organising the transistor islands such that p- and n-transistors are adjacent. The power connections are predefined and occupy tracks in all three metal layers available. A macro library was not provided, a macro-compiler is under development that will generate design specific macro libraries.

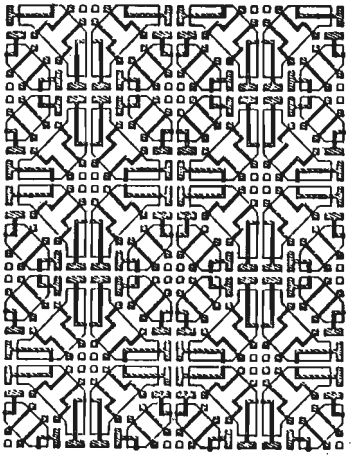


Figure 6.6. Basic core cell layout pattern of OCTAGON.

6.2 Gate array & design characterisation

The gate array master slices discussed in the previous section have all been described in GADL, together with a standardised macro library. The actual GADL descriptions are not discussed here, for a profound discussion of the various aspects of GADL the reader is referred to [Lippens87] and [Boogers90]. For an evaluation of the time required to to derive a gate array and macro library description in GADL, two aspects must be considered. The first aspect concerns the description of the actual structure required to construct correct designs, the second aspect concerns the fine tuning of the GADL descriptions in order to achieve a desired layout design performance. Given the master slice structure, a typical GADL description required about one day for an experienced GADL user. More time is required for the description of a macro library, depending on the size of the library. A typical example with six macros, each having two stamps, may require about one week. The fine tuning of a GADL description requires a specification of the cost function, to guide the detailed interconnect design. This is not a trivial task because cost indications may be supplied at the core cell level, the master slice level and the stamp level. Most of the time required for fine tuning is in the definition of a design implementation to monitor and to deduct the required cost specifications, which are not only specific for the design monitored. However, practical results

indicate that the time spent in fine tuning is in the range of one to two weeks. Concluding, the introduction of a new gate array family takes about one month, which is still neglectable compared to the effort that must be spent if a software redesign would be required.

Gate Array	master (Kbytes)	EQ-sets (bytes)	DR-sets (bytes)
UA6	14	144	0
GA440	3	59	0
TAL004	12	0	1742
GATE FOREST	4	63	0
HDGA	3	1406	45
OCTAGON	16	673	80

Table 6.2. Evaluation of data stored for gate array master slice description.

The amount of data required for a characterisation of the six gate arrays is summarised in table 6.2. The first column lists the amount of data required for the complete master slice description. The data includes the description of the core cells, the master slice floorplan and the predefined wiring on the master slice chip. The description of the equivalence relations and design rule sets are not included in the master slice description, but listed separately in the following two columns. The feasibility of the hierarchical description of the master slice by repetitions of the basic core cell patterns is evident. Also shown is that the amount of data required is related to the core cell structures described. The difference between UA6 and GA440 is caused by the complexity of the core patterns, and the difference between GA440 and TAL004 is caused by the more complex power net structure for TAL004. As expected, regular master slices require in principle less data than the channel gate arrays. For most Sea-Of-Gates patterns, a single core cell is sufficient to describe the complete floorplan, leading to a very small amount of data compared to the size of the gate array. The fact that the core patterns for the OCTAGON master slice are more complicated than those for the HDGA and the GATE FOREST is also reflected in the table. The rather large size of the equivalence table for HDGA is caused by the fact that equivalence

sets are used to describe internal stamp connections with straps.

In order to compare the different macro libraries of the different gate array libraries with respect to the description aspects, a standard library is created. The libraries contain a set of six different macros which are described for the five gate arrays. Due to the specific master slice structures the macros may have more than one stamp that implements the given function. For a correct comparison, the results shown in table 6.3 below, denote the average amount of data required for a single stamp of every macro.

Gate Array	dff	exor	inv	nand2	nand3	nand4
UA6	638	339	135	171	198	283
GA440	371	506	155	207	234	276
TAL004	785	556	247	257	282	299
GATE FOREST	1665	967	377	560	727	806
HDGA	3196	932	547	662	766	838

Table 6.3. Data amounts in bytes stored for a single stamp description.

The summarised results indicate that the amount of data required for the stamp descriptions is kept small by the use of space-graph updates, which are explained in chapter 3. The differences in the amount of data required partly reflect the suitability of the master slice structure to implement the function, and partly the effect of multiple customisation layers. The first two gate arrays, UA6 and GA440, have one customisation layer with fixed contacts to polysilicon and diffusion, whereas the others have two interconnect layers with programmable vias. Due to to the nand structures on the TAL004 gate array, the amount of data can be kept small because most functions fit easily on this gate array. The rather large amount of data required for the stamps on HDGA is caused by the fact that the earlier mentioned internal stamp connections with straps are described by equivalence updates in the stamps.

A last aspect concerns the evaluation of the amount of data required for the description of actually implemented designs.

Gate Array	initial	placed	routed	netlength	bytes per unit netlength
HDGA	3253	3199	12595	3128	3
TAL004	3254	3226	11291	3098	2.6
UA6	3255	3409	14418	2536	4

Table 6.4. Evaluation of design data (in bytes) of Rdc50 for three gate arrays.

An indication is given in table 6.4, summarising some experimental results for a single design, mapped onto three different gate arrays. The results, shown in this table, indicate that the amount of design data is reasonably under control in relation to the total interconnection length of the design. The first column shows the amount of data required for an initial design, before placement and routing. The reduction in data achieved after placement, in case of gate arrays HDGA and TAL004, is caused by the fact that default stamp names are used to indicate that the design is not placed yet. The total net length denoted in the fifth column includes the vias used for interconnections between different routing layers. The number of routing layers penetrated by a via is taken as the length of the via, and is added to the net length. The "bytes per unit net length" relate the total net length with the additional amount of bytes, used for a design after placement, compared to a placed design after routing of the required interconnection patterns. Evaluation of other designs confirm the results summarised in table 6.4. Although larger designs show a light increase of the number of bytes per unit net length, results obtained up to now do not exceed the 10 bytes per unit net length.

6.3 Layout design

For a flexible system as GAS, it must be expected that the flexibility must be paid with a performance loss during layout design. Performance in this context is usually given in terms of cpu-time and utilisation of the available design space. In the ideal case, a truly flexible system should perform equally well for different gate arrays. However, the key issue is how designs pose equally difficult layout design problems for different gate array families. Without claiming to be complete, the following aspects must be considered for a correct

interpretation of the layout design performance:

1) Master slice structures are not equally well suited to implement designs, and may be designed for efficient implementation of specific designs. Two examples that influence a design implementation are the routing transparency of the core cell patterns and the accessibility of the transistor terminals.
2) The degree of difficulty of a design implementation depends on the functionality of the suplied macro library. The wiring patterns of the stamps influence the routing in the same way as the core cell patterns. Badly designed stamps cause unnecessary routing limitations and therefore incomplete designs.
3) Design parameters such as aspect ratio and net weights also determine the completion of a design implementation.
4) If routing guidance is supplied, this guidance must be at an equal level for the gate arrays compared.

Table 6.5 summarises the design data of a medium-sized random logic design, implementing the function $y = 5x + 1$, with the input value (x) limited to eight bits.

The experimental data shows that, despite the differences of the gate array master slice patterns and macro libraries, the performance of GAS is (nearly) the same. This justifies the generality of the presented placement and routing strategies.

For low area utilisations, the cpu-times are dominated by the detailed routing. For all the examples listed in the first column above, the placement and global routing take each about 1 minute of cpu-time, the rest is required for detailed routing. This is caused by the scope enlarging during the detailed routing, which causes long run-times if the design can not be completed. A solution for this problem is to use the combined routing strategy with limited scope enlarging.

Higher area utilisations are achieved at the cost of a very long placement procedure. For the examples listed in the second column, the time required for placement was in the order of 150 minutes cpu-time. This rather poor performance indicates that a better routing guidance is required, but also that the current global placement is too slow for practical use. The design summarised in table 6.6 shows that, if the routing capability dominates the placement of a design, area

Gate array	Cost (statements)	Area Utilisation			
		50%		60%	
		Time (min.)	Routed	Time (min.)	Routed
UA6	5	9	100%	156	91%
GA440	4	8	100%	113	100%
TAL004	6	9	100%	207	99%
GATE FOREST	6	16	98%	217	63%
HDGA	6	8	100%	155	100%

Table 6.5. Evaluation of design implementation for 5xp1 on five different gate arrays with different area utilisation rates. Routing guidance is limited to at most 6 cost statements per core cell pattern. Run times are total design times on an Apollo DN10000.

utilisation up to 95% can be achieved within acceptable time and with still the same routing guidance. The results also indicate another important placement aspect that must be considered for high area utilisations: due to a chosen aspect ratio of the space on the master slice in which the design must fit, and the possible shapes of the stamps, a legal placement solution may not exist.

Another experiment, reported in [Veendrick90] leads to the same conclusion:

> With sufficient routing guidance and a good placement, the routing strategy incorporated in GAS is capable to complete designs that occupy more than 90% of the available design space.

Gate array	Area Utilisation					
	80%		90%		95%	
	Time (sec.)	Routed	Time (sec.)	Routed	Time (sec.)	Routed
UA6	33	100%	28	100%	not placed	
GA440	30	100%	30	100%	36	100%
TAL004	44	100%	not placed		not placed	
GATE F.	70	84%	70	56%	59	56%
HDGA	34	100%	35	100%	35	100%

Table 6.6. Evaluation of a random logic implementation of a five-input, single output exclusive-or on five different gate arrays. Run times are total design times on an Apollo DN10000.

For placement, these results show that incorporation of a standard cell placement strategy is essential in order to achieve high area utilisations without exhaustive fine tuning and with an acceptable cpu-time. This does not imply that the currently incorporated placement strategy is outdated. This placement strategy has already proven itself in situations with differently sized macros [Dewilde86], and will still be necessary for hierarchical design with differently shaped (soft-)macros.

Conclusions and Recommendations

In this thesis a first step towards a generalised approach to gate array layout design automation has been presented. The two major topics addressed concern the general characterisation of gate array families, and the embedding of layout design algorithms. The presented concepts have been implemented in a prototype Gate Array layout design System called GAS, consisting of 41 thousand lines of source code, written in the C-language. The evaluation of GAS with five different gate array families yields the following conclusions and recommendations:

GAS may be seen as a major contribution to the generalisation of the gate array layout design problem. The limited amount of data required for gate array and design descriptions shows that the presented concepts are not only of academic importance, but are also acceptable in an industrial environment.

The time required to adapt GAS to new gate array families is neglectable, because no software re-design is required. This allows for fast prototyping of new gate array master slice structures.

Although the layout design algorithms incorporated in GAS contribute to the provided flexibility, more research is required to achieve area utilisations that are acceptable in an industrial environment. The major problem is identified at the placement phase, where the flexibility is paid by a performance loss in both cpu-time required to achieve a solution, and the routability of the computed solution. It has been argued in this thesis that a placement object function must at

least incorporate net length and wire density penalties, in order to achieve acceptable results. With the current trend in Sea-Of-Gates gate arrays to macro libraries with uniform heights, a standard cell placement seems most appropriate in this context. Recent experiments done at Philips Research Laboratories confirm this observation, and show that the routing concepts incorporated in GAS are capable to solve the interconnect problem for a 10 by 10 bit pipelined multiplier, which was placed with a standard cell placement, and a high area utilisation (more than 90%). Embedding of a standard cell placement algorithm in GAS does not necessarily imply a loss of flexibility if the placement problem is first analysed. With the analysis of the heights of the macros to be placed, in conjunction with the legal positions of the stamps of the macros, it can easily be determined whether the posed problem is a standard cell placement problem or not. Even some not standard cell placement problems can be transformed into one, if some possible stamp choices of macros can be discarded. In this case, the annealing procedure is kept as backup.

Although much effort has been spent to develop a gate array design system which can be used by designers, not all design aspects have been considered in the current implementation of GAS. An important aspect of layout design concerns back-annotation. Back-annotation provides the necessary feed-back of the timing characteristics of the completely laid out design. In the current implementation the design specific wiring is directly available in the design description. This implies that the incorporation of a back-annotation procedure only requires an extension of the technology description for a gate array family for a correct determination of wiring resistance and parasitic capacitance.

In the current implementation of GAS, the routing design is dominated by the detailed routing. Most of the time is spent in the scope enlarging phase, where successively larger areas are searched to route incomplete nets. This phase can be limited by using the combined global and local routing strategy as presented in this thesis. Additional advantages of this combined routing strategy are in the feedback of boundary capacity values and the subsequent detouring of incomplete nets. The routing can be further extended by ripup and reroute techniques, which will directly pay off in the combined routing at both global and detailed routing level.

Although GADL descriptions of the gate array master slice and macro library are easily to derive, more feedback should be provided to users. The main problem is that besides layout patterns, design rules and router guidance must be incorporated in a GADL-description. A solution is to provide an interactive graphical tool, such that data associated with vertices can be shown by different views. Possible views are associated with information stored with edges and vertices of the wiring graph.

Because the routing is grid-based, some problems arise if different wire widths are required in a single design implementation. This can be solved by extending the router with the capability to introduce routing specific design rules which account for the fact that a single wire may occupy more than one track.

A last recommendation is to investigate the possibility to apply GAS for standard cell and even full custom interconnect design. Although this may seem rather surprisingly at first, it must be recognised that gate array interconnect design is in fact the most restrictive case of interconnect design. With the development of the Sea-Of-Gates gate arrays, the resemblance with the standard cell situation is evident.

References

[AMI82] *AMI Europe ULA Manual,* AMI Microsystems Limited, 1982.

[TI83] *TAL Family design Manual,* Texas Instruments Limited, 1983.

[Beresford84] Beresford, R., "Evaluating Gate-Array Technologies," *VLSI Design*, pp. 48-52, January 1984.

[Beunder87] Beunder, M., J. Kernhof, and B. Hoefflinger, "Effective Implementation of Complex and Dynamic CMOS Logic in a Gate Forest Environment," *Proceedings IEEE Custom Integrated Circuits Conference*, pp. 1-4, 1987.

[Boogers90] Boogers, W.A.P.M.P., "Redesign of the GADL compiler," M.Sc. thesis, Eindhoven University of Technology, The Netherlands, 1990.

[Burstein83] Burstein, M. and R. Pelavin, "Hierarchical Wiring of Gate-Array VLSI Chips," *Proceedings European Conference on Circuit Theory and Systems*, pp. 198-202, 1983.

[Dewilde86] Dewilde, P. ed., *The Integrated Circuit Design Book,* Delft University Press, The Netherlands, 1986.

[Frankle86] Frankle, J. and R.M. Karp, "Circuit Placement and Cost Bounds by Eigenvector Decomposition," *Proceedings IEEE International Conference on Computer Aided Design*, pp. 414-417, 1986.

[Gagliardi84] Gagliardi, M.P., "Understanding CMOS Gate-Array Cell Design," *VLSI Design*, pp. 78-80, February 1984.

[Garey79] Garey, R.G. and D.S. Johnson, *Computers and Intractibility,* Freeman, New York, 1979.

[Hsu86] Hsu, C.P., R.A. Perry, S.C. Evans, J. Tang, and J.Y. Liu, "Automatic Layout of Channelless Gate Array," *Proceedings IEEE Custom Integrated Circuits Conference,* pp. 281-284, 1986.

[Jess84] Jess, J.A.G., R.J. Jongen, P.A.C.M. Nuijten, and J.C. Bu, "A Gate Array System Adaptive to Many Technologies," *Proceedings IEEE International Conference on Computer Design : VLSI in Computers,* pp. 338-343, 1984.

[Koopman90] Koopman, R.H.J., R. Peset Llopis, and H.G. Kerkhoff, "Digital CMOS Sea-of-Gates Core Cells and Master Images," *Proceedings 16th European Solid-State Circuits Conference,* pp. 245-248, 1990.

[Korn82] Korn, R.K., "An Efficient Variable-Cost Maze Router," *Proceedings 19th Design Automation Conference,* pp. 425-431, 1982.

[Kubosawa87] Kubosawa, H., G. Goto, S. Tsutsumi, Y. Suehiro, and T. Shirado, "Layout Approaches to High-density Channelless Masterslice," *Proceedings IEEE Custom Integrated Circuits Conference,* pp. 48-51, 1987.

[Lee61] Lee, C.Y., "An Algortihm for path Connections and its Applications," *IRE Transactions on Electronic Computers,* pp. 346-365, September 1961.

[Leenstra83] Leenstra, J., *Designers Guide to Gate Arrays,* Twente University of Technology, The Netherlands, 1983.

[Li84] Li, J. and M. Marek-Sadowska, "Global Routing for Gate Array," *IEEE Transactions on Computer-Aided Design, Volume CAD-3, No. 4,* pp. 298-307, October 1984.

[Lippens87] Lippens, P.E.R. and A.G.J. Slenter, "GADL: A Gate Array Description Language," M.Sc. thesis, Eindhoven University of Technology, The Netherlands, 1990.

[Nuijten85] Nuijten, P.C.A.M., "Hierarchical Wire Routing of Gate Arrays," M.Sc. thesis, Eindhoven University of Technology, The Netherlands, 1985.

[Otten84] Otten, R. H. J. M. and L. P. P. P. van Ginneken, "Floorplan design using simulated annealing," *Proceedings International Conference on Computer Aided Design*, pp. 96-98, Santa Clara, 1984.

[Parng89] Parng, T. and R. Tsay, "A New Approach To Sea-of-Gates Global Routing," *Proceedings IEEE International Conference on Computer Aided Design*, pp. 52-55, 1989.

[Rubin74] Rubin, F., "The Lee Path Connection Algorithm," *IEEE Transactions on Computers, Volume C-23, No. 9,*, pp. 907-914, September 1974.

[Saeijs86] Saeijs, R.W.J.J., "Simultaneous Placement and Global Routing for Gate Arrays," M.Sc. thesis, Eindhoven University of Technology, The Netherlands, 1986.

[Soukup78] Soukup, J., "Fast Maze Router," *Proceedings IEEE Design Automation Conference*, pp. 100-102, 1978.

[VLSI87] Staff, VLSI Systems Design, "A Survey of Automatic Layout Software," *VLSI Systems Design*, pp. 78-89, April 1987.

[Ting83] Ting, B.S. and B.N. Tien, "Routing Techniques for Gate Array," *IEEE Transactions on Computer-Aided Design, Volume CAD-2, No. 4*, pp. 301-312, 1983.

[Veendrick90] Veendrick, H., D. van den Elshout, D. Harberts, and T. Brand, "An Efficient and Flexible Architecture for High-Density Gate Arrays," *Proceedings IEEE International Solid-State Circuits Conference*, pp. 86-88, 1990.

Index

Biography

André Slenter was born on July 26, 1961. He studied electrical engineering at the Eindhoven University of Technology, the Netherlands, and graduated with honors on October 24, 1985. Since then he has been working on a Ph.D. degree in the Design Automation Section until March 31, 1990. He is now with Philips Research Laboratories in Eindhoven.

His interests are computer aided design, layout design for digital and analog intergrated circuits, and software research for parallel and real-time applications.

STELLINGEN

bij het proefschrift van

André G.J. Slenter

A GENERALISED APPROACH TO GATE ARRAY LAYOUT AUTOMATION

1) Gezien de ervaringen met GADL betreffende de terugkoppeling naar de gebruiker van de ingevoerde gegevens, zal de vervanging van GADL door een interactief, grafisch invoerprogramma de gebruikersvriendelijkheid van GAS verder verbeteren.

   [Dit proefschrift]

2) Om optimaal gebruik te kunnen maken van parallellisme in een programma met een interne, globale data structuur, die intensief gebruikt en gemodificeerd wordt, verdient een "shared-memory" computer architectuur de voorkeur boven een "distributed-memory" architectuur.

   [Dit proefschrift]

3) Gezien recente resultaten dient de stelling dat de flexibiliteit van GAS ten koste gaat van de maximaal haalbare bezettingsgraad van een master-slice, heroverwogen te worden.

   [Dit proefschrift]
   [Veendrich, H., D. van den Elshout, D. Harberts, and T. Brand, "An efficient and Flexible Architecture for High-Density Gate Arrays", Proceedings IEEE International Solid-State Circuits Conference, pp. 86-88, 1990]

4) De toepassing van CRACKER als detailed routing algoritme voor GAS is slechts zinvol, indien de convergentie van het top level iteratie proces kan worden afgedwongen.

   [Gerez, S.H., "Local Wire Routing by Stepwise Reshaping", proefschrift TU Twente, 1989]

5) Een design data management systeem met een procedurele interface voor de design tools is, gezien vanuit software management oogpunt, een slecht concept.

[Dewilde, P., ed., "The Integrated Circuit Design Book", Delft University Press, The Netherlands, 1986]

6) Het succes van CAD-onderzoek zal in steeds grotere mate bepaald gaan worden door een goed software management beleid en de implementatie van dit beleid.

7) De toepassing van expert-systemen voor CAD moet worden gezien als een gebrek aan vertrouwen om tot een adequate en formele oplossing van de ontwerpproblematiek te komen en dient daarom te worden afgewezen.

8) De behoefte aan dienstplichtige academici, afgestudeerd in een technische discipline, is niet gebaseerd op het niveau van de te verrichten technische werkzaamheden en is daarom een niet goed te keuren verlangen om aan onderbetaalde werknemers te komen.

9) Een adequate en milieuvriendelijke oplossing voor het toenemende forensenverkeer kan alleen worden gevonden in een aanzienlijke uitbreiding van het metro-netwerk.