Structured layout design

Eindhoven University of Technology
Netherlands

Department of Electrical Engineering

Structured Layout Design

by

R.H.J.M. Otten

Eindhoven University of Technology Research Reports
EINDHOVEN UNIVERSITY OF TECHNOLOGY

Department of Electrical Engineering
Eindhoven The Netherlands

STRUCTURED LAYOUT DESIGN

By

R.H.J.M. Otten

EUT Report 80-E-111
ISBN 90-6144-111-0

Eindhoven
October 1981

CONTENTS

Summary

This report is a concise survey as well as an exposition of ideas about automation of layout design. In the first part the state and position of this part of CAD is considered. The central part of this report is a discussion of imperatives of a layout design system suitable for VLSI. Of course, such a system has to take account of the embedding into an integrated design system. However, layout design faces two other major problems. One results from industry's ability to pack over 10,000 gate equivalents into a single chip. Beside this increase of complexity today's micro-electronics technology made a variety of processes - each with its own set of design rules - available for integration. Diversity has been existing for a long time, but complexity raised the problem, since development of efficient systems for designing complex systems is costly and time-consuming. Layout design shares the complexity problem with any other design task. From the proliferation of different device technologies layout design seems to suffer most heavily. The last part of this report is a precursory presentation of an approach striving for conformance to the imperatives of the second part.

Otten, R.H.J.M.
STRUCTURED LAYOUT DESIGN.
Eindhoven University of Technology, Department of Electrical Engineering, Eindhoven, The Netherlands, 1981.
Eindhoven University of Technology Research Reports,
EUT Report 80-E-111.

Address of the author:

Dr.ir. R.H.J.M. Otten,
Automatic System Design Group,
Department of Electrical Engineering,
Eindhoven University of Technology,
P.O. Box 513,
5600 MB EINDHOVEN,
The Netherlands

Presently on leave with: IBM Thomas J. Watson Research Center,
P.O. Box 218,
Yorktown Heights, NY 10598,
U.S.A.

STRUCTURED LAYOUT DESIGN

1. Computer aided layout design

A layout of a system is any set of data that uniquely specifies the masks necessary for integrating the system. The layout tasks adressed in this report are those in which components or subsystems of fixed or variable shape have to be arranged within a given or as small as possible geometrical figure and to be interconnected by a network of conducting paths embedded in one or more layers while giving due consideration to technological, electronic and economic constraints.

1.1 Its context

1.1.1 Data base considerations

The evolution of silicon technology over the past decade has been so rapid that the development of computer aids could not maintain pace with it. Existing design methods cannot cope with the presently feasible scales of integration. Many CAD-tools are outdated and some projects for developing new ones were already obsolete before completion. Layout in particular seems to be destined to make a bottleneck in the design cycle. The Intel 8086 microprocessor, for example, required thirteen manyears merely for layout design [LAT79]. Yet it cannot be regarded as an isolated problem. Anyone in VLSI design must endorse Brooks' assertion that conceptual integrity is the most important consideration in system design [BRO75]. From the first conception to the last test the design must be guided by well-coordinated ideas taking into account the affects a decision has on all future design tasks. During the design process the design is to be stored as data on computers. So the integrity of a design is in fact the integrity of its data base. Thoughts connected with data base design should precede the program design for individual design stages. Questions like: "what data is needed, when is it needed, by which program?" should be answered. The answers will lead to a tentative data base configuration.

Design automation data can be divided into two types: design data and library data. The division is not based on a difference in logical or physical representation, but on how the data is utilized. *Library data* is utilized in a "read-only mode" by the program subsystems. The data is not changed during a design. It is accessed by pointer references and program subsystems may copy pertinent parts of the library. Library data can also be divided into two types: data stored in the master library and data stored in the user library. The *master library* is built, maintained and updated by a group of authorized people and protected against alterations by users. Many designs may reference data of this type. It typically represents standard components, complete with their simulation models and mask geometry. The *user library* contains data entered by the user and specific for his own design, for example, a layout structure defined by the user. *Design data* is the set of data that describes the actual state of the design. This set can also be divided into two classes: design data available to all program subsystems and design data exclusively pertaining to one particular program subsystem. The two classes are called *common design data* and *private design data* respectively.

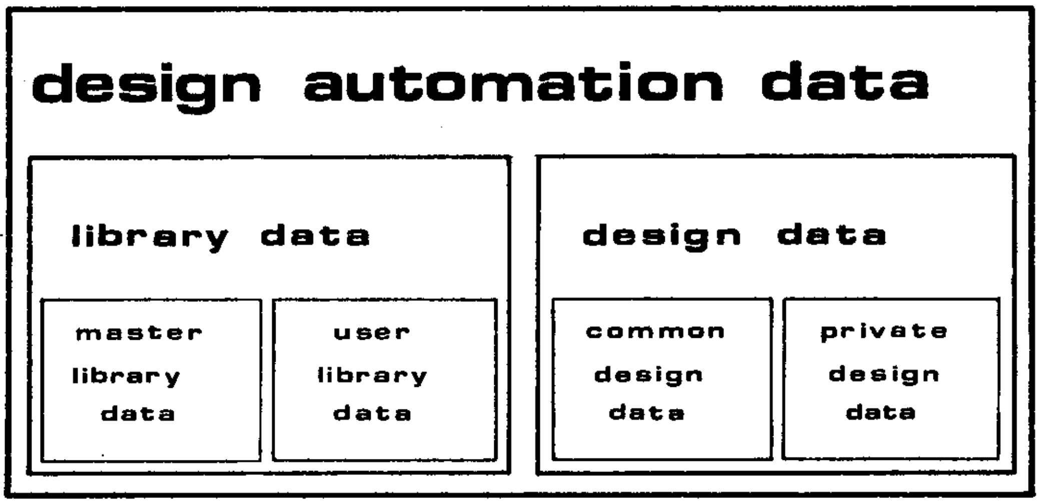


Figure 1.1: Interpretive division of data in a design automation data base

Circuit topology data is a typical example of common design data. It defines how modules - generic name for components and subsystems - are interconnected. These data are reflected in a structure known as the *potential graph* OTT76. It is a bipartite graph in which every module is represented by a *c-vertex*, and every signal (in layout literature "every net") is represented by a *t-vertex*. An edge indicates that a module represented by the incident c-vertex, and the signal represented by the incident t-vertex, have a pin in common. For now a *pin* can be seen as merely a mechanism relating modules to signals and reversely.

The notion of an incidence structure ^{DEM68} or hypergraph ^{BER70} is apparent if the structure is introduced in the following way. With each module a set of pins is associated. The set of electrically common pins is called a signal. So, consider the first set as a point, the signals as blocks, and the pins as flags. However, only a cumbersome concept is introduced this way, without a single advantage over the formulation in terms of conventional graphs.

Every program subsystem in an automated design system will utilize the portion of the data base that represents the structure of the potential graph. However, each will replace the modules by different models. The simulator will use a functional model adequate for its level of analysis. The layout design program needs geometries of masks.

The division of design automation data into the above defined classes induces a standard data flow for the program subsystems in the system. A subsystem interacts heavily with its private data base. It is profitable that the program has direct access to this data base. So, if possible, it will reside in primary storage devices while the concerned program subsystem is active. The data in the private data base is structured according to efficiency considerations derived from the specific task of the subsystem. This is not the case for the other classes of data. For any design of considerable size they have to be on secondary storage. Program subsystems extract required data from those data bases and, if necessary, restructure it and store it into their private data bases. After appropriate decisions are taken the common data base might be updated.

Up to now full automation of all design tasks has been unsuccessful or

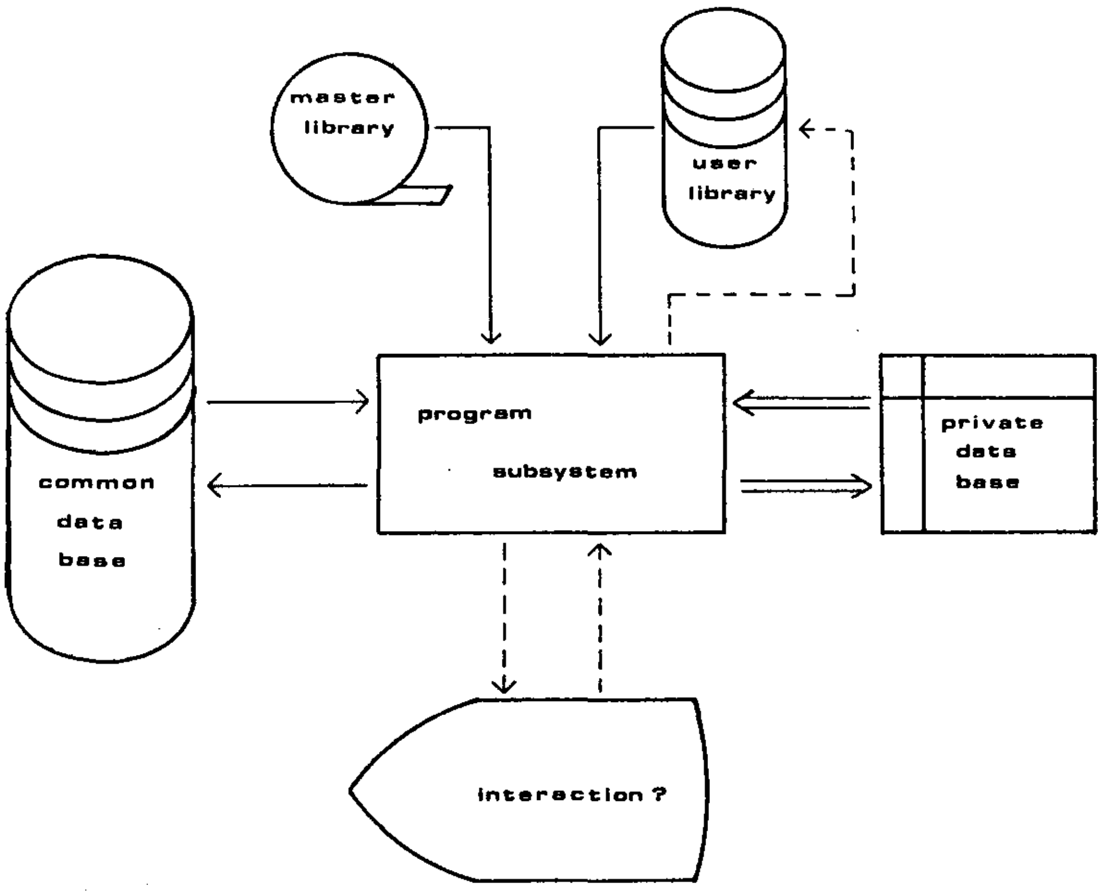


Figure 1.2: Masterplan of a data base configuration for program subsystems.

The comparison between a design system and a multi-story building has been presented at a symposium. The individual subsystems occupied a 'design' floor together with their private data base and, possibly, their interaction facilities. Communications with the common data base and the libraries, placed in the basement, were envisioned as elevators. Specification and supply could be localized on the main floor.

The symbols in the figure do not prescribe hardware; they only indicate relative accessibility even impossible. Most CAD systems therefore allow extensive human intervention. In that case users should be provided with the capability of restricting and delegating read and write access to the design data base. This blurs the distinction between the user library and the design data base and raises the problem of protection against concurrent and inconsistent updates. Integration of CAD tools is to a great extent hampered by these updating and protection requirements. Additionally, capabilities to display the data in a convenient way to users should be provided.

1.1.2 The design cycle

Figure 1.3 is an oversimplification (not an idealization) of the design of an integrated circuit. Automation of the integral design leads to numerous interface problems due to the enormous quantities of design data and design constraints. It stresses the need for a well-considered common data base. The consideration for the structure of this data base is to be derived from the design decisions to be taken and the requirement of efficient and reliable storing and retrieving relevant data.

Specification answers the problem of getting design data into the data base of the automation system. Graphical means have become more and more popular, but with the increase of complexity textual forms might surpass graphical specification in many automation activities. Attention has to be paid to the constraints forced upon the prospective user. His reluctance to use new tools has often been the insuperable problem of a CAD system.

Automatic synthesis is even not yet in its infancy. It only exists for very specific structures like PLA's. Nevertheless, the problem gets attention at some - mostly academic - places. It will certainly get more atten-

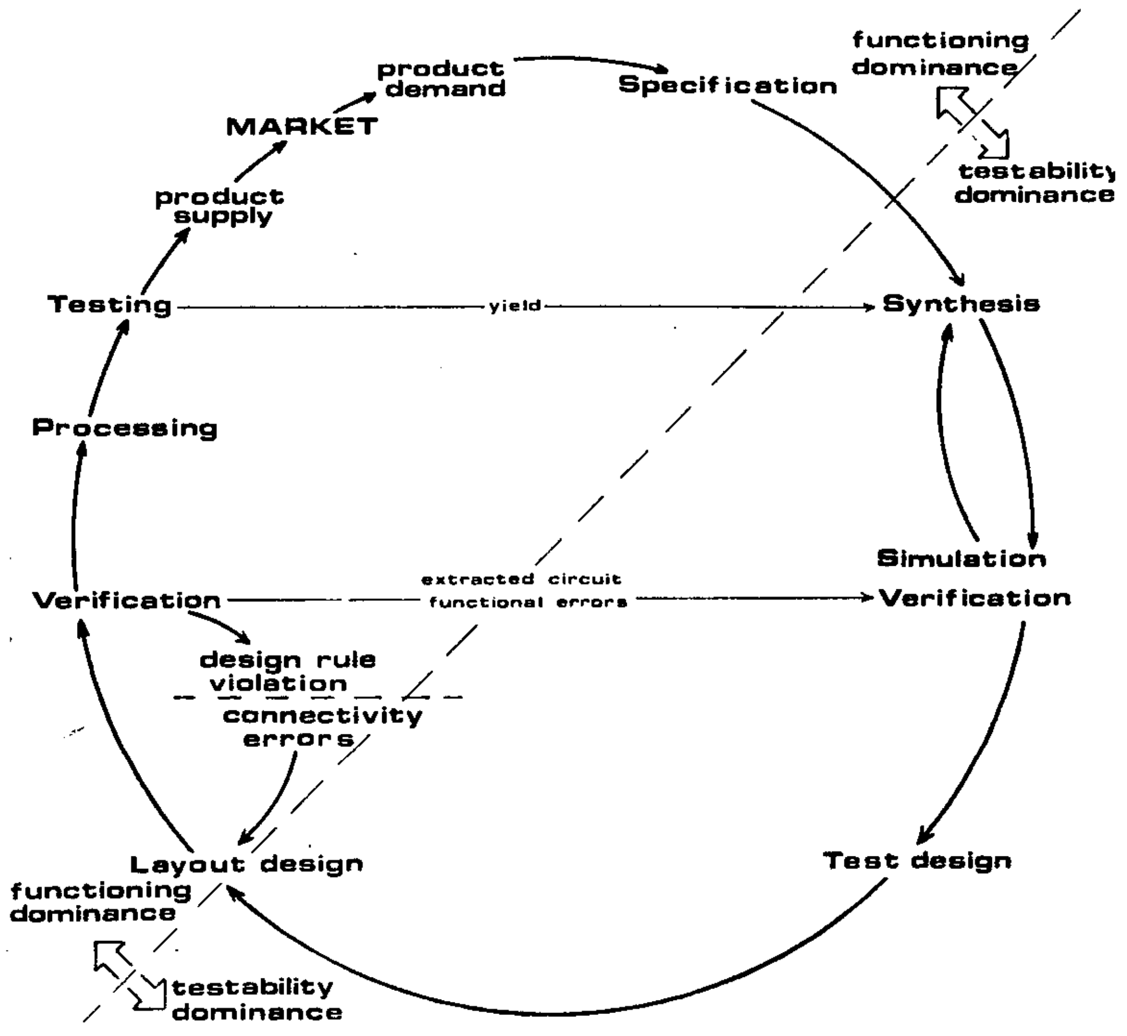


Figure 1.3: The design cycle. Half of the cycle is dominated by testability considerations!

tion in the future as the problem of producing correct designs is going to dominate all other cost factors.

Single-level simulation systems have gradually found acceptance among circuit designers in the seventies, but in the same decade their inherent limits were incurred. For complex circuits simulation at several levels is absolutely necessary, preferably simultaneously. Mixed-level simulation programs were the first answer to this need. However, more encouraging is the recent emergence of a transparent-level simulator which has a common approach to all levels while using a uniform data base [vBO81].

When the design is functionally specified down to the lowest level and the prediction of its operation and performance is satisfactory, the physical geometries have to be developed. This task is the topic of this chapter. The state-of-the-art will be briefly described in the next section.

Actual testing is separated from the other stages of the design cycle by the fabrication of the wafer. Current philosophies concerning testing inevitably lead to the conviction that testing should conceptually be related to the earlier parts of the design cycle. 'Design with testing in mind' is the accepted apothegm reflecting this conviction. The inability to develop a test method in line with this principle made testability to the most immediate problem of complex integrated circuit design.

The major activities in test design method development have been concerned with the gate-level. Those gate-level techniques are not attractive for large scale designs. For LSI circuits it is difficult to obtain suitable specifications, and when available, fault simulation and test generation programs turn out to be very expensive, and result in excessive test application times. Trends in logical design pose additional problems for which existing techniques are not adequate. Constraining the designer by testable design rule enforcement, such as the successful level sensitive scan design approach, only delays the awareness of the fact that present-day gate-level software cannot handle the immense volume of data to be processed, often demands impracticable modeling, and has a poor adaptability to technological evolution. More future seems to be in behavioral-level testing when a top-down design approach is adopted. With this technique testability analysis can be started at an early point in the design cycle. The volume of detail is often considerably reduced, the models are easy to prepare and to some degree independent of detailed realization, and not sensitive to technological changes. However, there is a lack of timing details. The break-through might finally be brought about by technological progress. Recently improved electron-beam techniques for the inspection of integrated circuits enhanced the observability of the design [WOL79]. Now it is possible to measure voltages at any point on the chip under test. Comparing response patterns extracted by these measurements with stored patterns of a standard model gives information about the presence of faults.

The impact of methods for increasing testability on layout design is not known. Of course, level sensitive scan design will increase the needed chip area, and the scan paths and shift lines will disturb the structure of the functional design. Layout techniques making use of such structure are degraded by these constraints, but not outdone. Electron-beam measurement might also affect the layout design stage, but probably any extra requirement can be hidden in a conventional set of design rules without

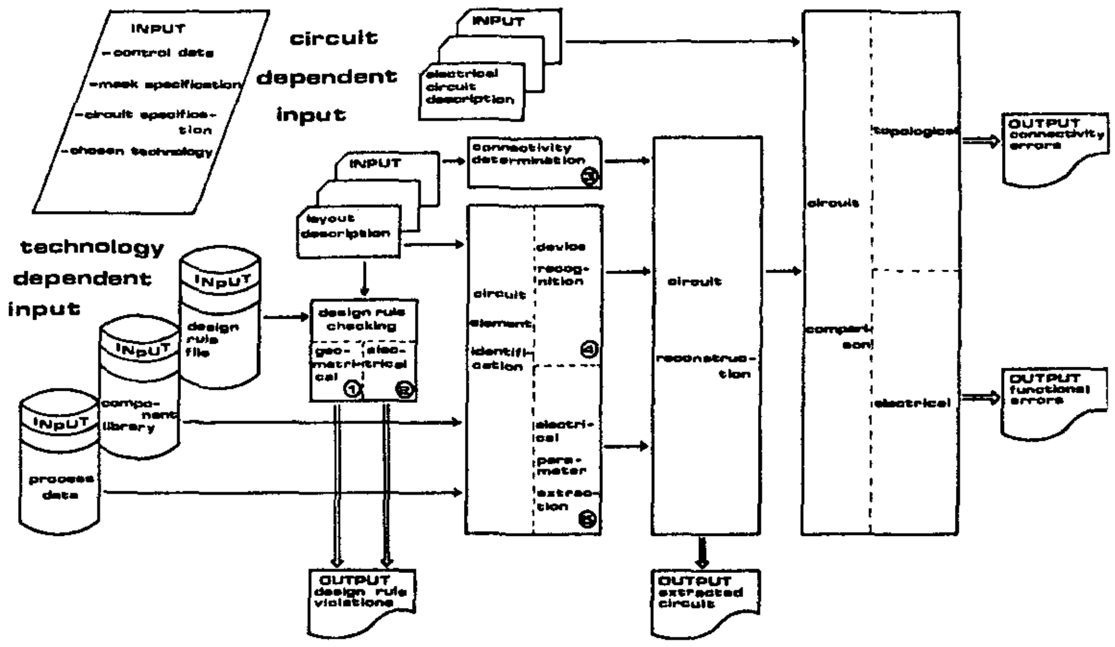


Figure 1.4: IC-layout verification

1. Geometrical design rule checkers determine conformance to layout rules such as minimum distances, tolerances, and overlap.
2. Electrical rule checkers search for illegal structures, such as crossing power lines, ground connections of individual components, cross-unders in power lines.
3. Connectivity checks trace the layout to determine which pins are connected to the same 'potential tree', to compare the result against an independent connection list.
4. Device recognition programs try to recognize components from the artwork features.
5. Electrical parameter extraction aims at determining of component parameters, load capacitances, coupling capacitances and many other parasitic elements introduced with the construction of the mask.

forcing a mutation in the methodology.

Checking layouts by an automatic layout verification system disencumbers people of the so-called eyeball hours, in which detailed computer-drawn plots (100 to 1000 times larger than the size of the actual circuit) are meticulously scrutinized to check for conformance to design rules. Some of these systems also extract electrical parameters (parasitics in particular) and verify whether the circuit behavior may still be expected to correspond to the intended behavior. The most heavily used automated aid in this class is the geometrical design rule checker. It measures certain geometrical relationships and checks by comparison whether and where the design rules of the concerned technology are violated. It is an indispensable tool when the layout of complex circuits is manually or interactively designed. However, existing programs have a number of serious drawbacks. First of all, their high average time and storage complexity. Secondly, many spurious errors are indicated. Thirdly, no program is yet capable of accommodating all design rules. Besides, existing algorithms are highly dependent on a restriction to orthogonal geometries which is presently not a bad trade-off, but with the advent of regular structures such as hexagonal arrays and technologies with more than two metallization layers current verification software is outdated. The other parts of a complete verification system are still in their infancy, and for custom design almost absent. Yet, it is of utmost importance that the artwork information is correct before offering it to the production department. Two answers are promising, both avoiding layout verification. One of them is symbolic layout combined with automatic compaction techniques. The other one is complete automation of the layout design task.

.1.2 Its present state

Initially, many of the computer aids developed for printed circuit boards were adapted for integrated circuit layout systems. This can be seen in many present-day designs where digitizers supporting manual design and placement-routing decompositions are still prevalent. In this section several approaches are characterized in an order of ascending degree of automation. In the second part of the paper some aspects of these approaches will be discussed in more detail.

1.2.1 Manual design with digitizer support

Problems in drawing highly precise artwork completely by hand made designers pass to digitizing techniques. A digitizer is a large back-lit drawing board which is connected to a minicomputer. Coordinates of each point the designer indicates on the board by means of a cursor or digitizing pen can be read into the computer on command. With a plotter artwork of the desired quality can be generated.

Often the configuration is combined with a cathode-ray-tube terminal. Beside data entry on-line error correction is possible with such a system. However, the decision about "what goes where" is still with the designer. No particular layout style is forced upon the designer if a style-dependent checking algorithm is absent. Widely used systems in this class are CALMA and APPLICON.

.1.2.2 Symbolic layout design

The earliest symbolic design systems substituted a set of symbols for the mask features. The designer manipulates these symbols observing a few simple rules for placement on a coarse grid. Though the designer still decides, the design is considerably faster at the cost of some restrictions on the layout style. Batch programming is feasible with these computer aids [GIB76].

With the coming of the dynamic color graphic display symbolic layout design evolved from an aid with rather incommodious alpha-numeric characters to one of the most promising approaches. The experience and cleverness of the designer is used for developing good layout topologies, since his task is only to obtain a relative placement and interconnection of symbols without observing hardly any design rule. An automatic program is capable of performing geometry transformation such as compaction and interconnection bending ("jogs"). The automatic program guarantees conformance to design rules which makes a design rule checker superfluous [HSI79].

1.2.3 Master slice approach

A master slice is a wafer processed up to the metalization layers. Each chip from such a wafer is identical as far as the kind and position of modules is concerned. Customization is only achieved through interconnection geometries. Computer aids in this approach are therefore routing programs. Full wiring completion is seldom automatically achieved and chances become very small when more than 80% of the functions on a chip are incorporated in the system.

1.2.4 Standard-cell

Functional cells of gate and register level are designed to conform to a common cell height and pin distribution, which often leads to non-optimal area utilization. These cells are to be placed in rows and interconnected through the intervening routing channels. The goal of placement as well as routing is to keep the channel widths as small as possible.

In principle no human interaction is required in layout design systems based on the standard cell approach. Simple designer intervention, however, appreciably enhances the placement techniques. Up to 500 cells standard cell programs perform very well, especially when design time dominates other cost factors such as yield, signal delay and power requirements. Its success was manifest for MOS-technology. The construction and maintenance of an up-to-date cell library has proven to be a significant overhead ^{PER77}.

1.2.5 Array layouts

Automatic generation of regular array structures such as programmable regular arrays from a functional specification such as a switching function is straightforward. To obtain high densities and small layouts functional minimization and decomposition techniques are applied. Though layout considerations are important, they are translated into terms consistent with these techniques, and therefore they are specific for the method of functional realization.

1.2.6 Building Blocks

There have been several attempts to solve the layout problem stated at the beginning of this chapter completely automatically. Starting from a functional circuit specification a layout has to be generated without any human intervention. Of some of the projects with such objectives successful completion has been announced, but acceptance in a production environment has not been reported. Besides, many results of these projects are of value only for some technologies.

More complex systems demand for more restrictions on the shape of the modules to be placed. Most current approaches restrict the shape to rectangles. One class of these approaches is referred to as the building-block method KAN76. The blocks are functional units with a predesigned layout within rectangular boundaries. The interior of a block is usually very efficiently packed, and the sizes and aspect ratios, therefore, are quite varied. Placement of these blocks in a rectangular area leaves many irregularly shaped areas unutilized. Consequently, building-block approaches often yield sparse layouts. Most programs of the building-block type separate placement and routing which even more degrades the area utilization.

1.2.7 Other computer aids

Many subtasks of certain layout styles have been developed. One of the most important aids, certainly for symbolic layout, but also in many other approaches, is compaction. The intention of compaction algorithms is to squeeze layouts to reduce the amount of 'dead area'.

Wirability prediction programs have been developed for master slice layouts HEL78. Such-like programs for other layout styles will become important in the future, because the area consumption by interconnections grows very fast with the increase of complexity. As early as possible during the design estimation of local wiring areas is important. Since not much is known in that stage the programs will be probabilistic in nature. As more information becomes available the estimations have to be revised to guide placement decisions with as much information as possible.

The classical serial decomposition of the layout design task has three parts: partitioning, placement and routing. The latter two have got much attention in literature.

Placement algorithms got a definite treatment in HAN72. The first routers were mainly versions of a breadth-first search algorithm on a grid ^{LEE61}. They work on a one-connection-at-a-time basis. If there are solutions in a particular stage the shortest among them will be found. However, look-ahead to avoid unnecessary blocking of future connections is difficult to implement. Later, many other grid routers were published, sometimes with quite original solutions like determining the area to be etched instead of the area to be covered by metal and amoebic movements to establish the routes. Those algorithms, however, only perform efficiently in labyrinth-like situations. For VLSI circuits storage complexity will inhibit application. Also grid-free routers have been developed. Especially successful is the line search router ^{HIG69}, which is considerably faster than 'wave front routers'. However, there is no guarantee for finding a path even if it exists, but it can be modified to abolish this defect. The algorithm works with two sequences of escape points from which horizontal and vertical line probes are started. The first two points are the pins to be interconnected. When probes of different sequences intersect the search is ended, and the route is reconstructed with the line segments between the escape points.

Though the line search router is also used for LSI circuits ^{LAU80}, the most successful routers for complex circuits are channel routers. The problem is decomposed into independent routing problems in small rectangular areas with pins on two opposite sides. The routes consist of vertical and horizontal pieces to be realized in at least two different layers. The subtasks are reduced to easy combinatorial problems that can be fastly solved without heavy memory requirements. The nets have to interconnect certain pins on the sides and may create points on two other sides to leave the channel area.

The first channel router ^{HAS71} has been labeled as the unconstrained left-edge algorithm. The wire segments are considered to be intervals [L,R] and a partial ordering is defined over the set of intervals ($[L_1, R_1] < [L_2, R_2] \Leftrightarrow R_1 < L_2$). Each actual track is subsequently filled with an unplaced interval having the lowest L greater than the preceding R. If no interval satisfies those conditions a new track is initiated until all intervals are assigned to a track. If pins are restricted to grid coordinates and contacts are never exactly opposite the algorithm gives an optimal solution. Invalidating this condition introduces constraints on the track assignment which might be cyclic. These cycles have to be broken for example by manipulating pin positions. Optimal solutions have been published ^{KER73}, but these branch-and bound techniques are very time-consuming. More freedom is created by allowing nets to be realized in more than one track. This led to trunkdivision and 'dogleg' algorithms ^{PER77}.

2. Structured layout design

It has been noted before that the situation in VLSI design is to a certain extent comparable with the software crisis of the late sixties. From this period of confusion structured programming emerged as a systematic process for mastering complexity. It is therefore expedient to examine the principles of structured programming upon their relevance to layout design. The results of such an examination are interwoven in the following discussion.

2.1 The inevitable hierarchy

There is a conjecture that complex systems evolve far more quickly if they are of hierarchic nature than non-hierarchic systems of comparable size, and that aspects of complex systems that are not hierarchic even elude human understanding and observation. Both in nature and in science many instances support this conjecture SIM62. VLSI systems will be just new examples of systems exhibiting hierarchic structure whether they evolve from stable intermediate forms (e.g. a single-chip microcomputer which combines a number of functions that previously occupied separate chips) or by a practicable design discipline (still to be developed, but certainly top-down organized).

A hierarchic system or *hierarchy* is a system composed of interrelated subsystems, each of the latter being hierarchic in structure, until some lowest level of elementary subsystems is reached. The systems in a hierarchy are called *modules*. A hierarchy can be represented by a directed tree. Each vertex in this tree represents a module. An arrow is pointing from a module to its direct subsystems (*submodules*). The incoming arrow of a module refers to its unique *supermodule*. The root represents the whole system. The elementary subsystems are represented by the leaves of the tree. In several approaches to layout design the submodules of the whole system play a distinct role. In order to aid memory when this set of modules recurs, these modules are related to the rather eccentric figure 6.5 by naming them *cardinal modules*. The set of cardinal modules covers the whole system. None of the other layers in the hierarchy must have this property.

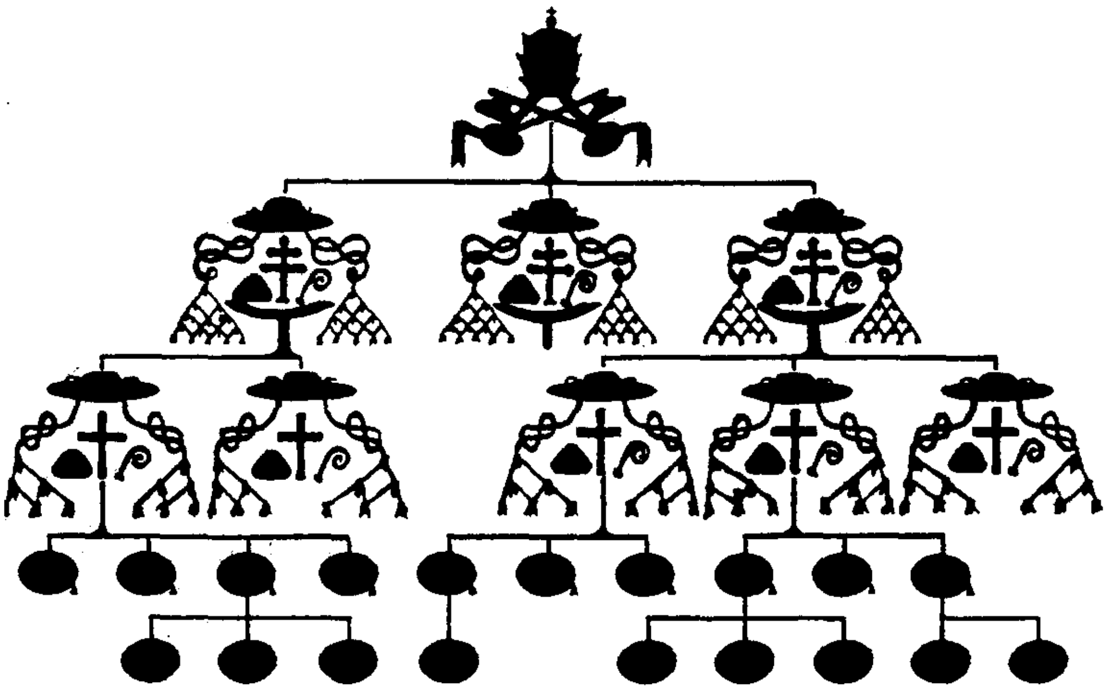


Figure 2.1: The etymological origin of the word hierarchy as an aid to memory when the notion 'cardinal modules' turns up

The hierarchy to be expected in VLSI systems is a functional hierarchy. The modules in this hierarchy realize partial functions of the system. These functions again are specified in terms of partial functions to be performed by lower modules except for the elementary modules. Thus, two kinds of modules are distinguished on the basis of this hierarchy: *cells*, units that are not divided into submodules, and *compounds*, units composed of submodules. Cells are the only technology dependent units of the system as far as their realization on the chip is concerned. The layout of certain cells is stored in a library, because of their frequent appearance. The layout of a cell may also be defined by the user. For both kinds of cells, *master cells* and *user cells* respectively, the layout is to be fetched intact from the place where it is stored, and inserted into the layout of a system. They are called *inset cells* to set them apart from *blank cells* of which the layout is to be determined by special technology dependent algorithms.

The functional hierarchy is to be supplied by the design system or by the designer. In the latter case the designer is constrained to make the inherent hierarchy explicit. Everybody in computer aided design knows how difficult it is to manage the introduction of a system with new constraints. When a complete system is delivered by the designer without an explicit hierarchy, it has to be partitioned on the basis of what seem to be reasonable criteria.

Partitioning is one of the classical problems in the physical realization of a system and it never was satisfactorily solved for systems for which it is needed most, namely large scale systems. When the system is complex, partitioning is a very complex task. The problems start already with the selection of the criteria. The most important consideration in decomposing a design is high block independence. Partitioning should therefore be organized in such a way that the relationships among blocks were minimized and the relationships among the elements of an individual block were maximized. In other words blocks should have a high internal strength and a low mutual coupling. The classical measure of block dependence in layout design is connectivity. It suffices for an after-the-fact judgement of the partitioning result. What is needed, however, is a guideline for producing an acceptable solution.

A partitioning method in use for building block approaches is min-cut placement GüN69. The acceptance can be explained by the principle of deferring detailed considerations as long as possible and the combination of partitioning and global placement. Each step a set of modules is partitioned into two blocks such that the number of signals common to both blocks is minimal. For this partitioning a modification of the Kernighan-Lin-algorithm KER70 is applied. This algorithm starts from an initial two-block partition, and improves this partition by interchanging the pair of elements which reduces the number of common signals most. This is repeated until all elements of one block have been involved in an interchange. The best intermediate result is taken. This gives a new two-block partition with which the procedure can be repeated. Possible modifications are concerned with excluding elements from the interchange operations, and taking module areas into account.

2.2 The structural restraint

In structured programming there exists the discipline to restrict control flow constructs to the Jacopini-structures BöH66. These structures are theoretically sufficient and ensure a straightforward mapping between the computational process and the program evoking it DAH72. Is there a similar rule concerning the structure of layouts as beneficial to layout design as the structuring principle is to programming? In the past many restrictions have been proposed with different degrees of success. Building blocks, standard cell, and bristle blocks are famed examples.

In the building blocks approach the only restriction is that the structure consists of abutting rectanglès. These rectangles must give room to a given set of layout problems with fixed shape. This problem has been translated to several mathematical models in order to apply known solution techniques. These translations mostly use a certain digraph representation of dissected rectangles BRO40. The name polar graph for these digraphs has found acceptance in layout design literature

A *polar graph* is an acyclic digraph with exactly one source and one sink which has a plane representation with the source and the sink on the same face boundary.

A rectangle partitioned into subrectangles, called a *rectangle dissection*, consists of two sets of parallel line segments, H and V. Any segment in H is perpendicular to any segment in V. To construct a polar graph associated with a rectangle dissection take either H or V as the set of vertices and connect two vertices if the corresponding segments contain sides of the same subrectangle. So, there is a one-one correspondence between subrectangles and arcs. The direction of the arcs must be consistent with position of the corresponding rectangle relative to the line segment.

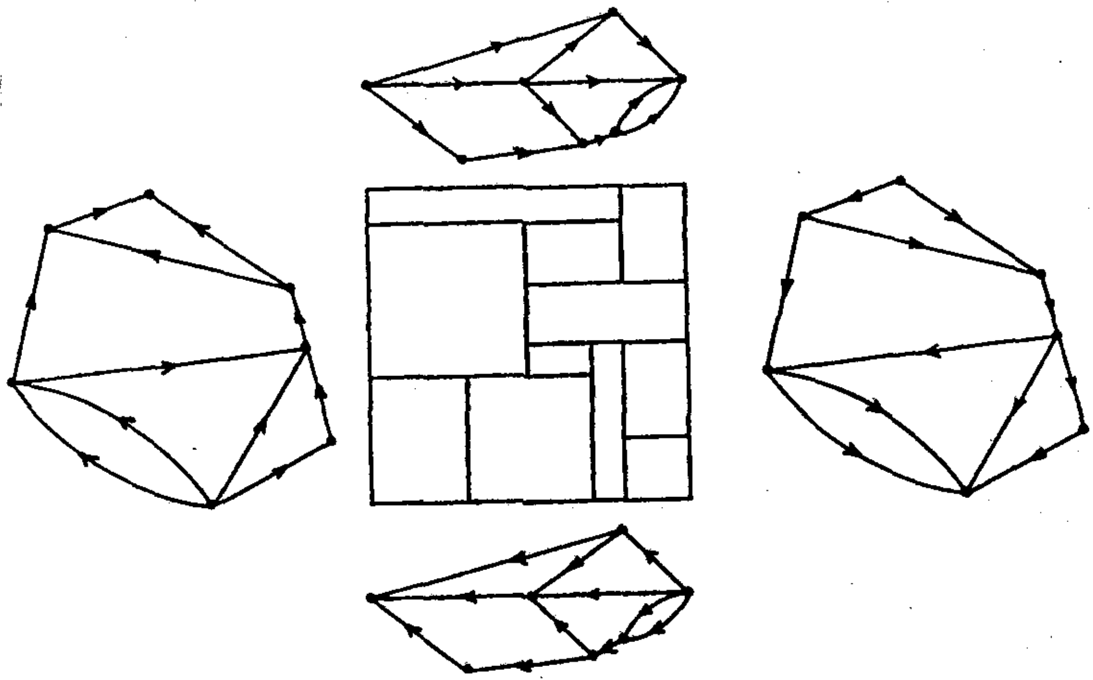


Figure 2.2: A partitioned rectangle and its associated polar graphs

Four, generally different, polar graphs are associated with each rectangle dissection. When taking H as the set of vertices the polar graphs are the same except for a reversal of all arc orientations. The same holds in case V is taken. The polar graphs with H-vertices are said to be the dual of the polar graphs with *V*-vertices.

When length and width of the subrectangles are assigned to the corresponding arcs a polar graph contains the same information as the rectangle dissection. These lengths and widths satisfy the Kirchhoff laws of netwerk theory. This observation is the key to many applications of this model in layout design. By formulating all other constraints as linear inequalities, the Kirchhoff equations and these inequalities form the simplex-tableau for minimizing the perimeter of the chip as a good approximation when the chip is kept from becoming very oblong [OTT76].

The modules to be placed have different, but fixed rectangular shapes and if there is a method to obtain a suitable polar graph the problem of minimizing the chip area can be formulated as a mixed-integer 0-1 linear optimization problem [ZIB74]. In the CALCOS system this formulation has been applied to LSI layout where the polar graph is obtained by a min-cut technique with alternately horizontal and vertical lines [LAU80].

In the layout style called *standard cell* or *polycell* the majority of the design rules are hidden in cells stored in the cell library. The cells have a rectangular outline. Two opposite sides of each cell have to be consistent with very strict rules: the same length and fixed pin positions for the nets common to all cells. By locating a cell alongside any other cell the corresponding pins are automatically interconnected. By arranging cells in parallel rows straight power, ground and clock lines are thus realized in each row. All other pins of a cell have to be located at the other sides. If both sides are used, cells are placed in single rows. If only one side contains individual pins cells are placed 'back-to-back' in double rows. Between the rows there are domains not occupied by cells. These domains are called *street channels*. By the arrangement of cells in rows as described all pins except those belonging to the common lines are facing a channel. These channels are used for realizing interconnections between pins. A net connecting pins at various locations may lie entirely within a single channel or interconnect more than one channel. *Avenue channels* perpendicular to the ones intervening the cell rows mostly contain the interconnections between the channels and between the common lines of each row. In order to avoid long interconnections *feed-through cells* are sometimes employed.

The actual layout stages of a standard cell design PER77 are:

1. Partitioning of the cells into rows by a crude global clustering algorithm or by a two dimensional placement consisting of a candidate cell selection on connectivity basis, an initial placement trying to minimize the total net length, and an iterative exchanging of cells to improve the placement taking into account net length, row capacity, etc.
2. Determining the sequence of cells within a row. The sequence of the cells influences the net length and the density of the channel. The local density of a channel is the number of interconnections that have to intersect the cross-section perpendicular to the cell rows at that spot. The *channel density* is the maximum value attained by the local density anywhere along the channel. From the channel density a lower bound on the channel width required to contain the associated interconnections can be derived. The major task of this stage and the next one is to minimize the necessary channel width.
3. Placement of the cells within a row. This is a rather straightforward task when the sequence of cells is known. However, the freedom left can be used to facilitate the tasks in the later stages of the design, in particular solving pin position conflicts.
4. Net decomposition and assignment of subnets to channels. Nets with pins in more than one channel have to be decomposed in order to route the channels one by one. Beside subnets in the channels containing the concerned pins, interconnections between these subnets have to be made. The avenue channels can be used if *street pin-outs* are created. In order to avoid very long interconnections interchannel feed-throughs often can be established either by using electrically equivalent pins on opposite sides of a cell or by inserting feed-through cells into a row.
5. Analysis of pin positions. By fixing the positions and orientations

of the cells, and thus the pin positions along the street channels *latitudinal constraints* are introduced, when two nets enter the channel from opposite sides in the same interconnection layer and at exactly or almost the same longitudinal coordinate. If longitudinal parts of a net are to be realized by only one straight *trunk* cyclic constraints on the position of trunks may occur. Some conflict situations can be eliminated by adjusting pin positions. If not all constraint cycles can be broken the remaining problems must be solved by the router by slackening the straight trunk requirement ('trunkdivision', 'doglegging')

6. Routing of the street channels. For each channel pin positions on both sides are known and a net list is available. The *net list* contains the nets. A *net* is a set of pins that must be interconnected. Some nets may also have a street pin-out at one or both ends of the channel. The numbers of interconnection layers and the clearances are imposed by the technology. In order to simplify the router task the routing on a particular level mostly occurs in only one direction, either longitudinal or latitudinal. Under these constraints the router has to route all of the nets successfully in the minimum possible area.

7. Routing of the avenue channels. After the routing of the street channels the position of the street pin-outs is known. Since the position of the other pins on the sides of an avenue channel were predetermined all relevant pin positions are known. The net list is also available, thus the routing of the avenue channels can be performed by the same algorithm as the routing of the street channels.

Standard cell is the most successful layout design automation of the seventies. It is capable of always achieving a layout with all connections completed and all local design rules obeyed, but it also allows a high degree of user control. The success of the method probably can be explained by its hiding of all design rules in the predefined cells except the clearances of the wiring. Thus it reduces the design task to one optimization: minimize the width of the individual channels.

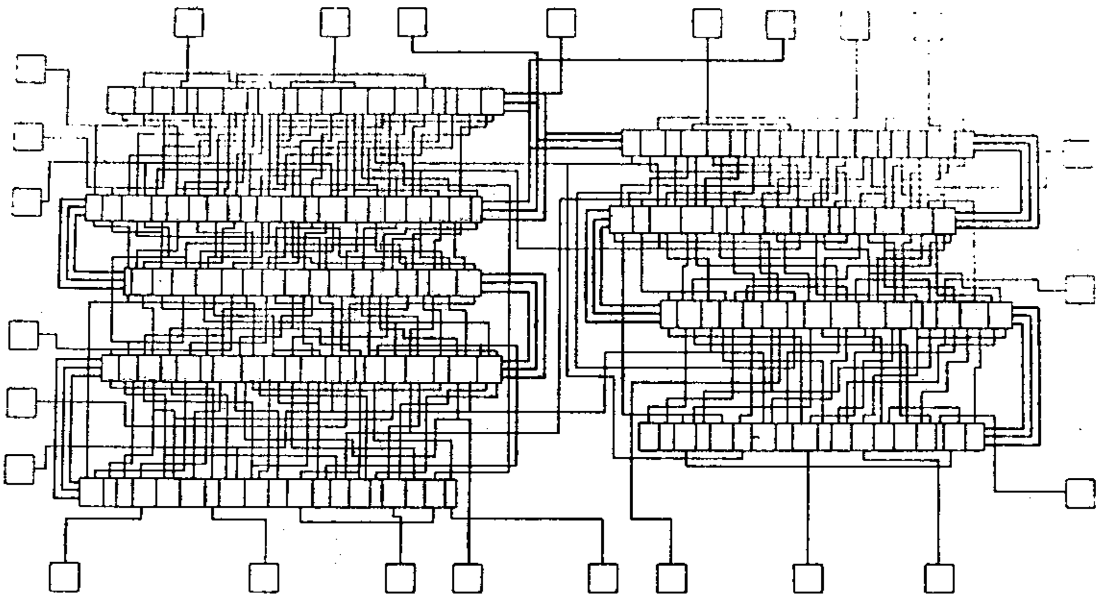


Figure 2.3: A standard cell layout

The bristle blocks system imposes a generic layout scheme at the cost of a restriction to processor chips with communication across databusses. For this limited class of circuits, however, the system fastly delivers a compact layout. The cells in the library have a certain flexibility that allows 'pitch matching' for simplifying interconnections between the data processing elements ^{JOH79}.

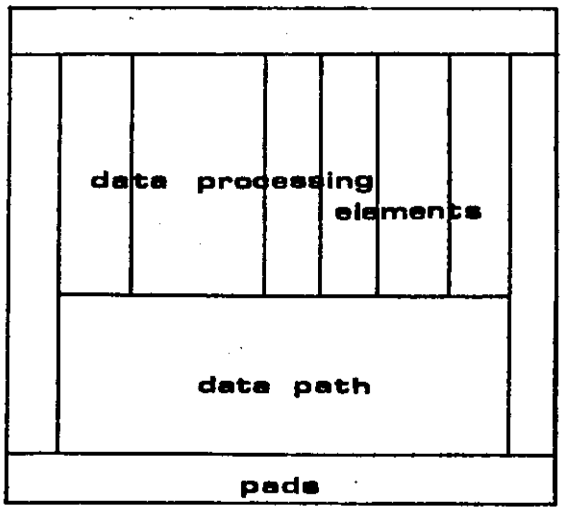


Figure 2.4: The format of a bristle block chip

Building blocks, standard cell, bristle blocks and many other standard form layout methods share the rectangular shape requirement, and there is no evidence that it was the limiting factor for the complexity the method can handle. Allowing arbitrary cell boundaries will certainly complicate layout design. Beside the rectangular form of the cells, another well-structuring principle is expedient as can be learned from careful examination of existing layout styles. This principle only allows layout structures which can be obtained by an operation called *slicing*. A single application of this operation divides a rectangle into smaller rectangles by parallel lines. The operation can be applied to each of the resulting rectanglès, *slices*, but with lines perpendicular to the preceding set of dividing lines. Slicing can be repeated to any depth, alternating the orientation of the dividing lines.

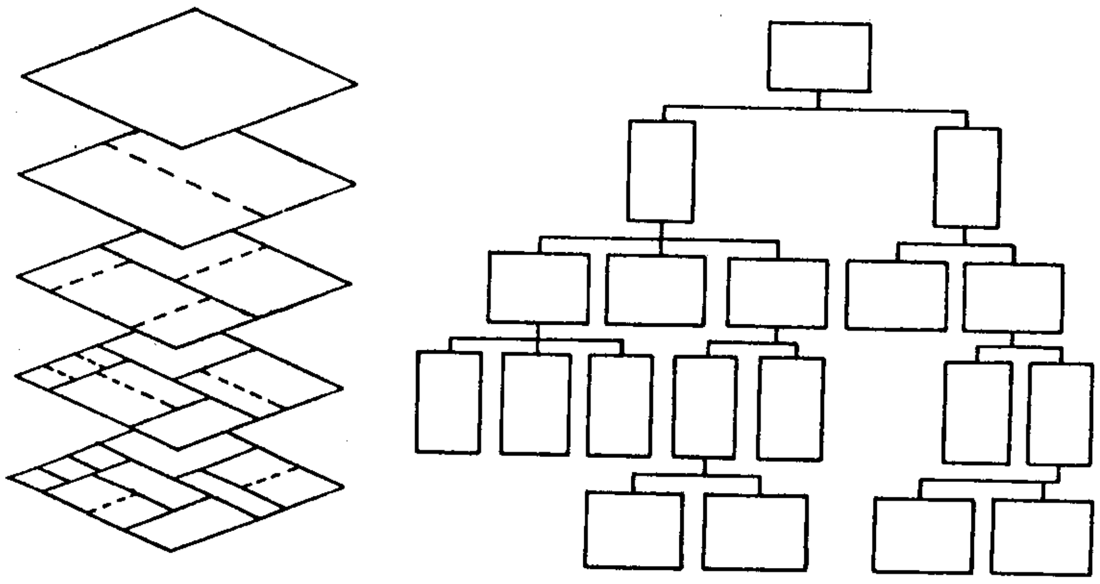


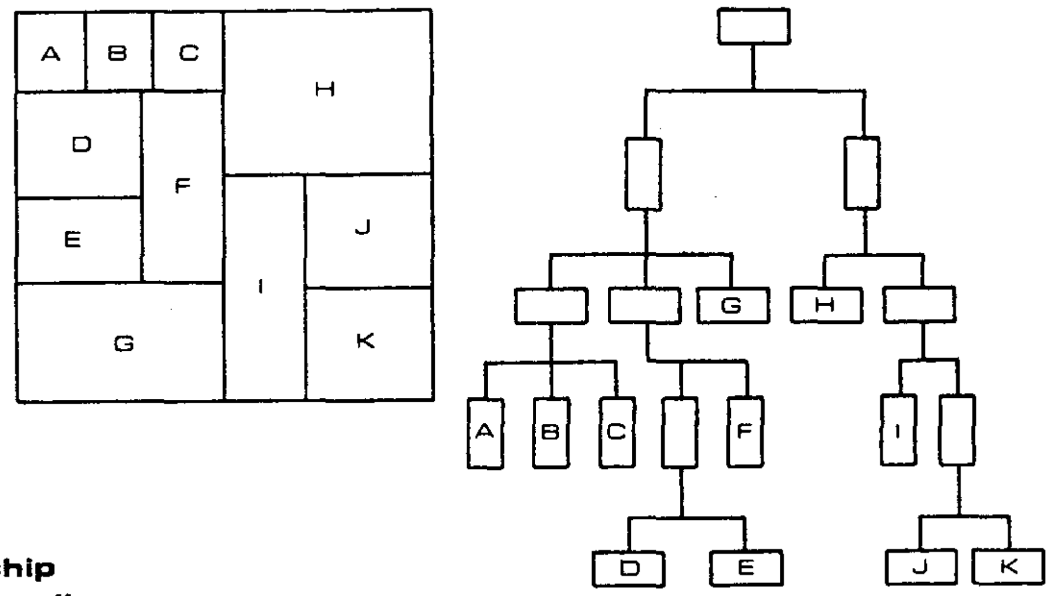
Figure 2.5: Slicing levels.

Slicing configurations can be represented by a rooted tree. To describe this tree the genealogical terminology is adopted. The whole rectangle is represented by the common ancestor. Each slicing corresponds with a parent and his children. These children are ordered according to the relative position of the associated subslices. Leaves represent slices to which no further slicing is applied. The structure tree represents the *genealogy* of the structure. Topologically a slicing structure is fixed by its genealogy. However, absolute coordinates may be generated later.

That the slicing concept yields a simple data structure is not surprising after the introduction of the genealogy tree. The genealogy tree is an ordered tree and consequently it naturally corresponds with a binary tree. In order to facilitate the traversal of the tree in both directions references to parents should be added. This leads to a triply linked tree where each vertex has, beside a pointer to the data of the corresponding slice, a pointer to its primogenitive, a pointer to the next sibling, and a pointer to its parent.

Another principle of structured programming is that a clear notation should support the logical design. Graphical representations have always been considered to be the apt form of communicating layout data. For VLSI systems textual forms are expected to be more efficient [GRA80]. A layout structure satisfying the slicing principle has a natural textual form, using a kind of block structure known in some programming languages. (Figure 2.6)

Both, a sliced layout and a hierarchic system, can be represented by rooted trees. It is tempting to identify the genealogy tree of the layout structure with the hierarchy tree of a given functional hierarchy. However, this will most likely result in a structure clash, because the functional hierarchy is not primarily based on layout considerations such as area, deformation, position, orientation, and ease of wiring. Only functional strength and connectivity are often correlated. Yet it is the functional hierarchy which is most easily supplied by the designer. So it seems to be expedient to accept the functional hierarchy as a starting point, and to modify the ensuing decomposition on the basis of criteria more directly related to layout. This modified decomposition should be suitable to be mapped onto a slicing configuration. This mapping will assign a slice to



```
chip
   slice
      slice A;B;C end
      slice
         slice D;E end
         F
      end
      G
   end
   slice H;
      slice I;
         slice J;K end
      end
   end
endchip
```

Figure 2.6: A slicing configuration, its structure tree and its textual form

each module in the hierarchy. Slices, however, will be assigned to groups of modules that do not constitute a supermodule in the given hierarchy. In order to obtain uniformity in treatment and description the class of compounds is extended by considering these groups as modules. These newly formed modules are called *program compounds*. The compounds that are present in the functional hierarchy, are called *user compounds*. By this extension a one-to-one correspondence between slices and modules is established.

Although slicing has some clear-cut advantages (not yet all mentioned), the question about the price paid for giving up the generality of building blocks must be answered. The answer is twofold. Firstly, the restriction imposed by the slicing principle is not very detrimental. Most manual designs are compatible with slicing, and some trials will readily show the range of the concept. Secondly, there are no cycles in the precedence constraints for the wiring areas as there generally are in building blocks KAW73. Moreover, in slicing structures at least one set of permissible sequences can be characterized by one simple condition: the wiring areas of a slice must be routed after the wiring areas of its child slices has been routed.

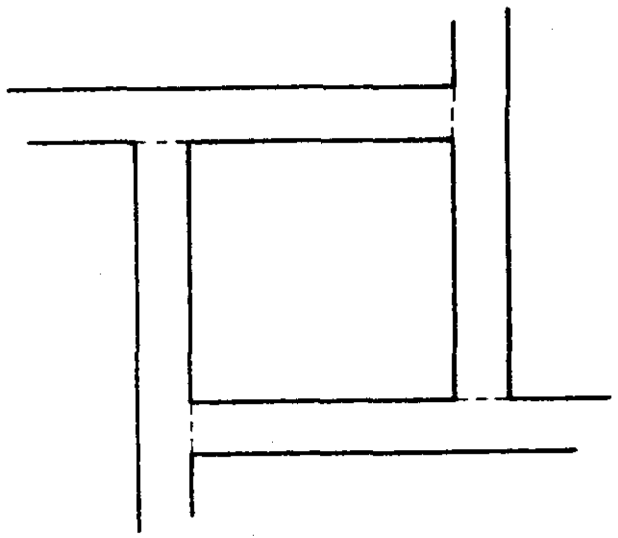


Figure 2.7: A configuration essentially excluded by the slicing principle

It is easy to prove that dissection rectangles satisfy the slicing principle fif their polar graphs are series-parallel graphs.

2.3 Wiring space management

Even a glimps at a number of chips of different complexity reveals the problem of the wiring space inflation. A lower bound on the wiring space is the total interconnect length times a constant imposed by technology (minimum line width + minimum spacing). It has been experimentally established that the total interconnect length is growing exponentially with the number of devices ^{HIG79}. Progress in technology cannot overcome this effect. Of course, more interconnection layers, tricks for distributing the supply voltage, flashing the clock signal onto the chip with light, and suchlike amendments will only partially offset the problem. Only a new design methodology can change the tendency. In many logic designs, for example, complex functions can be implemented in very regular patterns. In that case the increase in space required for interconnection and logic is kept close to linear.

Outside the regular patterns the wiring is an extra area consumer and should be treated as such. For that reason a new class of modules can be created. Their place is in between any pair of slices. Since they are not to be decomposed into submodules, they form a subset of the class of cells. To distinguish between the cells already present in the hierarchy and the newly created cells they are called *function cells* and *junction cells* respectively.

For each module area has to be reserved. However, it is impossible to assess the wiring space before any information concerning the placement of modules

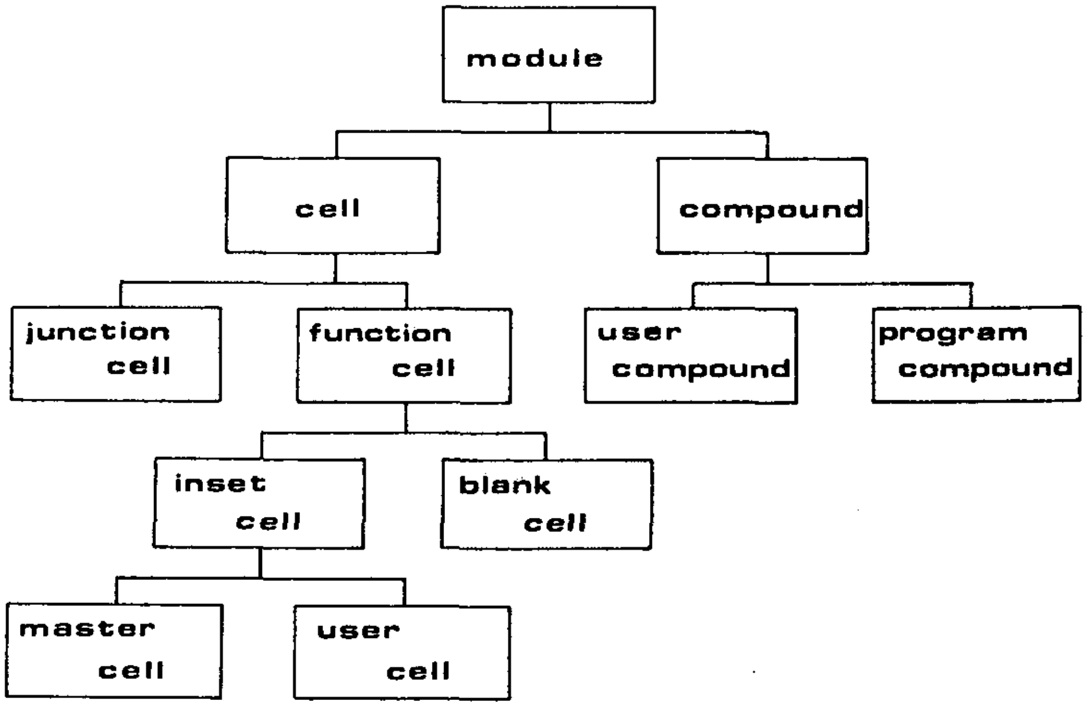


Figure 2.8: The classification of modules

is available. Topological data such as a genealogy makes area prediction feasible. As more information comes available the estimates have to be revised.

Stochastic models for estimating wiring demand did not get much attention in literature. For master-slice circuits a probabilistic model for wiring has been developed in order to predict wirability HEL78. That approach is not suited for estimating the local wiring demand in a slicing configuration in an early stage of the design. An attempt to model interconnections in custom integrated circuits has been published eGA80. It starts from a building block configuration. The rectangle partitioned into rectangles is viewed as a planar representation of the *channel graph*. It is assumed that pins are distributed over the edges of the graph according to a Poisson function, that interconnection paths are minimum distance paths along the edges of the channel graph while choosing between feasible minimum distance paths occurs with equal probability, and that interconnections have random lengths with an exponential distribution. Under these assumptions the local and maximum density on an edge of the channel graph can be estimated.

The occupation of junction cells by interconnections must be carefully controlled, to check the disproportionate growth of the wiring space. Two guidelines, both having a parallel in programming, are gainful in this respect.

The first one can be compared with the desirability of scope minimization in programming WUL73. Its translation into the layout environment is something like "keep wires as local as possible". The scope of a net can be reflected in the textual form suggested a few paragraphs earlier.

```
slice  external: < list of inherited nets >
       internal: <list of local nets>
       < slice body >
end
```

The pins interconnected by a certain net belong to function cells. The minimal subtree of the genealogy containing these function cells has as its vertices 'slices' with the concerned net in their lists. Only the common ancestor has the net declared as internal. Outside this minimal subtree only junction cells might contain a part of that net.

The other area-saving principle is pitch-matching. It consists of adjusting the pin position such that the interconnections can be made by crossing the channel without using a track. In programming it is comparable with reducing the interface complexity by using a suitable data structure common to the blocks.

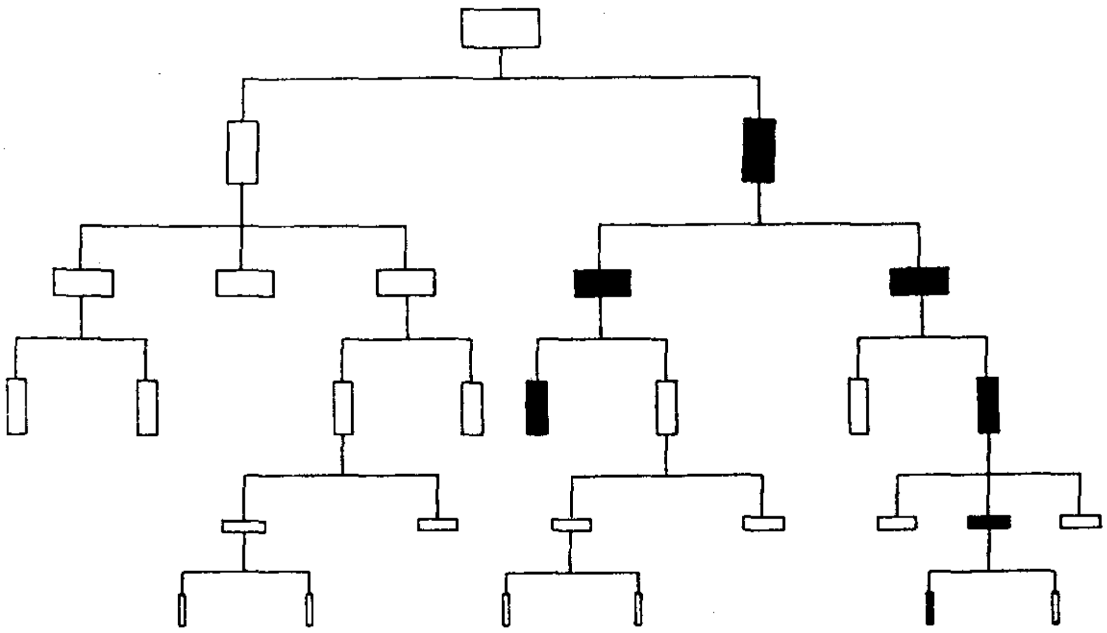


Figure 2.9: Apart from junction cells only slices in the minimal subtree of the concerned function cells are involved. However, even the junction cell of ~~the~~ common ancestor might contain the net!

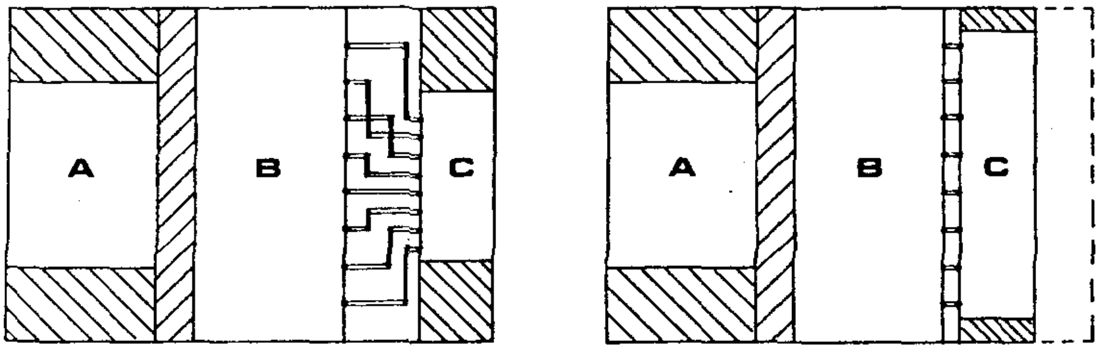


Figure 2.10: Stretching sometimes reduces the total area of its parent slice.

2.4 The flexibility imperative

The development of a structured program is a sequence of refinement steps that terminates when all instructions are expressed in terms of the concerned programming language ^{WIR71}. The whole point of programming by stepwise refinement can also be seen as delaying design decisions. This avoids committing the design prematurely to specific implementation ideas and increases the ease with which modifications can subsequently be made.

This postponement of design decisions is also of fundamental importance in layout design. Premature decisions lead to inefficient use of silicon area. Layout systems based on the building blocks approach manipulate rigid boxes and always end with a very low layout density, because of this a priori decision on the shapes of the modules.

The layout of a module may be realized in differently shaped areas if only the cells covered by the module and their mutual interconnections are accommodated. Of course, not all modules have the same degree of flexibility. Inset cells for example have fixed shapes. A layout system can only assign position and orientation to them. At the other extreme large portions of random logic have a lot of freedom as far as their circumscription is concerned. This flexibility must be used by a layout system to compensate differences in wiring space estimation, to allow for cell stretching in order to adjust the pin positions, and to obtain high packing densities.

In this respect layout design can be regarded as a gradual stiffening of modules, in which the modules get shape, position, orientation and pin positions. The moments of the stiffening step depend on the kind of module.

Although flexibility of modules has to be enforced, preferable shapes exist, also for modules with a high degree of flexibility. Deviations from these favored shapes can be measured in order to steer the stiffening process. The steering parameters or *deformations* are measured with formulas that depend on the type of the module. Inset cells have to fit in the assigned domains. If not, high deformations have to result. Other modules, with high flexibility, should have lower penalties for deformation. Square shapes are preferable for these modules. Adjustments to modules requiring big areas affect the final result stronger than modules with low area consumption. This should also be expressed in the deformation value.

2.5 Generality desirable

Developing layout design software is a costly affair. Considering the existing variety in technological processes and the rapid changes in fabrication methods, the need for largely general approaches becomes apparent. Sooner or later, however, the differences between the various sets of design rules will take their toll. Whether it is possible to keep the divergence concentrated in small parts of the program is an open question. The leaves of the genealogy tree, i.e. the cells of the system, are of course most heavily dependent on technology. Particularly the blank cells have to be filled by technology dependent algorithms. Though the complexity of the concerned problems is quite low, one may still ask which algorithms have to be used. For neither an established theory for automatic layout design nor a set of approved algorithms is available. The main reason is that for this scale of problems manual and interactive methods were preferred.

The most successful approach with a considerable degree of automation is the standard cell technique. As a first step towards more general layout methods one may ask whether other technologies allow for layout design along similar lines. Although the number of interconnection layers was expected to be the most stringent condition, a program similar to standard cell has been implemented for a single-layer technology [OTT80]. Beside the potential graph the algorithms in that program work upon a two dimensional *array of boxes* ('rectangular grid'). A *box* stands for a square unit-surface. The size of such a box depends on the components of the concerned circuit. It must be chosen such that each component can be efficiently fit in a number of these boxes. This number is called the *component size*. Boxes are arranged in rows and columns. Boxes in the first and the last columns and rows form the *border* of the array. Boxes are *adjacent* if they have a side in common.

To emphasize the parallelism the decomposition is kept similar to the one of standard cell systems given in section 2.2.

1. Partitioning of components in c-levels. Because of its bipartiteness the vertices of the potential graph can be partitioned such that no block in that partition contains c-vertices as well as t-vertices. Edges connect only vertices of different blocks in such a partition. Some of these partitions have the property that blocks have one-edge-connections to at most two blocks. Assuming that the graph is connected and that at least one block does not have one-edge connections to two different blocks, it is possible to order the blocks linearly such that only pairs of consecutive blocks have edges between each other. In that case the blocks are called *levels* and they are ordered by *level numbers*. A partitioning into levels is easily obtained: select any subset consisting either of c-vertices or of t-vertices for being the first block; each next block consists of the vertices not yet selected and connected by one edge to an already established block. This way of obtaining levels is often too rigid to be practical. Replacement of an edge by a chain of edges

with *pseudo-c-vertices* and extra t-vertices is allowed if the bipartiteness of the graph is preserved. These extensions and the choice of the first block can be used to get bonding pads in the chip border and to control the *c-level-claims* i.e. the minimal area required to contain all components represented in the concerned level. Level claims are also measured in the number of boxes. For reasons to be explained level claims have to be odd, but apart from that not greater than necessary. Accordingly the level claim for a certain level is obtained by summing up the component sizes of all c-vertices in that level, and in case the result is even increasing it by one. The extra box is the size of an extra c-vertex called the *parity vertex*. In order to determine the number of rows and columns the level claims are added together. The result is a lower bound on the number of boxes. In determining the array dimensions several rules are taken into account: small chip area, restrictions on length-width ratio and others. Futhermore, the number of columns have to be even and less than the level claim of all levels except the first one and the last one. Together with the oddness of the level claims the last requirement facilitates the placement procedure without appreciably affecting the quality of the layout.

2. Ordering of the c-vertices within a level. This ordering may be established under different criteria such as short total interconnection length, small number of crossings, and narrow wiring channels. If only one interconnection layer is available a planarization process is required. A suitable planarization algorithm is the "cascade method" OTT76 since it produces simultaneously an ordering of the vertices in the levels. The obtained orderings are recorded by *vertex positions*. In order to facilitate the discussion the *level representation of a graph* (LRG) has been introduced. The vertex position - if assigned - and the level number are considered as an abscissa and an ordinate respectively. Edges are represented by curves between the points assigned to its vertices by their vertex positions and level numbers. If the graph has been planarized these curves must be disjoint. Parity vertices obtain a greater vertex position than any other vertex in the level.

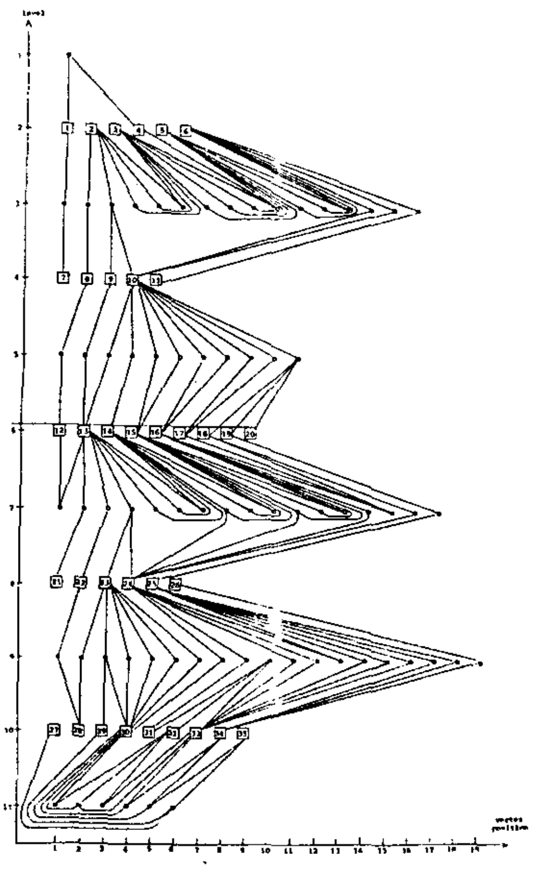


Figure 2,11:
The level representation of a planarized potential graph. Vertex positions and level numbers are chosen to be consecutive integers.

3. Placement of the components. The task of the placement algorithm is to establish a relation between the c-vertices and the array of boxes. On the basis of the graph, the level number, the vertex positions and the component sizes strips of boxes will be assigned to c-levels and connected sets of boxes in these strips will be assigned to c-vertices.
Strip assignment: A strip is a set of lexicographically consecutive boxes. They have exactly as many boxes as the respective level claim amounts to. Provided that components with sizes greater than two have flexible geometry, every component can be represented in the strip of its level by a connected set of boxes having the correct size. If the strips are ordered according to the associated level numbers, the strip assignment must be such that the lexicographical ordering of the boxes is a refinement of the strip ordering. This method of assigning strips causes interconnections to be needed only between components in the same or in successive strips. Only pseudo-c-vertices make interconnections between non-adjacent strips possible.

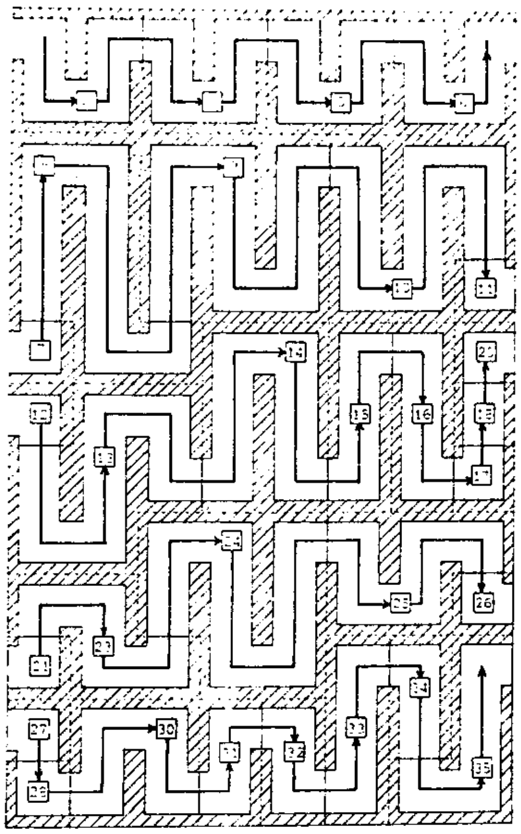


Figure 2.12:
Assignment of component domains to the components. The shaded areas have no other function than emphasizing the meander which is obtained by folding the c-level. The arrows indicate the ordering of the boxes in the strips

Box assignment: In order to preserve the ordering of c-vertices in their levels, boxes in a strip are ordered columnwise, traversing the columns alternately up and down. In case the level number is odd, the first column is traversed top-down; if the level number is even, the first column is traversed bottom-up. Thus the ordering of c-vertices in the levels is reflected in the placement of components in a strip. Since the level claims - except possibly the first and the last one - are greater than the number of columns, components represented by c-vertices with the extreme vertex positions of the level are placed in the border of the array. If the level claims of the extreme levels are less than or equal to the number of columns all the c-vertices of these levels are assigned to border boxes. Because of the odd-and-even conditions a connected set of boxes (a *component domain*) is assigned to each component. The truth of this assertion is immediately clear after observing that possible jumps in the horizontal strip boundaries occur alternately after an odd and an even numbered column. As a result of this procedure the

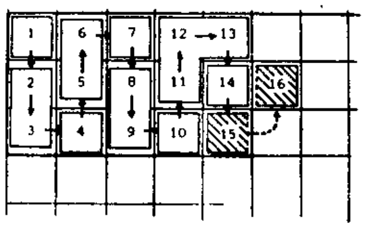


Figure 2.13:
If the odd-and-even conditions are not observed, components might get domains which are not connected.

c-levèl is supplied with the required number of boxes folded up to fit in the width of the array. The component domains of a c-level thus obtain a meanderlike structure (Figure 6.16).

4. Determination of the downsets and upsets. The corresponding step in standard cell methods, net decomposition, is considerably more complex than this step, because much of the work has been done already in generating an LRG. The procedure follows from the definitions. The *downset* of a c-vertex is the set of t-vertices in a level with a lower number and connected with that c-vertex via an edge. The t-vertices in a higher level that are connected with a c-vertex via an edge make its *upset*.

5. Transet analysis. Let us consider a homeomorphic mapping from the plane in which the LRG is onto the plane of the array of boxes which maps the LRG within the boundary of the array and the c-vertices on points in the associated components domains. The images of c-vertices in adjacent component domains are thought to be connected with the boundary by the shortest possible line segments. The images of these line segments under the inverse of the mentioned homeomorphic mapping are called *transcurves*. With each pair of c-vertices having a claim to interconnection by a transcurve and with each c-vertex demanding a connecting transcurve to the image of the array boundary a so-called transet is associated. A *transet* is defined as the set of t-vertices of which the t-stars (a *t-star* is a t-vertex with its incident edges) have to intersect the concerned transcurve irrespective of the chosen mapping. Three kinds of transets are distinguished. *Insets* are transets associated with c-vertices of the same level. *Intersets* belong to c-vertices from different levels. *Exsets* are transets involving the array boundary. Transets can be easily derived from the LRG. The whole procedure determining the transets comes to

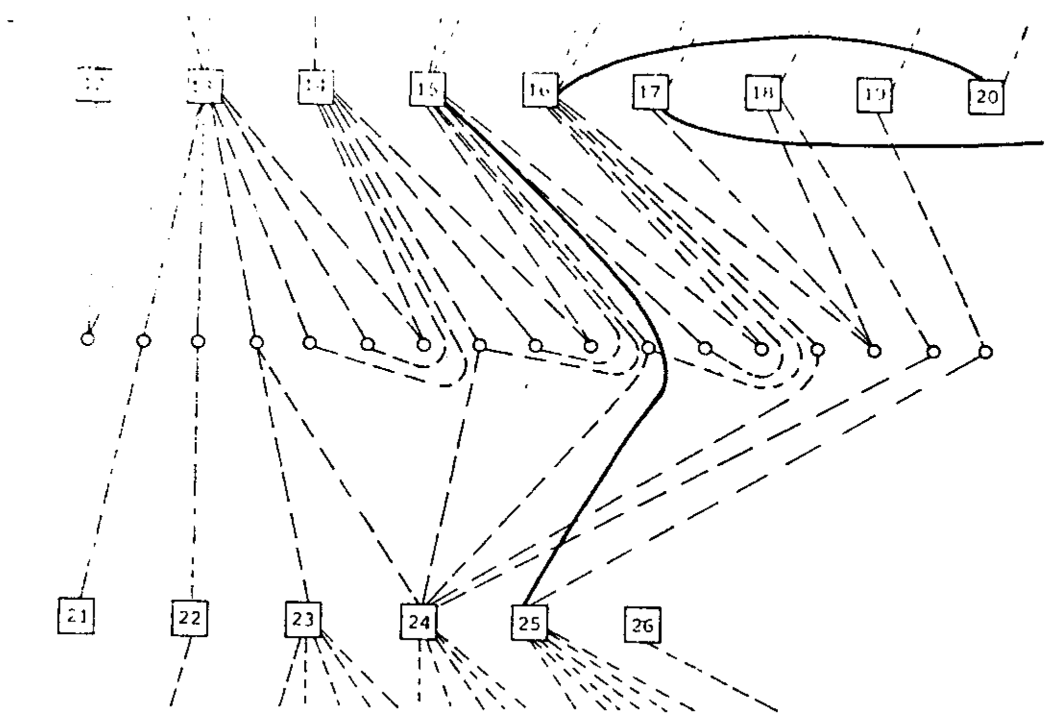


Figure 2.14: Three transcurves (full lines) are shown. Notice that the t-stars that are intersected by these transcurves cannot be avoided. These transcurves give rise to three different kinds of transets: the inset (16,20), the interset (15,25) and the exset (17). The subsets which have to be examined are (17, 18, 19) and (12, 13, 14, 15) for the inset (16,20) (16, 17, 18, 19, 20, 26) and (24, 23, 22, 21, 12, 13, 14) for the interset (15,25) and (18, 19, 20) and (12, 13, 14, 15, 16) for the exset (17)

forming two subsets of the concerned c-levels and examining the t-vertices of the concerned t-level on having edges to vertices of both subsets (Figure 2.14). To enhance uniformity in the procedures to follow downsets and upsets are also called transets. In the single layer case all these sets must be ordered. The ordering is automatically obtained by a systematical search for the elements of these sets. The system must be based on the sequence in which edges leave c-levels.

6. Routing of the horizontal channels. Wiring is restricted to *channels* between strips and some folds of the meanders. Only pseudo-c-vertices make deviations from this rule possible. *Horizontal channels* strictly follow the dividing lines between the strips which will be their centerlines. They extend from border to border without any break.

In the formation of the channels transets play a crucial role, because of the relation between their cardinality and the required channel capacity. If the transets are ordered sets - as for example in the single layer case - there is also a direct relation between these orderings and the sequence in which the interconnections are entering the channel and the relative positions of the wires in the channel. The consequence of being able to determine the required channel capacity and the pin sequence a priori is that one is no longer dependent on pathfinding algotithms liable to fail completion because of mutual blocking. Having found the pin sequence by juxtaposition of the respective transets coordinates have to be assigned to the pins in accordance with that sequence. In absence of a more specific rule one may uniformly distribute the pins over channel segments reserved for connections leaving the contiguous component domain (upsets or downsets!) or for connections entering the channel between two adjacent components domains (insets!). Often there are specific rules for example imposed by the layout of the individual components or necessary to avoid loop constraints. Anyway the relevant information for the router is a set of longitudinal *pin coordinates*, a specification of the pin position relative to the channel, and the t-vertex involved. The procedure for tracing the interconnections loosely adapts the idea of Lass' aperture LAS69. Here an *aperture* is a channel segment between two consecutive pin coordinates. The procedure completes all parts of interconnection nets that will fall into the range of the actual aperture before stepping to the next pin coordinate. The ordered sets of nets entering the rear side and exiting the front side are called the *old* and the *new buffer* respectively. Each net in the old buffer corresponds with a single point at the rear side of the aperture. The points in question are called *potential points*. They are distributed over the channel width. To each net in the new buffer a potential point at the front side is assigned. Potential points are the terminal points of interconnection segments to be completed within the aperture. If the same net is contained in both buffers the corresponding potential points will be connected. It is important to notice that the new buffer can be easily obtained from

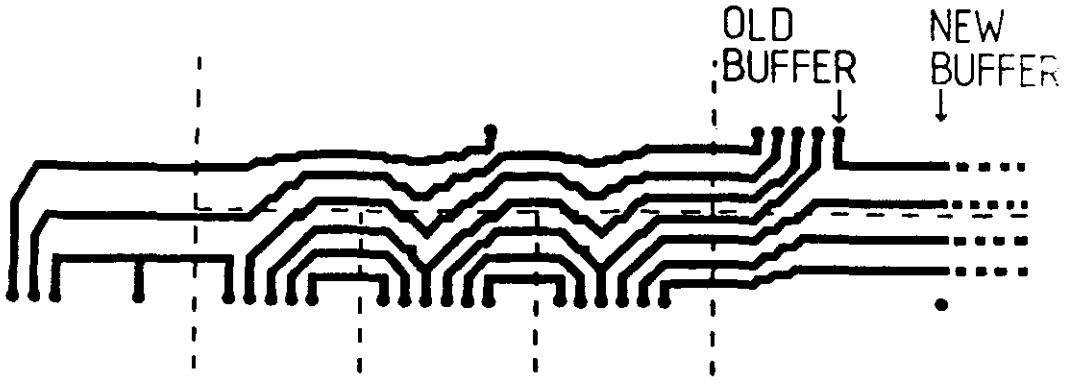


Figure 2.15: Formation of a channel

the old one, since differences are restricted to the outmost nets due to the entering or the termination of nets in the channel. New buffers are formed by duplicating the nets which have to be extended and by prefixing or postfixing eventual new nets. The procedure for assigning strips (stage 3) allows for one jump in a horizontal channel. Such a jump does not cause serious difficulties. The channel is divided into parts by a 45° cut at the place of a jump and one part is shifted over one grid unit. Corresponding nets are connected by vertical paths.

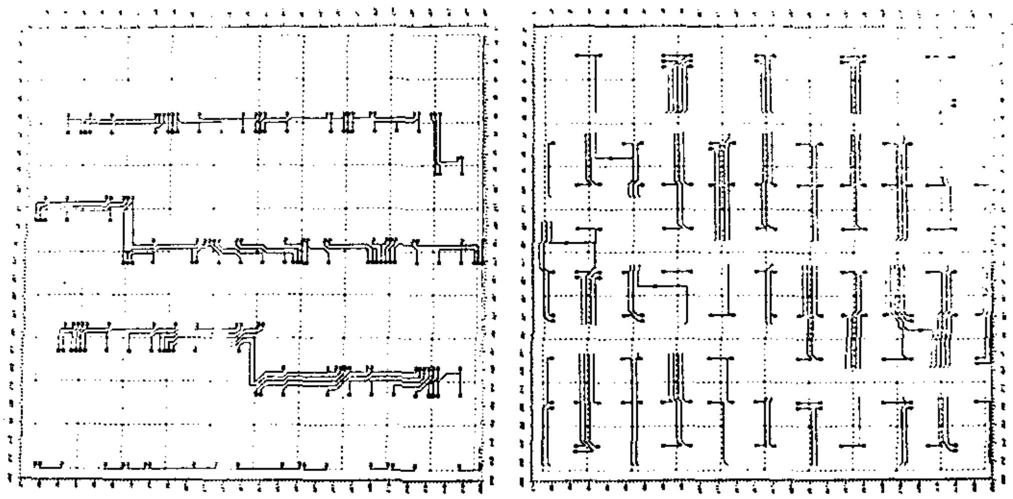
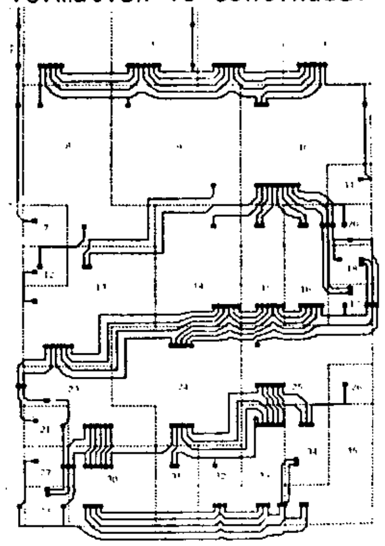


Figure 2.16: Results of the two routing phases

7. Routing of the vertical channels. Apart from some slight divergences vertical channels are formed in the same way as horizontal channels. They follow the vertical grid lines of the array. When they meet a horizontal channel, all the nets of the vertical channel are terminated (the new buffer does not become an old one). At the other side

nets have to be entered to form an old buffer from which the channel formation is continued.



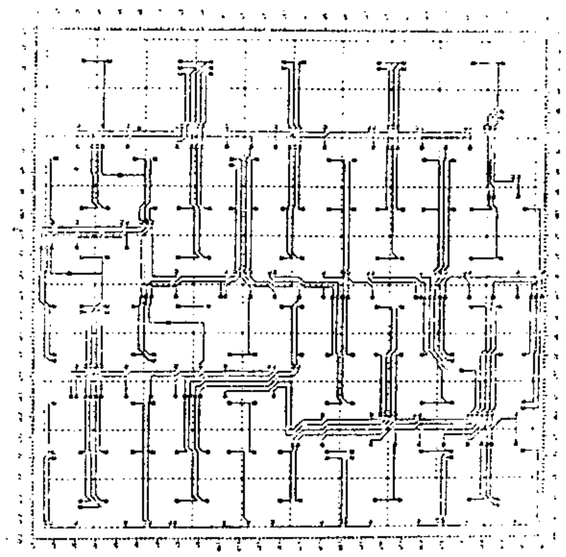


Figure 2.17: Two results of the router. On the left hand side is the completion of the example used throughout the description of the program. On the right hand side is the combination of the two results of figure 2.16

3. The genealogical approach

In the Automatic System Design Group of the Eindhoven University of Technology a project has been initiated to implement a system conformal to the imperatives of structured layout design. The first results were published in 1980 [SZE80]. The operating principles of this system, called SAGA, will be presented in this chapter.

3.1 Outline of the SAGA system

The core of SAGA can be divided into three successive parts that can be distinguished mainly on the basis of the way they treat the structure tree. The last part does not affect the structure tree, the first two parts do extend the structure tree, but in entirely different manners. The three parts are called GENEALOGIZE, PROCREATE, and INTERRELATE, because of an analogy with composing a saga (figure 3.1).

In the other sections the main lines of the system are sketched. The position of individual procedures is here indicated in an informal program description.

```
SAGA (hierarchy, {module characterization })
   procedure INSETFIT (slice)
      begin for each 'successor' inset cell do
         assign predescribed dimensions;
         adjust slice width to widest successor slice;
         fit other slices with minimum overall disturbance
      end

   procedure PERMUTE (slice)
      order successor slices to reduce the width to be expected:
```

```
    procedure REFLECT (inset cell)
       find optimal orientation;

    procedure IMPROVE (slice)
       begin INSETFIT
          for successor slices do
             begin PERMUTE (successor slice)
                for each 'second generation successor' inset cell do REFLECT
                ('second generation successor' inset cell)
             end
          PERMUTE (slice)
          for each 'successor' inset cell do REFLECT ('successor' inset cell)
       end

procedure DATASCRAPE (structure)
   collect as much information as possible (pin positions of the inset
   cells ("FETCH"), assign net to the junction cells, assign sets of pins
   to slice sides, call a "FILL" routine with correct technology and style
   dependent features to establish the layout of the blank cells);

procedure EXPAND (structure, {modules})
   expand structure with a slice;

procedure MERGE (user compound)
   minimize deformation by merging successor slices into program compounds;

procedure ROUTER (junction cell, technology, style)
   determine the contents by the appropriate algorithm;

procedure ASSEMBLE (slice)
   begin compact;
      determine relative coordinates of successor slices
   end

procedure GENEALOGIZE ({modules})
   begin select a germ module;
         assign a slice to the germ module;
```

Figure 3.1: The name and the terminology is chosen, because the procedure can be nicely compared with preparations a writer of a saga, a historical novel about a family, might make. His starting point is a number of historical facts he wants to fit in his own pattern. Since such a comparison also enhances comprehension of the program's operation, an example is given. The corresponding terms of the description in the text are given between brackets.

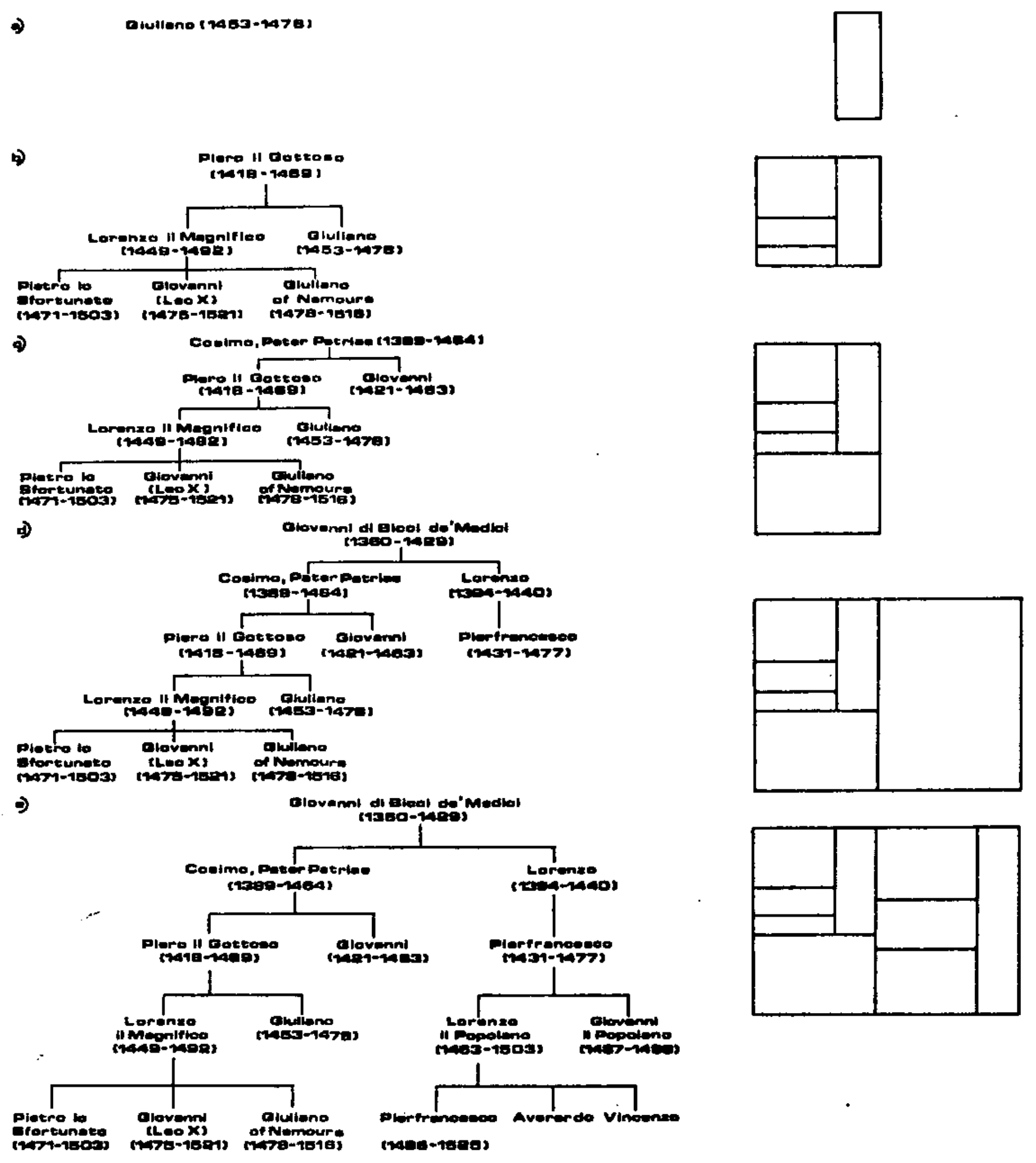
a) In the example the Medici are chosen as the family the saga is about. The writer has chosen Giuliano, the only brother of Lorenzo il Magnifico as the central character (germ module).

b) In order to find the family relations he needs a family tree in which all legitimate male adult Medici that lived before or during the life of Giuliano, are represented. He starts with constructing this family tree (structure tree) by finding all brothers of Giuliano with their children ordered according to their age (relative position). Giuliano had only one brother, Lorenzo, who was 4 years his senior and left 3 sons.

c) Then the writer passes on to the father, Piero il Gottoso, who is treated in the same manner. However, this Medici member had only one brother, Giovanni, whose only child died before reaching adulthood (so Giovanni is a leaf in the family tree which corresponds to a slice assigned to a cell).

d) The father of Piero il Gottoso was the famous Cosimo, the Father of the State. His younger brother Lorenzo begot only one son, Pierfrancesco. (In the structure tree the counterpart of an only-begotten son is an extra change of slicing direction). With this extension of the tree the 'genealogical phase' comes to an end, because the father of Cosimo and Lorenzo is the oldest Medici whose precise facts are known. (The structure is not expansible). He is to be the common ancestor of the characters in the saga.

e) Though lack of factual background prevents the writer from continuing tracing ancestors, he still is capable of extending the tree by finding children and grand-children of family members represented by leaves in the partial tree (user compounds). Pierfrancesco for example had two sons, of which the elder, Lorenzo il Popolano, had three sons. In one step of the 'procreative phase' the generations are thus extended. (In the slicing configuration the corresponding cutting line is first vertical, because Pierfrancesco was the only son of Lorenzo).


a)
Giuliano (1453-1478)
b)
Piero il Gottoso (1418-1469)
Lorenzo il Magnifico (1449-1492)
Giuliano (1453-1478)
Pietro lo Sfortunato (1471-1503)
Giovanni (Leo X) (1475-1521)
Giuliano of Nemours (1478-1516)
c)
Cosimo, Pater Patriae (1389-1464)
Piero il Gottoso (1418-1469)
Giovanni (1421-1463)
Lorenzo il Magnifico (1449-1492)
Giuliano (1453-1478)
Pietro lo Sfortunato (1471-1503)
Giovanni (Leo X) (1475-1521)
Giuliano of Nemours (1478-1516)
d)
Giovanni di Bicci de'Medici (1360-1429)
Cosimo, Pater Patriae (1389-1464)
Lorenzo (1394-1440)
Piero il Gottoso (1418-1469)
Giovanni (1421-1463)
Pierfrancesco (1431-1477)
Lorenzo il Magnifico (1449-1492)
Giuliano (1453-1478)
Pietro lo Sfortunato (1471-1503)
Giovanni (Leo X) (1475-1521)
Giuliano of Nemours (1478-1516)
e)
Giovanni di Bicci de'Medici (1360-1429)
Cosimo, Pater Patriae (1389-1464)
Lorenzo (1394-1440)
Piero il Gottoso (1418-1469)
Giovanni (1421-1463)
Pierfrancesco (1431-1477)
Lorenzo il Magnifico (1449-1492)
Giuliano (1453-1478)
Lorenzo il Popolano (1463-1503)
Giovanni il Popolano (1467-1498)
Pietro lo Sfortunato (1471-1503)
Giovanni (Leo X) (1475-1521)
Giuliano of Nemours (1478-1516)
Pierfrancesco (1486-1525)
Averardo
Vincenzo

Four steps in the genealogical phase and one in the procreative phase are illustrated. It is important for comprehending the operation of the program to note which generations are involved in each step, and when the two phases come to an end. The genealogical phase stops when the common ancestor is found (the structure is not expansible, i.e. the whole system is covered). The procreative phase continues until all potential parents are investigated (until all leaves represent slices assigned to cells). In the third phase the writer has to fill in the details of the lives and interrelations of his characters with different degrees of freedom depending on the historical facts known about them.

```
            structure: = assigned slice;
         while structure expansible do
            begin EXPAND (structure, {unplaced cardinal modules});
                    IMPROVE (new slice)
            end
            DATASCRAPE (structure)
      end

procedure PROCREATE (structure tree, {user compounds})
      begin while user compound leaves exist do
               begin take a user compound
                       represented by a leaf;
                       MERGE (user compound);
                       IMPROVE (new slice);
                       DATASCRAPE (new slice)
               end
      end

procedure INTERRELATE (structure tree)
      begin while common ancestor not visited do
               begin select a node with all successors
                       except juntion cells processed;
                       for each 'successor'junction cell of
                           the associated slice do
                           ROUTER ('successor'junction cell);
                           ASSEMBLE (associated slice)
               end
      end

begin data preparation;
         GENEALOGIZE ({principal modules});
         PROCREATE (partial genealogy, {user compounds});
         INTERRELATE (genealogy);
         data collection
end
```

3.2 Data preparation

Beside some auxiliary data derived from design rules and designer requirements the input for the SAGA-system consists of the (functional) hierarchy of the system to be integrated and the initial characterization of the modules represented in the hierarchy. With these data the program starts building up a *module data base*. This module data base will at any moment contain the actual characterization of each module. This characterization consists of:

1. The *identification part*: In this part some fixed information is encoded such as name in the input file, type of the module, and place in the hierarchy. Some variant data may be added such as flexibility class (discussed later) and level in the structure tree.
2. The *graphical part*: This part either contains a pointer indicating where the layout of the module is stored or a pointer referring to the data from which the layout is to be derived and by which algorithm.
3. The *form vector*: The components of this vector are the parameters in calculating the deformation and position cost of the module in a certain realization. The cost formula depends on the type of the module.
4. The *flexibility part*: Data subjected to the stiffening process such as dimensions and orientation (preliminary or definitive) and pin information (net to which the pin belongs, side of the rectangle in which the module is to be realized, pin position) is referred to by pointers contained in this part.

What data have to be immediately entered into the module data base depends on the type of the module. For master cells the input file contains a reference to the master library, where the layout, the form vector, and the list of external nets with the associated relative pin positions are stored. For user cells these data must be contained in the input file itself, and routed to the user library. A pointer indicating its place in the user library should be stored in the pertaining parts of the module characterization. The form vector, the list of external nets, and the data for technology and style dependent algorithms that have to determine the internal layout of blank cells, should be in the input file for all other function cells. Again pointers should be sent to the module data base, the bulk is to be stored in the user library. For user compounds the form vector should be evaluated from the form vectors of its submodules and some empirical provision for the area of junction cells.

The entire module data base resides in the common data base of the design system. Parts of it are from time to time copied in the private data base and an updated version might be sent back.

3.3 GENEALOGIZE

GENEALOGIZE only works on the set of cardinal modules. It selects one cardinal module that is going to be the *germ module*. This single slice is the first of a sequence of *expansible structures*. The expansion is performed by selecting a group of cardinal modules on the basis of a heuristic drop-out procedure. The heuristic is based on the cost calculated with the components of the form vector. It measures in a weighted manner the deformation these modules will incur and the connectivity with the expansible structure. To perform deformation calculations the width of the interjacent junction cells has to be estimated even with the sparse information available in this stage. The inexactitudes have to be compensated by the flexibility of the modules. The contribution of connectivity plays a role in preventing interconnection lengths from unnecessary inflation, but also in choosing a side of attachment by discriminating the four sides. Each expansion, therefore, tries all sides of the expansible structure, and chooses the one allowing the lowest cost. The selected group will form a program compound if it consists of more than one cardinal module.

In figure 3.2 two successive expansions are illustrated. Each time a slice is assigned to the selected group. This new slice is immediately subjected to some improvements concerning its dimensions in order to fit in inset cells, the sequence of child slices in order to minimize the maximally needed width, and the orientation of inset cells in order to simplify future wiring patterns.

The structure is no longer expansible when it contains all cardinal modules. Of such a structure as much data is extracted as possible in that stage of the construction. The contents of the inset cells are therefore fetched from the libraries, nets are assigned to junction cells, pins are distributed over the four sides of their modules, and blank cells are filled by special technology and style dependent algorithms.

GENEALOGIZE can be seen as an automatic chip-planning method. The decisions are dominated by form considerations. However, interconnections play a dominant role in area consumption. The justification of this chôice must come from practice. If interconnections should dominate the decision steps another chip-planning algorithm must be developed. It should derive and take into account length of routes, ease of wiring, area consumption, etc. The advantages of slicing are still valid and should be obtained.

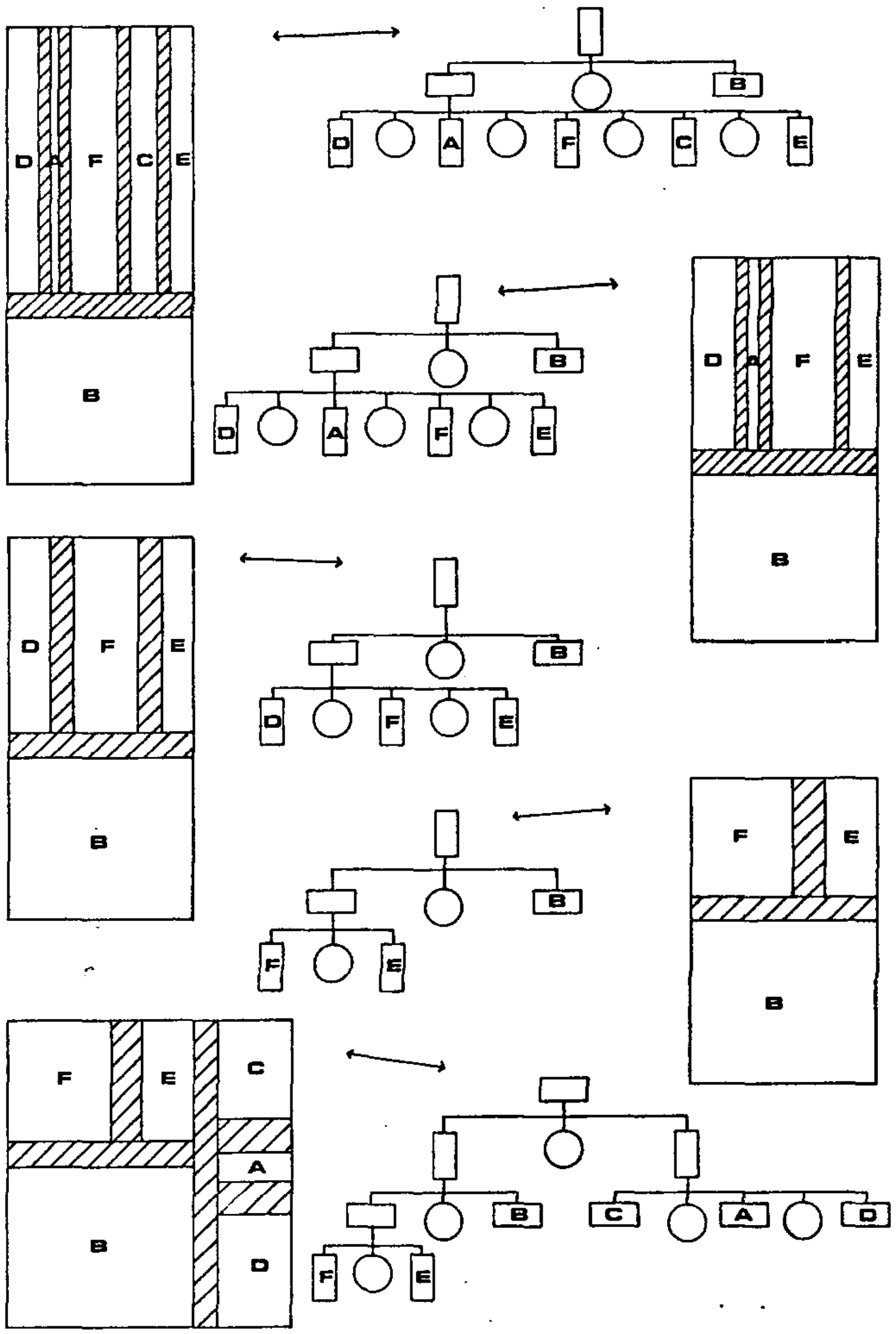

D
A
F
C
E
B
D
A
F
C
E
B
D
A
F
E
B
D
A
F
E
D
F
E
B
D
F
E
B
F
E
B
F
E
B
F
E
C
A
B
D
B
C
A
D
F
E

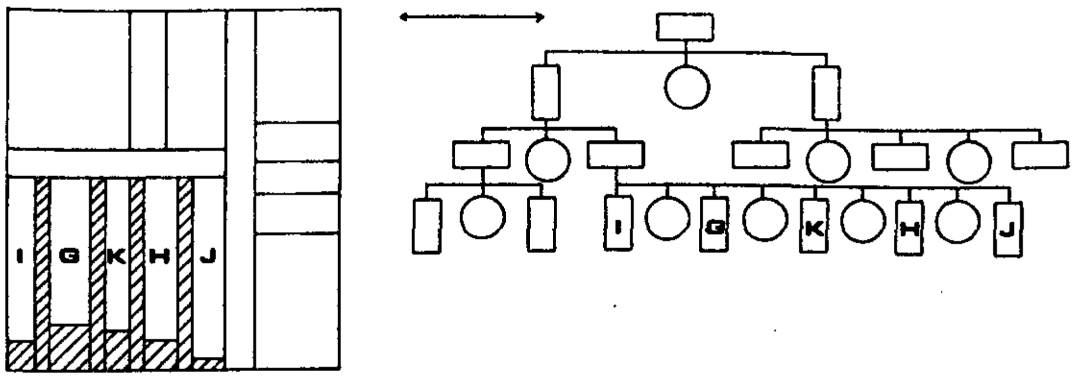
Figure 3.2: GENEALOGIZE starts with selecting a germ module among the cardinal modules, B in the figure. EXPAND starts with joining a slice containing all unplaced cardinal modules. Since the width of this slice is fixed by the side of the actual structure, the height has to be adjusted such that these modules and the estimated junction cells can be accommodated. For each individual module its deformation and its connectivity to the expansible structure is determined. These values and the components of the form vector are the arguments for the contribution of a module to the cost associated with the proposed structure. Next, the module with the highest contribution is removed, C in the figure. Now, the calculations are performed for a slice with all unplaced cardinal modules except the one removed. If the total cost and at least one individual contribution are lower than in the preceding structure the process is continued, otherwise the preceding structure is accepted for that side of the expansible structure. In the figure the process is continued until only one module was proposed for joining the expansible structure. Apparently, either the total cost or the individual contribution of that module increased, so the structure with two cardinal modules, E and F, is accepted. The entire process is repeated for all sides of the expansible structure, and the result with the lowest total cost is accepted as the new structure which is immediately improved. If this new structure is expansible the EXPAND procedure is applied to it, otherwise GENEALOGIZE is completed by extracting new data from the final structure. To prevent excessive deformation in the last steps of GENEALOGIZE the total cost of the preceding slice is added to that of the actual new slice and compared with the lowest first proposal (thus, with all unplaced cardinal modules) of EXPAND in the preceding step. If the latter is lower that proposal is still accepted.

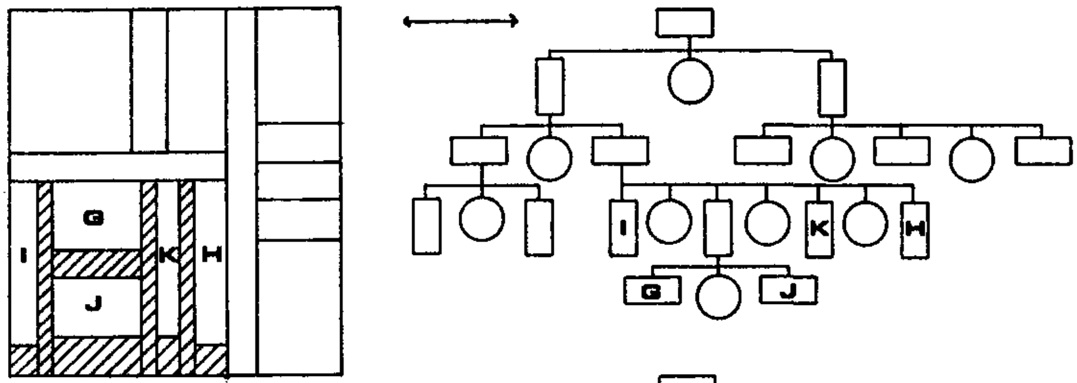
3.4 PROCREATE

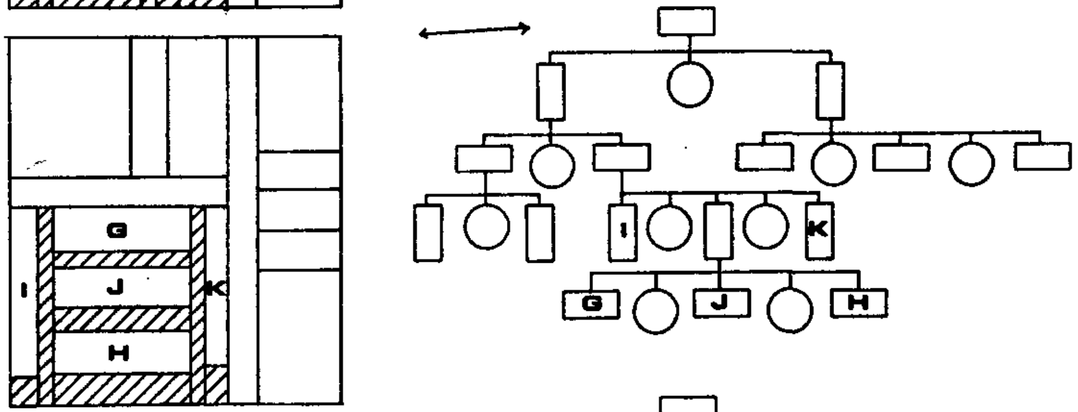
PROCREATE is directing all action to user compounds represented by leaves in the partial genealogy tree. As GENEALOGIZE this procedure also establishes program compounds, but the constraints are now essentially different. Whereas GENEALOGIZE utilized the fact that chip dimensions are quite free, PROCREATE attempts to fit the modules in given contours guided by the deformations the modules incur. However, the area assigned to a user compound is partially based on area estimations. If less area is needed than was expected an amount of spare area results. Spare area can be used to offset extra area demands of child slices. If more area is needed than expected extra area has to be demanded from ancestors. The nearer this ancestor is the sooner the disturbance is nullified.

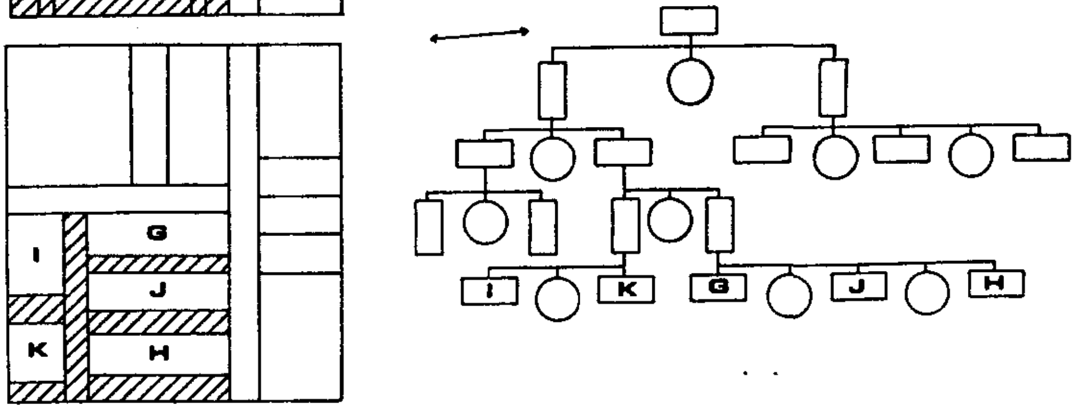
The dimensions of a slice and its child slices have to satisfy the equations in figure 3.3. By giving the widths of the junction cells estimated values, only one degree of freedom is left. This can be eliminated by constraining the dimensions of the slice, for example fixing one of them or the aspect ratio to a feasible value.

Figure 3.3: In PROCREATE user compounds are optimized by establishing new program compounds such that the sum of all individual deformations is reduced. This optimization is performed for each user compound by the MERGE procedure. It starts by assigning a slice to each submodule of the concerned user compound. The widths of the junction cells are again estimated by probabilistic techniques. Next, the dimensions of the slices are calculated, and the deformations associated with these dimensions are determined. Summation of these deformations gives the total cost of the proposed slice. The pair of child slices having the highest contribution to this sum is combined to form a program compound. This is a new proposal for the slice to be assigned to the user compound. The same calculations are repeated for this newly proposed slice. If the cost is lower than for the preceding slice proposal a new program compound is formed in the same manner. If the cost is not reduced the preceding slice configuration is accepted. In the figure the five submodules are initially assigned to child slices. G and J, apparently, have the highest deformation, and consequently have to be combined into one child slice. This new child slice and H have the highest deformations in the new configuration. So, they are combined. Now, I and K are the candidates to be merged which yields the last configuration in the figure. Combining the final child slices does not reduce the cost which means that the user compound gets a slice with two child slices containing I and K, and G, J and H, respectively


I
G
K
H
J
I
G
K
H
J


G
I
K
H
J
I
K
H
G
J


G
I
J
K
H
I
K
G
J
H


G
I
J
K
H
I
K
G
J
H

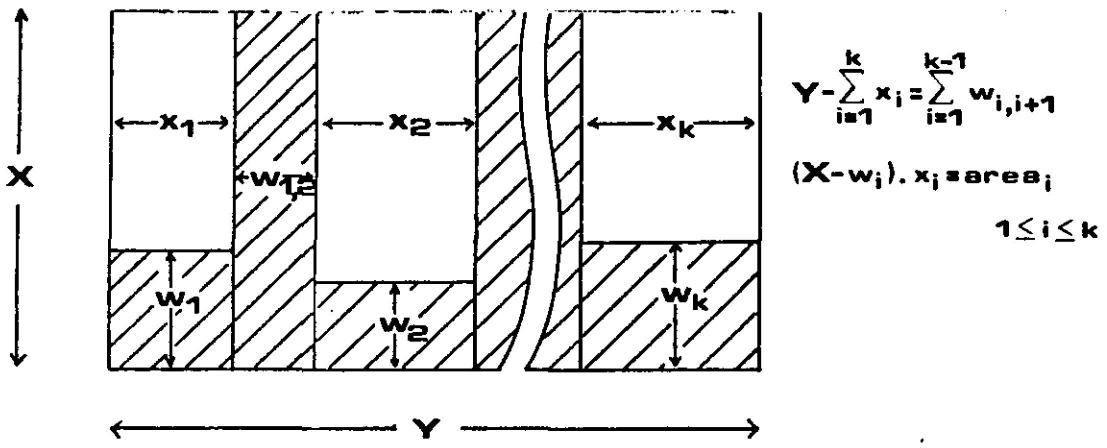


Figure 3.4: A slice and its associated set of equations (no variable is negative).

The final slice configuration assigned to the user compound is subjected to similar improvements and data determination as in GENEALOGIZE.

3.5 INTERRELATE

GENEALOGIZE and PROCREATE establish the genealogy of the system once and for all. INTERRELATE has to assemble the layout in a bottom-up fashion. Slices are to be treated one by one in a sequence satisfying the condition that no slice is treated before all its successor slices are treated. Visiting a node during the bottom-up traversal of the genealogy tree means: finding the contents of all junction cells of the corresponding slice and subsequently squeezing the slice longitudinally. The relative coordinates of the successor slices are recorded.

3.6 Stiffening of modules

One of the imperatives of structured layout design is the flexibility of the modules. During the design process the modules are subjected to a stiffening process. In SAGA a module can be in six different flexibility classes. In which class a module is depends on how much data have been acquired, particularly on what is known about the distribution of the pins, the dimensions and the position. At the end of the design all modules should be in the lowest class.

flexibility class		specification			
V	a	$\dot{P}$			
	b	$\ddot{P}$	$\dot{x}$	$\dot{y}$	
IV		$\dot{P}$	$\dot{x}$	$\dot{y}$	
III	a	$\dot{P}$	x	y	
	b	$\bar{P}$	$\dot{x}$	$\dot{y}$	
II	a	$\bar{P}$	x	y	
	b	P	$\dot{x}$	$\dot{y}$	
I		P	x	y	
Ø		P	x	y	T

Figure 3.5: The flexibility classes of the modules

$\dot{x}$,$\dot{y}$	provisional width, length of the module
x ,y	final width, length of the module
$\ddot{P}$	partial list of external nets of the module
$\dot{P}$	complete list of external nets of the module
$\bar{P}$	list of external nets distributed over the four sides of the module
P	As $\bar{P}$, but with partial or complete sequence constraints
T	module position

routine	inset cells	blank cells	junction cells	user compounds
EXPAND	V- IV	V- IV	- V	V- IV
MERGE	V- IV	V- IV	- V	V- IV
INSETFIT	IV-III	-	-	-
REFLECT	III- II	-	-	-
FETCH	II- I	-	-	-
DATASCRAPE	-	IV-III	V-II	IV-III
FILL	-	III- I	-	-
ROUTER	-	-	II- I	
ASSEMBLE	I- 0	I- 0	I- 0	III- 0

Figure 3.6: Flexibility transitions

Conclusion

In conclusion it may be stated that the design of correct lay-outs for densly packed large integrated circuits is a difficult but manageable task. The key to the solution of the problem is structuring the design problem. The present state of the art together with some of the new concepts assembled in this report present a fairly dependable foundation for further approaches.

Our optimism is based on the observation that many very complex systems (electronics and other) have been successfully designed and built in the past. The amount of design data of a large mainframe is in no way less than that of a future VLSI microprocessor chip. Still engineers and managers were able to cope successfully with those masses of data. For those design problems structural hierarchy was the essential tool as well. The hierarchy was imposed by the physical realisation via racks - back planes - printed circuit cards - chips. The design data were distributed over various designers all working on different levels of the hierarchy. Any one of them saw only a limited subset of the data, namely the ones being relevant for his task. The managers were responsible for the proper partitioning of the design task and data such that any designer saw not more data than he could cope with.

VLSI technology suddenly enhances the complexity on the chip level to the extend that a whole design crew has to be put to work in order to complete the design on one single chip within reasonable time. However, the design data created by any of the designers have to be assembled into one single set of masks. Moreover area, shape and structure of any of the modules have a great influence on the overall effectivity of the design. This results in iterative reshaping and re-structuring of the various modules and this process requires more communication between the designers of various modules than with previous realisation techniques. Consequently, the convergence of the design may be slow or even not at all present.

As it has been said this problem seems manageable. The essential breakpoint of VLSI design and -production will eventually be the testing problem. Whereas the testing of random access memory chips will probably be not too difficult, the testing of data processing chips is at the moment the essentially unsolved problem.

Acknowledgement

The subject 'Automatic Layout Design' was introduced into the group 'Automatic System Design' of Eindhoven University of Technology (Department of Electrical Engineering) by its initiator, Professor Dr. Jochen A.G. Jess. From the beginning the group aimed at full automation of the layout design task. Much study of literature, great effort in achieving this ambitious goal, and several good results can be recorded. Many scientists and students conduced to the philosophy set forth here. Major contributions are due to Dr. Marinus C. van Lier and Antoni A. Szepieniec. More authors than mentioned in the following list of references had an influence on the development of that philosophy.

References

BER70	Berge, C. GRAPHES ET HYPERGRAPHES. Paris: Dunod, 1970. Monographies universitaires de mathématiques, Vol. 37. English translation: Amsterdam, North-Holland, 1976. 2nd ed.
BöH66	Böhm, C. and G. Jacopini FLOW DIAGRAMS, TURING MACHINES, AND LANGUAGES WITH ONLY TWO FORMATION RULES. Commun. ACM, Vol. 9(1966), p. 366-371.
vBO81	Bokhoven, W.M.G. van PIECEWISE-LINEAR MODELLING AND ANALYSIS. Ph.D. Thesis. Eindhoven University of Technology, 1981.
BRO75	Brooks, Jr., F.P. THE MYTHICAL MAN-MONTH: Essays on software engineering. Reading, Mass.: Addison-Wesley, 1975.
BRO40	Brooks, R.L., C.A.B. Smith, A.H. Stone and W.T. Tutte THE DISSECTION OF RECTANGLES INTO SQUARES. Duke Math. J., Vol. 7(1940), p. 312-340.
DAH72	Dahl, O.-J., E.W. Dijkstra and C.A.R.Hoare STRUCTURED PROGRAMMING. London: Academic Press, 1972. A.P.I.C. studies in data processing, Vol. 8.
DEM68	Dembowski, P. FINITE GEOMETRIES. Berlin: Springer, 1968. Ergebnisse der Mathematik und ihrer Grenzgebiete, Band 44.
eGA80	El Gamal, A. STOCHASTIC MODELS FOR INTERCONNECTIONS IN INTEGRATED CIRCUITS. In: Proc. 1st IEEE Int. Conf. on Circuits and Computers (ICCC 80); Port Chester, N.Y., 1-3 Oct. 1980. Ed. by N.B.G. Rabbat. New York: IEEE, 1980. P. 77-79.
GIB76	Gibson, D. and S. Nance SLIC - Symbolic Layout of Integrated Circuits. In: Proc. 13th Design Automation Conf.; San Francisco, 28-30 June 1976. New York: IEEE, 1976. P. 434-440.
GRA80	Gray, J.P. STRUCTURED DESIGN NOTES. Pasadena: California Institute of Technology, 1980.

GÜN69 Günther, T.
DIE RÄUMLICHE ANORDNUNG VON EINHEITEN MIT WECHSEL-
BEZIEHUNGEN. Elektron. Datenverarb.,Vol. 11(1969), p. 209-212.

HAN72 Hanan, M. and J.M. Kurtzberg
PLACEMENT TECHNIQUES.
In: DESIGN AUTOMATION OF DIGITAL SYSTEMS. Ed. by M.A. Breuer.
Vol. 1: Theory and techniques.
Englewood Cliffs, N.J.: Prentice-Hall, 1972. P. 213-282.

HAS71 Hashimoto, A. and J. Stevens
WIRE ROUTING BY OPTIMIZING CHANNEL ASSIGNMENT WITHIN
LARGE APERTURES.
In: Proc. 8th Design Automation Workshop; Atlantic City,
June 1971. P. 155-169.

HEL78 Heller, W.R., W.F. Mikhail and W.E. Donath
PREDICTION OF WIRING SPACE REQUIREMENTS FOR LSI.
J. Des. Autom. & Fault-Tolerant Comput., Vol. 2(1978),
p. 117-144.

HIG69 Hightower, D.
A SOLUTION ON LINE-ROUTING PROBLEMS ON THE CONTINOUS
PLANE.
In: Proc. 6th Design Automation Workshop; Miami Beach,
June 1969. P. 1-24.

HIG79 Hightower, D.
CAN CAD MEET THE VLSI DESIGN PROBLEMS OF THE 80's.
In: Proc. 16th Design Automation Conf.; San Diego,
25-27 June 1979. New York: IEEE, 1979. P. 552-553.

HSU79 Min-Yu Hsueh
SYMBOLIC LAYOUT AND COMPACTION OF INTEGRATED CIRCUITS.
M.Sc. Thesis. Department of Electrical Engineering and
Computer Sciences, University of California, Berkeley, 1979.

JOH79 Johannsen, D.
BRISTLE BLOCKS: A silicon compiler.
In: Proc. 16th Design Automation Conf.; San Diego,
25-27 June 1979. New York: IEEE, 1979. P. 310-313.

KAN76 Kani, K., H. Kawanishi and A. Kishimoto
ROBIN: A building block LSI routing program.
In: Proc. IEEE Int. Symp. on Circuits and Systems;
Munich, 27-29 April 1976. New York: IEEE, 1976.
P. 658-661.

KAW73 Kawanishi, H., S. Goto, T. Oyamada, H. Kato and K. Kani
A ROUTING METHOD OF BUILDING BLOCK LSI.
In: Proc. 7th Asilomar Conf. on Circuits, Systems, and Computers; Pacific Grove, 27-29 Nov. 1973. Ed. by S.R. Parker. North Hollywood, Calif.: Western Periodicals, 1974. P. 119-123.

KER70 Kernighan, B.W. and S. Lin
AN EFFICIENT HEURISTIC PROCEDURE FOR PARTITIONING GRAPHS.
Bell Syst. Tech. J., Vol. 49(1970), p. 291-307.

KER73 Kernighan, B.W., D. Schweikert and G. Persky
AN OPTIMUM CHANNEL-ROUTING ALGORITHM FOR POLYCELL LAYOUTS OF INTEGRATED CIRCUITS.
In: Proc. 10th Design Automation Workshop; Portland, 25-27 June 1973. New York: IEEE, 1973. P. 50-59.

LAT79 Lattin, B.
VLSI DESIGN METHODOLOGY: The problem of the 80's for microprocessor design.
In: Proc. 16th Design Automation Conf.; San Diego, 25-27 June 1979. New York: IEEE, 1979. P. 548-549.

LAU80 Lauther, U.
A MIN-CUT PLACEMENT ALGORITHM FOR GENERAL CELL ASSEMBLIES BASED ON A GRAPH REPRESENTATION.
J. Digital Syst., Vol. 4(1980), p. 21-34.

LEE61 Lee, C.Y.
AN ALGORITHM FOR PATH CONNECTIONS AND ITS APPLICATIONS.
IEEE Trans. Electron. Computers, Vol. EC-10(1961), p. 346-365.

OTT76 Otten, R.H.J.M. and M.C. van Lier
LAYOUT DESIGN FOR BIPOLAR INTEGRATED CIRCUITS.
Ph.D. Thesis. Eindhoven University of Technology, 1976.

OTT80 Otten, R.H.J.M. and A.A. Szepieniec
GRAPH-ORIENTED APPROACH TO THE LAYOUT PROBLEM.
In: Proc. 13th IEEE Int. Symp. on Circuits and Systems; Houston, 28-30 April 1980. New York: IEEE, 1980. P. 956-959.

PER77 Persky, G., D.N. Deutsch and D.G. Schweikert
LTX: A minicomputer-based system for automated LSI layout. J. Des. Autom. & Fault-Tolerant Comput., Vol. 1(1977), p. 217-255.

SIM62 Simon, H.A.
THE ARCHITECTURE OF COMPLEXITY.
Proc. Amer. Philos. Soc., Vol. 106(1962), p. 467-482.

SZE80 Szepieniec, A.A. and R.H.J.M. Otten
THE GENEALOGICAL APPROACH TO THE LAYOUT PROBLEM.
In: Proc. 17th Design Automation Conf.; Minneapolis,
23-25 June 1980. New York: IEEE, 1980. P. 535-542.

WIR71 Wirth, N.
PROGRAM DEVELOPMENT BY STEPWISE REFINEMENT.
Commun. ACM, Vol. 14(1971), p. 221-227.

WOL79 Wolfgang, E., R. Lindner, P. Fazekas and H.-P. Feuerbaum
ELECTRON-BEAM TESTING OF VLSI CIRCUITS.
IEEE J. Solid-State Circuits, Vol. SC-14(1979), p. 471-481.

WUL73 Wulf, W. and M. Shaw
GLOBAL VARIABLE CONSIDERED HARMFUL.
ACM SIGPLAN Notices, Vol. 8, No. 2(Febr. 1973), p. 28-34.

ZIB74 Zibert, K.
EIN BEITRAG ZUM RECHNERGESTÜTZTEN TOPOLOGISCHEN ENTWURF
VON HYBRID-SCHALTUNGEN.
Diss. Technische Universität München, 1974.

EINDHOVEN UNIVERSITY OF TECHNOLOGY
THE NETHERLANDS
DEPARTMENT OF ELECTRICAL ENGINEERING