Stellingen
behorende bij het proefschrift

"Context-Driven Channel Routing"

door Patrick Groeneveld

1. Een 'klassiek' kanaalbedradingsprobleem met m kolommen is alleen dan niet oplosbaar als het m netten bevat die allen het kanaal kruisen en waaronder zich netten bevinden waarvan de beide aansluitingen in verschillende kolommen voorkomen

2. Sea-of-Gates chips zijn niet zozeer interessant vanwege de mogelijkheid om variabele-breedte kanalen te gebruiken bij een rij-georiënteerde ontwerpstijl, maar zeker ook vanwege de mogelijkheid om efficiënt reguliere structuren te implementeren.

3. Een layout wordt vaak subjectief beoordeeld op de aanwezigheid van lege ruimten of merkwaardige bedradingspatronen; dit zegt echter bijzonder weinig over de technische kwaliteiten van het patroon.

4. De nadelen van slicing-structuren zijn eerder psychologisch als praktisch.

5. Sponsoring vertoont sterke overeenkomsten met drugsgebruik: na een gewenningsperiode loopt de kwaliteit van het gebodene doorgaans terug, en er ontstaat een gevaarlijke afhankelijkheid.

6. Het gedogen van lease-, LPG- en diesel-auto's is in tegenspraak met het regerings beleid om het autogebruik te beperken.

7. Het minieme aantal vrouwen in technische studies toont aan dat de emancipatie in Nederland nauwelijks van de grond komt.

8. Voor een gezonde ontwikkeling van de Nederlandse economie is het noodzakelijk om het aantal meesters in de rechten te beperken.

9. Voor de veel gehoorde kreet "techniek: daar snap ik helemaal *niets* van!" moet men zich diep schamen.

10. De dienstplicht kost de samenleving aanzienlijk meer dan de post op de defensiebegroting doet vermoeden.

11. Het misleidende effect van verpakkingen wordt zeer duidelijk aangetoond door het succes van de CD: voor hetzelfde muzikale product, maar eleganter verpakt, betaald iedereen grif 15 gulden meer.

12. Daar vele uit het Engels geïmporteerde termen, zoals bijvoorbeeld 'green-peace' en 'Dutch', in de oorspronkelijke taal tamelijk oudbollig zijn of zelfs een negatieve klank hebben, dient het gebruik ervan in de Nederlandse taal te worden vermeden.

13. De PC is het opium van het volk.

CONTEXT-DRIVEN CHANNEL ROUTING

Patrick Groeneveld

CONTEXT-DRIVEN CHANNEL ROUTING

Proefschrift
ter verkrijging van
de graad van doctor
aan de Technische Universiteit Delft,
op gezag van de Rector Magnificus,
prof. drs. P.A. Schenck,
in het openbaar te verdedigen
ten overstaan van een commissie
aangewezen door het College van Dekanen
op donderdag 6 juni 1991 te 14.00 uur

door

Patrick Robbert Groeneveld

geboren te Heemskerk
electrotechnisch ingenieur

Dit proefschrift is goedgekeurd door de promotor prof.dr.ir. R.H.J.M. Otten.

Samenvatting

Dit proefschrift behandelt het bedradingsprobleem op een complexe geïntegreerde schakelingen (IC's). Er wordt een aantal algoritmen gepresenteerd voor het automatisch genereren van een bedradingspatroon.

Een moderne geïntegreerde schakeling bestaat uit een zeer groot aantal onderdelen. IC's met meer dan 1.000.000 transistoren zijn tegenwoordig al geen uitzondering meer. Een bedradingspatroon moet ervoor zorgdragen dat deze onderdelen op een zinvolle wijze met elkaar verbonden worden. Het converteren van een schematische beschijving van de verbindingen tussen de onderdelen in een fysiek bedradingspatroon op de chip is daarom geen sinecure. Het doel van het ontwerpproces is om de schemabeschrijving op een zo klein mogelijk chipoppervlak te realiseren. Hierbij moet tevens aan een aantal procesvoorschriften en elektrische specificaties worden voldaan. Op de meeste IC's neemt het bedradingspatroon meer dan de helft van het oppervlak in beslag. De kwaliteit van het gebruikte bedradingssysteem zeer belangrijk, aangezien de fabricagekosten vrijwel uitsluitend afhangen van het chipoppervlak (en niet van het aantal transistoren)

De chip bestaat uit een aantal onderdelen, die ieder voor zich weer uit andere onderdelen bestaan. Alleen dankzij deze hiërarchie blijft het ontwerp te overzien. Op elk hiërachisch niveau wordt een aantal onderdelen samengevoegd tot een nieuw onderdeel. In de meeste gevallen gebeurt dit samenvoegen uitsluitend door het genereren van een bedradingspatroon tussen de onderdelen. Ook dit z.g. *place and route* process valt uiteen in een aantal stappen. Ten eerste worden de onderdelen geplaatst. Vervolgens wordt een globaal pad gezocht voor elke verbinding in het schema. Dit pad beschrijft grofweg de manier waarop de draad tussen de onderdelen door laveert. Hierna worden de bedradingsruimten opgedeeld in zogenaamde *kanalen*. Als laatste stap worden de uiteindelijke bedradingspatronen per kanaal gegenereerd door een *kanaalbedrader*. De opdeling in kanalen is zeer gunstig omdat kanaalbedraders als enige kunnen garanderen dat alle draden ook daadwerkelijk gerealiseerd kunnen worden. In dit proefschrift worden een aantal algoritmen gepresenteerd voor het automatisch genereren van het fysieke bedradingspatroon in een kanaal. De algoritmen zijn specifiek

gericht op een goede aanpassing van het kanaal met de omringende onderdelen.

Veel kanaalbedraders genereren bedrading op een gaaspatroon: het z.g. *grid*. In hoofdstuk 3 wordt een nieuw algoritme gepresenteerd dat gebruikmaakt van een aantal contouren in plaats van het grid. Hierdoor wordt een betere afstelling op de fabricagetechnologie verkregen. Niet alleen kunnen hierdoor kompaktere bedradingspatronen gegenereerd worden. Het concept biedt ook de mogelijkheid om één van de meest notoire kanaalbedradingsproblemen grotendeels te voorkomen. In hoofdstuk 2 wordt het basisconcept van het algoritme geïntroduceerd. Bovendien worden de eigenschappen ervan vergeleken met die van conventionele kanaalbedradingsalgoritmen. Een grid-gebaseerde versie van het algoritme wordt gepresenteerd in hoofdstuk 4. Deze laatste ontwikkeling past uitstekend in de context van een nieuw type chip, de z.g. *sea-of-gates*. In hoofdstuk 5 wordt vervolgens ingegaan op het samenvoegen van de kanalen tot een groot bedradingspatroon.

Aan de opdeling in kleinere bedradingsgebieden kleven ook een aantal nadelen. Zo kruist het pad van vele draden een aantal kanalen. Door de onafhankelijke bedrading van de kanalen is het mogelijk dat deze draden onnodig door elkaar lopen en verward raken. Hierdoor wordt ruimte verspild. In hoofdstuk 6 wordt een nieuw algoritme voor het ordenen van deze draden gepresenteerd.

Tenslotte wordt in hoofdstuk 7 een krachtig plaatsings- en bedradingssysteem gepresenteerd. In dit systeem worden veel van de eerder gepresenteerde algoritmen toegepast. De resultaten van tests met een aantal gestandaardiseerde testciruits wijzen erop dat dit systeem de vergelijking met internationaal gerenommeerde systemen glansrijk kan doorstaan. Zowel het concept van de kanaalbedrader als de methode om de draden te ordenen worden ondertussen al industrieel toegepast voor de fabricage van grote IC's.

Contents

Chapter 1

Introduction

The density of integrated circuits has risen dramatically over the past quarter of a century. Currently chips with well over 1.000.000 transistors are no exception anymore. This puts them among the most complex man-made structures. With the increasing number of components the classification of IC's has changed over the years: from *large* scale integration (LSI) in 1970 over *very large* scale integration (VLSI) around 1980 into the *ultra large* scale integration (ULSI) of today. A clever copywriter is required to invent a new superlative for the next generation.

This nomenclature problem pales into insignificance beside the sheer scale of an IC. Manual design of the tens of thousands of wires and components on an IC is simply not feasible. To help meet the challenge, equipment is used which consists of the same components as those which it is designing: computers. The design automation of integrated circuit involves the development of computer programs. These programs are the *tools* of chip designer. Our ultimate goal is to find *the* tool which designs the chip without manual intervention. This dream is called *silicon compilation*. Philosophically one could say that we are looking for ways to make computers design new computers. In a sense this kind of reproduction is the major characteristic of life, which puts us on the same trail as Dr. Frankenstein. Fortunately we are not that far yet, which leaves a real challenge with many interesting topics.

As with many complex problems, IC design is attacked by breaking it up into a number of smaller problems. These problems are then solved step by step. With each step more detail is taken into account, a process which is generally called *stepwise refining*.

Another way to make the problem more manageable is by partitioning it into a number of smaller ones which are solved independently. Introducing *hierarchy* is the most general and widely spread used method to reduce complexity. The characteristic

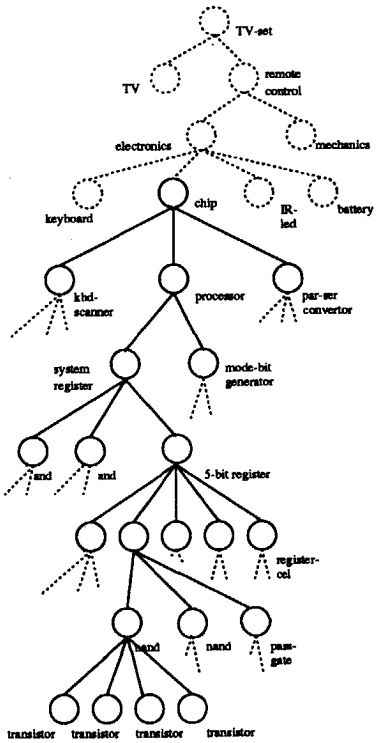

Figure 1: A (pruned) hierarchy tree of a television remote control set. With large integrated circuits many levels of hierarchy are created to keep up with the complexity of the circuit. All components on a chip are made up of transistors, which form the atomic modules.

of hierarchy is a recursive tree structure of *modules*. Any module consists of sub-modules, each of which can again be composed of sub-modules. Modules without sub-modules are called *atomic modules* or *leaf-modules*. Hierarchy can be represented in a tree (see figure 1). The aim of hierarchical partitioning is to create smaller, but similar, problems which can be solved closer to the optimal. This ***divide and conquer*** strategy is especially essential for the interconnection problem we address in this thesis.

Underlying a computer program is a formally described method called an ***algorithm***. In this thesis we focus on algorithms for the automatic interconnection of the components on an integrated circuit. This usually forms the very last step in an IC

	full-custom	semi-custom
row-based	standard-cell	gate-array
general-cell	macro-cell	sea-of-gates

Figure 2: Design style versus implementation style.

design trajectory. Obviously, the properties of the chip are of considerable influence on this process. The next section deals with the major chip types and the styles which are used to implement a circuit on them. In the final section of this chapter we will describe an overview of the rest of the thesis.

1.1 Integrated Circuits

On an IC only a limited variety of components can be implemented. In a CMOS fabrication technology two types of transistors are present, making up the entire arsenal of active components. Furthermore, a set of interconnection patterns (in a form which resembles wires) can be created to connect these components. The ability to implement other, passive components, such as resistors, is usually limited to the use of parasitical side-effects of the basic components. The variety of the leaf-modules is therefore very limited, in contrast to other man-made structures of similar scale and complexity. It is the sheer number of components and the ability to connect them properly which makes an IC work as intended.

The components are implemented by alternated deposition and selective removal of materials on a silicon substrate. This process can involve over 20 different masks to apply the sub-patterns on the silicon. The ensemble of masks, commonly called the *layout*, is the lowest level of control we have over what is implemented. A chip which was designed with full control over all masks of the production process is called *full-custom*.

The large number of mask make the production process of a chip an expensive and time consuming process. This can only be justified by sufficient production quantities or strict performance requirements. *Semi-custom* chips provide a more time- and cost-effective alternative. On a semi-custom chip the transistor pattern is fixed. The user only has control over a limited number of interconnection masks to connect the transistors. The production process can be standardized to a large extent. This reduces the processing cost and time. Semi-custom chips are suitable for smaller production volumes. There is, of course, a price to pay. The fixed pattern of the components imposes a number of restrictions, making the layout less area-efficient

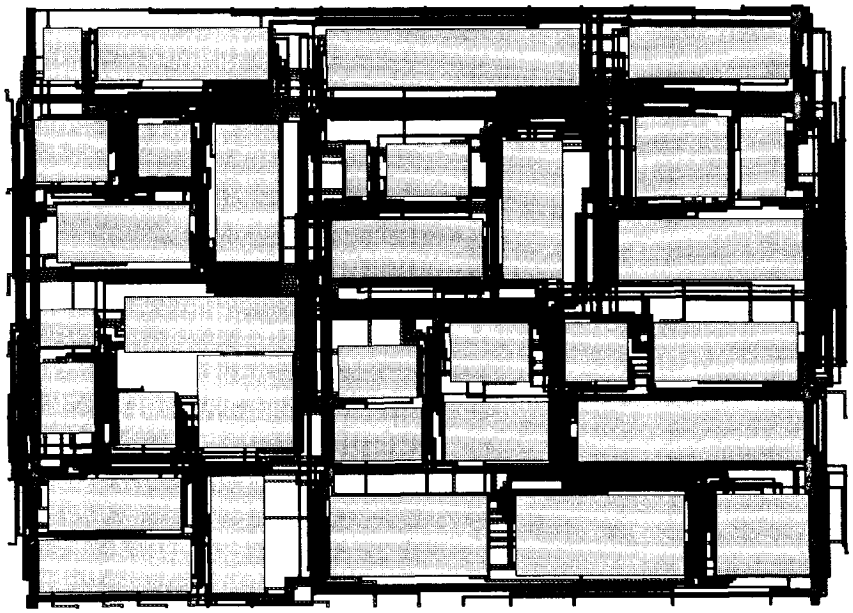

Figure 3: An example of the general-cell design style. The wires are routed in the areas around the modules. The contents of each module (not shown in this figure) can also be designed in the same general-cell manner.

and its electrical characteristics less optimal.

The jargon of the IC design community contains many two-word slogans (see figure 2). Full-custom and semi-custom are *implementation styles*, determining a technological (hardware) limit within which the layout of a circuit can be implemented. In contrast to that we also distinguish *design styles*, which specify self-imposed (software) constraints on the layout design. A design style captures the basic methodology which is used to map the components on the chip.

The approach in which the modules may have any size and various shapes has many names: *general-cell*, *macro-cell* or *building-block*. It is the most general design style because many classes of modules (such as RAM's, random logic, etc.) can be dealt with. Therefore it is very suitable for a hierarchical layout methodology in which each of the modules was routed previously at a lower hierarchical level. In general, the area occupied by the modules cannot be used for interconnect which confines the wiring to the areas between the modules. Figure 3 shows an example of full-custom chip in the general-cell design style.

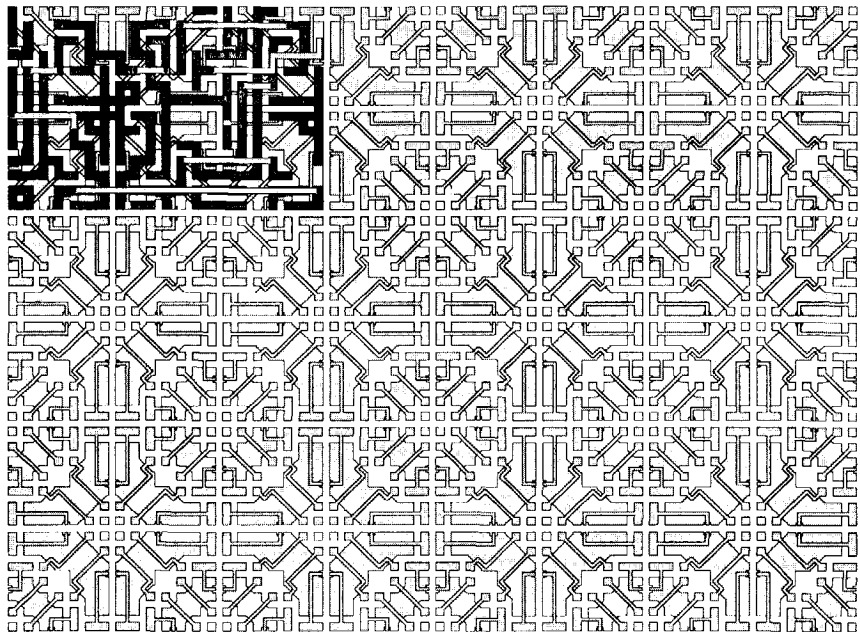

Figure 4: A portion of a sea-of-gates chip ([45]). In this case the transistors are arranged in an octagonal structure. In the left top corner a a full-adder is implemented this image.

In the *row-based* design style the modules are placed in horizontal rows with oblong routing areas (called *channels*) in between. The modules are often retrieved from a library of pre-designed *standard cells.* On a full-custom chip this style is therefore called *standard-cell.* For semi-custom chips the arrangement in rows is the most popular. These chips are called *gate-arrays.* The row-based approach has certain algorithmic advantages due to the uniformity of the modules. This makes them easier to design. On the negative side, a row-structure enforces an artificial framework on the layout for which not all circuits are suitable. The fixed size of the channel in the row-based approach causes very area-inefficient implementations of regular logic structures such as RAM's. Therefore it is mainly used for unstructured random logic. Another predominant property of the row-based layout design style is its nonhierarchical nature. The uniform module size prevent any hierarchy in the layout. The hierarchical circuit description of the chip must be 'flattened' before layout design.

Sea-of-gates chips have become increasingly popular during the past few years.

These semi-custom chips consist of a regular pattern of transistors which can be customized by a few aluminium interconnect masks, much like the traditional gate-array. Unlike the gate-array, however, there are no pre-defined routing areas on a sea-of-gates. The wires are laid over transistor cells belonging to the circuit and, if necessary, over unused transistors. This is the only (but significant) difference between these semi-custom design styles. It makes sea-of-gates chips a cheap and fast alternative for a 'full-custom-like' general-cell approach.

1.2 Overview

The automatic generation on interconnection layout on a modern ULSI circuit involves many hundreds of nets on each level of hierarchy. It a very important design step because it is not uncommon that about 50% of the chip area is consumed by wires. The large amount of wires makes a divide and conquer strategy almost inevitable. It is not feasible to attack the wiring area in one piece. In most routing systems, we can distinguish the following steps:

- Place the modules on the chip's surface.
- Perform *global routing* by finding a topological path for the wires of each net.
- Define *routing regions* by breaking the area around the modules into smaller areas: the *channels*.
- Perform *detailed routing* in each of the regions, one region at the time.

This thesis focuses on the last step, in which the actual wire pattern is generated in a small region. There is always an area of tension between the physical context of the region and the algorithm which is used to generate the the wire pattern in the region. Detailed routing algorithms usually operate on an abstract model of the real-life area (such as a graph). Their scope is generally limited to a specific problem, optimizing one aspect and disregarding the rest. Within their scope, they might produce optimal results, but this result does not necessarily represent a good layout.

The algorithms which we will present in this thesis share a common theme: they are especially targeted towards the embedding of the routing region with its surrounding context. In the following chapters, this 'context-driven' concept will be worked out in two ways. First of all, the major part of this book is dedicated to algorithms which are better equipped to handle various aspects of the routing region. Secondly, we propose a methodology to 'guide' the detailed router, by supplying information which is outside its scope. For example, consider the ensemble of smaller routing

regions which together forms the interconnect pattern of the entire chip. Each region must fit precisely between the surrounding modules and routing regions, just like the pieces of a jigsaw puzzle. The scope of a detailed router is limited to one region only. Therefore each region is routed without taking into account its context, which will generally lead to some inefficiency. Especially near the boundaries of the routing regions peculiar and nonoptimal layout patterns can be expected. For this reason, the boundaries of the routing region can usually be identified visually on the chip, like the sawtraces in an assembled a jigsaw puzzle. These 'scars' are an almost inevitable drawback of the divide and conquer strategy. Our goal is to avoid them by context-driven detailed routing.

The major part of this thesis dedicated to *channel routers*, which form a special and very powerful class of detailed routers. In chapter 2 we will first define the channel routing problem. Associated with the channel routing a number of typical problems emerge. Before addressing our approach to solve these problems, the foremost channel routing algorithms and their characteristics will be described. We will prove that our approach will find a solution if one exists. None of the 'traditional' channel routers can give a similar guarantee. The approach presented in chapter 2 is the backbone of two channel routers which will be described in in chapters 3 and 4.

The majority of channel routers use a 'grid' as framework. This framework is a rather abstract and inaccurate model of the layout on the chip. It requires, for instance, a uniform wire width of all wires and it requires that all wire segments must be aligned on the grid. In chapter 3 we will present a gridless channel router, which uses a set of contours instead of the grid as framework. In chapter 4 we will apply our approach to a channel router which is especially suitable for sea-of-gates chips. In this channel router we attempt to merge the best aspects of a channel router and a maze router.

The problem of region definition will be addressed in chapter 5. We will describe the general approach to arrange the channels such that 100% completion can be achieved for the entire circuit. We will also present a new macro-cell approach for sea-of-gates routing, based on the channel router presented in chapter 4.

In chapter 6 we present a method to prevent unnecessary wire twisting which results from the subdivision into smaller (independent) routing areas. In chapter 7, we conclude by presenting a comprehensive general-cell placement and routing system. Many of the algorithms described in this thesis were applied in this practical system.

Chapter 2

The channel routing problem

Channel routing is a special case of the routing problem in which a net list must be converted into a wire pattern in a strip between two modules. The strip is called *channel* due to its oblong shape and the fact that there are no obstructions inside. The terminals are placed exclusively on the boundary of the channel. The width of the channel can be adjusted according to the intensity of the wiring in the channel, just like a river expanding its boundaries in the rain season. In this chapter we will present our basic methodology for channel routing problem, which is primarily aimed at ensuring the routing of all nets.

Before addressing the actual channel routing problem, we will first examine its underlying basics. In 2.1.1 we will briefly define the relevant process design rules the wiring pattern should comply with. In 2.1.2 we will describe the *grid*, which is the underlying model of many detailed routers.

A number of - sometimes overlapping - subclasses of channel routers can be distinguished. Figure 5 shows an attempt to capture the classes in a Venn diagram. Each class has a slightly different problem formulation or routing strategy. In 2.2.1 we will first give a general formulation of the channel routing problem which covers most of the subclasses. Then, in 2.2.2 a more restricted formulation is given for the grid-based reserved layer model. The majority of the work which was published on channel routing uses that 'classical' model.

A cornerstone of our approach is the allocation of a reserved area around each terminal, which we called a 'territory'. In 2.4 we will define our *territory framework*. Then we will prove the conditions for which a solution can be found in our framework. By applying our territory framework for the classical channel routing problem, we are able to make a strong statement about the solvability of an instance of the classical channel routing problem. In 2.4.3 we will prove that an instance of the classical channel routing problem is solvable if it can be routed in our territory framework.

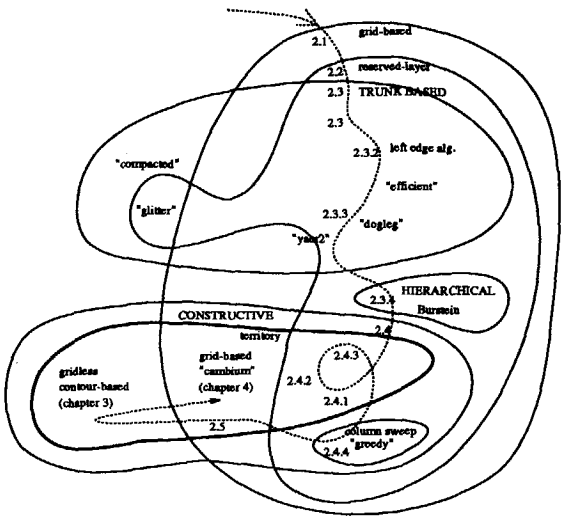

Figure 5: A Venn diagram showing the relation of our approach (the fat line) with other classes of channel routers. The structure of this chapter is indicated by the dotted line. The three sets indicated by capital letters indicate primary methodologies for channel routing. The sets 'grid-based' and 'reserved-layer' mark concepts according to which the channel is modeled. For convenience, only the two-layer channel routing problem is depicted.

Prior to that, in section 2.3, we will first review the major characteristics of the 'traditional' trunk-based channel routers. Many trunk-based channel routers are not able to guarantee completion for solvable instances of the classical channel routing problem. Figure 5 shows the structure of this chapter as a path through a Venn diagram.

2.1 Preliminaries

2.1.1 Mask layout

The output of a detailed router is a mask pattern suitable for a specific VLSI fabrication technology. A set of *design rules* forms the geometrical 'cookbook' of the chip. It contains structural rules specifying which mask patterns must be combined to create an (electronic) component. A MOS transistor, for instance, is created by crossing a

polysilicon mask pattern with a diffusion mask pattern. Apart from the structural rules the design rules also contain a number of numerical rules which specify the minimum (or maximum) widths and separations of various structures. In many cases the numerical rules resulted from technical imperfections or limitations such as the relative alignment of the masks or the resolution of the etching process.

In a CMOS fabrication technology the majority of the masks deal with the creation of the transistors and associated patterns (such as wells). Within the scope of this thesis these masks are less relevant because we can assume that the transistors are 'hidden' in the modules. In the following subsections we will describe the the wire patterns which are specifically relevant for the interconnection problem.

Layers

Interconnection patterns can be formed in a limited number of masks. They are arranged in a sandwich-like layer structure with insulating glass in between. Some of the layers (the *metal*[1] layers) are exclusively used for interconnection patterns, others may have a dual role. An example of the latter is polysilicon (commonly abbreviated as *poly*) which primarily forms the gates of the transistors but which may also serve as a wire. Its use, however, is restricted to shorter wires because the resistance of the poly layer is in most cases much larger than the resistance of the dedicated interconnect layers. Earlier processes had only two layers of interconnect, indicated as *metal* and *poly*. Currently all modern VLSI fabrication technologies provide more than one layer of metal interconnect. The following numerical design rules are relevant for our problem:

$\mathcal{S}_{layer}$ The minimum spacing in *layer*. This specifies the minimum separation between two unrelated objects (wires) to prevent a short-circuit.

$\mathcal{W}_{layer}$ The minimum wire width in *layer*. There is no upper limit on the wire width.

Typically, the spacings and widths are different for each layer. Apart from these geometrical rules, there are electrical parameters related to the layers. A formula to calculate the minimum width given the current load of a wire is given to prevent premature aging due to electromigration. In order to prevent premature aging due to electromigration, the minimum wire width given the current load is prescribed.

Contacts

A connection between two different layers is made by a *via* pattern (also called *contact*). Due to alignment imperfections the pattern of a via must be extended beyond

[1] Usually aluminium

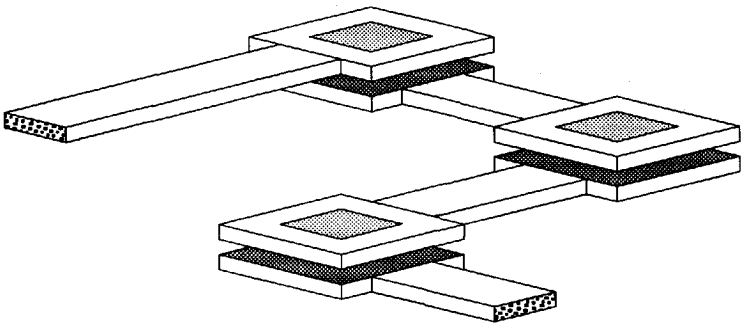

Figure 6: A staircase pattern is required to connect between non-adjacent layers. Stacked contacts are not allowed in most processes.

the actual contact 'hole'. For routing purposes the size of the via pattern in both layers it is connecting, is the most relevant parameter:

$\mathcal{V}_{l_1,l_2}$ The total width of the contact hole pattern in layer l_1, given that it connects the layers l_1 and l_2. A via is typically wider than a wire: $\mathcal{V}_{l_1,l_2} \geq \mathcal{W}_{l_1}$ and $\mathcal{V}_{l_2,l_1} \geq \mathcal{W}_{l_2}$.

In contrast to a printed circuit board, the vias in most VLSI technologies only connect between adjacent masks. It is generally not possible to 'stack' contacts on top of each other. Any electrical contact between layer 4 and layer 1, for instance, must therefore 'staircase' through layers 2 and 3 (see figure 6).

Apart from strict geometrical design rules which must be complied with, there are also some patterns which are not forbidden but which should be avoided. Usually the reliability of the circuit is affected. Vias reduce the reliability in two ways. First of all, a via is a connection between two adjacent masks. A small alignment error of the masks during the production process primarily affects vias. Secondly, a via is really a 'hole' in the insulating glass layer. The aluminium wire of the top layer bends down in the hole to connect to the lower layer. It is more likely that the wire cracks at this point.

2.1.2 Modeling the routing region

The *routing region* is the free layout area which is available for wires. To describe the routing region, we can view it as a collection of points in a 3-dimensional space which is spanned by Cartesian coordinate axes. We can orientate the x- and the y-axis such

that they span the plane of the silicon. For convenience, we will call a line along the x-axis *horizontal* and a line along the y-axis *vertical* for the remainder of this book. Along the z-axis there are only very few layers which need to be modeled. If there are ν interconnect layers in the process, we can model the layers by the planes $z = 1, z = 2, \ldots, z = \nu$. The 'wiring capacity' in the first two dimensions is a few orders of magnitude higher than in the z-direction, due to the limited number of layers. In this way the interconnect problem is rather 'flat', giving it a *quasi-planar* flavor. This planarity is exploited by many algorithms for layout generation. Each point (x, y, z), $x, y \in \Re, z = 1, 2, \ldots, \nu$ in this space either belongs to the routing region, or it does not. Let the set P denote the collection of points belonging to the routing region. For the definition of the routing area for the channel routing problem the following definitions are relevant.

Definition 1 ($\vec{u}$-convex regions) *Let $\vec{u}$ be one of the unit vectors which span up a 3-dimensional Euclidean space. A set P of points in this Euclidean space is called* $\vec{u}$-convex *if for each pair of points $s \in P, t \in P, t = \mu\vec{u} + s$ (the points s and t are on a line segment with $\vec{u}$ as direction vector) all convex combinations $\lambda s + (1 - \lambda)t$, $0 < \lambda < 1$ are also in P.*

Definition 2 ($\vec{u}$-convex extremes) *A point v in a $\vec{u}$-convex set P is called a* lower $\vec{u}$-extreme *of P if there is no vertex $s \in P$ with $s = \lambda\vec{u} + v, \lambda < 0$. Similarly, an* upper $\vec{u}$-convex extreme *is a point in P for which no vertex $s \in P$ exists with $s = \lambda\vec{u} + v, \lambda > 0$.*

For example, the area in figure 8 (on page 15) is vertically convex but not horizontally convex. If an area is horizontally convex, as well as vertically convex, it is called *isoconvex*.

The raw mask patterns which can be described by subsets in the space, are difficult to handle. Their manageability can be improved by capturing them in a model. All unnecessary details of the mask layout should be removed such that only the essence of the interconnection pattern remains. Many of the process-specific design rules can be hidden in this way. It can be compared to the work of abstract artists, who attempt to capture the essence of the matter in the artwork, rather than overwhelming the spectator with many less relevant side-effects. The basic requirement for this model is that it must be possible to translate the wire pattern it contains into a valid mask pattern. To capture obstacles in the model, even the opposite translation (mask $\rightarrow$ model) must be possible. Any simplification or abstraction inevitably involves a restriction in the solution space.

A first restriction deals with the resolution of the routing area. In principle, the routing area has an infinite resolution. There is, however, not much point in specifying

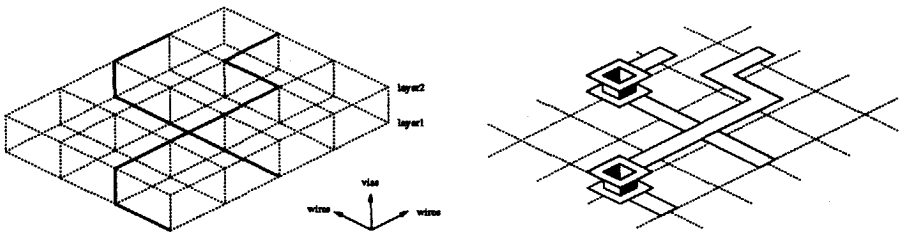

Figure 7: A wire pattern captured in a three dimensional grid graph with two layers (left) and the corresponding physical wire pattern (right).

feature sizes with nanometer resolution if the width of the wires is in the order of a few micrometers. In practice, all distances and feature sizes are therefore expressed in a unit called λ (lambda) [53, page 48]. In the C3TU process we currently use at Delft University of Technology, for instance, the width of a transistor gate is 8λ (1.6μ). With that, the resolution becomes discrete and the solution space of the problem shrinks considerably. For the routing problem it is preferred to have a representation of the routing space with an even lower resolution. In principle the resolution should be adapted to to the smallest useful geometrical object. For detailed routing, these objects are wire segments.

A second, commonly used simplification is to confine the layout to features aligned with the Cartesian coordinate axes. This *Manhattan style* layout simplifies the layout problem considerably, especially the implementation of algorithms.

These two restrictions, combined with the quasi-planar nature of VLSI technology make a *grid graph* a suitable representation of the layout area. A grid graph $G(V, E)$ is a rectilinear $(m \times n \times \nu)$ 3-dimensional array of vertices. Each vertex $V(c, r, l)$, $c = 1, 2, \ldots, m$, $r = 1, 2, \ldots, n$, $l = 1, 2, \ldots, \nu$ corresponds to a specific regular position (x, y, z) in the routing area. For each vertex $V(c, r, l)$, c denotes its 'column' number (an x-position), r denotes its 'row' number (an y-position) and l denotes its layer. The spacing between two adjacent grid rows or columns is called the *mesh width* or *grid spacing*. An edge exists between all adjacent vertices in the same row, column and layer.

A *wire* is a connected subset of $G(V, E)$. All vertices which belong to the wire are electrically equivalent ('connected'). A wire is converted into a mask pattern by translating its edges into mask segments. Each edge $\{ V(c, r, l),\ V(c+1, r, l) \}$ maps to a horizontal mask segment, while an edge $\{ V(c, r, l),\ V(c, r+1, l) \}$ maps to a vertical mask segment. An edge $\{ V(c, r, l),\ V(c, r, l+1))\}$ maps to a via between layers l and $l+1$. Figure 7 shows a wire layout and its associated grid graph.

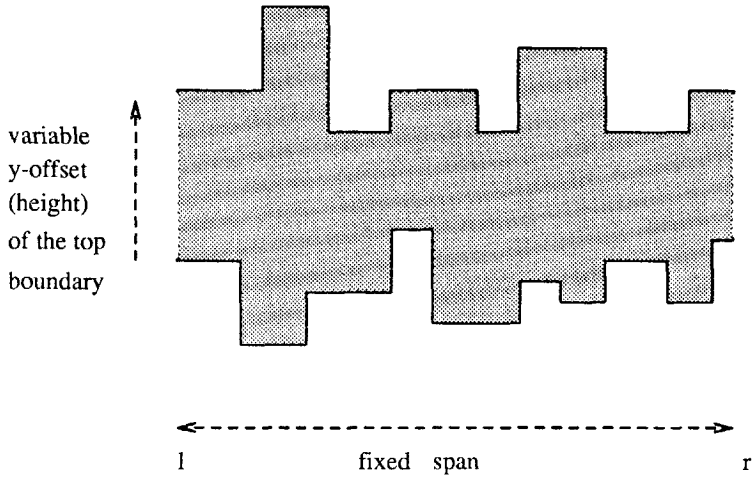

Figure 8: An example of a channel. The area is vertically convex, but not horizontally convex.

Due to its low resolution the grid framework can hide most of the process-specific design rules. Selecting the proper mesh width will ensure that the grid graph represents a wire configuration which can be translated into a wire pattern without design rule violations. For some sets of design rules, however, 'illegal' wire patterns can be represented in the grid graph. A notorious example of the latter are the stacked contacts if $\nu \geq 3$. These rules will have to be propagated to the wire generator. Obstacles can be modeled in the grid graph by removing specific edges. Notice that the mesh width is determined by the 'worst-case' spacing between two grid rows or columns. This causes some loss in area. In chapter 3 we will come back to this subject.

2.2 Formulation of the problem

2.2.1 A generalized problem description

For the remainder of this thesis we will discuss a horizontal channel, that is, a strip bounded by modules on the bottom and top sides. A horizontal channel is a vertically convex routing region. The horizontal *span* of the channel is given by the closed interval $[l, r]$. The vertically convex extremes of the channel are captured by two *boundary functions*. The bottom(top) boundary of the channel is given by the function $bot(x)$ ($top(x)$) which uniquely assigns one y-coordinate to each x-position in the

span of the channel ($l \leq x \leq r$). The offset of the top boundary is adjustable: $top(x) = top'(x) + h$, in which $top'(x)$ describes the actual boundary and h is a number called the *channel height.* Since the bottom and top modules may not overlap, the value of h is such that the upper boundary is always above the lower one: $\forall x, l \leq x \leq r : top(x) - bot(x) \geq 0$. Again, we assume that there are ν ($\nu \geq 2$) layers of interconnect available, numbered $1, 2, \ldots, \nu$.

Given is a set of terminals[2] $\{T_{\alpha,\beta}\}$, in which α specifies a net and β a terminal. A collection of nets $\eta = \{N_1, N_2, \ldots, N_n\}$ is given to specify the connectivity over the terminals: $\forall i\ N_i = \{T_{i,\beta}\},\ \beta = 1, 2, \ldots, \beta_i$. The 'smallest' net, that is, if $|N_i| = 2$, is called a *two-terminal net.* If $|N_i| \geq 3$ we call the net a *multi-terminal net.* In figures we will tag the terminals according to their net number. Each terminal is assigned to one of the 4 boundaries of the channel. We define the set BOTTOM to contain those terminals which are assigned to the bottom side of the channel. Similarly, the sets TOP, LEFT and RIGHT contain the terminals assigned to the top, left and right boundaries, respectively. Associated with each terminal $T_{\alpha,\beta}$ is a position $p_{\alpha,\beta} = (p^x_{\alpha,\beta}, p^y_{\alpha,\beta}, p^z_{\alpha,\beta})$, $l \leq p^x_{\alpha,\beta} \leq r,\ p^z_{\alpha,\beta} = 1, 2, \ldots, \nu$. It specifies the center coordinate and the layer of the terminal, which must be on the boundary of the channel. Therefore it is useful to distinguish between the terminals specified on each of the boundaries:

1. If $T_{\alpha,\beta} \in \text{BOTTOM} \bigcup \text{TOP}$, it is called a *fixed terminal.* The position $p_{\alpha,\beta}$ of a fixed terminal is uniquely determined by its x-coordinate and the corresponding boundary function: $p_{\alpha,\beta} = (p^x_{\alpha,\beta}, bot(p^x_{\alpha,\beta}), p^z_{\alpha,\beta})\ \vee\ p_{\alpha,\beta} = (p^x_{\alpha,\beta}, top(p^x_{\alpha,\beta}), p^z_{\alpha,\beta})$, $l < p^x_{\alpha,\beta} < r$.

2. If $T_{\alpha,\beta} \in \text{LEFT} \bigcup \text{RIGHT}$, it is called a *floating terminal.* It captures a net which is entering the channel on the left or right side. Only the side $p^x_{\alpha,\beta}$ ($p^x_{\alpha,\beta} = l\ \vee\ p^x_{\alpha,\beta} = r$) of a floating terminal is given. The y-position $p^y_{\alpha,\beta}$ ($bot(p^x_{\alpha,\beta}) > p^y_{\alpha,\beta} > bot(p^x_{\alpha,\beta})$) and layer $p^z_{\alpha,\beta}$ of a floating terminal will be determined by the channel router. We will require that $\forall i\ N_i : |N_i \cap \text{LEFT}| \leq 1\ \wedge\ |N_i \cap \text{RIGHT}| \leq 1$ (each net has at most 1 terminal on the left and right boundaries).

Furthermore, the pattern width of each terminal is given by $tw_{\alpha,\beta}$ ($tw_{\alpha,\beta} \geq \mathcal{W}_{p^z_{\alpha,\beta}}$). The pattern of a fixed terminal covers the interval $[p^x_{\alpha,\beta} - \frac{1}{2}tw_{\alpha,\beta}, p^x_{\alpha,\beta} + \frac{1}{2}tw_{\alpha,\beta}]$. Since the terminals on the fixed boundaries satisfy the design rules, the spacing between adjacent terminals in *layer* is at least $\mathcal{S}_{layer}$. Finally, the desired wire width of each net N_i in a layer is given by $nw_{i,layer}$ ($nw_{i,layer} \geq \mathcal{W}_{layer}$).

[2]Some authors refer to them as *pins* or *ports.*

The objective of channel routing is to connect the terminals according to the net list specification while satisfying the design rules and keeping the following parameters as small as possible:

- the channel height h,
- the number of vias and
- the wire length.

Minimizing the channel height is the main objective. We call an instance of the channel routing problem *solvable* if a design rule correct wire pattern exists which connects all terminals according to the net list specification.

2.2.2 The 'classical' channel routing problem

The definition of the problem in the previous subsection is very general. To allow the use of various algorithms, a more restricted definition of the channel routing problem is generally used. We will call this restricted version the *classical* channel routing problem. Most of the work published on channel routing is on the classical channel routing problem. It includes the following restrictions:

1. $\nu = 2$, that is, two interconnect layers are available for routing.
2. The *reserved layer*[3] model is used, which means that each layer contains wire segments in one direction only. In a 'horizontal' layer only horizontal wire segments are present. Similarly, a 'vertical' layer exclusively contains vertical wire segments. A scheme of alternating 'horizontal' and 'vertical' layers is defined.
3. The boundary functions $bot(x)$ and $top(x)$ specify the straight horizontal line $y = 0$.
4. The channel is modeled on a $m \times n \times \nu$ grid graph, consisting of m columns, n rows and $\nu = 2$ layers. Without loss of generality we can assume that the 'origin' of the channel is in the left bottom corner. The columns are numbered $c = 1, 2, \ldots, m$, the rows $r = 1, 2, \ldots, n$ and the layers $l = 1, 2$.
5. The fixed terminals are aligned with grid columns and are in the vertical layer. The terminal width $tw_{\alpha,\beta}$ and the net wire width $nw_{i,layer}$ are uniform for all nets.

[3]Sometimes called, quite inappropriately, the *Manhattan* model

In the Venn-diagram of figure 5 (page 10) the classical channel routing problem is indicated by the intersection of the sets marked 'grid-based' and 'reserved-layer'.

Due to the grid-graph framework we can express the channel height in the number of grid rows. These rows are generally called *tracks*. The number of tracks is the measure of height in the classical channel routing problem. The horizontal wire segments are called *trunks*. Vertical wire segments (e.g. for connecting a terminal to a trunk) are called *branches*. The *span* of a net N_i is the closed horizontal interval $[p^x_{i,l}, p^x_{i,r}]$, in which $T_{i,l}$ is the left-most terminal of N_i and $T_{i,r}$ its rightmost terminal (including the columns of the outside terminals).

The *density* d_c of column $x = c$ is the number of nets whose span includes that column. The density d of a given channel is defined by $\max_{1 \leq c \leq m} d_c$. Clearly, the density d sets the lower bound for the channel height in the classical model.

Definition 3 (Net characterization) *A net N_i is called* two-sided *if $N_i \cap$ BOTTOM $\neq \emptyset \wedge N_i \cap$ TOP $\neq \emptyset$. A net is* single-sided *if it is not two-sided.*
A pair of terminals $(T_{i,j}, T_{i,k})$ in N_i is called trivial *if one of the following conditions is true:*

1. $T_{i,j} \in$ BOTTOM *and* $T_{i,k} \in$ TOP *while* $p^x_{i,j} = p^x_{i,k}$ *and* $p^z_{i,j} = p^z_{i,k}$ *(the terminals share the same column).*

2. $T_{i,j} \in$ LEFT *and* $T_{i,k} \in$ RIGHT *while* $p^z_{i,j} = p^z_{i,k}$ *(their span covers the entire channel).*

The net is called trivial *if all its terminals are pairwise trivial.*

We need the characterization for trivial nets because those nets can be implemented by a straightforward wire pattern.

Clearly, the objective for the classical channel routing problem is to create the wire pattern which requires the smallest number of tracks. This *channel width minimization problem* has received much attention during the past two decades. Many approaches to tackle it have been published. Before outlining channel routing algorithms, it is useful to re-state the theoretical result which was derived by Szymanski in 1985:

Theorem 1 ([81]) *For the classical channel routing model it is an NP-complete problem to determine whether an arbitrary channel can be routed using a specified number of tracks.*

As a result, no channel width minimization algorithm that is efficient and exact at the same time is likely to exist for the classical channel routing problem. This can be seen as a justification for the development of heuristic algorithms.

2.3 Channel routing as a graph coloring problem

The oldest and most frequently applied approach to channel routing is the so-called *trunk-based* approach. The nets are viewed as a collection of horizontal trunks, to which all terminals are connected by branches. This condenses the channel width minimization problem to packing the trunks in the channel such that the smallest number of tracks is required. In its most simple form, each net is implemented by one single trunk. We will call this version of trunk-based approach the *single-trunk*[4] model.

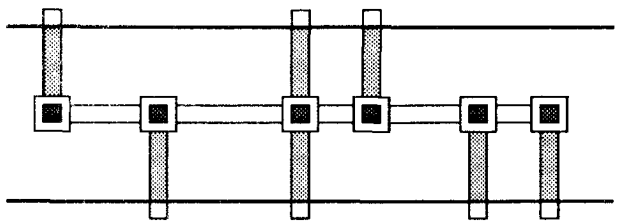

Figure 9: Routing in the single-trunk *model. One trunk spans the terminals of the net. All terminals are connected to this trunk by branches.*

For many years the trunk-based approach has been associated strongly with channel routing. To such extend even, that a channel router was implicitly an algorithm to order the trunks. In this section we will briefly discuss the characteristics of the trunk-based approach, as opposed to our 'territory approach' which will be present in the next section.

2.3.1 Trunk ordering graphs

Assume that the single-trunk model is applied. Let $tr(N_i)$ denote the track number to which the trunk of net N_i is assigned. Two important abstract properties can be derived to express the arrangement of the trunks:

Definition 4 (Horizontal constraint) *Trunks belonging to nets of which the span overlaps must be placed in different tracks. For the entire channel this is expressed by the relation* R_H*:*

$$R_H = \{(N_i, N_j) \mid [p^x_{i,l}, p^x_{i,r}] \cap [p^x_{j,l}, p^x_{j,r}] \neq \emptyset \wedge i \neq j\}_{n \times n}$$

[4]Sometimes called the *restrictive-* or the *jog-free* model.

With this relation a horizontal constraint *between two nets N_i and N_j is defined as:*

$$(N_i, N_j) \in R_H \;\Rightarrow\; tr(N_i) \neq tr(N_j) \tag{1}$$

The ensemble of horizontal constraints can be captured in a horizontal constraint graph $G_H = (\eta, R_H)$. *G_H is an undirected graph in which the vertices represent the set of nets η, and an edge $\{N_i, N_j\}$ exists if $(N_i, N_j) \in R_H$.*

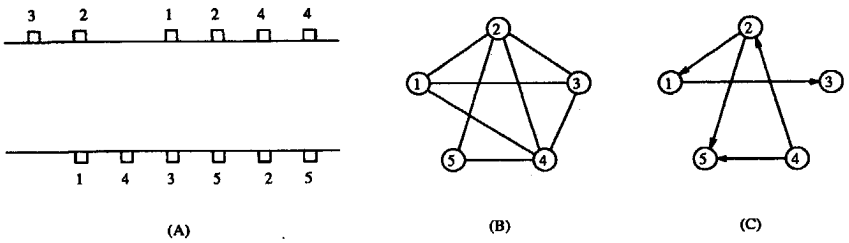

Figure 10: An example channel (a) with its associated horizontal- (b) and vertical (c) constraint graphs. The vertices denote trunks, which are equivalent to nets in the single-trunk model.

Definition 5 (Vertical constraint) *Consider two terminals, $T_{i,t} \in$* TOP *and $T_{j,b} \in$* BOTTOM *($i \neq j$), which share the same column. Then the trunk of net N_i must be placed above the trunk of net N_j since branches may not overlap within a column. For the entire channel this is expressed by the* vertical constraint *relation R_V:*

$$R_V = \{(N_i, N_j) \mid T_{i,t} \in \text{TOP} \wedge T_{j,b} \in \text{BOTTOM} \wedge p^x_{i,t} = p^x_{j,b} \wedge i \neq j\}_{\eta \times \eta}$$

A vertical constraint *from net N_i on net N_j can be defined as:*

$$(N_i, N_j) \in R_V \;\Rightarrow\; tr(N_i) > tr(N_j) \tag{2}$$

Clearly, the span of a pair of nets (N_i, N_j) must overlap in order to constitute a vertical constraint:

$$(N_i, N_j) \in R_V \;\Rightarrow\; (N_i, N_j) \in R_H$$

A directed vertical constraint graph ('VCG') *$G_V = (\eta, R_V)$ captures the ensemble of vertical constraints in the channel. In G_V the vertices represent the set of nets η and a directed edge (N_i, N_j) exists exactly if $(N_i, N_j) \in R_V$.*
The total constraint graph *$G_t = (\eta, R_V \cup R_H)$ is a partially directed graph which results from merging G_H and G_V. The graph is obtained by adding to G_V all undirected vertices in G_H which connect vertices that are not already adjacent in G_V.*

Figure 10 shows the constraint graphs G_H and G_V of a sample channel. A legal routing in the single-trunk model is an assignment of a track number $tr(N_i)$ to the trunk of each net N_i such that the horizontal constraints (inequality 1) and the vertical constraints (inequality 2) are satisfied. Any violation of these rules will cause a short-circuit. The problem is equivalent to finding a *rainbow coloring* of the partially directed graph G_t, in which the vertex color corresponds to the track number $tr(N_i)$. The term *rainbow* coloring was introduced by Lengauer [51, pages 544-545]. It denotes that the colors (track-numbers) are ordered according to their wavelength, due to the vertical constraints (inequality 2). In this way the channel width minimization problem in the single-trunk model is the problem of finding a rainbow coloring of G_t with the smallest number of colors. A solution for this problem can be found only if the VCG (or G_t) is acyclic. Without the presence of vertical constraints the channel width minimization problem is an ordinary graph coloring problem.

2.3.2 The left-edge algorithm

In 1971 Hashimoto and Stevens [38] proposed the - now classical - *left edge* algorithm to perform the graph coloring. The left edge algorithm processes the tracks one at a time, starting with the bottom track. As defined earlier, the span of net N_i is denoted by the interval $[p^x_{i,l}, p^x_{i,r}]$. It defines the horizontal interval which must be spanned by the trunk. The left edge algorithm fills each track with non-overlapping trunks in the following way. Starting from the left it selects the trunk $[p^x_{i,l}, p^x_{i,r}]$ which has the smallest left edge coordinate $p^x_{i,l}$ (which explains the name of the algorithm) and of which the net does not impose a vertical constraint on any still unplaced trunk. The trunk is assigned to the current track. Then it selects the next trunk which has the smallest $p^x_{j,l}$ such that $p^x_{j,l} > p^x_{i,r}$ and which also does not violate a vertical constraint. This process continues until no trunk can be selected under the specified conditions. In this way the tracks are filled in a left to right order, each time packing as many trunks as possible in the track. After completing the assignment of the trunks to the tracks, the branches to the terminals can be easily added. Obviously it is also possible to employ the same concept starting form the right edge and working left. This 'right edge' algorithm will achieve different results, of course. Several of these heuristic improvements on the left-edge algorithm have been published [23, 91, 90].

If there are no vertical constraints in the channel ($R_V = \emptyset$, such as in single-sided channels) the left-edge algorithm will find an optimal solution for which the channel height h equals the density d of the channel. Figure 15 (on page 26) shows the result of the left-edge algorithm on a small single-sided channel without vertical constraints.

With the presence of vertical constraints, however, the left-edge algorithm may use many more tracks (colors) than necessary. LaPaugh [48] proved that the channel

width minimization problem for the single-trunk model is NP-complete.

2.3.3 Completion guarantee with vertical constraints

A fundamental issue of the trunk-based approach are the vertical constraints. As soon as there is a cycle in the VCG (as in figure 12) there are no solutions to the graph coloring problem. The left-edge algorithm cannot select constraintless trunks. This jeopardizes the dream of the 100% completion guarantee.

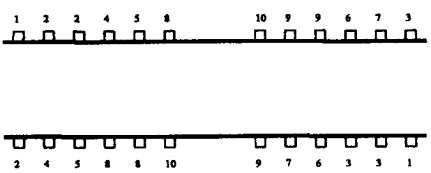

Figure 11: A sample channel (after [6]). It is a 'nightmare channel' for most trunk-based channel routers.

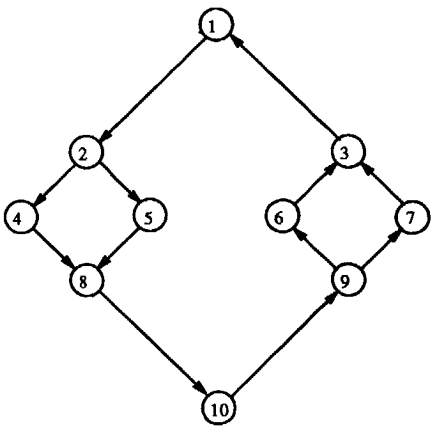

Figure 12: The corresponding vertical constraint graph of the channel is cyclic. Therefore this channel cannot be solved in the single-trunk model.

To solve the cyclic vertical constraint problem the cycles must be broken. This can be performed by splitting a net of the cycle into sub-nets, each with a separate trunk (see figure 13). The sub-nets are connected to one another by a vertical wire. In the VCG the vertex of the net is split, and with that the cycle should be broken. The resulting wire pattern in the net is called a *dogleg*, after the slight resemblance with the animal's anatomy. Two additional vias per dogleg are required for connecting the trunks by a branch. Apart from solving vertical constraints, doglegs may also reduce the channel height. The shorter trunks can be packed more efficiently in the channel. At the time (1976) this meant a breakthrough in channel routing.

In [23] it is suggested to break the net (and with that, to place the dogleg) in a column containing a terminal of the net (as in figure 13). This reduces the size

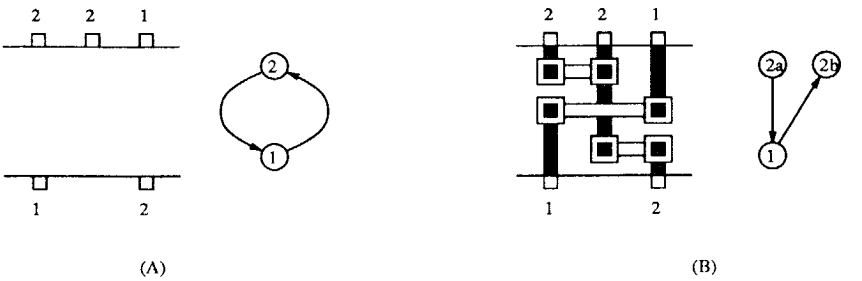

Figure 13: An elementary cyclic vertical constraint (a). It is broken by splitting one of the net into sub nets (b).

of the configuration space and it requires at most one additional via per dogleg. Unfortunately this approach will not solve all cyclic vertical constraints. Burstein's example channel (figure 11), for instance, can only be solved by placing the dogleg in another column. These *non-terminal doglegs* will break any vertical constraint, provided there are sufficient columns in the channel. Several heuristic approaches to place the non-terminal doglegs have been published [71, 66].

Also, some 'dirty tricks' have been applied to solve vertical constraint problems. YACR2 [71], for instance, allows vertical constraint violations in its implementation of the left-edge algorithm. It tries to solve them locally by maze routing, thereby leaving the reserved-layer model. For this purpose it is equipped with a number of dedicated maze routers which route a small area around the position of the constraint. Only if the maze routing fails it will add an additional track.

2.3.4 Hierarchical channel routing

An elegant hierarchical approach to channel routing was proposed by Burstein and Pelavin [6]. Rather than ploddering with constraint graphs they recursively divide the channel into smaller areas, each time global routing on a $2 \times n$ graph. The advantage of this method is that doglegs are within its search path for wire realization. Therefore no specific constraint breaking step is required. In fact, Burstein's algorithm generates many doglegs (see figure 14 and table 1). They were the first to solve Deutsch's difficult example [23], the benchmark for channel routers channel, in density without violating the reserved-layer dogma.

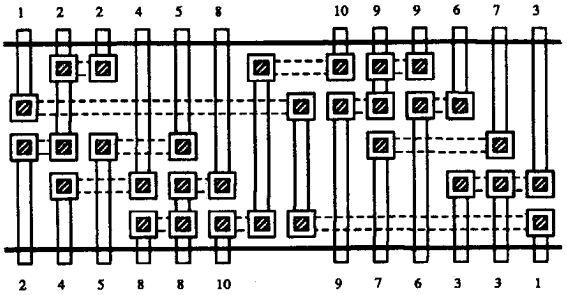

Figure 14: The result of [6] for the channel in figure 11. It contains doglegs *to break the cyclic vertical constraint and to reduce the channel height.*

2.4 A constructive method for channel routing with complete routability

The most important aspect of a channel router is its 100% completion guarantee. In this section we will provide a theoretical background for the routability of a channel in the classical model. A precise condition under which the channel routing problem is solvable will be defined. To prove this condition we define a framework for constructive routing (called the *territory framework*) which is the leitmotiv of the practical channel routers presented in chapters 3 and 4. We can prove that our method will always generate a solution if one exists in the classical model. This strong routability guarantee cannot be provided by the trunk-based channel routers described in the previous section. The trunk-based model they use covers a too restricted portion of the solution space of the classical channel routing problem.

2.4.1 Constructive routing

In the previous section we have described the trunk-based approach, in which the channel routing problem is defined as a trunk ordering problem. We have also described a hierarchical approach which attacks the channel by stepwise refinement. In this section we will define a third methodology, which we will call *constructive* channel routing. The basic property of constructive channel routing is the presence of a *partial routing* during the channel routing process. The partial routing implements a portion of the channel in great detail, which, in principle, will not be altered later on. The generation of the wires is performed in a number of steps. At each step, a set of wire segments is added to the partial routing. The process continues until all

nets have been fully implemented.

In this thesis, we will perform constructive routing on a *per net* basis. At each step of the sequential routing, the entire wire pattern of a single net $N_i \in \eta$ is added to the partial routing. This is a process which takes $|\eta|$ steps, that is, the partial routing passes through $|\eta| + 1$ states. Let $P_\alpha(I)$ denote the partial routing of an instance I of the classical channel routing problem. The integer α, $0 \leq \alpha \leq |\eta|$ marks the number of nets the partial routing $P_\alpha(I)$ contains. The initial state is $\alpha = 0$ (no net has been routed) and if $\alpha = |\eta|$ the partial routing $P_{|\eta|}(I)$ contains the final wire pattern implementing all nets. With each state α, a partial net list η_α ($\eta_\alpha \subseteq \eta$, $|\eta_\alpha| = |\eta| - \alpha$) is associated. It contains the nets which are still unrouted in state α. Obviously, $\eta_0 = \eta$.

2.4.2 Single-sided channels

Channels in which all fixed terminals are located on one side of the channel, are called *single-sided.* They form an easily solvable subset of the general channel routing problem. Due to the absence of vertical constraints, optimal solutions can be constructed using the trunk-based left edge algorithm (see figure 15). In that case, the wire of each net consists of one trunk only. In our constructive model, however, we allow nets to be implemented by an arbitrary wire pattern in the reserved layer model. The wire pattern of a net may contain any number of trunks and branches, as long as it connects the terminals. It is evident that other (still unrouted) nets could become unroutable without any restriction on the implementation of the wire. In the single-trunk model this unroutability is implicitly prevented by the restrictions of the model because a column only contains one branch by construction. To ensure the 'routability' of each (still unrouted) net $N_i \in \eta_\alpha$, we allocate a protected area around each terminal:

Definition 6 (Territory) *The* territory $t_{i,\beta}$ *of fixed terminal* $T_{i,\beta}$, $N_i \in \eta_\alpha$ *is the column* $x = p^x_{i,\beta}, z = p^z_{i,\beta}$. *As long as* $N_i \in \eta_\alpha$, *no branches of other nets are allowed in the territory.*

The territory $t_{i,\beta}$ must be seen as a protected column which reserves room for the branch connecting $T_{i,\beta}$ to a horizontal trunk. The height of the territory is 'infinite'. Immediately before a terminal is routed, its territory will be removed. The predominant rule in our framework specifies that 'foreign' territories may not be violated.

Definition 7 (Routable nets) *At a state* α *during the sequential routing of the nets, a net* $N_i \in \eta_\alpha$ *is called* routable *if a legal path can be found in* $P_\alpha(I)$, *connecting all its terminals within the rules of the territory framework.*

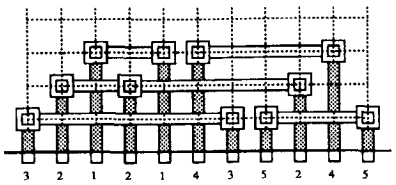

Figure 15: Routing using the single-trunk *model, in which all terminals are connected by branches to a single horizontal trunk.*

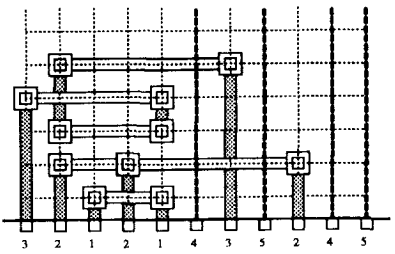

Figure 16: A partial routing $P_3(I)$ in the territory model, *after $\alpha = 3$ nets have been routed. Nets 4 and 5 are still unrouted. The territories of their terminals are indicated by the dashed lines.*

A net $N_i \in \eta_\alpha$ is called continuously routable *if it remains routable in any consecutive partial routing $P_\beta(I)$, $(\beta > \alpha)$, regardless of the order (and the way) in which the other nets are routed.*

Fact 1 (Routability of single-sided nets) *A single sided net is* continuously routable *in the territory framework.*

As a consequence, any single sided channel is solvable in the territory framework. It is easy to see why we can state this fact. Suppose that t is the track number of the highest trunk in a partial routing $P_\alpha(I)$ (see also figure 16). To route an arbitrary (single sided) net $N_i \in \eta_\alpha$ we can always place a trunk in track $t + 1$. All terminals of N_i can be connected to this trunk because the height of their territories is infinite. The amount of tracks required to complete the routing depends primarily on the algorithm which generates the wire of a single net.

2.4.3 Two-sided channels

Let us construct a two-sided channel by placing two single-sided channels opposite each other (see figure 17 on page 28). The channel consists of a bottom sub-channel and a top sub-channel with an interface in between. The common boundary of the sub-channels consists of m *crossings*. Any two-sided net must place a branch in at least one of these crossings. Obviously, a column can now be claimed by two territories. Therefore we define that the claims made by the territories end exactly at

the crossing. Let the set $C_\alpha(c), 1 \leq c \leq m$ contain those terminals whose territories occupy column $x = c$ in a partial routing $P_\alpha(I)$. Notice that $|C_\alpha(c)| \leq 2$ because each column contains at most one terminal on the bottom side and one on the top side. Focusing on an arbitrary column $x = c$ at the intersection between the two sub-channels, we can distinguish the following types of crossings:

- If $C_\alpha(c) = \{T_{i,s}, T_{j,t}\}$, $i \neq j$ we call the crossing *conflicting*. This case constitutes a 'clash' of two territories, similar to the familiar vertical constraint. Using the crossing for a branch would violate one of the territorial claims. Therefore we disallow a branch in a conflicting crossing.
- If $C_\alpha(c) = \{T_{i,s}, T_{i,t}\}$ we call the crossing *trivial*. The use of this crossing, capturing two territories of the the same net, is reserved for a branch of net N_i only.
- If $C_\alpha(c) = \{T_{i,s}\}$ we call the crossing *reserved*. A reserved crossing contains a territory on one side only. The use of this crossing is reserved for a branch of net N_i. All other nets may not place a branch in this crossing until N_i has been routed.
- If $C_\alpha(c) = \emptyset$ we call the crossing *free*. A free crossing may be used by branches of any net, since no territories will be violated.
- If a branch was assigned to crossing $x = c$, we call it *occupied*.

In a partial routing $P_\alpha(I)$ each crossing is tagged by one of these types. Figure 17 (on page 28) illustrates the different types of crossings $C_0(c)$ in an empty two-sided channel. Routing a net changes the type of some of these crossings because territories are removed and crossings could become occupied by branches. Figure 18 (on page 30), for instance, shows the partial routing of a two-sided channel in state $\alpha = 1$. Before any net was routed (that is, $P_0(I)$), the crossings in columns 1 to 10 were conflicting and in column 11 the crossing was free. In state $\alpha = 1$ net 1 has been routed. In $P_1(I)$ the type of crossings in columns 1 and 10 changed into reserved, while the crossing in column 11 became occupied.

The following can be stated about the routability of a net in our territory model:

Fact 2 (Routability of nets) *Given is a partial routing $P_\alpha(I)$ of an instance I of the classical channel routing problem. A net $N_i \in \eta_\alpha$ is* continuously routable *in our territory framework if at least one of the following conditions is true:*

1. $N_i \cap \text{BOTTOM} = \emptyset \ \vee \ N_i \cap \text{TOP} = \emptyset$
   (the net is single-sided)

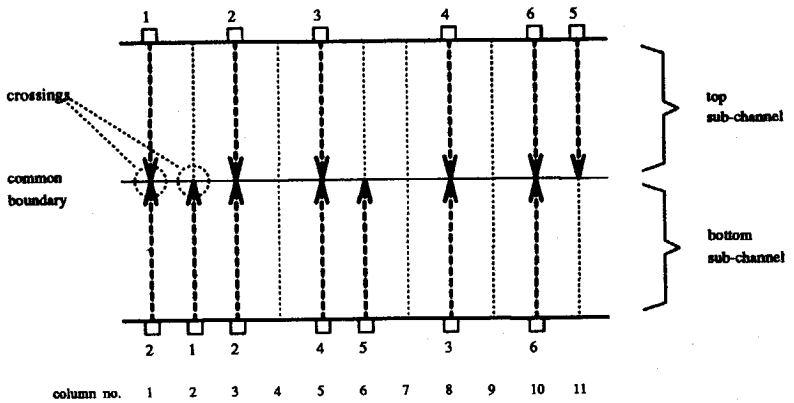

Figure 17: A two-sided channel, consisting of two half-channels. The dashed arrows indicate the territories of the terminals, which end at the common boundary. The terminals on the common boundary are called crossings. *The crossings in columns 1, 5 and 8 are* conflicting, *in columns 2, 6 and 11 the crossings are* reserved *and in columns 3 and 6 they are* trivial. Free *crossings can be found in columns 4, 7 and 9.*

2. $\exists c, 0 \leq c \leq m : C_\alpha(c) \subseteq N_i$
   (A terminal of N_i takes part in a reserved crossing or N_i contains a trivial pair of terminals)

In $P_\alpha(I)$ any net $N_i \in \eta_\alpha$ is routable *(but not necessarily continuously routable) if:*

1. *a free crossing exists in state α.*

For instance, in the empty channel of figure 17 the nets 3 and 4 are ***routable*** in state $\alpha = 0$ due to the presence of free crossings. All other nets in that channel are already ***continuously routable*** in $P_0(I)$. It is again easy to see why we can state this fact. Since the channel consist of two single-sided sub-channels, the routability of single-sided nets was already stated in fact 1. A two-sided net can be routed by splitting it into two single-sided nets, one in each sub-channel. As soon as there is a crossing available for a branch which connects these two subnets, the net is routable. The presence of trivial crossing or a reserved crossing for one of its terminals ensures that such a branch can always be placed, making the net continuously routable.

To make a further statement about the routability of a channel we need the following definition:

Definition 8 (Routable crossings) *Given is a partial routing $P_\alpha(I)$ of an instance I of the classical channel routing problem. The set of* routable crossings *$R(P_\alpha(I))$ is*

the union of the reserved and the free crossings in $P_\alpha(I)$.

For example, consider again figure 17. In the empty channel ($\alpha = 0$) the set $R(P_0(I))$ contains the crossings in columns 2, 4, 6, 7, 9 and 11. We are now ready to make a strong statement about the routability of a channel.

Lemma 1 (Snowballing routability) *An instance I of the classical channel routing problem is solvable in the territory framework if $R(P_0(I)) \neq \emptyset$.*

In other words: the presence of a single free or reserved crossing in the empty channel is sufficient to guarantee the completion of the problem.

Proof:

The general structure of the proof is that routing a net $N_i \in \eta_\alpha$ will never result in an empty set of routable columns $R(P_{\alpha+1}(I))$ at the next state. In this way at least one other net $N_j \in \eta_{\alpha+1}$ will be routable when the routing of N_i is completed. Routing N_j again results in a nonempty set of routable crossings $R(P_{\alpha+2}(I))$. This 'snowballing' process can be continued until all nets have been routed ($\alpha = |\eta|$).

Let us investigate the change in the number of routable columns due to the routing of an arbitrary routable net $N_i \in \eta_\alpha$. For convenience, we will define the change due to the routing of N_i as $\Delta R = |R(P_{\alpha+1}(I))| - |R(P_\alpha(I))|$.

The territories of the fixed terminals of N_i can be found in n_{conf} conflicting crossings, n_{res} reserved crossings and n_{triv} trivial crossings. Routing N_i will remove the territories. To investigate the effect on ΔR we can distinguish two cases:

1. The net is single-sided. In that case we can require that the net is implemented without using a crossing. Therefore $\Delta R = n_{conf}$ and, with that $\Delta R \geq 0$. Notice that, implicitly, $n_{triv} = 0$ for single sided nets.

2. The net is two-sided. In that case we can require that it is implemented using one crossing only. Therefore, $\Delta R = n_{conf} + n_{triv} - 1$ and with that $\Delta R \geq -1$. The wire pattern can, for instance, consists of two trunks, one in the bottom sub-channel and one in the top sub-channel. All terminals on the respective sides are connected to those trunks. Finally, a crossing is selected for a branch connecting the two trunks.

Summarizing, the only nets of which the routing results in a negative value for ΔR, are two-sided nets without conflicting or trivial crossings. Suppose that $|R(P_\alpha(I))| = 1$ at some state during the sequential routing of a net. To obtain $|R(P_{\alpha+1}(I))| = 0$ in the next state, a two-sided net $N_i \in \eta_\alpha$ must exist for which $n_{conf} + n_{triv} = 0$. Suppose that such a net exists. Since it is a two-sided net, the number of terminals with reserved crossings n_{res} is at least two. If $n_{res} \geq 2$, however, $|R(P_\alpha(I))| \geq 2$

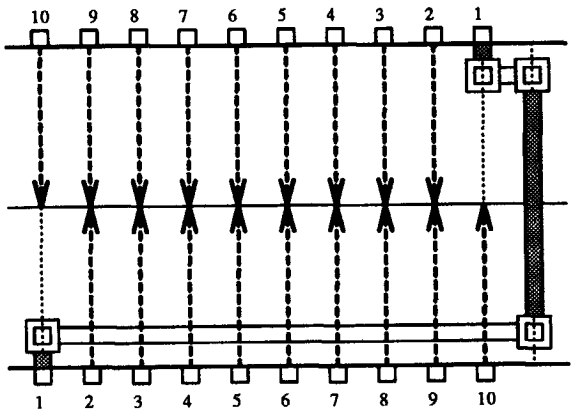

Figure 18: Illustration of corollary 1. This example is routable due to the free column. The picture shows the partial routing $P_1(I)$. The nets will be routed in the order 1, 10, 2, 9, 3, 8, 4, 7, 5, 6.

since reserved crossings belong to $R(P_\alpha(I))$. Therefore such a net does not exists if $|R(P_\alpha(I))| = 1$. As a result, if $|R(P_\alpha(I))| \geq 1$ then $|R(P_\beta(I))| \geq 1$ in any consecutive state $\beta > \alpha$ in the constructive routing of the nets in the territory framework. □

Since the territory framework is defined for the classical channel routing problem, we can state the following.

Corollary 1 *An instance I of the classical channel routing problem is solvable with at most one additional column.*

Until now we have identified cases and conditions under which the channel routing problem is solvable in the classical model. It is also interesting to be able to predict which problems are not solvable.

Theorem 2 *An instance I of the classical channel routing problem with m columns is* not solvable *if, and only if, it contains exactly m two-sided nets among which nontrivial nets exist.*

Proof:
We will first identify the cases which are, or can be reduced to, a channel routing problem in which $|R(P_0(I))| \geq 1$. In any of the following cases an instance I of the classical channel routing problem is solvable:

1. $|R(P_0(I))| \geq 1$
   As stated in lemma 1 the channel routing problem is solvable, at least in the territory framework. Notice that if $|\text{BOTTOM}| < m \ \vee \ |\text{TOP}| < m$ then $|R(P_0(I))| \geq 1$. Therefore all terminal positions on the bottom and top sides must be occupied should the problem be non-solvable.

2. A single-sided net N_i exists with $|N_i \cap \{\text{BOTTOM} \cup \text{TOP}\}| > 0$.
   As stated by fact 2, this net is always routable. Routing the single-sided net first results in at least one routable crossing, which again makes the entire channel routable.

3. $\exists N_i \in \eta : |N_i \cap \text{BOTTOM}| > 1 \vee |N_i \cap \text{TOP}| > 1$ (a multi-terminal two-sided net with at least 3 fixed terminals exists).
   Suppose that a net N_i has v terminals in the bottom sub-channel: $|N_i \cap \text{BOTTOM}| = v, v \geq 2$. In this case, routable crossings can be created by pre-routing a part of N_i. Routing the bottom subnet results in v routable columns. Only one terminal (with territory) remains as part of the original net. The net result is $v - 1$ routable crossings. Therefore $|R(P_0(I))| \geq 1$ after the partial net has been routed, which makes the entire channel routable.

4. All nets are trivial.
   This makes the solution trivial, hence it can always be solved.

In the only remaining case, the channel contains exactly m two-sided nets among which nontrivial nets exist. An example is this case is depiced in figure 19. We have to prove that this case is not solvable.

Suppose a routing for this case exists. The presence of a non-trivial net indicates that a set of trunks must exist which cover at least the span of a net. In the reserved layer model, each track in the channel contains at least one trunk. The track can be removed if this is not the case. Therefore the first track in the channel must contain at least one trunk. Consider an arbitrary trunk on the first track. It connects two branches, making these branches electrically equivalent. Each of the terminals on the bottom side is connected by a branch which spans from the terminal to at least the first row (see figure 19). Therefore any trunk on the first track will always connect two terminals on the bottom side. Since any pair of two terminals on the bottom side belongs to different nets, the trunk will always implement a short-circuit. Due to this contradiction such a legal routing cannot exist. □

This enables us to state the following about the sequential routing process in our territory framework:

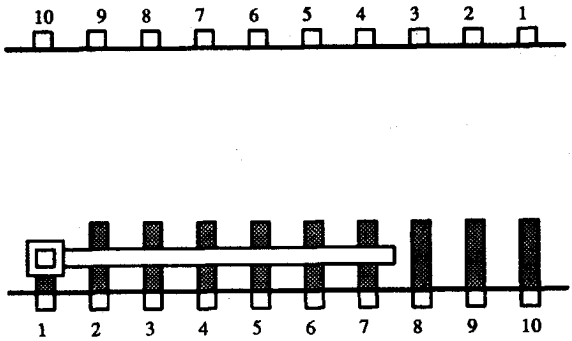

Figure 19: An illustration of an instance of the classical channel routing problem which is unsolvable. It is not possible to place a trunk in the first track without causing a short circuit.

Corollary 2 *For any solvable instance of the classical channel routing problem a solution can be found by sequential routing in the territory framework.*

None of the trunk-based channel routers can provide a similar guarantee. Notice that the non-solvable instances of the classical channel routing problem are quite rare in practice. Of all channels with a cyclic vertical constraint they form a small subset.

Until now, we have not made any statement about the wire which connects the terminals of a net. The only (quite mild) requirement is that the wire of a net must result in $R(P_{\alpha+1}(I)) \geq 1$. Apart from that, the wire may contain any number of branches and trunks, as long as no territories are violated. In this way it is not possible to make strong statements about the channel height h, which is dependent on the algorithm which adds the wire to the partial routing. The channel height is the sum of the heights of the two sub-channels. We can require,for instance, that the wires in each of the two sub-channels are routed in the single-trunk model. In that case each net will consume at most 1 track per sub-channel. To get the snowballing process started, it may be required to split one net, requiring one additional track. Therefore the worst-case channel height is $2|\eta| + 1$ tracks, under this assumption.

2.4.4 Per column constructive routing

Another constructive approach to channel routing was proposed by Rivest et. al. [73]. Instead of trying to fill the channel on a per-net or a per-track basis, they propose a heuristic approach to route the channel column by column. At each column there is only a limited number of situations, which can easily be dealt with. Usually the

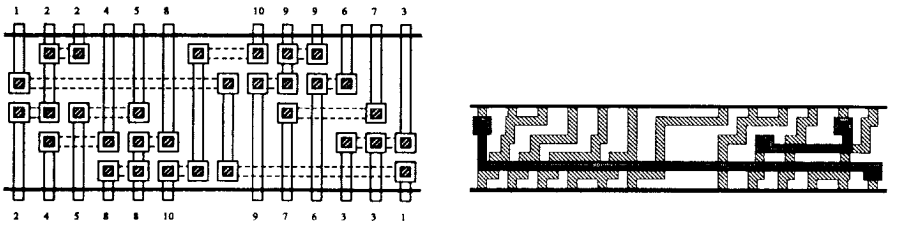

Figure 20: Two quite different results on the channel in figure 11 on page 22. The left figure shows Burstein's solution for the classical model, the right figure shows the result of the gridless router presented in chapter 3. In the latter the wires contain jogs *which reduce both the channel height and the number of vias significantly.*

decisions at a column are based on a number of common sense rules. Due to its constructive approach this algorithm was called 'greedy'. The simple and fast algorithm performs reasonably well on many channels. In some cases, such as the channel in figure 11, additional columns are required in the channel for completion. Therefore it cannot guarantee completion.

2.5 Breaking away from the classical problem

For the classical channel routing problem, many algorithms produce (close to) optimal results. We must bear in mind, however, that the classical problem is a very restricted by the framework. Much more could be gained by selecting other frameworks or removing restrictions to improve the layout and the interface with the surrounding context. In this section we will deal with a few of them, as a introduction to the channel routers presented in the next chapters.

2.5.1 Jogs

In the reserved-layer model of the classical channel routing problem all trunks are in one layer, while the branches are the other. In general, this limitation leads to sub-optimal layouts, containing considerably more vias than strictly necessary (see figure 20). Constructive routing methodologies can apply a less rigid framework than the trunk-based or hierarchical methods. Especially our territory framework can be easily adapted for wires with jogs. This can be done without affecting the completion guarantee.

2.5.2 Multiple-layer routing

With the advances in VLSI technology, the number of routing layers has increased. Based on trunk-based methods several modifications to route three or more layers have been made [19, 2, 3, 27]. As soon as there are two vertical layers the branches of the terminals may cross each other. With that, the vertical constraints are removed, or can be avoided easily. Therefore all efforts can be spent on packing the trunks in the horizontal layers. In [5, 47] the 'greedy' column-based heuristic was modified to handle multiple layers.

A strange phenomenon can be observed with many of the trunk-based attempts on multiple-layer channel routing. Most authors have focussed on the theoretical problem, thinking in terms of constraint graphs and tracks. In this way they lost track of the real practical problem which should be solved. This 'alienation' can be derived from a number of peculiar assumptions in the model which do not match with the physical properties of the problem. One of the most common assumptions is that each fixed terminal is present in all layers. In practical cases this is hardly ever the case. To condition the channel for these algorithms, each terminal must be accompanied by a pile of contacts (are in figure 6). Due to the contact spacing rules and the absence of stacked vias in many fabrication technologies this is not a practical solution. In this way the channel does not fit organically in between the surrounding modules. There is a considerable mismatch between the one-layer interface with the channel's context and the multiple-layer routing inside the channel. In both channel routers we present in this thesis a more realistic model is used.

2.5.3 Gridless Routing

The term gridless implies that we are leaving the grid graph as the framework which captures the wires. This should enable us to get rid of the most awkward constraints related to the grid graph as model of the routing area. In chapter 3 we will further discuss the specific grid-related constraints.

Chen [16, 17, 15] proposed a gridless variable wire width channel router which is essentially a generalization of a trunk-based router. He orders the variable width trunks such that minimum channel width is obtained. This is done by assigning directions to the undirected horizontal constraint edges in G_T such that the length of the longest (directed) path is minimized. He proposes a heuristic algorithm for this NP-complete problem. In similar approaches the trunks are compacted vertically (e.g. [86, 74]).

2.5.4 Maze and switch-box routing

A channel which has fixed terminals on 4 sides and of which the dimensions are fixed is called a *switch-box*. In an even more general routing problem is called *maze routing*. In that case the terminals can be positioned anywhere in the routing region. Both problems are older and more general that the channel routing problem, but they are also harder to solve.

Several techniques have been published to solve the problems. The majority of them are constructive. The most popular method was published in the early 1960tics by Lee [50]. He proposed a sequential method for area routing on a grid graph, with the goal of routing a printed circuit board. The nets are routed sequentially, each time generating the shortest path for a net using an adaptation of Dijkstra's vertex expansion algorithm [26]. Many variations and improvements on this theme, mainly suggesting ways to improve its speed by reducing the number of visited vertices in the grid graph (e.g. [59]). The large grid graph as framework is indeed a problem for the run time as well as the memory requirements. Different sequential approaches for general area routing were proposed by Hightower [40] and by Mikami and Tabuchi [55]. Their algorithms find paths by expanding lines around the terminals and determining when the lines hit objects or the destination. In this case the run time depends on the number of obstacles in the channel, not on the size of the grid. In this way the line search techniques have characteristics complementary to the graph-based Lee algorithms.

The nets are routed one after the other, each time 'greedy' selecting the shortest path. This shortest path may obstruct the path of other, still unrouted nets. Unlike our territory approach, there are no ways to guarantee the routability of the nets. So-called 'rip-up and reroute' techniques have been published to get around some of the problems (e.g. [77]). There is, however, no proven good scheme to select which net should be ripped up, nor a strategy to prevent this from happening.

None of the general area routing techniques, however, is able to guarantee 100% completion. A subdivision into routing areas which are switch-boxes will therefore never guarantee that all nets of the circuit can be implemented. Solving it generally requires manual intervention, sometimes to finish only the last few percent of the wires. If a completion problem is detected, it must be iterated back to the placement or global routing stage. This is not only time-consuming, it is also a hard to predict how much additional space is required. Besides that, routing areas with too few wires (below their capacity) are not detected, which is a waste of area.

Chapter 3

Contour-based gridless routing

3.1 Introduction

In this chapter we will present a channel router which is based on an adaptation of the territory framework. Its main characteristic is that it does not use a grid graph as underlying framework for the wires. Using a grid graph as the representation of the routing area involves a number of restrictions. The most prominent ones include:

- The pitch of the grid is determined by the 'worst case' spacing requirement over all layers. As discussed in 2.1.1, the size of a via patterns is usually wider than the wires ($\mathcal{V}_{l_i,l_{i+1}} \geq \mathcal{W}_{l_i}$). Since the grid model allows unrelated vias in adjacent tracks or columns the grid spacing s must be set such that:

$$\forall\, l, 1 \leq l \leq \nu : \begin{cases} s \geq \mathcal{V}_{l,l+1} + \mathcal{W}_l \\ s \geq \mathcal{V}_{l,l-1} + \mathcal{W}_l \end{cases}$$

  For some fabrication technologies this may lead to as much as 20% waste of space.

- A uniform wire width $nw_{\alpha,\beta}$ is require for all nets. Critical nets (e.g. power and clock) have to be treated separately.

- The terminals are required to align on grid lines.

Several approaches have been published to get around these grid-related constraints. The pitch problem, for instance, can be reduced by clever post-processing: variable track spacing [82], or compaction of the channel [22, 65, 13]. In [12] an algorithm is presented to condition a grid-based channel such that it is better 'compactable' by shifting the vias. To evade the terminal alignment constraint, a virtual

grid [71, 56] and 'gridding' the terminals by river routing [11, 65] have been proposed. Some of these 'hidden-grid' routers even obtained the unjustified appelation *gridless*. Only by abandoning the grid framework completely the drawbacks can be avoided. In the approach presented in this chapter we replace the grid framework by a set of contours. The contours form a very accurate model of the underlying layout pattern. In this way the algorithm is 'closer to the silicon' than algorithms which operate on a grid graph. More physical details of the channel and its context can be taken into account in this way. The contour framework is also flexible enough to allow jogs in any layer, which reduces the number of vias considerably.

The contour-based gridless router is not restricted to two layer routing. Instead, it can handle any number of layers. Although grid-free approaches to two- and three-layer channel routing have been published (e.g. [15], which is trunk-based), multiple-layer channel routing has remained the exclusive domain of the grid framework until now.

Our basic approach is to stack wires on top of each other. The nets are routed one at a time by a procedure which we called *net router*. It wraps the wires tightly around the contours to minimize the channel height. In this way the partial routing will contain a dense wire pattern. The framework and the net router will be discussed in sections 3.3 and 3.4, respectively. In section 3.6 we will describe the *scheduler* which determines the order in which the nets are routed. When the routing of all nets has been completed, the top side of the channel (including its pile of wires) is shifted as close as possible towards the bottom side. Finally the redundant wire jogs are removed by a wire straightening post processor. This algorithm and other additions will be discussed in section 3.7.

3.2 Preliminaries

As in the previous chapter we assume that the routing area is a horizontal channel. We use the general problem definition as described in subsection 2.2.1. This includes that the bottom and top boundaries of the channel can be irregularly shaped and that terminals can be specified at arbitrary positions on the horizontal edges of the bottom and top channel boundaries and at 'floating' positions on the left and right boundaries. A fixed terminal can have any width and can be in any layer. A set of nets η specifies the connectivity over the terminals. There are ν ($\nu \geq 2$) layers available for wiring. A scheme of alternating horizontal and vertical layers is presumed (e.g. for three-layer routing either the HVH- or the VHV-type can be chosen). Within certain limits wires in the vertical direction are allowed in a horizontal layer and vice versa.

In a grid-based channel router the grid incorporates all process dependent parameters. For the down-to-earth gridless router, however, the process design rules play a more intricate role in the routing process. For the contour-based router the following relevant design rule parameters are given:

$\mathcal{S}_{layer}$ The minimum spacing in *layer*.

$nw_{i,layer}$ The (user-specified) wire width of net N_i in *layer* ($nw_{i,layer} \geq \mathcal{W}_{layer}$).

$\mathcal{V}_{l_1,l_2}$ The total width of the contact hole pattern in layer l_1, given that it connects the layers l_1 and l_2. Vias exist only between adjacent layers.

It is assumed that no additional design rule constraints exist between layers. In this description we also assume that the orientation of all wire segments is rectilinear (Manhattan-style layout).

3.3 Routing Framework

The routing framework must provide the limits within which a new wire can be added to the partial routing. It should prevent short circuits and design rule violations. It must also ensure that routing a net does not make other nets 'unroutable'. During the routing process the channel will be split into a top part and a bottom part, each of which is a more or less independent sub-channel. Routing can take place on both sides of the channel independently since the height of the channel is adjustable.

3.3.1 Contours

We define a *contour* as a set of piece-wise constant line segments shielding the upper limit of previously laid wires in a layer. It can be seen as a 'protective cover' for existing wires of the partial routing. Future wires will have to be laid at minimum spacing $\mathcal{S}_{layer}$ from the contour. We make a fundamental distinction between horizontal and vertical layers. In a vertical layer two contours are used on each sub-channels (see figure 21):

- The *bottom contour* covers the wires touching the boundary on this side of the channel. If a wire crosses to the opposite side of the channel, the contour around it will bend up to 'infinity'. Future wires in the corresponding layer are not allowed to pass this contour obstacle. Initially the bottom contour is wrapped around the channel boundary.

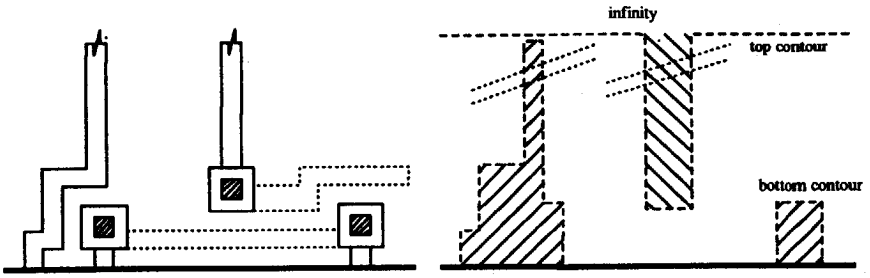

Figure 21: The bottom sub-channel in a 2-layer example channel (left). The shaded areas (right) are protected by the contours of the vertical layer.

- The *top contour* is wrapped around wires coming from the opposite side of the channel and ending above the bottom contour. Each top contour is initialized as a single horizontal line spanning the entire length of the channel and of which the height is 'infinite'.

A new wire in a vertical layer must be placed between the bottom and top contours. It will be blocked if the spacing between the contours is too small. In this way 'forbidden' intervals can be introduced.

In horizontal layers only one contour is used in each sub-channel: the bottom contour. Part of our strategy to ensure 100% completion is to require that the contour in a horizontal layer does not contain any unpassable obstacles (that is, forbidden intervals). In this way any pair of two horizontal positions in the channel can always be reached by a single continuous wire in a horizontal layer. A wire in a horizontal layer may not cross to the opposite side of the channel since this would introduce a forbidden interval.

3.3.2 Territories

Short-circuits with previously routed nets are prevented by the contour framework. To ensure the routability of the still unrouted nets, a *territory* is allocated around each terminal. In definition 6 the territory was defined as a column in the grid graph. In the contour framework a territory is a column-shaped area which is just wide enough for a via and which has infinite height. It is only valid in a vertical layer on the same sub-channel as the terminal itself. Wires of other nets in this layer are not allowed to enter the territory until the terminal has been routed. The dashed lines in the example channel in figure 22 (page 41) indicate the territory boundaries of the terminals of net 4. Since we do not allow a horizontal layer to contain forbidden

intervals, the terminals in horizontal layers obtain a territory in one of the adjacent vertical layers. Thus space is reserved for a via connecting the terminal to a vertical layer.

If the spacing between each pair of adjacent territories is at least $\mathcal{S}_{layer}$, all terminals can be connected to a wire in a horizontal layer, which guarantees routability. In section 3.7.1 we show how deal with the terminal spacing constraint emerging from territory overlap.

3.4 Net Router

The nucleus of this sequential contour router is the algorithm that connects the terminals of a single net to one another. It consists of three parts. First a *wire interval graph* is constructed. From the contours and the territories, a set of horizontal intervals can be derived in which a wire of the net could be laid without causing design rule violations or unroutability of nets later on. The idea is to capture the adjacencies of these intervals in a graph containing topological paths of the wire. Using the wire interval graph as framework a *global router* determines roughly the path of the wire. The weights of the edges in the graph enable a trade-off between the number of vias, the channel height and the wire length in each of the layers. Finally, the *detailed router* generates the actual wires based on the path in the wire interval graph. Notice that in each of the sub-channels we will route the nets in the single-trunk model.

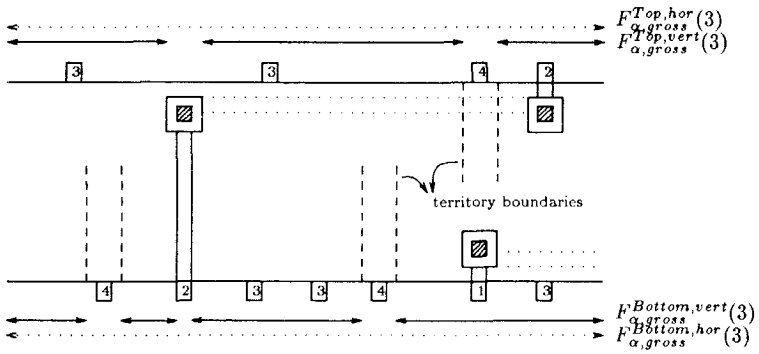

Figure 22: A partially routed channel in VH*-style. The intervals $F_{\alpha,gross}(3)$ of net 3 are indicated by arrows. The vertical layer is drawn using solid lines, the horizontal layer using dotted lines.*

3.4.1 Wire interval graph

Suppose that we are give a channel with a partial routing $P_\alpha(I)$ which is in state α of the constructive routing process. In this state, a net $N_i \in \eta_\alpha$ is selected as the next net to be routed. The goal is to represent the partial routing $P_\alpha(I)$ in a wire interval graph $G_{i,\alpha}(V_{i,\alpha}, E_{i,\alpha})$, which is an as accurate as possible representation of the free wiring intervals for net N_i. The free area is limited by contour obstacles in the channel and by territories of other nets. Contour obstacles occur if the spacing between a pair of bottom and top contours is too small to place the wire without causing design rule violations. More formally, a wire of net N_i is not allowed in those (horizontal) intervals where the (vertical) spacing between the pair of contours is smaller than

$$wd_{i,layer} + 2 \times \mathcal{S}_{layer}$$

We define

$$C_\alpha^{side,layer}(N_i)$$

to be the set of forbidden intervals for which the spacing is too small in the sub-channel on *side* and in *layer*. The territories of other (still unrouted) terminals also

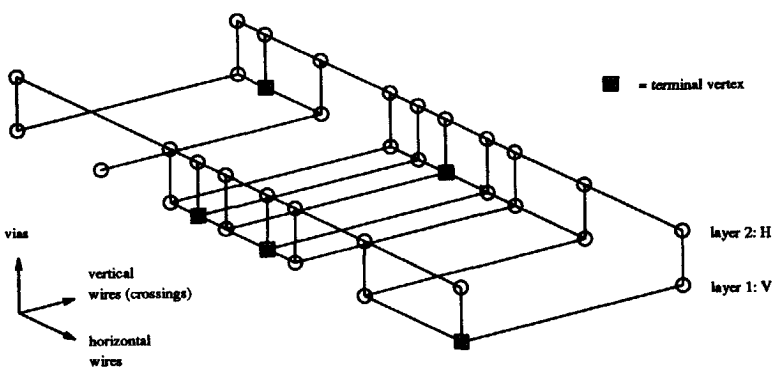

Figure 23: The wire interval graph $G_{3,\alpha}$ for net 3 in the partially routed channel of figure 22.

introduce 'forbidden' intervals. We declare T_j to be the set of intervals protected by the territories of net N_j. Obviously, $T_j = \emptyset$ if net N_j has already been routed. The intervals of

$$T_\alpha^{side,layer}(N_i) = \bigcup_{\{j \in \eta_\alpha \mid j \neq i\}} T_j$$

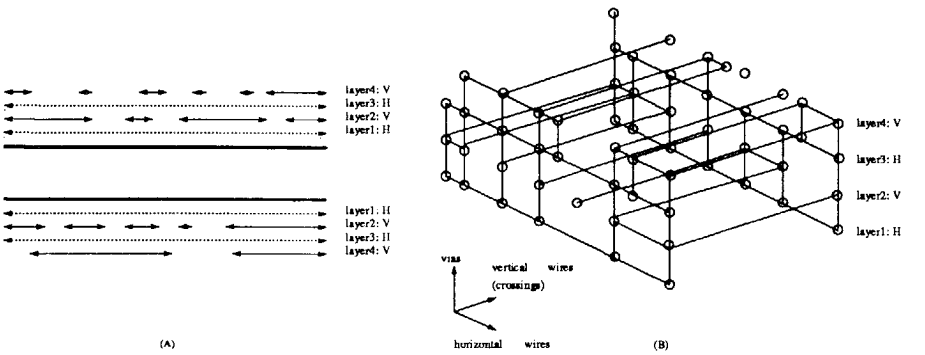

Figure 24: Intervals $F^{side,layer}_{\alpha,gross}$ in a four layer example (left). The right figure shows a 3-D view of the resulting 4-layer wire interval graph.

are the forbidden areas caused by territories in the sub-channel on *side* and in *layer*. It is the union of the territories of all other unrouted nets. Removing all forbidden intervals from the channel yields the *gross free space* of net N_i in state α:

$$F^{side,layer}_{\alpha,gross}(N_i) = \overline{C^{side,layer}_{\alpha}(N_i) \bigcap T^{side,layer}_{\alpha}(N_i)}$$

At this point the minimum spacing between an obstacle and the wire has not yet been taken into account. The *net* free space $F^{side,layer}_{\alpha}(N_i)$ is derived from the gross free space by shrinking both bounds of each interval by $\mathcal{S}_{layer}$ and deleting those intervals of which the resulting width is smaller than $nm_{i,layer}$. This set contains the intervals within which layout patterns for N_i can be placed on top of the contour.

Let $F_{\alpha}(N_i)$ be the set of all free intervals in all layers. The wire interval graph $G_{i,\alpha}(V_{i,\alpha}, E_{i,\alpha})$ is constructed by determining the adjacencies of the intervals. With each interval of $F_{\alpha}(N_i)$ a set of vertices in $G_{i,\alpha}(V_{i,\alpha}, E_{i,\alpha})$ will be associated. We also assign an x-position within the corresponding interval of $F_{\alpha}(N_i)$ to each vertex $v \in V_{i,\alpha}$. There are several ways to construct the interval graph. One of them is shown in figure 25. The interval graph $G_{3,\alpha}$ for the example in figure 22 is shown is figure 23. A complex 4-layer channel and its associated interval graph are shown in figure 24.

3.4.2 Global routing

The global routing of a net N_i is performed on its wire interval graph $G_{i,\alpha}(V_{i,\alpha}, E_{i,\alpha})$. An edge in this graph may represent a via, a horizontal or a vertical wire segment.

1. Represent each interval in $F_\alpha(N_i)$ by a pair of connected vertices. The edge between the two vertices corresponds to a horizontal wire.

2. Find the set of horizontal overlap intervals

$$O_\alpha^{side,layer1-layer2}(N_i) = F_\alpha^{side,layer1}(N_i) \bigcup F_\alpha^{side,layer2}(N_i)$$

for all pairs of adjacent layers and on both sides of the channel. For each of these intervals insert 2 pairs of connected vertices in the graph: one pair for each end of the overlap interval. Of a vertex pair one vertex is assigned to the corresponding interval of $F_\alpha^{side,layer1}(N_i)$, the other to the interval of $F_\alpha^{side,layer2}(N_i)$. The edge between them represents a possible location for a via. If the size of the overlap interval is too small for a via no vertex pair will be generated.

3. Find a similar set of vertical overlap intervals

$$O_\alpha^{layer}(N_i) = F_\alpha^{\mathbf{bottom},layer}(N_i) \bigcup F_\alpha^{\mathbf{top},layer}(N_i)$$

for all vertical layers. For each of these intervals generate 2 pairs of connected vertices. The pairs are placed at opposite sides of the overlap interval. Of a vertex pair, one vertex is assigned to the corresponding interval of $F_\alpha^{\mathbf{top},layer}(N_i)$, the other to the interval of $F_\alpha^{\mathbf{bottom},layer}(N_i)$. The edge between them represents a possible location for a wire to cross the channel. No vertex pair will be generated if the size of an overlap interval is too small for a wire ($< nw_{i,layer}$).

4. To represent the terminal locations of N_i we insert 'terminal vertices' in $G_{i,\alpha}$. A terminal vertex obtains the x-position of the terminal and is assigned to the corresponding interval (and layer) of $F_\alpha(N_i)$. The accuracy of the graph can be enhanced by inserting vertices and edges at aligning positions on the opposite side of the channel and in adjacent layers.

5. Terminals which are already connected to one another outside the channel are called *electrically equivalent*. Only one connection per set of electrically equivalent terminals is required. These terminals could also serve as 'feedthrough' to shorten the paths to other terminals of the net. This property can be represented adequately by zero-weight edges between the vertices of electrically equivalent terminals.

Figure 25: An algorithm to create the interval graph. The steps are illustrated in figure 26.

A number of heuristics can be applied to determine the weights of the edges in this graph. For instance, the weight can depend on the following factors:

- An estimate of the wire length in combination with the resistance in the layer.

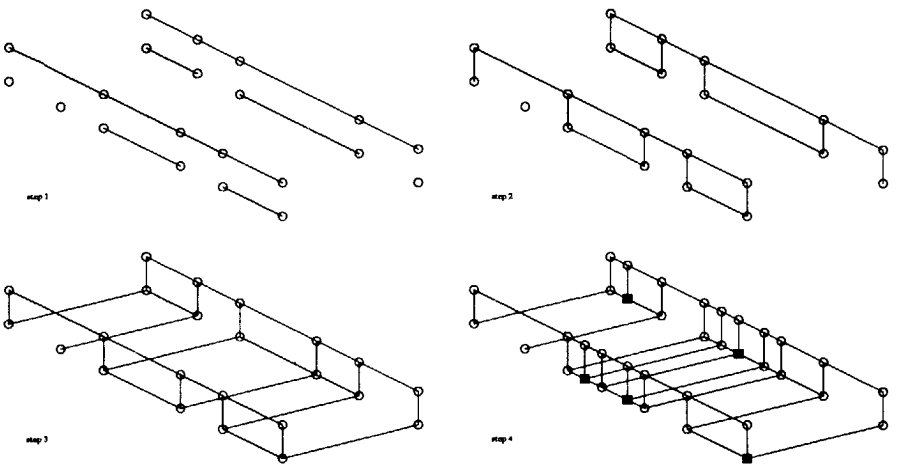

Figure 26: The major steps in the algorithm of figure 25 to create the interval graph of the channel in figure 22.

- The amount of cross-talk we allow between wires in adjacent layers.
- The amount in which the wire will contribute to the critical path for the channel height. The wiring density can be spread evenly over all layers by adjusting this cost parameter dynamically during at each state.

The weight parameters can be tuned to obtain a trade-off between conflicting quality factors. By increasing the cost of a via-vertex, for instance, more wires will be routed in a vertical layer, thereby obviating as many vias as possible (see Figures 27, 28 and 29).

The task of the global router is to find the minimum-cost path connecting all terminal vertices in $G_{i,\alpha}(V_{i,\alpha}, E_{i,\alpha})$. For arbitrary numbers of terminals this is an NP-hard Steiner tree problem. We used a modified version of Dijkstra's shortest path algorithm to find a path between the terminals. It produces acceptable results and never generates a cycle. The worst case complexity of the algorithm is $O(n^2)$ in which n is the number of vertices in $G_{i,\alpha}$.

3.5 Detailed routing

During the detailed routing phase the actual layout will be generated, based on the topological path in the wire interval graph. In this context a trunk is a set of connected

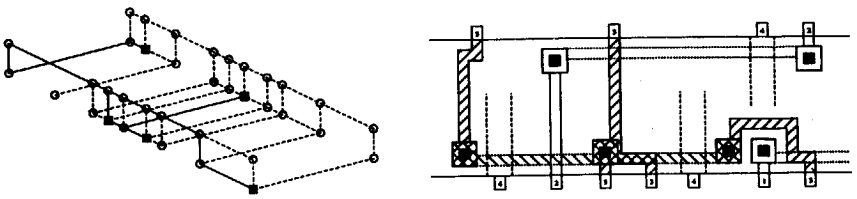

Figure 27: A path found for net 3 in G_3 of the channel in figure 22 (left). On the right the corresponding detailed routing is shown.

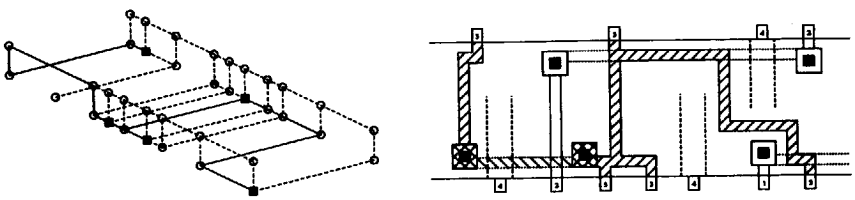

Figure 28: The cost penalties of the edges in G_3 can be used to reduce the number of vias.

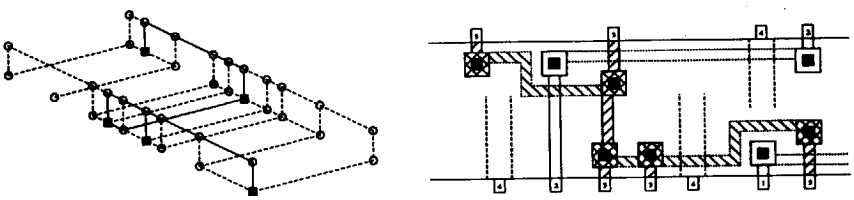

Figure 29: It is also possible to adjust the weights of the edges such that the wire length in the vertical layer is minimized. As a result the number of vias will increase.

vertices in the same layer and on the same side to which the path of the net belongs. The trunks will be wrapped around the contours. Branches in the wire interval graph map to wire segments which cross the channel. A quite straightforward sequence of steps to perform the detailed routing is shown in figure 30.

1. **Via placement.** A number of heuristics are applied to place the via(s) of each trunk. The via position depends mainly on the height of the contour in the horizontal layer. To prevent detours the positions of the other connections of the trunks to be connected are taken into account.

2. **Branch positioning.** The position of a vertical wire crossing the channel is determined according to the (local) contour heights on each side of the channel. Again the position of the other connections is taken into account.

3. **Wire generation in vertical layers.** Once the crossings and the vias have been placed they can be connected by a horizontal wire. In each trunk this wire is wrapped as close as possible around the contour. Some useless wire bends can be prevented by filtering the contour envelope.

4. **Layer conversion of terminals in horizontal layers.** In the following step a wire in a horizontal layer will be generated. Since there are no territories in a horizontal layer this wire could cover some unrouted terminals in the layer. Routability is maintained by diverting the threatened terminals to an adjacent vertical layer. The diversion is performed by generating a via which 'shifts' the terminal into the layer of its territory.

5. **Wire generation in horizontal layers.** As in the vertical layer a horizontal wire is generated which spans all the terminals and vias of a trunk. The wire is bent at minimum spacing around the contour to minimize the channel height. Useless wire jogs will be removed by a wire-straightening post processor.

6. **Contour update.** The contours generated in the previous steps cover the new wires of each trunk exactly. These new contours will be merged with the existing contours.

7. **Weight factor adjustment.** For performance reasons the wire interval graph $G_{i,\alpha}$ was derived from the interval graph in the previous state. The critical path in each layer is found by comparing the contours on the bottom and top. The weight of the edges in G which correspond more or less with the critical path is increased.

Figure 30: A detailed routing algorithm. It converts the global routing in the wire interval graph into a physical wire pattern. This pattern is added to the partial routing.

3.6 Scheduler

The main task of the scheduler is to determine the order in which the nets will be presented to the net router. In trunk-based routers the vertical constraint graph is used to guide the net ordering and routing. The VCG is built on the concept that each pair of 'facing' terminals induce a constraint. It does, however, not capture the ability to avoid conflicting crossings (vertical constraints) by jogging. It is a too rigid

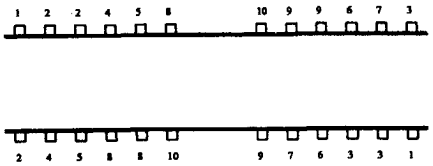

Figure 31: A gridded example channel with a cyclic vertical constraint (after [6]).

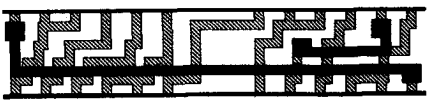

Figure 32: Routing produced by this algorithm for the channel in figure 31.

and abstract model to guide the ordering for this channel router. In our territory framework there are usually only a few nets which cannot be routed immediately due to the large search space in $G_{i,\alpha}$. In the example of figure 31, for instance, all nets are routable. The wire of some of the nets, however, contain detours and additional vias for doglegs. Their routing could be postponed to allow the more efficient nets to be routed first. There is a fair chance that with routing other nets first better paths become available. As more nets are routed, more territories (which are the main obstacles in the channel) are removed.

To capture the efficiency of a net N_i we derive a 'quality value' from its path in the graph $G_{i,\alpha}$. The length of the detours and the relative amount of vias per terminal could be used to compose this value. The bar graph in figure 33 shows the initial quality values ($\alpha = 0$) for the nets in the channel of figure 31. In this channel net 10 should be routed first since is the only net that does not require a dogleg or a detour. Therefore it has the highest quality value. Due to the routing of this net the quality values of nets 8 and 9 improves at state $\alpha = 1$ (figure 34). A straightforward algorithm for net scheduling would be to select every time the net with the best quality value. The final result of this procedure is shown in figure 32.

Determining the quality of a single net N_i is rather 'expensive' because it requires the construction of a wire interval graph $G_{i,\alpha}$ and a global routing. A total of $\frac{1}{2}|\eta|(|\eta|+1)$ of these calculations is required since the quality of all unrouted nets will have to be redetermined at each state α. To reduce the CPU-time consumption we apply a more 'greedy' scheduling algorithm. Instead of deriving an accurate quality value from the path in $G_{i,\alpha}$ a rough estimate of the quality is used, based on the following criteria:

- The 'single-side' criterion. Single-sided nets are more likely to have a high quality value since they can always be routed without detours and doglegs, provided there is no overlap of territories. Moreover, routing these nets removes

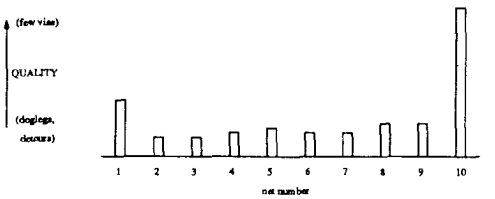

Figure 33: A bargraph showing the quality values of each net in the channel of figure 31, before any net is routed.

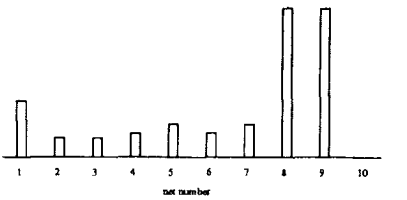

Figure 34: The quality values in $P_1(I)$ after net 10 has been routed.

territories without introducing new obstacles. In this way the number of possible sources of constraints decreases and, with that, the wiring freedom increases.

- A user-specified ordering of the net. This allows a way to impose a global wire ordering on the channel. This could prevent unnecessary wire twisting in adjacent channels (see chapter 6).

- Additional heuristics such as the number of terminals of a net.

The nets will be presented to the net router in this order. Suppose a net N_i is selected at some state α during the constructive routing process. The wires of N_i will not be added to the partial routing in two cases:

hard constraint: The net is not routable because there is no path in $G_{i,\alpha}$ which connects all its terminals.

soft constraint: The implementation of the path the net N_i does not come up with a certain threshold value q_{min} for the quality. This 'quality standard' is initially such that only nets without detours and doglegs are routed.

A successful routing of N_i is more likely to occur due to the net ordering. If the routing of N_i fails, the next net in order will be attempted in state α. The process continues until all nets are routed or all remaining (unrouted) nets have a constraint. In the latter case the quality standards must be temporarily lowered to enlarge the search space in $G_{i,\alpha}$. This algorithm is shown in figure 35. Although the worst case complexity of this greedy algorithm is still $O(|\eta|^2)$ the behavior for most practical channels is much better.

We added a simple heuristic to this algorithm to reduce the number of constraints during the routing process. Usually, the quality value of a net changes only if a

```
1   prepare routing area; make territories.
2   α := 0; U := η; H := ∅; S := ∅;
3   estimate the quality value q_i for each net N_i ∈ U;
4   determine threshold quality q_min;
5   while U ∪ S ≠ ∅ do
        { /* while a routable net exists */
6       select the net N_i ∈ U ∪ S which has the highest quality;
7       U := U\N_i;
8       route N_i, which results in a real quality value q_i;
9       if q_i ≥ q_min then
            { /* accept */
10          α := α + 1; U := U ∪ S ∪ H;
11          S := ∅; H := ∅;
12          re-evaluate the estimated q_i for each net N_i ∈ U;
13          re-determine threshold quality q_min;
            }
        else
            { /* reject */
14          remove the wires of N_i from the partial routing;
15          if q_i > 0 then
16              S := S ∪ N_i; /* soft constraint */
            else
17              H := H ∪ N_i; /* hard constraint */
18          if U = ∅ then
19              lower threshold quality q_min;
            }
        }
20  shift the top sub-channel as close as possible towards the
    bottom sub-channel;
```

Figure 35: The greedy scheduling algorithm which is used in the channel router. U is the set of unrouted nets, S is the set of nets with a soft constraint and H is the set of nets with a hard constraint. The state α indicates the number of routed nets in the partial routing. The net router is called in line 8.

terminal adjacent to a terminal of that net is routed. The routing of an adjacent terminal erases a territory which, on its turn, could have enabled a better path. 'Remote' nets usually have little influence on the quality. Therefore, we reattempt a

rejected net only if a net in its neighbourhood was routed.

Both algorithms discussed above usually achieve 100% completion. For the majority of channels (including the 'difficult example') even doglegs are not required. Only in a few cases do all remaining unrouted nets have a hard constraint. If the hard constraints are not caused by terminal spacing violations, the channel must be made longer by adding a column. In this very rare case the problem is caused by the absence of edges in $G_{i,\alpha}$ to cross the channel. By enlarging the channel it is always possible to create these edges. The only other (and more likely) source of hard constraints is spacing violations between territories. This problem can be solved by shifting terminals apart (see 3.7.1).

3.7 Further enhancements of the contour framework

3.7.1 Overlapping territories

In order to ensure 100% completion, it is required that the spacing between two adjacent territories is at least $\mathcal{S}_{layer}$. All terminals can be connected to a horizontal layer in this way. This results in an additional spacing constraint between terminals which have territories in the same layer. In many cases 100% completion can be achieved even if the spacing between some territories is smaller. This is usually not possible if there are arrays of overlapping territories. Our solution is to remove all wires from the channel if some nets are unroutable due to a territory spacing violation. Prior to the second attempt to route the channel, we shift the overlapping territories apart to fulfill the spacing constraint. A river routing pattern, which we call the *fan-out* pattern, connects each original terminal to its new territory. The territories can be shifted such that the height of the required fan-out pattern is minimal. Obviously the sum of the widths of the territories must be smaller than the length of the channel, otherwise the channel length must be increased.

3.7.2 Wire-straightening

During the routing process each wire is wrapped tightly around the existing contour to obtain an as compact as possible layout pattern. This process of 'greedy' compaction inevitably introduces many useless wire bends. Although the wire pattern is (design-rule) correct, capricious wire patterns have a number of drawbacks:

- The wires are longer than necessary, which introduces additional capacitance and resistance.

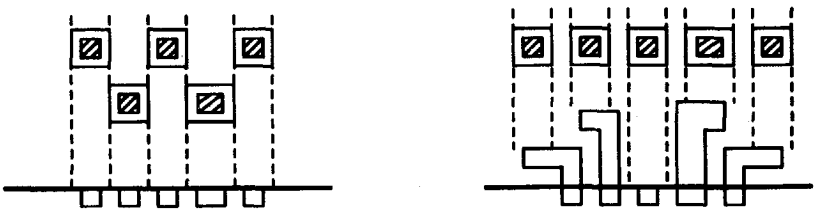

Figure 36: If spacing between the territories is too small (as is illustrated on the left), the territories overlap because there is not enough room for the vias. This may make some nets unroutable. Using a fan-out *pattern (as shown on the right) the terminals can be shifted apart such that the territory overlaps are removed.*

- Bumpy wire patterns require more disk space for storage, and more CPU-resources for post-layout verification (e.g. design-rule checks, layout-to-circuit extraction).

- The patterns are less aesthetic[1].

Fortunately, the wires in the channel can be straightened very effectively. Our wire-straightening algorithm uses two contours which mark the upper and lower limits of the area within which a wire can be re-shaped. These contours traverse through the pile of wires in reverse order as the wires were routed. Each time a wire is stretched between the contours such that it contains the fewest number of bends. Special care has to be taken while shifting a via since it connects to another layer. The 'shiftability' of each via is determined by scanning the contours in the adjacent layer. During the wire-straightening process it is also possible to reduce the length of wires in critical layers such as polysilicon. Vias can be shifted such that the wire length in these layers is minimized.

Notice that this wire straightener is not a conventional post-processor similar to the ones often used on grid-based routers (e.g. [22, 13]). It does not reduce the channel height, nor does it reduce the number of vias.

[1]Although this is an entirely irrational point, the results of a layout generation program will be judged by the 'looks' of the layout it produces. Any child is able to detect silly wires in a layout. A picture of the chip is often the 'hardest' evidence of years of work which can be presented to high-level management. To a layman, a picture showing many silly wires might give a wrong impression, even if those wires are not critical at all. Inefficiency in other stages of the chip design (e.g. at the functional level or at the circuit level) is much less evident.

3.8 Experimental results

A VHV-type version of the algorithm was implemented in C under the UNIX operating system. The program is integrated in our macro-cell routing system (see chapter 7). and in a number of layout assembly tools.

For the results presented here, we tuned the weight factors of the algorithm such that a trade-off between the number of vias and the channel height was achieved. No emphasis was put on reducing the wire length in certain layers. The quality measure for soft constraints was initially set such that all nets with doglegs are rejected.

The foremost benchmark for any channel router is *Deutsch's Difficult Example* [23]. Results for this is (gridded) channel have been published for nearly all channel routing algorithms. The density of the difficult example is 19. Table 1 shows the result of our algorithm on this example compared to a number of well-known routers. The design rules as specified in [24] were used for all results in this table. They specify two layers of interconnect. The wire width $\mathcal{W}$ as well as the spacing $\mathcal{S}$ are 1 unit, while the via sizes $\mathcal{S}_{l1,l2}$ and $\mathcal{S}_{l2,l1}$ are 2 units. To facilitate a comparison between the gridded results (expressed in the number of tracks) and gridless results, the grid-based channels were compacted in one dimension using the algorithm described in [22].

reference		no. of tracks	compacted height	no. of vias
[38, 68]	("left edge")	31	n.a.	290
[23]	("dogleg")	21	49	346
[91]	("efficient")	20	47	308
[73]	("greedy")	20	48	347
[6]	("hierarchical")	19	46	354
[71]	("YACR2")	19	47	287
[22]	("compacted")	19	45	n.a.
[74]	(manhattan)	gridless	46	271
[74]	(octagonal)	gridless	44.33	271
[13]	(via shift)	19	42	275
this work		gridless	45	265

Table 1: Comparison of various channel routers for Deutsch's Difficult Example. *For all channels the terminal pitch is 4. Sources: [71, 22, 74, 13, 68].*

The channel height of our gridless router depends very much on the terminal pitch. With a 2.0 terminal pitch the territories overlap which resulted in a massive

terminal pitch	original channel	$+\frac{1}{2}$ pitch shift	$-\frac{1}{2}$ pitch shift
2	353	356	357
3	48	52	54
3.5	48	49	53
4	45	46	46
5	43	45	46
6	43	44	44
7	42	43	43
10	41	41	41

Table 2: The channel height of Deutsch's (new) Difficult Examples *for various terminal pitches.*

Channel	THIS ALGORITHM		GLITTER ([17])	
	height	CPU-sec.†	compacted height	CPU-sec.‡
RIGHT ($+\frac{1}{2}$ pitch)	46	45	56	66
LEFT ($-\frac{1}{2}$ pitch)	46	47	62	78

Table 3: Comparison of the results for Deutsch's new difficult examples. *The terminal spacing is 4.* † *Run on an Apollo* DN-3000 *($\approx$ 1* MIPS*).* ‡ *Run on an* IBM *3090 mainframe ($>$ 20* MIPS*).*

fanout-pattern. The minimum pitch to prevent the overlapping territories is 3. One can expect that, in general, the channel height will reduce with increasing terminal pitch because better paths become available. As table 2 shows, this is confirmed by experiments.

Recently, two new and more difficult examples were suggested by the author of the original problem [24]. Essentially they are the same channels as the original difficult example. The difference is that the top half of the channel is shifted either half a terminal pitch to the left or to the right. In this way the number of vertical constraints increases considerably. Trunk-based algorithms are likely to run into considerable problems due to the uncreased amount of vertical constraints. Our territory algorithm should be less affected as long as there is sufficient space to cross the channel. Therefore Deutsch's new difficult examples from an interesting test case. Table 2 summarizes our results for them. Only with smaller terminal pitches

Figure 37: A result on Deutsch's new Difficult example. The figure shows the channel with $+\frac{1}{2}$ pitch shift of the top half of the channel ('RIGHT').

the channel height is affected by the offset in the new difficult examples. The number of rejected nets (with hard or soft constraints) increases considerably. In table 4 this is shown by the 'acceptance ratio', which is number of accepted attempts as a percentage of the total number of attempts to route a net.

To the best of our knowledge only Chen has published results for the new difficult examples so far [17]. He used his *glitter* gridless channel router [16]. As can be seen in table 3, our algorithm produces significantly smaller channels than glitter's trunk-graph based approach. Our constructive approach also burns considerably less CPU-time. Figure 37 shows a plot of our result for one of the 'new difficult examples'.

In table 4 the results of a number of channel are shown. The channels marked (6) were obtained from a large real-life chip which was processed in a double-metal 1.6μ CMOS process. They have rugged channel boundaries and variable-width wires for power and clock wires. A few 'fan-out' wire patterns were required in most of them to separate territories or to route overlapping terminals. Figure 39 shows one of these channels.

As can be derived from table 4 the CPU-time consumption depends very much on the complexity of the problem. If many obstacles (territories) are present the number of rejected attempts to route a net will increase due to hard and soft constraints. The execution time also depends on the number of events (contour-bends) present in the channel. The contour bends are caused by terminals. We used a number of test channels to measure the CPU-time consumption under controllable conditions. The channels were generated using a random generator such that they only differ in the number of terminals and as little as possible in other properties. The boundaries of

channel	no. of nets	no. of terms	channel height	no. of vias	total wire length(1)	accept. ratio(2)	CPU-time (3)
diff.(4)	72	302	48	264	12955	61%	11s/16s
diff. $+\frac{1}{2}$ (4)	72	302	52	271	13417	16%	42s/45s
diff. $-\frac{1}{2}$ (4)	72	302	54	272	13616	10%	44s/47s
CK2(5)	21	75	271	57	17456	100%	2s/5s
chan1(6)	40	94	(7)	85	2343	95%	6s/8s
chan4(6)	48	142	(7)	130	2317	63%	14s/18s
chan6(6)	11	22	(7)	18	3146	93%	2s/3s
chan7(6)	10	73	(7)	61	5890	85%	5s/7s

1) The total wire length after straightening.

2) The acceptance ratio of the net scheduler. This is the number of successful attempts as percentage of the total number of attempts to route a net.

3) CPU-time on an Apollo DN-3000 work station ($\approx$ 1 MIPS) including all overhead such as database I/O. The first value is for routing without wire straightening. The second value includes wire straightening.

4) Deutsch's Difficult Example, terminal pitch = 3.

5) The second example channel published as figure 6 in [16]. The units are millimeters.

6) Variable wire width channels, see text. The units are microns.

7) Irregular channel boundaries.

Table 4: Data and results for various channels.

the channels are packed with terminals at minimum spacing. No significant differences were detected between channels with 2, 3 or 4 terminals per net. Figure 38 shows the results of the tests.

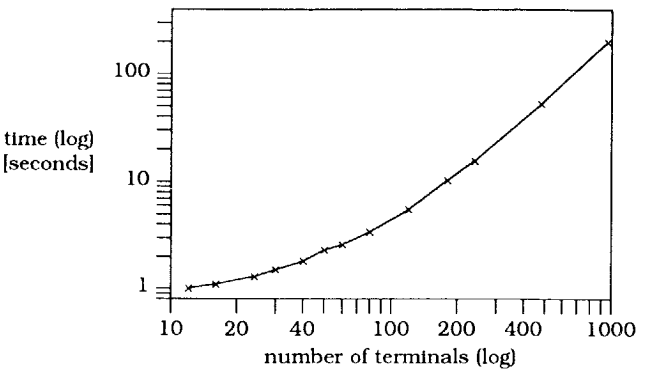

Figure 38: Total execution time versus the number of terminals in random channels. The data were obtained on a DN-3000 work station ($\approx$ 1 MIPS).

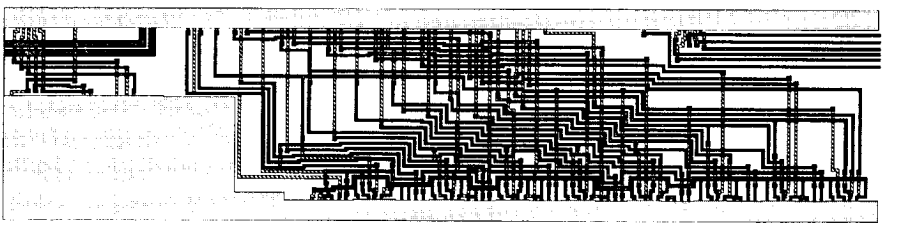

Figure 39: A complex channel taken from a real-life chip.

Chapter 4

Grid-based channel routing on sea-of-gates

4.1 Through-the-module detailed routing

For most full-custom technologies the compact design of the modules makes that the area which the modules occupy can hardly be used for interconnect. This forces the interconnect to take place in the areas around the modules. On a semi-custom sea-of-gates chip the situation is slightly different. Due to the design of the fixed pattern of the transistors, an implementation of a module has generally some excess routing capacity. The circuit in figure 40, for example, has a considerable amount of routing capacity left over in the first metal layer, while the second metal layer was not used at all. The routing capacity of the 'porous' sea-of-gates modules would be wasted if the interconnect is restricted to the area around the blocks, as was the case for full-custom macro-cell routing.

Many *over-the-cell* channel routers have been introduced (e.g. [25, 46, 78, 20]) which make use of the available capacity in the modules. They subdivide the channel into three parts: a middle part which is an ordinary channel and two parts which are located over the upper and lower modules. The latter parts are single-layer routing areas which can therefore only be used for planar routing patterns. The general approach to route these channels is as follows. As the first step, as many as possible terminals are routed over the cells by planar wire patterns. As a result, some terminals are connected. In a second step the net list for the remaining channel is constructed by choosing net segments between the terminals. Finally the channel area is routed by a channel router.

An over-the-cell channel router must be seen as a pre-processor for an ordinary channel router. It reduces the channel density by pre-routing some of the nets in a

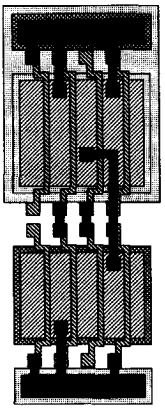

Figure 40: A two-input nand implemented on the 'fishbone' sea-of-gates image [37]. Only a few metal 1 wires (the dark top-level patterns) are required to implement the circuit, leaving a considerable amount of wiring capacity for future interconnect.

single layer *over* (and not *through*) the module. The approach is suitable for gate-arrays and standard cells in which all cells are implemented in one metal layer. It is, however, not possible to make use of the entire routing capacity because over-the-cell routing is restricted to a single layer without obstacles. On a sea-of-gates the routing capacity is generally available in more layers. Moreover, each of the layers may contain obstacles.

A more rigorous way to take the interconnection capacity of the modules into account, is to model the interconnection pattern of the modules as obstacles in the routing area. In this way there is no distinction between module area and routing area. Due to the presence of obstacles, the problem must be attacked by a maze router. As discussed in chapter 2, the most prominent drawback of the maze router is the uncertainty that it can complete the routing successfully. It relies on very accurate placement and global routing algorithms.

In this section we will present a method which combines the capabilities of a maze router with the characteristics of a channel router. We will define a new type of routing area which, unlike the channel router, may contain obstacles, and of which the size is adjustable, unlike the maze router. At first sight the method appears to have very few similarities with the contour-based routing presented in the previous chapter. The underlying method, however, is again based on constructive routing using territories.

4.2 The routing area

4.2.1 Grid graph

The fixed slots of the transistors in a sea-of-gates image and the nature of the obstacles make the grid graph an appropriate model for our purpose. We start off from a routing area (we will call it just *channel*) which is very similar to a grid-based maze-routing problem. We can assume that the channel is a rectangular area. Obstacles in the channel are modeled by removing specific edges in the grid graph. In most cases the wire patterns of the modules form the obstacles. With this ability to model obstacles any channel shape can be captured by the rectangular channel.

4.2.2 Tear lines

The channel is dissected into two parts by a straight line which we called the *tear line*. If the tear line is horizontal we call the channel horizontal. Similarly, a vertical channel is dissected by a vertical tear line. For the following description we will assume a horizontal channel. The tear line dissects the grid graph $G(V, E)$ into two subgraphs: G^+ containing the upper part of the channel and G^- containing the lower part of the channel. It marks the junction along which the channel can be enlarged to create additional routing space. To ensure that the additional space is useful, we require that the tear line may only be crossed by wires in a perpendicular layer. In the case of a horizontal channel the tear line must be crossed in a vertical layer. This can be easily enforced by removing the edges of the grid graph corresponding to horizontal layers which are dissected by the tear line. Obstacles may cross the tear line only in a vertical layer and they must be 'tearable' at the tear line.

4.2.3 Territories

As with the conventional channel routing problem, we distinguish two types of terminals:

Floating terminals which are located on the left and right sides of the channel and which are in a horizontal layer. The exact position and layer of the floating terminals will be determined by the channel router.

Fixed terminals which are located in a fixed position and layer in the routing area. In many cases the terminal is placed on a wire, which makes the entire wire an electrically equivalent *terminal pattern*.

We will use the familiar 'territory framework' to ensure the routability of the fixed terminals. For this context the territory of a terminal is a temporary wire connecting the terminal to the tear line. It ensures that the terminal can be connected to a horizontal wire by guarding an 'escape route' to the tear line. Wires of other nets are blocked by the territory. Since the tear line may only be crossed in a vertical layer, the territory connects to the tear line in a vertical layer.

The presence of the territory for each terminal is an essential component in our strategy to ensure 100% completion. It can be seen as a way to convert a 2-dimensional configuration of terminals, which is typical for the maze routing problem, into a channel router-like 1-dimensional arrangement of terminals along a line. The guarantee for completion will be lost if the conversion fails (that is, if not all territories can be found). For a channel router which is primarily targeted for general-cell interconnect, this is not a severe restriction. The terminals of a general-cell module must be connectable to the module boundary.

4.3 Primary algorithm

Once all territories have been generated, the actual routing process can take place. Accrding to our constructive concept, the nets are routed one at the time by a *net router*. It performs maze routing of a single net in the routing area. Any appropriate maze router can be used for that purpose. In a prototype we adopted a version of Dijkstra's vertex expansion algorithm. On the grid framework this algorithm can take into account various peculiarities of the underlying sea-of-gates image while ensuring to generate a path if one exists. It can be modified easily to prevent it from generating a wire which crosses the tear line in a horizontal layer. Another advantage of this wire generator is its ability to handle an arbitrary number of layers.

A *scheduler* again determines the order in which the nets will be routed. If there is insufficient routing area, the channel is 'torn open' along the tear line. All wires crossing the tear line (including the territories) are stretched such that the original connectivity remains unchanged. Since the tear line is only crossed by wires in a vertical layer, this operation will always result in free horizontal tracks alongside the tear line. With that, the routability of single-sided nets is ensured. Figure 41 shows the routing process in an example channel. In the next sections we will discuss some aspects of this router in more detail.

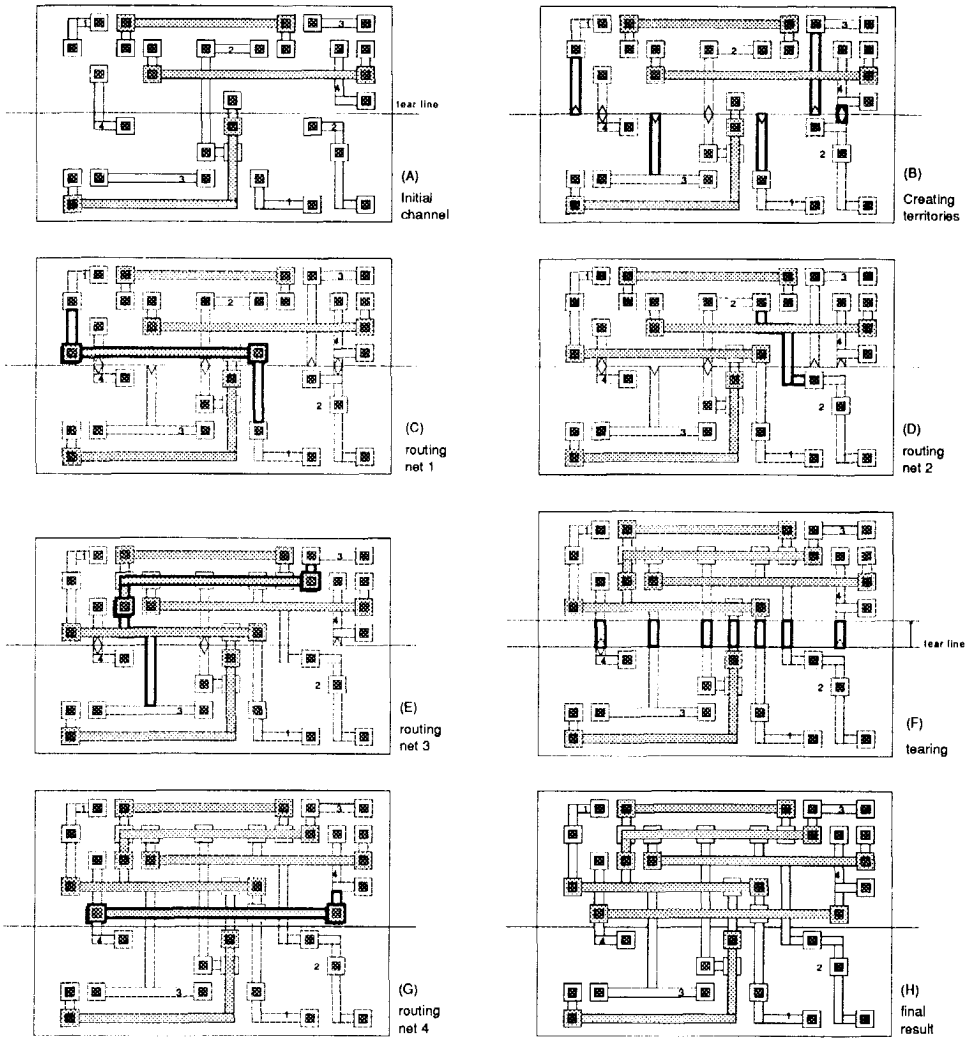

Figure 41: The routing process illustrated on a small 4-net example channel. The channel is dissected by a tear line *(a). Notice that the channel contains obstacles and also that the terminals are placed on wire patterns. A* territory *(b) connects each terminal to the tear line (marked by the triangles). In the next steps (c-e), the nets are routed one at a time, using the entire routing region. Net 4 cannot be routed without allocating additional wire space (f,g). The channel 'tears' open along the line to accommodate the additional wire.*

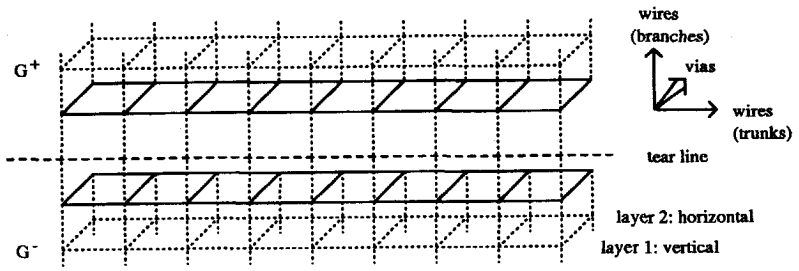

Figure 42: An example of the cambium *in a two-layer channel. It consists of two tracks alongside the tear line (indicated by the solid lines). The tear line can only be crossed in the vertical layer.*

4.4 Cambium

This channel router relies on the fact that the wires can be routed through the modules, avoiding the obstacles. This capability is hard to predict. In some cases the major part of the modules area is unused. In the worst case, however, all paths are blocked by obstacles. This causes the nets to be routed in the area which is 'grown' alongside the tear line. To model this ability to grow, two additional tracks are inserted at the tear line junction between G^+ and G^- in the grid graph. One track is placed above the tear line, the other below it (see figure 42). We metaphysically call them the *cambium*[1] of the channel. The two tracks in the cambium do not map onto physical positions in the channel. They only model the ability to grow alongside the tear line.

The channel must be torn open along the tear line to accommodate the wire if a horizontal track of the cambium is used during the routing of a net. With that, the grid graph is also grown by inserting a new row in the cambium. The cambium should only grow if no feasible path is available through the cells. This can be enforced by assigning much higher cost values to the horizontal edges in the cambium than to comparable edges in the grid graph. The available routing resources in the modules are consumed first in this way.

For example, consider the channel in figure 43. The height of the channel is initially 0 tracks. A similar situation will occur if no routing capacity is available through the modules. The terminals are in the upper or in the lower row. Their territories are vertical wires (branches) touching the tear line. Notice that the grid

[1]The cambium is the line of tissue which is the center of growing activity in a tree or a plant. On one side the cambium deposits wood, on the other side it deposits bark.

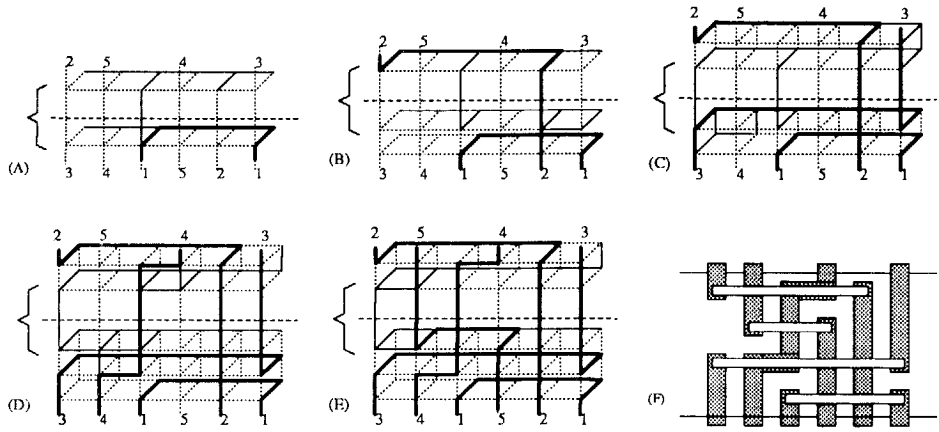

Figure 43: An illustration of the growing process alongside the cambium. The example depicts a two-layer channel without any routing capacity through the modules. The cambium is indicated by the brace. Figures a-f show the sequential routing of nets 1 to 5, respectively. Each time the routing of a net requires a horizontal segment in the cambium, the channel is enlarged by tearing it open along the tear line, pushing the track of the cambium into the channel. Figure f shows the layout of the final result. The dotted edges in the grid graph are blocked for the routing of a net by the territories or wires of other nets. Notice that the cyclic vertical constraint between nets 4 and 5 is solved by jogging in the vertical layer.

graph, including cambium, resembles the global routing graph of the contour based channel router in figure 29 (page 46). The bottom side of the channel is modeled by the lower parts of the cambium; the top side is modeled by the upper row of the cambium. The edges crossing the tear line are the *crossings* which may carry branches to connect to the opposite side of the channel. Some of the edges are occupied by wires, some are occupied by the territories of other nets. The routing problem is therefore very similar to the gridless channel router in chapter 3. Again, the ability to jog makes the vertical constraint graph inappropriate to determine the routing order. Therefore the greedy scheduling algorithm as presented in chapter 3 is used.

The grid graph captures many possible configurations for the implementation of the net. This includes adventurous detours through the grid graph. The free tracks in the cambium together with the territories, ensure that each terminal can be connected to a horizontal wire. In 2.4 we proved that only in a few very rare cases the routing of a net cannot be completed by a territory-based algorithm. The ability to jog does not change the completion guarantee.

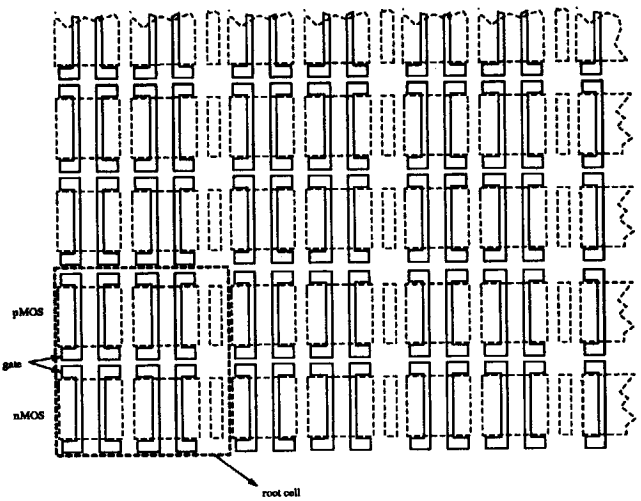

Figure 44: The left bottom corner of an example sea-of-gates chip. The image is made up of a root cell *(the dashed box) which is repeated a large number of times in both directions.*

Another important observation is that, apart from the 'active' tear line which splits the graph into two parts, there can any number of 'dormant' horizontal or vertical tear lines. The Lee-type wire generator can be adapted such that each of the tear lines in the channel can only be crossed by a wire segment perpendicular to the tear line. This enables the channels to overlap. In chapter 5 we will come back to the opportunities which arise from overlapping routing areas.

4.5 Adaptation to a sea-of-gates image

In the previous description, we did not take into account any of the characteristics of the underlying sea-of-gates chip. In this section we will address various aspects of the application of the channel router to a sea-of-gates chip.

4.5.1 Sea-of-gates structure

As with any semi-custom chip, the shapes, positions and sizes of the transistors of a sea-of-gates are fixed. This non-programmable part of a sea-of-gates is called the *image*. The image includes all non-programmable masks and sometimes a fixed

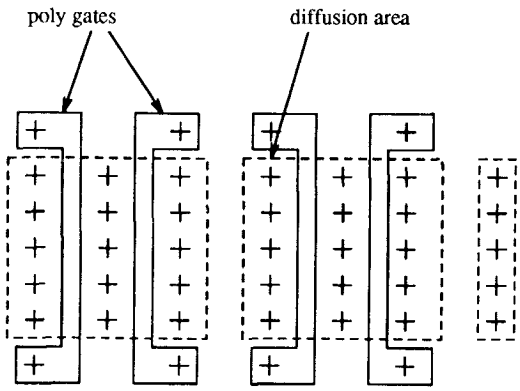

Figure 45: The bottom half of the root cell in the image in figure 44 on page 66. The crosses mark the positions where vias can be placed to connect the transistors.

pattern in the programmable masks (such as for power wires). Many sea-of-gates images have pre-defined power rails in the first metal layer. On the popular gate-isolation type images [58] and even on its most recent high-density 3-layer derivatives (e.g. [87]) a lattice of fat power metal 1 rails is running parallel over the entire length of the chip. This forces the metal 1 layer globally into the horizontal direction. As a result, metal 2 is generally used for vertical wires.

4.5.2 Regularity of the image

In nearly all practical realizations the image is made up of a large number of repetitions of a *root cell* (see figure 44). Compared to the total size of the core area of the chip, the root cell is very small. It usually consists of only a few transistors, sufficient to make an elementary logic gate. A small core cell is generally preferred over a larger one. Several complications occur if the core cell is much larger than a typical atomic cell (e.g. a 2-input nand). First, there is the practical problem that a number of layout implementations of the same logic cell must be designed and stored. Secondly, and most important, it becomes harder to hide the peculiarities of the image for the layout algorithms.

We assume that the routing area is modeled on a grid graph G. Underlying the grid is the image, which consists of a repetition of the root cell. We define the vector $\vec{b} = (b_x, b_y)$ to contain the dimensions (that is, the width b_x and height b_y) of the bounding box of this root cell, expressed in grid units. The vector $\vec{b}$ is called the *repetition vector*, because it determines the offset over which the image repeats. The

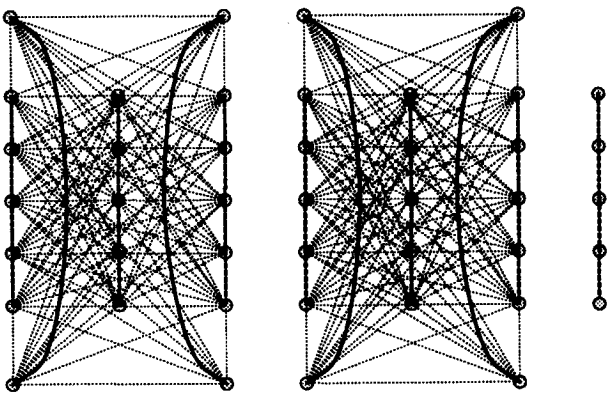

Figure 46: The feed graph $F(V, E)$ of the root cell in figure 45. The types of the edges are indicated in the figure: solid lines mark the feed edges, *the dark dashed lines mark the* restricted feed edges *and the light dotted lines mark the* cohesion edges. *Notice that the root cell contains three disjunct graphs.*

vector $\vec{b}$ in the image of figure 44, for instance, is $(7, 14)$. On a semi-custom chip any module may only be translated over the vector $\vec{t} = K\vec{b}$:

$$\begin{pmatrix} t_x \\ t_y \end{pmatrix} = \begin{pmatrix} k_x & 0 \\ 0 & k_y \end{pmatrix} \begin{pmatrix} b_x \\ b_y \end{pmatrix}$$

in which k_x and k_y are integers. Any other translation will result in an incorrect circuit. The repetition vector $\vec{b}$ therefore determines the minimum increments by which the channel can be torn open.

4.5.3 Routing graph containing feeds

Although not user-definable, certain features in the image can be used for routing purposes. The gates in figure 44, for instance, have two connections. Therefore they could also serve as a wire segment. We call these image-specific features *feeds*. They are modeled by a *feed graph* $F(V, E)$, in which the vertices correspond to specific 'grid' location in the routing area. On each of these grid positions a via could be placed to connect from the metal 1 layer to the feeds.

An edge between two vertices exists if there is some electrical dependency between those vertices. Figure 46 shows an example of a feed graph. We deliberately use the rather vague term 'electrical dependency' because three types for the edges are distinguished:

1. *Free feeds*, marking equipotential vertices which can actually be used by the router as an ordinary wire segment. The two polysilicon gate connections in figure 45, for example, can be used as a feed. In figure 46 the dark solid lines represent the corresponding edges.

2. *Restricted feeds*, which also mark equipotential vertices, but which cannot be used to feed through a wire. In figure 45 there are 5 equipotential locations connecting the source (or drain) of each transistor. The resistance of the diffusion source is too high, however, to use it as a feed. Furthermore, since the connection is made through active area, the transistors adjacent to the feeds are not necessarily shut off. The edges of restricted feeds are marked by the dark dashed lines in figure 46.

3. *Cohesion edges*, marking any other electrical dependency. They indicate any electrical interference between the two vertices it spans. In figure 46 the cohesion edges are indicated by the light dotted lines.

The feed graph F is appended to the grid graph by inserting edges between the vertices of corresponding via positions. A Lee-type detailed router can use the free feeds to implement wires.

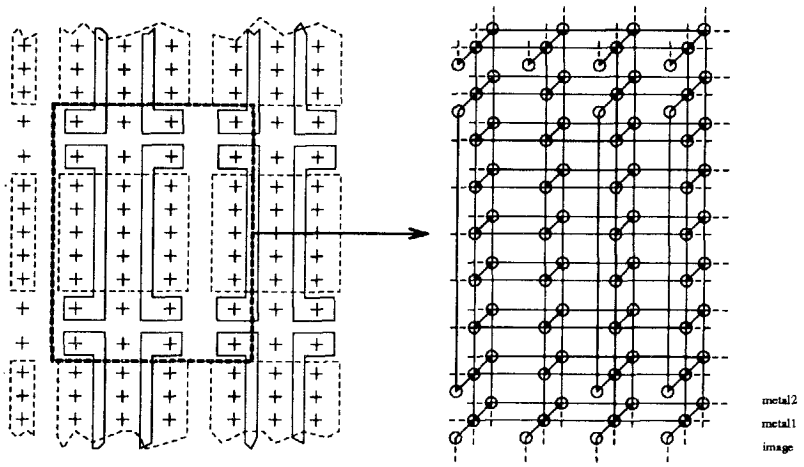

Figure 47: A small portion of the image of figure 44 on page 66 (left). On the right the associated routing graph (including free feeds) is shown.

4.5.4 Locations for tear lines

Some constraints are imposed on the positions of the tear lines. They are required to prevent the router from tearing apart certain 'atomic' features in the image. In the feed graph F the cohesion edges indicate the electric dependencies of the image. As a result, the tear line should not cut any edges in F. Cutting along these 'natural' lines will not affect the electrical circuit.

In a gate-isolation image (see figure 40) no vertical tear line can be defined without cutting cohesion edges in F. In that case the isolation transistors can be used to remove cohesion edges. In general, the modules for a gate-isolation sea-of-gates image are already terminated by an isolation transistor.

Chapter 5

The context of the channel router

The channel routers presented in chapters 3 and 4 are used to route only a small region of the chip. The circuit is generally partitioned in many such regions. In this chapter, we will address a strategy to define the regions in a large circuit such that the strong points of the channel router (its 100% completion guarantee and load adjustable area) are fully exploited. The flexible size of the channel area will impose some constraints on the channel definition.

In the gate-array and standard-cell design styles the decomposition into sub-regions is fixed: alternating rows of cells and channels. In this chapter, we will address the problem of region definition for general-cell layout. Since the modules have non-uniform sizes, the subdivision into routing areas is less dogmatic than in the row based approaches. The first step to attack the general-cell layout problem is the placement of the modules. For simplicity, it is generally assumed that the modules have a rectangular shape, which explains why they are sometimes called *blocks*. The placement is a non-overlapping arrangement of the modules in the chip area. In most technologies the area occupied by the modules is blocked for wires, forcing most of the routing to take place in the areas around the modules.

5.1 The channel routing order

5.1.1 The channel digraph

The elementary junction between two channels is the 'T' type intersection as depicted in figure 73 on page 98. A fixed boundary of a channel, represented by the cross bar of the T, meets the floating boundary of another channel, which is represented by the base of the T. The width of the floating boundary must be known to determine the fixed boundary of the cross-bar channel. Therefore the 'base' channel must be routed

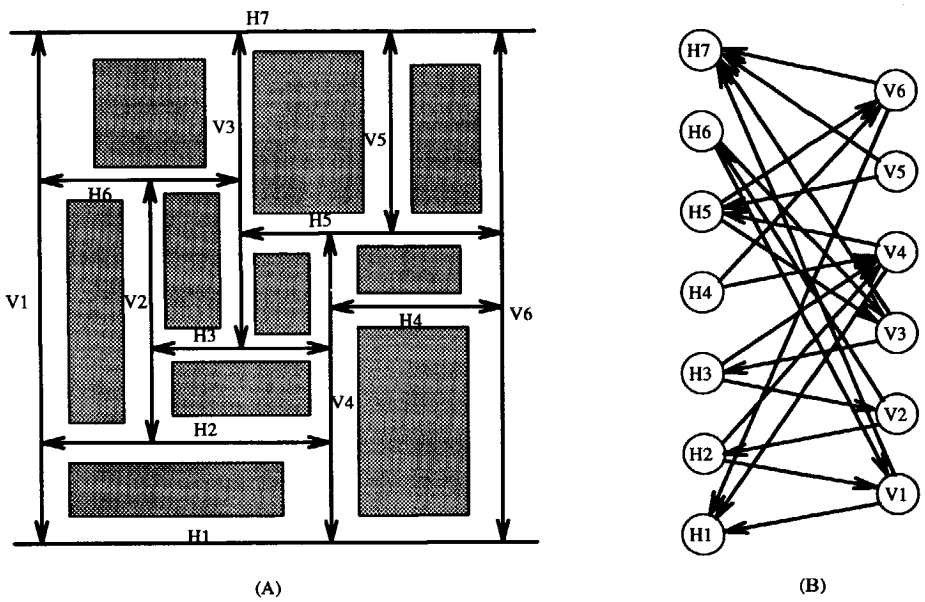

Figure 48: A placement with its rectangular dissection (a). The arrow heads indicate the channel ends. Figure b shows the corresponding channel digraph. The directed edges indicate the channel routing order constraints.

prior to the cross-bar channel at any 'T' type channel intersection.

To provide a framework for the routing areas, we can capture the placement of blocks by a *rectangular dissection.* It is constructed by dissecting the rectangular routing area by orthogonal lines such that each of the rectangles achieved in this way contains exactly one module. An example of a placement and its associated rectangular dissection is shown in figure 48a. Suppose the maximal line segments in the rectangular dissection represent channels. Then each (internal) channel is spanned between two 'T' intersections, in both of which it forms the base. As described in the previous paragraph, each 'T' intersection introduces a routing order constraint between the channels. The graph which captures all routing ordering constraints between the channels is called the *channel digraph* of the dissection [29]. Its vertices represent horizontal or vertical channels. A directed edge from channel i to channel j indicates that i must be routed prior to j (that is, i and j share a 'T' intersection with i as base). Figure 48b shows the channel digraph of the rectangular dissection in figure 48.

In the channel digraph a familiar problem arises: cyclic ordering constraints. As soon as the channel digraph contains a cycle, no feasible channel routing order can be determined. The example of figure 48, for example, contains the cycles V3→H3→V4→H5→V3 and V3→H3→V2→H6→V3. Figure 49a shows the classic configuration which leads to a cyclic ordering constraint with 4 edges. It can be proven that the channel digraph is acyclic if and only if it does not contain this 4-edge 'windmill' pattern [79].

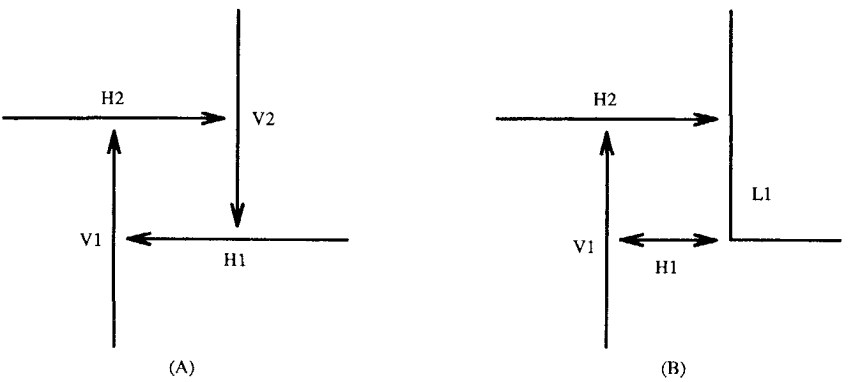

Figure 49: The 4-edge windmill *arrangement of channels which results in a cyclic routing order constraint (a). It is characteristic for any non-sliceable placement. The windmill cycle can be broken by an* L-shaped channel *(b).*

5.1.2 L-shaped channels

To break the cyclic vertical constraint in the channel routing problem, tricks like doglegging are successfully applied. A cyclic channel routing order constraint could be 'broken' by inserting one *'L'-shaped* channel in the cycle. Figure 49b illustrates topologically how the L-shaped channel breaks the cyclic ordering constraint. The channels can be routed in the order H1→V1→H2→L1. Figure 50 shows L-shaped channels in more detail. The top boundary of an L-shaped channel can be moved in two directions, instead of only the vertical direction in an ordinary horizontal channel. The solution to break the channel routing order constraints by inserting L-shaped channels may look quite attractive at first sight. Unfortunately, the L-shaped channel routing problem turns out to be a non-trivial generalization of the ordinary channel routing problem.

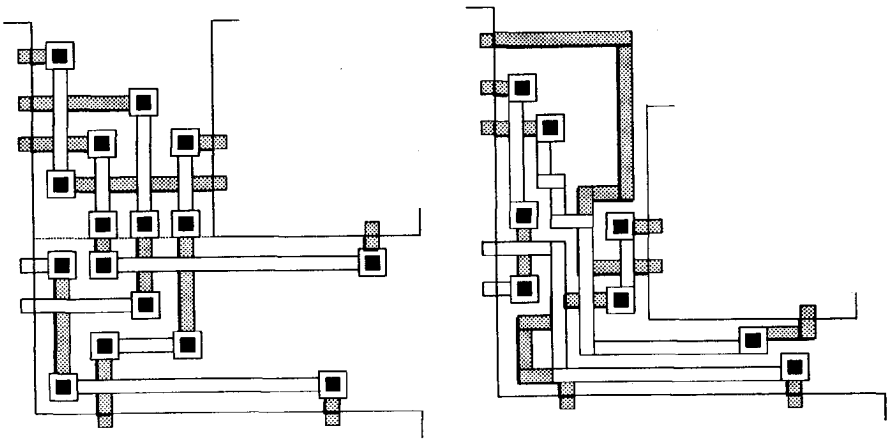

Figure 50: Two versions of the same L-shaped example channel. The figure on the left shows a straightforward grid-based approach, in which the channel is split into an 'ordinary' vertical channel and a 3-sided horizontal channel. The latter can be routed using the greedy channel router [73]. The dotted line indicates the boundary of the two channels. The figure on the right shows the same channel routed using a modified version of the gridless contour router (chapter 3). The wires bend around the corner.

A straightforward approach to the L-shaped channel routing problem is to split the channel into a horizontal and vertical sub-channel. The vertical channel is routed first by an ordinary channel router, while the horizontal channel is routed by a channel router which can handle fixed terminals on three sides. The column-sweep 'greedy' channel router [73] is suitable for this approach. Several iterations are required, because the height of the horizontal channel is fixed by the previously routed vertical channel. An example of an L-shaped channel which was routed using this strategy is depicted in figure 50a.

An alternative approach can be based on the contour-based router as presented in chapter 3. We can define a 'corner' which splits the channel into a horizontal and a vertical section. The contours and the wires fold around the corner (see the right of figure 50). The heights of both the horizontal and vertical sub-channels are fixed due to wires which are crossing to the opposite side of the channel. Again, several iterations are required to converge to the proper offset of the top part of the channel. Furthermore, the territories of the terminals in the corner must be secured to ensure 100% completion. This imposes a constraint on the placement of the terminals near the corner.

Another alternative was proposed by Chen [14], who suggests to convert the channel into a horizontal channel by breaking it in the corner. This forces all branches to be routed under a 45 degree angle. On its turn, this angle requires the terminals to be spaced an additional factor $\sqrt{2}$ further apart. Chen's L-shaped channel router is integrated in the BEAR general cell routing system [21, 64] of the UC Berkeley.

Despite all effort which has been devoted to L-shaped channels, no feasible algorithm has been published. The many constraints and the iteration loop make L-shaped channel routing very fragile. With that, most of the charms of a channel router are lost. In most real-life routing systems they are usually replaced by switch-box routers.

5.2 Slicing

Another way to deal with the cycles in the channel digraph is to prevent them beforehand. Rectangular dissections of which the channel digraph is acyclic are called *slicing structures*. Slicing structures were introduced by Otten [62] in 1980, mainly with the aim of floor plan design. They can be obtained by recursively dividing the placement into smaller *slices*. Each slice either contains a single module, or is itself dissected into (smaller) slices. The hierarchy implied by the recursion is captured in a *slicing tree*, in which each vertex represents a slice and of which the leaves are modules. Figure 51 shows an example of a placement and its associated slicing structure.

The subset of all placements which can be captured by a slicing structure is called *sliceable placements*. They can be recognized by the property that their rectangular dissection does not contain the windmill pattern of figure 49a. Besides providing a conflict-free channel framework, a slicing structure has a number of other interesting features. First of all, there is a practical aspect: it can be stored in recursive tree data structures which are easier to handle than more general representations. Secondly, many optimization problems for slicing structures can be solved efficiently, whereas the corresponding problems for arbitrary rectangular dissections are NP-hard.

Evidently, only a small portion of all possible placements is sliceable. The advantage of slicing structures, however, strongly outweigh this restriction. Only slicing structures can be routed entirely by channel routers since they capture a conflict-free channel definition. Therefore 100% completion can be achieved without iteration loops.

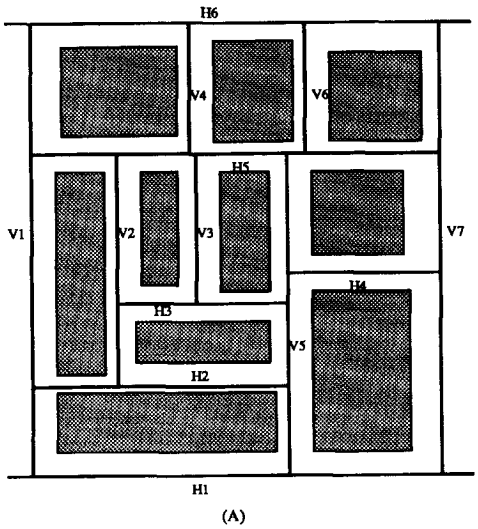

Figure 51: A sliceable placement with one of its slicing structures.

5.3 Channel definition for full-custom general cell layout

5.3.1 Topological channel definition

The rectangular dissection of a sliceable placement may contain a number of '+' type line intersections. To map the dissection on a slicing channel structure, each '+' type crossing must be split into two 'T' type intersections, either horizontally or vertically. All '+' crossings in the placement depicted in figure 51, for example, were split horizontally. In some cases splitting in the wrong direction may even result in a non-slicing structure. Due to the presence of '+' type crossings, a number of different slicing structures can be defined for a sliceable placement. Selecting a better channel structure has usually more impact on the overall circuit size than using a better channel router. The additional freedom provided by the '+' type crossings can be exploited to select a proper slicing channel structure. Hong Cai [10, 7] defines *bonuses* for splitting '+' type crossings into a particular direction. The bonus of a crossing is comprised of a number of heuristic parameters, such as the 'flow' of the wires through the channels and whether the resulting channel belongs to a 'critical path' of the circuit. The critical path the contains those channels of which the height

contributes to the overall circuit size. Cai presents a polynomial time complexity algorithm to derive the optimal conflict-free channel structure according to the bonus values.

Once such a conflict-free channel definition has been determined, the circuit can be assembled by routing the channels one at the time, according to the routing order. In chapter 7 we will present an entire routing system based on this approach.

5.3.2 Geometrical channel definition: terminals on vertical channel boundaries

The congestion of wires is most critical at channel junctions, where many wires are packed tightly together. In a general-cell circuit the length critical paths is generally determined by the wires at a limited number channel junctions. The criticality of these junctions can be reduced by placing terminals on the vertical edges of the bottom or top channel boundary. In this way, the intensity of the 'flow' of wires will be spread out over a larger area. This will, on its turn, reduce the congestion. Figure 52 illustrates this concept.

A basic requirement is, of course, that the channel router allows terminals on the vertical edges of the (fixed) bottom and top boundaries. Until now, placing vertical terminals on fixed boundaries has been the exclusive territory of (grid-based) switch-box routers. Approaches with switch-box routing areas at channel junctions have been published (e.g. [18]). Within limits it is also possible to use the (variable wire width) channel router of chapter 3 for this purpose. We implemented a pre-processor which converts the 'vertical terminals' into horizontal ones with non-overlapping territories. A wiring pattern similar to the fan-out pattern as discussed in section 3.7.1 is used. The result of the process is illustrated in figure 52.

Obviously, the terminal capacity on the vertical boundary is limited. It is not even possible to guarantee 100% completion, since the conversion from vertical into horizontal terminals could fail. The terminal capacity can, however, be estimated accurately before the channel is created. The distance to the nearest obstacle on the channel boundary limits the number of vertical terminals on a channel boundary.

5.4 Channel definition for semi-custom general cell layout

The row-based design style is the dominating design style for semi-custom chips. Also for sea-of-sates chips (which evolved from gate arrays) many layout systems apply that

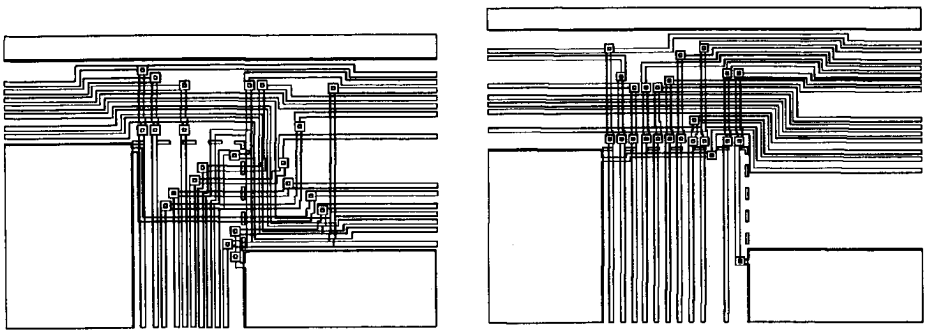

Figure 52: Two versions of the same 'T' type channel intersection in a real-life chip. The 'vertical terminals' (in the version shown on the left) enable a better area utilization which results in a considerable decrease of channel height. The channel boundaries are marked by the dashed line.

style. In this section we will describe how the channel router of chapter 4 can be used to introduce a general-cell design style on sea-of-gates.

Again, we will require that the general cell placement is sliceable, such that a conflict-free channel definition can be determined. With the channel router, this should guarantee 100% completion for the entire circuit. The basic idea is to make the slicing channel structure coincide with tear lines in the image. The wires are routed along the channels. Because the channel router handles obstacles, the wires may also traverse through the modules. The amount by which the channels overlap with the modules can be adjusted. Only if the routing capacity over the channels is insufficient, additional wiring space is created by 'tearing'.

As a first step, we define for each layer the direction of preference for the entire chip. Usually a pre-defined power wire layout forces a scheme of alternating horizontal and vertical layers.

5.4.1 Overlapping routing areas

In full-custom channel routing the entire routing area is subdivided into channels. No overlap of the channels is allowed because the channel router requires an obstacle-free routing area. The channel router we presented in chapter 4, however, does allow obstacles in the routing area. These channels may overlap because the wire pattern of one channel can be regarded as an obstacle in the other. This opens a number of interesting opportunities.

As described in the previous chapter, more than one tear line may be present in

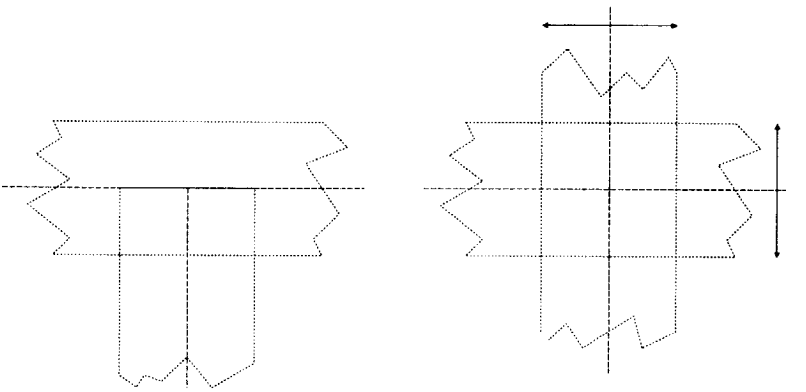

Figure 53: Overlapping channels at 'T'- and '+' type intersections. The channel routing order constraint at a '+'-type crossing is removed. The height of each routing region (and with that, of the overlap area) may vary.

the channel. Certain edges in the grid graph have been removed such that the wire generator will never generate a wire crossing a horizontal tear line in a horizontal layer and vice versa. The perpendicular tear lines at a '+'-type channel intersection (see figure 53) can therefore co-exist in a channel. As a result, there is no channel routing order constraint emerging from '+'-type crossings in the rectangular dissection. This removes a number of edges in the channel digraph. Also, the 'T' type intersections with the 4 outer channels can be treated as '+' type intersections, removing the routing order constraint. In the case of a regular 'chess-board' placement of modules no channel routing ordering constraints are present at all. In this way it is possible to route the nets one at a time over the entire circuit.

5.4.2 Routing across the interface of two overlapping channels

The global path of a net (as determined by the global router) may traverse through a number of channels. If the routing areas do not overlap, the connection between the regions is performed by *junction terminals* which are placed on the boundary interface between two channels. Once the position of the junction terminal is fixed, the path of the wire is forced to touch it. The terminal position is determined by the channel which is routed first, without taking much care about the situation in the neighboring channel.

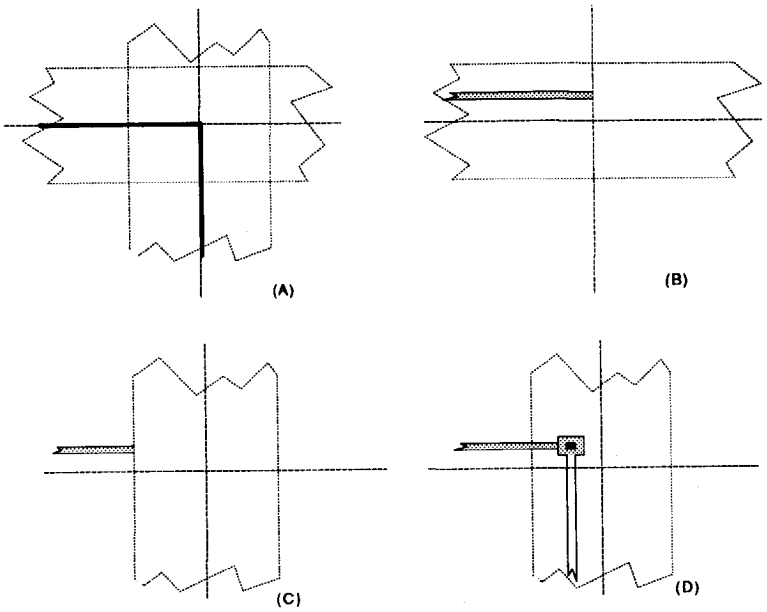

Figure 54: Routing across regions with overlapping channels (see text).

In contrast to non-overlapping channels, connecting wires between overlapping channels can be performed in a less rigid way. Consider the elementary region junction in figure 54a. The global route of the wire crosses from the left of the horizontal channel to the bottom of the vertical channel. In this example we will route the horizontal channel first (the order in which they are routed is irrelevant, though). The left side of the horizontal wire is a floating (junction) terminal in a vertical layer, aligned with the vertical tear line. Therefore the routing of the horizontal channel results in a wire which ends at an arbitrary position on the vertical tear line (figure 54b). The routing of this net can never fail because junction terminal is floating and placed in a vertical layer. In the next step, the vertical channel is routed. The junction terminal, in this case, is a fixed terminal on the (fixed) left side of the channel. An interesting property is that the part of the horizontal wire which extends in the vertical channel can be removed (figure 54c). The old path only serves as territory to ensure routability. The routing in the vertical channel creates an entirely new wire, which is adapted to the local wiring situation (figure 54d). With wider channels, the overlap area is larger, which enables the wires to 'cut corners'

through the modules to reduce the wire length. This is a major advantage over conventional approaches with non-overlapping routing regions and rigid, one-sided junction terminal placements.

This method to route across perpendicular channels can be used as a 'cornerstone' for a general-cell routing system for sea-of-gates layout. With a sliceable placement with a conflict-free channel structure the detailed routing can be performed in a similar manner as a full-custom general-cell routing system.

Chapter 6

Global wire ordering

6.1 Wire twisting

As stated in chapter 1, a 'divide and conquer' strategy is employed in an attempt to tackle the scale of a large VLSI circuit. The routing area is subdivided in a number of smaller channels. At the interface between two adjacent channels, terminals must be placed to connect nets of which the path crosses the region boundary. We call these additional terminals *junction terminals*. Notice that the junction terminals are a result of the divide and conquer strategy. Without breaking up the circuit into smaller routing areas these terminals are not required. With this subdivision, their position and ordering interferes with the routing in two independent regions.

The placement of the junction terminals is usually performed in a 'bottom-up' manner by the detailed router. This method is commonly employed if a channel router is used for detailed routing. A channel router does not allow a fixed (junction) terminal placement at the 'left' and 'right' ends of the routing region. Instead, the channel router itself determines the terminal placement and ordering of these floating terminals. This usually results in twisted wires and wasted space, as figure 55 shows. In this example, twisted wires occur because the vertical channels were routed prior to the horizontal ones. Within each vertical channel the terminal ordering (and with that, the wire ordering) is optimal. The internal ordering within a single region, however, is not influenced by the ordering of the wires in other regions. Therefore the wires may change relative positions in each region, which makes their paths through the regions nothing less than capricious. A 'synchronized' junction terminal ordering which is dependent on the global context of the channels (as in figure 56) could have prevented unnecessary twisting of the wires.

Our goal is to determine a 'top-down' ordering for the junction terminals such that unnecessary wire twisting is prevented and unavoidable wire crossovers (we call

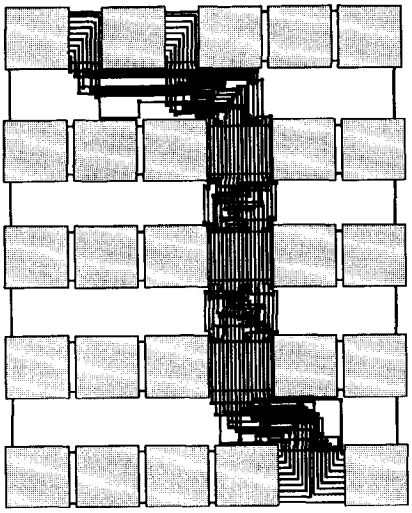

Figure 55: An example circuit. The twisted wires occur because the regions are routed independently.

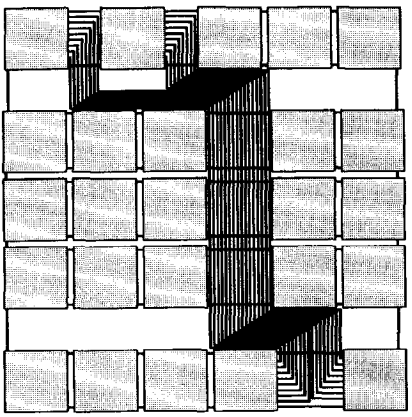

Figure 56: The same circuit as in figure 55, but routed with a synchronized wire ordering.

them *crossings*) are cleverly assigned to specific routing regions. This strategy has the following benefits:

- The regions should be easier to route because the terminals are in a structured order. The wiring area will decrease and, with that, most likely the overall chip area. Switch-box routers and maze routers have a better chance to achieve 100% completion within the given area.
- Reducing the number of twisted wires leads to better electrical characteristics of the wiring pattern. Fewer twisted wires means fewer vias and less capricious wire paths, which reduces the the resistance and capacitance of the wires.
- Busses will be routed in a consistent order since they are made up of an array of wires which share the same topology.

Rivest was one of the first to incorporate a terminal placement that takes into

account the global path of each net. In his 'PI' routing system [72], he uses an iterative force-directed algorithm to place the junction terminals. In a nutshell, his algorithm works as follows.

First, all junction terminals are placed in the center of their region boundary. Then the junction terminals are processed one after another. For each junction terminal a new position is calculated which is the mean of its current position and the positions of the neighboring terminals to which it is connected. To differentiate the various situations, terminals which are further away from the junction terminal are weighted less, while a terminal adjacent to the junction terminal obtains a higher weight. In this way a new position is found on the region boundary, which is rounded to the nearest unoccupied grid position on the region boundary. The process is repeated a number of times for each junction terminal until an equilibrium configuration is achieved. The iteration also enables the positions of remote terminals to propagate over the entire length of the net. Notice that the final result is an exact geometrical pin positioning.

For his symbolic router [57], Ng proposes a technique which he called *bouyancy calculation* to prevent twisted wires within a single region. The disadvantage of Ng's as well as Rivest's approach is that neither guarantees the terminal placement will yield the minimum number of wire twists. They are both heuristic approaches.

In this chapter we derive a condition under which the junction terminals can be placed such that there is a minimum number of wire twists. Based on this condition we present a non-iterative algorithm which, given this constraint, assigns the wire crossovers to specific routing regions. Unlike previous approaches, our algorithm does not use exact geometrical positions. Instead, a junction terminal ordering is based on a topological representation of the circuit. In this way, we can order the junction terminals before the detailed placement of the blocks is performed.

6.2 Problem formulation

Given is a wiring area and a set of modules placed within this area. Terminals ($T_{\alpha,\beta}$, in which α specifies a net and β a terminal) are present on the boundaries of the modules and the wiring area. A set of nets η is given to specify the connectivity over the these terminals:

$$\forall i \; N_i = \{T_{i,\beta}\}, \; \beta = 1, 2$$

Also, at this point, we assume that all nets have two terminals. In section 6.6 we will show how to deal with multi-terminal nets. To capture the topological paths of the nets together with the positions of the terminals and the modules, we introduce a *placement graph*, denoted by $G(V, E)$. It is a plain graph which is similar to the

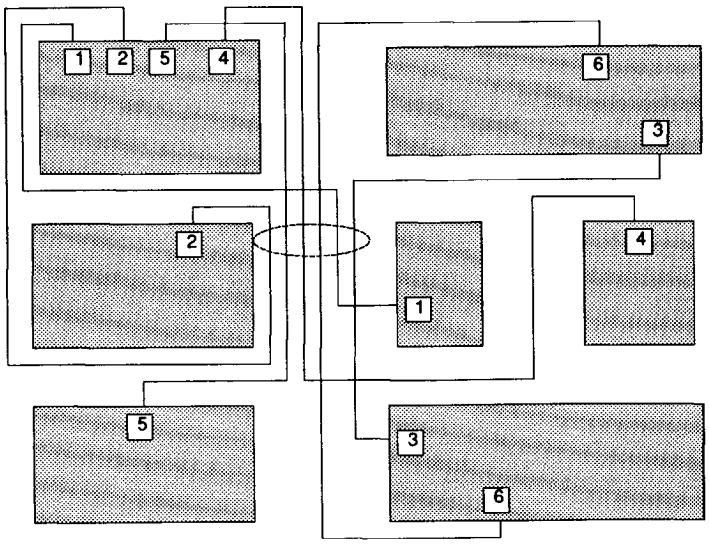

Figure 57: Sample circuit.

floor plan graph used as framework by global routers. The placement graph contains three types of vertices:

1. A *terminal vertex* represents a terminal of a net. There is a one-to-one correspondence between each terminal $T_{\alpha,\beta}$ and a terminal vertex.

2. A *region vertex* corresponds to a wiring region between two adjacent modules.

3. A *junction vertex* marks the adjacencies between two or more region vertices.

The placement graph contains edges between a terminal vertex and the region vertex which corresponds to the adjacent wiring area. This edge links the terminal to the wiring region, preserving the relative ordering of the terminals. The graph also contains edges between junction vertices and their adjacent region vertices. For example, figure 58 shows the placement graph for the circuit in figure 57. We will use this circuit throughout this chapter.

The global router has determined the path of each net. Within our framework the topological path P_i of net N_i is given as a connected subgraph of G:

$$P_i \subset G$$

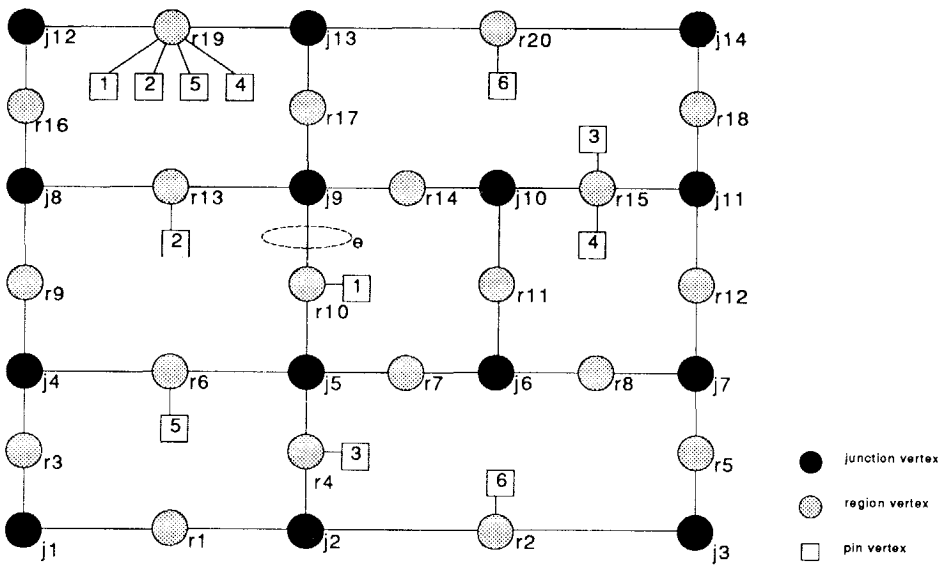

Figure 58: The placement graph $G(V, E)$ of the example circuit in figure 57.

Each path starts and ends at a terminal and is a sequence of consecutive edges and vertices of G. For example, the path of net 3 in figure 57 contains the vertices {t3, r15, j10, r14, j9, r10, j5, r4 ,t3} and all intermediate edges. We can assume that no cycles are present in the path, so each vertex or edge of G occurs only once in P_i.

The interface between two adjacent routing regions corresponds to an edge between a region vertex and a junction vertex in $G(V, E)$. We call such an edge a *junction edge*. Consequently, the order of the junction terminals at a region boundary corresponds to the order of the wires in a junction edge. The problem is now to determine the wire ordering in all junction edges such that:

1. The number of wire crossings is minimized.

2. If wires must cross, the position of that intersection is placed such that the intersection causes the least amount of increase in the size of the circuit.

The first objective is the most important one. It can be seen as keeping the wiring as planar as possible: wires which do not have to cross should not twist. Wires which have to cross should do that only once.

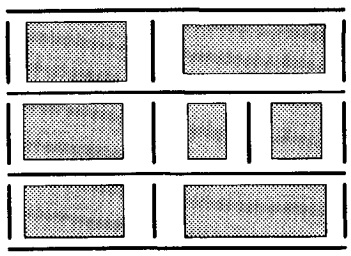

Figure 59: A possible slicing channel structure as region definition for the channels in figure 57. The channels are marked by the dark lines.

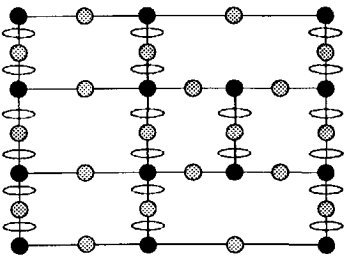

Figure 60: The region boundaries of the channels in figure 59 correspond to the circled junction edges *in the global routing graph.*

INPUT:
The placement graph $G(V, E)$ and the path of each net.

OUTPUT:
An ordering for the junction terminals at each junction edge.

METHOD:

1. Select an arbitrary (but still undetermined) junction edge e.
2. Determine a feasible ordering for the junction terminals.
3. Split all nets in e into 2 two-terminal subnets by removing the edge e from their path and inserting a junction terminal in each of them. The junction terminals are added to the placement graph G in the order just determined.
4. If there are still undetermined junction edges: go to step 1.
5. End.

Figure 61: The basic algorithm to order the junction terminals.

Our basic approach is to determine wire orderings for the junction edges one after another using the algorithm in figure 61. Notice that we process the junction edges in arbitrary order. Figure 62 illustrates the effect of step 3. For each of the 6 nets

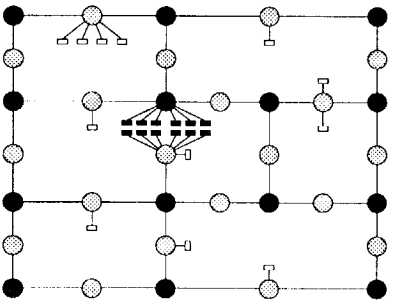

Figure 62: The graph $G(V, E)$ of figure 58 after ordering of the circled edge e.

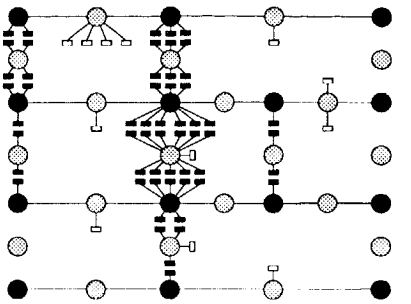

Figure 63: The placement graph after all junction edges have been processed.

which cross the edge e, a pair of terminal vertices was added in the specified ordering to $G(V, E)$. Finally, after the ordering of each junction edge has been determined, each routing region will be a connected 'island' in the graph G (figure 63). The key issue of the algorithm is in step 2. In the following section we will derive a necessary and sufficient condition for the ordering to result in the minimum number of crossings. A heuristic way to place the crossings under this condition will be described in section 6.5.

6.3 A condition for the minimum number of crossings

Let e be an arbitrary junction edge between a junction vertex and a region vertex in $G(V, E)$. We begin by deriving two trees, each representing a planar realization of the wires on one side of e. We define the set η_e to contain those nets which are dissected by the region boundary represented by e ($\eta_e \subseteq \eta$). Since no cycles are present, the path of a net in η_e can be decomposed into three disjoint subsets: the edge e itself, a part on one side of e (marked P_n^+) and a part on the other side of e (P_n^-):

$$P_n = P_n^+ \cup \{e\} \cup P_n^-$$

On both sides of e a plain tree can be constructed which represents the *fan-out* pattern of the bundle of the wires in e. The tree T_e^+ is constructed by merging the paths P_n^+ of all nets in η_e (see figure 64). The root of the tree is the vertex on the '+' side of

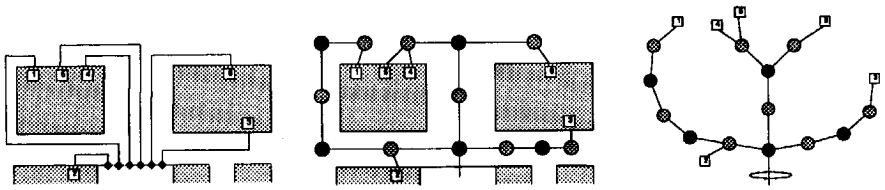

Figure 64: Illustration of the construction of the fan-out tree T_e^+ in the example circuit. The tree captures the bundle of wires on the top side of the region boundary e (the circled edge in figure 57 on page 86).

the edge e and the leaves are the terminals. Starting from the edge e, two paths share vertices and edges until they separate into different edges. Since the tree is plain, the relative ordering of the edges in the graph can be preserved in the relative ordering of the branches of the tree. If the paths of two nets separate first and then join again at another vertex in G (the *reconvergent nets*) the latter vertex will occur twice in the tree. For instance, the vertex `r19` appears twice in the tree T_e^+ (see figure 65). The tree T_e^- is derived in a similar way.

The two trees capture the 'planarity relation' between the wires, as seen from the edge e. By traversing pre- or post-order through a tree, each time marking the leaves, the 'planar' ordering for the junction terminals is obtained for this tree. In the tree T_e^+ of the example (figure 65), for instance, traversing clockwise along the leaves results in the net ordering $2 \rightarrow 1 \rightarrow 4 \rightarrow 5 \rightarrow 6 \rightarrow 3$. We call this order *planar* because it is the only configuration of the junction terminals for which the bundle of wires on the top side of e can be routed without any wire crossings. In figure 64 the wires are drawn in the planar ordering of T_e^+.

We now introduce the graph C_e to capture the planar orderings of both trees. It contains, on one side, the planar ordering of the junction terminals in T_e^+ and, on the other side, the planar ordering of the junction terminals in T_e^-. In C_e there is an edge between the terminals of each net. In general, C_e is a non-planar graph which means that wires have to cross. Figure 66 shows the graph C_e for the example.

As next step we define the *planarity* $p_{i,j}$ of two nets $N_i \in \eta_e$ and $N_j \in \eta_e$ as:

$$p_{i,j} = \begin{cases} 1 & \text{if nets } N_i \text{ and } N_j \text{ have to cross} \\ 0 & \text{if nets } N_i \text{ and } N_j \text{ are planar or } i = j \end{cases}$$

The planarity values can be derived easily from C_e. In the example (figure 66), for instance, nets 4 and 5 have to cross while nets 5 and 6 are planar. The minimum number of crossings of the wires in η_e can be calculated by adding the planarity values

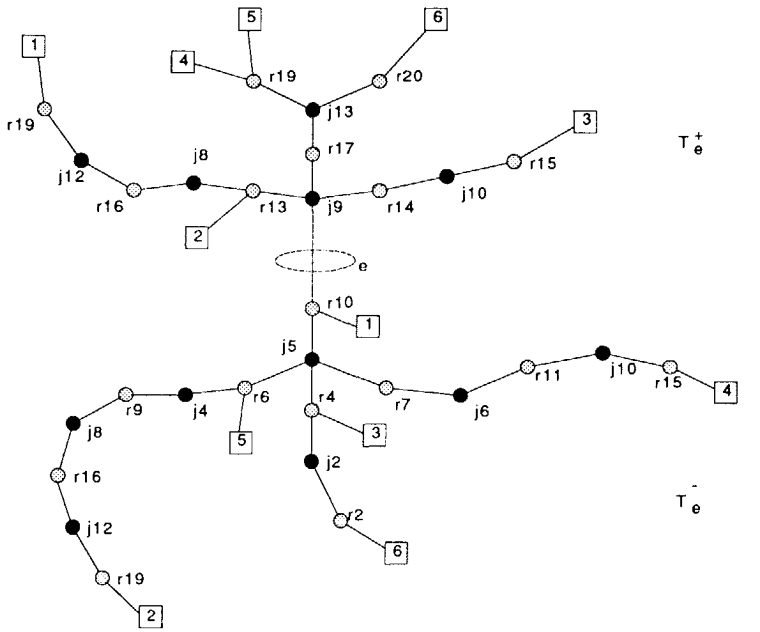

Figure 65: The two fan-out trees T_e^+ and T_e^- of the circled junction edge in figure 57.

of all pairs of nets in η_e:

$$cr_e = \frac{1}{2} \sum_{N_i \in \eta_e} \sum_{N_j \in \eta_e} p_{i,j} \tag{3}$$

We need the factor $\frac{1}{2}$ because each crossing is counted twice.

We now perform MITOSIS[1] on the graph (or *cell*) C_e, splitting it into two new graphs: C_e^+ on top of the edge e and C_e^- below it (see figure 67). Each of the new cells contains all nets (the *chromosomes*) of η_e. A set of junction terminals connects the cells at their mutual boundary. We are now ready to make the following statement about the number of crossings in the new cells:

Theorem 3 *A two-cell system $\{C_e^+, C_e^-\}$ which resulted from mitosis of the cell C_e has the minimum number of crossings if, and only if, the following condition holds:*

$$\forall N_i, N_j \in \eta_e : \; p_{i,j}^+ + p_{i,j}^- = p_{i,j} \tag{4}$$

[1] Mitosis is the elementary biological process of cell division.

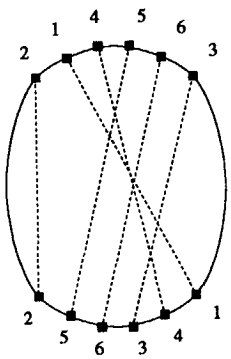

Figure 66: (left) The graph C_e which can be derived from the trees in figure 65. It captures the relative ordering of the leaves in the plain trees T_e^+ and T_e^-.

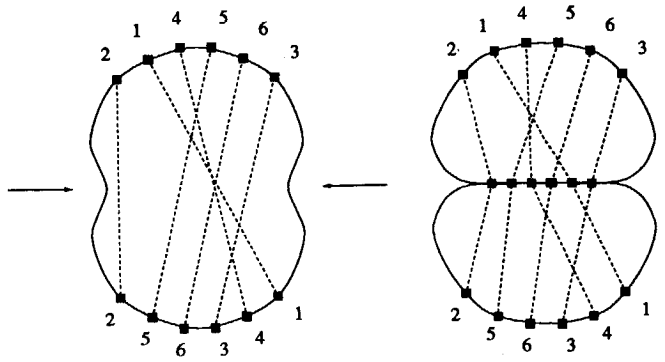

Figure 67: (middle, right) The process of MITOSIS. *The graph splits into two new subgraphs at the boundary representing e.*

in which $p_{i,j}$ is the planarity of nets N_i and N_j in C_e and $p_{i,j}^{+(-)}$ is the planarity of those nets in $C_e^{+(-)}$.

Proof:
For any pair of two nets $N_i, N_j \in \eta_e$ in the cell C_e two cases can be distinguished:

1. $p_{i,j} = 1$
   If nets N_i and N_j have to cross the ordering of the junction terminals will always match a planar ordering in one of the sub-cells. Consequently the two wires will cross in the other sub-cell. Therefore:

   $$p_{i,j} = 1 \quad \Rightarrow \quad p_{i,j}^+ + p_{i,j}^- = 1 = p_{i,j}$$

   Hence the relative ordering of the junction terminals of the two nets cannot introduce additional crossings in this case.

2. $p_{i,j} = 0$
   If nets N_i and N_j are planar, one ordering for the junction terminals can be chosen such that they are also planar in the two sub-cells. The other ordering introduces a wire crossing in each of the sub-cells. Therefore:

   $$p_{i,j} = 0 \quad \Rightarrow \quad \begin{cases} p_{i,j}^+ + p_{i,j}^- = 0 = p_{i,j} \\ \textbf{or} \\ p_{i,j}^+ + p_{i,j}^- = 2 \neq p_{i,j} \end{cases}$$

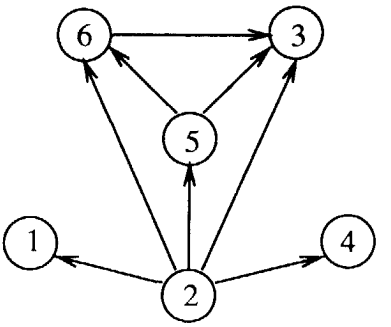

Figure 68: The planar ordering constraint graph O_e of the graph in figure 66, using a left-to-right convention. Net 2, for example, must be placed left of all other nets.

The total number of wire crossings of the two-cell system is the sum of the crossings in each sub-cell. The lower bound for this parameter is the total number of crossings in C_e:

$$cr_e^+ + cr_e^- \geq cr_e$$

As given by (3), cr_e is the sum of the mutual wire crossings of every pair of nets. Therefore the lower bound can only be achieved if (4) holds. □

Corollary 3 *To obtain the minimum number of wire crossings in the 2-cell system $\{C_e^+, C_e^-\}$ the following condition is necessary and sufficient:*

$$\forall N_i, N_j \in \eta_e : \text{ if } p_{i,j} = 0 \text{ then } p_{i,j}^+ = 0 \text{ and } p_{i,j}^- = 0 \qquad (5)$$

In other words: the minimum number of crossings is obtained if all nets which are planar in C_e are also planar in C_e^+ and C_e^-. We can capture the ordering constraints for the junction terminals implied by (5) in a directed *planar ordering constraint graph* O_e. Each vertex of this graph represents a net. A directed edge indicates an ordering constraint between the nets. Figure 68 shows the graph O_e of the example. In this figure a directed edge from net N_i to net N_j indicates that the junction terminal of N_i must be placed left of the junction terminal of net N_j. We can state the following about this constraint graph:

Lemma 2 *The planar ordering constraint graph O_e implied by (5) is acyclic.*

proof:
Consider an arbitrary net N_i of O_e. In the graph C_e the edge between the terminals

splits this graph into two parts, say a left and a right part. Directed constraint edges to net N_i can only originate from nets which have both terminals in the left part. Suppose a cycle is present. Then for some net N_i there must exist an edge representing a constraint from a net above N_i to net N_i itself. This contradicts the previous statement. □

Corollary 4 *It is always possible to find a junction terminal ordering which obeys (5).*

Mitosis can be performed on each of the junction edges (that is, region boundaries) in the circuit. As long as (5) holds at each fission, no wire twisting can be introduced. The fission corresponds to step 2 of the basic algorithm on page 88. With each fission the entire ordering relations of all nets crossing an edge e are traced down to the leave terminals. This allows us to process the junction edges in arbitrary order while ensuring the minimum number of wire crossings.

6.4 Complexity

If a circuit contains m modules there are $O(m)$ vertices in G. In worst case each of the $|\eta|$ nets spans all region- and junction-vertices of G. Therefore the upper bound for the size (and construction time) of a fan-out tree is $O(|\eta| \times m)$. Since there are at most $O(m)$ floating edges, the order of the algorithm is $O(|\eta| \times m^2)$. The speed of the algorithm can be improved by processing the junction edges in such an order that the graph G is partitioned into disjoint subgraphs.

6.5 Crossing placement

In the previous sections we derived that preventing wire twist of planar nets at each junction is sufficient to obtain the minimum number of wire crossings. Within the limits implied by this constraint usually many different orderings can be chosen for the junction terminals at step 2 of the algorithm (figure 61). Basically we can observe that if two nets have to cross, their intersection will be either on the left or on the right of the edge e, depending on the ordering of their junction terminals. In other words: the ordering of the junction terminals pushes each wire crossing implicitly to one side of the junction. Hence it is possible to assign the crossings to specific routing areas. This *crossing placement* should be such that the chip area and routability will be least affected. The criteria to place the crossings are usually of a very heuristic

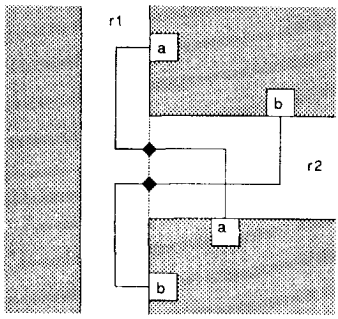

Figure 69: Example intersection. In this case there are two ways to order the junction terminals. This one will generally be preferred.

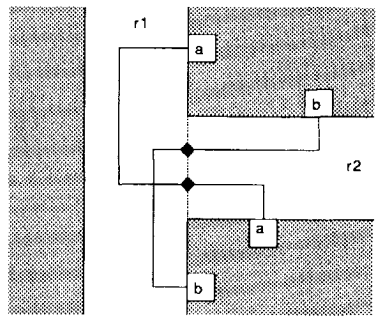

Figure 70: By ordering the junction terminals in this way the density in the vertical will be increased.

nature. At this stage (that is, before the actual detailed routing) only predictions can be made of the effect of a certain crossing placement on the size and routability of the circuit.

In this section we propose a way to determine the crossing placement using the fan-out trees T_e^+ and T_e^-. A number of weighted ordering relations can be extracted from each junction or region vertex in the trees. The relations express the 'separation weights' between wires in the edges (= wires) emanating from the vertex. Our aim is to 'balance' the ordering weights of the nets on each side of the edge e. The crossing of a pair of non-planar nets will be pushed to the side with the lowest separation weight. We will represent the separation weights using arrows. The line $a \rightarrow b$, for instance, indicates that net N_b must be placed below (or to the right of) net N_a.

The path of any pair of wires branches into different edges at specific vertices in the trees T_e^+ and T_e^-. There is a number of ways for wires to split up into edges. Currently we distinguish one special case which will be weighted more than other branches of wires into different edges. Consider the two non-planar nets in figures 69 and 70. Of the two possible orderings $b \rightarrow a$ (figure 70) and $a \rightarrow b$ (figure 69) the latter is preferable since it results in the smallest density for both regions. Selecting the ordering $b \rightarrow a$ increases the wiring density in the vertical channel `r1` and, with that, possibly the total circuit area. We use a double arrow ($\rightarrow\!\!\rightarrow$) to symbolize the additional separation weight of this branch in the tree (see figure 71). Table 5 shows ordering weights for the example circuit if the two types of separation weights are distinguished. Obviously it is possible to distinguish other wire separation criteria.

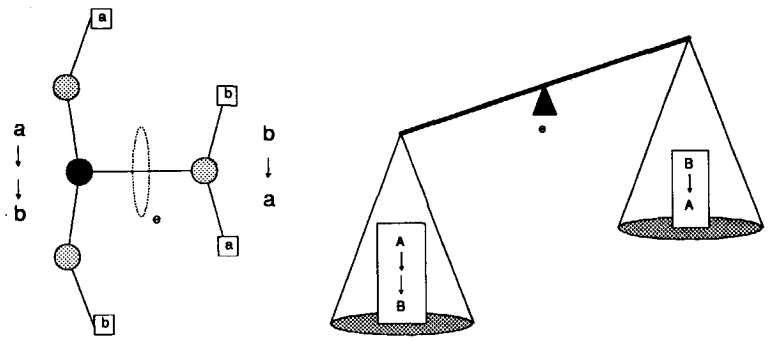

Figure 71: The representation of the intersection of figures 69 and 70 in a fanout tree. Selection of the preferred ordering is achieved by using different separation weights vertices in the fan-out tree. The arrows symbolize the separation weights.

Tree T_e^+		Tree T_e^-	
vertex	weights	vertex	weights
j9	(1, 2) → (4, 5, 6)	r10	(2, 3, 4, 5, 6) → 1
j9	(4, 5, 6) → 3	j5	(3, 6) → 4
j9	(1, 2) →→ 3	j5	(5, 2) → (3, 6)
r13	2 → 1	j5	(5, 2) →→ 4
j13	(4, 5) →→ 6	r4	6 → 3
r19	4 → 5	r6	2 → 5
Planar ordering constraints (see figure 68)			
2 ⟶ (1, 3, 4, 5, 6)		5 ⟶ (3, 6)	6 ⟶ 3

Table 5: Ordering weights in the example circuit using a left to right convention for both trees.

The total weight on each net can be expressed in a $|\eta_e| \times |\eta_e|$ matrix $A_{|\eta_e|}$. Each element a_{ij} in the matrix is the total ordering 'weight' which N_j imposes on N_i (assuming a certain convention, in our case left to right). In the example we weighted a single arrow 1 and a double arrow 2. If a planar ordering constraint edge exists from N_j to N_i the value of a_{ij} is set to P, which is a number. For the example this results in the matrix $A_{|\eta_e|}$ as shown in table 6.

A greedy algorithm can be used to obtain a feasible and weighted ordering from this matrix. Each time the net of which the column has the smallest weight is selected

net	1	2	3	4	5	6
1	-		2	1	1	1
2	P	-	P	P	P	P
3	1		-	1		
4	1		1	-	1	2
5	1		P	2	-	P
6	1		P	1	1	-

Table 6: The matrix $A_{|\eta_e|}$, which expresses the separation weight which the nets impose on each other. Essentially it contains the same information as table 5.

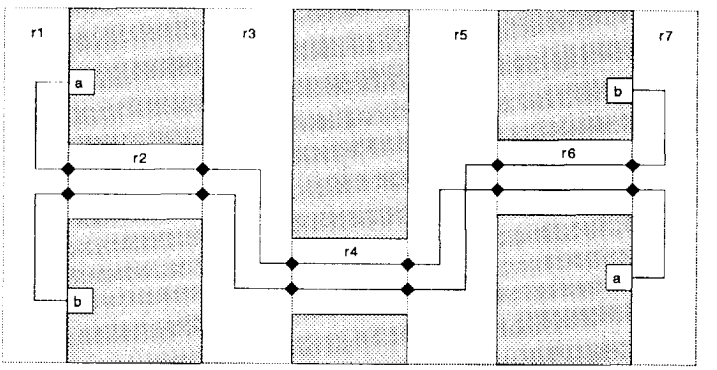

Figure 72: An example in which the separation weights are equal (see text).

from A. By making the planar ordering constraint weight P a very big number the final ordering will automatically fulfill (5). As stated by corollary 4, it is always possible to select a net which has no planar ordering constraint. A new $(|\eta_e| - 1) \times (|\eta_e| - 1)$ matrix $A_{|\eta_e|-1}$ can now be constructed by removing the row and column of the net just selected. This process is continued until the order of all nets has been determined. Each time the selected net is removed from the matrix. In our example the greedy algorithm will result in the wire ordering drawn in figure 57 (page 86).

It is also possible to take more detailed wire separation criteria into account. Consider the situation depicted in figure 72. The wire crossing will have to be placed in one of the seven routing regions. Obviously a crossing in regions r1 or r7 should be avoided. If the weight factors described until now were used, it is possible that such a configuration is found. This is caused by the fact that the separations in those

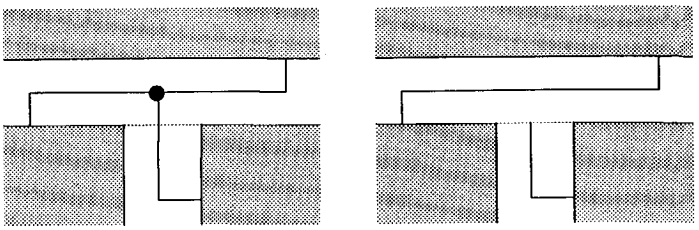

Figure 73: Illustration of a situation in which a multi-terminal net can be split at a 'T'-type junction in its path.

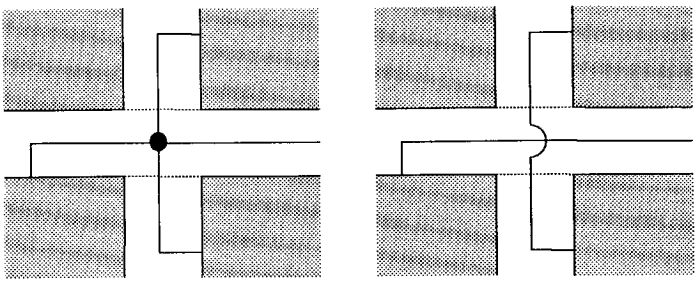

Figure 74: Splitting multi-terminal nets at '+'-type crossings into two ordering-disjoint nets.

regions are weighted equally when one of the corresponding junction edges is ordered first. This situation can be avoided by weighting remote separations less, depending on their depth in the fanout tree. In this way the crossing will automatically be pushed 'inwards'. This solution in drawn in figure 72.

6.6 Multi-terminal nets

Until now we only considered two-terminal nets. Multi-terminal nets must be decomposed into a number of two-terminal nets without seriously affecting the main objectives of the algorithm. We propose a decomposition process which consists of two steps. First the nets are split at each 'T' and '+' -type intersection of their path. Consider the two situations in figures 73 and 74. In both cases the order in which the vertical wire segment will be routed is independent of the order of the horizontal wire segment. For ordering purposes we will model these nets by two independent 'ordering-disjoint' nets. As a second step we will consider the two ends of

CIRCUIT name	CIRCUIT no. of modules	CIRCUIT no. of nets	IMPROVEMENT routing area†	IMPROVEMENT no. of vias
AMI33	33	123	1.8%	-0.6%
XEROX1	10	203	3.7%	0.1%
XEROX2	10	203	1.3%	1.7%
sl1	17	116	3.6%	7.5%
sl2	17	116	0.6%	4.8%
v0	19	225	0.8%	-0.6%
ph	14	362	1.7%	5.3%
eurom	39	591	0.2%	-3.5%
tdplo1	13	74	0.9%	4.2%
tdplo2	13	74	2.9%	14.6%
tdplo3	13	74	1.3%	0.4%

Table 7: Results for a number of real-world circuits. The column marked †shows the gain in routing area which was measured. AMI33 *and* XEROX *are* MCNC *benchmark circuits.*

the wire segment to be the terminals of the net, ignoring all terminals in between. It is possible to refine the second step by distinguishing several additional topologies of multi-terminal nets.

6.7 Some experimental results

We implemented the algorithm in our general-cell placement and routing system (see chapter 7). This system uses the contour based gridless channel router as presented in chapter 3 to perform detailed routing in the regions. We had to modify this channel router such that it accepts pre-defined terminal orderings. A fixed junction terminal ordering turned out to interfere heavily with the internal optimization of the channel router. The constructive channel routing algorithm relies on wire (and with that terminal-) ordering. Certain channel router specific problems (such as cyclic vertical constraints) are also solved indirectly by wire ordering. Therefore channels routed with a pre-specified terminal ordering are generally wider and contain more vias the constructive channel router of chapter 3. Notice that this negative effect of terminal ordering does not occur with switch-box detailed routers because they require a pre-defined terminal ordering in any case.

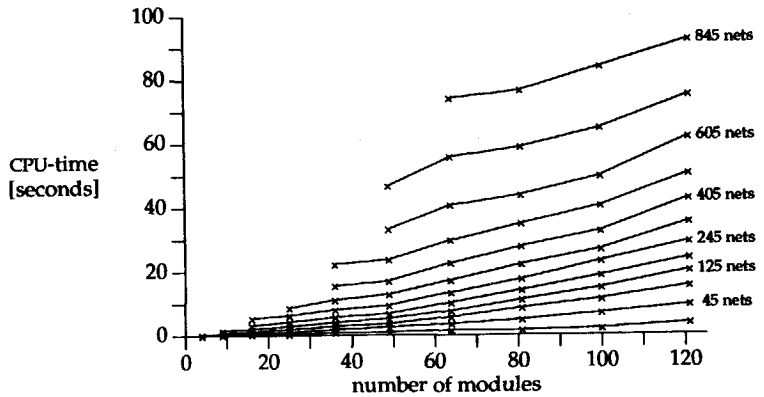

Figure 75: The number of modules versus the CPU*-time for wire ordering. The test circuits were chess-board like (such as in figures 76 and 77), with a variable number of random nets. The algorithm was run on an Apollo DN-3000 work station. The vertical axis shows the net ordering time.*

Table 7 shows the experimental results for a few 'real-world' circuits. The percentage shows the improvement between routing with and without global wire ordering. All other parameters were kept the same: placement, channel structure, global routing, channel router settings etc. We used the same separation weight parameters as discussed in section 6.5. For circuit size the global optimization outweighs the local optimization of the channel router in all tested cases. For the number of vias this is not always the case. This is probably caused by the fact that the detailed router sometimes looses track of the ordered terminal placement. Especially with wide minimum-spaced busses the channel router 'messes up'.

As could be expected, the CPU-time consumption of the algorithm is marginal. For our implementation it is at most 15% of the global routing time (that is, less than 1% of the total routing time). Figure 75 shows the results of CPU-time measurements on test circuits which were created by a random circuit generator. In these test circuits the modules are arranged in a chess-board like matrix with a variable number of random nets between them. Figures 76 and 77 show an example of a 4×4 test circuit.

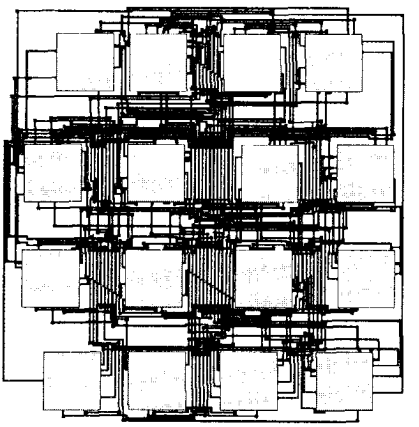

Figure 76: A chess-board like test circuit with random nets routed without wire ordering.

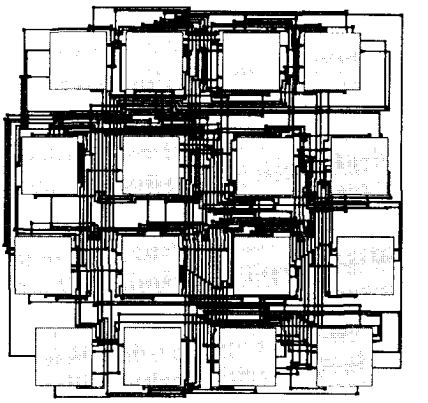

Figure 77: The same circuit as figure 76, but routed with the global wire ordering algorithm presented here.

Chapter 7

Concluding remarks

In this thesis we have described some aspects of the detailed routing problem in a more or less bottom-up order. The focus was on detailed routing and its wider context. In the sub-problems we have dealt with, it was assumed that a placement of the modules already existed. In this chapter we will first briefly review the relevant aspects of the earlier design steps. Most of the algorithms described in this thesis are embedded in a general-cell routing system. In section 7.2 it will be briefly described. Finally, the practical results which were achieved by this system will be evaluated in section 7.3.

7.1 Arranging the modules

7.1.1 Planning

The first step in a top-down layout design process is usually a *planning* phase, in which the design is partitioned into a number of modules. Preliminary estimations of the size of the modules are used to determine a *floor plan*. It is a rough geometrical arrangement of the modules, which also determines the desired shape of the modules. The idea is that the freedom of shape in the modules should be exploited such that a good 'fit' of the ensemble of blocks is achieved. After the introduction of the accompanying verb in 1979 [69], automatic *floor planning* has received much attention. Otten [61] introduced slicing structures into floor planning, which can serve as a computationally attractive framework for optimization algorithms. As discussed in chapter 5, the slicing structure is especially attractive since it captures a conflict-free channel structure for detailed routing.

With floor planning the geometrical problem of fitting the blocks is inherently considered more important than the wiring problem. Despite the fact that over 50%

of the chip area is consumed by wires most automatic floor planners treat the connectivity of the modules as a side-effect. By exploiting the structure in the connectivity and shaping the modules such that they can efficiently accommodate a specific wire structure a considerable gain can be expected. This approach, called *wire planning*, promises to be an interesting research topic for the coming years. One of the few papers on this subject [4] describes a wire planning approach for stack structures.

7.1.2 Automatic placement

If the layout of the modules is already given, we are dealing with the *placement* problem, where the exact locations and orientations of the modules must be found. A fundamental problem at this stage is that a number of parameters are not known beforehand. The wire length and the size of the routing area are parameters of which only a very rough estimate can be made because the actual routing has not yet been performed. Algorithmically the problem is worse still. Even if the connectivity between modules is ignored (the two dimensional pin-packing problem) the placement problem is NP-hard [30].

Based on various models, a number of placement algorithms have been published. *Min-cut* [49] is based on recursively bi-partitioning [28, 42] the set of modules such that the number of connections between the two sets is minimized while maintaining a certain area balance. This computationally attractive (i.e. fast) method generates a slicing structure by construction. A disadvantage of the slicing-tree based cutting is that, once a cut is made, further contact between the resulting slices is lost. Kleinhans et. al. [44] propose a variation on the min-cut method which also looks 'over the edge' of a previously made slice. Other methods, such as the constructive (e.g. [69, 41, 67]), analytical (e.g. [76]) and force-directed (such as [70, 89, 83]) model the modules as rectangular objects a gridless layout area. Therefore the resulting placement is not necessarily sliceable.

A method which is being applied frequently on the placement problem is *simulated annealing*. It is based on a simulation method in thermodynamics [54]. The basic idea is to assign a cost (or quality-) value to each configuration. For the placement problem the cost may depend on the size of the placement, the overlap of the modules and the estimated wire length. At each step the placement is changed at random by some operation, such as a swap of two modules. The change is called a perturbation. A perturbation which improves the quality of the placement is always accepted while a perturbation which deteriorates the placement is accepted with a probability of $e^{-\frac{\Delta E}{T}}$. In this formula, ΔE is the change in quality due to the perturbation and T the temperature control parameter. Initially the temperature is such that nearly all moves are accepted. Kirkpatrick et. al. [43] were the first to proposed to gradually

decrease the temperature. In this way the placement slowly 'cools off', settling in configurations with a better quality. The main advantage of the slow cooling process is that it allows the placement to escape from a local optimum. It can be proven that the placement will 'solidify' in the configuration with the best quality value if an unlimited number of perturbations and an infinitely slow cooling schedule are used. The initial temperature, the number of perturbations, the choice of perturbations, the cooling schedule and the stop criterion are all important parameters that affect the speed of the algorithm and the quality of the final result. Otten and van Ginneken [63] have published methods to derive optimal values for some of these parameters.

Simulated annealing is being applied extensively on layout problems. TimberWolf [75] has become one of the 'classic' standard-cell placement systems based entirely on simulated annealing. The uniform block size and the row-based design-style of the standard-cell approach enable a simple and adequate model of the physical placement. Perturbations are defined as a swap of two blocks, while the quality of a placement configuration is measured by weighting the total estimated wire length and the area of the bounding box (determined by the length of the longest row).

General-cell placements are harder to capture in an appropriate model for simulated annealing. The modules cannot be assigned to 'slots' in a model of the chip area, due to their variable shape and size. This complicates the procedure to select useful perturbations. Wong and Liu [88] propose to implement the perturbations as operations on the slicing tree. Although this method has the advantage of generating a sliceable placement, a drawback that a single perturbation may span a considerable distance in the solution space. Therefore aggressive steps between remote local minima are possible even at low temperatures. Such 'rude' perturbations work against the delicate local optimization (the 'crystalization') which is required close to the temperature at which placement solidifies. Recently, Upton et. al. [84] published a modified and improved version of Wong and Liu's method. They report reasonably good results with a more accurate routing area estimation technique. Other approaches for general cell placement (e.g. [80]) define perturbations as arbitrary moves of modules. The overlap of the modules is penalized to ensure that the final result is legal. Obviously, this method does not generally result in a sliceable placement.

Many millions of perturbations are required during the cooling process. Therefore simulated annealing requires a simple model of the placement and the cost of each configuration must be updated quickly. Especially for general-cell placement the models are rather inaccurate because the area of the channels is not properly taken into account. Due to this large margin of error, the best quality configuration is not necessarily a good placement. Apart from the total interconnect length, many other aspects (such as the channel structure) have a considerable influence on the final size.

Most of them are neglected during placement.

7.1.3 Manual placement

In the standard-cell design style over 10.000 modules can be placed and routed in one pass. Obviously the placement of this type of circuits is virtually impossible without an automatic placer. Despite many efforts on automatic general-cell placement algorithms, however, the automatically produced placements usually have to face defeat against the manually produced ones. An important observation at this point is that the number of modules in a general-cell placement is much smaller than in a standard-cell placement. In general up to 20 blocks are routed at one level of hierarchy in the general-cell style. Notice that this hierarchical layout design style, which enables reduction of the module count, is not possible with standard cells. Manual placement of a small amount can be performed quickly and will generally achieve superior results.

A human being is able to consider different aspects of the problem at the same time. No algorithm is able to combine shape fitting of the blocks, the channel structure and the connectivity of the blocks in one pass. A human being is also fast in learning from its mistakes. The placement can be iteratively improved using quick feedback from the routing process. The time it takes to find acceptable placements manually is usually very short. Computer assistance is often used in this manual optimization process. This includes functions such as total wire length estimation, routing area estimation, indication of move directions of the blocks and various display and manipulation commands.

7.2 A general-cell routing system

The majority of the algorithms presented in this thesis have been implemented in a routing system for full-custom general-cell VLSI. The system is generally known as the *Delft Placement and Routing System.* As the underlying concept the slicing channel structure was adopted, mainly because of the 100% completion guarantee in combination with a channel router.

The placement and routing system is one of the tools in NELSIS IC design environment. NELSIS was a joint project of the three Dutch technical universities. Apart from the placer and router, the system also contains circuit level synthesis tools, layout and circuit editors, layout extraction tools and simulators. The design data is stored in a common database around which a *Design Management Interface* is built to enable consistent and standardized access [85]. The standardization on a common

database allows an easy integration of the router with other tools, without having to revert to cumbersome translation programs. In this way the following (typical) design steps to generate a module can be performed in a matter of minutes:

1. Net list generation.
2. Net list simulation (pre-layout).
   If the simulation is unsatisfactory go to step 1.
3. Placement and routing.
   Try a number of different placements.
4. Layout extraction
5. Switch-level or SPICE simulation of the extracted net list.
   If the simulation is unsatisfactory go to step 1 or step 3.

In this section we focus step 3, that is, the Delft Placement and Routing system. As input, the system expects the following items in the database:

- A net list description of the *father* module, containing the desired connectivity between its sons.
- The layout description of the *son* modules, including the position of the terminals.

The task of the system in to generate the layout of the father cell. This layout will consist of the son modules as instances and a wiring pattern which implements the given net list description. Obviously, the objective is to create a father module which is as small as possible.

As a first step, a topological placement is generated. A number of automatic placers and an interactive placer are available for this task. A popular method is to generate an automatic placement first, and then improved it manually using the interactive placer. The result must be a sliceable placement, which is stored in the *floorplan* view of the database. More than one placement of the same circuit can be stored. To facilitate the manual placement process, the interactive placer can show the connectivity of selected modules.

In the next step the global router is called. It finds a topological path for each signal net through the areas around the blocks. The channel capacity is taken into account to prevent congested areas. The global router can also efficiently use free wires in blocks (the *feed-throughs* and/or *wire-throughs*) to shorten wire length. The

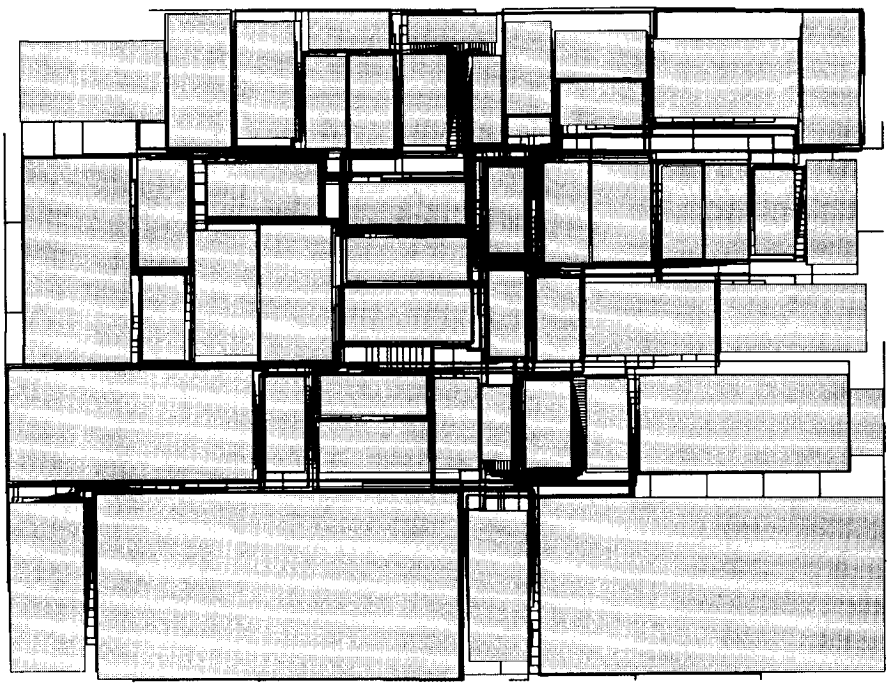

Figure 78: The AMI49 *benchmark circuit, as routed by our system.*

global router automatically detects feed-throughs by looking at the connectivity of the son cells. The global router is described in more detail in [7].

Power nets are treated differently by the global router. They are routed by growing two interdigitated trees, each from opposite sides of the chip. In this way the power nets are kept planar, that is, in a single layer and avoiding vias. Their width is calculated according to the electromigration constraints, which is given as a formula by the design rules. The power nets will be routed by the channel router as nets with a special ordering and width. Figure 3 (on page 4) shows an example of a circuit in which the power wires were routed using this strategy.

In contrast to many other macro-cell routing systems we perform the channel definition after the global routing phase. The channel definition algorithm as published in [10] is used. It takes the 'flow' of wires through the channels into account. Since congestion at channel junctions strongly influences the final chip size, a connectivity-biased channel structure is of paramount importance.

Detailed routing is performed entirely by the channel router presented in chapter 3. This channel router is embedded in an interface program, which takes care of the administrative tasks during detailed routing. A number of improvements were also implemented as part of this program:

- Ordering the floating terminals on the left and right side of the channel. The algorithm described in section 6 is used for this task.
- Determining the *detailed placement*. The height of the channel is determined by the channel router, but the lateral offset of the bottom and the top parts can be determined by a detailed placer. The algorithm we employ [34] is quite straightforward. For each channel a number of values for the lateral offset of the blocks is evaluated. The estimated channel height (based on the density) and the overlap on the left and right sides are weighted in the evaluation. In this way busses of terminals are more likely to be abutted, which reduces the via-count and the channel height.
- Performing the placement of the terminals of the father module. In this way the density in the 4 outer channels can be reduced drastically. This feature is especially useful for hierarchical routing.

The final result is a new father cell in the layout-view of the database. Table 8 shows the turn-around times of the entire system for a number of characteristic circuits.

7.3 Application and evaluation of the algorithms

At Delft university of technology, the placement and routing system has been used for the design of a number of large chips. The first chip, a 'cordic' signal processor with a very regular structure, has been successfully processed. It is currently operating as a computer component. A cellular logic processor chip for image processing applications has just returned from the factory. It is currently being tested. A number of other designs also have been routed by our system.

The concepts of the entire routing system, as presented in the previous section, were adopted in a layout system of Philips Components. The channel router, as a stand-alone tool, is integrated in a procedural layout generation tool called MODGEN, which was developed at Philips Research Laboratories. In this environment it is used to 'glue' modules together. Its capability to handle variable wire width and rugged channel boundaries made the channel router especially suitable for this task. Many modules for commercial signal processing chips have been assembled in this way.

	BBL1 (XEROX)	BBL2 (AMI33)	BBL3 (APTE)	BBL4 (AMI49)	BBL5 (HP)
no. of blocks	10	33	9	49	11
no. of nets	203	123	97	408	83
no. of I/O pins	2	42	73	22	45
global routing time†	0:07	0:18	0:04	0:26	0:03
detailed routing time†	0:41	1:08	0:26	1:55	0:28
Average net length	2800	1200	5200	2200	2600
Average no. of vias per net	4.5	7.8	4.5	5	4.8
Approx. no. of rectangles‡	7500	7000	3800	14700	3000

† The typical 'gross' response times: the time elapsed between the command was entered and the prompt returns. The turn around time for a placement is the sum of the global and detailed routing time. The global routing time includes automatic channel construction. The values were measured on a HP9000-835 computer ($\approx$ 15 MIPS).

‡ The number of layout boxes generated by the router. We use it as a measure for the size of the circuit.

Table 8: Characteristics and results of the MCNC *benchmark circuits.*

Every other year the Microelectronics Center of North Carolina (MCNC) organizes the International Workshop on Layout Synthesis. Prior to the workshop server benchmarks for various layout problems are distributed. This distribution has become *the* set of benchmarks for layout algorithms. At the 1990 workshop five general-cell benchmark circuits were made available, covering a spectrum of problems for routing systems. The characteristics are summarized in table 8. For each benchmark a set of modules, a terminal placement and a set of design rules were specified. A two-layer production process was prescribed, in which the design rules are such that the size of a via is the same as the width of a wire. Obviously, this favors conventional grid-based routers. In most real-life fabrication technologies the width of the via patterns is larger than the minimum width of a wire.

The AMI33 (figure 3) and AMI49 (figure 78) circuits contain many modules of approximately the same size. Also the net list is quite unstructured. Therefore many reasonably good placements are expected to exists for these circuits. These two benchmarks originated from the UC Berkeley, who obtained them from AMI. It is doubtful whether they were extracted from a real-life chip. The XEROX (figure 79), HP and APTE circuits are more realistic. They consist of a few very large modules and a few small ones, connected by a rather structured net list. Only very few feasible placements exist for them. This was demonstrated during the 1990 workshop, where

all systems came up with almost the same placement for them. Therefore these benchmarks compare the quality of the routing, rather than the placement of various systems.

system/ref.	BBL1 (XEROX) area	rel.(1)	BBL2 (AMI33) area	rel.(1)	BBL3 (APTE) area	rel.(1)	BBL4 (AMI49) area	rel.(1)	BBL5 (HP) area	rel.(1)
Delft P&R	26.2	0.0%	2.46	0.0%	53.7	0.0%	48.7	0.0%	11.36	0.0%
Bear [64]	28.1	7.3%	2.67	8.5%	55.0	2.4%	50.8	4.3%	13.10	15.3%
Seattle S. [84](2)	28.6	7.5%	2.42	-2.3%	54.8	2.0%	48.9	0.4%	11.85	4.3%
Kyoto [60](3)	27.5	4.5%	2.24	-8.9%	54.1	0.7%	51.5	5.7%	12.1	6.5%
TWolfMC [80]	-		2.57	4.5%	-		-		-	
Grca/GDT [39](3)	28.8	9.9%	2.88	17.1%	-		-		-	
Frodo [52](3)	29.1	11.1%	2.92	18.7%	-		55.7	14.4%	-	
Vital(4)	31.7	20.9%	3.21	30.0%	-		-		-	
Mosaico [1](4)	29.0	10.7%	3.10	26.0%	-		-		-	

(1) Relative performance compared to our result.

(2) Terminal placement not according to given specification. This 'trick' reduces the density in the 4 outer channels considerably.

(3) Primarily placement or floor planning algorithms. The routing is performed by another system.

(4) Results of 1988 MCNC workshop. No recent figures are known.

Table 9: Comparison of the most recent results on the general cell benchmark circuits of the 1990 MCNC International Workshop on Layout Synthesis. All data is for 'level 1', that is, the power nets are routed as ordinary nets.

Table 9 captures all results on the MCNC general-cell benchmark circuits published so far. Bear [21, 64] is a comprehensive layout system developed in the group of prof. Kuh at UC Berkeley. It can use a wide variety of detailed routers, such as a switch-box router (Mighty [77]) a channel router and even an L-shaped channel router [14]. Therefore it is not restricted to sliceable placements. The majority of the benchmark results they reported, however, had sliceable placements. The Mosaico system [1] was developed in the group of prof. Sangiovanni-Vincentelli at UC Berkeley. It can handle a limited class of non-sliceable placements. Detailed routing is performed on a grid using Mighty [77], Chameleon [2] or Yacr2 [71]. The row Kyoto shows the result of a branch-and-bound placer developed at Kyoto University [60]. The routing was performed by an unknown commercial system. TimberWolfMC [80] is a macro-cell router developed in prof. Sechen's group at Yale university. As with all TimberWolf versions, simulated annealing is the algorithm used for placement and global routing. TimberWolfMC subdivides the area around the modules into smaller

routing regions which are not necessarily channels. Therefore it can handle the non-sliceable placements which are created by its placer. It performs a maze routing on each of the areas using Mighty [77]. Obviously, no 100% completion guarantee can be given. In TimberWolfMC the junction terminals at the region boundaries are ordered using the algorithm presented in chapter 6. Seattle Silicon [84] is a commercial router from the company of the same name. Grca and Frodo are floorplanning and placement algorithms, of which the result was routed by another system. Finally, Vital is an in-house router at MCNC.

As indicated by table 9, our system consistently produces good results for all benchmark circuits. Since the placement of the XEROX, HP and APTE benchmarks was almost the same for all systems, these circuit form reasonable benchmarks to compare the detailed routing strategies and algorithms. Therefore we may conclude that the 'context-driven' approach to the detailed routing problem has proven to be successful. The better tuning of the algorithms towards their environment gave our system a decisive edge.

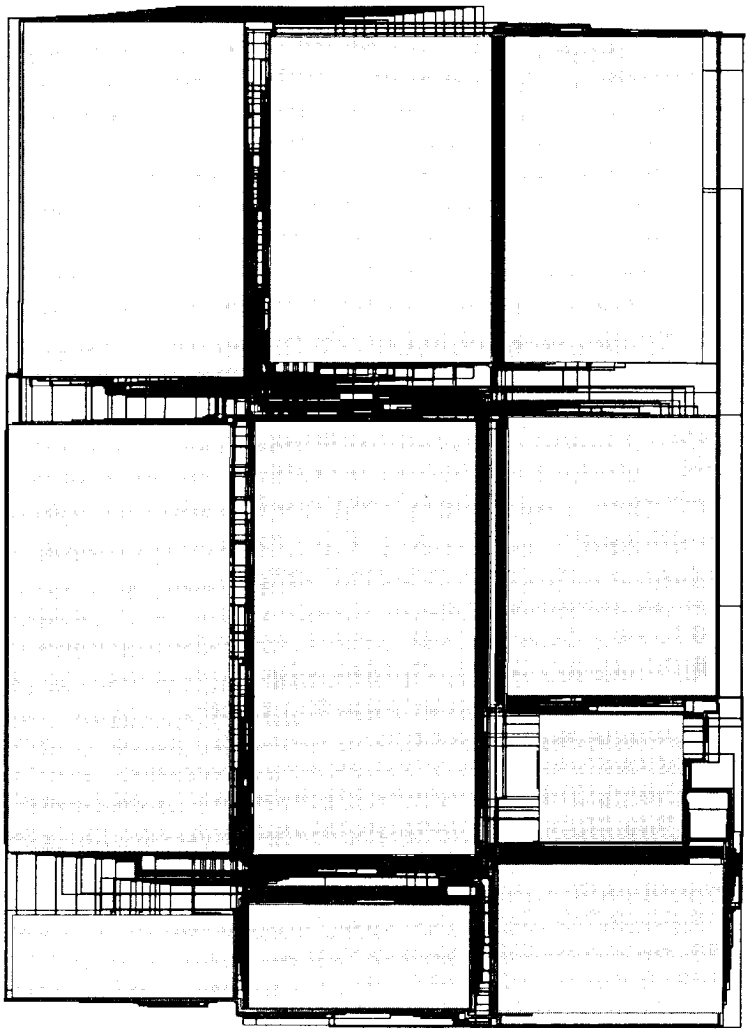

Figure 79: The XEROX *benchmark circuit contains a number of very large modules.*

Bibliography

[1] M. Bearslee, J. Burns, A. Casotto, M. Igusa, F. Romeo, and A. Sangiovanni-Vincentelli. Mosaico: an integrated macro-cell layout system. *Proc. International Workshop on Placement and Routing*, 6.1, May 1988.

[2] D. Braun, J. Burns, S. Devadas, H.K. Ma, K. Mayaram, F. Romeo, and A. Sangiovanni-Vincentelli. Chameleon: a new multi-layer channel router. In *Proc. 23th Design Automation Conference*, pages 495 502, ACM/IEEE, 1986.

[3] D. Braun, J. Burns, F. Romeo, K. Mayaram, S. Devadas, H.-K.T. Ma, and A. Sangiovanni-Vincentelli. Technigues for multi-layer channel routing. *IEEE Transactions on Computer-Aided Design*, CAD-7(6):698–712, October 1988.

[4] R.K. Brayton, C.L. Chen, J.A.G. Jess, R.H.J.M. Otten, and L.P.P.P. van Ginneken. Wire planning for stackable design. In *Proc. International Symposium on VLSI Technology, Systems and Applications*, pages 269–273, May 1987.

[5] P. Bruel and P. Sun. A "greedy" three layer channel router. In *Proc. Custom Integrated Circuits Conference '85*, pages 298–300, Portland, Oregon, USA, May 1985.

[6] M. Burstein and R. Pelavin. Hierarchical channel router. In *Proc. 20th Design Automation Conference*, pages 591–597, ACM/IEEE, 1983.

[7] H. Cai. *Routing channels in VLSI layout.* PhD thesis, Delft University of Technology, March 1989.

[8] H. Cai and P. Groeneveld. *Delft Placement and Routing System: User's Manual.* Delft University of Technology, 2 edition, 1989.

[9] H. Cai and P. Groeneveld. An overview of the delft placement and routing system. *Delft Progress Report, Special Issue on Signal Processing and Computer-Aided Circuit Design*, 12(3):291–303, 1988.

[10] H. Cai and R.H.J.M. Otten. Conflict-free channel definition in building-block layout. *IEEE Transactions on Computer-Aided Design*, 8(9):981–989, 1989.

[11] H. Cai, H. Verheyen, R. Nouta, and P. Dewilde. An automatic routing system for general cell vlsi circuits. In *Proc. Custom Integrated Circuits Conference '85*, pages 68–71, Portland, Oregon, USA, May 1985.

[12] Y. Cai and D.F. Wong. Optimal via shifting in channel compaction. In *Proc. European Design Automation Conference*, pages 186–190, Edinburgh, GB, February 1990.

[13] C.K. Chen and D.N. Deutsch. Improved channel routing by via minimization and shifting. In *Proc. 25th Design Automation Conference*, pages 677–680, ACM/IEEE, 1988.

[14] H.H. Chen. Routing l-shaped channels in nonslicing-structure placement. In *Proc. 24th Design Automation Conference*, pages 152–158, ACM/IEEE, June 1987.

[15] H.H. Chen. Trigger: a three-layer gridless channel router. In *Proc. International Conference on Computer-Aided Design '86*, pages 196–169, IEEE, November 1986.

[16] H.H. Chen and E.S. Kuh. Glitter: a gridless variable-width channel router. *IEEE Transactions on Computer-Aided Design*, CAD-5(4):459–465, October 1986.

[17] H.H. Chen and J-F. Lee. Compacted channel routing on deutsch's new benchmarks. In *Proc. International Workshop on Layout Synthesis*, page 4.1, Microelectronics Center of North Carolina, Research Triangle Park, North Carolina, May 1990.

[18] N.P. Chen. Building block routing - a symbolic approach. In *Proc. Custom Integrated Circuits Conference '88*, page 11.2, Rochester, NY, May 1988.

[19] Y.K. Chen and M.L. Lui. Three-layer channel routing. *IEEE Transactions on Computer-Aided Design*, CAD-3(2):156–163, May 1984.

[20] J. Cong and C.L. Liu. Over-the-cell channel routing. *IEEE Transactions on Computer-Aided Design*, CAD-9(4):408–418, April 1990.

[21] W. Dai, H.H. Chen, R. Dutta, M. Jackson, E.S. Kuh, M. Marek-Sadowska, M. Sato, D. Wang, and X. Xiong. Bear: a new building-block layout system.

In *Proc. International Conference on Computer-Aided Design '87*, pages 34–37, IEEE, November 1987.

[22] D.N. Deutsch. Compacted channel routing. In *Proc. International Conference on Computer-Aided Design '85*, pages 223–225, November 1985.

[23] D.N. Deutsch. A dogleg channel router. In *Proc. 13th Design Automation Conference*, pages 425–433, ACM/IEEE, 1976.

[24] D.N. Deutsch. Two new and more difficult channel routing problems. *IEEE Transactions on Computer-Aided Design*, CAD-8(4):448, April 1989.

[25] D.N. Deutsch and P. Glick. An over-the-cell channel router. In *Proc. 17th Design Automation Conference*, pages 32–39, ACM/IEEE, 1980.

[26] E.W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Matematik*, 1:269–271, 1959.

[27] R.J. Enbody and H.C. Du. Near-optimal n-layer channel routing. In *Proc. 23rd Design Automation Conference*, pages 706–714, ACM/IEEE, 1986.

[28] C.M. Fiduccia and R.M. Mattheyses. A linear-time heuristic for improving network partitions. In *Proc. 19th Design Automation Conference*, pages 175–181, ACM/IEEE, 1982.

[29] U. Flemming. Representation and generation of rectangular dissections. In *Proc. 15th Design Automation Conference*, pages 138–144, ACM/IEEE, 1978.

[30] M.R. Garey and D.S. Johnson. *Computers and Intractability - a Guide to the Theory of NP-Completeness*. W.H. Freemann and Co. San Francisco, 1979.

[31] P. Groeneveld. A Multiple-Layer Contour-Based Gridless Channel Router. *IEEE Transactions on Computer Aided Design of Integrated Circuits and Systems*, CAD-9(12), December 1990.

[32] P. Groeneveld. On global wire ordering for detailed routing. In *Proc. 26th Design Automation Conference*, pages 155–161, ACM/IEEE, Las Vegas, U.S.A., June 1989.

[33] P. Groeneveld. Wire Ordering for Detailed Routing. *IEEE Design and Test of Computers*, 6(6):6–17, December 1989.

[34] P. Groeneveld and H. Cai. The delft placement and routing system. In G.W. Zobrist, editor, *To appear in Progress in Computer Aided VLSI design*, Ablex Publishing Corp., Norwood, New Jersey, USA, 1991.

[35] P. Groeneveld and H. Cai. The delft placement and routing system. *Proc. International Workshop on Placement and Routing*, 6.2, May 1988.

[36] P. Groeneveld, H. Cai, and P. Dewilde. A contour-based variable-width gridless channel router. In *Proc. International Conference on Computer-Aided Design '87*, pages 374–377, IEEE, Santa Clara, USA, November 1987.

[37] P. Groeneveld and P. Stravers. *All you ever wanted to know about Fish, Fishbone and Gothi.* Delft University of Technology, January 1991.

[38] A. Hashimoto and S. Stevens. Wire routing by optimizing channel assignment within large apertures. In *Proc. 8th Design Automation Conference*, pages 155–169, ACM/IEEE, 1971.

[39] A. Herrigel. Grca: a global approach for floorplanning synthesis in vlsi macro-cell design. In *Proc. International Conference on Computer-Aided Design '90*, pages 152–155, IEEE, November 1990.

[40] D.W. Hightower. A solution to line routing problems on the continuous plane. In *Proc. 6th Design Automation Workshop*, pages 1–24, 1969.

[41] C.S. Horng and M. Lie. An automatic/interactive layout planning system for arbitrarily sized building blocks. In *Proc. 18th Design Automation Conference*, pages 293–300, ACM/IEEE, 1981.

[42] B.W. Kernighan and S. Lin. An efficient heuristic procedure for partitioning graphs. *The Bell System Technical Journal*, 291–307, February 1970.

[43] S. Kirkpatrick, C.D. Gelett, and M.P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, May 1983.

[44] J.M. Kleinhans, G. Sigl, and F.M. Johannes. Gordian: a new global optimization/rectangle dissection method for cell placement. In *Proc. International Conference on Computer-Aided Design '88*, pages 506–509, IEEE, 1988.

[45] R.J.H. Koopman, R. Peset LLopis, and H.G. Kerkhoff. Digital cmos sea-of-gates core cells and master images. In *Proc. 16th European Solid-State Circuit Conterence (ESSIRC)*, pages 245–248, Grenoble, France, September 1990.

[46] H.E. Krohn. An over-the-cell gate-array channel router. In *Proc. 20th Design Automation Conference*, pages 665–670, ACM/IEEE, 1983.

[47] F.E. Kroon. *A multiple layer channel router for gate array environments.* Master's thesis, Delft University of Techology, March 1990.

[48] A.S. LaPaugh. *Algorithms for Integrated Circuit Layout: an Analytic Approach.* PhD thesis, Laboratory for Computer Science, MIT, Cambridge, MA, November 1980.

[49] U. Lauther. A min-cut placement algorithm for general cell assemblies based on a graph representation. In *Proc. 16th Design Automation Conference*, pages 1–10, ACM/IEEE, 1979.

[50] C.Y. Lee. An algorithm for path connection and its applications. *IRE Transactions on Electronic Computers*, EC-10(3):346–365, 1961.

[51] T. Lengauer. *Combinatorial algorithms for integrated circuit layout. Applicable theory in computer science*, Wiley - Teubner, 1990.

[52] T. Lengauer and R. Müller. A robust framework for hierarchical floorplanninning with integrated global wiring. In *Proc. International Conference on Computer-Aided Design '90*, pages 148–151, IEEE, November 1990.

[53] C.A. Mead and L.A. Conway. *Introduction to VLSI systems. Addison Wesley series in computer science*, Addison Wesley, 1980.

[54] N. Metropolis, A. Rosenbluth, M. Rosenbluth, A. Teller, and E. Teller. The monte-carlo method in statistical physics. *J. Chem. Phys*, 21:1087, 1953.

[55] K. Mikami and K. Tabuchi. A computer program for optimal routing of printed circuit boards. In *IFIPS proceedings H47L*, pages 1475–1478, 1968.

[56] C.H. Ng. A gridless variable-width channel router for macro cell design. In *Proc. 24th Design Automation Conference*, pages 633–636, ACM/IEEE, 1987.

[57] C.H. Ng. A symbolic-interconnect router for custom ic design. In *Proc. 21st Design Automation Conference*, pages 52–57, ACM/IEEE, 1984.

[58] I. Ohkura. Gate-isolation: a novel basic cell configuration for cmos gate arrays. In *Proc. Custom Integrated Circuit Conference*, pages 307–310, 1982.

[59] T. Ohtsuki. Maze-running and line-search algorithms. In *Advances in CAD for VLSI, Volume 4*, pages 99–131, North-Holland, New York, 1986.

[60] H. Onodera, Y. Taniguchi, and K. Tamaru. Branch-and-bound placement for building block layout. In *Proc. International Workshop on Layout Synthesis*, page 11.1, Microelectronics Center of North Carolina, Research Triangle Park, North Carolina, May 1990.

[61] R.H.J.M. Otten. Automatic floorplan design. In *Proc. 19th Design Automation Conference*, pages 261–267, ACM/IEEE, 1982.

[62] R.H.J.M. Otten. Complexity and diversity in ic layout design. In *Proc. ICCC*, pages 764–767, 1980.

[63] R.H.J.M. Otten and L.P.P.P. van Ginneken. Stop criteria for simulated annealing. In *Proc. International Conference on Computer-Aided Design '88*, pages 549–553, 1988.

[64] M. Pedram, W. Dai, M. Marek-Sadowska, and Y. Ogawa. Ongoing research and development of the bear layout system. In *Proc. International Workshop on Layout Synthesis*, page 1.1, Microelectronics Center of North Carolina, Research Triangle Park, North Carolina, May 1990.

[65] D.B. Polk. A three-layer gridless channel router with compaction. In *Proc. 24th Design Automation Conference*, pages 146–151, ACM/IEEE, 1987.

[66] B.T. Preas. Channel routing with non-terminal doglegs. In *Proc. European Design Automation Conference*, pages 451–458, 1990.

[67] B.T. Preas and C.W. Gwyn. Methods for hierarchical automatic layout. In *Proc. 15th Design Automation Conference*, pages 206–212, ACM/IEEE, 1978.

[68] B.T. Preas and M.J. Lorenzetti. *Physical design automation of VLSI systems.* The Benjamin/Cummings Publishing Company, 1988.

[69] B.T. Preas and W.M. vanCleemput. Placement algorithms for arbitrarily shaped blocks. In *Proc. 16th Design Automation Conference*, pages 474–480, ACM/IEEE, 1979.

[70] N.R. Quinn. The placement problem as viewed from the physics of classical mechanics. In *Proc. 12th Design Automation Conference*, pages 173–187, ACM/IEEE, 1975.

[71] J. Reed, A. Sangiovanni-Vincentelli, and M. Santomauro. A new symbolic channel router: yacr2. *IEEE Transactions on Computer-Aided Design*, CAD-4(3):208–219, July 1985.

[72] R.L. Rivest. The pi (placement and interconnect) system. In *Proc. 19th Design Automation Conference*, pages 475–481, ACM/IEEE, 1982.

[73] R.L. Rivest and C.M. Fiduccia. A greedy channel router. In *Proc. 19th Design Automation Conference*, pages 418–424, ACM/IEEE, 1982.

[74] J. Royle, M. Palczewski, H. Verheyen, N. Naccache, and J. Soukup. Geometrical compaction in one dimension for channel routing. In *Proc. 24th Design Automation Conference*, pages 140 145, ACM/IEEE, 1987.

[75] C. Sechen and A. Sangiovanni-Vincentelli. The timberwolf placement and routing package. *IEEE Journal of Solid-State Circuits*, SC-20(2):510–522, April 1985.

[76] L. Sha and R.W. Dutton. An analytical algorithm for placement of arbitrarily sized rectangular blocks. In *Proc. 22nd Design Automation Conference*, pages 602–608, ACM/IEEE, 1985.

[77] H. Shin and A. Sangiovanni-Vincentelli. A detailed router based on incremental routing modifications: mighty. *IEEE Transactions on Computer-Aided Design*, CAD-6(6):942–954, November 1987.

[78] Y. Shirashi and Y. Sakemi. A permeation router. *IEEE Transactions on Computer-Aided Design*, CAD-6(3):462–471, May 1987.

[79] K.J. Supowit and E.A. Slutz. A placement algorithm for custom vlsi. *Computer-Aided Design*, 16(1):45–50, 1984.

[80] W. Swartz and C. Sechen. New algorithms for the placement and routing of macro cells. In *Proc. International Workshop on Layout Synthesis*, page 1.2, Microelectronics Center of North Carolina, Research Triangle Park, North Carolina, May 1990.

[81] T.G. Szymanski. Dogleg channel routing is np-complete. *IEEE Transactions on Computer-Aided Design*, CAD-4(1):31–41, January 1985.

[82] R.Y. Tsui. Variable track space channel routing. In *Proc. International Conference on Computer-Aided Design '86*, pages 200–203, November 1986.

[83] K. Ueda, H. Kitazawa, and I. Hrada. Champ: chip floor plan for hierarchical vlsi layout design. *IEEE Transaction on Computer-Aided Design*, CAD-4(1):12–22, January 1985.

[84] M. Upton, K. Samii, and S. Sugiyama. Integrated placement for mixed macro-cell and standard-cell designs. In *Proc. 27th Design Automation Conference*, pages 32–35, ACM/IEEE, June 1990.

[85] N. van der Meijs, T.G.R. van Leuken, P. van der Wolf, I. Widya, and P. Dewilde. A data management interface to facilitate cad/ic software exchanges. In *Proc. ICCD '87*, pages 403–406, 1987.

[86] L.P.P.P. van Ginneken. Gridless routing of general floor plans. In *Proc. International Conference on Computer-Aided Design '87*, pages 30–33, November 1987.

[87] H.J.M. Veendrick, A.J.M. van den Elshout, D.W. Harberts, and T. Brandt. An efficient and flexible architecture for high-density gate arrays. *IEEE Journal of Solid-State Circuits*, 25(5):1153–1157, October 1990.

[88] D. F. Wong and C. L. Liu. A new algorithm for floorplanning design. In *Proc. 23rd Design Automation Conference*, pages 101–107, ACM/IEEE, 1986.

[89] L.S. Woo, C.K. Wong, and D.T. Tang. Pioneer: a macro-based floor-planning design system. *VLSI systems design*, August 1986.

[90] T. Yoshimura. An efficient channel router. In *Proc. 21st Design Automation Conference*, pages 38–44, ACM/IEEE, June 1984.

[91] T. Yoshimura and E.S. Kuh. Efficient algorithms for channel routing. *IEEE Transactions on Computer-Aided Design*, CAD-1(1):25–35, January 1982.

Acknowledgments

I would like to thank all people who contributed in any way to the development of the 'Delft placement and routing system'. I experienced Delft University of Technology as a very stimulating and dynamic research environment. My promotion work was supported by Philips Research Laboratories, Eindhoven, The Netherlands. The cordial relation with the people of the 'NatLab' provided valuable input and ideas for my work.

About the author

Patrick Groeneveld was born in Heemskerk, the Netherlands on may 21, 1964. In may 1982 he received the VWO diploma from the 'Rijnlands Lyceum' in Sassenheim. In september of that year he started his study of electrical engineering at Delft University of Technology in Delft, the Netherlands. In 1987 he received the Ingenieur degree (the equivalent of an M.Sc). In june 1987 mr. Groeneveld joined the Laboratory for Network Theory of the faculty of Electrical Engineering at Delft University of Technology, where he embarked on his work towards a Ph.D. degree doing research on the placement and routing of VLSI circuits under the supervision of Prof. dr. ir. P.M. Dewilde. Since the beginning of 1988 he switched to the group of Prof. dr. ir. R.H.J.M. Otten, while continuing to work on the same subject. Most of his work has been published in international journals [36, 35, 32, 33, 31, 9, 34, 8, 37]. In june 1989 mr. Groeneveld received a 'best paper award' at the 26th Design Automation Conference in Las Vegas.