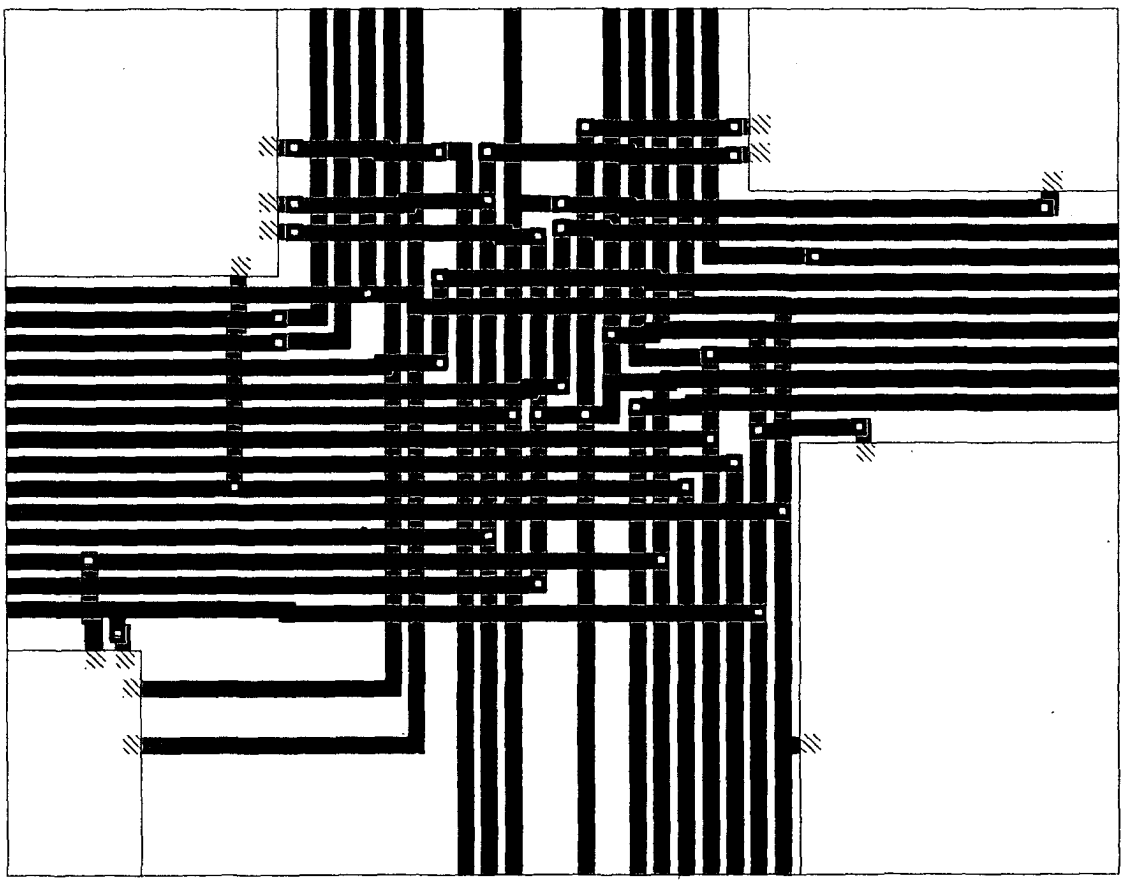
ROUTING CHANNELS IN VLSI LAYOUT



Hong Cai

ROUTING CHANNELS IN VLSI LAYOUT

ROUTING CHANNELS IN VLSI LAYOUT

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 30 maart 1989 te 16.00 uur

door

Hong Cai

geboren te Changsha
electrotechnisch-ingenieur

Delft University Press/1989

Dit proefschrift is goedgekeurd door de promotoren
prof.dr.ir. P. Dewilde en prof.dr.ir. R.H.J.M. Otten

CIP-gegevens: Koninklijke Bibliotheek, Den Haag
ISBN 90-6275-536-4

Stellingen behorende bij het proefschrift:

ROUTING CHANNELS IN VLSI LAYOUT

door

H. Cai

1. Gegeven een *floor-plan-graaf* kan een conflict-vrije kanaalstructuur gevonden worden in polynomiale tijd, waarvan de gerealiseerde bonus van kanaalsplitsingen optimaal is.

2. De methode, voorgesteld door W.M. Dai e.a. in "Routing Region Definition and Ordering Scheme for Building-Block Layout" (IEEE Transactions on Computer-Aided Design CAD-4(3) pp. 189-197, Juli 1985), om *empty rooms* in floor-plan-grafen te elimineren is niet correct.

3. De beperkingen van de *slicing topologie* worden ruimschoots gecompenseerd door de voordelen ervan.

4. Voor de productie van bruikbare CAD software is het essentieel dat ontwikkelaars werken in een omgeving van IC ontwerpers.

5. De betrouwbaarste en uiteindelijk ook meest kosten-besparende methode om universiteits-software te gebruiken in een industriele omgeving is het volledig herschrijven ervan.

6. Daar de oosterse- en westerse geneeswijzen elkaar goed aan kunnen vullen is de Nederlandse benaming 'alternatieve geneeskunde' voor de eerste misleidend.

7. Gezien de hoge connectiviteit tussen de centrale bibliotheek van de TU en de medewerkers van de universiteit verdient het aanbeveling de centrale bibliotheek in het centrum van de universiteit te plaatsen.

8. Onderzoekers van Kunstmatige Intelligentie kunnen uit de studie van het "go"-spel een beter begrip van menselijke denkprocessen verkrijgen dan uit onderzoek van het schaakspel omdat "go" meer gebaseerd is op feature-recognition en pattern-directed-reasoning dan op tree-search en look-ahead methodes.

9. Voor het vervaardigen van goede artikelen is het beschikken over typographische en esthetische kennis minstens zo belangrijk als de beschikbaarheid van een goede en gebruikersvriendelijke textverwerkingsfaciliteit.

10. Weten het niet-weten, dat is hoog.
Niet weten het weten, dat is een ziekte.
Wie ziek is van die ziekte is niet ziek.

Lao-tse in Tao-Te-Tjing *

---

* Uitgever: Ankh-Hermes, Devender, 1986, 11e dr.
Vertaling: Ir. J.A. Blok.

CONTENTS

SUMMARY

In this dissertation we present a number of algorithms for the automatic routing of interconnections in VLSI building-block layout.

The layout of this type of integrated circuits consists of a set of rectangularly shaped cells (the building blocks), which compose the design and which are placed in a two dimensional finite space, and of the interconnections between the terminals on the periphery of these cells in accordance with the schematic of the circuit to be implemented. The goal of the layout design is to achieve as small as possible a chip area that will satisfy a set of constraints.

Layout optimization is a very complex combinatorial problem. A divide and conquer strategy is used to solve the layout problem by dividing it into a sequence of subproblems. The layout problem is traditionally divided into several steps: floorplanning, placement, the definition of routing channels, global routing and detailed routing. In this dissertation algorithms for the topological definition of channels, the global routing and the geometrical definition of channels are presented.

The topological definition of channels determines the decomposition of the routing area into routing channels and the order in which the channels are routed. In contrast to traditional approaches we define and order the channels after the global routing. This approach possesses the advantage that global routing information can be taken into account to select the optimal channel structure. We present a polynomial algorithm for the channel definition and ordering problem. The existence of a conflict-free channel structure is guaranteed by enforcing a sliceable placement.

In the global routing step a decision on the path taken by each of the interconnection nets is made. Algorithms for finding the shortest connection path are described. A separate algorithm is developed for the power net

routing, because the two power nets must be planarly routed with variable wire width.

Before a channel can be physically routed by a channel router, the geometrical channel envelope must be determined. In order to minimize the channel density the relative position of the blocks adjacent to the channel is first optimized. Moreover, an ordering of the terminals on the channel ends is determined to minimize the number of wire crossings in the channel intersection areas.

In the last chapter an integrated placement and routing system for generating building-block layout is briefly described. Most of the algorithms presented have been implemented into this system. Some experimental results and design experiences in using the system are also presented. Very good results have been obtained.

SAMENVATTING

In dit proefschrift wordt een aantal algoritmen gepresenteerd voor het automatisch bedraden van interconnecties in VLSI schakelingen.

Een geintegreerde schakeling bestaat uit een aantal functionele bouwstenen (blokken) waaruit het ontwerp is samengesteld en de bedrading tussen deze bouwstenen. Het uiteindelijke doel van het layout ontwerp proces is om een zo klein mogelijk chip oppervlakte te realiseren, waarbij tevens voldaan moet worden aan een aantal electrische specificaties en proces voorschriften.

Het bovengenomende layout ontwerp proces is een zeer complex combinatorisch optimalisatie probleem. Daarom wordt een "verdeel-en-heers" principe gebruikt waarbij het probleem in een aantal deelproblemen opgedeeld wordt. Het plaatsings- en bedradingsgedeelte van het layout probleem wordt opgedeeld in een plaatsings stap waarin de blokken geplaatst worden op de chip, een bedradingskanaal definitie stap waarin de bedradingdsruimte in zogenaamde "kanalen" opgedeeld wordt, een globale bedradings stap die bepaalt hoe de netten verbonden worden in de bedradingsruimte en tenslotte een gedetailleerde bedradings stap waarin de uiteindelijke fysieke bedrading gegenereerd wordt. In dit proefschrift worden algoritmen gepresenteerd voor de kanaal definitie en de globale bedradings stap.

Tijdens de topologische kanaal definitie worden zowel het opdelen van de bedradingsruimte in bedradingskanalen als de bedradingsvolgorde van de kanalen bepaald. In tegenstelling tot de gebruikelijke benadering defineren wij de kanalen en de kanaalvolgorde pas na de globale bedradings stap. Dit heeft als voordeel dat de globale bedradings informatie gebruikt kan worden bij het selecteren van de optimale kanaalstructuur. Een efficient algoritme voor het kanaal definitie en ordenings probleem wordt gepresenteerd.

Bij de globale bedradings stap wordt een beslissing genomen over de paden die door de interconnectie netten gevolgd zullen worden. Verscheidene algoritmen voor het vinden van het kortste pad worden behandeld. Een special algoritme is ontwikkeld voor het bedraden van de voedings netten. Deze netten moeten planair bedraad worden met een variabele draadbreedte.

Voordat een kanaal fysiek bedraad kan worden door een kanaalbedrader moet eerst de contour van het kanaal bepaald worden. Om de hoeveelheid bedrading binnen een kanaal te minimaliseren wordt de relatieve positie van de blokken aangepast. Verder wordt een volgorde van de terminals op de uiteinden van de kanalen bepaald om het aantal kruisingen te minimaliseren in de kanaalovergangsgebieden.

In het laatste hoofdstuk wordt een krachtig plaatsings- en bedradingssysteem gepresenteerd. De meeste van de gepresenteerde algoritmen zijn geimplementeerd in dit systeem. Met dit systeem zijn zeer veel belovende resultaten behaald.

1. INTRODUCTION

1.1 Layout and layout styles

The mask patterns with which an integrated circuit (IC) is realized on silicon is called the *layout* of the IC. The layout of integrated circuits consists of the placement of devices (cells) composing the design in a two-dimensional finite space, and of the interconnections of the pins of these devices according to the schematic of the circuit to be implemented [Souk81].

IC design is divided into several design stages. Commonly used design stages are the functional, the logic, the circuit or schematic and the physical or layout design stage. Layout design is a crucial phase of the IC design process, because the area usage, the speed and the power consumption of the circuit depend for a great deal on the quality of the layout. Layout design is also one of the most time-consuming steps in the IC design cycle, because full geometrical details of each individual transistor and wire segment must be drawn correctly. According to actual measurements it takes about 45% of the total design time [Sout83]. The goal of the layout design is to complete the placement and interconnections of the design in the smallest possible area that will satisfying a set of constraints. There is a variety of constraints, for example: design constraints based on the layout style, technological constraints due to design rules and the number of layers that can be used for routing, and performance constraints, e.g. the timing of the logic to be implemented.

An IC can be implemented in different ways. Although the classification in the literature is not standardized, we make use of that which classifies implementations into two categories, namely, *semi-custom* and *full-custom*. In the semi-custom style most masks of the layout image are prefabricated, only the interconnection layers can be customized by the designer. In full-custom ICs all mask layers must be processed as specified by the designer.

Semi-custom ICs are cheap to process and provide a fast design-to-realization-turnaround time. However, the area usage and performance of a semi-custom IC is usually poorer than a comparable well-designed full-custom IC. Layout styles can also be distinguished by the cell sizes and cell arrangement. Both *gate-array* and *standard-cells* use a regular layout for the cells. Cells are arranged in rows and routing channels are defined between the cell rows. Automatic layout systems were introduced first for these layout styles, because of their regular structure which is amenable to layout automation. In gate-arrays the channel widths are fixed. Wirability therefore is an issue of paramount importance in gate-array designs. In contrast, the *macro-cell* or *building-block layout* style which uses cells of arbitrary shape and size is much less regular and more difficult to automate. Cells can occupy any positions on the chip and routing areas are not predefined. This design style is most flexible and provides the designer with full control over the quality of the layout. Note that standard-cells and macro-cells can be mixed to implement the layout of the same circuit.

The recent development of a gate-array technology known as *sea-of-gates*, where the entire area of the chip is covered by devices densely packed, has created new excitement in the IC community. This technology promises excellent area utilization and does not have the routing-area restriction that standard gate-array architectures have, while allowing the prefabricated benefits of standard gate-array technology.

In this thesis we will further concentrate on the macro-cell or the building-block layout (BBL) style. Not many companies have a production automation system for building-block layout design, mainly because of the immaturity of the design tools. As an increasing number of full-custom ICs are being produced, there is an increasing demand for well-developed automation techniques for building-block layout to cope with the growing complexity of the integrated circuits and to shorten the design-turnaround time

[Sang87].

1.2 Layout design methodology

The integration density on silicon is ascending rapidly. VLSI chips containing a million devices are already a reality and there is no indication that this growth in complexity will reach its limit in the near future. Therefore, hierarchy becomes an essential ingredient in layout design. It provides a way to manage the complexity of the design and to coordinate the team work between the designers. In mapping the logic decomposition of the circuit into geometric layout two methodologies are commonly used, *top-down* and *bottom-up*.

Top-down layout design starts at an early stage of the design process by partitioning the chip surface into a number of functional blocks. Each block can in turn be partitioned into smaller subblocks. The size and shape of the functional blocks are estimated only. When more detailed design data becomes available, the layout is gradually refined. For instance, the shape and size of the functional blocks are determined more and more precisely and the I/O pin positions on these blocks are chosen to optimize quality in terms of overall area of the chip, wire density, delays on the critical paths, etc.. Also, the wiring areas are taken into account. Full geometrical design of the individual blocks is postponed until late design stages. This design process is also called the *stepwise-refinement* [Ginn84] of the layout. The method can be applied if flexibility in the cell shapes and pin positions is available, which is the case, for example, when standard cells are used to construct the building blocks, or software module generators with some flexibility are available.

The bottom-up design method uses a set of fixed cells to build the circuit. These cells are often predesigned library cells or blocks with very little flexibility in changing shapes and pin positions. As a chip is usually designed hierarchically, top-down and bottom-up design methodologies can also be

seen as the two ways the hierarchy (or the decomposition) tree of the circuit is traversed, either from top to bottom or from bottom to top. In practice, during the evolution of a design, the hierarchy tree is continuously traversed in both directions. Hence none of these two methods can be strictly followed. A combination is usually used [Colb82].

1.3 Divide-and-Conquer

Layout optimization is a very complex combinatorial optimization problem. Even the simpliest versions of the layout optimization problem are NP-complete or NP-hard [Sahn80] which implies that we cannot expect to find exact solutions in a time polynomial in the size of the problem. As a consequence most of the algorithms proposed for the optimal layout problem are heuristic.

Heuristic algorithms explore the solution space quickly but cannot guarantee to produce the optimum solution. In general, they restrict the size of the solution space to be examined. Solutions found are usually locally optimal solutions. In spite of the fact that a good heuristic algorithm may produce excellent solutions for a wide variety of problems, one does not have the comfort of the theorists in justifying the quality of the solutions claimed. Aside from questions arising from the computer implementation of the heuristic algorithms, where considerations involving data structure, space-time trade-off, portability, flexibility, modularity, etc. can justify the way of doing things, the soundness of the procedure can only be demonstrated through extensive experimentation on sample instances [Lin75].

An important universal principle for constructing heuristics to solve complex problems is the principle of *divide-and-conquer*. This principle may be applied in two ways, by dividing complex problems into a sequence of problems of a simpler nature for which efficient heuristics or even exact algorithms are available, or by dividing a large problem into problems of a

smaller size and then combining the solutions to the subproblems to provide the solution to the original problem.

Layout design is traditionally divided into the placement of the cells and the routing of the interconnections. As we will see, these two problems are further subdivided into more specialized and better understood subproblems. Most systems for building-block layout decompose the placement and routing problem in the following flow of subproblems. At first, a floorplan of the chip components is derived at the *floorplanning* step. In this step, the components of the chip may or may not be completely characterized in terms of area, power or speed. Next, the components of the layout are *placed* on the area that is assigned to the chip, possibly with some information about the area needed to complete the routing of the interconnections. The interconnections are routed in the area called the routing area that is not taken by the components. A global decision on the path in the routing area taken by each of the nets connecting the components is then made in the *global routing* step. To complete the detailed routing of the connections the routing area is subdivided into smaller routing regions in order to reduce the problem's size and complexity. These routing regions are physically routed one after another by a *detailed router*. The resulting pieces of the layout are then put together to constitute the total layout of the chip. If not all the nets can be routed in the available routing area, the placement of the components must be modified to make more room for the routing. These steps could be iterated several times to improve the resulting layout. For example, the routing of the chip can be used to evaluate the quality of the placement of the components and hence improvement on the placement could be made after a routing iteration.

1.4 Channel definition in building-block layout

The main contribution of this thesis is in the area of channel definition in the building-block layout-design process. We interpret channel definition in a broad context here. We consider it to be all the design steps between the

placement of the blocks and the detailed routing of the interconnections, for example, the global routing, the construction and ordering of channels. In building-block layout the cells differ in size and may be located in any position. Hence, in contrast to gate-array and standard-cell layout, the decomposition of the routing area into smaller routing regions, called *channels*, is often not unique. How the routing area can be decomposed depends, of course, also on the available detailed routing tools. Channel definition plays an important role in achieving a good final layout, because global decisions in planning the wiring flows on the chip are made at this stage. After this phase the routing problem is decomposed into problems of a smaller size which will be geometrically routed by the detailed router.

In our approach the channel definition phase consists of a number of steps. First, the routing area is divided into smaller rectangular routing regions. A global router decides through which regions the path of the interconnection nets will run, taking into account the total wire length, the chip area, the wiring congestion, etc.. Out of the small routing regions routing channels are defined. First, the topological relationships are established between the channels, and between the channels and the building blocks. Also an ordering of the channels is determined in which the channels will be physically routed. After this step the routing problem is more or less broken down into smaller routing problems in the channels. The interfaces between the channels are then defined and the exact shape of the channel boundary (which is usually not rectangular) is determined before a channel is sent to the detailed router.

Algorithms will be presented, both heuristic and exact, for various steps in the channel definition phase. A shortest path and a Steiner tree heuristic algorithm are proposed for the global routing of the nets. Special attention is paid to the power nets which have to be planarly routed. A polynomial algorithm is developed for the topological channel definition and ordering

problem. A novel idea is to define and order the channels after the global routing step. This idea is based on the philosophy of taking as many channel topologies as possible into consideration and of postponing the selection as long as possible. Furthermore, detailed placement optimization, channel density minimization and optimal net ordering techniques are presented.

1.5 An overview of the thesis

Since placement and routing are related problems, we first give some preliminaries on the floorplanning and placement problem in chapter II before the routing-related algorithms are presented. A special layout structure, the slicing structure and sliceable placements, which is a very important concept for the rest of the thesis, is outlined.

In chapter III a graph model is defined to represent the routing regions and methods to construct such a graph are described. Then, an algorithm is presented which defines and orders a set of routing channels constituted from the smaller routing regions. This algorithm takes the global routing data into account in order to optimize the wiring flow and to minimize the dead area. The complexity of the algorithm will be proved to be polynomial bounded.

Chapter IV is devoted to the global routing-related issues. Efficient algorithms on finding the shortest connection path of the nets are presented. Over- and through-the-cell routing capabilities are outlined. A separate algorithm is developed for the planar power-net routing. This algorithm guarantees a planar routing of the two power nets if one exists.

In chapter V the problem of the geometrical channel definition is studied. In this step the exact shape of the channel boundary is determined. The relative positions of the blocks adjacent to the channel to be routed are optimized in order to minimize the channel density. Although the channels are routed one at a time, the nets passing the channel intersections are synchronized by

indicating a preferred terminal ordering on the channel ends to prevent unnecessary wire crossovers.

Finally, in chapter VI a powerful placement and routing system for the building-block layout is highlighted. Many of the ideas presented in the thesis have been implemented in this system. Some design experiences with this system and the obtained results will also be presented.

1.6 References

Colb82. B. W. Colbry and J. Soukup, "Layout Aspects of the VLSI Microprocessor Design," *Proc. ISCAS*, pp. 1214-1228 (1982).

Ginn84. L.P.P.P. van Ginneken and R.H.J.M. Otten, "Stepwise Layout Refinement," *Proc. ICCAD*, pp. 30-36 (Oct. 1984).

Lin75. S. Lin, "Heuristic Programming as an Aid to Network Design," *Networks* **5** pp. 33-43 (1975).

Sahn80. S. Sahni and A. Bhatt, "The Complexity of Design Automation Problems," *Proc. 17th Design Automation Conference*, pp. 402-441 (1980).

Sang87. A. Sangiovanni-Vincentelli, "Automatic Layout of Integrated Circuits," pp. 113-193 in *Design Systems for VLSI Circuits, Logic Synthesis and Silicon Compilation*, ed. G. De Micheli, A. Sangiovanni-Vincentelli, P. Autognetti, Martinus Nyhoff Publishers (1987).

Souk81. J. Soukup, "Circuit Layout," *Proceedings of The IEEE* **69**(10) pp. 1281-1304 (October 1981).

Sout83. J. R. Southard, "MacPitts: An Approach to Silicon Compilation," *Computer*, pp. 74-82 (December 1983).

2. FLOORPLANNING AND PLACEMENT

2.1 The floorplanning and placement problem

Floorplanning and placement are related problems, both deal with the arrangement of the circuit components on the chip surface. Floorplanning is often considered as a generalization of the placement problem.

Placement is defined to be the task of assigning precise locations and orientations on the chip surface to the components of the design so that a number of goals are achieved, such as minimizing the chip area, minimizing the total wire length and minimizing the signal delays while sufficient routing area is allocated for the wiring. Layout of the components is assumed to be rectangular in shape. The positions on a block at which the component can be connected to the other parts of the circuitry, called terminals, are located on the periphery of the block. A net list specifying which terminals have to be interconnected is also given.

The simplest version of the placement problem, the two dimensional pin-packing problem, in which the connectivity between the rectangular blocks is ignored is already a NP-hard problem [Gare79]. Allowing non-rectangular blocks would make the problem even more complicated. The different goals of the placement are difficult to cast into a single objective function that can be handled by an algorithm. Usually a more restricted objective is used by placement algorithms. When a placement procedure derives a good placement as measured by the restricted objective, it is hoped that the placement is also good as measured against the actual goals.

With the same goals in mind floorplanning is one of the first stages in the design of the circuit. In this step, a designer partitions a large design into macro-modules and selects optimal relative positions, sizes, aspect rations and terminal positions of the modules. As pointed out in the previous chapter, information about the exact size, shape and terminal positions of the

building blocks may be unknown at the initial phase of floorplanning. For example, if a functional module is implemented with standard-cells, then the number of rows used for the module is not necessarily fixed. The use of many rows will produce a block that is tall and skinny; using a few rows will produce a block that is short and fat. If the module is implemented with a PLA, topological operations such as folding and partitioning can be used to vary the aspect ratio and terminal positions of the block. These flexibilities can be exploited at the floorplanning stage in order to obtain a better area utilization and circuit performance.

Thus, floorplanning is more general and less structured than the placement problem and is more difficult to solve. Although both aim at an optimal geometrical layout, floorplanning is more concerned with the topological than with the geometrical structure of the layout. As a consequence the aspects of floorplanning that have been investigated most are related to the determination of the relative positions of the blocks.

2.2 Slicing structure and sliceable placement

Placement algorithms generate a geometrical placement of the blocks in terms of absolute coordinates while floorplanning algorithms usually only derive a topological relative positions of the blocks, a floorplan. A floorplan can be represented by a *rectangular dissection* which is a geometrical configuration consisting of a finite number of non-overlapping rectangles that together completely cover an enveloping rectangle. Given a geometrical placement it is possible to construct an associated floorplan in the form of a rectangular dissection such that each of the rectangles contains one and only one of the blocks. A placement and an associated relative rectangular dissection is shown in Fig. 2-1.

A special case of a rectangular dissection is the *slicing structure* [Otte82] which is obtained by recursively dividing rectangles into smaller ones by

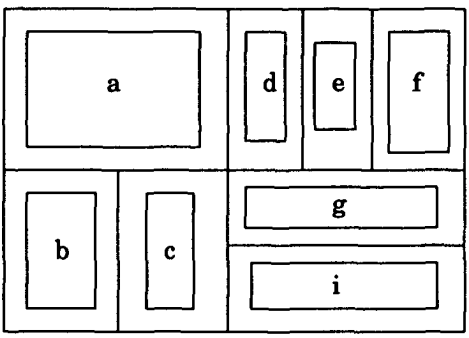


Figure 2-1. A placement and an associated rectangular dissection.

parallel lines, called *slice lines*. The floorplan of the placement in Fig. 2-1 is a slicing structure. A *slice* is either an undissected rectangle or a rectangle dissected into smaller rectangles by slice lines. The slices obtained by dissecting a slice over all its parallel slice lines are called the *child slices* or *subslices* of the dissected slice which is called the *parent slice* of the subslices.

Slicing structures are particularly interesting in building-block layout design, because as we will see, routing of these structures can be performed by the most effective tools available today for this task: the channel routers. This is due to the fact that conflict-free channel orders can be defined for a slicing structure. This issue will be fully explained in the next chapter. For now, a channel router is a tool which can route a region called a channel with fixed terminals on the upper and lower boundaries and floating terminals along the left and right sides.

Beside the possibility of using channel routers a slicing structure has a number of other interesting features. It can be represented by simpler data structures than more general topologies. The topology of a slicing structure can be represented by a tree, the *slicing tree*, with each slice represented by a node, and arcs from each node representing a parent to each node

representing one of its child slices, leaving the parent's node in accordance with the ordering of the child slices. Many optimization problems for slicing structures can be solved efficiently, whereas the corresponding problems for arbitrary rectangular dissections are NP-hard. Not only computationally is a slicing structure an attractive topology, it also offers more flexibility in achieving area efficient layouts. For instance, before a channel is routed, the channel densities can be minimized by lateral block shifting along the sides of the channel, because the absolute positions of the blocks can be adjusted during the detailed routing without destroying the slicing topology. Furthermore, it is also very suited for hierarchical layout design.

A placement can be derived in various ways, by a floorplanner, a placer or manually. To construct a routing system which is independent of the placement method we decided to use a geometrical placement of the blocks as the starting point of the routing process. For the reasons mentioned above, we restrict ourselves to the class of *sliceable placements*, placements which can be recursively bisected by straight lines. An associated slicing structure floorplan can be obtained for a sliceable placement. Note that in general there is no unique slicing structure for a given sliceable placement because the existence of crossing slice lines. For example, for the placement in Fig. 2-1 two slicing topologies exist, one uses the maximal horizontal line segment as the first slice line, the other uses the vertical one as the first slice line. Hence, a sliceable placement contains more information than a slicing structure. Beside the presence of multiple floorplan topologies in a placement expected wiring space can also be extracted from the placement.

2.3 Automatic floorplanning and placement

To achieve high quality layouts floorplanning and placement is a key problem. It is also regarded as one of the most difficult problem in automatic layout of integrated circuits. Due to the nonuniform size and shape of the building blocks row-oriented placement techniques for gate-array and standard-

cell layout are not applicable. In this section we will give a brief review of frequently used floorplanning and placement algorithms for building-block layout design. Some of them produce slicing structures or sliceable placements and some of them generate a general topology.

Min-cut algorithm. A well-known method is the min-cut algorithm. The basic procedure is based on the recursive application of a bi-partitioning algorithm [Fidu82, Kern70]. Lauther applied it first to the macro-cell placement problem [Laut79]. Basically, the min-cut algorithm partitions the set of blocks into two subsets by either a vertical or a horizontal line so that the interconnections between the two subsets is minimized and a predetermined area balance criterion is satisfied. This procedure is recursively applied to the two subsets until each subset contains one and only one block. Obviously, this method always produces a slicing topology.

An example of a recent application of the min-cut algorithm for floorplanning is the Mason system [LaPo86]. In this system a topological floorplan is obtained by a modified min-cut algorithm. The floorplan is then converted into a geometrical placement by determining the block shape and coordinates using an algorithm proposed by Otten [Otte83] which is based on the shape constraint relations. Routing area is also taken into account in this system.

Constructive method. This method is sequential in nature. A seed block (usually a large macro with heavy connection to others) is selected first. Recursively, a new block is chosen based on high connectivity to the already-selected blocks; it is then placed adjacent to the ensemble by applying some heuristic scheme of best fit. There are many variations to this method, in the selection of the seed and the selection of the new block [Prea79, Horn81, Prea78].

Force-directed method [Quin75]. The basic idea of this class of methods is to represent the interconnections between the blocks with a set of forces.

Attractive forces are defined between strongly connected blocks and repulsive forces between weakly connected blocks. The blocks are then moved to their equilibrium position to minimize the sum of forces acting on the blocks. Various heuristics can be used to remove/prevent overlaps in the resulting floorplan. Many layout systems employ a force-directed algorithm, e.g. [Woo86, Ueda85].

Simulated annealing. Simulated annealing has received much attention, mainly because of its ability to escape from local minima. It is a general optimization technique. When applied to combinatorial optimization it generates moves randomly. Moves that reduce the cost are called downhill, and those that increase the cost are called uphill. All downhill moves are accepted, uphill moves are accepted with a probability of $exp(-\frac{\Delta E}{T})$, where ΔE is the increase in cost and T is the temperature. The temperature is decreased by $T := \alpha T$ for some constant α, $0<\alpha<1$, until the stop criterion is reached. A certain number of moves are generated and checked before a decrease in temperature is allowed. The initial temperature, the number of moves generated at each temperature and the rate of decrease of temperature are all important parameters that affect the speed of the algorithm and the quality of the final result. This variety of parameters makes also that the behavior of the method is difficult to control.

Simulated annealing was introduced into layout by Kirkpatrick et al. [Kirk83]. The first application of simulated annealing to macro-cell placement was presented in [Jeps83]. The floorplanning problem has also been attempted using this method [Otte84, Wong86, Sech88].

Recently, Dai and Kuh proposed a combined floorplanning and global routing method [Dai87]. In this approach, modules are hierarchically clustered according to their size and connectivity in a first bottom-up phase. The number of components at each level of the hierarchy is fixed to a

maximum of five. In the second phase a top-down approach is followed, where the blocks in the cluster are placed and globally routed.

Many other methods have been experimented to tackle the floorplanning and placement problem, e.g. analytical methods [Sha85], dual graph algorithms [Hell82], etc. Several review papers are available that cover a large part of published results, e.g. [Souk81, Prea86, Sang87]. Some experimental results of comparisons of several algorithms can be found in [Cai88].

2.4 Manual floorplanning and placement

In spite of many efforts to find automatic floorplanning and placement techniques, the automatically produced floorplans and placements often have to face defeat against the manually produced ones. If the number of blocks is not large, which is usually the case in hierarchical designs, manual floorplanning/placement often produces superior results. A human being is able to consider different aspects of the two dimensional problem at the same time, for example the shape fitting of the blocks and the connectivity among the blocks. A human being is also fast in learning the mistakes and the floorplan/placement can be iteratively improved using feedback from the routing process.

For these reasons the floorplan and placement of a design is often derived manually or semi-automatically using a graphical editor, e.g. [Woo86, Anwa85]. In such systems an initial floorplan/placement is usually automatically generated which is then interactively improved by the designer. Computer assistance can be used in improving the layout. 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.

2.5 Wiring space

The area used by a particular layout depends on the area of the modules to be placed and the area needed to complete the interconnections. If the routing area is not considered, then the area used depends on how well the modules can be packed together. However, an optimal packing of the modules may result in an inferior layout after the routing area is added to make the interconnections of all nets possible, because in modern chips wiring may occupy 50% of the total chip area. Hence, it is very important to estimate the amount of routing area needed accurately and to determine where the routing area should be added. Failure to allocate the correct amount invariably force substantial placement alteration by the router and result in substantial total wire length increases as well as chip area increases. An accurate placement is one that closely matches with the placement of the blocks in the final layout after the routing.

As most floorplanning/placement algorithms do not have accurate routing area estimation procedures, the space allocated for the wiring prior to the routing itself usually differs from the wiring space actually required. The accuracy of the placement has great influence on the global routing accuracy, and what is more, on the routing channel definition process, because the channels are defined based to the geometrical placement of the building blocks. However, after one global routing iteration the global routing data can be used to estimate the routing area more accurately. If it is in the floorplanning phase one can still adjust the block shape and positions in conjunction with the required routing space [LaPo86]. Here, we propose an algorithm to adjust the wiring space in the placement phase, i.e. all blocks have fixed shape. The algorithm does not require a sliceable placement as input and does not guarantee a sliceable placement as output either.

Inspired by the constraint graph method in the layout compaction field [Cho85] we construct two *channel-position graphs,* one in each direction, to

represent the adjacencies of the wiring regions and the blocks. In the vertical channel-position graph each vertex represents the top side edge or the bottom side edge of a block and an edge represents a dimension, either a block or a wiring region between two blocks. Two additional vertices, s and d, are added to the graph as the source and destination vertices. The horizontal channel-position graph is defined similarly. An example of a pair of channel-position graphs is shown in Fig 2-2. Note that this is a more refined definition of the channel-position graphs compared to the one in [Prea78].

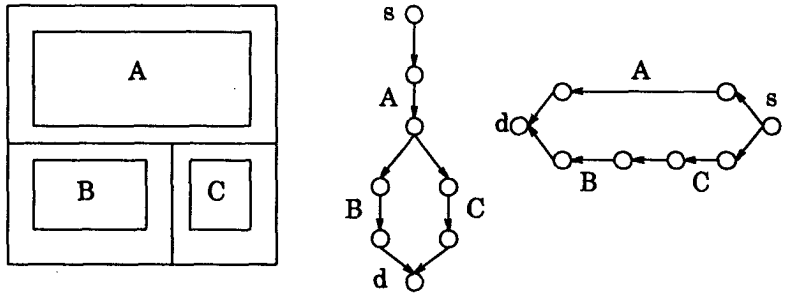


Figure 2-2. Channel-position graphs.

The algorithm repositions the blocks such that proper wiring space is provided. The blocks are moved in one direction at a time. Let us consider the vertical direction. First, the channel-position graph is derived from the block placement. All paths from the source vertex s to the destination vertex d in the channel-position graph are searched and sorted in the order of decreasing path length. The length of a path is the sum of the length of its edges which represent the size of the blocks and the size of the wiring regions. The size of the blocks is known. However, the size of the wiring regions needs be calculated. For this purpose the global routing routine is invoked to estimate the needed wiring space. How the global routing is carried out is explained in a later chapter.

The length of the longest path (also called the critical path) from 's' to 'd' is an estimate of the chip height H. If the blocks are repositioned such that the estimated chip height is realized, the routing regions on the paths shorter than H will contain some empty space or dead space. The algorithm uses an "average slack" method to distribute the total dead space uniformly among the wiring regions. To each wiring region a fraction of the total dead space is assigned, with a percentage proportional to its width. Consequently some channel segments will become wider than the minimum required width. As a wiring region can be a part of more than one path, the paths are processed in the order of decreasing path length. This ordering implies that more critical paths are processed first. After a wiring region is assigned a width it is "locked" against further change. If, in a path, some channel segments are locked, the remaining dead space on the path is assigned to the unlocked wiring regions. Finally, using the new wiring region size the blocks are moved to their longest distance positions from the destination vertex.

Compared to the case in which the blocks are moved directly to the longest path distance using the minimum wiring-region size an advantage of this method is that the dead space is distributed among the wiring regions evenly over the chip area instead of being concentrated in the upper and right chip boundaries. This results in a more balanced layout and provides the detailed router the best chance to utilize the routing area optimally. The procedure may be iterated in order to achieve a more accurate placement. The horizontal and vertical directions are processed in an alternative order.

Another application of the procedure is to evaluate the placement. At this stage of the design if one discovers that the placement is not optimal and can be improved one can directly go back to the placement phase without spending time in the detailed routing steps.

2.6 Transforming a placement to a sliceable placement

Since not every placement program is guaranteed to deliver a sliceable placement, we present a heuristic algorithm to transform a given placement to a sliceable one. The capability to adjust placements for sliceability enables design systems to use the best placement tools available, and still profit from the advantages that the slicing restraint entails. Of course, the adjustments should not defeat the virtues of the result of the placement algorithms.

A placement can be constructed either automatically or manually. Many placements are derived in a top-down manner by a floorplanning system. Some floorplanning and placement algorithms construct a slicing structure automatically, for example, the min-cut algorithm [Laut79], and some implementations of the simulated annealing method [Otte84, Wong86]. If the placement is derived manually, the designer usually does not keep the sliceability in mind while optimizing the placement. In this section we present a procedure which modifies a given placement into a sliceable placement with minor placement modifications.

The procedure can also be used to modify a placement, either sliceable or not, into a sliceable placement containing more slicing topologies by creating crossing slice lines. This is motivated by the following observation. Often, a placement can coincidentally exclude some slicing topologies. For example, Fig. 2-3(a) shows a placement with only one possible slicing topology. With a slight re-positioning of the top-left block crossing slice lines are created as shown in Fig 2-3(b). With this placement two slicing topologies are possible depending on the slicing direction.
In general it is advantageous to consider as many routing channel structures as possible in order to optimize the chip surface. The choice is left for the channel definition algorithm after the global routing. Note that if the original placement is a sliceable placement all original solutions are still present in the modified placement.

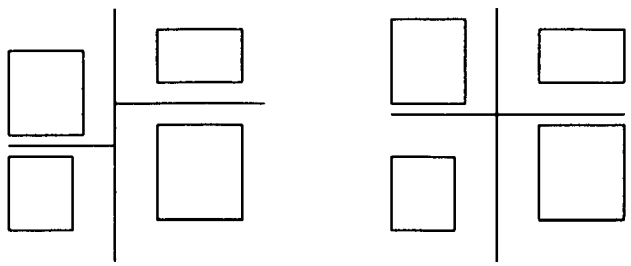


Figure 2-3. Creating slice line crossings.

To generate a sliceable placement from a given placement we adopted the shrink factor method originally proposed by Otten [Otte82a] to derive a slicing structure from a point configuration. This method was later modified by van Ginneken [Ginn87] to derive a slicing structure from a placement of blocks. The selection of slice lines is determined by shrink factors. Let (x_i,y_i) be the coordinates of the center of block i, with width w_i and height h_i. The shrink factors in the x direction s_x and in the y direction s_y of a pair of blocks n and m are defined as:

$$s_{x(m,n)} = \frac{2 \mid x_m - x_n \mid}{w_m + w_n} \qquad s_{y(m,n)} = \frac{2 \mid y_m - y_n \mid}{h_m + h_n}$$

A slice line that subdivides a slice into two sets of blocks M and N has a shrink value:

$$S_x(M,N) = \min_{m \in M, n \in N} s_x(m,n) \qquad S_y(M,N) = \min_{m \in M, n \in N} s_y(m,n)$$

If the slice can be subdivided by a slice line without intersecting any block, one of the slice lines has to possess a shrink value greater than or equal to 1, otherwise all slice lines will have a shrink value less than 1. The slice line with the largest shrink value should be selected for the next slicing operation. Van Ginneken [Ginn87] has proposed a procedure to calculate the shrink values of all slice lines in $O(n^2)$ time where n is the number of blocks in the slice.

Our method differs from the original one in that instead of selecting one slice line at a time we select multiple slice lines in both directions. The amount of placement distortion, and hence the number of slice line crossings is controlled by the minimum shrink value. The minimum shrink value is the minimum shrink value a slice line must have in order to be selected. Of course, at each slicing level at least one of the slice lines must be selected. This is the one with the largest shrink value, even if it has a shrink value less than the minimum shrink value. The minimum shrink value, S_{min}, can be defined by the user, for example, S_{min}=0.95. At each slicing level all slice lines in both directions with a shrink value not less than S_{min} are selected. In this way a number of crossings of slice lines are created and the slice is subdivided into more than two subsets, see for example Fig. 2-4. After the slice lines have been selected the process is repeated in all subsets containing more than three blocks.

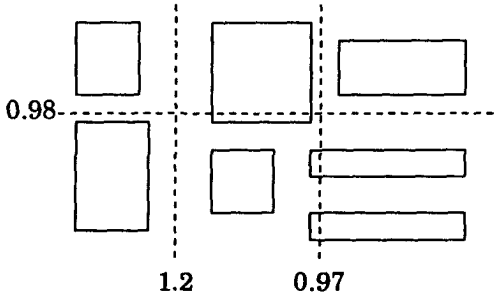


Figure 2-4. Selected slice lines and their shrink values.

After the slicing operation the blocks are bottom-up geometrically repositioned. This is done in such a way that the selected slice lines become free which means they do not intersect any blocks. Notice that if the original placement is a sliceable placement and S_{min} is set to 1, the resulting placement is equal to the original placement.

2.7 References

Anwa85. H. Anway, G. Farnham, and R. Reid, "PLINT Layout System for VLSI Chips," *Proc. 22nd Design Automation Conference*, pp. 449-452 (1985).

Cai88. H. Cai and J. J. A. Hegge, "Comparison of Floorplanning Algorithms for Full Custom ICs," *Proc. Custom Integrated Circuit Conference*, (May 1988).

Cho85. Y. E. Cho, "A Subjective Review of Compaction," *Proc. 22nd Design Automation Conference*, pp. 396-404 (1985).

Dai87. W. M. Dai and E. S. Kuh, "Simultaneous Floor Planning and Global Routing for Hierarchical Building-Block Layout," *IEEE trans. on CAD* 6(5) pp. 828-837 (September 1987).

Fidu82. C. M. Fiduccia and R. M. Mattheyses, "A Linear-Time Heuristic for Improving Network Partitions," *Proc. 19th Design Automation Conference*, pp. 175-181 (1982).

Gare79. M.R. Garey and D.S. Jonson, *Computers and Intractability - A guide to the Theory of NP-Completeness,* W.H. Freemann and Co. San Francisco (1979).

Ginn87. L.P.P.P. van Ginneken, "Gridless Routing for Generalized Cell Assemblies," *Submitted to IEEE Trans. on CAD*, (1987).

Hell82. W. R. Heller, G. Sorkin, and K. Maling , "The Planar Package For System Designers," *Proc. 19th Design Automation Conference*, (1982).

Horn81. C. S. Horng and M. Lie, "An automatic/interactive layout planning system for arbitrarily sized building blocks," *Proc. of 18th Design Automation Conference*, pp. 293-300 (1981).

Jeps83. D. W. Jepson and C. D. Gelett, "Macro Placement by Monte Calro Annealing," *Proc. of ICCD*, pp. 495-498 (1983).

Kern70. B. W. Kernighan and S. Lin, "An Efficient Heuristic Procedure for Partitioning Graphs," *The Bell System Technical Journal*, pp. 291-307 (February 1970).

Kirk83. S. Kirkpatrick, C.D. Gelett, and M.P. Vecchi, "Optimization by Simulated Annealing," *Science* **220**(4598) pp. 671-680 (May, 1983).

LaPo86. D. P. LaPotin and S. W. Director, "Mason: A Global Floorplanning Approach for VLSI Design," *477-489* **CAD-5**(4)(October 1986).

Laut79. U. Lauther, "A min-cut placement algorithm for general cell assemblies based on a graph representation," *Proc. 16th Design Automation Conference*, pp. 1-10 (1979).

Otte82a. R. H. J. M. Otten, "Automatic Floorplan Design," *19th Design Automation Conference*, pp. 261-267 (1982).

Otte83. R.H.J.M. Otten, "Efficient Floorplan Optimization," *Proc. ICCD*, pp. 499-501 (1983).

Otte84. R.H.J.M. Otten and L. van Ginneken, "Floor-plan Design using Simulated Annealing," *Proc. ICCAD*, pp. 96-98 (1984).

Otte82. R.H.J.M. Otten, "Layout Structure," *Proc. IEEE Large Scale Systems Symposium*, pp. 349-353 (Oct. 1982).

Prea86. B. Preas and P. Karger, "Automatic Placement: A Review of Current Techniques," *Proc. Design Automation Conference*, pp. 622-629 (1986).

Prea78. B. T. Preas and C. W. Gwyn, "Methods For Hierarchical Automatic Layout," *Proc. 15th Design Automation Conference*, pp. 206-212 (1978).

Prea79. B.T. Preas and W. M. vanCleemput, "Placement Algorithms for Arbitrarily Shaped Blocks," *Proc. 16th Design Automation Conference*, pp. 474-480 (1979).

Quin75. N. R. Quinn, "The Placement Problem as Viewed from the Physics of Classical Mechanics," *Proc. 12th Design Automation Conference*, pp. 173-187 (1975).

Sang87. A. Sangiovanni-Vincentelli, "Automatic Layout of Integrated Circuits," pp. 113-193 in *Design Systems for VLSI Circuits, Logic Synthesis and Silicon Compilation*, ed. G. De Micheli, A. Sangiovanni-Vincentelli, P. Autognetti, Martinus Nyhoff Publishers (1987).

Sech88. C. Sechen, "Chip-Planning, Placement, and Global Routing of Macro/Custom Cell Integrated Circuits Using Simulated Annealing," *Proc. 25th Design Automation Conference*, pp. 73-80 (1988).

Sha85. L. Sha and R. W. Dutton, "An Analytical Algorithm For Placement Of Arbitrarily Sized Rectangular Blocks," *Proc. 22nd Design Automation Conference*, pp. 602-608 (1985).

Souk81. J. Soukup, "Circuit Layout," *Proceedings of The IEEE* **69**(10) pp. 1281-1304 (October 1981).

Ueda85. 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) pp. 12-22 (January 1985).

Wong86. D. F. Wong and C. L. Liu, "A New Algorithm for Floorplanning Design," *Proc. of the 23rd DAC*, pp. 101-107 (1986).

Woo86. L. S. Woo, C. K. Wong, and D. T. Tang, "Pioneer: A macro-based floor-planning design system," *VLSI systems design*, pp. 32-43 (August 1986).

3. TOPOLOGICAL CHANNEL DEFINITION

3.1 Channel and channel order

Given a placement routing of the interconnections can be roughly divided into two steps, global routing and detailed routing. The global routing step determines the "topological" path of each interconnection net in the routing area and the detailed routing step generates the physical wiring. The primary objective of the routing is to achieve a 100% routing completion (i.e. all nets completely routed) in as small as possible an area. To achieve a 100% routing completion the detailed routing scheme should be capable of adjusting the block positions to provide the correct wiring space for the wiring regions, because the exactly required wiring space of a wiring region is unknown until it is routed. As we will show, using a channel router to do the detailed routing for a sliceable placement this placement adjustment during the detailed routing is feasible. In order to use a channel router the routing area must be partitioned into channels prior to the detailed routing. The process of partitioning the wiring regions into channels is called the *topological channel definition* or just the *channel definition*. The sequence in which the channels are routed is determined by a *channel ordering* procedure. The result of channel definition and ordering is a *channel structure* of the placement.

Channels are horizontal or vertical space strips between the building blocks. Three types of channel incidence, called *channel junctions*, can occur: the four 'L' type channel junctions at the corners of the layout, 'T' type channel junctions and '+' type channel junctions (also called *crossings* for short). An example of each is shown in Fig. 3-1.

The channels will be sequentially routed . They must be ordered in such a way that of the two channels meeting at a 'T' junction the channel represented by the stem of the 'T' is routed before the channel represented by

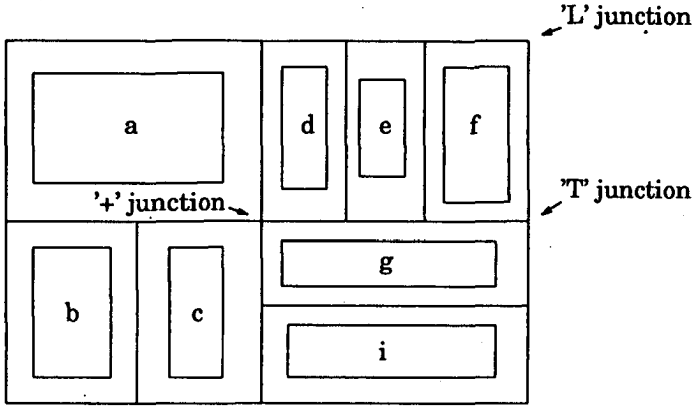


Figure 3-1. Types of channel junctions.

the bar of the 'T' is, for a channel router works with fixed terminal positions along its sides and floating terminal positions on its ends. After the stem channel of the 'T' is routed the terminals at the ends of the stem channel, which are also on the sides of the bar channel, will have fixed positions. Due to their inherent symmetry the other two types of channel junctions do not allow such a general statement about local sequence preferences. In the case of 'L' type channel junctions the choice hardly has an impact on the final routing result, and can therefore be based completely on more global sequencing criteria. The choice can be seen as converting the 'L' type junction into a 'T' type, reflecting the sequence selected. The case of a '+' type junction, however, is more intricate. Channel routers alone cannot handle such a configuration. Switch box routers and other two-dimensional routers have been resorted to. However if we split one of the two channels into two parts that form the stems of two 'T' type junctions the problem is transformed to channel routing problems. The unsplit channel is the bar for both new junctions. There are obviously two ways of converting a crossing into two 'T' type channel junctions. If the vertical channel is unsplit it is called a *vertical conversion* of the crossing, otherwise if the horizontal channel is unsplit it is

called a *horizontal conversion*, see Fig. 3-2.

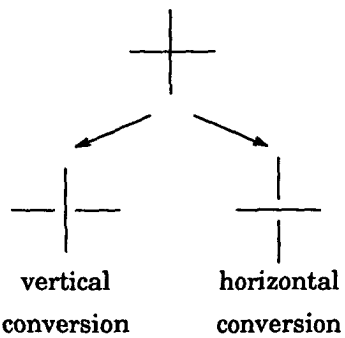


Figure 3-2. '+' type channel crossing conversion.

Once all junctions have been converted into "T" type junctions their local sequence preferences can be collected in a directed graph, the *channel digraph*. If this digraph does not have any cycles a simple topological ordering routine yields a feasible routing order for the channels. We speak of a *conflict-free* channel structure in that case. Whether cycles do occur may also depend, of course, on how crossings have been converted into pairs of "T" type junctions. It is fairly obvious that each conflict-free channel structure corresponds to a slicing structure of the placement and the channels correspond to the slice lines. Hence, for a sliceable placement there always exists a set of crossing conversions for the placement that results in a conflict-free channel structure.

Beside sliceability, there are more criteria to be taken into account when the channels are defined, e.g. the avoidance of bends in bundles of wires by keeping the main wiring flow as straight as possible, and the enhancement of the flexibility in adapting the relative positions of the building blocks. Criteria as these can be translated into local preferences for converting crossings, provided some information about the wiring flow, and the congestion to

be expected, is available. Such information can be generated by doing the global routing *before* the channel definition. This can be realized because the data structure on which the global router is run does not depend on the channel definition. On the basis of the information available after the global routing every crossing can get a preferred way of conversion. This preference can be weighted by assigning a number expressing the relative benefit or bonus of one way of converting the crossing over the other. Compared to other systems, doing the topological channel definition after the global routing is a unique feature of the approach proposed in this thesis.

Before the channel definition and ordering algorithm is presented a graph representation of the wiring regions is first introduced in section 3.2. In section 3.3 an algorithm is presented that selects the optimal slicing structure consistent with a given sliceable placement. Optimal means that the maximum sum of bonuses of crossing conversions is realized.

3.2 Graph representation of the wiring regions

3.2.1 The floorplan graph

The wiring regions must be represented by a model used as the data structure for the subsequent routing steps. An often used graph model to capture the incidences of the wiring regions in channel routing based systems is an undirected graph, the *floorplan graph*. Since in our approach the floorplan graph is derived from geometrical data, namely, the placement of the blocks, we need first introduce some geometrical concepts, the *tile planes* and the *walls* [Dai85]. The whole layout area is divided into rectangles referred to as *tiles*. Tiles which represent blocks are called *solid tiles* and tiles which represent routing space are called *space tiles*. We define two tile planes: the *horizontal tile plane* and the *vertical tile plane*.

Definition 3-1: The *horizontal tile plane* is a set of space tiles where all tiles are non-overlapping maximal horizontal strips. This means that no

space tile has other space tiles immediately to its right or left. The *vertical tile plane* is defined in a similar way. □

The horizontal and vertical tile plane is unique [Oust84]: there is one and only one decomposition of the space for each arrangement of solid tiles.

Definition 3-2: A space tile is called *dominant* if the interval of the tile includes the intervals of all its adjacent space tiles. Corresponding to each horizontal or vertical dominant tile a line segment can be drawn along the hart line of the tile, this line segment is called a *wall.* □

An example of a set of horizontal and vertical dominant tiles for a placement is shown in Fig. 3-3 (a) and (b) indicated by the shaded area. The corresponding walls are shown in (c). Note that the width of a dominant tile can be zero.

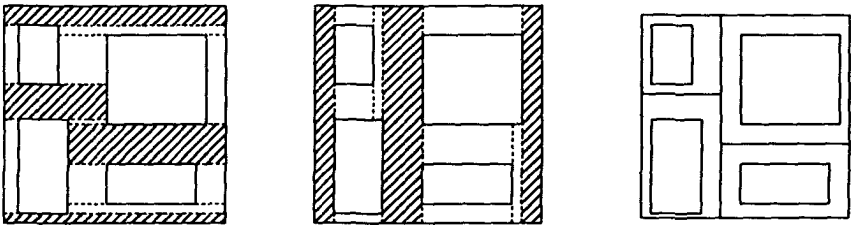


Figure 3-3. (a) horizontal dominant tiles (b) vertical dominant tiles, (c) walls.

A floorplan graph can be directly derived from the set of walls. Let us call a portion of a wall between two adjacent junction points of the set of horizontal and vertical walls a wall segment or a channel segment. The floorplan graph is defined as follows.

Definition 3-3: A *floorplan graph* is an undirected graph G = (V, E), where V represents junctions of channel segments and there is an edge (v_i,v_j) in E if the junction corresponding to v_i and the junction corresponding to v_j

are connected by a channel segment. Channel segments are thus represented by the edges in E. □

An example of a floorplan graph is shown in Fig. 3-4.

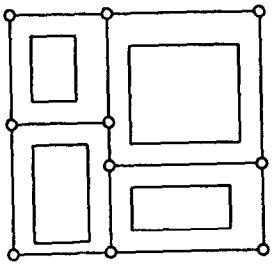


Figure 3-4. A floorplan graph.

A region bounded by channel segments but containing no channel segments is called a *room*. A room which does not contain a block is called an *empty room*. A floorplan graph may sometimes contain empty rooms, an example is shown in Fig. 3-5.

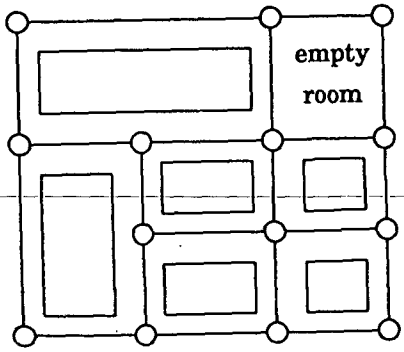


Figure 3-5. A floorplan graph containing an empty room.

For the purpose of channel definition we must derive an empty-room-free floorplan graph which is consistent with a slicing structure. Obviously, such

a floorplan graph exists only for sliceable placements.

3.2.2 Construction of a floorplan graph

3.2.2.1 Existing approaches

To construct a floorplan graph from a placement two different approaches have been reported. The first one is the hierarchical approach [Laut85], in which the set of blocks is recursively partitioned ("sliced") by straight slice lines until each subset consists of only a single block. The selection of the slice lines is on the basis of block deformation prediction. Such a procedure automatically yields a slicing structure. The set of slice lines together with the placement boundary lines constitute an empty-room-free floorplan graph by construction. The slice lines can be used as routing channels in the detailed routing phase, in which the inverse order of their generation is already a valid channel order. The second approach is the flat approach [Kimu83, Dai85]. In this approach the blocks are treated all at the same time. Line segments are extented from the boundaries of the blocks until they hit a block or the chip boundary. Then the number of line segments is reduced by merging overlapping line segments. The remaining horizontal and vertical line segments form the walls of the placement from which a floorplan graph can be derived.

The hierarchical approach is generally faster than the flat approach, and it yields a slicing structure with all the advantages of that topology. However, since it treats the two subsets divided by a slice line independently, it looses the flexibility that can be represented by crossings. Only "T" type channel junctions are produced in this approach. Also wiring flow considerations are not taken into account in constructing the channel structure. In contrast to the hierarchical approach, the floorplan graph constructed by the flat approach offers more flexibility in the channel definition process, but may contain empty rooms. The problem of eliminating empty rooms from the floorplan graph has been first pointed out by Dai et al. [Dai85]. However, the

method to eliminate empty rooms proposed in their paper does not always yield a correct solution [Cai89].

3.2.2.2 Floorplan graph construction algorithm

To construct an empty-room-free floorplan graph from a sliceable placement, an algorithm is proposed [Cai88] which combines the flat and the hierarchical approaches. First, a floorplan graph is generated using the flat approach according to [Kimu83]. Then possible empty rooms in the graph are eliminated by deleting edges adjacent to the empty rooms. For this purpose the placement is recursively bisected along the walls. This means that at each recursion level the blocks and the channel segments are partitioned into two subsets such that each contains at least one block. This process is continued until each subset contains exactly one block. Simultaneously, a *partitioning tree* is constructed. At each node of the tree the blocks and the channel segments inside the corresponding subset are recorded.

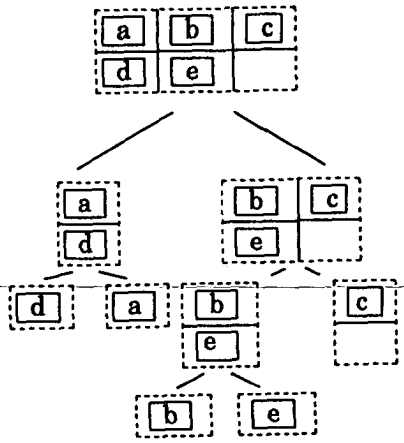


Figure 3-6. A partitioning tree.

An example of such a partitioning tree is shown in Fig. 3-6. The dashed lines represent the boundary of each subset. Note that the channel segments on

the boundary of a subset are not stored at the corresponding node, only those inside the subset are stored. After the tree is constructed the remaining channel segments in all leaf nodes are removed. The result is an empty-room-free floorplan graph. The floorplan graph construction algorithm based on the idea described is shown in Algorithm 3-1.

Algorithm 3-1: *Floorplan graph construction*

INPUT: a sliceable placement

OUTPUT: a floorplan graph.

METHOD:

/* derive the initial floorplan graph with the flat approach */
Step 1): Draw horizontal and vertical lines extending the edges of each block. If such a line intersects an edge of another block, then cut it at the intersection.

Step 2): If, for any two line segments thus drawn, the interval of one is wholly covered by that of another and there is no block between them, then merge them.

Step 3): Determine the intersection points of the two set of lines; keep only the line segments between two intersections.

/* remove empty rooms with the hierarchical approach */
Step 4): Initialize the root of the partitioning tree with all blocks and line segments.

Step 5): Partition (root).

Step 6): Generate the floorplan graph by assigning a vertex to each line intersection and edges between the adjacent vertices.

Procedure *Partition (S)*

if the number of blocks in S ≡ 1
{
Remove all remaining line segments in S.
}
else
{
Choose a slice line which partitions S [see 3.2.3.3].

Divide the blocks and line segments in S into S' and S".
(the line segments on the slice line are not taken)

Partition (S'), Partition (S").
}

3.2.2.3 Heuristic to choose a slice line

"Choose a slice line which partitions S" in the procedure *"Partition"* is crucial for the quality of the resulting floorplan graph. Generally, in the presence of empty rooms, different partitioning sequences result in different floorplan graphs. For instance, for the placement in Fig. 3-7 (a) two empty-room-free floorplan graphs are possible, Fig. 3-7 (b) and (c). Intuitively, the horizontal slice line should be chosen in the example which results in the floorplan graph shown in Fig. 3-7 (b), because of the large wiring flow on that slice line (represented by the dashed lines); otherwise it will result in wiring detours, Fig. 3-7 (c).

The following heuristic has been resorted to choose a slice line. As shown in the example the choice should depend on the wiring flows (traffic) in the channel segments. Therefore, the global router is called to derive the number of wires in each channel segment in the initial floorplan graph. A slice line usually consists of several channel segments. To each possible slice line a weight is assigned which is equal to the sum of the number of wires in its channel segments that are adjacent to an empty room. Channel segments that are not adjacent to an empty room do not contribute to the weight of the

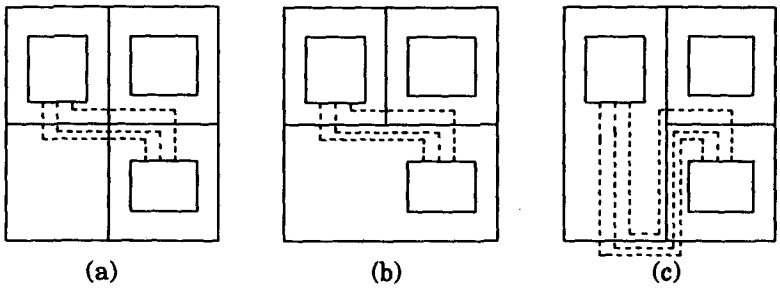


Figure 3-7. Empty-room-free floorplan graphs.

slice line. Moreover, wire segments which pass a channel segment get a higher weight than those which start or end in it, because passing wires imply that the channel segment is on the shortest path of those wires. Therefore deleting this channel segment would cause routing detours. The slice line with the highest weight is chosen.

Notice that bisection guarantees the existence of a conflict-free channel structure later on when the '+' channel crossings are converted. The channel order, however, is not yet determined.

3.2.3 Channel digraph

After the '+' channel crossings in the floorplan graph are converted into pairs of 'T' type junctions a channel structure is defined. The channel ordering can be captured in a so-called *channel digraph* [Flem78]. A channel digraph is a directed graph G=(V,E) where each vertex corresponds to a channel and there is an arc (v_i,v_j) for each 'T' type channel junction of which v_i is the stem channel and v_j is the bar channel. Figure 3-8 shows a channel structure and the corresponding channel digraph. From the definition of the channel digraph it follows that a cyclic channel order in the channel structure corresponds with a cycle in the channel digraph. Hence a conflict-free channel structure, which necessarily is a slicing structure, corresponds to a

acyclic channel digraph.

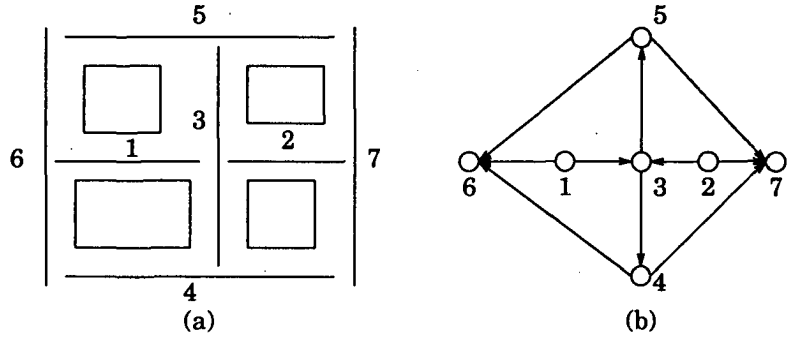


Figure 3-8. (a) a channel structure, (b) the corresponding channel digraph.

3.3 Selecting the optimal channel structure

3.3.1 Criteria for channel crossing conversions

Given a floorplan graph of a sliceable placement more than one channel structure may exist due to the presence of '+' type channel junctions. The final channel structure is determined by how the '+' type junctions are converted into 'T' type junctions. Unfortunately, not all combinations of crossing conversions yield a conflict-free channel structure, although the existence of at least one is guaranteed by the sliceability of the placement (see Fig. 3-9).

Beside the cyclic ordering constraint any local preference criteria can be specified to influence the direction in which a crossing will be converted. Two criteria are used in this thesis.

One is that the channels on the critical (i.e. the longest) paths in the channel-position graphs should be made as short as possible. In this way flexibility in adjusting the blocks and channels on the critical path is obtained. For example, a congested channel segment not on the critical path will in that case not widen its neighboring channel segment on the critical

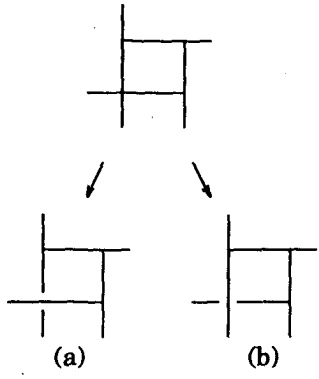


Figure 3-9. For the channel crossing in the figure the conversion in (a) is not allowed, because it results in a cyclic channel ordering.

path. We call this the *critical path isolation* criterion (see also Fig. 3-10).

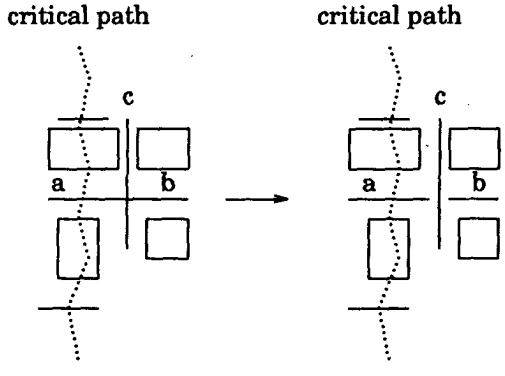


Figure 3-10. Critical path isolation criterion. The crossing in the figure prefers a vertical conversion, channel 'a' on the critical path is then separated from channel 'b' by the vertical channel 'c'.

To calculate the length of the critical path of the chip the size of both the blocks and the channel segments must be known. Since the global routing is already performed we have an estimate of the width of the channel segments.

Another criterion is that the decision should depend on the wiring flow across the '+' channel crossing area. It is preferable to use the direction with the largest wiring flow for the bar channels in order to minimize the number of wiring jogs and to avoid congestions in the channel crossing area. We call this the *major flow* criterion (see also Fig. 3-11).

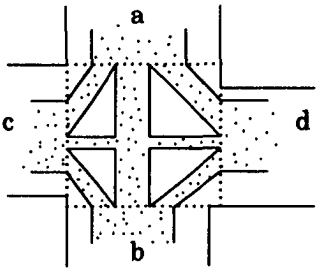


Figure 3-11. The major flow criterion. An example of the wiring flow in a '+' type channel crossing area. Width of the dotted area represents the wiring flow. In this example the crossing should be converted vertically, since there is a larger wiring flow between 'a' and 'b' (the major flow) than between 'c' and 'd'.

We assign a bonus which is a positive real number, to each of the two directions in which a '+' type crossing can be converted, indicating the preference of converting the crossing in this direction. It is called the *conversion bonus* of a crossing in that direction. Each criterion contributes a bonus w_i to the conversion bonus of the crossings. For a '+' type crossing adjacent to a channel segment on a critical path, a bonus is assigned to the conversion in the direction of the critical path (the critical path criterion). Furthermore, for each '+' type crossing a bonus, is added to the conversion in the largest wiring flow direction (the major flow criterion). To all other ways of crossing conversions a bonus 0 is assigned. We define the one with the highest total sum of crossing conversion bonuses realized among the conflict-free channel

structures to be the optimal channel structure.

3.3.2 A polynomial algorithm for channel crossing conversions

In this subsection we will present an algorithm that finds an optimal channel structure for a placement given a floorplan graph. In [Cai88] a straightforward algorithm was proposed in which all conflict-free combinations of channel crossing conversions are enumerated and then the best one is selected. Since each crossing can be split in two ways, the number of channel structures is 2^n where n is the number of '+' type channel crossings. Hence the worst case time complexity of that algorithm is $O(2^n)$. The algorithm presented in this subsection (see also [Cai89a]) has a polynomial bounded time and space complexity.

The channel definition problem can be naturally decomposed into a sequence of individual stage optimization problems. The problem is to bisect (i.e. slice) a placement recursively such that the total bonus of crossing conversions is maximized. At each stage a slice line must be selected. As a slice line divides a slice into two subslices, it partitions the original problem into two independent problems. The optimal solution of the problem for a slice can be obtained by searching for a slice line whose own contribution to the quality added to the optima of the two subslices is maximum.

Let $w(l)$ be the bonus of a slice line l which is defined as the sum of the bonuses of the crossing conversions obtained using this slice line. Corresponding to each slicing of a slice a bonus of the slice is defined which is equal to the sum of the bonuses of the slice lines on all levels of the slicing. We define $y_k(s)$ as the maximal bonus of a slice s at slicing level k from the root slice. Thus $y_k(s)$ is the quality of the optimal slicing of the slice s. Based on these definitions we can write the slicing-transformation equation for the system as:

$$y_k(s) = \max_{l_s} \left\{ w(l_s) + y_{k+1}(subs_1(s,l_s)) + y_{k+1}(subs_2(s,l_s)) \right\} \qquad (3.1)$$

where $subs_1(s,l_s)$ and $subs_2(s,l_s)$ represent the two subslices of s partitioned by the slice line l_s. This equation means that the maximal bonus of a slice is equal to the maximum of the sum of the following three items: the bonus of the slice line and the maximal bonus of the two subslices divided by the slice line. For each slice s $y_k(s)$ is to be evaluated for all possible candidate slice lines l_s of s. Notice that this approach is similar to the dynamic programming strategy [Bell57] in solving staged optimization problems, however, the stages (i.e. the slicing levels) are not serial in our case.

The algorithm can be implemented on a graph data structure, called a *slicing graph*. In a slicing graph each slice is represented by a vertex and all candidate slice lines of a slice are also represented by vertices; there is an arc from a slice vertex to each of its candidate slice-line vertices and there is an arc from each slice-line vertex to each of the two vertices representing the two subslices divided by the slice line. A simple example of such a slicing graph for a three block floorplan is shown in Fig. 3-12. To calculate the maximum bonus of each slice the algorithm will visit all arcs once and only once in a depth first order, starting from the root-slice vertex. After this step the channels can be obtained by, starting at the root slice, recursively collecting the slice line from which the maximum bonus of the slice is obtained and the slice lines of the two subslices.

Summarizing, the optimal channel structure selection algorithm is shown in Algorithm 3-2.

To derive the complexity of the algorithm we need first introduce two lemmas.

Lemma 3-1: Given n blocks the total number of possible rectangular slices is bounded by $O(n^2)$.

Proof: Since a floorplan graph $G=(V,E)$ is a planar graph, it possesses the following properties: $|V| - |E| + n = 1$ (i.e. the Euler's formula) and

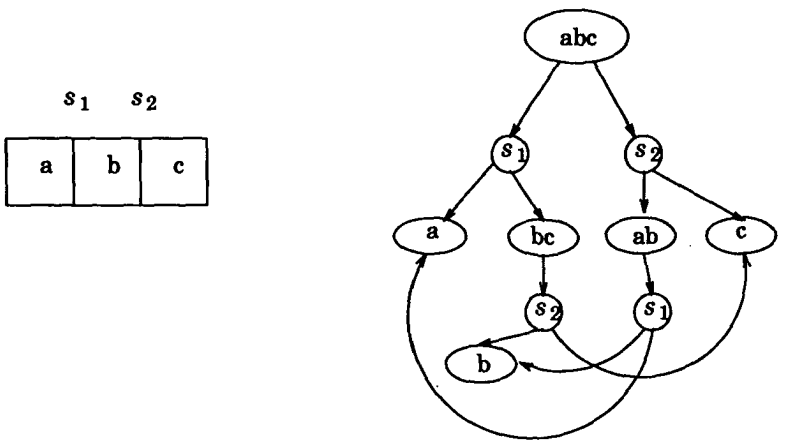


Figure 3-12. The slicing graph of a sample placement. Slices are represented by elliptic shaped vertices and slice lines are represented by cyclic shaped vertices.

Algorithm 3-2: *Optimal channel structure selection*

INPUT: a floorplan graph.

OUTPUT: a conflict-free channel structure.

METHOD:

Step 1): Assign conversion bonus to the channel crossings.

Step 2): Construct the slicing graph.

Step 3): Calculate the maximal bonus of the slice vertices according to equation (3.1) in a depth first order.

Step 4): Generate the channels by traversing the slicing graph from the root slice to recursively collect the slice line from which the maximum bonus of the slice is obtained and the slice lines of the two subslices.

$|E| \leq 3|V| - 6$ where $|V|$ is the number of vertices, $|E|$ is the

number of edges and n is the number of blocks in the floorplan. This gives $|V| \geq (n+5)/2$. Furthermore, we have $|V| \leq 2n$, because each of the n rooms in the floorplan has four corners and the minimum degree of a vertex in a floorplan graph is 2. Therefore, the number of vertices $|V|$ in a floorplan graph for n blocks is in the order of $O(n)$.

As a rectangle is completely determined by two diagonal corner points, a possible rectangular slice in a floorplan graph is determined by two vertices. Hence, the upper bound of the total number of possible rectangular slices in a floorplan graph with $|V|$ vertices is $\frac{1}{2}|V|(|V|-1)$ which is in the order of $O(n^2)$ where n is the number of blocks in the floorplan. □

Lemma 3-2: Given n blocks the total number of possible slice lines in all possible slices is bounded by $O(n^3)$.

Proof: Since in a floorplan of n blocks the number of slice lines in a slice cannot exceed $n-1$ and the total number of possible slices is bounded by $O(n^2)$ (see lemma 3-1), the total number of possible slice lines in all possible slices is therefore bounded by $O(n^3)$. □

Theorem 3-1: The time and space complexity of the optimal channel structure selection algorithm is bounded by $O(n^3)$ where n is the number of building blocks.

Proof: The number of vertices in the slicing graph is the sum of the number of all possible slices and the number of all possible slice lines in the slices which is bounded by $O(n^2)$ and $O(n^3)$ respectively according to lemma 3-1 and lemma 3-2. The number of arcs is three times the number of slice-line vertices. Hence the space complexity is bounded by $O(n^3)$.

The time complexity consists of two parts, one consumed by the procedure to construct the slicing graph, the other spend by the algorithm to calculate the maximal bonus of each slice vertex. To construct the slicing graph one

needs to find all possible slice lines for each slice. Given a slice all possible slice lines bisecting the slice can be found by a linear search on the floorplan graph which has a size of $O(n)$. For all possible slices which is bounded by $O(n^2)$ the total number of operations is then bounded by $O(n^3)$. Furthermore, a slice-line vertex must be connected to the vertices of the two subslices divided by the slice line. Each slice is represented only once in the slicing graph. The question whether the two subslice vertices are already present in the graph can be answered in a constant time by storing the slice vertices in a hash table using the coordinates of the slice as hashing parameters. The algorithm will visit all arcs once and only once, hence the time spend by the algorithm to calculate the maximal bonuses is bounded by $O(n^3)$ according to lemma 3-2. Consequently, the time complexity of the algorithm is bounded by $O(n^3)$. □

3.3.3 A locality property of the problem

Let us call the conversion direction of a crossing with the largest bonus the *preferred* direction of the crossing. If the bonuses in both directions are equal both directions are preferred. Suppose there exists a slice line which converts all crossings on the slice line in their preferred direction, then we claim that if we choose this slice line and optimally solve the channel definition problem in the two subslices independently we will obtain an optimal solution to the original problem. Such a slice line is called an *optimal slice line*.

This is supported by the following reasoning. Suppose we have an optimal channel structure which does not contain this optimal slice line. If we modify the channel structure such that the optimal slice line is used which implies that all crossings on this slice line will be converted in their preferred direction, we can show that the new channel structure is not worse than the original one. Firstly, a slice line corresponds to a vertex in the channel digraph with only incoming edges. Using the optimal slice line could never create a

cycle, because a vertex in a cycle must have both incoming and outgoing edges in the channel digraph. Secondly, because the crossings on the optimal slice line are converted in the preferred direction, the total bonus obtained could never be worse than the original channel structure. Hence, the new channel structure could never be worse than the original one.

Although this property does not reduce the worst case complexity of the algorithm, using it will greatly speed up the algorithm in most cases. At each stage of the algorithm whenever there is an optimal slice line at that stage, this slice line can be selected directly regardless of other slice lines.

3.3.4 An example

We will illustrate the algorithm by a simple example shown in Fig. 3-13.

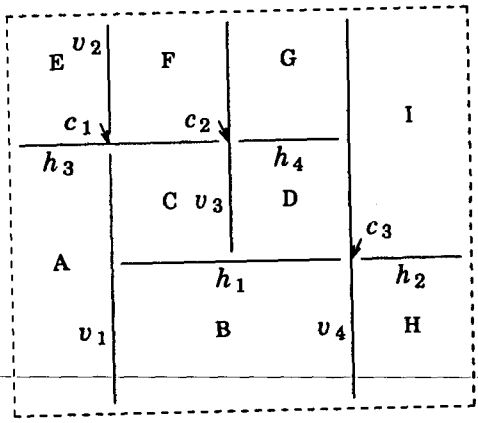


Figure 3-13. A channel definition and ordering example.

In the example the 9 building blocks are denoted by the letters A to I. The following bonuses of the three crossings c_1, c_2 and c_3 are given:

$$w(c_1^v) = 0 \quad w(c_1^h) = 2$$
$$w(c_2^v) = 1 \quad w(c_2^h) = 0$$
$$w(c_3^v) = 1 \quad w(c_3^h) = 0$$

in which c_1^v denotes the vertical conversion of c_1; and c_1^h denotes the horizontal conversion of c_1. The crossings are initially converted in their preferred direction, as shown in Fig. 3-13.

Let us denote a slice by the set of blocks in the slice, for example, *ABCDEFGHI* denotes the root level slice. At the root level the slice line v_4 is an optimal slice line.
Hence,

$$y_1(ABCDEFGHI) = w(v_4) + y_2(ABCDEFG) + y_2(HI).$$

If a slice does not contain any '+' type channel crossing its maximal bonus is 0. Hence,

$$y_2(HI) = 0$$

On the left side of v_4 there is no optimal slice line, thus the two non-optimal slice lines $v_1 \cup v_2$ and $h_3 \cup h_4$ must be considered. Hence,

$$y_2(ABCDEFG) = \max\begin{Bmatrix} w(v_1 \cup v_2) + y_3(AE) + y_3(BCDFG) \\ w(h_3 \cup h_4) + y_3(EFG) + y_3(ABCD) \end{Bmatrix}$$

where

$$y_3(AE) = y_3(EFG) = y_3(ABCD) = 0.$$

Since h_1 is an optimal slice line in *BCDFG*

$$y_3(BCDFG) = w(h_1) + y_4(B) + y_4(CDFG)$$

in which

$$y_4(B) = 0.$$

And finally using the optimal slice line v_3 in *CDFG*

$$y_4(CDFG) = w(v_3) + y_5(FC) + y_5(DG) = w(v_3).$$

Evaluating the equations numerically we obtain the following results. First the bonuses of the slice lines are

$w(v_4) = w(c_3^v) = 1,$

$w(v_1 \cup v_2) = w(c_1^v) = 0,$

$w(h_3 \cup h_4) = w(c_1^h) + w(c_2^h) = 2,$

$w(h_1) = 0$ and

$w(v_3) = w(c_2^v) = 1.$

Hence,

$$y_4(CDFG) = 1$$

$$y_3(BCDFG) = 0 + 0 + 1 = 1$$

$$y_2(ABCDEFG) = \max\begin{Bmatrix} 0+0+1 \\ 2+0+0 \end{Bmatrix} = 2$$

in which the slice line $h_3 \cup h_4$ is the chosen slice line for the slice *ABCDEFG*. Finally,

$$y_1(ABCDEFGHI) = 1 + 2 + 0 = 3$$

Thus, in the resulting channel structure the crossings are converted in the following way, c_1^h, c_2^h and c_3^v. The resulting optimal channel structure is shown in Fig. 3-14.

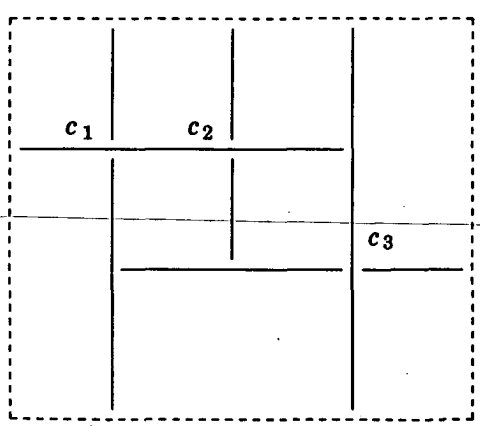


Figure 3-14. Resulting channel structure of the example.

3.3.5 Concluding remarks

Experiments have shown that the quality of the final routing result is strongly dependent on the choice of the channel structure. The channel structure derived with our approach tends to ease the wiring at the detailed routing stage, because wiring information is used to guide the decision in which direction a channel crossing should be converted. Preliminary results on several test chips have shown that using our algorithm an average improvement of 2.5% on the chip size and the net length has been achieved compared to a randomly selected conflict-free channel structure.

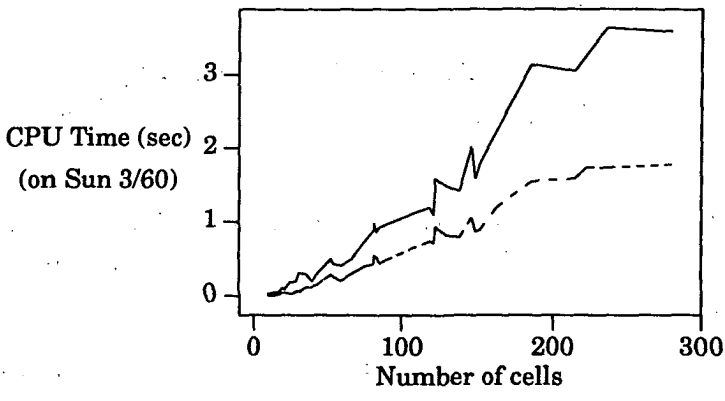


Figure 3-15. Run-time statistics of the algorithm. The dashed curve is the run time when the locality property of the problem is used. The solid curve is the run time when this property is not used.

Although we have shown that the worst case complexity of the algorithm is bounded by $O(n^3)$, in practice, the average complexity is only slightly more than linear. Figure 3-15 shows the run-time statistics of the algorithm for 35 different placements with the number of blocks ranging from 10 to 280. A placement is obtained by first generating a row and column based arrangement of square shaped blocks including some additional ones and then

randomly merging some neighboring blocks into larger blocks. Experiments have shown that by applying the locality property 50% saving of the run time can be achieved in most cases.

3.4 References

Bell57. R. Bellman, *Dynamic Programming,* Princeton University Press, Princeton, New Jersey (1957).

Cai88. H. Cai, "Connectivity Biased Channel Construction and Ordering for Building-Block Layout," *Proc. 25th Design Automation Conference*, pp. 560-565 (June 1988).

Cai89. H. Cai, "On Empty Rooms in Floor Plan Graphs: Comments on a Deficiency in Two Papers," *To appear in IEEE Transactions on CAD*, (1989).

Cai89a. H. Cai and R.H.J.M. Otten, "Conflict-Free Channel Definition in Building-Block Layout," *To appear in IEEE Transactions on CAD*, (1989).

Dai85. W. M. Dai, T. Asano, and E. S. Kuh, "Routing Region Definition and Ordering Scheme for Building-Block Layout," *IEEE Transactions on Computer-Aided Design* **CAD-4**(3) pp. 189-197 (July 1985).

Flem78. U. Flemming, "Representation and generation of rectangular dissections," *Proc. 15th Design Automation Conference*, pp. 138-144 (1978).

Kimu83. S. Kimura, N. Kubo, T. Chiba, and I. Nishioka, "An Automatic Routing Scheme for General Cell LSI," *IEEE Transactions on Computer-Aided Design* **CAD-2**(4) pp. 285-292 (October 1983).

Laut85. U. Lauther, "Channel Routing in a General Cell Environment," *Proc. VLSI*, Tokio, Japan, pp. 389-398 (1985).

Oust84. J. K. Ousterhout, "Corner Stitching: A Data-Structuring Technique for VLSI Layout Tools," *IEEE trans. on CAD* **CAD-3**(1) pp. 87-100 (January 1984).

4. GLOBAL ROUTING

A global route of a net is a "topological" path in the routing regions and, if allowed, over and through the blocks connecting all terminals of the net. A topological path of a net specifies only the relative positions of the path with respect to other objects in the layout, not the detailed geometrical layout of the net. Global routing is the process of finding a global route for each net of the circuit such that a number of objectives are achieved. Commonly used objectives are the minimization of the chip area and the total net length. However, for the two power nets a special topology is required, because usually they must be routed in a single metal layer, these nets may therefore not cross each other.

In this chapter subsequent steps in global routing will be discussed. First, the data structure on which the global router is run will be presented. Then, algorithms are outlined in section 4.2 in finding a global route for a net. In section 4.3, a scheme in which the nets are routed is described. In section 4.4, routing over and through the blocks is handled. Finally, in section 4.5 an algorithm will be presented for the planar power-net routing.

4.1 Implementation of the floorplan graph

A variation of the floorplan graph defined in the previous chapter is implemented for the purpose of global routing. Each edge in the original floorplan graph is replaced by two edges and a vertex between them representing a channel segment. An example of such a graph is given in Fig. 4-1.

This modification is done for several practical reasons: (1) usually, to route a net globally on the original floorplan graph temporary vertices must be added to the graph to represent the terminals of the net; with this implementation the graph modification is not needed; (2) the data structure representing the channel segment vertices can be shared by the channel-position graphs; this leads to a fast calculation of the critical (longest) path in

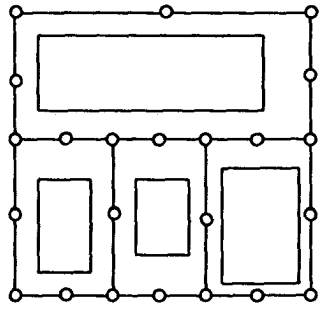


Figure 4-1. A floorplan graph for the global routing.

the channel-position graphs; (3) as we will show in a later section, this graph can easily be extented to incorporate the ability of routing through and over the cells.

4.2 Shortest connection algorithms

The problem of finding a global route for a net is reduced to finding the shortest connection path or the minimum cost connection path on the floorplan graph. Two cases are distinguished, namely, finding the shortest path between two terminals and finding the shortest connection tree connecting more than two terminals.

4.2.1 Background

The best known algorithm for searching a shortest connection path connecting two points in a plane with obstacles is due to Moore [Moor59] and Lee [Lee61]. This algorithm finds a path on a maze by expanding a search wave from one point to be connected until the other point to be connected is reached. Many variations and generalizations of the Lee-Moore algorithm have been proposed [Hoel76, Rubi74, Souk81, Xion81]. It has found wide application in PCB and IC routing systems, e.g. [Cies82, Roth83, Pers84]. Hightower [High69] proposed another algorithm using line segments as the

basic unit instead of a grid cell. This algorithm is much faster than the Lee-Moore algorithm, however, it cannot guarantee a solution even if one exists. In general graphs with non-negative edge lengths the Dijkstra's algorithm [Dijk59] is widely used to search the shortest path between two vertices on the graph, e.g. [Hass82, Rive82, Syed82, Wisn84, Prea85].

The problem of finding the minimum connection tree for more than two points in a plane is commonly referred to as the *Steiner problem* and the minimum connection tree is called the *Steiner tree* in graph theory. The Steiner problem in general graphs is defined as follows:

Definition 4-1: *Steiner problem in graphs:* Given a weighted connected graph $G(V,E)$ and a set of vertices P, $P \subseteq V$, find, among all subgraphs in $G(V,E)$, the subgraph g with the least total weight so that P is connected in g. □

The Steiner problem in graphs is a generalization of the *minimum spanning tree* problem which is the case where $P \equiv V$. Although the minimum spanning tree problem can be solved in polynomial time, the Steiner problem in graphs is a NP-complete problem [Even79]. Dreyfus and Wagner have given an exact algorithm [Drey72] which has a complexity exponential in $|P|$ ($O(|V|3^{|P|} + |V|^2 2^{|P|})$). Hence, we must rely on heuristic algorithms to tackle the problem in a realistic way.

4.2.2 Two terminal nets

Suppose we have a source terminal, t_s, to be connected to a destination terminal, t_d. In general, we wish to find a minimum cost path, where we will assume cost to be the length of the path. Given a floorplan graph, we assign a cost to the vertices. Vertices corresponding to a channel segment are assigned the length of the channel segments, other vertices are assigned a zero cost. The cost of a path is simply the sum of the cost of the vertices along the path. Later, we will show how other factors can be considered

when calculating the cost of a path, but for now, we will assume that we are trying to minimize the wire length. Terminals are positioned on the boundary of the blocks. A terminal is linked to the vertex corresponding to the channel segment to which it faces. In order to calculate the accurate length of the path the distances from this vertex to the neighboring vertices are adjusted according to the position of the linked terminal.

Our shortest path algorithm [Cai85, Cai86] is based on the Dijkstra's algorithm. However, akin to many implementations of the Lee-Moore algorithm a list, F, is used containing all vertices which are on the frontier of the search wave. They are the only vertices from which the search can expand. Furthermore, a list, P, is used containing all vertices for which the minimum cost path to t_s is already found, these vertices are called the *permanent* vertices.

Initially, the vertex linked to terminal t_s is placed in F. A search proceeds by taking a vertex, u_c off the F list, finding its successors, adding the successors to the F list, and then place u_c on the P list. u_c is chosen to be the vertex in F with the minimum cost. Successors of u_c are all its neighboring vertices in $(V-P)$ which can be quickly found, because an *incidence list* data structure is used for the floorplan graph. The cost of the successors will be recalculated. If the old cost of a successor is larger than the new cost expanded from u_c, the new cost will be assigned to the successor. The search process terminates if the selected vertex u_i is the vertex linked to the destination terminal, t_d. In the implementation it is important to keep pointers from each successor back to its parent vertex. These pointers provide the means for following back the path to the source vertex once the search has terminated by finding the destination vertex. The procedure form of the algorithm is given in Algorithm 4-1.

The time complexity of the original Dijkstra's algorithms is $O(|V|^2)$ where $|V|$ is the size of V, if it is applied to a completely connected graph. Since the floorplan graph is sparse, (maximum *degree* of a vertex is four),

Algorithm 4-1: *Shortest path algorithm*

INPUT: G(V,E), t_s and t_d

OUTPUT: a path connecting t_s and t_d

METHOD:

Let $l(u_i)$ be the cost from u_i to t_s.
Let $c(u_i)$ be the cost of vertices u_i.
Let F be the list of frontier vertices.
Let P be the list of permanent vertices.
Let $u(t_i)$ be the vertex linked to terminal t_i.

Step 1): $l(u(t_s)) = 0$, for all other vertices u_i, $l(u_i) = \infty$.
$F = \{u(t_s)\}$.

Step 2): For all u_i in F find u_c for which $l(u_c) = \min[l(u_i)]$.
$F = F - \{u_c\}$, $P = P + \{u_c\}$.
If ($u_c \equiv u(t_d)$)
Trace path, stop.

Step 3): For all neighbors u_i of u_c in (V - P)
$l(u_i) = \min(l(u_i), l(u_c) + c(u_i))$.
$F = F \cup \{u_i\}$.
Goto step 2.

$| F |$ is much smaller than $| V |$. As the operations in step 2 are only carried out for vertices in F and the operations in step 3 are only done for maximally three neighboring vertices of u_c, the time complexity of our implementation is much smaller. Experimental results on test circuits indicate that the average complexity is actually around $O(| V |)$.

4.2.3 Multi-terminal nets

For nets connecting more than two terminals a connection tree must be constructed. We propose the following Steiner tree heuristic. First, the center of gravity point of the terminals to be connected is determined whose

coordinates are the average x and y coordinates among the terminals. A seed terminal is selected which is the terminal with the shortest Manhattan distance to the center of gravity point. Initially, the partial connection tree is set to the seed terminal and the coordinates of the center of gravity point are also set to the coordinates of the seed terminal. The other terminals are then ordered in increasing Manhattan distance to the center of gravity point and are connected one after another to the partial tree using the shortest path algorithm. The procedure form of the Steiner tree heuristic to connect a multi-terminal net, n, is shown in Algorithm 4-2.

Algorithm 4-2: *Steiner tree algorithm*

INPUT: G(V,E) and a set of terminals T_n to be connected.

OUTPUT: a connection tree R connecting the terminals

METHOD:

Let C_n be the center of gravity point of T_n.
Let $d(t_i)$ be the Manhattan distance between terminal t_i and C_n
Let R be the partial connection tree.

Step 1): Determine C_n.

Step 2): For all t_i in T_n find t_c for which $d(t_c) = \min[d(t_i)]$.
$R = \{t_c\}$.
$T_n = T_n - \{t_c\}$.
Set the coordinates of C_n to the coordinates of t_c.

Step 3): For all t_i in T_n find t_j for which $d(t_j) = \min[d(t_i)]$
Connect t_j to R using the shortest path algorithm.
$R = R \cup$ {the path found}
$T_n = T_n - \{t_j\}$.

Step 4): if T_n is not empty
goto step 3.

Note that the shortest path algorithm that finds the shortest path between two terminals is used to connect a terminal to a partial connection tree. The only modification to the algorithm is to terminate the search if any vertex belonging to the partial tree is taken off the F list. We have experimentally compared our terminal ordering scheme to other alternative terminal ordering schemes. The ordering schemes compared are:

scheme 1: Algorithm 4-2.

scheme 2: Randomly select a seed terminal; the next terminal is the one which is nearest to the partial connection tree [Prim57].

scheme 3: First, two terminals with the shortest Manhattan distance are connected; then, the terminal which is nearest to the partial connection tree is selected for connection [Kimu83, Roth83].

scheme 4: Similar to scheme 3, however, the two terminals with the longest Manhattan distance are connected first [Kurt85].

scheme 5: The terminals are sorted from left to right in increasing value of the x coordinate and are connected in this order [Hana65, Fowl85].

We have tested on 234 multi-terminal nets in 4 different circuits. The experimental results are shown in Table 4-1. In most cases our method has produced the best results.

4.2.4 Remarks

Further optimization of the algorithms described are still possible. For speed optimization, the size of the search tree can be pruned by giving the vertices in the direction towards the destination terminal a higher priority to expand. For example, in Clow's algorithm [Clow84] the expanded corners are given a cost value based on the summation of the distance from the source terminal and the Manhattan distance to the destination terminal. It is proved that by doing so, many unnecessary searches can be eliminated.

method	total net length			
	chip1 no. nets: 134	chip2 no. nets: 27	chip3 no. nets: 40	chip4 no. nets: 33
scheme 1	11532	3304	3980	2378
scheme 2	11735	3307	4025	2399
scheme 3	11685	3271	3987	2448
scheme 4	12640	3373	4162	2684
scheme 5	11942	3491	4320	2447

Table 4-1. Experimental results with different terminal ordering schemes.

Xiong [Xion86] added the idea of wave propagation and diffraction to the search algorithm allowing the search tree to be further pruned by lowering the priorities of the corners whose wave fronts are diffracted by some blocks. In constructing Steiner trees, instead of treating the terminals independently, Hsu [Hsu87] has shown that better results can be achieved by modeling the yet unconnected terminals as small magnets. These magnets exert small attractive forces on the vertices in expansion by means of modifying the cost function. In this way the search process is biased towards these unconnected terminals.

4.3 Multi-net routing scheme

A moderate VLSI circuit contains hundreds of nets. It is obvious that connecting each net with the shortest path does not necessarily imply that the global result will be optimal. This is because that some wiring regions may become congested due to the limited number of available wiring layers resulting in chip area increases. To obtain a good global result the individual nets must somehow be correlated with each other during the sequential routing

process. Obviously, this correlation can only be based on estimations at the global routing phase, because the detailed routing is not carried out yet. In general, wiring congestions should be avoided in critical parts of the circuits. It is desirable to distribute the wiring smoothly throughout the layout. This can be achieved by incorporating penalties in the critical regions of the circuit into the global routing cost function. However, by doing so the order in which the nets are routed becomes important, because a routed net will influence the costs of the remaining nets in passing through certain routing regions.

4.3.1 Net ordering scheme

The nets are routed one at a time according to a certain order. Among many net ordering schemes, the "simplest net first" scheme has been proven to be effective in the practical environments. This ordering is defined as follows. Let N_i be the number of terminals in the net i, and T_i be the total number of terminals positioned inside the smallest rectangle which covers all terminals in net i. Then, the net to be routed is selected in the increasing order of $aN_i + bT_i$ where a and b are parameters. The idea behind this ordering scheme is that nets with many terminals and which interfere most with other nets are put off until the end, because they have more alternative short connections compared to the other nets.

4.3.2 Cost function and re-routing scheme

A cost function for using a channel segment i incorporating penalties and minimum length constraints is set up as follows:

$$C_i = d_i + \alpha p_i$$

where d_i is the length of the channel segment, α is a constant and p_i is the penalty function for using this channel segment. To avoid wiring congestion the penalty function must depend on the degree of the occupation of the channel segment. Therefore we must know the capacity of the channel segment which describes the effective number of available wiring tracks of the channel

segment. The following penalty function is used in our system:

$$p_i = \begin{cases} 0 & \text{if } c_i \geq g_i \\ (g_i / c_i)^n & \text{otherwise} \end{cases}$$

where c_i is the capacity of channel segment i, g_i is the congestion value of channel segment i and n is a constant not less than 1. The congestion value of a channel segment is initialized at 0 and increased by 1 each time a net is routed through this channel segment. The penalty function can be interpretered as the following. In channel segments where the usage does not exceed the capacity wires are routed based on the shortest distance. However, when the usage of a channel segment exceeds the capacity of that channel segment, the channel segment will exert a resistance to the coming nets forcing them to choose alternative routes.

The above "one-run" routing scheme works well if the placement is reasonably accurate, which means that the routing space (channel capacity) allocated matches closely to the actually required routing space. If this is not the case the iterative re-routing strategy [Kurt85, Fuku87, Lins84] might work better. In this scheme each net may be tried several times. In the first pass all nets are routed based on the shortest connection which provides a reasonable background of the congestion in the routing area. After this pass the critical channel segments are discovered which are the channel segments on the longest (critical) path in the channel-position graphs. Increasing the width of these channel segments will usually increase the chip size. Penalties are used in these critical channel segments, for example by adding an extra cost to the critical channel segments. Some or all nets in the critical regions are then selected for re-routing. After each iteration the critical regions and their penalties are updated again.

The first scheme is fast and is suitable for accurate placements. The iterative scheme is more general and compensates the drawback of the sequential routing of the nets.

4.4 Routing over and through the cells

Saving of the wiring space is achieved when, instead of going around the cells (blocks) some nets are routed through or over the cells, see for an example Fig. 4-2.

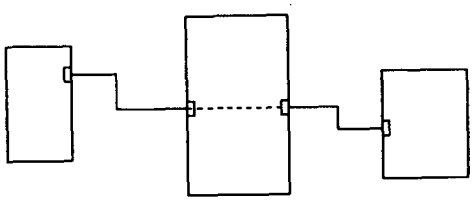


Figure 4-2. Routing through the cells

Since in the channel based routing systems the cell area is actually prohibited for routing, only limited capability of over- and through-the-cell routing can be provided. In our approach it is assumed that the connections over or through the cells are pre-specified in terms of internal connections in the cells. Two types of internal connections are distinguished. In the first case, a particular net can often be connected to a block at more than one location. Thus, there could be many terminals on a block which are internally connected and hence electrically common. It would be advantageous to let the router decide which terminal it prefers to connect. Wiring one of these terminals would wire all, and the net could continue out the other terminals, if necessary. Notice that a set of internally connected terminals of this type belongs to a particular net. In the second case the internally connected terminals do not belong to a particular net and can be freely used by any net. These internal connections are not connected to a gate in the cell. This is the case when some blocks have wiring paths available over the cells, e.g., in multi-metal layer technology where a designer may put some wires of an unused metal layer on top of the cells. The global router will decide if it is advantageous to use this type of internal connections during the routing of

the nets in order to shorten the wiring length. The first type internal connections are called *feedthroughs* and the second type internal connections are called *wirethroughs*[Elme84]. The shortest path algorithm described in section 4.2 has been extended to handle feedthroughs and wirethroughs.

4.4.1 Feedthroughs

The shortest path algorithm that finds a shortest path between two terminals is modified to find a path between two groups of internally connected terminals. The path will connect one terminal from each group. The frontier vertex list F is initialized to the vertices linked to the internally connected source terminals. The search process will terminate when one of the internally connected destination terminal vertices is taken off the F list. During the path tracing, beside the two vertices connected by the path all the other vertices which are internally connected to these vertices by a feedthrough are taken into the (partial) tree. In this way the net may continue out the other terminals during the routing of the remaining unconnected terminals of the net.

4.4.2 Wirethroughs

Routing with wirethroughs is more complicated, because a free wirethrough is available to all the nets while a feedthrough belongs to a particular net. For the purpose of routing efficiency the floorplan graph is modified accordingly to model the wirethroughs. We restrict ourselves to wirethroughs with two terminals on the block.

Normally each channel segment and channel intersection is represented by a vertex in the floorplan graph. If wirethroughs are used the graph is enhanced with wirethrough vertices connected to the channel-segment vertices that are linked to the terminals of the wirethroughs. To each wirethrough vertex a routing capacity is assigned according to the number of wirethroughs it represents. Each time a wirethrough is used, the capacity of

its corresponding vertex is decreased by one. When the capacity of a wirethrough vertex becomes zero, the vertex is closed for further usage. Fig. 4-3 shows an example of a modified floorplan graph, c1 is a wirethrough vertex with a capacity 3. Dashed lines in the figure represent the wirethroughs.

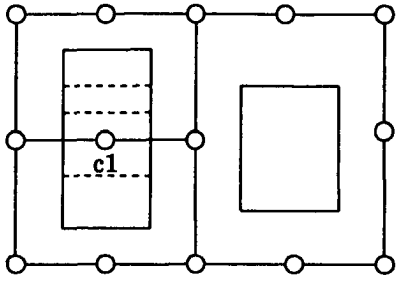


Figure 4-3. Enhanced floorplan graph

In the shortest path algorithm, the frontier vertices will expand to the wirethrough vertices in the way similar to the other vertices. The cost for using a wirethrough vertex is a function of the length of the wirethrough. If a wirethrough vertex has a capacity larger than one, the wirethrough resulting in the shortest path is selected and is locked for other nets.

4.5 Power net routing

Integrated circuits are fed by at least two power nets, a ground (GND) net and a power supply (VDD) net. The power nets must be routed in metal layer, since neither poly-silicon nor diffusion are suitable for the heavy currents on these nets. Even in double metal process it is often preferable to route the power nets in a single metal layer, leaving the other metal layer free for the routing of the other nets. As a consequence the power nets may not cross each other. This constraint restricts the topology of the paths of these nets. We call it the *planar routability constraint* of the power nets.

Another observation about power nets is that the more current they have to carry, the wider they must be. Metal wires that are required to carry too much current suffer metal migration. If this happens, the atoms move within the wire leaving a break in the conductor. Typical process rules specify the minimum wire width as a function of the maximum current density. A secondary criterion in power net routing is thus to minimize the layout area occupied by the power wires.

Algorithms to route the power nets given one pad for each of the two power nets are reported in [Syed82a, Roth81, Lie82, Moul83, Xion86a]. Some of them impose restrictions on the pad or terminal positions [Roth81, Lie82]. Two interdigitated connection trees, one for each net, with the root at the pad and leaves at the terminals are usually constructed. To guarantee the planar routability of the power nets, it has been proven [Syed82a] that given one power supply pad and one ground pad the planar routability can always be guaranteed if for every block there exists a cut (a line that intersects a block boundary at exactly two points) separating the power supply terminals and ground terminals. In this section we propose a more general algorithm for the planar power net routing (see also [Cai88]) where the number of pads for each net is not restricted to one.

More than one pad per power net is needed in the cases, (1) where the number of pads per power net on an integrated circuit is not restricted to one, so that the restriction on the power terminal ordering on the blocks mentioned above can be relaxed allowing for more flexible power net routing; (2) to ease the current load of each pad; (3) to shorten the wiring length, especially in the case of hierarchical layout construction. For example, in Fig. 4-4 (a) three pads are placed on the boundary of the hierarchical module (bounded by the dashed line) to avoid parallel running power wires which is the case if only one pad was used, see Fig. 4-4 (b).

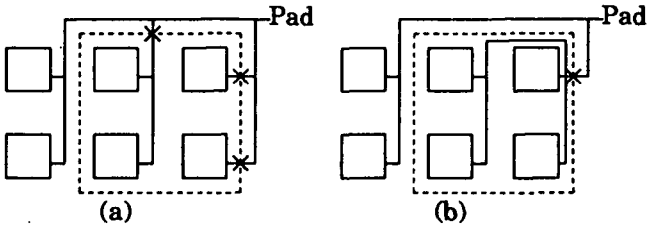


Figure 4-4. Hierarchical layout structure.

In the following subsection some basic definitions are given. In subsection 4.5.2 conditions to guarantee a planar power-net routing are established by deriving a formula to calculate the minimum number of pads needed for a given placement. In subsection 4.5.3 an algorithm is presented for a planar power-net routing given a number of pads not less than the minimum. In subsection 4.5.5 the algorithm is illustrated by an example.

4.5.1 Definitions and problem formulation

In building-block layouts, a circuit (or a hierarchical module) consists of a number of rectangular blocks of arbitrary size. The blocks may have any number of power terminals on the block boundaries. Let us call the power terminals on the periphery of the circuit, *pads* and the power terminals on the blocks, *terminals*. Obviously, terminals will become pads when we go down in the hierarchy. Some terminals on a block are internally connected by a *feedthrough*. It is assumed that only one terminal of a group internally connected terminals needs to be connected externally.

Definition 4-2: *Equivalent block (EB):* the equivalent block of a block is obtained by (1) removing all but one terminal of each group of consecutive power terminals belonging to the same net, and (2) removing all but one terminal of each group of internally connected terminals, in an order such that the number of remaining terminals on the block is minimal.

Definition 4-3: *Equivalent terminal (ET):* the equivalent terminals on a block are the remaining terminals on the equivalent block of the block.

An example of an equivalent block is shown in Fig. 4-5. The dashed line is a feedthrough.

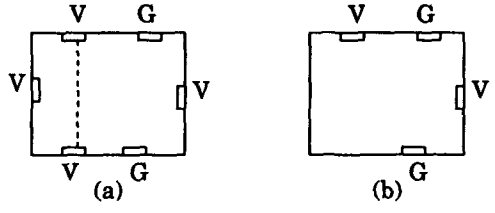


Figure 4-5. (a) a block (b) its equivalent block.

Definition 4-4: *Minimum number of pads (MNP) of a net:* The minimum number of pads (MNP) of a power net is the minimum number of pads required to guarantee a planar power net routing.

The planar power net routing problem can be seen as follows: the pads and the terminals on the blocks must be divided into a number of clusters each of which contains a single pad and some terminals of the same net. A tour of a cluster is a closed path of the terminals and the pad. It is known that an optimal tour of points in the plane never crosses itself. If we can ensure that the tours of the clusters do not cross each other, a planar routing can be achieved. In this case, for each cluster, a metal wire can then be placed inside the tour to connect all terminals to the pad enclosed by the tour without crossing other power wires. An example of such a clustering is shown in Fig. 4-6. In the example two VDD and one GND pads are specified. All terminals are connected. The tours are indicated by the dotted lines. VDD terminals are indicated by a 'V' and GND terminals are indicated by a 'G'.

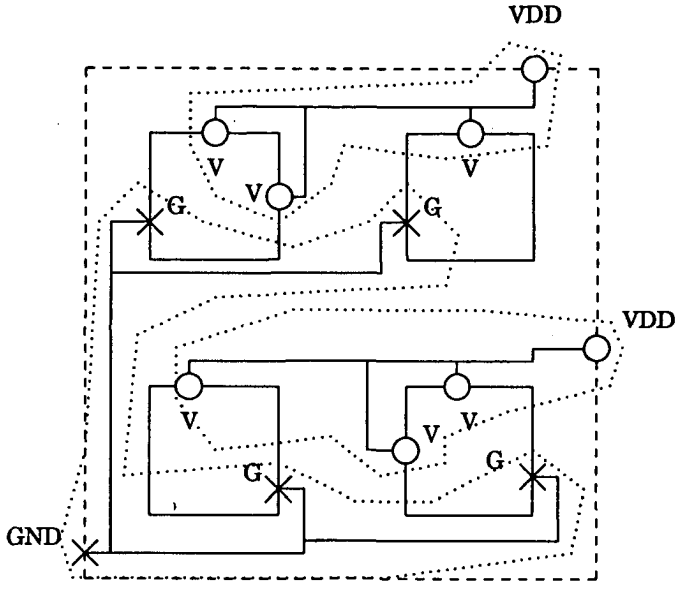


Figure 4-6. Planar routing of the power nets.

4.5.2 Conditions to guarantee a planar routing

In this subsection we consider the problem of finding the minimum number of pads, MNP, required for each net to guarantee a planar routing for the two power nets. It is assumed that the relative order of the pads in different nets is not fixed.

Lemma 4-1: For a single block, a planar routing exists if and only if the minimum number of pads (MNP) for each net is equal to the number of ETs of the net.

Proof: Sufficiency: since there is one pad for each ET and the order of the pads is not fixed, it is obvious that a planar routing exists. Note that if an ET can be connected to a pad the original terminals merged into the ET can also be connected to the pad.

To prove the necessity part of the lemma let us assume that a planar routing

exists while two ETs share a common pad. From the definition we know that no two ETs of the same net can be adjacent to each other. The two possible ways to obtain a connection for these two ETs are shown in Fig. 4-7 (a) and Fig. 4-7 (b). In both cases there are other ETs isolated from their pads, hence no planar routing exists for either case. This contradicts our assumption and the lemma has been proved. □

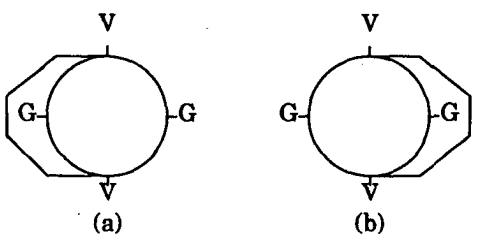


Figure 4-7. No planar routing exists for a block that has two ETs connecting to the same pad.

Lemma 4-2: For two blocks a planar routing exists for the two power nets if and only if the minimum number of pads (MNP) for each net is equal to the total number of ETs of the same net on the two blocks minus one.

Proof: Since the VDD ETs and the GND ETs of a block always interleave, we can always construct a composite block from the two blocks such that one pair of VDD ETs shares a common VDD pad and one pair of GND ETs shares a common GND pad. Figure 4-8 shows such a composite block. Then, using lemma 1, it is clear that if the number of pads for a net is equal to the total number of ETs of the net minus one, a planar routing can be achieved.
To prove the necessity part of the lemma let us assume that a planar routing exists while the number of pads of a net is less than the total number of ETs of the net minus one. From lemma 1 we know that no two ETs on the same block can share a common pad. Suppose that two pairs of ETs of the same net but on different blocks share two different pads, it is not hard to see that the closed circle formed by the two connections will always isolate some ETs of

the other net from the outside due to the fact that the ETs of the two nets interleave. Hence no planar routing exists. Contradiction. □

Note that the MNPs for the two power nets are equal, because the number of VDD ETs is equal to the number of GND ETs on each block. Therefore we can speak of the MNP of the circuit. For example, the MNP of the circuit in Fig. 4-8 is (5 - 1) = 4.

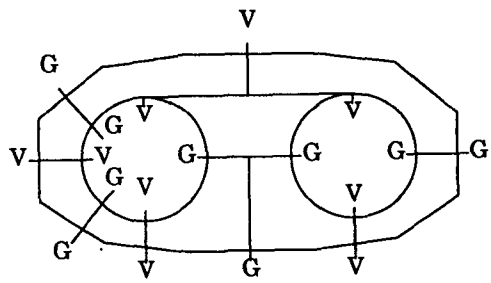


Figure 4-8. Planar routing for two blocks.

The following theorem presents the MNP condition to ensure a planar routing of n blocks in two power net routing.

Theorem 4-1: A planar routing for two power nets with each net connecting to each of the n blocks exists ($n \geq 1$), if and only if the minimum number of pads (MNP) of each net is equal to the total number of ETs of the net minus $(n-1)$.

Proof: We prove the theorem by induction. For one block and two blocks the proofs are given by Lemma 4-1 and Lemma 4-2 respectively. Suppose the theorem is correct for n blocks. Thus the following expression is true:

$$MNP_n = T_n - (n-1)$$

where MNP_n is the minimum number of pads for n blocks and T_n is the total number of ETs of one net on the n blocks.

The case of n+1 blocks can be considered as a composite block of a basic block, j, and a composite block of n blocks. Suppose the number of ETs of the net on block j is T_j. The total number of ETs of the net on n+1 blocks, T_{n+1}, is then $T_{n+1} = T_n + T_j$. Because the minimum number of pads on the composite block of n blocks MNP_n is actually the number of ETs on the composite block, according to Lemma 4-2 the minimum number of pads for n+1 blocks is:

$$\begin{aligned} MNP_{n+1} &= MNP_n + T_j - 1 \\ &= T_n - (n-1) + T_j - 1 \\ &= T_n + T_j - n \\ &= T_{n+1} - ((n+1)-1) \end{aligned}$$

Hence the theorem is also correct for n+1 blocks, and the theorem has been proved. □

Notice that if each block has only one VDD ET and one GND ET, according to Theorem 1, the minimum number of pads per net is one. This special case has already been proved in [Syed82a].

4.5.3 A planar power net routing algorithm

We propose an algorithm for the planar topological routing of two power nets which does not restrict the number of pads per net to one. The algorithm consists of three parts. In the first phase, a terminal clustering algorithm is performed which divides the set of terminals and pads into clusters each of which contains exactly one pad and some terminals. In the second phase a topological routing path is found for each cluster. Finally, in the third phase wire widths are calculated. Instead of using a "flat" approach by connecting one terminal at a time or one tree at a time [Russ85, Haru87], we follow a hierarchical approach by a top-down terminal clustering and then a bottom-up path routing. This approach considers all terminals at the same time at each hierarchical level and guarantees a solution if one exists.

4.5.3.1 Top-down terminal clustering.

By terminal clustering we mean that a cluster of terminals of the same net is determined for each pad (a terminal in the cluster is said to be assigned to the pad). The task of the terminal clustering algorithm is to assign each terminal to a pad while the planar routability is guaranteed. Notice that this clustering is not required if there is only one pad for each of the nets. After the clustering the clusters belonging to the same power net can be considered as different nets and their trees will not be connected to each other.

To guarantee a planar routing not only the number of pads is crucial but also the order of the pads on the circuit boundary. A planar routing can be achieved if the pads are ordered in such a way that if we replace the circuit by its EB, the number of resulting ETs of each net is not less than the MNP of the circuit. Let us call this the *MNP constraint.* Assume a number of pads not less than the MNP is given for each power net and a position or a desired side is provided for each pad. First, we do a virtual placement of the floating pads on their desired side in such an order that the MNP constraint is satisfied. The blocks are replaced by their EBs, however, without removing the consecutive terminals.

The underlying idea of the clustering algorithm is the following. The clustering algorithm partitions the placement recursively according to the given channel-routing order. This means that, starting at the root level, a module is recursively partitioned into two submodules by a horizontal or a vertical channel. Notice here that the power-net routing must be done after the channel definition. For each of the two submodules, first the MNP is calculated, then pads on the module are assigned to the submodules. They are placed on the boundary of each of the submodules so that two conditions are satisfied: (1) in the submodules a planar routing is guaranteed; (2) a planar routing is guaranteed between the module and the two submodules.

Basically, depending on the MNPs and the relative sizes of the two submodules the pads on the module are divided into two groups and assigned to each of the submodules. To guarantee the planar routing in the submodules the number and order of the pads assigned to the submodules must satisfy the MNP constraint in the submodules. Due to this constraint some pads of the parent module may need to be assigned to both submodules. On the other hand, referring to the proof of Lemma 4-2, to guarantee a planar routing between the parent module and the submodules, the number of pads assigned to both submodules may not be more than two, one for each net, and in such an order that the "reverse cyclic order" [Syed82a] is satisfied. This order implies that if there are two pads assigned to both submodules, then starting from the pads on the two submodules assigned from the same parent pad, and moving along the corresponding boundaries in opposite directions, it should be possible to reach the pads assigned from the other parent pad without encountering any other pads (e.g. Fig. 4-8). Theorem 4-1 ensures that this pad assignment can always be realized. Furthermore, the wiring length is taken into consideration by placing the pads on the submodules as close as possible to the corresponding parent pads. In Fig. 4-9 an example of such a pad assignment is given. The numbers inside the (sub)modules are the MNPs of the (sub)modules. The VDD pad v_2 (Fig. 4-9(a)) on the module is assigned to both submodules (Fig. 4-9(b)) to guarantee the planar routing in the submodules.

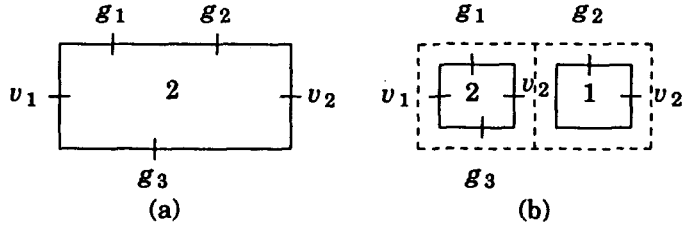


Figure 4-9. (a) a module (b) partitioning in submodules.

If there is only one pad left for each net on a module, no further partitioning is needed. All terminals of the net inside the module are assigned to the pad. If the number of blocks inside the module becomes one, the terminals on the block are assigned to the nearest pads while maintaining the planar routability to the pads.

The pseudo code of the terminal clustering algorithm is given in Algorithm 4-3.

Algorithm 4-3: *Terminal clustering algorithm*

INPUT: A placement and a number of pads

OUTPUT: An assignment of each terminal to a pad.

METHOD:

Calculate the MNP of the circuit.

If the circuit is planar routable (checking the MNP)
{
Assign virtual positions to the pads on the circuit boundary.

Assign_pad (circuit)
}

```
Procedure Assign_pad(module)
  {
  If (the number of pads per net ≡ 1
  or the number of blocks inside the module ≡ 1)
    {
    Assign the terminals inside the module to the pads
    }
  else
    {
    Partition the module into two submodules.

    Calculate the MNPs for the two submodules.

    Assign pads on the boundaries of the submodules
    (keeping the planar routability).

    Assign_pad(submodule1), Assign_pad(submodule2).
    }
  }
```

4.5.3.2 Bottom-up path routing.

After the terminals are assigned to the pads the routing becomes rather simple. The clusters of the same power net are considered as different nets. The topological path routing of the power nets is also performed hierarchically, however, bottom-up, in the opposite direction as the terminal clustering. Channels with lower routing orders are considered first. The two neighboring modules of a channel will form a composite module after the channel is processed. The path is locally optimized in wiring distance while keeping the planar routability constraint satisfied.

Basically, terminals facing the boundary of the composite module are connected directly to the boundary; a terminal inside the channel is routed to the end of the channel in the direction of the pad it is assigned to. The pad order on each hierarchical level determined in the terminal clustering phase must be maintained. Therefore, the terminals assigned to the same pad must be routed to the consecutive positions on the boundary of the composite module

and the order of the terminals assigned to different pads must obey the assigned order of the pads. To satisfy these conditions terminals outside the channel must sometimes be routed through the channel. For instance, in Fig. 4-10 if one VDD pad is placed on the bottom side and one GND pad is placed at the top side, one solution is to route the VDD terminal on the left module through the channel to the bottom side of the composite module.

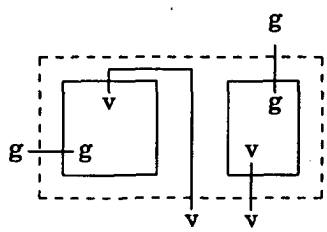


Figure 4-10. Local power net routing.

4.5.3.3 Wire width calculation.

After the topological routing is done a tree is formed for each pad of which the root is at the pad and the leaves are at the terminals connected to the pad. Given the current demand of each power terminal, the width of the power wire segments corresponding to each branch of the tree can be calculated. Although any accurate methods, for example [Chow87], can be adopted here to calculate the wire width we used a simple scheme by accumulating the current demand from the leaf terminals to the root and converting the current demand on each tree branch into wire width. The relationship between the metal wire width and the current density is usually provided in the design rules.

4.5.4 An example

To illustrate the algorithm the following example is worked out stepwise. In Fig. 4-11 a placement of three blocks is shown. VDD terminals are indicated by a 'v' and GND terminals by a 'g'. Two VDD pads, v_1 and v_2, and two

GND pads, g_1 and g_2, are placed on the circuit boundary, one on each side. The two routing channels are also shown in the figure.

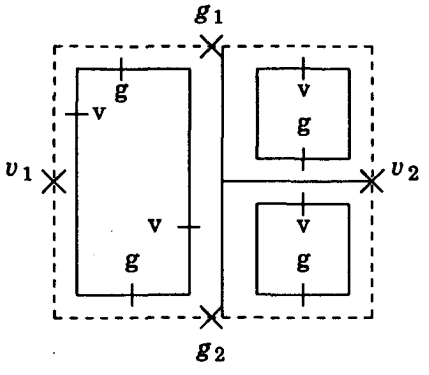


Figure 4-11. A power net routing example.

4.5.4.1 Terminal clustering.

Since the MNP of the circuit is 2 and there are two VDD and two GND pads this example is planar routable. In the first step pads are assigned to the first level of the hierarchy, see Fig. 4-12. V_2 and g_2 are assigned to both submodules, because the MNP of module m_1 is 2.

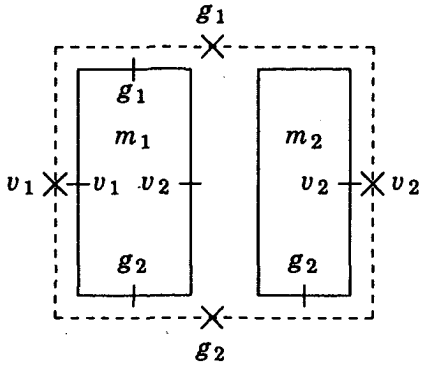


Figure 4-12. Terminal clustering at the first hierarchical level.

Since the number of blocks in m_1 is one and the number of pads per net on m_2 is one, all terminals can be assigned to the pads in the second step and the terminal clustering is finished, see Fig. 4-13.

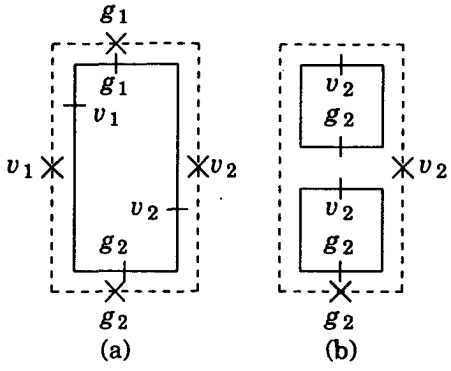


Figure 4-13. Terminals are assigned to the pads.

4.5.4.2 Path routing.

The path routing is carried out starting from the lowest level. In the first step the second level of the hierarchy is routed, see Fig. 4-14.

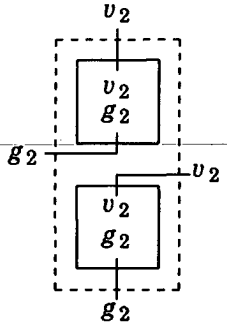


Figure 4-14. Path routing at the second hierarchical level.

In the second step the first level of the hierarchy is routed using the result of the first step, see Fig. 4-15.

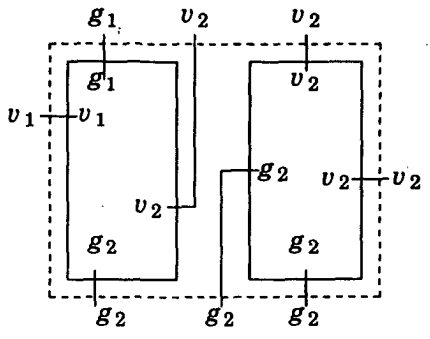


Figure 4-15. Path routing at the first hierarchical level.

After these steps the routing from the boundary to the pads can be simply done. The final result is shown in Fig. 4-16.

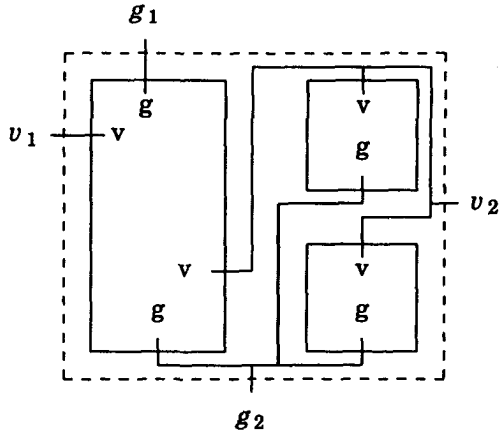


Figure 4-16. The result of the power net routing.

4.5.5 Conclusions

A novel algorithm is presented in this section to solve the planar routing problem for two power nets. More than one pad per power net is allowed which results in a forest of multiple trees. The conditions to guarantee a planar routing are outlined by deriving the minimum number of pads (MNP) requirement. The algorithm guarantees a solution if these conditions are satisfied. The algorithm is efficient because of its hierarchical nature. The nets are processed all at the same time which prevents net or terminal ordering. Further, it provides a good starting point to tackle the planar routing problem for more than two power nets or critical nets.

4.6 References

Cai86. H. Cai and P. Dewilde, "Automatic Routing in a General Cell Environment," pp. 3.1-3.24 in *The Integrated Circuit Design Book*, ed. P. Dewilde, Delft University Press, Delft (Jan. 1986).

Cai88. H. Cai, "Multi-Pads, Single Layer Power Net Routing in VLSI Circuits," *Proc. 25th Design Automation Conference*, pp. 183-188 (June 1988).

Cai85. H. Cai, H. Th. Verheyen, R. Nouta, and P. Dewilde, "An Automatic Routing System for General Cell VLSI Circuits," *Proc. Custom Integrated Circuits Conference*, pp. 68-71, Portland, USA (May 1985).

Chow87. S. Chowdhury, "An Automated Design of Minimum-Area IC Power/Ground Nets," *Proc. 24th DAC*, pp. 223-229 (1987).

Cies82. M. J. Ciesielski and E. Kinnen, "An Analytical Method for Compacting Routing Area in Integrated Circuits," *Proc. of 19th Design Automation Conference*, pp. 30-37 (1982).

Clow84. G. W. Clow, "A Global Routing Algorithm for General Cells," *Proc. 21th Design Automation Conference*, pp. 45-51 (1984).

Dijk59. E. W. Dijkstra, "A Note on Two Problems in Connection with Graphs," *Numerische Math.* **1** pp. 269-271 (1959).

Drey72. S. E. Dreyfus and R. A. Wagner, "The Steiner Problem in Graphs," *Networks* **1** pp. 195-207 (1972).

Elme84. P. C. Elmendorf, "KWIRE: A multiple-technology, user-reconfigurable wiring tool for VLSI," *IBM J. RES. Develop.* **28**(5) pp. 603-613 (September 1984).

Even79. S. Even, *Graph Algorithms,* Pitman (1979).

Fowl85. C. Fowler, G. D. Hachtel, and L. Roybal, "New Algorithm for Hierarchical Place and Route of Custom VLSI," *Proc. ICCAD*, pp. 273-275 (1985).

Fuku87. M. Fukui, A. Yamamoto, R. Yamaguchi, S. Hayama, and Y. Mano, "A Block Interconnection Algorithm for Hierarchical Layout System," *IEEE trans. on CAD* **CAD-6**(3) pp. 383-391 (May 1987).

Hana65. M. Hanan, "On Wiring for Large Scale Integrated Circuits," *IBM Research Rep.*, pp. PC-1375 (Feb. 1965).

Haru87. S. Haruyama and D. Fussell, "A New Area-Efficient Power Routing Algorithm for VLSI Layout," *Proc. ICCAD*, pp. 38-41 (1987).

Hass82. J. E. Hassett, "Automatic Layout in ASHLAR: An Approach to the Problem of 'General Cell' Layout for VLSI," *Proc. 19th Design Automation Conference*, pp. 777-784 (1982).

High69. D. W. Hightower, "A solution to line-routing problems on the continuous plane," *Proc. 6th Design Automation Workshop*, pp. 1-24 (1969).

Hoel76. J. H. Hoel, "Some Variations of Lee's Algorithm," *IEEE Trans. on Computers* **C-25**(1) pp. 19-24 (January 1976).

Hsu87. Y. C. Hsu, Y. Pan, and W. J. Kubitz, "A Path Selection Global Router," *Proc. DAC*, pp. 641-644 (1987).

Kimu83. S. Kimura, N. Kubo, T. Chiba, and I. Nishioka, "An Automatic Routing Scheme for General Cell LSI," *IEEE Transactions on Computer-Aided Design* **CAD-2**(4) pp. 285-292 (October 1983).

Kurt85. J. M. Kurtzberg and E. J. Yoffa, "ACE: A Congestion Estimator for wiring custom chips," *INTEGRATION, the VLSI journal*, (3) pp. 113-127 (1985).

Lee61. C. Y. Lee, "An algorithm for path connections and its applications," *IRE Trans. Electron. Comput.* **EC-10** pp. 346-365 (Sept. 1961).

Lie82. M. Lie and C. S. Horng, "A Bus Router for IC Layout," *Proc. 19th Design Automation Conference*, pp. 129-132 (1982).

Lins84. R. Linsker, "An iterative-improvement penalty-function-driven wire routing system," *IBM J. Res. Develop.* **28**(5) pp. 613-624 (Sept. 1984).

Moor59. E. F. Moore, "The shortest path through a maze," *Proc. Int. Symp. on the Theory of Switching Circuit, in Annals of the Harvaed Computation Laboratory,* **30, pt II** pp. 285-292 Cambridge, Mass.: Haevard University Press, (1959).

Moul83. A. S. Moulton, "Laying the Power and Ground Wires on a VLSI Chip," *Proc. 20th DAC*, pp. 754-755 (1983).

Pers84. G. Persky and L. V. Tran, "Topological Routing of Multi-bit Data Busses," *Proc. 21th Design Automation Conference*, pp. 679-682 (1984).

Prea85. B. Preas and C. S. Chow, "Placement And Routing Algorithms For Topological Integrated Circuit Layout," *Proc. ISCAS*, pp. 17-20 (1985).

Prim57. R. C. Prim, "Shortest Connection Network and Some Generalizations," *Bell System Tech. J.* **36** pp. 1389-1401 (1957).

Rive82. R. L. Rivest, "The 'PI' (Placement and Interconnect) System," *19th design Automation Conference*, pp. 475-481 (1982).

Roth83. H. Rothermel and D. A. Mlynski, "Automatic Variable-Width Routing for VLSI," *IEEE Transaction on Computer-Aided Design* **CAD-2**(4) pp. 271-284 (October 1983).

Roth81. H. J. Rothermel and D. A. Mlynski, "Computation of Power Supply Nets in VLSI Layout," *Proc. 18th DAC*, pp. 37-42 (1981).

Rubi74. F. Rubin, "The Lee Path Connection Algorithm," *IEEE Trans. on Computers* **C-23**(9) pp. 907-914 (September 1974).

Russ85. D. W. Russell, "Hierarchical Routing of Single Layer Metal Trees in Compiled VLSI," *Proc. ICCAD*, pp. 270-272 (1985).

Souk81. J. Soukup and J. C. Royle, "On Hierarchical Routing," *Journal of Digital Systems* **5**(3) pp. 265-289 (1981).

Syed82. Z. Syed, A. E. Gamal, and M. A. Breuer, "On Routing for Custom Integrated Circuits," *Proc. 19th Design Automation Conference*, pp. 887-893 (1982).

Syed82a. Z. A. Syed and A. E. Gamal, "Single Layer Routing of Power and Ground Networks in Integrated Circuits," *J. of Digital Systems* **6**(1) pp. 53-63 (1982).

Wisn84. J. A. Wisnieski and R. C. Peters, "Global Routing in a Rectilinear Macrocell Environment," *Proc. ICCAD Conference*, pp. 60-62 (1984).

Xion86. J. G. Xiong, "Algorithm for Global Routing," *Proc. 23rd Design Automation Conference*, pp. 824-830 (1986).

Xion81. J. G. Xiong and T. Kozawa, "An Algorithm for Searching Shortest Path by Propagating Wave Fronts in Four Quadrants," *Proc. 18th Design Automation Conference*, pp. 29-36 (1981).

Xion86a. X. M. Xiong and E. S. Kuh, "The Scan Line Approach to Power and Ground Routing," *Proc. ICCAD*, pp. 6-9 (1986).

5. GEOMETRICAL CHANNEL DEFINITION

5.1 Introduction

Detailed routing is realized by routing the channels by a channel router in the sequence according to the channel order determined. The channel router takes as input a channel *envelope* which is a description of the geometrical channel boundary and the terminal information on the boundary. The task of the channel router is to connect the terminals physically using a given number of routing layers such that the channel width is minimized.

The topological channel definition has determined the relative positions of the blocks and the channels. Prior to sending a channel to the channel router the geometrical envelope of the channel must be determined. The process of defining the envelope of a channel is called the *geometrical channel definition.*

Channel routers are very well developed tools to perform the physical wiring. However, while the wiring within each individual channel might be done near to optimal by the channel router, the global result may be far from optimum, because a channel router optimizes only a small part of the layout each time. To cope with this locality problem global information about the channels and the net paths should be taken into account when the channel envelopes are defined so that a good global routing result can be achieved. Despite the importance of this problem it has escaped the attention of most researches in the past. In this chapter we consider the different steps of the geometrical channel definition problem keeping the "global" aspects in mind.

One of the steps is the detailed placement of the building blocks. An advantage of using slicing structures is that the positions of the blocks can be adjusted during the detailed routing. Therefore, in addition to conflict-free channel orders, routing in the channels can be optimized by adjusting the placement of the blocks. This can be realized by shifting the blocks along the

channel sides to find optimal positions. This freedom of placement adjustment will be exploited by the geometrical channel definition algorithm to optimize the wiring in the channels.

Another interesting problem is the optimization of wiring in the channel intersection areas. Wiring congestion often occurs in the channel intersection areas because of many wire crossovers in those areas, for channels are routed separately, and the only dependency between the channels is the ordering constraint. To obtain a good global result one should know where the wires exiting the ends of a channel are going to in the neighboring channels. For example, a bundle of wires (e.g. a signal bus) which passes through several channels should be kept as straight and parallel as possible. As channel routers do not have the knowledge of the world outside the channel envelope an ordering preference for the nets exiting the channels should be indicated to avoid unnecessary wire twisting in the channel intersection area.

This chapter is organized as follows. Section 5.2 is devoted to a more precise definition of the channel envelope. Section 5.3 outlines how a channel is geometrically created. In section 5.4 an algorithm is described to optimize the relative positions of the blocks. In section 5.5 an algorithm is proposed to determine a terminal ordering at the channel ends in order to avoid unnecessary wire twisting. Finally, in the last section a brief description of a channel router will be given.

5.2 Channel envelope

There are two types of channels, vertical and horizontal. Let us consider the horizontal channels. The channel envelope of a horizontal channel is defined by the following elements.

(1) A horizontal channel is a closed rectilinear simple polygon with the boundary consisting of four distinctive portions: left and right sides; upper and lower sides (or boundaries).

(2) A left or a right side is a single vertical line segment and is called an open side or an end of the channel. They are fixed in location while their lengths may be changed to complete the routing in the channel. The upper and lower boundaries are horizontally monotone, i.e., when we traverse them from left to right, the x-coordinates are non-decreasing. The line segments constituting a boundary are called edges of the boundary. Furthermore, if it is required to guarantee the routability of the channel, the location of the channel ends may also be changed by adding extra wiring columns in the channel.

(3) Terminals on the upper and lower boundaries have fixed positions and may not be located on the vertical edges. Terminals on the channel ends have floating positions and are called floating terminals. The exact positions of these terminals are determined by the channel router. However, some channel routers accept an ordering preference of the floating terminals on the channel ends.

(4) The length of a channel is the distance between the channel ends. The width of a channel is the distance between the lowest horizontal edge of the lower boundary and the highest horizontal edge of the upper boundary.

Figure 5-1 shows an example of a horizontal channel. The terminals with the same net name must be connected to each other in the channel. The channel ends are indicated by the dotted lines.

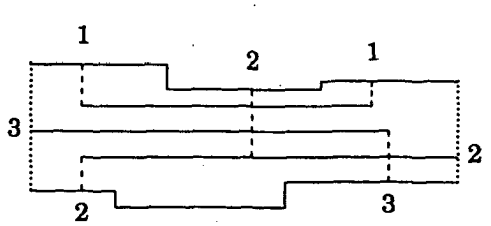


Figure 5-1. A horizontal channel.

5.3 Composing a channel

A layout is composed by fitting the building blocks and the routing channels together. A channel can be routed if the terminals on both channel boundaries have fixed positions. The channel ordering scheme outlined in chapter III guarantees a feasible ordering of the channels. Each time a channel is routed the width of the channel is returned by the channel router. The two blocks adjacent to the channel are shifted into a distance which is just enough to fit the routed channel. Together with the channel the two blocks are then merged into a *composite* block as shown in Fig. 5-2.

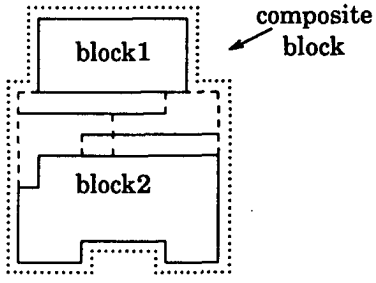


Figure 5-2. A composite block.

This procedure continues until all channels are routed. The last composite block is the layout of the total circuit. Note that although the building blocks are rectangularly shaped, a composite block is usually rectilinear in shape, because the two adjacent (composite) blocks along the two sides of a channel usually differ in shape and size. Each composite block has an east, a west, a south and a north side, each of which may consists of several consecutive horizontal and vertical edges. A horizontal channel always ends at the east and west side of the resulting composite block while a vertical channel always ends at the south and north side of the composite block. In the detailed routing system no distinction is made between a building block and a composite block. Hence, we may relax the restriction of strict rectangular shapes of the

building blocks. The blocks could have rectilinear shapes, with the restriction that terminals must be established at the east, west, north and south side of the block as in a composed block, see Fig. 5-3 for an example.

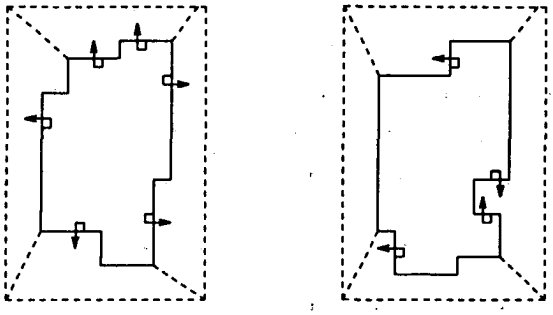


Figure 5-3. (a) a correct rectilinear shaped building block. (b) an incorrect one.

The shape of a channel is determined by the neighboring boundaries of the adjacent blocks. As the adjacent blocks are often rectilinear in shape the routing channels obtained are also rectilinear in shape, see for an example Fig. 5-4.

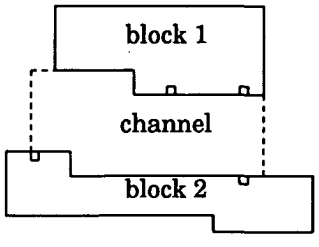


Figure 5-4. A rectilinear shaped channel.

For a horizontal channel the initial position of the left open side is

determined by the minimum of the position of the leftmost terminal on either the upper or the lower adjacent block and the position of the corner of the shorter block whichever gives the leftmost value (see Fig 5-4). The initial position of the right open side is determined in a similar way. Normally the channel length is fixed, however, it can be enlarged by the channel router to guarantee the routability of the channel. A vertical channel is treated identically to a horizontal channel except for a rotation over 90 degrees. After the envelope of a channel is determined and a connection list is obtained from the global routing data, the channel data can be sent to the channel router.

5.4 Relative positioning of building blocks

5.4.1 Relative positions of blocks

As explained in the previous section each time a channel is routed a composite block is constructed containing the channel and its two adjacent (composite) blocks. The width of the channel which is the distance between the two blocks is determined by the channel router. The lateral relative position of the two blocks, on the other hand, can be determined before the channel routing. This relative position is called the *offset* of one block with respect to the other, as is illustrated in Fig. 5-5.

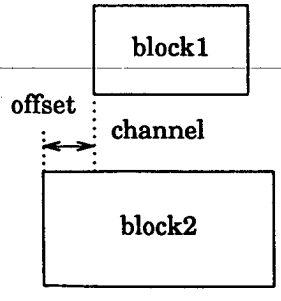


Figure 5-5. The offset of two blocks.

Although not crucial for the routing completion, the lateral relative position of the blocks can have great impact on the routing result, because the channel envelope, hence, the resulting channel width depends on the offset. An example of a channel routed at different offsets is shown in Fig. 5-6.

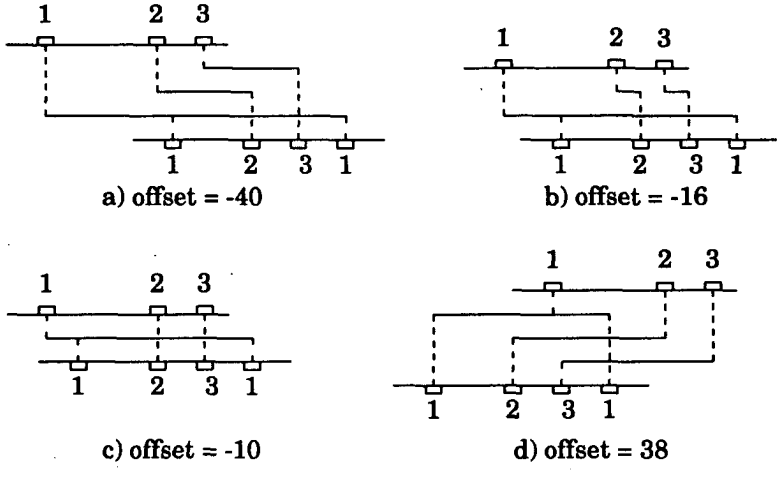


Figure 5-6. A channel routed at different offsets.

In this section an algorithm is presented to calculate the optimal offset of the blocks. We define the optimal offset of two blocks to be the offset at which the width of the current channel is minimized without enlarging the chip dimension in the channel direction. The channel direction of a channel is defined to be the direction parallel to the upper and lower boundaries of the channel. Hence the channel direction of a horizontal channel is the horizontal direction.

An algorithm to determine the optimal offset for rectangular channels is proposed by LaPaugh et. al [LaPa83] in which the optimal offset is defined as the offset which results in the minimum channel density regardless of other channels. Algorithms to determine the optimal offset for non-rectangular channels are proposed in [Kimu83, Cai86]. The algorithm proposed in

[Cai86] takes also the wiring congestion in the adjacent channels of the current channel into consideration. Based on the global routing data an approach is proposed in [Fuku87] which modifies the placement before the detailed routing phase. To achieve a good result we believe that both local and global aspects should be considered. This is done by the algorithm presented in this section.

The algorithm is divided into two phases, a global phase and a local phase. The global phase determines a range of allowed offsets between the two adjacent blocks of a channel. It ensures that if an offset is chosen within the range, the chip dimension in the channel direction will not be enlarged. In the local phase the optimal offset is determined which is the offset within the allowed offset range that results in the minimum channel width. We first discuss the local phase.

5.4.2 Determining the optimal offset given an offset range

In this subsection an algorithm will be described to determine the optimal offset of two blocks given the allowed offset range which is specified by the minimum offset d_{min} and the maximum offset d_{max} where $d_{max} \geq d_{min}$. If we shift one block from the minimum offset d_{min} to the maximum offset d_{max} with respect to the other block the offset yielding the smallest channel width is the optimal offset. Instead of actually routing the channel at each offset which would be very time consuming channel width is estimated by calculating the channel density at each offset.

5.4.2.1 Channel density of rectangular channels.

Let $l_i(d)$ be the leftmost terminal of net i at a given offset d, $r_i(d)$ be the rightmost such terminal, w_i be the width of net i (including the wire spacing). Let $t_i(x,d)$ be the contribution of net i to the density at position x when the offset is d which is defined as:

$$t_i(x,d) = \begin{cases} w_i & \text{if } l_i(d) \le x \le r_i(d) \text{ and } l_i(d) \ne r_i(d) \\ 0 & \textit{otherwise} \end{cases}$$

Then the channel density T_d for a rectangular horizontal channel containing n nets is defined as:

$$T_d = \max_x \sum_{i=1}^{n} t_i(x,d)$$

For example, the density of the channel in Fig. 5-7 is 4 if w_i is 1 for each net.

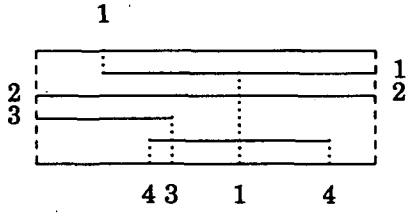


Figure 5-7. A channel with a density 4.

Several types of terminals can be distinguished. We define the leftmost terminal of a net to be the *begin terminal* of the net and the rightmost terminal of the net to be the *end terminal* of the net. Other terminals of the nets are called *dummy terminals*, since they do not affect the channel density. However, the type of the terminals can change when the offset is changed.

The channel density is a good estimate of the expected channel width, because most channel routers route a channel within its density. The density of a rectangular channel can be easily calculated by scanning the channel from left to right and counting the contribution of the begin and end terminals. A begin terminal increases the channel density by w_i while an end terminal decreases the channel density by w_i.

5.4.2.2 Channel density of non-rectangular channels.

In rectilinear shaped channels the rugged channel boundaries also influence the channel density, because "bay" areas can be used effectively to minimize the channel width. To calculate the density of a rectilinear shaped channel the shape of the boundaries must be taken into account which implies that a change in the channel boundary must be reflected in the channel density calculation. In routing examples we observed that the influence of a change, also called an indentation, in the channel boundary not only takes place at the position where the change occurred but spreads out to a distance before or after the position, because usually only one wire can be bended at a given position in the channel in order to follow the channel boundary. Therefore, the modeling in [Kimu83] which updates the density changes caused by the indentations only at the indentation positions is not adequate. The modeling proposed in [Cai86] is adopted here in which an indentation is modeled by placing an array of *pseudo terminals* on the boundary directly before or after the indentation position. These pseudo terminals act like begin or end terminals in influencing the channel density depending on the change direction of the boundary. The pseudo terminals are placed on the outer edge adjacent to the indentation position. We say that a *shadow edge* of the vertical edge is (virtually) created on the channel boundary which is the interval spanned by the pseudo terminals. This idea is illustrated in Fig. 5-8 (a).

If the boundary change increases (or decreases) the channel density the corresponding pseudo terminals will also increase (or decrease) the density by the same amount, one minimum wire width per terminal. Consequently the length of a shadow edge is equal to the corresponding vertical edge. However, if there are net terminals located on a shadow edge the length of the shadow edge is increased or decreased accordingly (see Fig. 5-8 (b) and (c)). Hence the length of a shadow edge is not constant over different offsets, as net terminals may change type over different offsets. This gives some

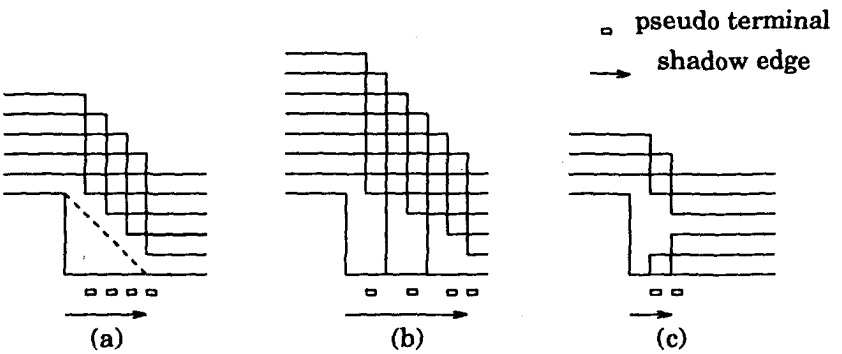


Figure 5-8. Shadow edges.

complications in the computation. Another problem is the overlapping shadow edges. If two shadow edges overlap they are replaced by two non-overlapping equivalent shadow edges, see for example Fig. 5-9.

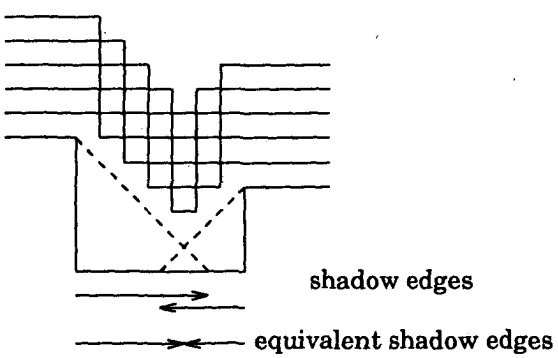


Figure 5-9. Equivalent shadow edges.

This process is repeated until the channel does not contain any overlapping shadow edges. Now we can forget about the rugged boundary of the channel and consider the rectilinear channel as a rectangular channel with some additional pseudo terminals reflecting the boundary changes. After this translation the channel density of the rectilinear channel can be calculated as

it was of a rectangular channel. One can also see this modeling of rectilinear shaped channels as a modeling that cuts off all sharp corners of the channel, see the shaded area in Fig. 5-10.

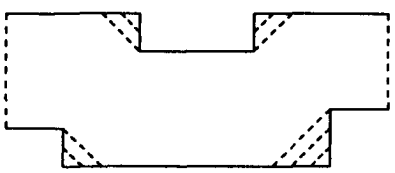


Figure 5-10. Sharp corners of a rectilinear channel are cut off.

5.4.2.3 The number of relevant offsets.

Given the minimum offset $d_{\min}$, and the maximum offset $d_{\max}$, we only need to calculate the channel density at the offsets within this range where a density change might occur. We observed that if one of the blocks is shifted laterally from $d_{\min}$ to $d_{\max}$ with respect to the other block the channel density can only change at positions where two terminals on opposite sides of the channel become aligned. Dummy terminals do not affect the channel density. Since a net may consist of many terminals and most of them are dummy many aligned positions need not be considered. As a consequence the number of offsets that need to be considered can be further reduced. The relevant offsets where a density change may occur are the positions, where an alignment of non-dummy terminals occurs and where a terminal changes type. These offsets are sorted in a list, the offset list.

5.4.2.4 The algorithm and its complexity.

The optimal offset is the one which results in the minimum channel density. This position is found by calculating the channel density at each relevant offset from the offset list. The channel density at each offset is calculated by scanning the channel from left to right and examing the contribution to the density of all begin, end and pseudo terminals (Algorithm 5-1).

Algorithm 5-1: *Optimal offset algorithm*

INPUT: top and bottom blocks, the offset range: $d_{\min}$ and $d_{\max}$.

OUTPUT: the optimal offset: d_{opt}.

METHOD:

Add pseudo terminals.

Determine all relevant offsets and sort them in the offset list.

For (all offsets i in the offset list)
{
Calculate the channel density: T_i.
If (T_i < minimum density found so far)
$d_{opt} = i$.
}

The time complexity of the algorithm is determined by the sorting procedure of the Cartesian difference of the top and bottom terminals to derive the offset list and by the for loop. The sorting of the Cartesian difference of two sets takes $O(n^2 logn)$ where n is the number of relevant terminals. The number of relevant offsets is bounded by $O(n^2)$. The calculation of the channel density at a given offset takes $O(n)$ time. Hence the time the for loop takes is bounded by $O(n^3)$. As a consequence the time complexity of the algorithm is of the order $O(n^3)$.

Experiments have shown that the modeling of the non-rectangular channels is fairly accurate [Knij88]. The comparison between the estimated channel width at the optimal offset and the channel width produced by the channel router [Groe87] based on the testing of 51 channels is depicted in Fig. 5-11. The estimated channel widths are set on the horizontal axis, and the real channel widths produced by the channel router are set on the vertical axis. The deviation from the optimal line, the 45 degree line (y=x), is small, about

80% of the test cases are within 5% from this line.

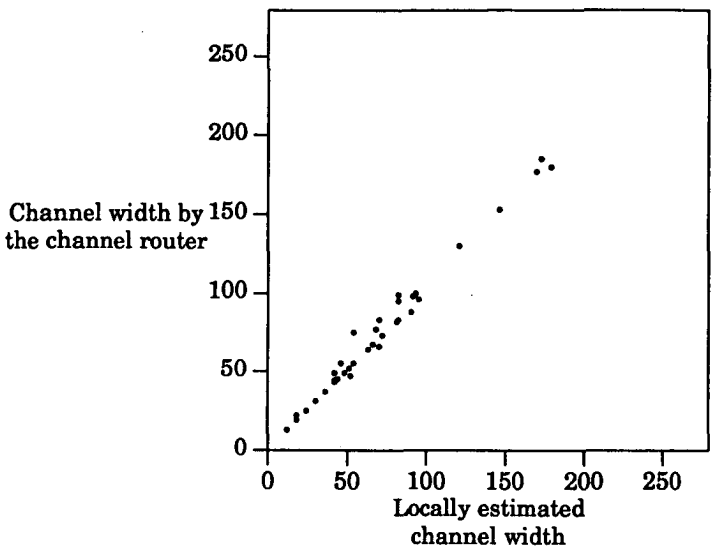


Figure 5-11. Comparing the estimated channel width to the real channel width.

5.4.3 Determining the offset range

The offset range between the two blocks adjacent to the current channel is determined with the help of the two channel-position graphs (see chapter II). Let us call the estimated width and height of the chip at the time prior to the routing of channel i W_i and H_i. If a block is not on the longest path of the channel-position graph in the channel direction of the current channel it still has some freedom to move in this direction without enlarging the longest path. The range of positions a block can take in a given direction without harming the longest path is also called the *slack* of the block in this direction. Consider the horizontal direction, the slack of a block, expressed in the minimum position, $l_{\min}$, and the maximum position, $l_{\max}$, is determined as

follows. Assume, that the length of the longest path from the left side of the block to 'd' in the horizontal channel-position graph is l_d, and, the length of the longest path from the left side of the block to 's' is l_s. Then

$$l_{\min} = l_d$$
$$l_{\max} = W_i - l_s.$$

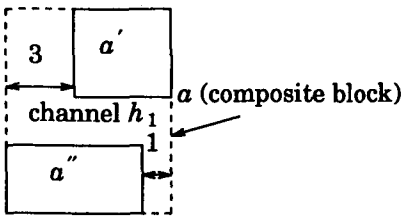
Prior to the routing of a channel, say a horizontal channel, the slacks of the two adjacent blocks in the horizontal direction are calculated. Assume, they are $l_{min\,1}$ and $l_{max\,1}$ for block 1 and $l_{min\,2}$ and $l_{max\,2}$ for block 2, the offset range of block 2 with respect to blocks 1, $d_{\min}$ and $d_{\max}$, is then

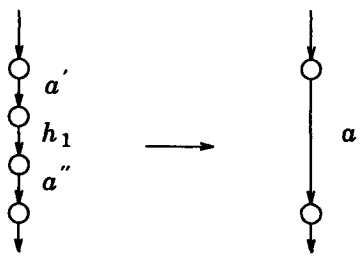
$$d_{\min} = l_{min\,2} - l_{max\,1}$$

and

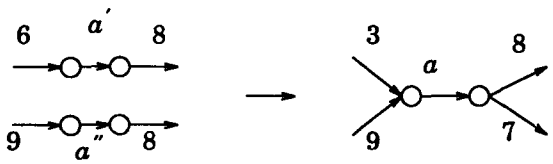
$$d_{\max} = l_{max\,2} - l_{min\,1}.$$

The channel-position graphs are updated with the actual routing data after the routing of each channel. The two adjacent blocks together with the channel are replaced by a composite block. In case of a horizontal channel the chain of the three edges representing the two adjacent blocks and the channel in the vertical channel-position graph are replaced by a single edge representing the composite block and the two parallel edges representing the two adjacent blocks in the horizontal channel-position graph are contracted to a single edge representing the composite block, see Fig. 5-12. After this operation possible parallel edges between two vertices are merged into one to which the largest weight among the parallel edges (channel segments) is assigned. Since a composite block is modeled as a rectangle in the graphs it will usually contain some empty space. To compensate this inaccuracy the widths of the channel segments adjacent to the composite block are adjusted. This implies that the width of a channel segment is decreased with the same amount as the width of the empty space in the composite block adjacent to it, see for example the numbers in Fig. 5-12. Note that the updating of the width of the channel segments is only done in the channel direction.





updating the vertical channel-position graph



updating the horizontal channel-position graph

Figure 5-12. Updating the channel position graphs, block a', channel h_1 and block a'' are replaced by the composite block a. Weights on the edges represent the channel segment widths.

5.4.4 Concluding remarks

The optimization of the detailed placement of building blocks is an important factor in achieving near to optimal layouts. A new algorithm has been presented which adjusts the relative positions of the building blocks during the detailed routing phase. Experimental results have shown that in many

cases the algorithm improves the routing result at an expense of about 25% more CPU time. It attempts to maximize the abutments of terminal arrays (busses) and to maximize the utilization of "bay" areas in the rectilinear shaped channels. Figure 5-13 shows the same part of a chip; on the left side the optimal relative block positioning algorithm was used; on the right side the algorithm was not used. The difference is very clear.

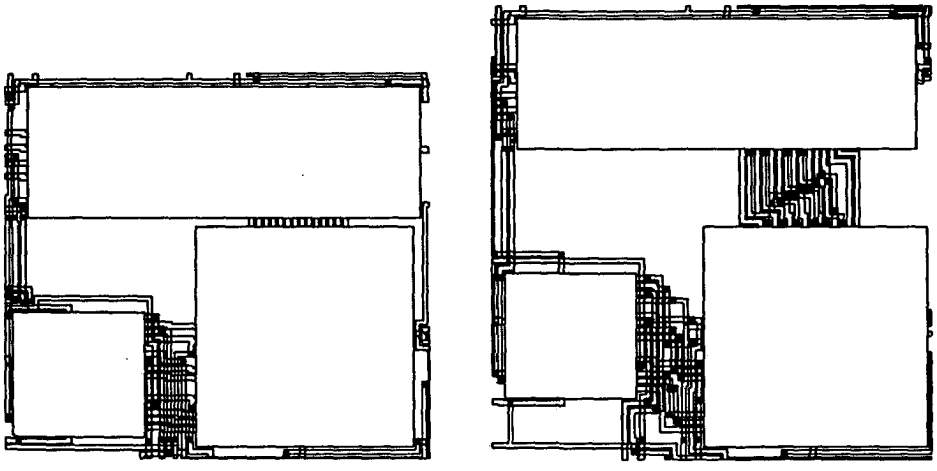


Figure 5-13. A chip example, left side: the algorithm was used; right side: the algorithm was not used.

5.5 Floating terminal ordering at channel ends

5.5.1 The floating terminal ordering problem

Routing congestion often occurs at the channel intersection area, which is often caused by an arbitrary ordering of the floating terminals at the channel ends. This is because the terminal ordering at the ends of a channel is determined by the channel router regardless of the global topology of the nets. This problem is illustrated in Fig. 5-14 where congestion occurs in the horizontal channel caused by the inconsistent orderings of the floating terminals

at the ends of the two vertical channels.

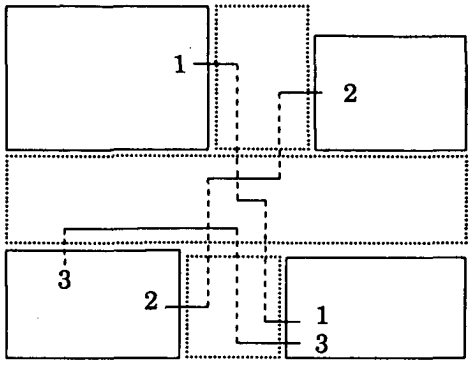


Figure 5-14. Wiring congestion in the horizontal channel caused by the inconsistent ordering of the floating terminals of the two vertical channels.

Given a channel structure the global routing specifies through which channels a net is connected. The actual geometrical realization of these connections are performed by the channel router which has only a limited view of the total layout. Obviously, there exist many different geometrical realizations for a giving global routing. To minimize the number of wire crossovers and unnecessary wire bends the nets should be routed as "planar" as possible. As the channels are routed one at a time the channel router has only information of connections within the current channel. Therefore we should synchronize the routing in the individual channels by enforcing a preferred ordering of the floating terminals on the channel ends from a global point of view.

A heuristic algorithm applied by the 'PI' system [Rive82] to obtain a wiring with as few wire crossovers as possible is the so-called "crossing placement" technique which places the floating terminals on fixed positions on the channel ends. If this approach is used two dimensional routers (e.g. switch

box router) are needed to performed the detailed routing. In the channel based routing systems the detailed routing is done by a channel router only, no fixed positions can be specified for the terminals at the channel ends. Therefore we propose a different approach which only specifies a relative ordering of the floating terminals at the channel ends. Guided by these orderings the channel router still can complete the wiring of the channel in a conventional way of channel routing.

We tackle the problem in two steps, a local step and a global step. In the local step a terminal ordering on each channel end is determined based on the connection information in the bar channel of the channel intersection. In the global step a terminal ordering propagation process attempts to keep a consistent ordering of wire bundles in the channels they pass.

5.5.2 Local terminal ordering

A channel envelope consists of two types of edges, edges adjacent to a block are called *block edges* or *hard edges*, and edges adjacent to a neighboring channel are called *channel intersection edges* or *soft edges*. On a hard edge the terminal positions are fixed while on the soft edges the terminal positions are not determined prior to the detailed routing.

In the local step a terminal ordering on each soft edge is determined so that the nets can be routed as planar as possible. Since the terminals on a soft edge do not have a fixed position yet, it is assumed that they are placed at the middle of the edge. We divide the terminals on a soft edge into three groups, a R group, a L group and a T group. The R group consists of terminals at the leftmost position of the nets they belong to. The L group consists of terminals at the rightmost position of their nets, and the T group consists of the other terminals on the edge which are terminals in a net between the two outmost terminals of the net. In the special case that the only two terminals of a net are located at opposite positions of the channel we assign them

to either the R or the L group but not to the T group. An example of such a division is shown in Fig. 5-15.

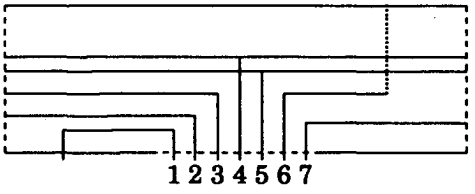


Figure 5-15. Terminal grouping on a soft edge of a channel. On the horizontal soft edge (dashed line) the L group consisting of the terminals 1, 2 and 3, the R group consisting of the terminals 6 and 7 and the T group consisting of the terminals 4 and 5.

The ordering assigned to the terminal groups on a horizontal edge is L, T and R from left to right denoted by L -> T -> R. Similar ordering assignments are applied to the soft edges of the vertical channels. Terminals in the T group are not further ordered, however terminals in the R and L group are further ordered. Multiple terminal nets are first reduced to two terminal nets by temporarily removing all terminals of a net except the leftmost and the rightmost. To find an order of the terminals on the soft edge the channel boundary is scanned to find the terminals which are connected to one of these terminals. If the channel boundary is scanned from one side of the edge to the other side and the direction is chosen consistently (e.g. clockwise), the order of the terminals encounted during the scan is the order assigned to the terminals on the soft edge. If two terminals are located on the same position (i.e. terminals on other soft edges) no mutual ordering is assigned to their corresponding terminals on the soft edge. For the example shown in Fig. 5-15 the terminal ordering on the horizontal soft edge is 1 -> (2,3) -> (4,5) -> 6 -> 7 from left to right. The order of terminals in a pair of parentheses is not constrained. Notice that terminals 2 and 3 are not mutually ordered because

they are connected to terminals on another soft edge. Only soft edges on the upper and lower channel boundaries are processed, but not on the channel ends. In this way the local terminal ordering on a channel intersection edge is determined by the wiring pattern in the bar channel of the 'T' channel intersection. In the next subsection we will see how the ordering preference of the stem channel can be taken into account.

After the local step there may still be some terminals in the R and L groups on a channel end with no mutual ordering constraint. These terminals are usually terminals of nets that only pass the channel. To keep a bundle of wires to run parallel and straight without unnecessary crossovers the terminal ordering of these nets on the channel intersection edges they pass should be kept consistently. This is the task of the terminal ordering propagation procedure in the global step.

5.5.3 Terminal ordering propagation

In this subsection a procedure is proposed to perform the terminal ordering propagation. With this procedure a consistent wire ordering of a bundle of wires can be obtained over the channels it passes.

The basic idea is to split a large bundle of wires into smaller bundles and to assign a consistent order to these smaller bundles over the whole length of the bundle. This procedure is applied recursively until a wire-to-wire ordering is obtained in each bundle. A *bundle* is defined to be a set of unordered wires in the channels. The locations where a bundle passes can be identified by finding a set of unordered terminals in the R or L group of the channel intersection edges.

The terminal ordering propagation algorithm processes the bundles from large to small. The largest bundle on the chip is found by searching for the largest set of unordered terminals in the R or L group on the channel

intersection edges. From the channel intersection edge where this set is found adjacent channel intersection edges are searched whose L or R group also contains the same unordered set. When the search terminates a chain of channel intersection edges are found. This chain of channel intersections is called the *path* of the bundle.

The two channel intersection edges where the path terminates are called the *termination intersections* of the path. A path terminates at a channel intersection where the bundle splits into smaller bundles or the bundle branches from another bundle of the same nets. The first type of termination is called a *splitting termination* while the second type of termination is called a *branching termination*. The termination of a bundle can be identified as follows. If the path of the bundle comes from the stem channel of a channel intersection, then it is a splitting termination if the unordered set splits into smaller sets on the channel intersection edge. The ordering of the smaller sets is called the *termination ordering* of the bundle at that end. Otherwise it is a branching termination if the unordered set appears in the T group of the channel intersection edge. If the path of the bundle comes from the bar channel of a channel intersection and no further channel intersections can be traced which contain this unordered set, then a local search in the stem channel must take place to determine the type of termination. If the type is a splitting termination the termination ordering is also derived by the local search.

Based on the termination information of the bundle a bundle ordering is determined and assigned to all channel intersection edges on the path of the bundle. Three types of combinations of the two path terminations can occur: first, the two termination intersections are both splitting termination intersections; second, one is a splitting termination and the other is a branching termination; and third, both terminations are branching terminations. In the first case, the bundle ordering is set to the termination ordering of one of the

termination intersections. In the second case, the bundle ordering is set to the termination ordering of the splitting termination intersections. Finally, in the third case, an arbitrary order is chosen for the wires in the bundle. This bundle splitting procedure is repeated until all terminals in the R and L groups of the channel intersections are assigned an order. The procedure form of this algorithm is shown in Algorithm 5-2.

Algorithm 5-2: *Terminal ordering propagation algorithm*

While (there are unordered terminals in the R or L groups)
{
Find the largest bundle.

Trace the path of the bundle.

Determine the bundle ordering.

Propagate the ordering to the channel intersection edges on the path.
}

For the example shown in Fig. 5-14 the result of the terminal ordering propagation is shown in Fig 5-16.

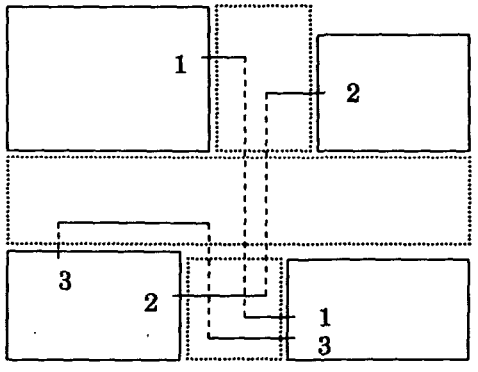


Figure 5-16. A terminal ordering propagation example.

In this example the local ordering on the channel intersection edge of the lower vertical channel and the horizontal channel is 3 -> (1,2) while the local ordering on the channel intersection edge of the upper vertical channel and the horizontal channel is (1,2). Hence the largest bundle is (1,2) and the path of the bundle consists of the two channel intersection edges. Both terminations are splitting terminations. Suppose we choose the termination ordering of the upper intersection edge as the bundle ordering which is 1 -> 2. This ordering is propagated to the lower channel intersection edge, so the ordering on this edge becomes 3 -> 1 -> 2. This gives the resulting layout in Fig. 5-16.

5.5.4 Correctness and complexity

An important property of the terminal ordering propagation algorithm is that it never introduces unnecessary wire crossovers. This can be justified easily. When a bundle is split into smaller bundles the algorithm always keeps the smaller bundles planar at least at one end of the bundle, because it always uses the termination ordering of a splitting termination as the bundle ordering whenever there is one. This can be symbolically illustrated. In the case that the two terminations are splitting terminations and both have the same termination ordering, then the smaller bundles are ordered planarly, see Fig. 5-17.

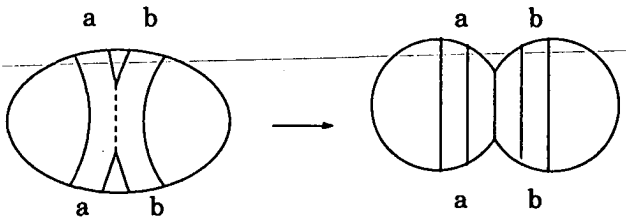


Figure 5-17. Planar bundle splitting.

In the case that the two terminations are splitting terminations and they have different termination orderings, since one of the termination ordering is

used as the bundle ordering the corresponding end of the bundle is kept planar. This implies that the wire crossings are pushed to the other end of the bundle, see Fig. 5-18. Hence, the bundle splitting operation only moves the unavoidable wire crossings in a certain direction. The case that one of the termination is a branching termination can be explained in a similar way.

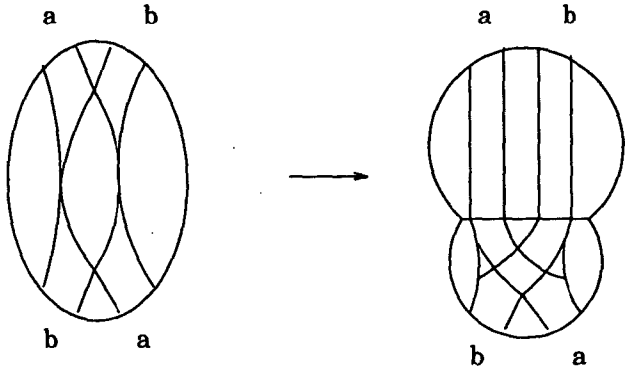


Figure 5-18. Non-planar bundle splitting, the wire crossings are pushed to one end of the bundle.

The complexity of the algorithm is in the order of O(mlog(n)) where n is the number of nets and m is the number of blocks, because in the worst case each bundle is split into only two smaller bundles, so the while statement can maximally have O(log(n)) iterations. Furthermore, each bundle can be processed in O(m) time because the number of channel intersection edges is in the order of O(m).

Notice that if a conflicting ordering occurs at the two terminations in the case that both terminations are splitting terminations a choice must be made to decide which of the two termination orderings will be used as the bundle ordering. This decision effects the locations where the wire crossovers are placed. A simple heuristic is to estimate wiring congestions in the two ends of the bundle and use the ordering of the most congested end as the ordering

of the bundle. This will push the wire crossings to the less congested end.

The floating terminal ordering procedure is most effective if it is used in conjunction with the relative block positioning procedure proposed in the previous section. This is especially so when there are many '+' type channel crossings in the floorplan graph, because the block positioning procedure can keep the two 'T' type channel junctions converted from a '+' crossing close to each other.

To achieve the desired goal, the local ordering on the channel intersections should be accurate. The accuracy of the local orderings depends on the accuracy of the width of the channels which in turn depends on the accuracy of the placement. This indicates once more that an accurate initial placement is important.

5.6 Channel routing

Channel routing is the problem of finding a physical wiring in a as small as possible channel using two or more interconnection layers given the envelope of the channel and a connection list. Since the length of a channel is usually fixed, the channel router will attempt to minimize the channel width. A good channel router is essential for a routing system, because it is the only part of the routing system where physical layout is generated. An example of a section of a routing channel is shown in Fig. 5-19.

Although we will not consider the channel routing problem in detail in this thesis, we will give a brief review of what is going on in this area, for the sake of completeness. The channel routing problem has been proven to be a NP-hard problem [Szym85]. Traditionally, a channel is a rectangular area in which two connection layers are used for wiring, one in each direction. Wires are placed on grids, i.e. on rows (or tracks) and columns. The first channel routing algorithm was proposed by Hashimoto and Stevens [Hash71]. Following them many channel routing algorithms have been proposed, see e.g.

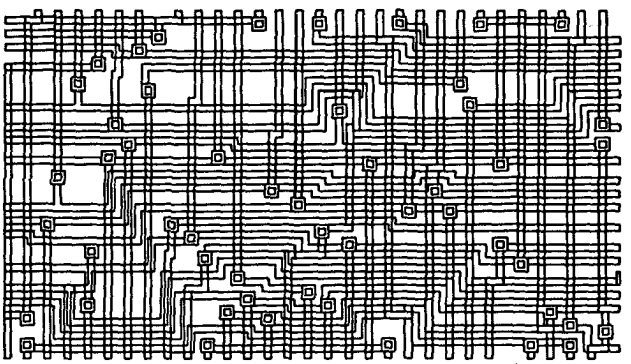


Figure 5-19. A channel routing example

[Deut76, Yosh82, Rive82a, Burs83, Reed85].

Recently, several gridless channel routers were reported, [Deut85, Chen86, Royl87, Ginn87] and the one we have developed in our system [Groe87]. These channel routers offer the designer much more flexibility compared to the grid-based routers, usually at cost of more computation time. A gridless channel router does not require terminals to be located on grid points. No columns and rows are used, only the wire width, spacing and via size are considered. In contrast to grid-based channel routers where the pitch of the grid has to be set to the "worst case" spacing design rules of all connection layers, the wires can usually be placed much closer to each other in gridless channel routers. Consequently, in processes where the via size is larger than the wire width, gridless channel routers usually need less area. Wires with variable width can be easily handled by gridless channel routers. This is especially useful for power and clock nets. Furthermore, channels are not restricted to rectangular shapes. Rectilinear shaped channels can be handled naturely.

The channel router in our system [Groe87] runs in two stages, contour routing and wire straightening. During contour routing a set of contours

along the two boundaries of the channel are maintained to protect the already routed wires. A new wire is bended as close as possible along the contour. This achieves minimum design rule spacing between routing geometries and provides good usage of the channel area. Once the contour router has fitted all nets into the channel the wire straightening procedure removes redundant jogs and makes the wires as straight as possible without widening the channel. This reduces the net length and the number of wire bends and helps improving chip yields. An important feature of this channel router is that the layer which is allocated to a preferred direction, can also be used in the other direction. This allows the channel router to solve most of the "cyclic vertical constraints" [Yosh82] without introducing doglegs and to minimize the number of vias.

This channel router can be easily extented to accept a preferred terminal ordering on the channel ends enforced from the outside. This can be realized, because the nets are routed sequentially during the contour routing, so the ordering of the nets can be adjusted to accommodate the preferred terminal ordering at the channel ends.

To reduce the terminal density at the channel intersection area another powerful extension to the channel router is to allow terminals to be located on the vertical edges of the channel boundaries (assuming the channel a horizontal). This feature is effective because when a channel is composed the two blocks adjacent to the channel usually have different sizes, which results in an empty edge on the channel boundary near the channel end. If we can move some floating terminals on the channel end to this empty edge the "bay" area in the higher-order neighboring channel can be utilized more effectively. An example of such a situation is shown in Fig. 5-20.
This feature creates terminals on vertical edges of the top horizontal channel which must be handled by the channel router. A simple way to route this channel is to use a preprocessor which transfers this channel into a

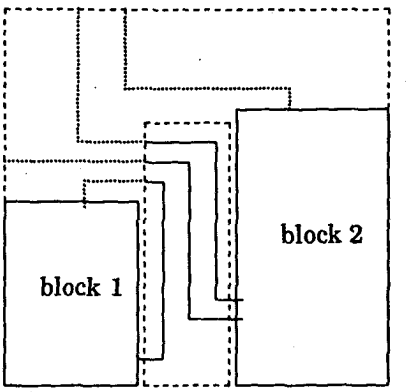


Figure 5-20. Terminals on the vertical edges of a channel

conventional channel by extenting the terminals on the vertical edge in the horizontal direction so that they can be separately reached in the vertical direction. The number of terminals one may place on a vertical edge depends on the routability of the neighboring channel.

5.7 References

Burs83. M. Burstein and R. Pelavin, "Hierarchical Channel Router," *Proc. 20th Design Automation Coference*, pp. 591-597 (1983).

Cai86. H. Cai and P. Dewilde, "Attacking the Problem of Minimizing Channel Density," *Proc. International Symposium on Circuits And Systems*, pp. 353-356, San Jose, USA (May 1986).

Chen86. H.H. Chen and E.S. Kuh, "Glitter: A Gridless Variable-Width Channel Router," *IEEE Trans. on CAD* 5(4) pp. 459-465 (October 1986).

Deut85. D. Deutsch, "Compacted Channel Routing," *Proc. ICCAD* , pp. 223-225 (1985).

Deut76. D. N. Deutsch, "A Dogleg Channel Router," *Proc. 13th Design Automation Conference*, pp. 425-433 (June 1976).

Fuku87. M. Fukui, A. Yamamoto, R. Yamaguchi, S. Hayama, and Y. Mano, "A Block Interconnection Algorithm for Hierarchical Layout System," *IEEE trans. on CAD* **CAD-6**(3) pp. 383-391 (May 1987).

Ginn87. L.P.P.P. van Ginneken and J.A.G. Jess, "Gridless Routing of General Floor Plans," *Proc. ICCAD*, pp. 30-33 (1987).

Groe87. P. Groeneveld, H. Cai, and P. Dewilde, "A Contour-Based Variable-Width Gridless Channel Router," *Proc. ICCAD*, pp. 374-377, Santa Clara, USA (1987).

Hash71. A. Hashimoto and S. Stevens, "Wire routing by optimizing channel assignment within large apertures," *Proc. 8th Design Automation Workshop*, pp. 155-169 (1971).

Kimu83. S. Kimura, N. Kubo, T. Chiba, and I. Nishioka, "An Automatic Routing Scheme for General Cell LSI," *IEEE Transactions on Computer-Aided Design* **CAD-2**(4) pp. 285-292 (October 1983).

Knij88. P. Th. Knijf, "CWE, A channel width estimation program," *Msc thesis, No. 88-86, Delft University of Technology, Dept. of EE*, (1988).

LaPa83. A. S. LaPaugh and R. Y. Pinter, "On Minimizing Channel Density by Lateral Shifting," *Proc. ICCAD Conference*, pp. 121-122 (1983).

Reed85. J. Reed, A. Sangiovanni-Vincentelli, and M. Santomauro, "A New Symbolic Channel Router: YACR2," *IEEE Transaction on Computer-Aided Design* **CAD-4**(3) pp. 208-219 (July 1985).

Rive82. R. L. Rivest, "The 'PI' (Placement and Interconnect) System," *19th design Automation Conference*, pp. 475-481 (1982).

Rive82a. R. L. Rivest and C. M. Fiduccia, "A 'Greedy' Channel Router," *19th Design Automation Conference*, pp. 418-424 (1982).

Royl87. J. Royle, M. Palczewski, H. Verheyen, N. Naccache, and J. Soukup, "Geometrical Compaction in One Dimension for Channel Routing," *Proc. Design Automation Conference*, pp. 140-145 (1987).

Szym85. T. Szymanski, "Dogleg Channel Routing is NP-Complete," *IEEE Transaction on Computer-Aided Design* **CAD-4**(1) pp. 31-41 (January 1985).

Yosh82. T. Yoshimura and E. R. Kuh, "Efficient algorithms for channel routing," *IEEE trans. on CAD* **CAD_1**(1) pp. 25-35 (Jan. 1982).

6. A PLACEMENT AND ROUTING SYSTEM

Most algorithms presented in this thesis have been implemented into a placement and routing system for VLSI building-block layout at the Delft University of Technology. It is called the Delft placement and routing system [Cai88]. In this chapter an overview of the structure of the system will be given. Some design experiences with the system and experimental data will also be presented. Finally, the author will attempt to give some directions of future developments in this area.

6.1 The Delft placement and routing system

6.1.1 The system components

The system has been written in the C programming language under the Unix operating system. The graphical part of the system is implemented on X window version 11. These choices have ensured high portability of the system. All major workstation vendors provide such a programming environment.

The system consists of both interactive and fully automated programs. Four modules constitute the system:

- IPP: an interactive placement program.
- GLROUTE: an automatic/interactive global router.
- SEQU: a detailed routing scheduler.
- GCR: a gridless channel router.

IPP is the front-end tool of the chip designer at the placement/floorplan level. It operates on a graphic terminal or a workstation and is menu-driven. It provides the designer facilities to view existing placements, channel structures and the connectivity between the blocks in order to judge the quality of

the placement. Commands are provided to create a new placement and to optimize an existing one. An example of a placement and the connectivity of a chip displayed by IPP is shown in Fig. 6-1 where the connectivity is represented by the spanning tree of each net.

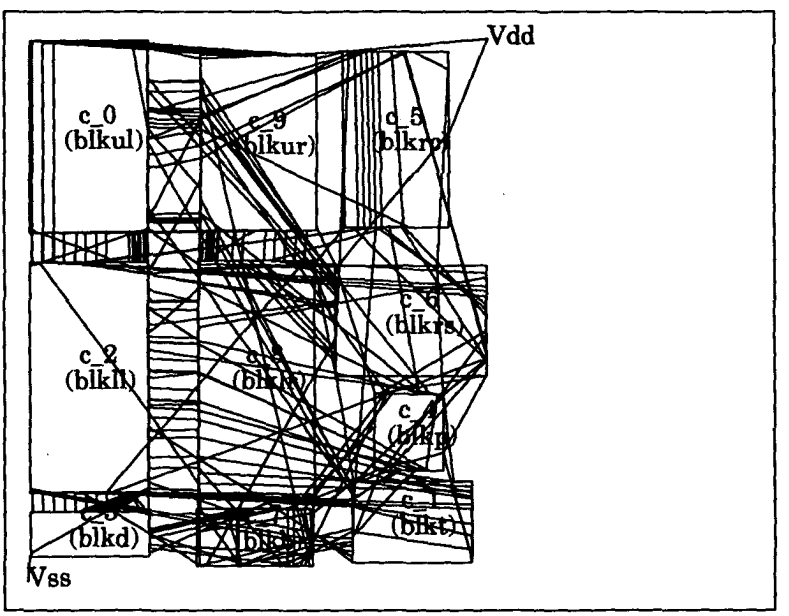


Figure 6-1. A chip floorplan displayed by IPP.

The global router GLROUTE performs the following tasks. Optionally it can perform the wiring space assignment according to the longest path algorithm to obtain a more accurate placement. Subsequently, it constructs an empty-room-free floorplan graph from the placement, performs the global routing on it and defines a conflict-free channel structure. As the last step, it planarly routes the two power nets and calculates the width of these nets in each channel. Furthermore, it provides a limited capability to control the different steps interactively.

The program SEQU takes care of the geometrical channel definition and the layout composition. Before a channel is routed SEQU first determines the optimal relative position of the two adjacent blocks. It then determines a preferred ordering of the floating terminals on the channel ends. When the channel envelope is defined it calls the channel router to perform the physical routing. The channels are processed in the channel order determined by GLROUTE. After a channel is routed the resulting channel width and terminal positions at the channel ends are returned by the channel router. Using this data SEQU generates a composite block which replaces the routed channel and its two adjacent blocks.

The channel router GCR is a powerful contour-based gridless channel router which can handle rectilinear shaped channels. Due to the gridlessness it puts very few constraints on the building block designers. Terminals on the blocks are not restricted to locate on grid positions. Terminals in the same layer can be placed at the minimum spacing distance of the layer. Terminals on non-interacting layers are allowed to overlap. Terminals and nets can have a specified width larger than the design rule required minimum. The current version of the channel router can handle up to four routing layers.

6.1.2 Supporting hierarchical designs

VLSI circuits are often designed hierarchically to master the complexity. On the top level of the layout hierarchy the chip core is connected to the *bonding pads* in order to communicate with the outside world. These pads have fixed positions on the chip boundary. In the routing system the wiring to the pads is handled in the same way as the wiring between the blocks. Four temporary blocks are created on the boundary of the chip, one at each side, containing the bonding pads. The bonding pads are modeled as terminals on these blocks. Figure 6-2 shows an example of the bonding pad blocks and the four outmost channels of the chip (the dashed lines). Note that the

four outmost channels are routed with a fixed channel width.

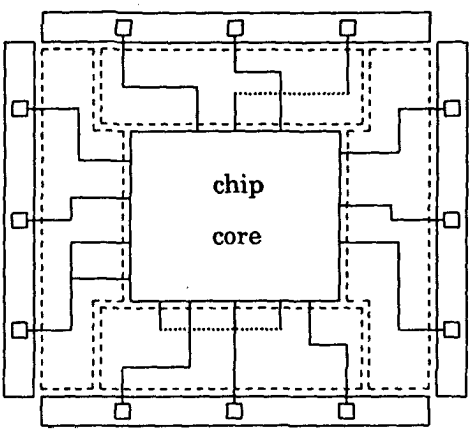


Figure 6-2. Bonding pad routing of a chip.

If the module to be routed is not on the top level of the layout hierarchy then it is not desirable to place the terminals of the module on fixed positions. In that case the length of the four outer blocks can be adjusted according to the size of the module core and the positions of the module terminals can also be adjusted to abut to the terminals connected to them on the module core in order to shorten the wire length, see Fig. 6-3. In this way when the routed module is used as a submodule on a higher level in the hierarchy, generally no zigzag long wires will be introduced despite the presence of adjacent channels running parallel to each other on different levels of the hierarchy.

6.1.3 Design rule independency

The system is fully design rule and technology independent. A design rule file which can be customized by the user is read by the system. The design rule file specifies the necessary fabrication rules for the wiring layers, for example, the minimum wire width, minimum wire spacing, the via sizes and the relationship between the current and the minimum wire width required

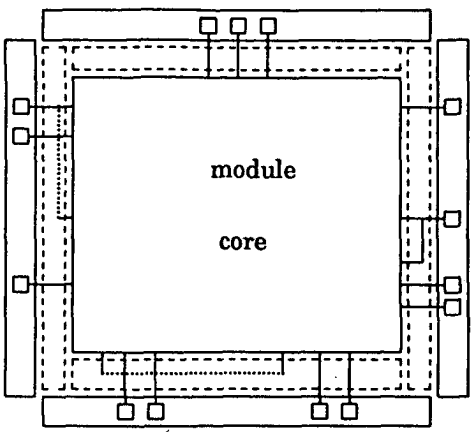


Figure 6-3. Terminal routing of a hierarchical module.

in order to prevent metal migration. Moreover, rules can be provided to influence the behavior of the system, for instance, the maximum length of high resistance layers (e.g. poly) the router is allowed to use. An example of a design rule file is given in Fig. 6-4.

6.1.4 Data management

The management of the large amount of data involved in the design of a VLSI chip becomes an increasingly important issue. Large number of tools operating at various design stages must be able to communicate with each other in an convenient way without using cumbersome translators. The power of a design system is the true integration of the design tools in the sense that the data is controlled by a unified method to keep the data consistency, to reduce the data redundancy and to maximize the data sharing. Without this integration individual tools are often useless to the designers. The placement and routing modules are integrated into the Nelsis IC design system which provides such an environment [Dewi86]. In this system the tools operate on a common repository of design data, the data base, while a

```
/*
 * Design rule file for a single metal NMOS process
 */

mask[poly] = np;                          mask[diff] = nd;
mask[metal1] = nm;

contactmask[poly] = nc;                   contactmask[diff] = nc;

spacing[diff][diff] = 2;                  spacing[diff][poly] = 1;
spacing[poly][poly] = 2;                  spacing[metal1][metal1] = 2;

width[diff] = 2;                          width[poly] = 2;
width[metal1] = 3;

contact[diff][poly] = 2;                  contact[diff][metal1] = 2;
contact[poly][metal1] = 2;

overlap[poly][metal1] = 1;                overlap[metal1][poly] = 1;
overlap[diff][metal1] = 1;                overlap[metal1][diff] = 1;
overlap[poly][diff] = 1;                  overlap[diff][poly] = 1;

length[diff] = 8;                         length[poly] = 50;
minlength[poly] = 30;                     minchanwidth = 0;

current[metal1] = w / 0.8;                power_line[metal1] = 0.8 * i;
```

Figure 6-4. An example of a design rule file.

data management system (DMS) stores and maintains the data [Wolf88].

The data management system stores the design data at different levels of abstraction into different *views* and maintains the consistency between design objects in these views. Currently, a circuit view, a floorplan view and a layout view have been implemented. The placement and routing tools operate mainly on the floorplan view. However, they obtain the netlist data from the circuit view and store the wiring output in the layout view. The transactions between the tools and the data base are handled via a data

management interface (DMI) which relieves the tool developers of the burden of a detailed understanding of the storage formats used by the DMS. This extra layer also decouples the tools and the DMS such that the tools will not be affected by any changes to the internal structure of the DMS.

6.2 Experimental and design results

6.2.1 Test chips

Throughout the development of the system a large number of sample chips have been tested over and over again. Many of them were originally obtained from the industry. In this subsection we will show the layout and statistics of three test chips of different sizes.

The statistics of a module in an image processing chip (chip1) and two test chips (chip2 and chip3) is shown in Table 6-1. Lengths are measured in micro meters.

STATISTICS OF THE TEST CHIPS			
	chip1	chip2	chip3
No. of cells	12	17	21
No. of nets	83	168	275
No. of terminals	352	541	1067
Chip area W × H	1672 × 1480	3815 × 3140	3470 × 3976
Wiring area / total area	62.63%	27.62%	33.33%
Total net length	52671	262906	474096
Total no. of vias	199	886	1813

Table 6-1: Statistics of the test chips.

The routing time ranges from 2 to 5 minutes on an Apollo DN-3000 workstation for these chips. Figure 6-5, Fig. 6-6 and Fig. 6-7 depict the layout of these three test chips.

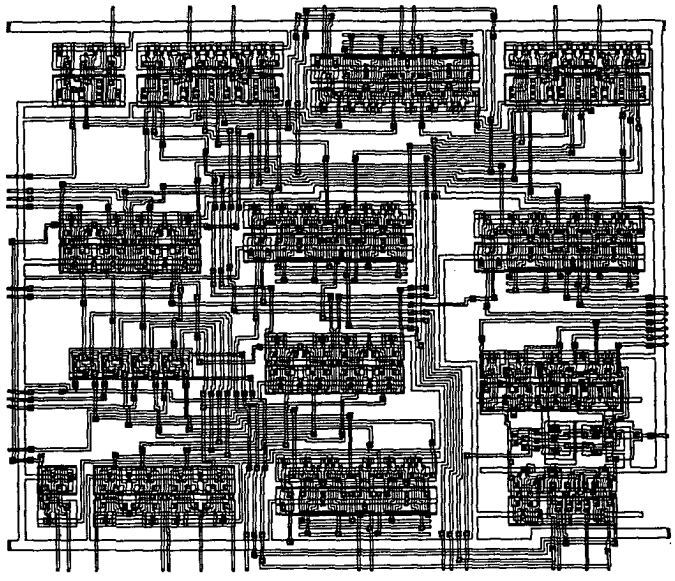


Figure 6-5. Chip1, a module in an image processing chip.

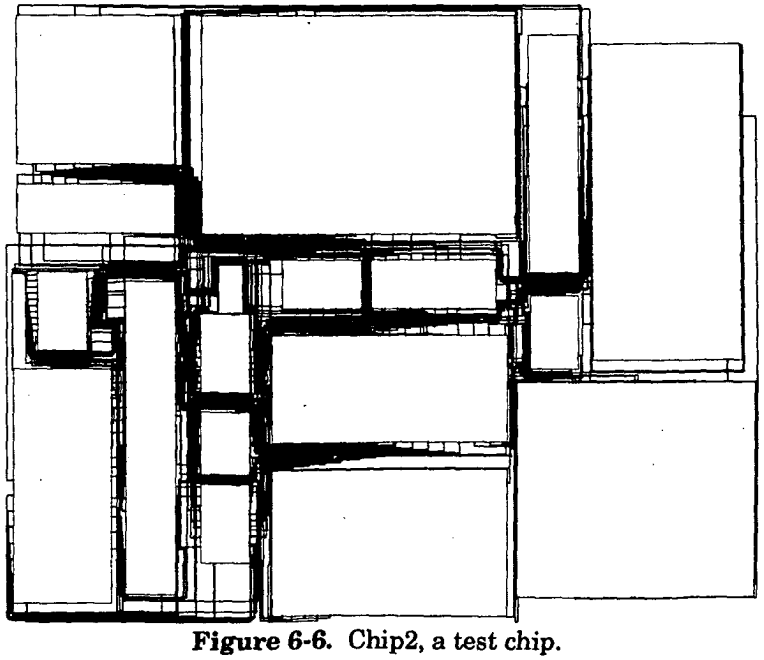


Figure 6-6. Chip2, a test chip.

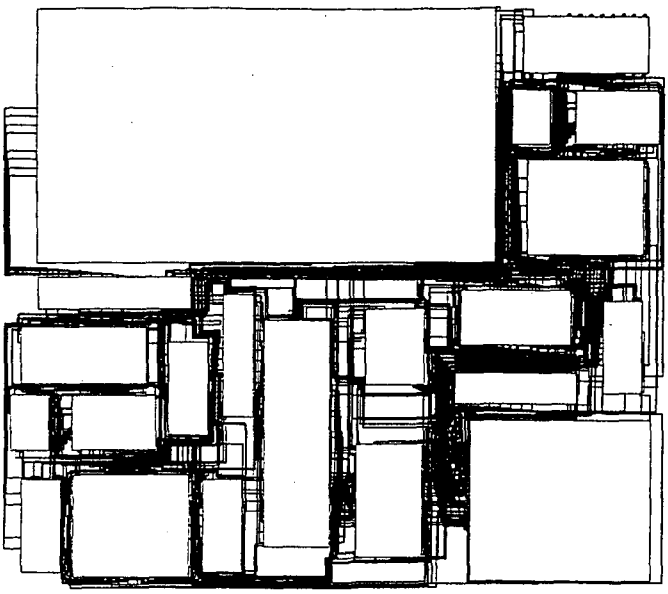


Figure 6-7. Chip3, a test chip.

6.2.2 Real chips have been designed

Two large VLSI chips have been designed at the Delft University of Technology of which the placement and routing has been produced by this system. One is a signal processing chip, the CORDIC processor (Coordinate Rotation Digital Computer) which is currently being tested. The chip has a pipeline architecture and contains more than 70,000 transistors. A photograph of the chip layout is given in Fig. 6-8.

The other one is a pixel processing chip, called CLP (Cellular Logic Processor) containing about 30,000 transistors. This chip is currently being processed. Extensive simulation and design rule checking have verified the correctness of the layout generated by the routing system. The feedback from the designers has provided valuable suggestions in improving the quality and the functionality of the software.

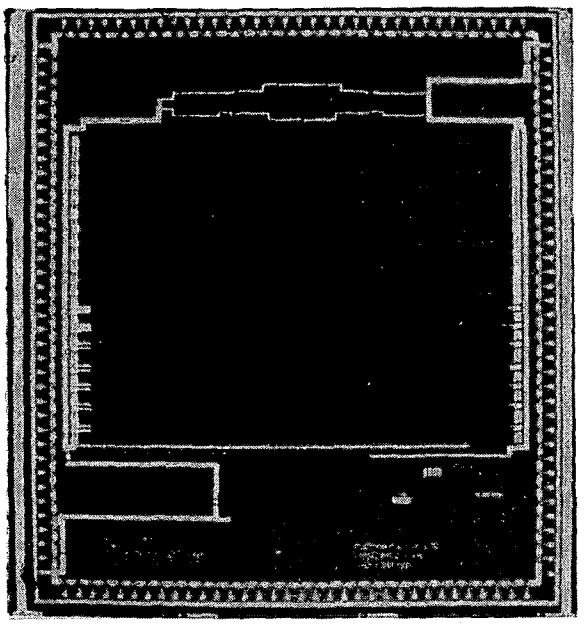


Figure 6-8. Layout of the CORDIC chip.

6.2.3 The benchmark chips

To compare the Delft placement and routing system with other building-block layout systems we participated in an international benchmark competition at the International Workshop on Placement and Routing held in MCNC (Microelectronic Center of North Carolina), Research Triangle Park, North Carolina, USA in May 1988.

Two building-block chips were compared, a 10 block, 203 net chip named Xerox and a 33 block, 123 net chip named Ami33. They were distributed to the participants prior to the workshop. The results of the competition are shown in table 6-2.

From the table we can conclude that the Delft system has achieved the best results for both chips. We should point out that the total wire length is not trivial to measure and may contains some inaccuracy. The competitors were

1988 MCNC BENCHMARK COMPETITION RESULTS						
system	Ami33			Xerox		
	area	wire length	# vias	area	wire length	# vias
DELFT P&R	2.60	151656	967	26.57	615104	925
SEATTLE	2.94	125000	862	28.63	762000	1235
MCNC_MG	3.12	134599	763	31.71	865712	1029
MOSAICO	3.16	151824	885	29.01	650009	1173
BEAR	3.05	131244	1092	28.47	633494	1101

Table 6-2. 1988 MCNC benchmark competition results.

the BEAR [Dai87] and MOSAICO [Bear88] system from the University of California at Berkeley, the MCNC_MG system from MCNC, and an industrial system from Seattle Silicon. The layout of the two chips produced by the Delft placement and routing system are shown in Fig. 6-9 and Fig. 6-10.

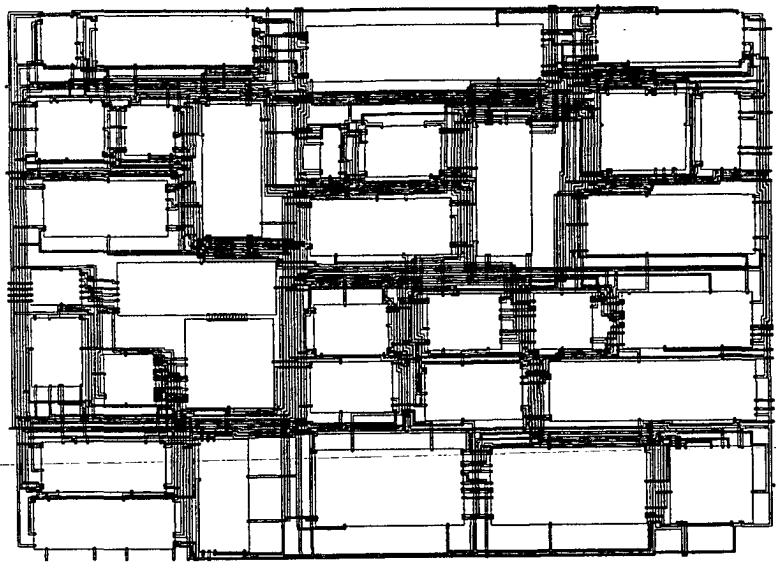


Figure 6-9. Layout of the benchmark chip, Ami33.

6.3 New Challenges

With the continuous advancing of the IC technology design methods and tools must also be continuously improved. To conclude the thesis, in this

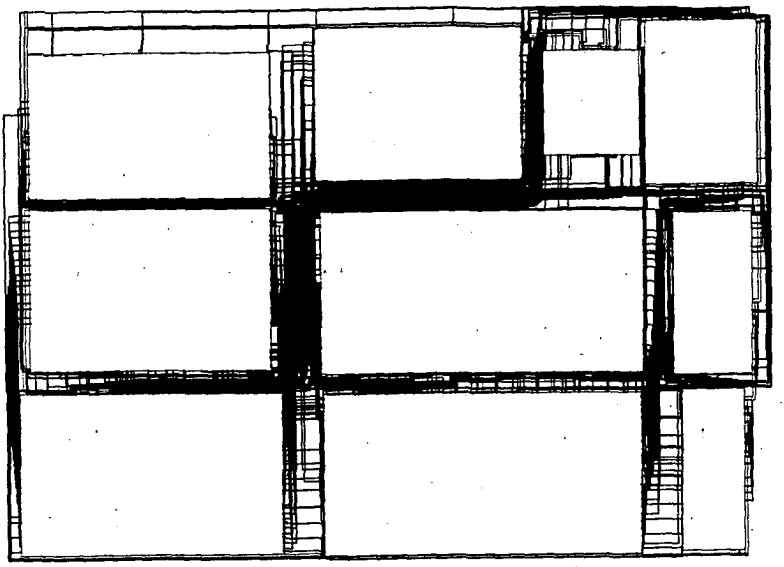


Figure 6-10. Layout of the benchmark chip, Xerox.

section we look at a number of topics which are expected to attract more attention of the researchers in the near future.

6.3.1 Multi-metal technology

With the availability of multi-metal processes more sophisticated over-the-cell routing strategies are required. Since the layout of most cells is realized using only one metal layer, interconnection wires in the other metal layers are allowed to cross over the cells. Hence, a region on each cell can be defined for each metal layer in which interconnection wiring in the layer is allowed. The router should be able to exploit these regions efficiently to minimize the channel area and the net length.

If only one metal layer can be used for the over-the-cell routing, the choice of the direction of the wires over the blocks is important, because once the direction of a wire is chosen it prohibits other wires in the same layer to cross

over the block in the perpendicular direction. The channel based routing approaches can still be used but must be enhanced to incorporate the routing regions on the blocks. Beside the need for multi-layer channel routers, the task of the global routing becomes more complicated. Furthermore, a different power net routing strategy is also required. Although the planarity is not the primary criterion in multi-metal chips an efficient layer assignment of the power wires is still important to achieve an optimal routing result.

6.3.2 Timing considerations

In building block based circuits the delay time for each signal path incorporates both the cell and the interconnect delays. With increasing chip area and decreasing active device sizes, the delay on the interconnections has an increasing effect on the circuit performance. In fact, in modern VLSI circuit the dominant delay time is the time to send the signals down the connection wires, not in the active components. As a consequence, the effect of interconnect delay on the circuit performance becomes an increasingly important design consideration.

Most of the work done on layout has been concentrated on optimizing area. Coupling timing analysis with placement and routing will become a necessity for the future layout systems. Advanced layout systems must take timing considerations into account during each layout synthesis stage [Teig86]. This implies that beside the geometrical considerations each layout tool should also use the timing criticality in making decisions. Early and frequent feedback from the analysis of how the layout is progressing towards meeting the timing requirements allows the designer to maintain close control over the performance of the design and to make small adjustment where necessary.

6.3.3 Expert systems

The unstructured nature of some of the layout problems makes them difficult to tackle by algorithmic approaches. In fact, human design experts often produce superior layout compared to the automatically synthesized layout. Human beings are particularly strong in solving two dimensional problems. They rely on experience, intuition and understanding of geometrical relationships to complete the task. They are able to consider different aspects of the problem at the time. With the advance of the Artificial Intelligence technology, especially in the area of knowledge based expert systems, it becomes feasible to code human expertise into a computer. Such systems attempt to capture the style and expertise of the human designers and provide flexibility to deal with a wide range of design applications. It is generally expected that expert systems will make a significant contribution towards solving various difficult CAD problems.

A knowledge based system in the form of an expert system consists of three major components: a knowledge base, composed of a set of production rules encoding the human expertise, a working memory presenting the current state of the system and an interpreter (also called an inference engine) which controls the system's activity. The rule-based approach has the advantages that it is modular, so that rules containing the domain knowledge can be added, deleted or modified without directly affecting other rules, and it is uniform in structure with all knowledge being encoded in the same constrained syntax [Wins84, Myer86, Haye84].

Progress has already been reported in using expert systems to solve layout problems, for example, in the module area estimation [How86], in floorplanning [Wata86, Yu87, Birm85], in global routing [Cai87] and in detailed routing [Joob85]. Many more exciting applications of expert systems are expected to come.

6.4 References

Bear88. M. Bearslee, J. Burns, A. Casotto, M. Igusa, F. Romeo, and A. Sangiovanni-Vincentelli, "Mosaico: An Integrated Macro-Cell Layout System," *International Workshop on Placement and Routing, MCNC*, (May, 1988).

Birm85. W.P. Birmingham, R. Joobbani, J. H. Kim, D. P. Siewiorek, and G. York, "CLASS: A Chip Layout Assistant," *Proc. ICCAD*, pp. 216-218 (1985).

Cai88. H. Cai and P. Groeneveld, "Delft Placement and Routing System User's Manual," The Nelsis IC Design System Documentation, Delft University of Technology (1988).

Cai87. H. Cai, "A Rule Based Global Routing Optimizer," *Proc. International Symposium on Circuits And Systems*, pp. 43-46 Philadelphia, USA (May 1987).

Dai87. W. M. 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," *Proc. ICCAD*, pp. 34-37 (1987).

Dewi86. P. Dewilde, ed., *The Integrated Circuit Design Book, Papers on VLSI Design Methodology from the ICD-Nelsis Project,* Delft University Press, Delft, The Netherlands (Jan. 1986).

Haye84. F. Hayes-Roth, D. A. Waterman, and D. B. Lenat (ed), *Building expert systems,* Addison Wesley, Reading , MA, USA (1984).

How86. M. M. How and B. Y. M. Pan, "AMBER - A Knowledge-Based Area Estimation Assistant," *Proc. ICCD*, pp. 180-183 (1986).

Joob85. R. Joobbani and D. P. Siewiorek, "WEAVER: A Knowledge-Based Routing Expert," *Proc. 22nd Design Automation Conference*, pp. 266-

272 (1985).

Myer86. W. Myers, "Introduction to Expert Systems," *IEEE Expert*, pp. 100-109 (Spring 1986).

Teig86. S. Teig, R. L. Smith, and J. Seaton, "Timing-Driven Layout of Cell-Based ICs," *VLSI SYSTEMS DESIGN*, pp. 63-73 (May 1986).

Wata86. H. Watanabe and B. Ackland, "Flute - A Floorplanning Agent for Full Custom VLSI Design," *Proc. of the 23rd DAC*, pp. 601-607 (1986).

Wins84. P. H. Winston, *Artificial Intelligence (second edition)*, Addison-Wesley Publishing Company (1984).

Wolf88. P. van der Wolf and T.G.R. van Leuken, "Object Type Oriented Data Modeling for VLSI Data Management," *Proc. Design Automation Conference*, pp. 351-256 (1988).

Yu87. M. L. Yu, "Fork - A Floorplanning Expert for Custom VLSI Design," *Proc. ICCD*, pp. 34-27 (1987).

ABOUT THE AUTHOR

Hong Cai was born in Changsha, People's Republic of China, on May 26, 1961. In 1978 he graduated from the Eleventh High School in Changsha, China. After a short stay of 3 monthes at the University of Science and Technology of China in Hefei, China, in February 1979 he came to the Netherlands for a technical study course.

From March to September 1979 he followed an introductory Dutch language course in Delft, the Netherlands. He then started the study of electrical engineering in September 1979 at the Delft University of Technology. In November 1984 he received the Ingenieurs' degree (the equivalent of an M.Sc.) from the Department of Electrical Engineering, Delft University of Technology.

In December 1984 Mr. Cai joined the Laboratory for Network Theory of the Department of Electrical Engineering at Delft University of Technology, where he embarked on his work towards a Ph.D. degree doing research on placement and routing algorithms for VLSI circuits under the supervision of Prof. dr. ir. P. Dewilde. Since the beginning of 1988 he has continued his research under the supervision of Prof. dr. ir. R.H.J.M. Otten at the same university.