Simultaneous placement and global routing for gate arrays

Saeijs, R.W.J.J.

Award date:
1986

Eindhoven University of Technology
Department of Electrical Engineering
Automatic System Design group

SIMULTANEOUS PLACEMENT AND GLOBAL ROUTING FOR GATE ARRAYS

by R.W.J.J. Saeijs

Master thesis
reporting on graduation work
performed in 1985
under supervision of prof. dr.-ing. J.A.G. Jess

Abstract

This report is concerned with the combination of placement and global routing in a single design automation scheme to be used in a general gate array context. This is a fundamentally new approach to gate array layout design, based on a hierarchical global routing technique developed by M. Burstein, which is merged with a hierarchical placement approach using min-cut bipartitioning.

However, it has appeared that a straightforward application of this suggested method stands or falls with the applied gate array modelling with respect to both placement and global routing, as well as the relationship between these physical design aspects. The report comprises an extensive reorientation on these gate array design fundamentals, in order to explain that the failure diagnosis of the original modelling is not inspired by a specific type of gate array. On the contrary, it will be shown that the Burstein model only applies to a small class of rather extreme gate array structures with appropriate design prerequisites.

For most gate arrays, and certainly for a technology and type independent gate array design system, a more complicated and abstract modelling is required. Such a flexible system is developed at the Eindhoven University of Technology. The new modelling approach discussed in this report not only concerns the system component reported on here, but also extends to stand-alone placement and global routing phases.

The new strategy proposed is described in terms of consequences of a general model on placement and global routing alone and in combination. Algorithms for necessary additional stages and extensions and adaptations of the original procedure are described and linked with program implementation aspects.

CONTENTS

Introduction

The Eindhoven University of Technology participates in the ICD-NELSIS project on integrated circuit design which is supported by the European Community and the Dutch government (see [12]). In this context, the Automatic System Design group at the Department of Electrical Engineering is engaged in the development of a flexible gate array design.

This system offers a general design automation facility that is adaptable to different integration technologies and gate array types. The present report deals with a new component in this system.

1. Gate array layout design

1.1 Gate arrays from an IC development point of view

Today's process technologies allow for electronic circuits of an ever-increasing complexity to be produced as a single integrated circuit. However, as more and more functions can be brought together on a silicon substrate, these products become less and less general, calling for lower production volumes. Only pre-eminently general-purpose VLSI components such as memories and microprocessors are marketable as standard parts, whereas custom IC's must be developed for dedicated applications. If the demanded volume is large enough, one can afford developing fully custom chips. But for moderate volumes *semi-custom* techniques, partly dating back to the pre-VLSI era, are the only way of cost-effectively filling specialised needs.

The basic idea is to make unique IC's from standard ingredients. There are basically three ways of putting this idea into practice. In order of increasing cost, density and development turnaround time the options are:

- field-programmable components: completely processed IC's allowing differentiation via fuse links etc.
- master-slices: partly processed IC's individuated by unique interconnection masks.
- standard-cell techniques: IC's to be processed with an entirely personalised mask set, but designed using small building blocks from a library of standard layout parts supplied by the manufacturer.

Master-slices are used for both analog and digital applications. Sometimes these two kinds of circuitry are combined on a single slice.

Gate arrays make up the largest part of the digital master-slice market. In principle a gate array master-slice consists of many identical basic layout parts patterned and arranged in such a way that, with appropriate custom masks, elementary (gate-level) digital subcircuits can be configured and interconnected. Gate array applications range from medium-speed arrays used for replacing discrete logic to very high-speed emitter-coupled and current-mode logic arrays for mainframe computers.

1.2 Gate arrays from a design point of view

The increase of the complexity of integrated circuits mentioned in the previous section has also left its mark upon layout design methods. Not only because of the growing role of design automation, but also for designs done mainly by hand decomposition and the use of *macro* libraries containing predesigned layout parts representing often-used functional circuit components have become necessities in order to reduce total design time, costs and, in case of manual designs, design error probabilities.

In particular digital IC design efforts can be reduced enormously this way and the more so because many technology-dependent details can be confined to the once-only layout design for macros which can be stored together with simulation data etc. For analog circuits, where layout parametrisation is much more important, the effect on design efforts will be less spectacular, but if analog circuit design is to profit from (partial) design automation, a similar design methodology seems essential.

Once a collection of macros is available, the circuit for a specific layout design task can be described in terms of a *net list*, consisting of a list of *modules* which are instances of macro types and a list of *nets* that can be regarded as sets of module contacts to be interconnected.

Note: The name "macro" is often used with different meanings. Sometimes it designates a function (i.e. as a part of a circuit), in other contexts the layout representation of the function is meant. This may give rise to confusion, especially for gate arrays where several layout representations may exist for one and the same functional circuit component. In this report a macro will be regarded as a circuit component corresponding with a specific functional behaviour. A second source of confusion will be avoided by using "macro" for generic types (with *formal contacts*) only, as distinct from "module" which is used exclusively for instances of these generic types (possessing *actual contacts* assigned to specific nets).

From the composition of a netlist (corresponding with a macro library) onwards, integrated circuit layout design is reduced to *placement* (choosing locations for modules on the chip) and *routing* (determining wire routes for the interconnections of modules. Usually the routing task is subdivided in a global and a local phase.

The placement/routing design stage has a different aspect according to the integration approach. There are roughly

three possible situations, viz. (in order of increasing design rigidity):

General context (for full custom circuits):

- *Placement*: Macro layouts can be of variable proportions (but are usually constrained to rectangular shapes) and can be placed anywhere on the chip area (obviously, provided that no overlap occurs).

- *Routing*: All space not occupied by layout segments of placed modules can be used for wiring and can be enlarged almost arbitrarily.

Polycell context (e.g. for semi-custom standard cells):

- *Placement*: Macro functions correspond with layout cells which are rectangular with equal dimensions along at least one axis and can be abutted. As to placement restrictions, modules form single or double (back-to-back) rows.

- *Routing*: Wiring channels of variable width between the cell rows are used for routing, in combination with the remaining space on the outside of the cell structure. Additional routing provisions are often available in the form of feedthrough cells that can be used to route efficiently from one wiring channel to another. Not all connections have to be routed explicitly, since powerlines are usually connected implicitly by way of abutment.

Gate array context:

Placement and routing for gate array master-slices are in some respects quite different from placement and routing in case of complete mask set programmability.

- *Placement*: Gate array placement is concerned with assigning a fixed collection of layout parts on the master-slice to different functions, rather than determining completely c.q. largely unconstrained locations for a fixed collection of functional layout parts. But for many gate arrays the elementary layout parts on the master-slice cannot be readily interpreted as objects for placement and routing:

  In order to obtain for these gate arrays layout equivalents of functional macros that can be placed and routed on the basis of a netlist, small patterns of custom mask wiring must be added, representing internal

macro connections. The functionality of a gate array macro then follows from the combination of fixed non-custom mask layout segments on the master-slice and a variable custom-mask layout pattern which can be regarded as a *stamp* corresponding with a unique macro function when placed on a limited set of *legal positions*, viz. at those locations where it covers the same fixed master-slice layout pattern. In general more than one stamp can be used to realise a specific macro function: e.g. mirrored versions to be placed where the master-slice pattern occurs in reverse, but also alternatives with completely different outlines may be necessary (for example if placement restrictions are strongly non-uniform).

However, even if master-slice patterns alone constitute small placeable and routable items in terms of which a netlist might be defined, then many design considerations (for manual and automatic design alike) often lead to a stamp-based design approach in terms of somewhat larger circuit components. A further discussion of this point is deferred until section 3.1.

Note: In this report the denomination "stamp" will be used for any layout counterpart of macros in terms of which a module decomposition for a circuit netlist can be made, even if no actual custom-mask programming (except of course contact hole definition) is concerned in its placement. Again, stamps as generic types are to be distinguished from their instances as physical representations of modules.

- *Routing*: The most important gate array design hindrance is the fact that the routing space is a priori limited because of the fixed master-slice structure. But not only the amount of routing space makes routing for gate arrays a tough design job, for the form in which this routing space is available introduces, for many gate array types, an extra difficulty:

  In the routing contexts described above, net connections are made at the edges of placed modules and wires are embedded, using two or more wiring layers, in channel-like areas without obstacles, abstracting from actual layers and vias with according design rules by defining grids so as to yield simplified routing models. However, gate arrays in their diversity make a very different routing picture:

  - The master-slice principle of stacking custom wiring layers on non-custom layers defining active

areas makes that at least some wiring must be routed over active areas, establishing net connections of modules more on the inside of module allocation areas.

- The number of programmable masks ranges from one to as many as six. So, very different degrees of programmability with regard to the number of wiring layers, contact holes, layer-to-layer vias etc. exist.

- The variable distribution of internal macro wiring after stamp placements produces irregular wiring areas.

- Another unique aspect of gate array routing is the presence of wire segments on fixed positions (defined by master-slice masks) that can (read: must) be used to complete routes in personalisation mask layers. These may include underpaths, fixed contact holes and vias etc. Especially for gate arrays with a single programmable mask such features are of vital importance and exert a decisive influence on the routing task.

- Making optimal use of routing space (which is necessary because of the bounds set by the master-slice structure) may require taking non-trivial design rules into account such as layer- and location dependent spacing rules etc.

Finally, powerlines in custom layers have fixed positions, i.e. are not to be routed explicitly.

- *Contact assignment*: This is a gate array layout design aspect that does not strictly belong to either placement or routing as they were defined in the beginning of this section and nevertheless plays an important role in the design of many gate arrays. Contact assignment can be regarded as an intermediate between placement and routing and is concerned with choosing the geometric counterparts of functionally equivalent macro contacts such as the input connections of a NAND-gate, which are indistinguishable in terms of the functional logic network, but are represented by different layout segments.

  In some logic integration technologies high gate array densities can be attained with very small building blocks that can only be routed successfully thanks to

the interchangeability of connections to these building blocks.

1.3 Gate array layout design automation

Up to now the majority of gate array design tools have been developed by gate array manufacturers and are dedicated to specific gate array types or, at best, to specific integration technologies and/or gate array master-slice architecture styles. As to full design automation software, no flexible technology- and type indepent systems are available as yet. The software package developed at the Eindhoven University of Technology intends to fill the need for such a system [16].

The gate array design system aims at flexibility with regard to foundries and integration technologies. For this reason the system provides a *gate array interface* enabling a user to create the environment for designs using his own kind of gate array and his own macro library. This interface consists of:

- An *image compiler* for a gate array description language. This fairly simple language allows to specify the complete gate array image. It comprises the definition of a variable number of layers, the definition of fixed and/or programmable wires per layer, fixed and/or programmable vias between layers, underpaths etc. A simple hierarchy and the use of various repetition constructs make it possible to describe the entire gate array very concisely.

- A *design rule compiler* for a design rule description language. By way of this language, which is closely connected to the image description language, various design rules such as spacing rules for wires and vias can be described. These rules can be dependent of layer, direction, type of vias, position on the gate array etc.

  The image description and design rule description are compiled into a general data structure which contains the entire gate array landscape, routing constraints etc. This data structure serves as a base for mapping macro instances, local routing etc.

- Also a *macro compiler* will be provided which will allow to build a macro library containing macro images (described analogously to the gate array image), legal positions for placement, terminals for wiring etc.

Once a macro library has been built and the image and design rules have been compiled for a specific type of gate array, the *design interface* can be used for designing circuits with this type of gate array. The design interface consists of:

- A *net list compiler* for a circuit description language. In this language the circuit to be built can be specified in terms of a net list. Additionally, the user can specify restrictions for the design such as predefined macro positions, predefined bonding pad positions etc. The net list compiler checks with the macro library and yields an internal design file format.

The *algorithmic core* of the system consists of:

- A *placer* using simulated annealing for placement of differently sized and differently shaped macro instances optimising both wire length and wire congestion.

- A *hierarchical global router* based on the Burstein global routing algorithm and extended to improve results for single-row oriented CMOS gate arrays ([23], [24]).

- A *local router* using an enhanced Lee routing algorithm ([28]).

- As an alternative for separate placement and global routing: a program for *simultaneous placement and global routing*, combining placement by min cut partitioning and the Burstein global routing algorithm in a hierarchical fashion.

As a backup, the system will provide a *graphical editor* for both placement and (global) routing.

The fourth algorithmic system component is the subject of this report. As yet the first and only attempt to merge the placement and wiring phases into an apparently general hierarchical approach, i.e. to provide simultaneous solution of the placement and wiring aspects of physical design, has been by Burstein [4].

It has turned out that a correct modelling of global aspects of gate array images (with respect to placement and global wiring) is as vital to the success of a technology- and gate array type independent layout system as the general detailed modelling of images which is used (in connection with the image and design rule gate array interface components

mentioned in the previous paragraph) for the local router.

In this respect the gate array model used by Burstein [4] has shown to be inadequate in a general gate array context with respect to placement and global routing individually, as well as with regard to the combination of these phases. In fact, the straightforward application of a placement hierarchy for gate arrays by way of network bipartitioning followed by an explicit placeability analysis on every level of hierarchy owes its applicability, claimed by Burstein [4], to the features of a rather unusual, extreme gate array style.

Unfortunately, generalisation of the required modelling has major effects on both placement and global routing strategies, calling for adaptations and additions. However, on the simultaneous placement and global routing approach repercussions are heaviest, calling for a far more complicated procedure than the one originally proposed by Burstein [4].

The next three chapters present a reorientation on the fundamental nature of the global aspects of layout design for a large range of gate array types. The goal is to bring out the discrepancy between placement and global routing aspects (and their relation) for different gate arrays that follows from an inventarisation of the implications of modelling conceptions hitherto intuively assumed to be generally applicable.

Since placement and routing for gate arrays are strongly influenced by the fixed geometry of master-slices, a good insight into the nature of the images of gate arrays (over the whole scale of types that is to be covered by a flexible design system) is very important. An attempt is made to describe the basic idea and the most fundamental differences between three different over-all layout approaches used for existing gate arrays. Also, it is shown that starting from an intuitive "cell" notion may yield a very different picture for the archetypes that will be defined to capture these basic differences.

2. Fundamentals and variations

2.1 Gate array archetypes

Looking at a completed gate array (i.e. master-slice plus customisation masks layout) as a 2-dimensional surface X-Y, one can distinguish between:

- *Gate area*: the central part of the gate array chip containing the main part of the logic circuit composed of elementary digital gates.

- *I/O area*: the outside of the chip, consisting of bonding pads and associated I/O circuitry.

Looking at the same completed gate array as a 3-dimensional structure X-Y-Z of various layers stacked vertically (i.e. in the Z-direction), one can distinguish between junction areas and function areas (reading areas in the sense of 3-dimensional parts of the image structure):

- In *junction areas* the wiring representing the interconnections between elementary functional gates is concentrated. This wiring is in custom-mask layers but can also be partly in non-custom layers (in the form of fixed wiring segments such as underpaths etc.).

- In *function areas* the elementary gate functions are concentrated. These functional areas contain the active and passive elements on the master-slice (defined by non-custom masks) plus all internal wiring patterns configuring these elements into subcircuits with gate functions. Internal wiring can be in non-custom layers, but may extend to custom masks in case of stamp-based design situations.

It is important to think of the above distinction in 3 dimensions. The reason for this is that if one tries to project these areas onto a 2-dimensional surface (which is what must be done for placement and global routing since these are essentially based on 2-dimensional abstractions of the total design), it turns out that the result is fundamentally not the same for all gate arrays.

In order to explain this in general terms three gate array archetypes will be introduced here, representing three different layout approaches along which the gate array master-slice idea may have been put into practice for specific gate arrays. These archetypes do not correspond directly with specific integration technologies, logic families or customisation features. However, they yield

different image structures with major consequences for placement and global routing.

The essence of the differences between the archetypes is illustrated in fig. 2.1. The figures are no more than a simple indication of the idea behind each type, and by no means reproductions of exact geometrical proportions and relations. The height of the junction blocks should not be interpreted as reflecting only the actual number of wiring layers, but rather as representing a relative degree of routing difficulty: the more layer restrictions, anomalous design rules, irregular wiring bays, routing obstacles, fixed segments etc., the more difficult will routability problems be, indicated by by a shallow junction block. Nor should one conclude from this illustration that junction areas and funtions areas can be discriminated sharply as well-marked rectangular areas in actual images.

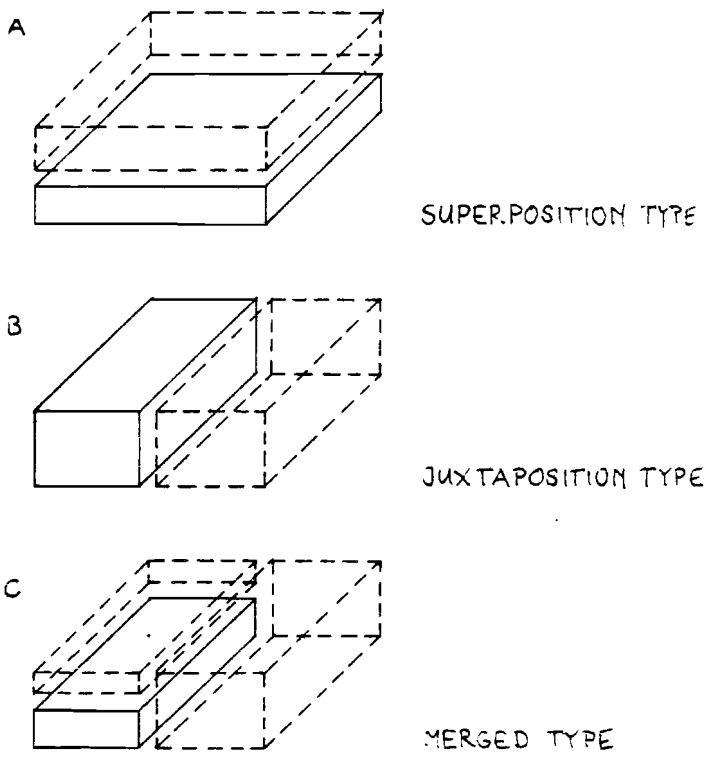


figure 2.1

- *Superposition type*: For this type of gate array junctions and functions are more or less confined to separate layers: interconnection wiring is routed on top of active area parts directly corresponding with placeable modules and connections to placed modules are made vertically by way of a programmable contact hole mask. Unused active area offers full wiring capability.

- *Juxtaposition type*: This type is, in a sense the very antipode of the previous one. Here junction areas and function areas share more or less the same layers: interconnections are wired between, around and, if necessary, through active areas and internal function wiring from stamps.

- *Merged type*: The third archetype is an intermediate type between the above extrema. Wires can be routed over active areas but under difficult conditions (few layers, non-trivial design rules, obstacles etc.). Therefore additional junction areas for unimpeded wiring in all custom layers are provided by keeping parts of the master-slice surface free from fixed function area segments.

2.2 Cells

The reader who is familiar with gate array documentations and specifications will have noticed that the word "cell" has carefully been avoided in the foregoing. The reason for this is the fact that this word is used in three different contexts (image description, placement and global routing), suggesting wrongly (for gate arrays in general) the existence of a universal geometric conception that can serve as a bridge between these three layout design aspects.

The notion of a cell is used to be able to speak of the gate array image in terms of a repetition of some basic building block, many identical instances of which are regularly arranged to form a 2-dimensional array. So, cells are essentially abstractions on behalf of descriptions of gate array aspects in terms of 2 dimensions only.

- *Image cells*: With respect to the image geometry, cells usually correspond with basic layout patterns on the master-slice, i.e. in non-custom masks. According to the kind of gate array these may be I/O cells, gate cells, channel cells etc.

- *Placement cells*: When placement is formulated in terms of cells, a cell corresponds with the largest unit of active area that can be assigned only in whole to a specific function in the placement phase. Together, placement cells correspond with the 2-dimensional projection of the function areas mentioned in the previous section.

- *Global routing cells*: For global routing, cells correspond with lumps of wiring space without serious obstacles where routing is relatively easy, chosen such that actual local routability of individual nets can be effectively captured by concentrating on global restrictions in terms of numbers of nets (see further section 3.2). Roughly, all wiring cells together will make up the 2-dimensional projection of the main junction areas from section 2.1.

  Note: It is recommendable not to think of global routing cells in general as contiguous rectangular areas with sharp limits that can be pointed out in the image (see section 3.2).

Remark: There is even a fourth notion of cells, namely the cell conception which has been used as a language construct in the present version of the image description language. This cell differs from the image cells mentioned in this section, for it is used in a (simple) hierarchy concept and is used also for describing empty areas (with no master-slice features).

The above discussion already hints at the fact that the proportions of these cells and their relation to one another may be very different for different gate arrays. By no means do they collapse into one single cell notion.

2.3 Gate array examples

Example 1 [9]: superposition type (fig. 2.2):

Basic characteristics:

- integration technology: STL.

- customisation masks:

  1. contact hole
  2. metal1
  3. via metal1-metal2
  4. metal2
  5. (via metal2-metal3)

6. (metal3)

Particularities:

- The extra collector contact (h) allows to use Schottky diodes of unused placement cells elsewhere as outputs of a used placement cell instead of the diodes of this cell itself.

- The placement cells can be regarded as primitive gates, though not as direct equivalents of ordinary logic gates: they are *multi-output* gates instead of *multi-input* gates.

- Contact assignment of the outputs (a,b,c,d,e,f) must be done in consideration of wiring efficiency (e.g. by aligning contacts of the same net in different cells).

Example 2 [29]: juxtaposition type (fig. 2.3):

Basic characteristics:

- integration technology: CMOS.

- customisation masks:

  1. metal

Particularities:

- This gate array has fixed wiring segments in polysilicon: fixed *underpaths* in the wiring channels (image cell 2) as well as fixed *feedthroughs* that can be used to pass through active areas (in image cell 1).

- The choice of the global routing cells is in fact partly arbitrary: the subdivision of the channels (which run across the entire gate area) may also be done otherwise, yielding a larger or smaller number of global routing cells along the X-axis.

- Placement cell 1, 2 and 3 correspond with a complementary transistor pair connected to a single polysilicon gate track. Although these pairs cannot be assigned independently individually to different functions (because of shared source/drain diffusions), the three of them may be shared between two different stamps. Therefore they must count as different placement cells.

- Placement cell 4 has two single transistors plus a diagonally opposite pair sharing one and the same polysilicon gate. This configuration has been added in order to simplify the implementation of transmission gates. All four transistors will always be assigned to the same function macro, so together they make up a single placement cell.

Example 3 [32]: juxtaposition type (fig. 2.4):

Basic characteristics:

- integration technology: CMOS.

- customisation masks:

    1. metal

Particularities:

- This gate array has fixed wiring segments in polysilicon, namely fixed *underpaths* in the wiring channels (image cell 2).

- The choice of the global routing cells is in fact partly arbitrary: the subdivision of the channels (which run across the entire gate area) may also be done otherwise, yielding a larger or smaller number of global routing cells along the X-axis.

- Although the two transistors forming one placement cell have no fixed connection, they will always be assigned to the same function (by way of appropriate stamps) because of the complementary nature of CMOS logic. Therefore they can be regarded as forming a single placement cell.

Example 4 [30]: merged type (fig. 2.5):

Basic characteristics:

- integration technology: STTL.

- customisation masks:

    1. contact hole
    2. metal1
    3. via metal1-metal2
    4. metal2

Particularities:

- This gate array has complicated design rules:
  - layer- and direction dependent spacing rules
  - via shadowing rules
  - at most 4 out of 5 multi-emitter inputs (a,b,c,d,e) may be used
- The placement cells can be regarded as primitive gates.
- Contact assignment for the inputs (a,b,c,d,e) is extremely important, not only with respect to contact alignment, but also because of the complicated design rules.

Example 5 [*31*]: *merged type* (*fig. 2.6*):

Basic characteristics:

- integration technology: I2L.
- customisation masks:
  1. shallow N+ diffusion (for emitter definitions and underpaths)
  2. contact hole
  3. metal

Particularities:

- There has been made no a priori distinction between input and output positions on the master-slice: this distinction is programmable by way of the shallow N+ diffusion mask.
- The placement cells can be regarded as primitive gates, though not as direct equivalents of ordinary logic gates: they are *multi-output* gates instead of *multi-input* gates.
- Contact assignment (here of both inputs and outputs) is extremely important since only one layer (metal) can be used to route over the active areas.

figure 2.2

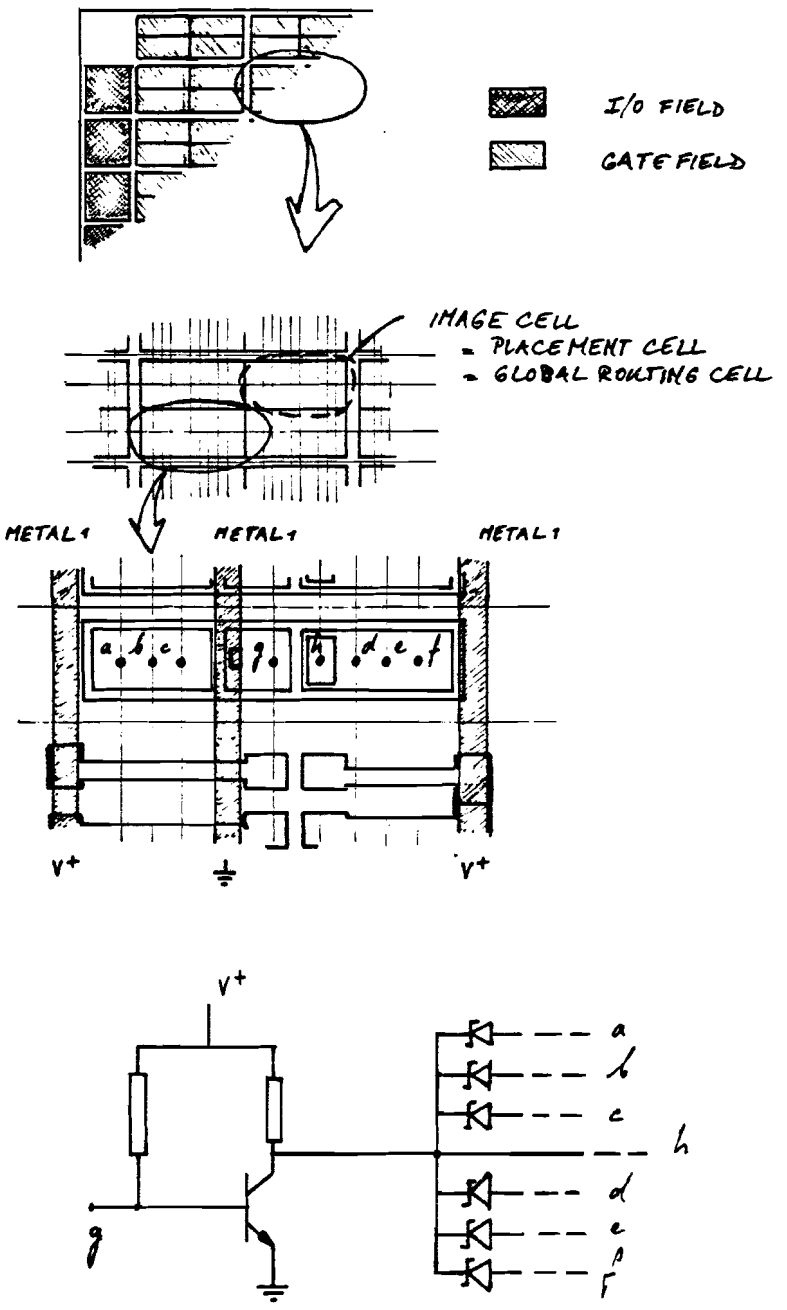

I/O FIELD
GATE FIELD
IMAGE CELL
- PLACEMENT CELL
- GLOBAL ROUTING CELL
METAL 1
METAL 1
METAL 1
a b c g h d e f
V+
V+
V+
a
b
c
h
d
e
f
g

figure 2.3

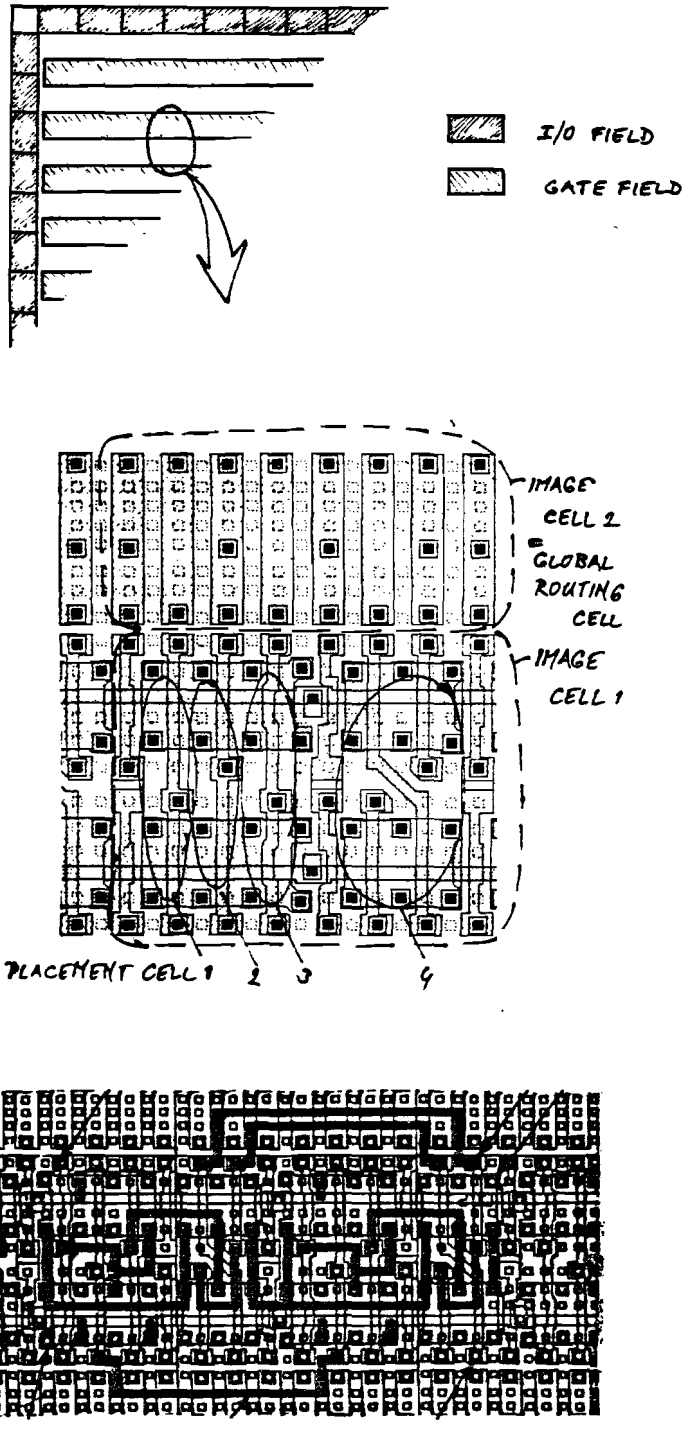

I/O FIELD
GATE FIELD
IMAGE CELL 2
GLOBAL ROUTING CELL
IMAGE CELL 1
PLACEMENT CELL 1
2
3
4

figure 2.4

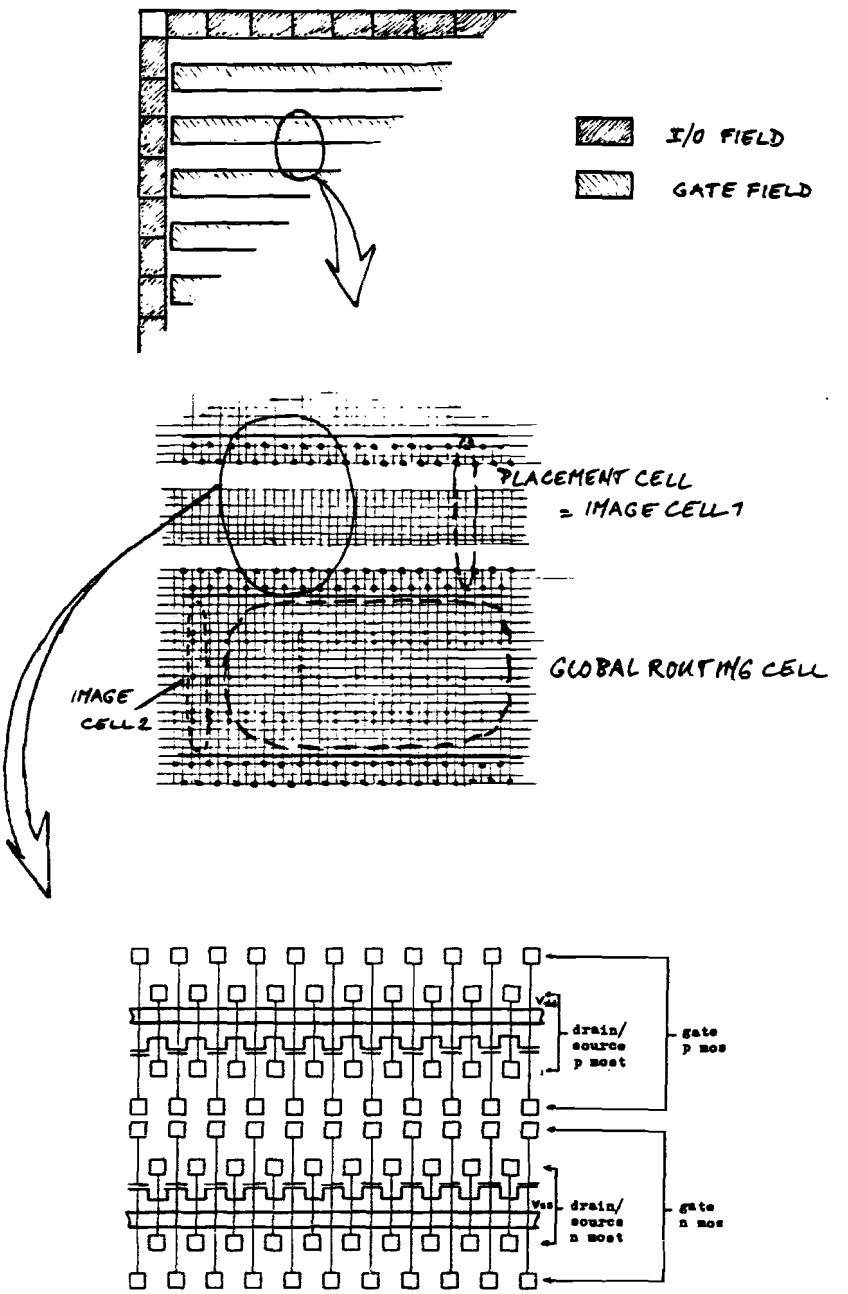

I/O FIELD
GATE FIELD
PLACEMENT CELL
= IMAGE CELL 1
GLOBAL ROUTING CELL
IMAGE
CELL 2
drain/
source
p most
gate
p mos
drain/
source
n most
gate
n mos

figure 2.5

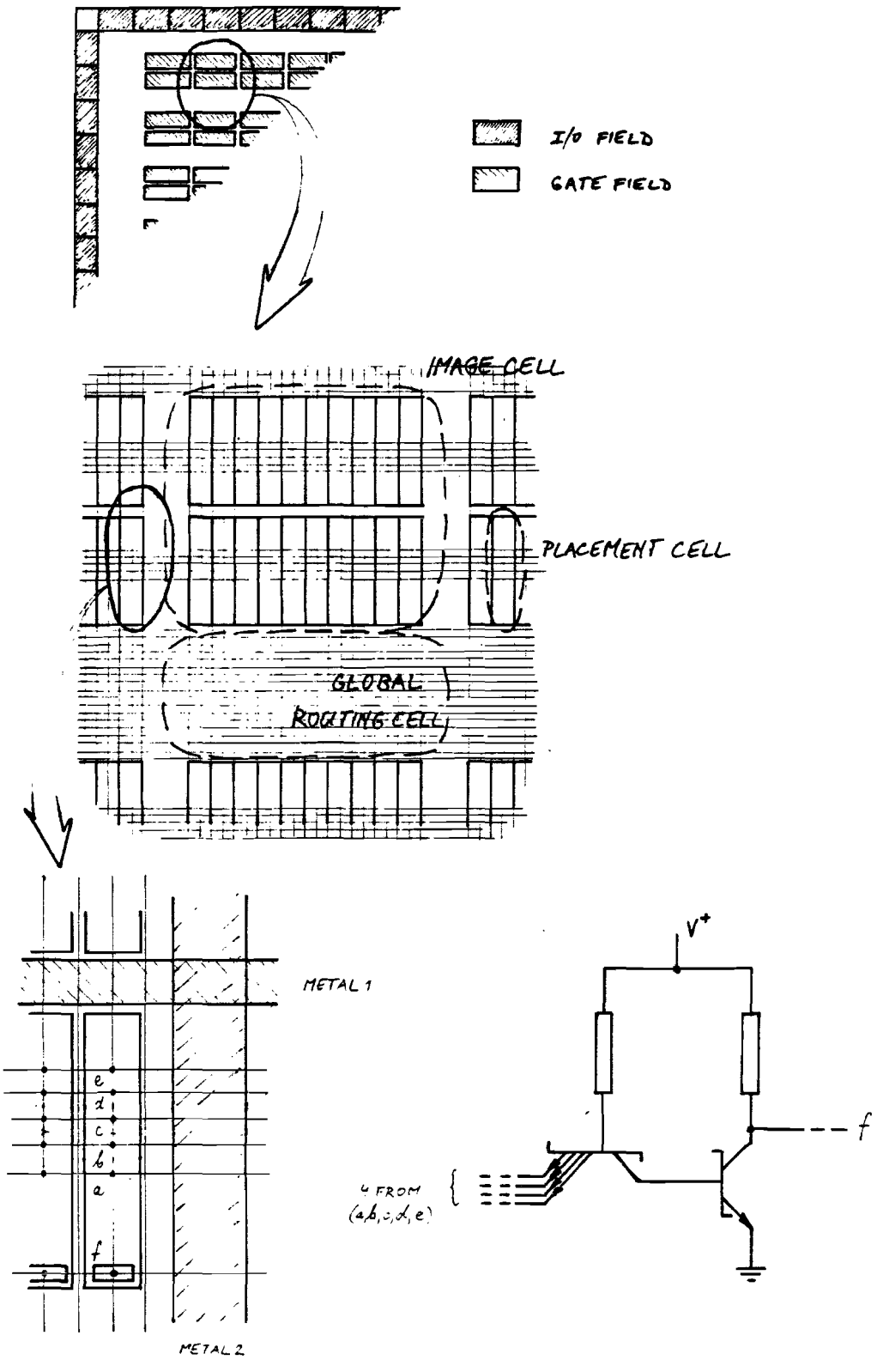
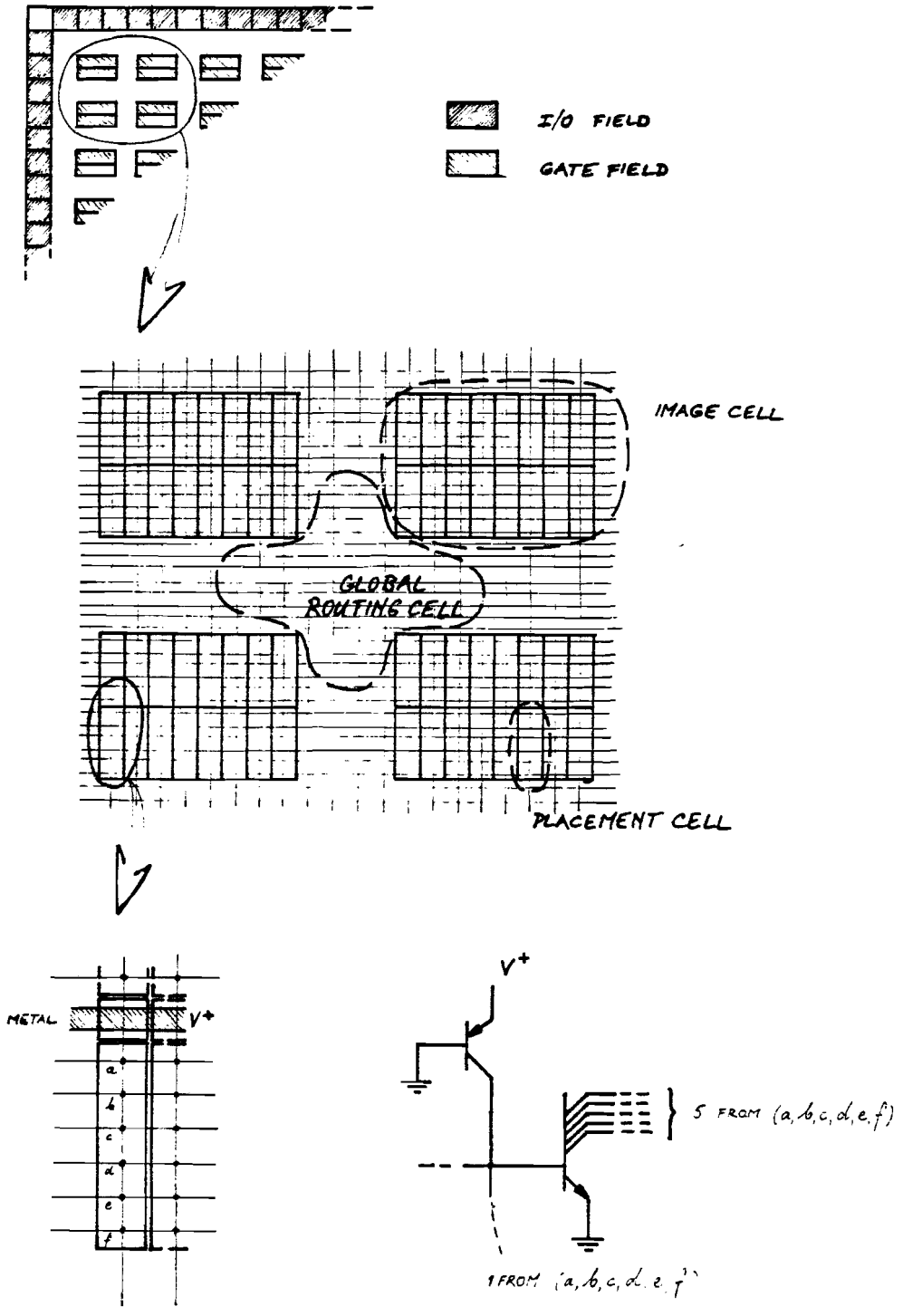

I/O FIELD
GATE FIELD
IMAGE CELL
PLACEMENT CELL
GLOBAL
ROUTING CELL
METAL 1
METAL 2
e
d
c
b
a
f
V+
f
4 FROM
(a,b,c,d,e)

figure 2.6


I/O FIELD
GATE FIELD
IMAGE CELL
GLOBAL
ROUTING CELL
PLACEMENT CELL
METAL
V+
a
b
c
d
e
f
V+
5 FROM (a, b, c, d, e, f)
1 FROM (a, b, c, d, e, f)

3. An informal description

3.1 Placement

The notion of placement cells allows to abstract from 3-dimensional function areas of various proportions, variably interspersed with junction areas:

Placement cells are units for function assignment, which can be thought of as concentrated in a single point in a 2-dimensional abstraction of the gate array image. Placement of a stamp (in the general sense of section 1.2) can now be formulated as *covering* a set of points (read: occupying a set of cells).

The power of a cell-based placement formulation is maximal if single placement cells can be interpreted as individual objects for placement, i.e. if modules from a netlist can be physically realised by one placement cell (in itself or with additional custom mask wiring that makes no use of other placement cells): then placement comes down to assigning modules one-to-one to placement cells.

However, more often than not this is not so for existing gate array types. Therefore, in general a stamp is assigned a *shape* of cells and placing a stamp on one of its legal positions means occupying a cell area with this shape in such a way that a reference point of the stamp shape coincides with the legal position.

Elementary functions corresponding with macros can, in general, be of different *classes*, e.g. gate functions and I/O functions. These different classes correspond with distinct *fields* of placement cells, for example a gate field (corresponding with the central gate area) and an I/O field (I/O area on the outside of a chip). Since every macro function must belong to exactly one class, stamp choices for macros must belong to exactly one, corresponding, field: legal positions per stamp are such that only cells belonging to the stamp's field can be occupied by its placement.

Placement within different fields can be regarded as a collection of separate placement problems, one for every field, only related by the use of some combined objective function (see section 7.1) to be optimised.

Note: There may be more than two fields, for instance an extra gate field corresponding with areas dedicated to special gate functions such as flip-flops. Also, somewhat artificial fields may be used in a second design phase for some gate arrays (see chapter 10).

It will be assumed that legal position restrictions are the only kind of *independent* placement restrictions for individual stamps. Furthermore, it is assumed that the only kind of *interdependent* restrictions for placement of modules together is that stamp shapes must be placed without overlapping.

The nature of placement objects and restrictions for the basic gate array types is given below:

- *Superposition type*: Because junction blocks and function blocks are essentially stacked, they can be distributed uniformly over the chip surface close together, forming more or less contiguous areas, subdivided in small cells of equal proportions such as in example 1 of section 2.3. This results in relatively simple placement restrictions for both single-cell placement and placement of multi-cell stamps.

- *Juxtaposition type*: Placement for this archetype is essentially stamp-based, because:

  - A juxtaposition structure is often used with integration technologies that don't have a natural logic primitive (as with for example I2L, STL and STTL where all logic is reduced to NAND functions) which can be compactly realised by using area-efficient diffusion structures (multi-collector transistors, multi-emitter transistors and multi-anode Schottky diodes respectively).

  - For such technologies (CMOS, NMOS etc., but also ECL and CML) elementary gate functions must be physically realised by custom-layer connections of very small building blocks (complementary transistor pairs, loads and drivers, single transistors and resistors, respectively).

  - In order to keep function areas as compact as possible, often-used connections are made in master-slice masks (e.g. common source/drain diffusions and common gates for CMOS). Therefore the above-mentioned very small building blocks cannot be assigned to functions independently. Assignment must be done for all building blocks of a function at once, and in such a way that internal wiring is as compact as possible.

  - For the optimisation of internal wiring of stamps very specific gate array features can be used.

Comparable results cannot be expected from a general gate array system.

- *Merged type*: Usually this type of gate array has placement cells which in themselves could be taken as placement objects (such as in example 4 and 5 of section 2.3). However, the following design considerations call for a stamp-based approach (especially for a flexible design system):

  - Placement of individual placement cells should be done taking cell alignment and contact assignment into account. These aspects are extremely gate array type dependent.

  - Similarly, obtaining good routing results in the narrow junction areas on top of the function areas is hardly possible for a technology- and type independent gate array system: the special situation created by contact assignment and non-trivial design rules cannot be captured effectively in general heuristics.

  - Stamp wiring includes the wiring of contacts to the main junction areas (wiring channels). So, after stamp placement the remaining routing job is essentially concentrated in wiring areas with more general routing properties. Of course, any usage of space left over on top of active areas is welcome, but the vital role of specific features of individual gate array types for over-all routability is greatly reduced in this way.

The following reason for the usage of stamps apply to all archetypes:

- For the stamp configuration specified by a manufacturer in the gate array macro library, a specific electrical performance can be guaranteed (with regard to timing etc.).

- The combination of placement cells assigned to the same function provides a natural clustering, reducing the size of the placement problem and probably improving total placement results.

- A circuit designer usually wants to look at his design in terms of a variety of functions, i.e. macros corresponding with stamps of variable proportions.

3.2 Global routing

With appropriate global routing cells it is possible to abstract from wires of individual nets in 3-dimensional routing areas by formulating a global routing problem of reduced resolution in terms of 2-dimensional areas in which several nets are embedded together:

It is assumed that the areas on the gate array image that correspond with global routing cells are roughly positioned in relation to each other in rows and columns. Thanks to the fact that global routing is concerned with the central part of gate arrays only (which, in terms of global resolution, has a regular uniform array-like structure), one may forget about actual proportions and shapes and think of the global routing structure as a 2-dimensional array of unit cells as drawn in figure 3.1.

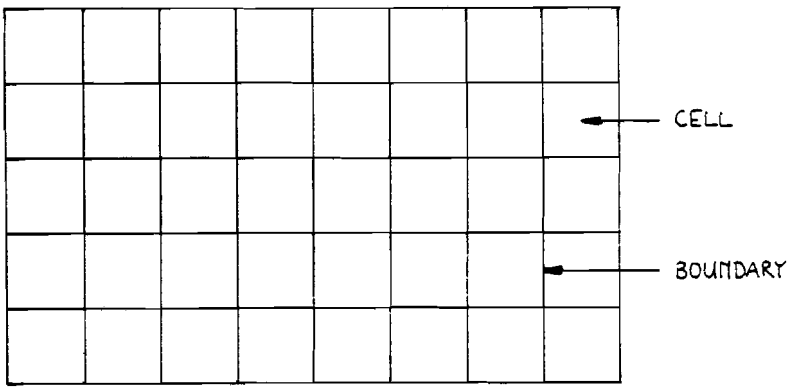


figure 3.1

The basic idea of global routing for gate arrays is to capture the effects of local routing constraints in the form of *capacities*, yielding global routing constraints that can be expressed in terms of *numbers of nets*, instead of single wires.

The most important type of global routing constraints reflects the estimations of the number of nets (whichsoever) that can be routed from one global routing cell's wiring area to adjacent global routing cell areas. These constraints are represented by *boundary capacities* defined for all boundaries in the abstract representation of the global routing structure (fig. 3.1).

Apart from restrictions on routing *between* cells, there may

also be constraints concerned with routing *in* cells, which are used to avoid non-routable accumulations of nets entering a cell from various directions. In general, this can be done by defining *cell capacities* with respect to certain combinations of global routes of different nets making use of a cell. These capacities are based on estimations of the number of nets that can be globally routed with respect to a cell in a certain way (which is to be discussed later on).

For the global routing problem actual image segments to be interconnected (after having been assigned to specific nets by placement) are represented by *terminal cells* in the global routing cell array. The presence of a terminal cell for a certain net implies on a local routing level that connections to actual image segments assigned to this net can be made in this cell.

In general, these segments may embrace larger portions of the image, or consist in a tree of several electrically connected segments in different layers (possibly including stamp wiring in custom-mask layers), in such a way that connections can be made in more than one global routing cell. In this case these segments c.q. segment trees can be used to route this net from one cell to another without making use of free space open to all nets, the usage of which is constrained because of boundary capacities. In principle, these segments and segment trees can be regarded as abstract equivalent positions for this specific net, i.e. as net-specific fixed global route parts. They will be called *multi-cell terminals* in this report. In a general context, such fixedly interconnected terminal cells need not be adjacent.

The global routing task now consists in interconnecting all single-cell terminals and multi-cell terminals by constructing a global route in terms of crossings of cell boundaries. The only *interdependent* constraints for the global routes of all nets together are assumed to be:

- The number of nets crossing a cell boundary (without using net-specific fixed interconnections between cells of a multi-cell terminal) should not exceed the boundary capacity.

- The number of nets featuring certain global routing characteristics in a cell should not exceed the cell capacity.

In most cases capacities can only be estimations. Then "should not exceed.." must be understood as "should

preferably be at most..", and one might add that the consummation of capacities should be judged in terms of a continuous scale of *cost values* rather than in terms of *overflow* or no overflow (i.e. excess of capacity or not). Therefore, capacities need not be integer-valued, but might be real-valued in a general gate array context.

The value of a global routing stage stands of falls with its relevance for the subsequent local routing phase. Therefore the main bottlenecks on the local routing level should correspond with cell boundaries on the global routing level, whereas global routing cells themselves should contain no serious obstacles. Also, local terminal segments can only be represented by terminal cells if no predictable obstructions will prevent a local router from actually establishing connections in these cells. These requirements yield quite different global routing definitions for the basic gate array types:

- *Superposition type*: Here wiring areas are in principle superimposed on active areas and routing is done in two (or more) custom layers, usually using different layers for X-direction and Y-direction wire segments ("jogs", "wrong way wires", are prohibited). In consequence, local routing can be formulated in terms of X- and Y-direction *tracks* running over the entire chip surface which can be regarded as parallel lines in one direction per layer to be joined by programming vertical vias (figure 3.2).

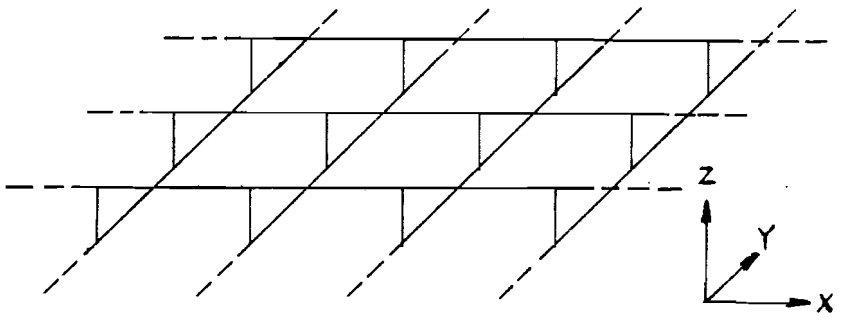


figure 3.2

The initial image is as uniform as can be: not even powerlines form obstacles, as they run parallel to tracks in the corresponding layers. Connections to macro functions are made vertically (i.e. in Z-direction) at single points on the lowest-layer track lines by exposing active segments by means of

programmable contact holes.

By keeping the track convention for stamp wiring too, the only effect of placement on the uniformity of the wiring substrate is that some tracks may be blocked along a short distance, because they are used for internal macro connections at that spot.

- *Global routing cells*: Because the custom layers form one contiguous wiring area without a priori blockades, the choice of global routing cells is somewhat arbitrary: boundaries can be chosen as sharp lines on the chip surface at almost any position *between* two track lines.

- *Terminals*: It is convenient to choose global routing cells equal to placement cells. Thus all contacts of a module assigned to a single placement cell correspond with the same terminal cell, keeping open the possibility to postpone contact assignment till after the global routing stage. Even if stamps occupying more than one placement cell are placed, contacts can always be represented by one single terminal cell. The only possible exception occurs if (large) stamps are used for realising functions with internal feedback connections (flip-flops etc.). In this case functional contacts correspond with internal wiring extending to several terminal cells. As long as this is not the case, no net-specific facilities need be represented in the global routing model.

- *Boundary capacities*: For the superposition archetype both X- and Y-boundary capacities can be regarded as *track capacities*.

- *Cell capacities*: The notion of cell capacities can for superposition gate arrays, provided that the above-mentioned desing restrictions are prescribed, be applied to incorporate via conditions (due to technology rules) in the global routing problem definition. This is done by specifying *via capacities*, bounding the number of nets bending within a cell (because each turn of wire is realised by a via). The number of vias that can be placed in a cell can, in general, only be estimated as a function of the track capacities and technology dependent "shadowing" rules [20].

Summarising the foregoing, superposition type gate

arrays with appropriate uniform image structures and multi-layer programmability can yield a relatively simple global routing problem definition which, by the way, can be extended directly to local routing by decreasing the number of tracks per cell until single-track resolution (in this sense it preserves a certain resemblance to the routing of channels and switch boxes for full custom designs): c.f. Burstein [5].

- *Juxtaposition type*: This type of gate arrays has lumps of junction area and function area next to each other instead of on top of each other. There are basically two ways in which the alternation of active areas and wiring areas on the chip's surface can be structured (taking only the gate area into account, because global routing is concerned with the central substrate area only):

  - *Row structure*: active areas are arranged in rows, with intervening channel rows (fig. 3.3A).

  - *Island structure*: active areas form single islands, surrounded by channel rows and columns (fig. 3.3B).

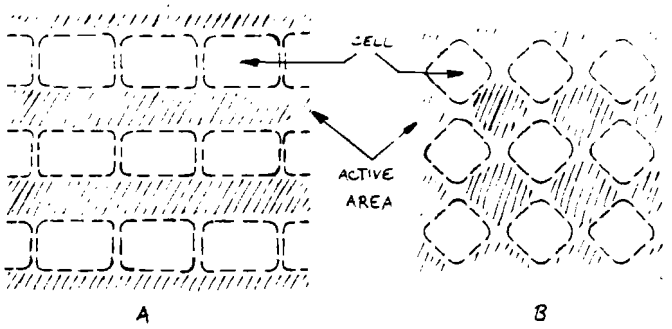


figure 3.3

Routability for this archetype must often be achieved with no more than one custom mask, making optimal use of fixed wire segments.

  - *Global routing cells*: Active areas are essential blockades for local wiring. Therefore cells on the global level should not contain active areas. This yields the following rough cell definition:

  - For row-structured gate arrays of the juxtaposition typem, global cells are formed by cutting channel rows in an adequate number of parts.

  - With island-structured gate arrays the intersections of X- and Y-direction channels form the heart of the global routing cells.

- *Terminals*: Connections to placed stamps can usually be made at the edges of the channels at specific positions (for instance fixed vias to polysilicon and diffusion layers coming up on both sides of channel rows as in the CMOS examples 2 and 3 of section 2.3), but sometimes also to internal stamp wires that are routed by way of the channels.

  Especially with row-oriented image structures, contacts can often be made from different sides, i.e. in different channels. For such gate arrays it is extremely important to take account of the net-specific global routing features that follow from this fact. Often, routability can only be achieved for such row-structured gate arrays by making optimal use of net-specific feedthrough facilities for jumping from one channel row into another.

- *Boundary capacities*: For island-structured gate arrays, both X- and Y-boundary capacities are essentially *channel capacities*, whereas boundaries for row-oriented juxtaposition types reflect channel capacities in one direction, and *feedthrough capacities* in the orthogonal direction.

  Remark: Sometimes, short-distance connections between adjacent stamps in a row structure can be made over the active area. Such additional routing possibilities cannot be taken into account in a global routing stage. Therefore such connections should be established beforehand by introducing a short-range local routing phase *before* the global routing (of relatively long-range interconnections). This problem is discussed further in the next paragraph on routing of merged types for which it has a more prominent role.

- *Cell capacities*: It is obvious that via capacities are nonsense in the context of a juxtaposition

archetype. However, the notion of cell capacities can here be applied in another way for about the same purpose: avoiding that too many nets are crammed into one global routing cell, yielding a locally non-routable situation. This can be done by letting cell capacities be consummated by all nets in a cell (except in the trivial situation of a net with a terminal cell which is not used, i.e. not connected explicitly by a global route in terms of boundary crossings), instead of bending nets only. So, cell capacities can be regarded as *over-all capacities*, limiting the total number of nets that must be routed in this cell (in no matter which way) by a local router.

- *Merged type*: As regards the subdivision of the chip surface in wiring channels and active areas, the merged type resembles the pure juxtaposition type: the merged type too can be either row-oriented (e.g. example 4 of section 2.3) or island-oriented (example 5 in the same section).

  However, the active areas of merged type gate arrays are subdivided in smaller parts which can be assigned to different functions and must be interconnected afterwards by routing over the active areas. For example: the active area rows of a row-structured merged type image may as a matter of fact consist of two rows rather than one if placement cells are placed back-to-back as in example 4 of section 2.3. But such a back-to-back double row should correspond with a single global cell boundary because routing over these double rows is quite more difficult than routing in the intervening channels.

  Usually, automatic routing for this type of gate array is subdivided in two separate routing steps ([11], [14]): one step for routing over the active area and a subsequent step that accepts terminal positions on the edges of the channels that have been determined in the first step. The second step has to do with routing in channels only and consists of a global and a local routing phase.

  In a technology- and type independent design automation system, the first step described above cannot be performed because of its heavy dependence on details of specific gate arrays. In section 3.1 it was argued that a stamp-based apprach may offer a way out of this difficulty by reducing the role of routing over active areas. However, in order to establish the link to the

above-mentioned second phase, and thereby to global routing, some adaptation is till needed after placement has been done. This adaption can have the form of a short-range local routing in the active areas, serving two purposes:

- making connections between adjacent stamps by means of short routes over the active area: the corresponding contacts should either disappear for the global routing task or, if more interconnections of the same net are still to be routed, appear as net-specific global features in the sense of multi-cell terminals.

- routing local terminals in the middle of the active areas to the edge of one or more channels, so that these can be meaningfully represented by global terminal cells corresponding with the channel parts concerned.

After placement and the additional step sketched above, merged type global routing is essentially the same as global routing for pure juxtaposition type gate arrays.

Note: If two layers are available for the remaining routing in the channel areas (e.g. example 4, section 2.3), then a design convention as in the case of the superposition archetype might be applied on the local routing level. However, the relationship of the global and the local routing level is completely different: there is no uniform structure of tracks as lines running over the whole chip with programmable contacts as points on every track, and cells are no rectangles neatly subdividing one contiguous wiring area. Global routing bends cannot be translated straightforwardly into local turns of wire. Therefore, the via capacity concept cannot be used here: cell capacities must represent over-all capacities as with juxtaposition gate arrays.

3.3 Combination of placement and global routing

From the previous two sections and a retrospection of section 2.3, it follows that there are considerable differences in the nature of placement and global routing for the three gate array archetypes. Especially the relationship between these design aspects depends heavily on the image structure. There is no universal relationship between the application of cell notions to either design stage.

The influence of placement results on the global routing situation is discussed below, including the answers on the question whether this influence can be described on a global level, or whether it must follow from local-level image inspections.

- *Superposition type*: This type of gate array is the only type for which placement cells and global routing cells may be equal, as was argued in section 3.2. The effect of placement in terms of global routing cells consists in the local blocking of tracks for global wiring through the cells. The correspondence of the above-mentioned cell notions in this case, allows to predict these blockings without knowing where a stamp is placed: the number of blocked tracks depends on one stamp only.

  However, the blocking effect of two neighbour cells must be combined to yield boundary capacities. In principle, this depends on the way how blocked tracks on opposite sides are positioned with regard to each other. In case the number of blocked tracks is variable, this implies that in general the boundary situation must be examined on the local level. Only if blocked tracks are concentrated (forming a "wall" extending in one direction with increasing number of blocked tracks), then it is possible to calculate the boundary capacity by looking at the *numbers* of blocked tracks only. This is for example the case if the track usage is as in fig. 3.4. It is not the case in example 1 of section 2.3.

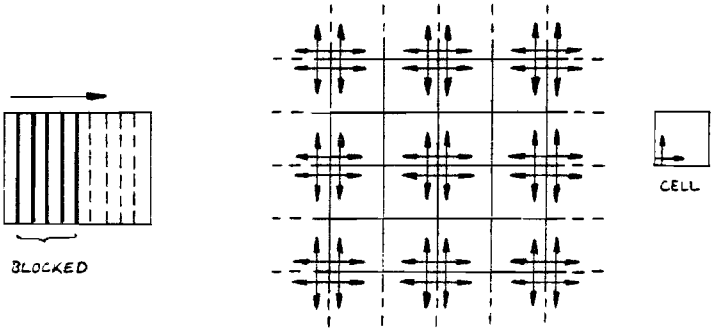


figure 3.4

Via capacities are calculated from the number of remaining tracks within a cell (i.e. initial number

minus blockings) for both directions.

Terminal cell positions are independent of the environment in which stamps are placed.

- *Juxtaposition type*:

  - Row-oriented structures: Usually the influence of placing stamps on capacities as well as terminal positions cannot be specified in global terms because the position of a stamp with respect to global cells in not known in advance. Consequently, the reductions of channel capacities (due to internal stamp wiring routed in the channels) and feedthrough capacities (due to usage of active areas for macro functions) must be determined by local image inspections. The same holds for cell capacity reductions and the assignment of contacts to global routing cells (including net-specific feedthrough possibilities). An additional reason for composing the global routing landscape from local-level inspections is the role of short-range connections in the active areas (see section 3.2).

  - Island-oriented structures: Here, stamps can be related to channels in both directions. Therefore, positions with respect to global routing cells follow directly from stamp definitions and are irrespective of actual locations.

- *Merged type*: The need for a separate short-range local routing phase preceding global routing discussed in section 3.2 clearly indicates that placement effects must be translated for a global routing phase by explicit inspection of the local image situation.

4. A formal model

4.1 Placement

- *Notations*:

  G : 2-dimensional representation of the gate array substrate.

  $X = \{x_i\}$: set of modules for a specific design.

  $m(x)$: macro function of a module $x \in X$.

  $M = \cup_i \{m(x_i)\}$: set of macros.

  $S(m) = \{s_i(m)\}$: set of stamp choices for physical realisation of a macro $m \in M$.

  $S = \cup_i S(m_i)$: set of stamps.

  $\Psi(s) \subset G$: shape of area occupied by a stamp $s \in S$.

  $\Lambda(s) = \{\lambda_i(s)\}$, $\lambda_i \in G$: set of legal positions of a stamp $s \in S$.

  $(s(x), \lambda(s(x)))$: placement of a module $x \in X$.

- *Failure of the Burstein model*: Burstein [4] represents a gate array as a uniform array of cells, i.e. (see figure 4.1):

  $G = [1..p] \times [1..q]$

  This implies that cells from different fields must fit in one and the same subdivision in rows and columns. This is often not the case.

  Stamp shapes are restricted to rectangles. Furthermore, it is assumed that a macro has just one appearance, i.e. just one shape with one set of legal positions.

  Note: This does not imply that there is only one stamp choice for the image. Several macro stamps with the same shape (in terms of cells) might be incorporated in the Burstein model by taking the union of all legal position sets of all stamps as the legal position set for the macro. Together with the restriction to rectangle shapes, this allows to accomodate mirror symmetries and 180-degree rotation symmetries of images and stamps.

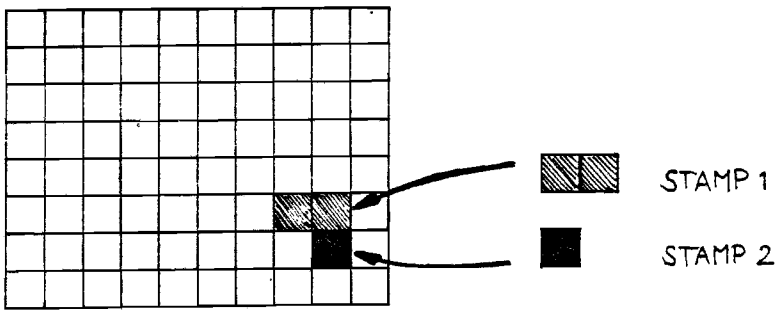


figure 4.1

However, in general different stamp shapes are possible for one macro function (and placement restrictions such as in example 2 of section 2.3 make it necessary to be able to choose). Moreover, shapes need not be rectangular (i.e. in example 4 in section 2.3).

Not so much as a model feature but as a design automation prerequisite, Burstein [4] assumes that the majority of macros are assigned to a single cell. Besides, the shapes of the multi-cell minority are more or less confined to a maximum area of 2x2 cells. Also, macros of the same shape with equal positions are grouped in so-called books, and it is assumed that the number of book types is small. All these preconditions are needed to make a branch-and-bound placeability analysis possible in practice (see section 6.3). Unfortunately, hardly any of them is realistic in a general gate array context.

- *General model*: In general, different fields cannot be fitted into one array of rows and columns. This, plus the fact that placement cells may be very small compared to stamps (example 2 and 3, section 2.3), makes that the general model is formulated in terms of general image coordinates, instead of cells. For example:

$$G = R^2$$

Formal stamp shapes are represented by sets of disjoint rectangles. The same representation is used for representing field shapes on the substrate. See fig. 4.2.

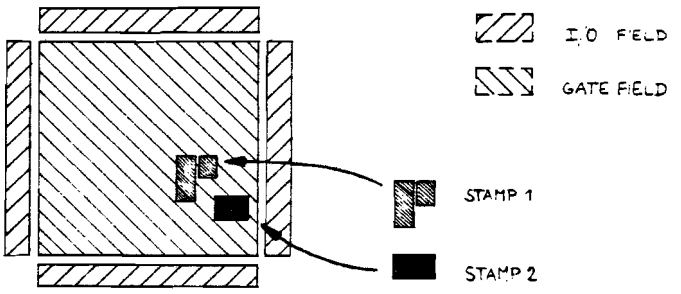


figure 4.2

4.2 Global routing

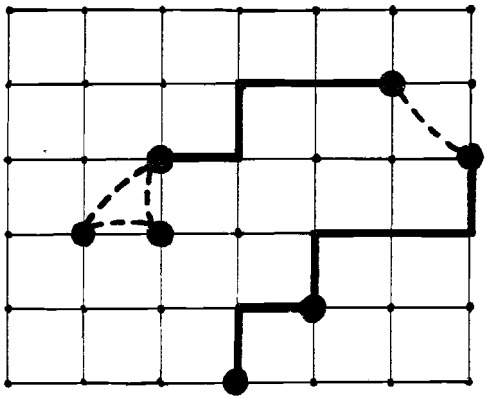
- *Notations*: The global routing model in terms of a grid (analogously to the local routing model in a general context: [16], [28]) is obtained by composing a *lattice graph* G (V,E). This graph has a vertex v ∈ V for every global routing cell and an edge e ∈ E for every cell boundary. So, any two vertices associated with neighbour cells are adjacent.

  b(e) ∈ R : boundary capacity of an edge e ∈ E.

  c(v) ∈ R : cell capacity of a vertex v ∈ V.

- *Failure of the Burstein model*: With Burstein ([4], [5]) the netparts to be interconnected are *single-cell terminals*, represented by single vertices in the lattice graph. A route of a net is a *subtree* of this graph containing all terminal vertices. The global routing problem consists in determining routes for all nets satisfying boundary and cell capacity constraints associated with edges and vertices respectively.

- *General model*: In a general context the only difference is that netparts to be interconnected are in general *multi-cell terminals* (sets of terminals already interconnected due to net-specific routing features). Thus, a global route will in general be a *forest* of subtrees of the lattice graph with the property that separate subtrees share cells of multi-cell terminals in such a way that all multi-cell and single-cell terminals are connected.



LATTICE GRAPH WITH GLOBAL ROUTE OF A NET
(INCLUDING MULTI-CELL TERMINALS)

figure 4.3

4.3 Combination of placement and global routing

- *Failure of the Burstein model*: Burstein [4] has one cell definition which is used for both placement and global routing. Previous sections have clearly shown that this is absolutely impossible for juxtaposition type and merged type gate arrays: the very nature of the layout approach for these archetypes, which is based on a division of the master-slice surface in main wiring areas and active areas, implies that placement cells and global routing cells correspond with different parts of the chip surface. By the way, this is essentially the same as with placement and global routing in full custom situations.

  The Burstein model can only be applied to gate arrays based on the superposition principle, and even then only if image geometry, mask programmability and design conventions satisfy conditions discussed afore. Altogether, the model lacks the generality that it may suggest at first sight, and its applicability is confined to a rather extreme type of gate array with appropriate design prerequisites.

- *General model*: From section 3.3 it follows that the composition of the exact global routing situation must in general include a local-level inspection of the gate array image after placement (possibly combined with a short-range local routing phase, as was argued in that

section). This, together with the absence of a common cell notion, creates a rather paradoxical point of departure for an attempt to do placement and global routing simultaneously. The following chapter describes how, nevertheless, a combined strategy is possible in the form of a hierarchical sequence of subproblems which starts in a Burstein-like high-level situation and gradually diverges from the original strategy to yield a low-level situation based on the general modelling.

5. A methodology

5.1 A hierarchy of subproblems

The formal gate array layout models composed in the previous chapter, indicate that both placement and global routing are essentially space allocation problems in a 2-dimensional geometry. This space allocation must, in either case, be done simultaneously for a large number of interdependent design elements: with placement these are stamp shapes for modules, with global routing these are global wire routes for nets.

Also, in each case element interdependency imposes constraints on feasible solutions: the combination of space allocations for the above-mentioned elements must satisfy certain *admissability criteria*. Basic admissability criteria for general gate array placement and global routing are, respectively:

- module allocations must be non-overlapping.

- no routing overflows, i.e. no part of the global grid should carry more nets than capacities allow to be wired locally.

The result of both placement and routing is concerned with the over-all *optimality criterion*: minimal total wire length.

A solution method of the combined problem that can be derived from the introduced models and the above criteria must in practice be based on heuristic strategies. One heuristic choice that has been made for almost all similar design automation initiatives up to now, is the definition of separate, subsequent placement and global routing design steps (followed by a final local routing phase). Burstein [4] has suggested a fundamentally new design approach, allowing the integration of these steps into one. His method is based on hierarchical strategies that have been developed individually for placement and global routing independently.

The idea behind these strategies is to apply a divide-and-conquer approach to the 2-dimensional space allocation aspect which is common to both placement and global routing. This is done as follows:

For every element space allocation is performed gradually by stepwise confinement of the space available for allocation. These confinement steps are done for all elements together (because of element interdepency). The intermediate results

between two steps can be formulated in terms of the allocation assignments to parts of the entire gate array. These parts will be called *blocks*.

With every step these blocks are subdivided into new smaller blocks and choosing between these new blocks for every module implies a new intermediate solution. Essentially such a step comes down to allocation with increased resolution. By regarding the smaller blocks into which a large block is subdivided as descendent blocks, a hierarchy of blocks can be defined.

5.2 Bisection hierarchy

Because of the basic rectangularity of gate arrays, blocks are chosen as rectangles with sides along the X- and Y-axis. The highest hierarchical level (root) is formed by a single block, namely the entire chip area. The second level of hierarchy is obtained by dissecting the chip area into two approximately equal parts by way of a *cutline* parallel to one of the coordinate axes.

At any level, the chip area is divided into blocks by means of cutlines along both axes. The next (lower) level's block division is created by cutting rows of block rectangles into halfs by means of a set of cutlines parallel to one of the coordinate axes. Usually both directions are used in turn. The bisection hierarchy is illustrated in fig. 5.1.

5.3 Application to placement and global routing

The definition of the bisection hierarchy, i.e. the definition of the sequence of cuts and the position of cutlines, must be guided by the application of the hierarchy concept to global routing:

Cutlines are chosen so that they fall in between cell rows in the abstract global routing cell array. In this way, bisection blocks on any level correspond with rectangular-shaped subsets of the cell array. These sets will be called *supercells*.

It is possible, analogously to the construction of a global routing problem from a local routing problem, to construct a "superglobal" routing problem in terms of supercells in the supercell array on any hierarchical level:

- *Terminals*: Supercell terminal positions follow from the union of terminal cells per supercell. Multi-cell terminals occuring in more than one supercell correspond with multi-cell terminals on the supercell

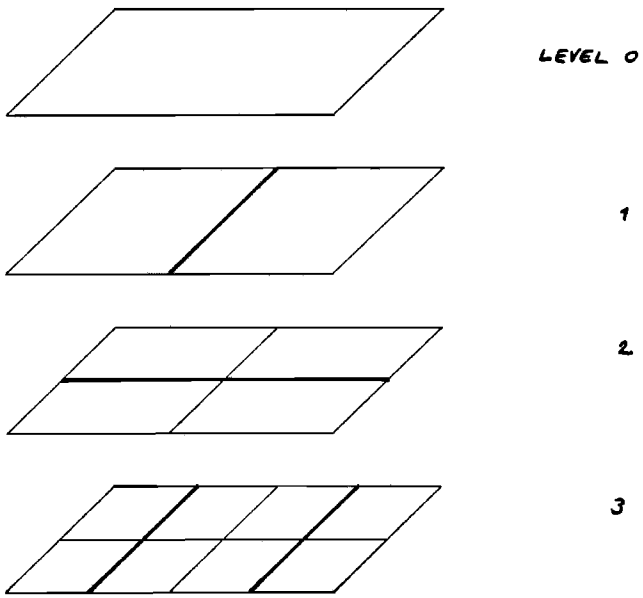


figure 5.1

level.

- *Boundary capacities*: Superboundary capacities should represent estimations of the number of nets that can be safely committed to be routed from a supercell to its neighbour without risking overflows on lower levels later on. Such an estimationi must necessarily be based on heuristics. Burstein [4] suggests a superboundary capacity definition based on the idea of track blocking for superposition type gate arrays (see section 4.3). A discussion of supercapacity definitions is deferred, since the combination with placement introduces a new aspect: supercapacities are to be determined while a detailed placement is yet unknown.

- *Cell capacities*: For cell capacities the same holds. However, in general cell capacity constraints are less vital: boundary capacities are much more important on higher levels.

With respect to placement cells, a notion of supercells is not applicable. This is because in general, some placement cells may be dissected by cutlines. There are for two reasons for this fact:

- Placement cells cannot be arranged in one and the same array of rows and columns (cells from different fields usually have different proportions and arrangements) (see sections 3.1 and 4.1).

- Especially in case of juxtaposition type gate arrays, dissection of placement cells follows necessarily from the fact that cutlines fall between global routing cells. This is because global routing and placement cells are essentially juxtaposed.

Because of these reasons a placement hierarchy with a more abstract relationship to the chip geometry has to be conceived:

The basic generalisation of placement for intermediate hierarchy levels is the *assignment* of modules to blocks without determining actual positions. The assignments essentially imply a confinement of the placement position candidates to locations roughly within the chip regions corresponding with these blocks (this is discussed further in chapter 6). However, on low levels mere assignment is in general no longer meaningful. The consequences of this fact will be discussed in the next chapter.

In the beginning of this section the application of the hierarchy concept to a stand-alone global routing has been discussed. The reduction of actual terminal cell positions to supercell terminals was the essential idea: actual positions within a supercell are not distinguished. This suggests that the general-level global routing problem might already be composed without these exact locations having been determined at all.

This is the key to a simultaneous placement and global routing methodology: regarding contacts of modules assigned to a supercell as terminals for global routing on the supercell level and estimating capacities of supercells and superboundaries yields a generalised global routing problem, although an exact placement has not yet been determined.

6. A placement strategy

6.1 Hierarchy evolution

On the top level, where the chip corresponds with a single block, all modules are assigned to this block by definition. The next level is attained by cutting the chip in two. Solving the generalised placement problem on this level comes down to *bipartitioning* the set of all modules. In general, the supercell assignment on level i+1 can be obtained by bipartitioning the set of modules of the level-i supercells that are cut by a new cutline.

Since modules of different fields cannot be interchanged, bipartitions of different fields are essentially independent (except with regard to the combined optimisation criterion). So, as far as generalised placement restrictions are concerned, the bipartitioning of different fields can be regarded as separate generalised placement problems: with regard to placement the intersection of supercells and fields is important, rather than the supercells themselves.

The generalisation for arbitrary levels of *independent* placement restrictions for individual modules in the form of legal positions is the following:

A module may be assigned to a supercell if and only if its macro function has at least one shape with such legal positions as to be placeable entirely within the contours of the *region* associated with the supercell block. The definition of this region will be discussed later on.

On the other hand, *interdependent* restrictions on supercell assignments are (as a generalisation of non-overlap requirements for the final placement):

The occupation of a supercell should not be too large, i.e. only so many modules may be assigned to a supercell as can be placed together in the end (on the lowest level).

6.2 Mutations

There are three main reasons because of which the afore-sketched hierarchy evolution cannot be applied straightforwardly on low levels:

- At a certain level, further bisection yields blocks smaller than large stamp shapes. Then assigning modules with large shapes is no longer meaningful, since it is absolutely certain that they cannot be placed in the end in the regions associated with supercells on the

new level.

- Even if, judging from their sizes, modules can be placed individually in block regions of a low level, their combined placement may be impossible, because of different shapes and legal position restrictions.

- In case of juxtaposition of placement cells and global routing cells, a low-level assignment of modules (which will finally have to occupy placement cells) to global routing supercells (which are small sets of global routing cells) is per definition no longer meaningful. In this case a final placement must have stamp locations associated with (super)cell boundaries rather than global routing (super)cells.

6.3 The impossibility of a placeability analysis

Placement procedures based on a bisection hierarchy have been used successfully for separate placement phases in various layout design situations: min-cut placement (Breuer [2]), placement by partitioning (Corrigan [10]). However, in these situations none of the above-mentioned three difficulties occurred. Burstein [4] has made an allowance for the first two only, in the following way:

The interpretation of assigning modules to a supercell is strictly geometric: modules assigned to a module will eventually be placed somewhere within the block corresponding with the supercell. For this purpose an explicit placeability analysis is performed, ensuring that modules assigned to one and the same supercell can be placed together in the supercell area in at least one way. This is done by trying to construct a legal placement for each new supercell (on level i+1) after the bipartitioning of the modules of an old supercell (on some level i).

The matching algorithm used by Burstein [4] is a combination of an exhaustive search for a matching of multi-cell modules and a single-cell matching using the well-known Hopcroft-Karp algorithm for the construction of a maximum matching of bipartite graphs [15].

The multi-cell matching is done book-wise (see section 4.1) with a branch-and-bound method, ordening books according to size. This method has exponential worst case run time. But Burstein [4] argues that in practice search exhaustion occurs only at low level when the matching problem is so small that even complete search terminates quickly.

This argument relies heavily on the happy circumstance of a

superposition type gate array with one and the same cell definition for placement and global routing and with few multi-cell macros, and many identical sets of legal positions. From chapter 2 and 3 and section 4.1, it follows that such a placeability analysis is infeasible in general.

Burstein [4] uses the described placeability analysis in order to be able to adhere to a strict hierarchy, where hierarchical placement evolutions for different sites on the chip become totally independent once they have been separated by a cutline. If a matching in a block of level i+1 fails, the hierarchy evolution stops at level i: the matching from the previous level (level i) is taken as the final placement for the area corresponding with the previous-level supercell.

Thus, placement evolution of adjacent supercells remains independent, but the hierarchical progress is terminated on a different level for different parts of the chip.

6.4 The impossibility of a strict hierarchy

Unfortunately, the independence of supercells can no longer be maintained in a general gate array context. One reason is the infeasibility of a placeability analysis a la Burstein [4], as was argued in the previous section. However, the impact of the third point mentioned in section 6.2 is more fundamental: combining a hierarchical strategy for both placement and global routing yields, for many gate arrays, a hierarchy interpretation for which final placements under cutlines are unavoidable, so that independence of supercells is impossible per definition. This second reason already holds for juxtaposition type and merged type gate arrays if all modules have single-cell shapes with the same set of legal positions (i.e. ideal placement restrictions)!

The foregoing indicates that a more abstract hierarchy interpretation has to be conceived. On high levels modules are assigned to supercells, but on low levels they are fixed on actual locations that no longer correspond with supercells. High-level assignments are no longer definite decisions confining the remaining set of location candidates once and for all, but act as guidelines for low levels, deviations of which cannot be precluded.

The choice of generalised placement restrictions on intermediate hierarchy levels should be so as to minimise the number and/or consequences of such low-level deviations. The level on which modules can no longer be merely assigned, but have to be *fixed* on actual locations follows from these restrictions: if applying a new cutline yields block regions

too small to contain any shape of a macro, or if the new regions after bipartitioning have been assigned relatively many modules (with a relatively high chance of being non-placeable together), then the assignment hierarchy has to be deviated from. This is done by *fixing* some of the modules, i.e. determining a final placement (choosing a shape and appropriate legal position).

Fixation of several modules together must be done in a branch-and-bound matching scheme. There are two reasons for allowing different modules to be fixed on different levels:

- The worst-case complexity of the branch-and-bound algorithm depends exponentially on the number of pairs of choices for shape and legal position for all modules: complexity reductions as with Burstein [4], due to the single-cell module majority for which a polynomial matching algorithm exists and a book-wise multi-cell matching simplification, don't hold in general. Therefore, the number of modules involved in a matching attempt on a certain level should be as small as possible.

- Premature fixation of modules reduces the final placement optimality.

In general, there are two possible states for a module:

- *Loose*: The module has been assigned to a supercell (i.e. is still part of the partitioning hierarchy).

- *Fixed*: The module has been placed (a shape and legal position have been determined) and no longer falls under the hierarchy.

The generalised placement restrictions (see section 6.1) for the hierarchy evolution (which is concerned with loose modules only) must take the above-mentioned fixation effects into account. This may be done as follows:

- In view of the fact that a final placement may have all modules intersected by cutlines instead of occupying areas between these cutlines, the independent assignment restrictions for individual loose modules is adapted as follows:

  A loose module can be assigned to a supercell (in anticipation of a final fixation possibly under cutlines) if and only if its macro function has at least one shape with such legal positions as to be placeable entirely within the countours of the *region*

associated with the supercell block. This region is an enlarged rectangle around the original block rectangle. It is created by expanding the block rectangle (i.e. multiplying the distance of a rectangle side to the heart of the block rectangle) with a constant *expansion factor* α_X c.q. α_Y:

$$\alpha_X \geq 1, \quad \alpha_Y \geq 1$$

α_X and α_Y allow a parametrisation of the placement strategy for different gate array types (they might even depend on the hierarchy level).

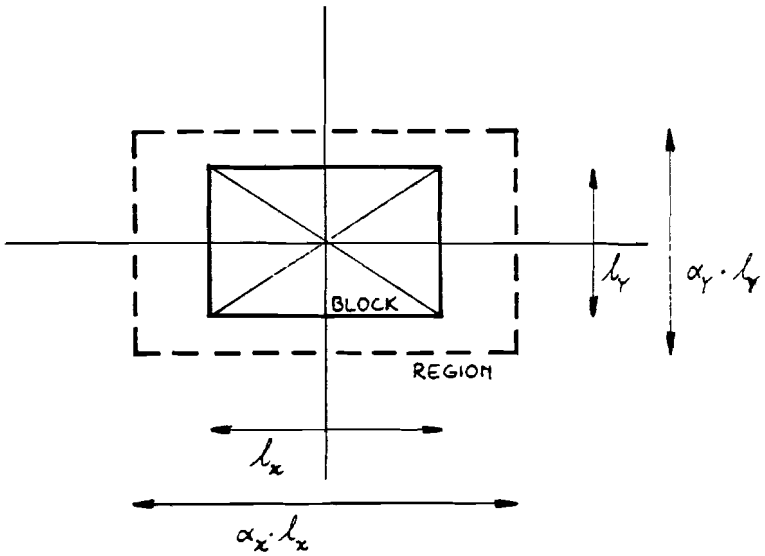


figure 6.1

- $\alpha = 1$: This is a special case corresponding with the strictly geometric hierarchy interpretation from Burstein [4], Breuer [2], Corrigan [10] etc.

- $1 < \alpha < 1.5$: With $\alpha > 1$, regions of adjacent blocks overlap, which corresponds with the interdependence following from the relationship between placement areas and global routing cells (for juxtaposition type and merged type gate arrays) discussed before. Overlap of regions with block regions of non-adjacent other blocks is not meaningful. This imposes an upper bound on α: e.g. if all blocks on a certain level are of equal proportions, forming a uniform contiguous array, then the following condition must be satisfied:

$$\alpha_X \leq 1.5,\ \alpha_Y \leq 1.5$$

Thus, a region overlaps no more than 8 other regions (figure 6.2).

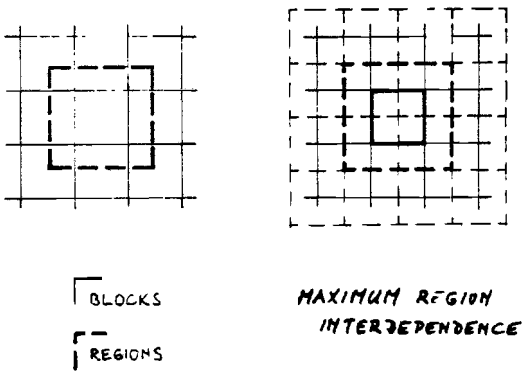


figure 6.2

- α = 1.5: This is another special case, corresponding with a basic juxtaposition type arrangement, where placement cells and global routing cells form similar arrays, but with the heart of placement cells (i.e. active areas) corresponding with the boundaries of global routing cells (i.e. wiring areas). With "ideal" placement restrictions, viz. single-cell modules with equal legal positions, the indicated region-definition postpones fixation of modules to the very last level.

- Since placeability of many diverse shapes with different legal positions can never be guaranteed, restrictions on the assignment of a set of loose modules to a supercell can only be based on the total area occupation:

  The placement hierarchy evolution (which is still applicable in its pure form on high levels) is based on the independence of assignments for different supercells. Therefore area restrictions should be based on the available area per block. For lower-level situations where some modules may already have been fixed, this also holds: interdependent assignment restrictions for loose modules follow from the

available area per *block* (although the independent assignment restrictions follow from the placeability in the enlarged *region*). The available area per block must be reduced by the area occupied by fixed modules.

Note: The available area per block is not the same as the block area: different fields have separate area restrictions (since loose modules of these fields are not interchangeable), based on the area of the intersection of the field and a block.

As to the area occupied by loose modules, this must be calculated from the average area per module (for more shapes with different areas can be chosen). For the calculation of this average only shapes placeable within the block region should be taken into account.

The region definition given before has the following aspects for the transition from one level to the next:

- The cutline (creating two new blocks on the lower level) corresponds with two separate region lines (as part of the region rectangles of the new blocks).

- The old region lines parallel to the cutlines correspond with the demarcations of the new region rectangles on the side opposite to the cutlines, but with different coordinates. See figure 6.3.

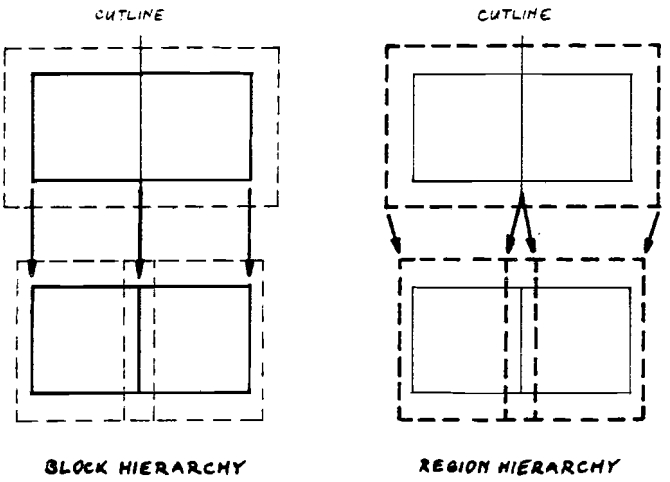


figure 6.3

It is possible to divide the region transition in two phases (fig. 6.4A):

- *Creation* of the new regions due to cutting the old block into new half blocks. The old region lines stay where they are.

- *Contraction* of the outer boundary lines (i.e. the old region lines) of the new regions towards the cutline.

The use of this distinction follows from the different relationship of region placeability in both phases with bipartitioning. Let this be illustrated by figure 6.4B, where the region rectangle at the previous level is subdivided by the region lines from the new level:

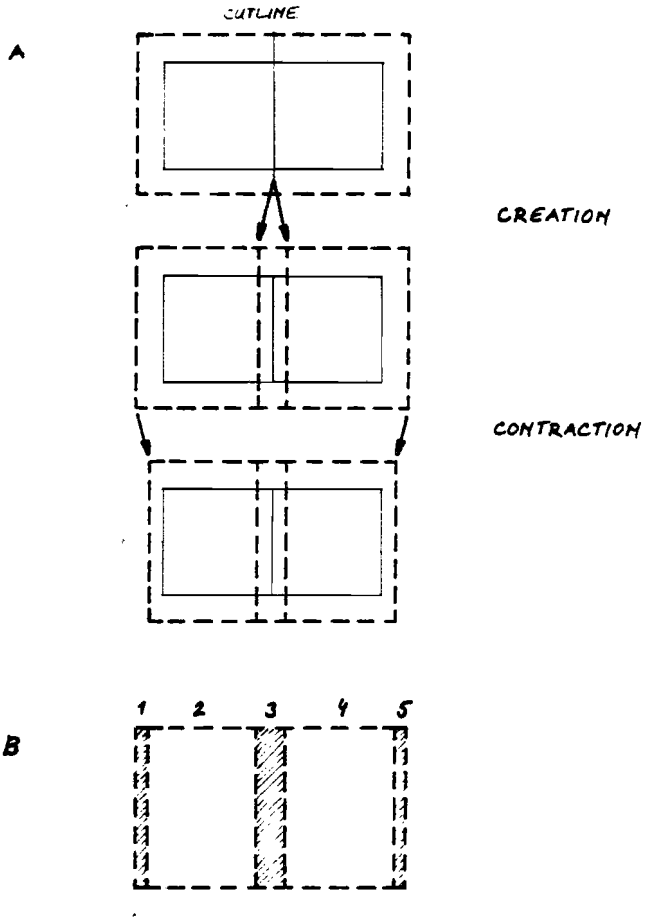


figure 6.4

- Loose modules assigned to the old region must have at least one legal placement within the old region rectangle (i.e. area 1+2+3+4+5). For those modules that cannot be placed in either new region (i.e. 1+2+3

and 3+4+5) after the creation phase, it must hold that all legal placements within the old region rectangle must overlap with area 1+2 as well as 4+5. From this it follows that these modules are large modules, sure to be intersected by the cutline.

- For modules that become non-placeable on the lower level only after contraction (i.e. non-placeable in 2+3+4), it must hold that any placement overlaps area 1 or area 5. So, these modules are smaller and will probably be placed under the old cutlines, corresponding with the block boundaries opposite to the cutline.

From the above it follows that it is advantageous to do bipartitioning between the creation phase and the contraction phase: large modules are fixed beforehand, whereas smaller ones for which a choice between locations on either side of the cutline may still exist are fixed afterwards. So, partition balancing restrictions are derived from the intermediate region definitions between creation and contraction!

Finally, the contraction phase for a block of a certain level and the subsequent creation phase of its lower-level descendents may be joined. In this way creation of half regions for a block is done immediately after this block has been assigned loose modules by bipartitioning its "parent" block.

Thus, modules that are non-placeable after contraction on level i and modules non-placeable in either half region after creation on level i+1 are treated in the same *rearrangement* phase. This corresponds with the fact that these are probably of comparable sizes, because the former are relatively small with regard to the level i block whereas the latter ones are relatively large with respect to the level i+1 (half) blocks (see above).

The above-mentioned *rearrangement* phase consists in the following:

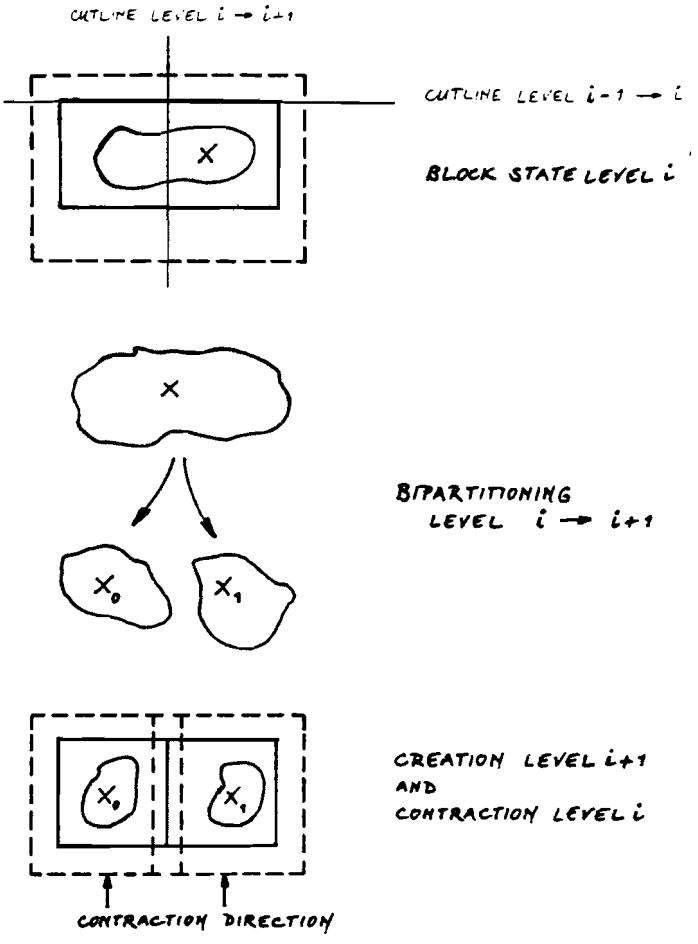
Modules that are not placeable in lower-level descendents of a block region on a certain level any more can not be assigned meaningfully as loose modules on these lower levels. Therefore an attempt to fix their placement is done by way of a branch-and-bound search of placements within the old region. Modules that are placeable in lower-level regions remain loose.

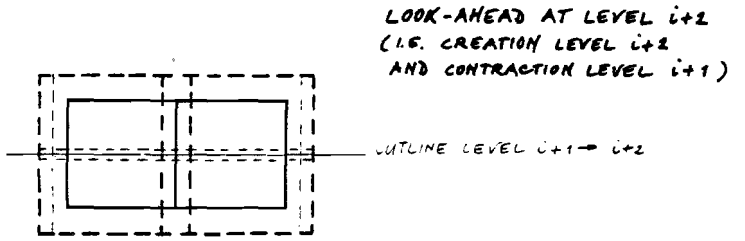
If the fixation attempt in the old region fails for some

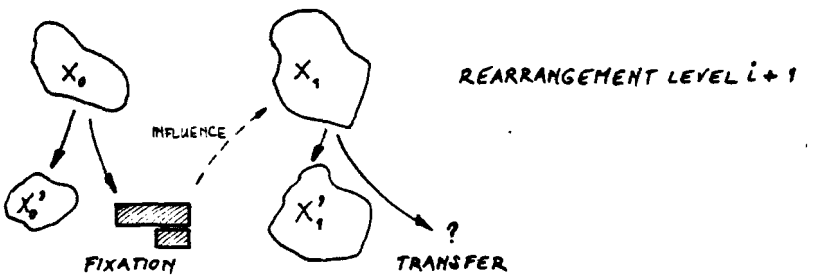
modules, then these must be transferred to other regions of the old level where they may be either assigned loosely to descendents (in case of lower-level placeability) or undergo another fixation attempt. Both *fixation* and *transfer* will be discussed further later on.

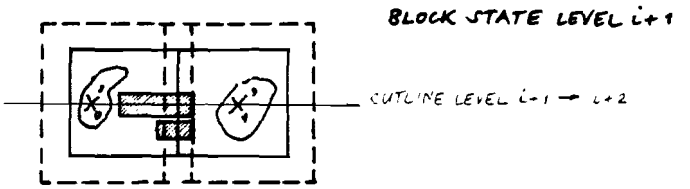
The result of the rearrangement phase must be a combination of fixed and loose modules satisfying the super cell restrictions discussed earlier in this chapter. The entire placement procedure (bipartitioning, contraction, creation and rearrangement) is sketched in figure 6.5.

figure 6.5




LOOK-AHEAD AT LEVEL i+2
(I.E. CREATION LEVEL i+2
AND CONTRACTION LEVEL i+1)
OUTLINE LEVEL i+1 → i+2


X_0
X_1
INFLUENCE
X_0'
X_1'
FIXATION
?
TRANSFER
REARRANGEMENT LEVEL i+1


BLOCK STATE LEVEL i+1
OUTLINE LEVEL i+1 → i+2

7. A routing strategy

7.1 Hierarchy evolution

For global routing alone, the transition of one level to a new level is concerned with constructing a global route of double resolution in terms of new half supercells (created by cutting old supercells along a number of parallel cutlines). This problem can be divided in a number of independent subproblems, each concerned with a single cutline:

Every cutline divides a row (or column) of cells (stretching from one side of the supercell array to the opposite side) into a *strip* of pairs of lower-level supercells: see figure 7.1. The length N of a row or strip increases with every level.

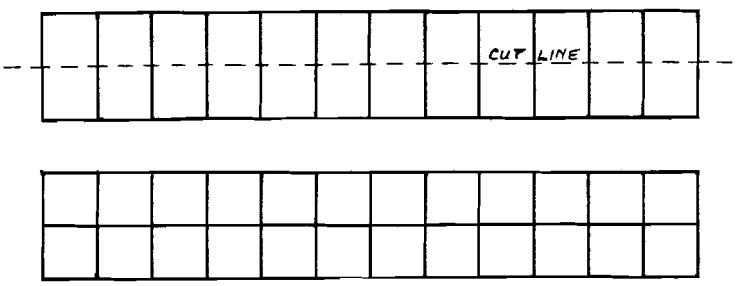


figure 7.1

All global route parts crossing external boundaries of the 1xN row can be preserved: in the 2xN strip they can be represented as pseudo-terminals (corresponding with connections to the outside). Together with the projection of actual low-level terminals onto the supercells in the 2xN strip, they form the vertices to be interconnected by a tree (or a forest of trees, in case of multi-cell terminals: see section 5.3) in the 2xN lattice graph associated with the formal global routing model. An example is given in fig. 7.2.

Routes in the 2xN strip may take *detours*, i.e. are not confined to the descendents of the supercells belonging to the route of the previous level. In principle, these detours may extend along the entire strip. However, computation time considerations (see later on) call for a maximum detour length.

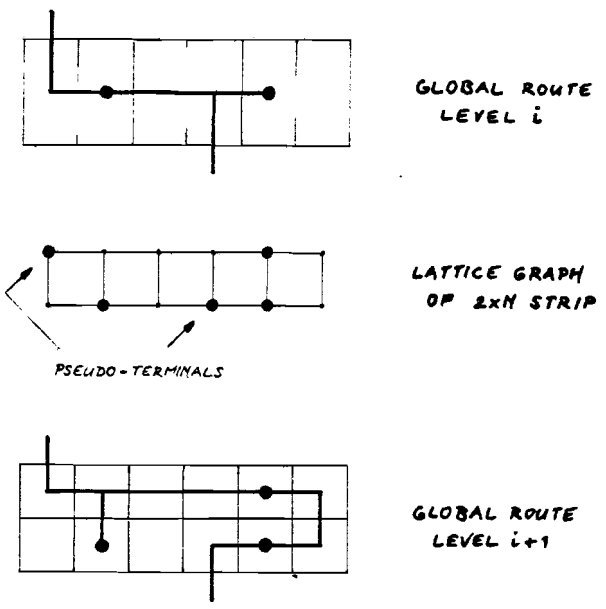


figure 7.2

In certain cases there are even stronger detouring restrictions. These are due to the following complication:

In general, a row of supercells on the old level may contain separate global route parts not connected within the row, but externally via the part of the global route in the rest of the supercell array. After bisectioning this row, the new route in the 2xN strip should keep the sets of terminals and pseudo-terminals corresponding with the above-mentioned separate global route parts unconnected, for otherwise a loop would be created.

So, the routing problem for one net in this case is actually a combination of several separate problems. Loops are avoided by defining for each problem a maximum detour in such a way that every pair of vertices in the 2xN lattice graph can be used by one part of the net only. See figure 7.3.

7.2 Mutations

Since higher-level capacities can only be heuristic estimates (see section 3.2 and 5.3), correct global routes on a certain level may nevertheless cause overflows on lower levels. In this case a partial rerouting must be done on this level.

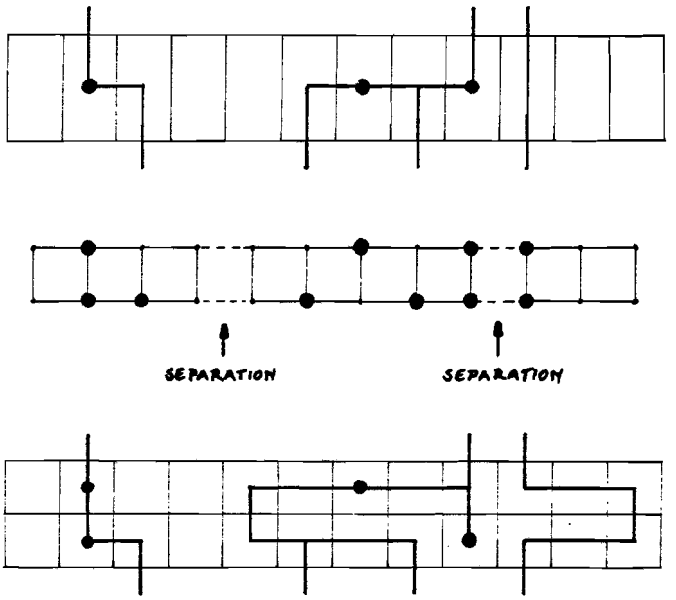


figure 7.3

Usually, overflow problems cannot be solved by concentrating on 2xN strips of supercells only. Therefore, rerouting must be concerned with larger portions of the global routing supercell array on a certain level. This can be done by means of a Lee-router [21], which is also used in the local routing phase [28].

The application of the Lee routing concept to the global supercell routing problem is straigthforward: edge costs are the same as are used for the 2xN routing algorithm (see chapter 11). It is argued in chapter 11 that, in the case of general modelling, edge costs are sufficient to cover all capacity restrictions.

For rerouting of overflowed nets, global route edges causing overflows are ripped up, and the subtrees now unconnected are reconnected by newly added routeparts. See figure 7.4.

Note: If only boundary overflows are rerouted and if this is done for single overflow edges individually, then the rerouting problem is essentially a two-terminal connection problem. The original Lee routing algorithm was confined to this case [21]. In all other cases where more than two route parts may remain unconnected (due to the rip-up of several overflow edges and/or vertices) the rerouting problem is concerned with more than two terminals to be connected.

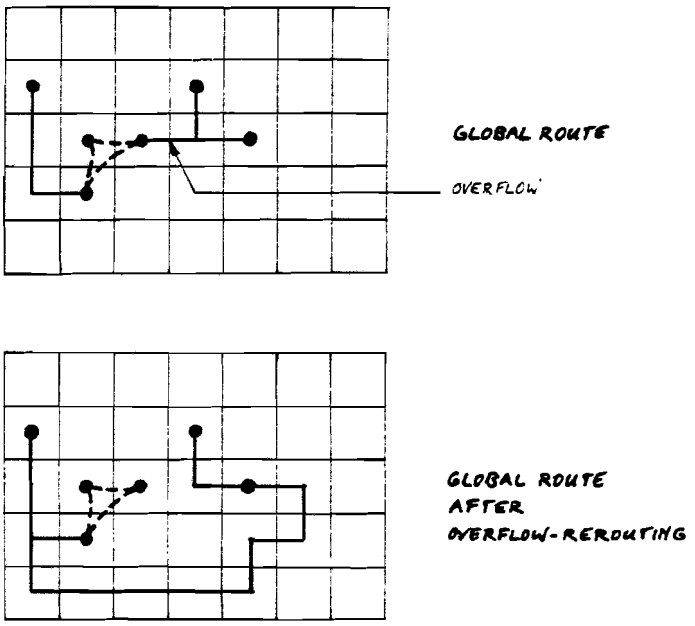


figure 7.4

However, the original Lee routing algorithm may be extended to this case, as is described in [28]. Note that multi-cell terminals in the global routing problem play the same role as equivalent positions for the local router [16].

8. A combined placement and global routing strategy

8.1 Hierarchy evolution

The key to a successful simultaneous evolution of placement and global routing is the objective function for the bipartitioning of loose modules. The idea is to take routing results of the previous level into account for the placement step that determines the assignment of modules to supercells on a new level. This is done by letting connections to the outside of a supercell row to be bisected into a 2xN strip act as extra "forces" on the modules to be partitioned (in addition to "clustering-forces" between the modules of a supercell set themselves):

The basic optimisation goal for the bipartition of the set of loose modules of a supercell is to minimise the contribution to the total wire length per net that can be predicted from the resulting placement distribution in itself. This means: avoiding a distribution that will increase the total wiring length anyhow, independent of the results of the subsequent global routing.

In the case of a bipartitioning of the entire circuit [26] or in the application to a stand-alone placement procedure [2], the objective function to be minimised is defined as the number of nets that is cut by the cutline between the resulting partition halfs. Or, stated otherwise, the number of nets for which connected modules occur in both partition halfs.

For the combination with global routing this idea is extended by taking into account the fact that connections from a supercell to the outside of the 2xN strip (following from the previous-level route) have the same effect on the net cut set size as contacts to loose modules in either half of the bipartition. Thus, external connections are included as pseudo-contacts for min-cut bipartitioning, analogously to their incorporation as pseudo-terminal cells for the global routing steps (see section 7.1).

There is one essential difference with the min-cut strategy as in Breuer [2]: because of the effect of external connections on individual supercells, every bisection of a supercell forms a separate min-cut partitioning problem (with its own cut set definition). This is in contrast to Breuer [2] and Corrigan [10], where one cut set is derived from the total distribution of modules in all supercells bisected by a cutline running from one side of the chip to the other.

Thus, min-cut bipartitioning in [2] and elsewhere, is applied to an entire row of cells (even though optimisation is done by iterating over single super cells) whereas simultaneous placement and global routing requires a min-cut criterion for dividing individual supercell module sets.

However, on high levels, where subsequent detour routing is less probable, it will pay to introduce a coupling between the partitioning of adjacent supercells (based on the route of each net that will result if no detours are necessary). For the level-1 situation depicted in figure 8.1a/b, this will immediately be clear. But in a similar situation on somewhat lower levels (fig. 8.1c/d), distribution *c* is still evidently to be preferred above partition *d* (with respect to the considered net).

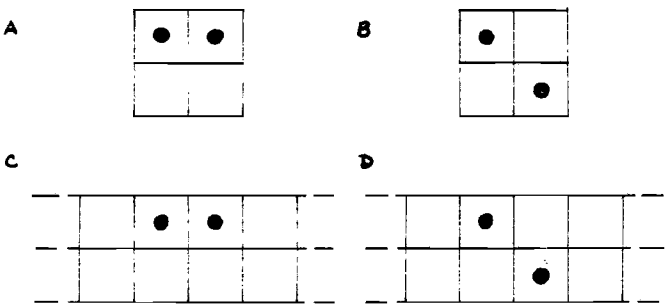


figure 8.1

The above-mentioned coupling can be established by incorporating certain contact distributions in adjacent cells that have already been partitioned as extra pseudo-contacts in the net distribution count during the bipartitioning of a new supercell.

Note that even adjacent supercells that have not been partitioned may influence the partitioning of their neighbours, because of the fact that the presency of some nets in one or both of the supercell's halfs can already be predicted from the terminal positions of fixed modules or the pseudo-terminal positions due to external connections. (There may be even loose modules that have been pre-assigned to a partition side because of non-placeability in the region associated with the the opposite side. This is discussed further in chapter 10).

8.2 The min-cut objective

In all, the min-cut objective is defined as follows for each supercell of any level:

Let S be the supercell (say of level i) to be partitioned. Let H_0 and H_1 be its two halfs, i.e. the descendent supercells on level i+1. The number of connections (or *incidence number*) of net n in half H_k (k=0,1), denoted by $\nu_k(n)$, is defined as the sum of the following terms:

- The number of loose modules with contacts to net n that have been assigned to H_k.
- The number of terminals in H_k stemming from fixed modules.
- +1, if H_k has an external connection (i.e. to the outside of the 2xN strip), 0 else
- Depending on the hierarchical level i, a coupling with adjacent supercells may be taken into account as an extra term +1 (for each neighbouring supercell S') if all of the following conditions are satisfied:
  - On level i, net n was routed from S to S'.
  - For the corresponding halfs H_0' and H_1' of supercell S' it holds that:

$$\nu_k'(n) > 0 \quad \text{and} \quad \nu_{1-k}'(n) = 0$$

Net n is said to be cut as a result of partitioning (i.e. to belong to the *cut set* of the partition (H_0, H_1) of supercell S) if and only if

$$\nu_0'(n) > 0 \quad \text{and} \quad \nu_1'(n) > 0$$

Of the four terms contributing to the number of connections of each half, the last two represent the influence of the global routing on level i on the placement result in terms of loose module assignments on level i+1.

The use of this influence is illustrated in figure 8.2 for two modules connected by a 2-terminal net n. In all conventional placement algorithms, performing placement as a separate phase, net n will contribute to a force of attraction between both modules.

However, the simultaneous placement and routing strategy

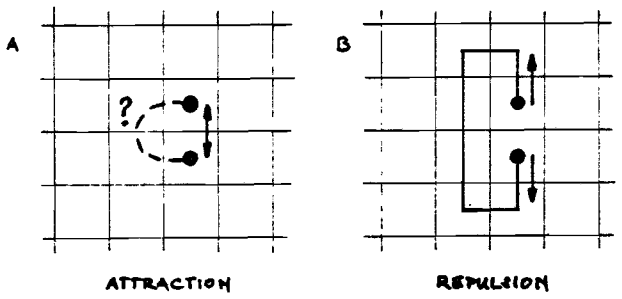


figure 8.2

treats n not just like an abstract net. This net may be wired on previous levels of hierarchy and it could take a certain detour because of the limited global routing capacities. It may look like figure 8.2B. In that case this net will contribute to a force of repulsion on both modules.

8.3 Mutations

So far, the advantage of the influence of the global routing on the placement evolution has been pointed out. This influence is particularly effective on relatively high hierarchy levels, as was already argued. Unfortunately, the fact that low-level placement mutations may be necessary (see chapter 6) makes that there is a disadvantageous reversed influence: low-level placement mutations affect the validity and optimality of global routes on these lower levels.

The extent of this reversed influence can be illustrated by way of the following example:

Let n be a 3-terminal net. Let a, b and c be the three modules connected to net n. Suppose that the placement step on level i has assigned a, b and c as loose modules to the i-th level supercells corresponding with the terminal cells in figure 8.3A. Suppose also that the subsequent global routing step (i.e. for level i) has routed net n as shown in figure 8.3A.

Now, consider the next placement step that has to produce an assignment of loose modules to supercells of level i+1 in such a way that the conditions given in chapter 6 are satisfied. As was described in chapter 6, this may cause mutations of the placement that do not fall within the hierarchy framework: some modules may have to be fixed, others may be transferred to other (i+1)-th level supercells

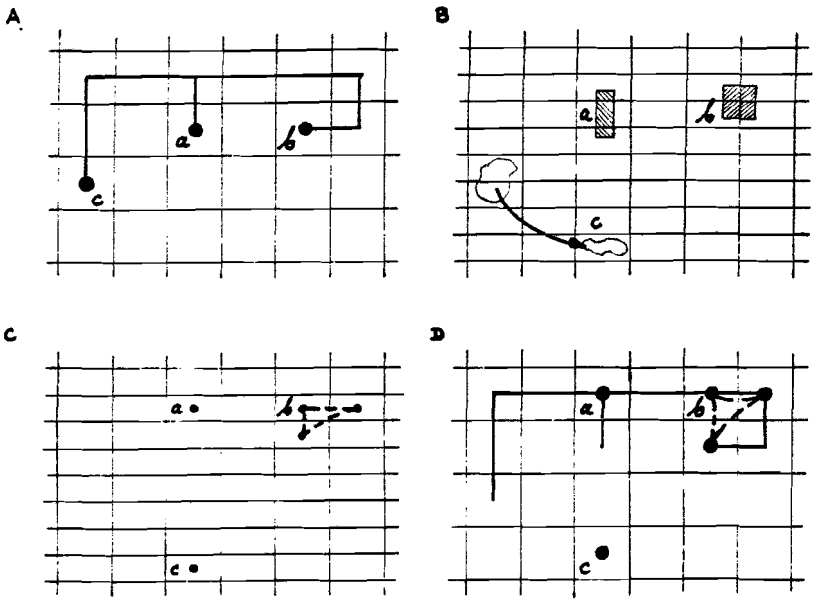


figure 8.3

than the descendents of the supercell they were previously assigned to (on level i). In chapter 6 this *rearrangement*, and its definition as an extra step in the transformation of the placement result of previous levels to a new level, was discussed.

Suppose that the (i+1)-th level rearrangement step includes mutations for the modules with contacts of net n, namely (figure 8.3B):

- Module a and b have been fixed.

- Module c has been transferred.

As was discussed in chapter 6, fixations may yield locations for modules that are intersected by cutlines of previous levels. For the fixations of module a and b as drawn in fig. 8.3B this is the case. This has serious consequences for the position of global terminals of net n in the supercell array of level i+1: the supercells on this new level that are terminal cells for net n may include other cells than descendents of the terminal cells of level i.

This is the case for module a after fixation: the actual terminal position of this module's contact to net n is on the outside of the supercell to which module a was assigned on level i (as loose module), i.e. on the outside of the

level-i terminal cell corresponding with the same contact.

Moreover, fixed modules may introduce multi-cell terminals, whereas on higher levels only single-cell terminals were represented. For instance, module b in figure 8.3 yields a multi-cell terminal on level i+1: it also comprises cells falling on the outside of the previous-level single terminal cell.

With transfers, the relationship between terminals on level i and level i+1 has disappeared completely: see module c in the example of net n.

In chapter 6 the hierarchy evolution with regard to global routing was explained: the fundament of this evolution was the fact that terminal positions on a lower level are confined to the descendents of supercells on a higher level. In order to indicate the disastrous effect of placement mutations, their effect can be translated in terms of terminal mutations with respect to the global route of level i: if the new terminal positions (in terms of level-(i+1) cells) are regarded in terms of level-i cells, then a comparison with the original level-i route yields (figure 8.3D):

- The terminal of module a has disappeared and reappeared in an adjacent cell: the route part formerly connecting this terminal makes no sense anymore.

- The single-cell terminal representing module b's contact in a loose state has "spread" to become a set of three fixedly interconnected terminal cells. As a consequence, the old global route now constitutes a loop!

- The terminal corresponding with module c has popped up at an entirely different cell position. It is no longer connected to the other terminals. Also, it has left behind a large part of the original route that serves no purpose anymore.

From this example it can be concluded that the i-th level route cannot always be used directly for the construction of a global route on level i+1: placement mutations make that the previous-level terminal distribution may no longer apply to the situation on the next level of hierarchy.

Therefore, the previous route may have to be altered before 2xN routing can be done in order to let a new-level route evolve. Clearly, this alteration is concerned with the entire supercell array of level i and not confined to the

immediate environment of the old route for each net.

So, an extra step involving the checking whether placement mutations have affected the validity of the global routing results on the previous level for each net and the alteration of those routes that have indeed been affected, is necessary. This step will be called *reconnection* step, and is not to be confused with *rerouting* which is discussed in chapter 11.

8.4 Reconnection

Reconnections of different nets are of course interdependent. For every net, reconnection may include:

- Removing "dangling ends": this necessitates a rip-up after detection of these dangling ends by tree-search labeling techniques, which is in fact the same function as must be performed after wavefront expansion in the Lee routing algorithm (see [21], [28]).

- Removing loops: this may be done straightforwardly after detection by way of a simple labeling procedure combined with depth-first search of the complete route. However, a more refined solution is the application of a Lee-routing step in the landscape formed by the route of the net: wavefront expansion is done along the edges of the graph formed by the route of the net with loops. The enhanced multi-terminal version as in [28] can be used very well for this problem.

- Routing connections to unconnected terminals (or, in general, unconnected subtrees). Again, this involves a straightforward use of a multi-terminal Lee-router as described in [28].

Strictly speaking, the routes of unaffected nets might be left unchanged. However, both fixations and transfers can influence the value of the original routing result in yet another way: they may change the capacity landscape quite drastically:

Loose modules can only be taken into account by taking some estimate of capacity reduction as guideline for determining supercell capacities (cf. average number of blocked tracks with Burstein [4]). However, fixed modules make the final capacity situation (which, as was argued in chapter 3, can only be determined by local inspection). Unlike with uniform superposition gate arrays fitting in with the Burstein model, for many gate arrays these estimates cannot be adequate in catching the large differences between the

capacity influence of a module in different placements (with respect to stamp choices, locations and environment interdependence): see e.g. the gate array of example 4 in section 2.3.

So, even if the new level's capacity situation with more details and unexpected details (due to fixations and transfers) is translated back to the smaller resolution of the previous level supercells, then this may yield such a different picture that overflow routing may be combined with reconnection routing, giving a reconnection step involving possibly all nets.

8.5 The impossibility of a hierarchy retreat

Given the fact that low-level placement mutations may destroy higher-level routing results to such an extent as described above, one might ask: shouldn't one do global routing all over again, starting at the top level of hierarchy and using the stand-alone global routing strategy described in chapter 7, taking the level-i placement result as the basis of the pseudo-bottom-level global routing problem.

That the answer must be 'no', can be illustrated by considering the same sort of situation as was used earlier in this chapter to explain the advantage of a simultaneous placement and global routing strategy:

Consider two 2-terminal nets n_1, n_2 connecting (different) pairs of modules assigned to the same supercells. The only difference between these nets is that their global routes on level i are different: net n_1 is routed as a straight connection, whereas n_2 takes a detour (figure 8.4A). Consequently, the forces on the modules of each net in the same supercells have opposite directions during bipartitioning. Thus, the placement on level i+1 tends to yield different module assigments for net n_1 and n_2 as in fig. 8.4B.

Now, suppose that the depicted placement on level i+1 is taken as the bottom-level global routing problem definition for a completely new routing hierarchy that starts again on the top level. Then, the terminal distribution for the global routing step on level i is the same for both nets: in fact, there is nothing except the name of the nets to distinguish the routing situation for n_1 and n_2.

Therefore, this routing step might produce a solution that is the same as in figure 8.4B, but with the route of n_1 and n_2 interchanged (figure 8.4C). But now the placement on

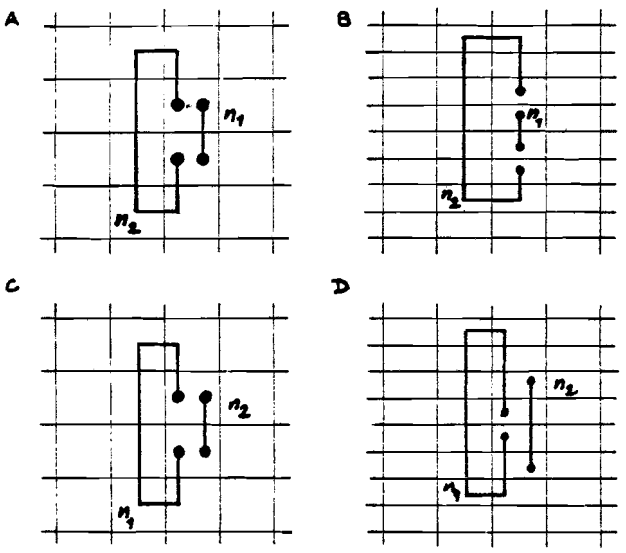


figure 8.4

level i+1 is bad instead of good (fig. 8.4D): the wire length profit originally obtained by using the first route on level i, has been undone and even turned into a deterioration as compared with results obtained for stand-alone placement techniques!

8.6 The total strategy

The conclusion must be that there is no hope for a simultaneous placement and global routing strategy if gate arrays have such complicated placement characteristics that many transfers and fixations with large consequences are necessary and/or if capacity consequences of lower-level placement restrictions are too inpredictable to be captured on higher levels with a largely undetermined placement in terms of only supercell assignments: in this case one can go neither forwards nor backwards in the hierarchy without throwing away any optimality and possibly all chances on final routability.

Therefore, the techniques introduced in this chapter and chapter 10 for coping with the consequences of the general situation in a flexible gate array layout system, should be seen as extensions and adaptations enabling to use a simultaneous placement and routing methodology even if a limited number of deviations from the original hierarchy (preferably on the lowest level only) cannot be avoided. The hope is that the average gate array design will have sufficiently room for manoeuvre to meet this placeability

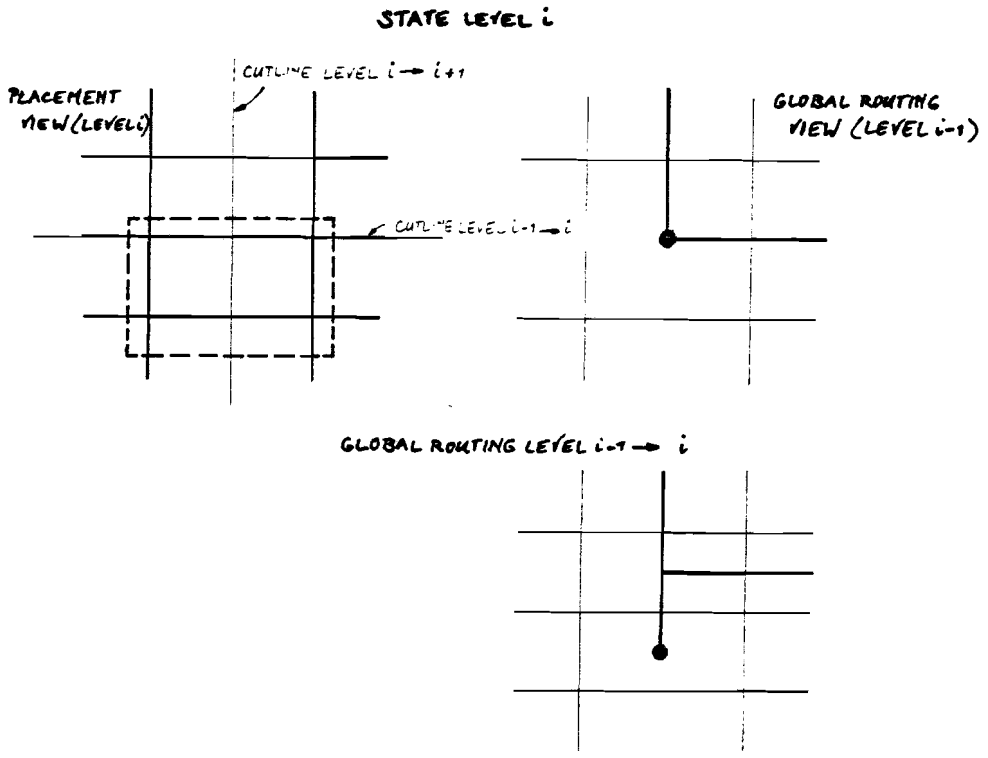
expectation: after all, a gate array is not a jig saw puzzle!

Given that hierarchy deviations cannot be avoided altogether, the damage they cause should be kept to a minimum. For this reason some reordering of steps is possible, based on the following:

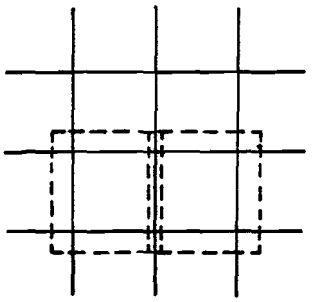
One should realise that the global routing results of level i is not needed until during bipartitioning on level i+1. However, this bipartitioning is preceded by a rearrangement step (which, as was discussed in chapter 6, belongs partly to level i but also partly to level i+1).

It follows that global routing for level i should preferably be done immediately after this rearrangement step instead of before it: in this way only the (i-1)-th level route has to be reconnected, and not the i-th level route. The total strategy obtained in this way is summarised in figure 8.5.

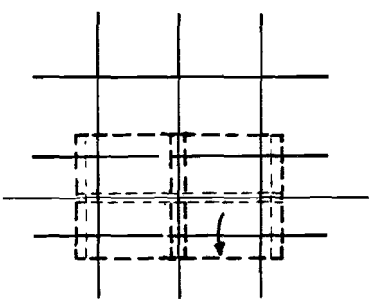
figure 8.5



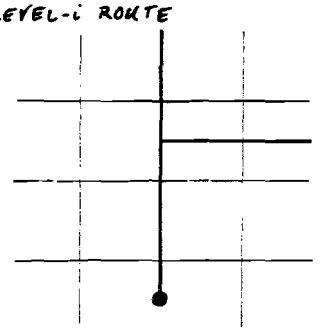
BIPARTITIONING LEVEL $i \rightarrow i+1$,
CREATION LEVEL $i+1$ AND CONTRACTION LEVEL i



LOOK-AHEAD AT LEVEL $i+2$
AND REARRANGEMENT LEVEL $i+1$

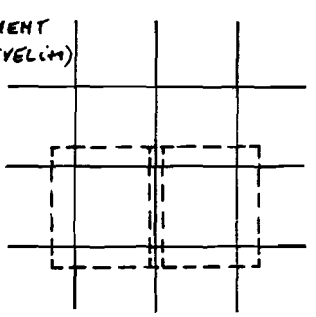


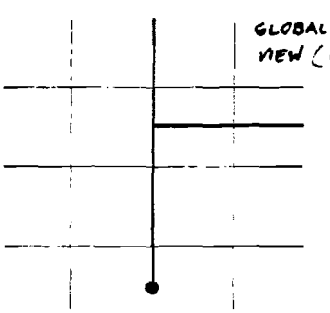
RECONNECTION OF LEVEL-i ROUTE



STATE LEVEL $i+1$

PLACEMENT
VIEW (LEVEL $i+1$)





GLOBAL ROUTING
VIEW (LEVEL i)

9. Min-cut bipartitioning

9.1 Problem definition

The bipartitioning problem (for individual fields per supercell) was introduced in the previous chapter:

Let a set of modules X be given. Each module $x \in X$ has an average area A(x). Also, two supercells H_0 and H_1 are given with available area (i.e. area not yet occupied by fixed modules of the field) A_0 and A_1 respectively.

The min-cut partitioning problem is to determine two subsets X_0, X_1 such that the size of the cut set defined in the previous chapter is as small as possible. X_0 and X_1 must satisfy:

$$X_0 \cup X_1 = X, \quad X_0 \cap X_1 = \emptyset$$

$$\Sigma_{x \in X_i} A(x) \leq A_i, \quad i = 0 \text{ and } i = 1$$

The latter condition is the *balance condition* for the partition (X_0, X_1).

Heuristics for the min-cut partition problem may concentrate on two algorithmic stages of the total solution:

- *constructive* algorithms for constructing X_0 and X_1 directly from X.
- *improvement* algorithms that try to improve an initial partition (X_0, X_1), yielding a better solution (X_0', X_1').

Although improvement might start from a randomly constructed initial partition, empiric results have proved that a combination of improvement and a more sophisticated constructive algorithm is worthwhile ([3], [19]). Here, both stages of the proposed heuristic are discussed in reverse order, starting with improvement. The reason for this is that the basic idea behind the improvement algorithms, which can be found in the recent literature, may be rather naturally extended to yield a new constructive algorithm.

9.2 Partition improvement

Partition improvement is done by performing a number of successive small alterations of the initial partition. This idea is common to a large class of heuristics for a wide scale of combinatorial optimisation problems. It can be

formulated in problem-independent terms as follows [25]:

Let P be an instance of a combinatorial optimisation problem. Basically, solving P comes down to choosing out of the set F of all feasible solutions (i.e. solutions satisfying all boundary conditions) the one that is best according to some cost function c(f), defined for all feasible solutions $f \in F$.

Now, let every feasible solution $f \in F$ correspond one-to-one with a *state* s(f) belonging to the *state space* of some abstract machine. The state transitions of this machine, i.e. (abstract) *moves* in the state space, correspond with transformations of one feasible solution f into another feasible solution f':

$$s = s(f) \rightarrow s' = s(f')$$

With regard to the total algorithmic complexity these transformations should only make "small" changes in the partition state: in general only a limited number of new states can be reached from an old state s:

The subset $E(s) \subset F$ of feasible solutions that can be reached from state s by a single move is called the *environment* of s for the class of moves concerned.

The basic idea for an improvement heuristic is to obtain a final near-optimal solution by starting from an initial state and performing moves with the aim of reaching a final state with costs as low as possible. The sequence of moves can be either non-deterministic or deterministic. In the latter case, moves $s \rightarrow s'$ are usually chosen by searching the best new state $s' \in E(s)$ in the environment of the old state s: this is a so-called *local search* heuristic. A non-deterministic approach of min-cut partitioning can be based on simulated annealing: Kirkpatrick [18]. However, for min-cut partitioning a successful deterministic heuristic is known which is much faster and yet has proved to yield good solutions.

For the min-cut partition problem the state space is obvious: feasible solutions are all partitions (X_0, X_1) satisfying the conditions mentioned earlier.

The basic idea for the heuristic presented here was developed by Kernighan and Lin [17] for a slightly different problem, viz. the bipartitioning of more or less uniform grahps that can be obtained in a network partitioning context by representing modules as vertices and defining an edge between every pair of modules having contacts to a

common net.

Schweikert and Kernighan [26] have shown that the original graph-cut model does not properly reflect the objective for network partitioning if nets can have more than two terminals. They have proposed a net-cut oriented usage of the same heuristic technique.

However, their algorithm is based on partitioning problems for modules of equal proportions. As basic move (in the sense of partition state transformation) they used the exchange of two modules from different sides of the cutline.

In the general gate array context, modules may have considerable differences of proportion (average area). Therefore, pairwise exchange is not meaningful: here a move will consist in the transfer of a single module from one partition half to the opposite half. Based on this type of moves (which were already investigated in Shiraishi-Hirose [27]) an extremely fast algorithm has been developed by Fiduccia-Mattheyses [13]. In the next sections an enhanced version of their algorithm will be described.

9.3 Pass-wise improvement

The simplest algorithm based on a local search heuristic would be a pure "hill-climbing" procedure that selects only moves yielding improvement (and, of course, satisfying the balance criteria). Such an algorithm would stop at a locally optimal partition, i.e. a partition for which no move exists that would result in a smaller cut set size.

However, such a procedure is likely to get stuck with a bad locally optimal solution. (Burstein [??] has shown this empirically for the case of a move definition consisting in pairwise interchange of equally sized modules: it is obvious that results will be even worse for the move definition that is used here.)

The conclusion must be that if better solutions are to be obtained in the end, then moving away from local optima must be possible, i.e. moves that do not improve the partition immediately (but, on the contrary, yield a deterioration at first) must be tolerated (on a tentative basis). However, just picking the best move (yielding the highest decrease of the cut set size or, in case of a locally optimal state, causing the smallest increase) might result in a non-terminating "thrashing" of the module-moving process (in abstract terms corresponding with an infinitely circling state space walk).

Therefore a *locking strategy* is applied that subdivides the entire procedure in a number of *improvement passes*:

Initially, all modules may be transferred from one partition side to the opposite side, i.e. all modules are *free* to move. However, once a module has been moved to the other side, it is *locked*. The lock status prevents this module from returning to its old partition side in the same pass. Thus, at every intermediate state, the set of state space moves is confined to those moves corresponding with transferring free modules. So, the environment of states is variable.

Ultimately all modules will be locked. This marks the end of the improvement pass. The basic idea is now to retrace the sequence of moves until the best intermediate state is restored: this state is returned as the result of the improvement pass. After this pass, all modules are freed and a new improvement pass is started. This procedure is repeated until a pass yields no further improvement.

The change of the cut set size during a single pass is illustrated in figure 9.1A. The change of the net cut set size in terms of passes is illustrated in figure 9.1B.

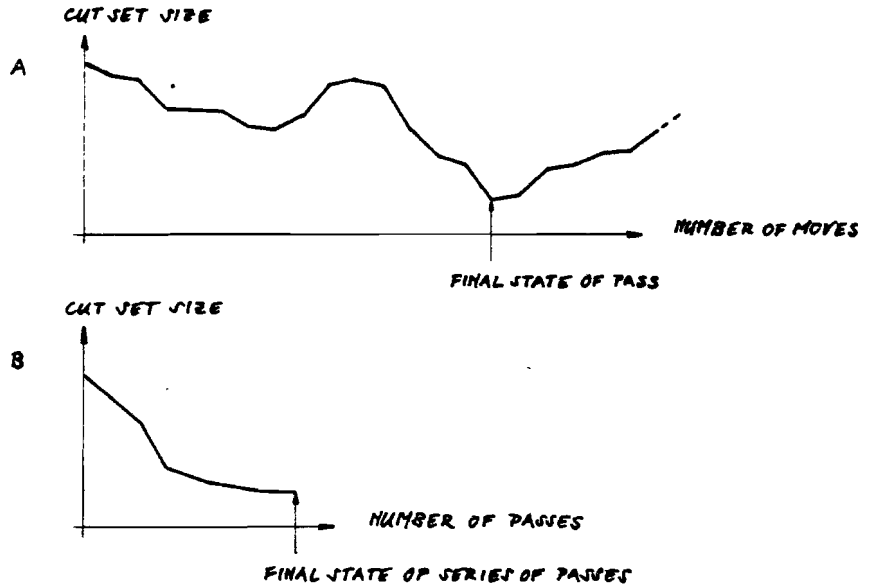


figure 9.1

9.4 The pass algorithm

During a pass, the selection of new moves must be restricted to transfers of free modules. The choice of the module to be moved must be based on both its effect on the size of the current cut set size and its effect on the balance condition. This is done as follows:

At every intermediate partition state s one can define a *gain* $\gamma(x)$ for every one of the remaining free modules (i.e. for every abstract move alternative). This gain equals the decrease of the cut set size that would result from moving module x to the opposite partition side. An example of module gain values is given in figure 9.2.

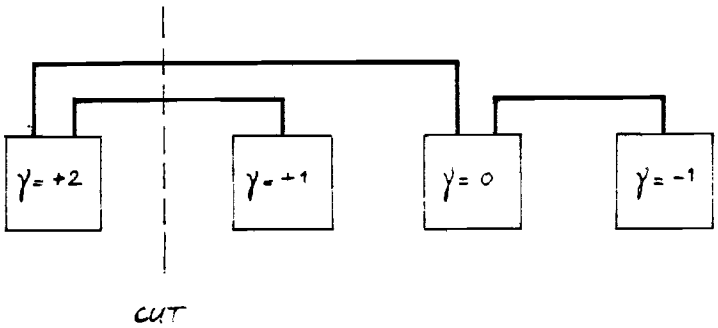


figure 9.2

In general, the contribution of each net to the gain of modules connected to this net can be -1, 0 or +1. This contribution depends on the distribution of modules over the partition sides: changing the partition state changes this distribution, so after every move the gains of the remaining free modules may have to be updated. Fortunately, this can be done in a very efficient way by choosing a special datastructure for representing a sorted list of module gains and replacing each gain recomputation by an appropriate sequence of simple gain increments and decrements which can be done in constant time.

The main part of this special datastructure is a pair of bucket arrays (one for every partition side) used for maintaining a list of free modules per side, sorted according to their gains. Using bucket sorting is possible because of the restricted set of values which module gains may take on:

It is easy to see that the gain $\gamma(x)$ of some module $x \in X$ must satisfy:

$$-p(x) \leq \gamma(x) \leq p(x)$$

where p(x) is the number of different nets that module x is connected to.

Note: p(x) need not be the same as the number of contacts c(m(x)) of the module's macro function m(x). This is because several macro contacts may be connected to the same net. Also, the possibility exists that, apart from the implicit power connections in the gate array image, some additional macro contacts are to be routed to the power lines. In this case these should not count for the min-cut placement criterion (nor should they appear in the global routing problem, by the way). This yields the following condition for p(x) (which will be called the number of "pins" henceforth):

$$0 \leq p(x) \leq c(m(x))$$

For gate arrays, the number of contacts per macro is small. Let p be the maximum number of "pins" per module:

$$p = \max_i [\ p(x_i)\ ]$$

Then an array of 2p+1 buckets is enough for keeping a sorted list of all free modules per partition side. For each bucket array a *maximum-gain* index is maintained which is used to keep track of the bucket having a module of highest gain. Also, the netlist must be represented in two (dual) forms: modules-per-net and nets-per-module.

In principle, free modules from either partition side can be chosen as moves for obtaining a new state: a highest-gain module from the partition side with the highest-gain non-empty bucket is chosen, unless transfer of this module would cause imbalance.

Note: In general the available area of both partition sides need not be of the same order. This is especially the case for the bipartitioning of non-lumped fields, e.g. I/O fields forming a ring of cells. In this case, the number of moves from the large side to the small side is restricted: moving more modules from the large partition side than the small side may contain is impossible. Therefore in this case, a pass terminates before all modules are locked.

Once a module has been chosen, its transfer to the opposite partition side (and locking) involves several changes:

- The balance situation must be updated.

- The distribution of nets connected to the moved module must be updated.

- The gains of other modules connected to these nets may have to be changed as a consequence. This is done net-wise by incrementing/decrementing gains of connected free modules only if necessary because of critical changes of the net distribution.

The complexity of an entire improvement pass can be determined as follows:

- Choosing a new module for transfer to the opposite partition side can be done in constant time if the max-gain indices indicate the highest-gain non-empty buckets on either side. Thus the complexity contribution of the selection steps is of the order of the number of moves, which is (at most) equal to the number of modules to be partitioned: $O(|X|)$.

- Balance situation updates can be done in constant time per move. So, the total complexity contribution per pass is: $O(|X|)$.

- Net distribution updates per move must be done for every net connected to the moved module. If every module is moved once (yielding the maximum number of moves of a pass), then all contacts, or rather all "pins" (in the sense used earlier), of all modules give rise to exactly one net distribution update.

  Let q be the total number of pins:

  $$q = \Sigma_i \, p(x_i)$$

  Then the complexity of net distribution updating for one pass amounts to: $O(q)$.

- The initial gain computation (at the beginning of a pass) comprises inspection of the net distribution for every net per module: $O(p(x))$ per module x. Summed over all modules this yields: $O(q)$.

- The complexity of gain updates follows from the following argument:

  Let a net be called *critical* if there exists a free module on it which if moved would change the net's cut state. Let $\rho_i(n)$ be the number of free modules in partition half H_i that is connected to net n. Then it is easy to see that net n is critical if one of the

following conditions holds (for i = 0 and/or i = 1):

$$\nu_i(n) = \rho_i(n) = 1 \quad \text{and} \quad \nu_{1-i}(n) > 0$$

$$\nu_i(n) = 0 \quad \text{and} \quad \rho_i(n) > 0$$

Note that the difference $\nu_i(n) - \rho_i(n)$ equals the number of locked modules connected to net n plus the contribution of external connections and fixed terminals to the net distribution count $\nu_i(n)$.

It is now clear that the gain of a module, previously defined in terms of its effect on the cut set, depends only on its critical nets. This means that if the net is not critical, its cut state cannot be affected by a move. What is more important, a net which is not critical either before or after a move cannot possibly influence the gains of any of its modules.

From the foregoing it follows that a critical distribution must have:

$$\nu_i(n) = \rho_i(n) \leq 1$$

Every time a module of net n is moved, $\nu_i(n)$'s will change: $\nu(n)$ decreases on the side of origin and increases on the opposite side. However, $\rho_i(n)$'s can only decrease (since the number of free modules can only decrease). Now, $\rho_i(n)$ can only decrease two times so that the above-mentioned distribution condition holds. So, at most two modules connected to net n can be moved while net n is or becomes critical. Ergo, at most two times the change of distribution for net n may effect the gains of its modules.

Updating the gain of a single module can be done in constant time. Since each module's gain may have to be updated for changes in the distribution of each one of its nets, but no more than two times per net (as argued above), the total time spent on updating the gain of a module x is: $O(p(x))$. Summed over all modules, this yields the following updating complexity per pass: $O(q)$.

Taking all complexity contributions together, the following over-all pass complexity results:

$$O(|X|) + O(|X|) + O(q) + O(q) = O(q)$$

q can be regarded as the "size" of the total network to be

partitioned. So, the discussed pass algorithm is essentially *linear*.

Note: the complexity proof given in Fiduccia-Mattheyses [13] is different and more elaborate.

As to the total complexity for a series of passes, both Fiduccia-Mattheyses [13] and Krishnamurthy [19] have reported form experiments that the total algorithm takes typically three of four passes to converge. Therefore, they hypothesise that the total algorithm has linear complexity in most practical situations.

9.5 Further improvement by recursion

The introduction of the pass algorithm in the beginning of this chapter has emphasised the role of single-module moves. However, there is another way to look at a pass:

The result of a pass is a new partition that differs from the initial partition in that subsets of the initial sets of modules for each partition side have been exchanged. One can see this as another sort of abstract move in a partition state space. This move corresponds with another kind of environment definition in the sense used in the beginning of this chapter, namely the environment of a state s that is formed by all states that can be obtained by interchanging subsets (of variable size) of the module sets of each partition side.

The pass algorithm can be regarded as a heuristic for determining the best exchange of subsets. The size of these subsets is not specified in advance, but rather chosen to make the improvement as large as possible. The pass algorithm is a heuristic sequential approximation for the best subset exchange, necessary because a full search of the extended environment mentioned above is infeasible. This is the approach taken by Kernighan-Lin [17].

In terms of pass moves, the over-all strategy is a hill-climbing "variable-depth" local search heuristic [25].

Kernighan and Lin [17] observed from experiments with small instances of the graph partitioning problem that the final solution of a sequence of passes might stay far from an optimal partition if attaining the optimal partition would involve the scrambling of a large number of modules. In the case of the graph partitioning problem, there is a symmetry of partition sides. Therefore, the subset exchanges that are least likely to be found are those concerning the interchange of about half of the module sets of each

partition side: this is the subset exchange that results in the greatest partition change in a symmetrical situation.

Therefore they have suggested a technique for creating a sort of *orthogonal* starting point that differs maximally from the result of a series of passes, with the expectation that starting a new series of passes from this "orthogonalised" partition will eventually come nearer to an optimum (because optimal exchanges will be concerned with smaller subsets):

9.6 Orthogonalisation

Let (X_0, X_1) be the partition resulting from a series of passes. Then X_0 is divided into two parts, say X_{00} and X_{01}. Similarly, X_1 is divided into X_{10} and X_{11}. As orthogonal starting point one can take either one of the following partitions:

$$(X_{00} \cap X_{10},\ X_{01} \cap X_{11})$$

$$(X_{00} \cap X_{11},\ X_{01} \cap X_{10})$$

From this new initial partition a new series of passes is started. If a better partition is found at the end of this series, then this partition is divided up into two parts and a further improvement of the orthogonalised partition is tried again. If not, the situation before the division of X_0 and X_1 is restored.

How are X_0 and X_1 to be subdivided? Clearly, they should preferably be divided in such a way that the two parts have very few nets in common. So, dividing X_0 and X_1 are also min-cut bipartitioning problems. In principle, one could use a recursive approach down to the level where partition halfs consist of single modules. However, this would be very inefficient. Krishnamurthy [19] reports surprisingly good results for a 2-level approach: partitioning X_0 c.q. X_1 is done without further orthogonalisation steps.

However, the min-cut partition problem considered here is slightly different because of its relation to global routing: the symmetry of partition sides mentioned earlier does not hold here due to the contributions of external connections and fixed terminals to the net distribution. Besides, the sizes of X_0 and X_1 may not be the same in general, as was argued before.

The extra contributions to the net distribution have to be taken into account during the partitioning of X_0 c.q. X_1. Therefore, there is only one possible starting point after

orthogonalisation, viz.:

$$(X_{00} \cap X_{10},\ X_{01} \cap X_{11})$$

Unequal sizes of X_0 and X_1 call for a special balance criterion for each partition (X_0 c.q. X_1):

Let A_{ij} be the maximum available area on side j during the partitioning of X_i. Then A_{ij} is given by:

$$A_{ij} = [\ A_j \Sigma_{x \in X_i} A(x)\ ] / [\ \Sigma_{x \in X} A(x)\ ]$$

9.7 Improvement parametrisation

Krishnamurthy [19] has proposed a generalisation of the linear heuristic described in this chapter. The possibility of a further amelioration of the Fiduccia-Mattheyses improvement procedure is suggested by the fact that the limited heuristic of choosing one module, which when moved to the other side will yield maximal improvement in the cut set size, still leaves a number of random choices made within the algorithm. Thus, the quality of the partitions produced seems to be influenced by for instance the order of the modules investigated.

In figure 9.3 an example is given for two modules A and B which are candidates for being moved to the other side. Either module move permits the deletion of one net (n_1 and n_2 respectively) from the cut set, whereas the status of net n_3 and n_4 remains unchanged. Thus, both A and B have gain +1, and either one could be selected for the move, arbitrarily. However, moving B is probably preferable because it would then permit the deletion of yet another net (n_4) through a subsequent move.

Krishnamurthy [19] suggests the following generalised technique for capturing this sort of "foresight" and eliminating the influence of random choices:

Instead of a single gain value, a gain vector

$$(\gamma_1, \gamma_2, \ldots, \gamma_k)$$

of length k is associated with every free module. The m-th component is called the gain value of degree m.

- $\gamma_1(x)$ is the direct cut set gain resulting from moving module x to the other partition side.

- $\gamma_2(x)$ reflects the change of the first-degree gain caused by moving module x, i.e. the possibility of cut

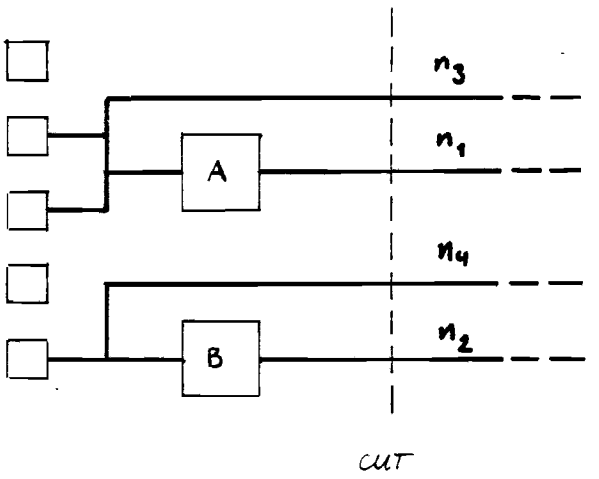


figure 9.3

set changes via a subsequent move.

- γ_k incorporates the contribution (of moving module x) to (k-1)-degree gains of other modules.

In general, the m-th degree gain indicates the effect of a move on the (m-1)-th gains of other modules ($1 < m \leq k$). The exact definition of $\gamma_m(x)$ is given below:

Consider a net n connected to module x. Suppose that module x is free on partition side H_i ($i = 0$ or $i = 1$). Then the contribution of net n to the m-th level gain $\gamma_m(x)$ is the sum of the following two terms:

- +1 if $\nu_i(n) = \rho_i(n) = m$ and $\nu_{1-i}(n) > 0$, 0 else. Thus, net n contributes positively to the m-th level gain if removing module x from H_i permits net n to contribute positively to the (m-1)-th level gain of some other free modules on side H_i connected to net n. The condition $\nu_i(n) = \rho_i(n)$ says that net n should have no locked connections (including external connections and fixed terminals) on side H_i. This is because locked connections will prevent deleting net n from the cut set, since the locked connections cannot be removed from H_i during the rest of a pass.

- -1 if $\nu_i(n) > 0$ and $\nu_{1-i}(n) = \rho_{1-i}(n) = m - 1$, 0 else. Thus, net n contributes negatively to the gain of degree m if locking module x on side H_{1-i} will make a positive gain contribution of level m-1 for some free modules on side H_{1-i} impossible. Of course, if net n is

already locked on side H_{1-i} (i.e. $\nu_{1-i}(n) \neq \rho_{1-i}(n)$), then this is not the case.

The total gain of degree m is the sum of the total contribution per net over all nets.

The reader can easily verify that the first-level gain $\gamma_1(x)$ is indeed the same as the single gain value $\gamma(x)$ that was used in the unparametrised version of the pass algorithm described in the beginning of this chapter.

The gain vectors of all free modules are ordered lexicographically, since the lower-degree gains are more important than the higher-degree gains. Again, a bucket sorting technique is used for maintaining the gains of free modules during a pass. This is possible because

$$-p(x) \leq \gamma_m(x) \leq p(x)$$

holds for all m, $1 \leq m \leq k$. For vectors of length k the total bucket array length is:

$$(2p + 1)^k$$

Using a similar argument as was used for the single-gain case earlier in this chapter, the complexity of the parametrised pass algorithm can be proved to be O(kp). The complete proof can be found in Krishnamurthy [19], together with experimental reuslts supporting the theoretical analysis. The experiments showed a considerable amelioration with regard to the quality of the final solution as well as with respect to the consistency (i.e. random choice influences have decreased).

How should k be chosen? Krishnamurthy [19] has proposed the following technique for choosing a suitable value for k which depends on the network to be partitioned. The choice is based on the following argument:

Assume that the gain vectors of all modules are uniformly distributed over the $(2p + 1)^k$ possible vector values. (This assumption will not be precisely true in reality.) Then there would be

$$|X| \; / \; (2p + 1)^k$$

modules per vector value. In order to yield a maximum discrimination of modules (no random choices) without excessive bucket storage requirements, the number of modules with the highest gain should be 1. This yields:

$$k = \log |X| \; / \; \log (2p + 1)$$

9.8 Constructive partitioning

Both Burstein [3] and Krishnamurthy [19] have experimentally shown the crucial role of a good starting partition. The initial partition should not be random, but it should be constructed by some clustering heuristic.

Burstein [3] suggests several constructive algorithms. However, here a new algorithm is proposed that uses a more sophisticated heuristic (similar to the parametrised improvement heuristic described in the previous section) and which all the same preserves a linear complexity by using the same sort of datastructures and incremental/decremental updates as Fiduccia and Mattheyses [13] did for the improvement algorithm described in section 9.4.

An essential difference is that in the construction phase every module is associated with two moves, each corresponding with a "transfer" from the initial unpartitioned set of modules X to a partition side set X_0 c.q. X_1. The constructive algorithm comes down to choosing between either move for each module. This is done sequentially in a non-predetermined order on the basis of the following *force* criterion:

A module x is defined to be subject to two different forces, each representing a heuristic "attraction" measure for the effect on the final cut set size of assigning module x to a partition side.

The unparametrised version of the algorithm uses a single force value $f_i(x)$ for a move corresponding with assigning module x to partition side set X_i ($i = 0$ or $i = 1$). $f_i(x)$ is the sum of the force contributions of each of the nets connected to module x. These contributions are defined as follows:

Net n has a positive contribution +1 to $f_i(x)$ if x is the last unassigned module connected to net n and if the distribution of net n satisfies:

$$\nu_i(n) > 0 \quad \text{and} \quad \nu_{1-i}(n) = 0$$

This situation is the analogon of the critical net status introduced for partition states during improvement. In all other situations net n has a zero contribution to $f_i(x)$.

Analogously to an improvement pass, the algorithm selects at any time the move with the highest force value, unless this

move would cause imbalance in which case a move to the opposite partition side is chosen. For the purpose of avoiding imbalance, the highest-force move must be available for both partition sides as a consequence, so separate bucket arrays are used for keeping a sorted list of moves for each of the two partition sides.

Performing a move of module x to X_i consists in updating the net distribution, locking module x on side H_i, removing both moves associated with module x from the bucket arrays and updating the forces of modules that are still free (i.e. unassigned).

9.9 Construction parametrisation

Analogously to the improvement case, force vectors

$$(f_{i,1}, f_{i,2}, \ldots, f_{i,k})$$

may be introduced in order to obtain a more sophisticated, parametrised constructive heuristic. The contribution of net n to the m-th level force value $f_{i,m}(x)$ is non-zero (i.e. +1) if and only if there are m unassigned modules connected to net n and:

$$\nu_i(n) > 0 \quad \text{and} \quad \nu_{1-i}(n) = 0$$

Note that $f_{i,1}(x)$ equals the single-valued force defined in the previous section.

For all m, $1 \leq m \leq k$, $f_{i,m}(x)$ satisfies:

$$0 \leq f_{i,m}(x) \leq p(x)$$

This gives for all force vector components $f_{i,m}$:

$$0 \leq f_{i,m} \leq p$$

Consequently, the total length of each bucket array should be:

$$(p + 1)^k$$

Analogously to the improvement pass algorithm, force vectors are ordered lexicographically.

The new constructive algorithm's complexity can be shown to be linear by a similar argument as was used for proving the improvement pass complexity: O(kp). Also, the same sort of heuristic calculation of a suitable value for k can be applied. Here, the result is:

10. Rearrangement

10.1 Problem definition

In chapter 8 a rearrangment step was introduced in order to cope with unavoidable low-level mutations deranging high-level hierarchy evolutions. The rearrangement problem can be stated as follows (for one field, since different fields are independent with regard to placement restrictions):

Let B be the set of blocks on level i (arranged in rows and columns). In chapter 6 an enlarged region was associated with every block: let R(b) be the notation for the region of block $b \in B$. Before rearrangement, R(b) is yet uncontracted (see chapter 6).

The bipartitioning step that immediately precedes the rearrangement step has assigned all loose modules to blocks. These assignments, together with the locations of previously fixed modules form the input of the rearrangement algorithm for each field. The output of the algorithm must be a similar (if possible, the same) mixture of loose module block assignments and fixed module placements that satisfies on this level and subsequent levels the generalised placement restrictions discussed in chapter 6, namely:

- Individual placeability in region R(b) of loose modules assigned to block b (for all blocks $b \in B$).

- No over-occupation of any block $b \in B$.

The rearrangement problem is to construct an output combination of loose assignments and fixed placements that satisfies these requirements on this level and will satisfy these requirements also on the next level (look-ahead) by changing the input situation as little as possible. "As little as possible" should be understood as: with minimum derangement of the validity of the previous-level routing result (see chapter 8).

10.2 Rearrangement possibilities

Changing the input situation can be done in two ways:

- Fixation of loose modules.

- Transfer of loose modules.

The latter option should only be done if really necessary, since transfers over large distances cause maximum derangement with respect to the global route that has been

determined on the basis of the input situation. The heuristic algorithm proposed here tries to keep mutation damages to global route validity to a minimum by first trying to fix loose modules.

However, the former option is a source of interdependence between blocks: fixations may result in placement locations intersecting several blocks. Thus, fixation of a loose module previously assigned to a block b may yield a placement partly occupying a neighbouring block b' in such a way that b' is now over-occupied.

For which loose modules should fixation be taken into consideration? Exactly those that cannot be meaningfully assigned as loose modules on subsequent levels. There may be two reasons for a module x to be no longer meaningfully assignable:

- The contraction of the block regions on the bipartitioning level makes module x non-placeable in the region R(b) of the block b it is assigned to.

- Subsequent region creation on the next level may yield two descendent regions neither of which can contain module x.

The criterion in the second case is stronger: non-placeability in a region implies non-placeability in descendent regions. Therefore, the selection of modules to be fixed can concentrate on non-placeability in newly-created next-level regions (i.e. be based on *look-ahead* to the next level).

Now, where may loose modules be fixed? On the one hand, one should try to let the fixed placement correspond as much as possible with the loose block assignment hitherto assumed: the fixation should cause minimum disturbance of the global routing picture, i.e. the fixed module location should fall as much as possible (preferably completely) within the block area corresponding with the supercell that represented the module's contacts as terminal cells in the last global routing step.

On the latest level of bipartitioning (this is one level lower than the last globally routed level: see chapter 8) loose modules have been assigned to blocks if and only if the modules were placeable in the corresponding regions. These regions had been created, but not yet been contracted (in fact, their contraction might be the very reason for their present fixation, as was discussed above). The present fixation attempt should therefore take only shape-position

pairs into account which correspond with placements inside these regions.

Unfortunately, the regions of various blocks will be overlapping in general. Thus fixation of modules in one block may take away part of the available area in other neighbouring blocks. As a result of this, modules in these other regions may no longer be fixable. This effect is to be avoided as much as possible. Therefore the following fixation scheme is proposed for reducing the transfer effects of region fixation interdependence:

10.3 Fixation pass

All blocks are ordered according to their degree of occupation. Starting with the highest-occupation block, the placeability of loose modules in the future half regions is checked. This check concerns the dimensions of the half region after creation (uncontracted). For each loose module x there are three possible outcomes:

- Module x is placeable in both half regions. In this case module x will partake in the next bipartitioning step.

- Module x is placeable in only one of the half regions. Now module x is directly assigned (*pre-assigned*) as loose module to the half region it can be placed in.

- Module x cannot be placed in either half region. This implies that module x must be fixed.

For all modules to be fixed, a branch-and-bound scheme is used to find a combined placement.

Suppose that all modules can be fixed together (i.e. in the uncontracted previous-level region). Then the above-discussed region fixation interdependence implies that, as a result of these fixations, the occupations of adjacent blocks may be changed (increased). In order to be able to update the ordering of blocks that are still to be treated for fixation, a bucket sorting is used: the entire range of possible occupation degrees is subdivided in a number of intervals, each of which corresponds with a bucket containing items called *zones* that represent blocks with an occupation degree in this interval. Since at most 8 adjacent blocks may be influenced, the total updating effort is bounded by a constant time bound per block. Every block zone for which loose modules have been checked on placeability in half regions is removed from the bucket array.

Now, consider the case in which a complete fixation fails, i.e. some modules to be fixed are left over as infixables. Then these modules are removed from the block and put in a transfer list for future assignment or fixation elsewhere in the block array. After this, the block is also removed from its bucket.

However, not only unfixable modules may have to be transferred: transfers may also be necessary because of the second (interdependent) generalised placement restriction discussed in chapter 6, viz. because of over-occupation (too many loose modules for too little available area). In such a case some loose modules have to be transferred as well.

So, over-occupation should be checked on (for both half regions, as well as their combination). This is done by an *occupation analysis* (consisting in checking the occupations concerned with every block, in an arbitrary order) that follows after the above-described fixation pass. (The fixation pass consists in checking the placeability of all loose modules and fixations, per block, in a constantly updated order of decreasing occupation.)

10.4 Transfer passes

If no modules have been listed for transfer (during the fixation pass or the subsequent occupation analysis), then the following step in the hierarchy evolution scheme can be performed (this is the global routing step: see chapter 8). If not, then one or more *transfer passes* will be necessary in order to find new block assignments or fixed placements for the modules to be transferred. A transfer pass consists in the following:

All blocks are represented in the same bucket structure as during the afore-mentioned fixation pass, and again they are sorted according to their degree of occupation. However, since the chances for transfer modules are best in those blocks that are least occupied, the bucket array is now searched in the reverse direction, starting with the non-empty bucket corresponding with the lowest degree of occupation.

For every block, the list of modules to be transferred is gone through. For modules of the transfer list the same placeability check is done as was described earlier for the fixation pass:

- If a transfer module is placeable in both half regions, then it can be assigned as loose module to the block and take part in the subsequent bipartitioning step.

- If the transfer module is placeable in exactly one of the half regions, it is directly assigned to the half region in question.

- If it cannot be placed in either half region, but can be placed in the whole region, then a fixation of the module may be attempted. Fixation is done in the same simultaneous branch-and-bound scheme as is used for the fixation pass.

Of course, the assignment in the first two options can only be performed if no over-occupation of half regions or whole region would result. Therefore an occupation analysis should be done immediately (in contrast to the occupation analysis after the fixation pass, which was done for all blocks together, as a separate pass).

If a module is not even placeable in the whole region associated with the block under examination, or if the fixation attempt is not successful, then the module remains on the transfer list (this is also the case if over-occupation prevents the module from being assigned loosely).

However, if a fixation attempt succeeds, it may influence the occupation degree in adjacent blocks. Since this may cause over-occupation, a check must be done immediately: if over-occupation occurs, then some loose modules of the adjacent block must be moved to the transfer list (to be transferred in addition to the modules already in the transfer list). Also, the changed occupation degree must be represented correctly in the bucket array by updating the adjacent block's bucket entry. Of course, this updating only concerns adjacent blocks that were not yet treated in the same pass (i.e. for which possibilities to accomodate transfer modules have not yet been examined): blocks that have been treated have been removed from the bucket array.

If at the end of the transfer pass the transfer list is not yet empty, a new transfer pass is started by filling the bucket array again and repeating the same procedure of examining blocks in order of increasing occupation degree. The only difference with the previous transfer pass is the size of the whole regions: the region associated with each block is increased temporarily (half regions stay unchanged). This is based on the fact that if a module is non-fixable in one block because of its large proportions, then it may be difficult to place the module in other blocks, for block regions on the same level will be of comparable sizes. So, after having been left over from the fixation pass and the first transfer pass, fixation in the second transfer pass may occur as yet because of the

increased region proportions.

In principle, more transfer passes may follow (with gradually increasing regions) until no transfer modules are left. In case of persistent non-fixability the following last resort is possible: undoing all fixations in the (increased) region of a block and reperforming a branch-and-bound fixation attempt together with modules from the transfer list.

10.5 A 2-phase simultaneous approach

A last suggestion for obtaining a good simultaneous placement and global routing result for gate arrays that differ very much from the original Burstein model is the following:

It applies to the case where the ratio of the number of legal positions per unit of gate array surface area divided by the average stamp shape area is relatively large, i.e. where many placements exist that can be obtained one from another by translating a stamp over a distance that is only a fraction of the stamp's own proportions.

In this case the hierarchy deviation's distribution over several levels is most likely to leave some modules that are non-placeable in the end. This is because the area-count for loose module occupation reflects final placeability constraints for loose modules least effectively in the case of such gate arrays. This can be illustrated by comparing the gate arrays of example 2 and 3 in section 2.3:

The image cell proportions in example 2 make that the distance between neighbouring legal positions of a stamp are of the order of stamp lengths, whereas the stamp shapes in example 3 extend over many equivalent legal positions. As a consequence, judging final placeability expectations for loose modules from the block occupation will be more adequate for the gate array of example 2.

As a result of this vital difference, example 3 is more likely to finish with a large-scale rearrangement on the lowest levels (which spoils much of the optimality attained on higher levels) than the gate array of example 2: paradoxically, although the total placement restrictions in example 2 are much nastier, example 3 will be more difficult for the simultaneous placement and global routing strategy, just because it fits least with the original placement-cell based picture in the Burstein approach [4]!

Since the extensive rearrangement that may result in example

3 will throw away much optimality, a subsequent reperforming of the entire placement and global routing strategy (from the root level hierarchy onwards) is suggested. The only difference with the first phase is that the set of stamp choices and the legal positions per stamp type are confined to those of the final placement result of phase 1.

Thus, the final matching from the first phase is preserved (to guarantee a bottom-level placeability without rearranging) whereas the positions and stamp choices of modules that are instances of the same macro may be arbitrarily exchanged. Moreover, although terminal positions are part of the normal hierarchy evolution (including unpredictability of low-level multi-cell terminal configurations on higher levels), the exact capacity landscape is known right from the beginning of the second phase.

11. 2xN routing algorithm

11.1 Differences with stand-alone global routing

The 2xN routing algorithm will not be discussed extensively in this report, since its essentials have, in all details, been described by Nuyten [23], [24]. Here the differences between the 2xN routing in the context of simulatneous placement and global routing and the 2xN routing for a completed a priori placement as in [5] will be concentrated on. Most of the differences are concerned with datastructuring and the retrieval of the individual 2xN problems per net from the total hierarchy framework:

The global routing representation must be kept consistent with mutating placement situations and the interweaving of, on the one hand, hierarchy levels (as described in chapter 8 and 10) and (on the other hand) of full-scope block array steps, 2xN strip-scope subproblems and local block-scope stages. In this complicated situation, datastructures and conversion between these based on ordered-list representations cannot be used. Also 2xN problem extraction has to be done in a completely different way, using new labeling techniques, datastructures etc.

Moreover, vertex-based route representations, as used in the stand-alone global router, are fundamentally unsuited to the simultaneous problem that has an undetermined placement that will deviate on low levels form the high level hierarchy evolution: after transfers etc. non-valid routes must be recoverable.

The essential differences can be found in chapter 12. Details follow from the source program itself. In this chapter a short overview of the 2xN routing approach and the consequences of the general modelling in contrast to the original Burstein model is discussed.

11.2 The inadequacy of the 2x2 integer programming approach

In a general gate array context, the routing of nets in the 2xN strip must be done one at a time. The divide-and-conquer heuristic that reduces the total 2xN problem to a collection of 2x2 routing problems to be solved net-order independently by an integer programming technique (see [5], [6], [7], [8]), is not applicable for gate arrays in general, because of the following reasons:

- Non-uniformity of the 2xN capacity landscape makes it impossible to capture the strip's bottlenecks effectively in the reduction to the 2x2 problem

hierarchy. Burstein [4] does not use it either for the simultaneous gate array placement and global routing methodology.

Note: The 2x2 hierarchy has nothing to do with the essential global routing hierarchy described in this report: it is a hierarchy of heuristic reductions of 2xN strip routing problems to a number of 2x2 problems concerned with the routing between four variable-level supercells (belonging to yet another supercell abstraction).

- Net-specific features in the global routing landscape (introduced by multi-cell terminals) cannot be accomodated in the integer programming formulation of constant complexity (fixed tableau size). A variable integer programming formulation with net-specific constraint entries (as discussed in [22]), is infeasible from the point of view of computation complexity.

By the way, the 2x2 integer programming approach is described in detail in the first version of the report on global routing by Nuyten [23], but the present version of the stand-alone global router (reported on in the revised version of the report [24]) does not contain a 2x2 algorithm anymore.

11.3 The 2xN routing problem per net part

So, 2xN routing is done in a one-net-at-a-time fashion. Fortunately, the influence of net ordering is confined to the 2xN solution: the total routing on the final level is only weakly dependent on net ordering.

The original 2xN algorithm ([4], [5]) was based on single-cell terminal modelling assumptions. It is a linear algorithm that finds the minimal cost tree interconnecting a set of terminals in the 2xN grid.

For a given net (or net part, in case a net yields several subproblems which concern subtrees that are to be kept unconnected by way of detouring constraints: see chapter 7) the 2xN routing problem has the following form (in case of single-cell terminals):

Let the 2xN grid be represented as a lattice graph $G_{2xN}(V,E)$ as was described in section 4.2. Terminal positions (including pseudo-terminals: see chapter 7) correspond with a set of terminal-vertices $T \subseteq V$.

With every edge $e \in E$ a cost value $\chi(e)$ is associated which is a function of the boundary capacity b(e) of the corresponding supercell boundary. Usually, the edge costs are exponentially decreasing with boundary capacities, but the exact function parameters can only be determined by an experimental "fine tuning" of the algorithm on realistic problems.

Although Burstein does not use them in the case of simultaneous placement and global routing [4], cell cost $\phi(v)$ may be added for each vertex $v \in V$. In Burstein [5], [6], [7], [8] and Nuyten [23] these are used, in the form of via costs, to represent via capacity constraints (used for restricting the number of "bending" routes). But the general cell capacity definition in chapter 3 (for the general gate array context, where global "bending" constraints, corresponding with via constraints in case of 2-layer superposition type conventions, are not meaningful) yields cost values that can be incorporated by simple extension modifications, based on the same principle.

Finally, the problem definition for strip routing of a net part comprises also a specification of the first and the last vertex column in the 2xN graph that follow from detouring constraints.

The solution of the net part problem is the minimal interconnection tree I, i.e. the set of edges $I \subset E$ that forms a tree connecting all terminal vertices $t \in T$, and for which the total cost

$$\Sigma_{e \in I} \chi(e)$$

is minimal.

11.4 The 2xN dynamic programming algorithms

The Burstein 2xN algorithm is a modification of the algorithm developed in [1], which finds a Steiner tree connecting all terminal vertices. The (rectilinear) Steiner tree corresponds with the solution in the special case where all costs are equal.

The minimal interconnection tree (for arbitrary cost values) is constructed by using a *dynamic programming* approach where every column (vertex pair) between the first and the last following from detouring constraints corresponds with a decision stage at which each of four possible partial trees is extended to yield new minimally-weighted partial trees. The four possible partial trees are defined as follows (supposing that the sequence of stages corresponds with

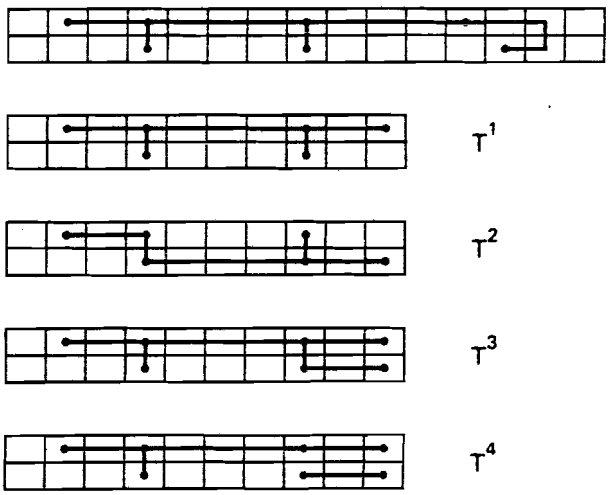
moving from left to right starting at the column of the leftmost terminal vertex and proceeding up to the column of the rightmost terminal vertex):

- T1 is the minimal cost tree interconnecting all terminals at the left of the stage column plus the upper vertex of the stage column.

- T2 is the minimal cost tree interconnecting all terminals at the right of the stage column plus the lower vertex of the stage column.

- T3 is the minimal cost tree interconnecting all terminals at the right of the stage column plus both vertices at the present column.

- T4 is the minimal cost forest consisting of two different tree (each connected to one vertex of the stage column) such that all terminals to the left of the stage column are interconnected by either one of the trees (this means that the trees have to be joined later, i.e. on some column on the left of the stage column).



Routing by Dynamic Programming.

figure 11.1

It is possible to compute the "trees" 1,2,3 and 4 at the next column from the trees at the previous column. The exact procedure is given in details in [23].

In the present version of the global router the only forms of multi-cell terminals that are allowed are constituted by fixed connections between adjacent supercells. In the second version of the above report [24], the extension of the Burstein dynamic programming algorithm to this case is described. As was argued in chapter 3 and 4, general gate array modelling necessitates other forms of multi-cell terminals.

A straightforward extension of the dynamic programming concept is only possible if multi-cell terminals do not "skip" a column in the lattice graph: in that case, an optimality-preserving decision cannot be made at each stage from the partial tree state on the previous stage alone (for optimal decisions more than two columns must be taken into account).

However, since the occurrence of such multi-cell terminals will be rather rare, one can use a heuristic subdivision of the original problem into two separate dynamic programming problems: one for columns to the left of the skipped column and one for columns on the right of the skipped column.

The complexity of the routing of a single net is linear in the number of columns between the bounds set by the detouring constraints.

11.5 Combined routing of several nets

After a net has been routed, its route is imbedded in the 2xN strip by updating boundary capacities for those boundaries that are crossed by an edge of the minimal interconnection tree. Similarly, cell capacities are updated.

In general, there may be a considerable difference (not only in absolute terms, but also with regard to the relative costs of different edge alternatives) between the cost landscape during the routing of the first nets (when hardly any capacity has been consummated yet) and during the routing of the last net when the presence of other net routes is clearly felt by means of high cost values for crowded boundaries (possibly infinite, in case of overflows).

Therefore, it may be worthwhile to do *rerouting* for nets that have been routed earlier on. The rerouting of a net n

consists in removing the route of the net, updating capacities, computing new cost values and executing the 2xN dynamic programming procedure again.

There are several options for choosing the order of nets for a rerouting pass, and the number of rerouting passes can be variable. Experimental observations on the practical use of these options are given by Nuyten [23].

12. Algorithm implementation

12.1 General issues

In this chapter the foregoing is summarised and restated in terms of implementation aspects. It is intended as the bridge between the textual descriptions of strategies and algorithms from the previous chapters on the one hand and the program sources on the other hand. Some tricky details have been omitted because they can only be explained by going into many more implementation details falling outside the scope of this report. Also, the routing components of the program have not been described in all details, although considerable work has been spent on them (in particualr on their insertion in the mutating placement datastructuring). This is partly because their description has to do with details that can be found in [23] and [24] and the implementation of the stand-alone global router. Furthermore, the Lee routing implementation for reconnection has not been completed: it should be based on certain extensions that have been developed at first as contributions in discussions on the local router [28].

The combination of placement and global routing which both evolve in a hierarchical manner, but with mutating mutual influences which require carefully separated datastructure frameworks, make that the total datastructure is complicated. Therefore the essential structure has been illustrated in this chapter by a number of figures. All details that are of minor importance and which can be found in the program text, e.g. fields of all essentiali *items* (i.e. strongly problem-related data objects) and the less important lists, arrays etc., have been omitted.

Because of the large number of lists etc., the main structure in the form of item relationships has been given by way of the shorthand notation indicated in figure 12.1.

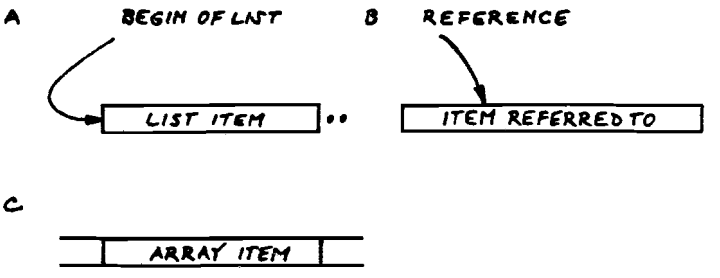


figure 12.1

In the other figures that accompany the algorithm descriptions in a pseudo-language formulation, only the relevant part of the datastructure that is involved in the described algorithm steps is shown. By the way, the complexity of the placement-routing interactions and the interleaving of respective hierarchy levels makes that some of the corresponding algorithm steps have been represented here in a way that is different from the modular program construction. The completed parts of the program itself will clearify the subtleties.

Figure 12.2 shows the structuring of input-related items.

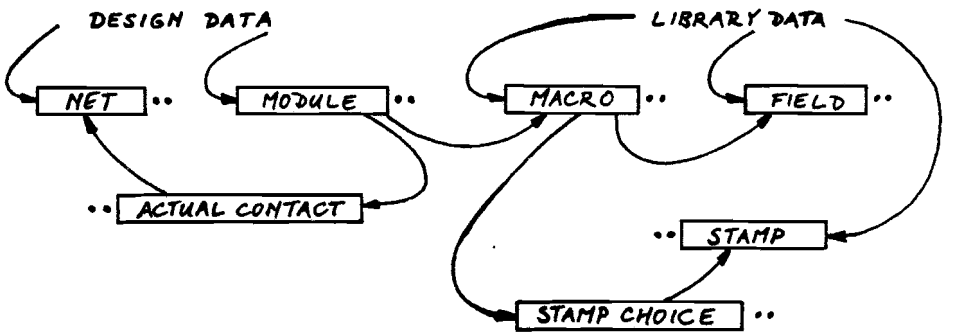


figure 12.2

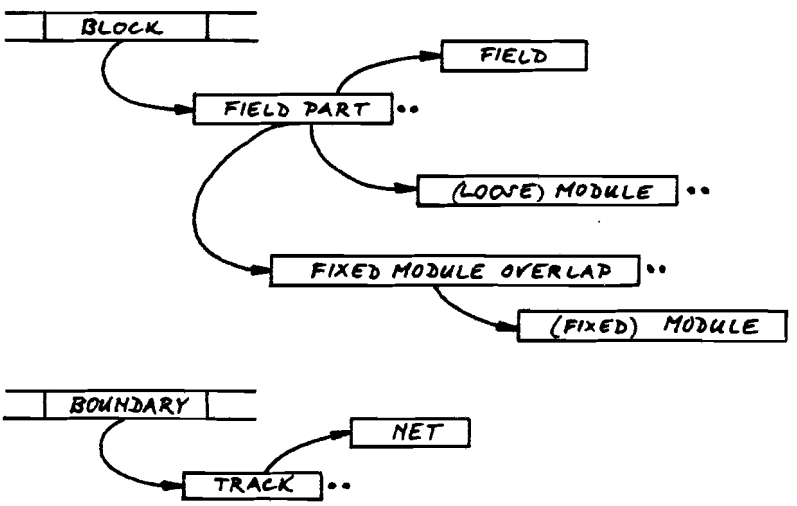


figure 12.3

12.2 Main hierarchy framework

The main datastructure for representing the state at each level is shown in figure 12.3.

```
begin
 input design data and library data;
 compose block of level 0;
 i:= 0;

 repeat
  if i < i_max
   then
    bipartitioning level  i → i+1;
    contraction level i+1;
    creation level i+1;
    if i < i_max-1
     then
     , look-ahead at level i+2;
     else
      look at final regions equal to blocks;
    fi;
    rearrange level i+1;
    reconnect route of level i;
  fi;
  i:= i+1;

{ state level i. }

  if i > 0
   then
    split boundaries level i-1 → i;
    split blocks level i-1 → i;
    global routing level i-1 → i;
  fi;
 until i = i_max;

{ state level i_max, i.e. bottom level. }

 output resulting design data;
end;
```

12.3 Bipartitioning

```
begin
 for all rows to be cut
  do
   for all blocks to be cut
    do

     add coupling-contribution to net distributions (if appropriate);
     add pre-locked connections;

   { establish initial partition: }

     for all fields in block
      do

     { partition loose modules of field: }

       compose list of pins of (involved modules) per net;
       construct;
       improve;
       lock modules;
       remove list of pins;
      od;

   { iterate over fields: }

     repeat
      with some field
       do
        unlock modules;
        improve;
        lock modules;
       od;
     until no further improvement;

     for all fields
      do
       unlock modules;
      do;

     subtract per-locked connections;
     subtract coupling contribution (if appropriate);

    od;
  od;
end;
```

12.4 Improvement by recursion

```
begin
 perform improvement passes;
 repeat

{ orthogonalise: }

  for both partition sides
   do
    set modules of other partition side apart;
    scale down balance restrictions;

  { partition the set of modules from the old partition side: }

    construct;
    improve;
   od;
  join partition results of both old partition sides' module sets;

  perform improvement passes;
 until no improvement after last orthogonalisation;

 restore partition state after last orthogonalisation;
end;
```

12.5 Pass-wise improvement

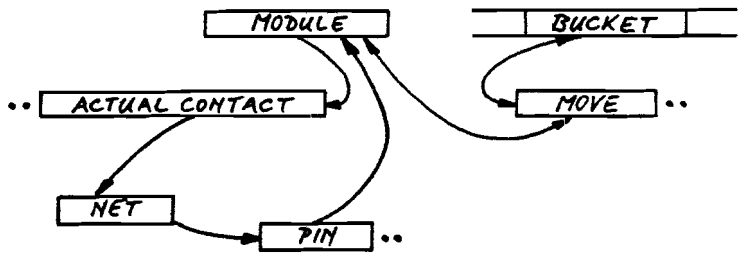


figure 12.4

```
begin
 repeat

{ improvement pass: }

  for all modules to be partitioned
   do
    compute gain;
    enter corresponding move in bucket array;
   od;

  repeat
   choose highest-gain move causing no imbalance;
   lock corresponding module on opposite partition side;
   remove move from bucket array;
   for all unconnected nets
    do
     update net distribution count;
     if critical distribution change
      then
       update gains of moves associated with connected free modules;
     fi;
    od;
  until no more moves possible;

  remove remaining moves from bucket array;

{ restore best intermediate partition state: }

  for all moves done after the best partition side
   do
    undo move;
   od;

 until no improvement in last pass;
end;
```

12.6 Constructive partitioning

The datastructure during the constructive partitioning step is essentially the same as with the improvement pass algorithm. Therefore, see figure 12.4. The most important difference is that every module refers to *two* move items instead of *one*.

```
begin
 for all modules to be partitioned
  do
   for both partition sides
    do
     compute force;
     enter corresponding move in bucket array;
    od;

 repeat
  choose highest-gain move causing no imbalance;
  lock corresponding module on corresponding partition side;
  remove move from bucket array;
  for all connected nets
   do
    update net distribution count;
    if critical distribution change
     then
      update forces of moves associated with connected free modules;
    fi;
   od;
 until no more moves possible;

 remove remaining moves from bucket array;
end;
```

12.7 Rearrangement

The pick-up-and-refix resort in case of persistent non-placeability has been omitted for the sake for brevity: it is a trivial extension of the transfer pass, making use of the same fixation scheme by way of branch-and-bound approach. The datastructuring during rearrangement step is shown in figure 12.5.

figure 12.5

```
begin
 collect field parts per field;
 for all fields
  do
   spread field parts of this field over block array;
   enter zones in bucket array;

 { fixation pass: }

   repeat
    with highest-occupation zone
     do
      check placeability in half regions for all modules;
      try to fix non-placeable modules;
      update occupations of adjacent zones influenced by fixed modules
      enlist non-fixable modules for transfer;
      remove zone from bucket array;
     od;
   until no more zones left;

 { occupation analysis: }

   for all zones
    do
     if over-occupation
      then
       enlist some loose modules for transfer;
      fi;
    od;
```

(see next page)

```
 while transfer list not empty
  do

 { transfer pass: }

   re-enter zones in bucket array;
   repeat
    with lowest-occupation zone
     do
      check placeability in halfregion for transfer modules;
      try to fix non-placeable modules that are placeable in whole reg
      for adjacent zones influenced by fixed modules
       do
        update occupation degree;
        if over-occupation
         then
          enlist some loose modules for transfer;
        fi;
       od;
      assign placeable modules in case of no over-occupation;
      remove zone from bucket array;
     od;
   until no more zones left;

   enlarge whole region temporarily in case of other subsequent passes
  od;

 recollect spreaded field parts;
end;
```

12.8 2xN routing

The retrieval of different routing problems per net that are to be kept unconnected and the computation of detouring constraints for this purpose is difficault. The simulation of a finite-state machine used in the stand-alone global router ([23], [24]) is infeasible because of its use of an implicit ordering of vertices in the datastructure.

Here the retrieval of 2xN routing problems from a 1xN routing situation is treated in the same way as in the case of rerouting or "sliding-window" routing (see [5], [23]) by adding a simple transformation of 1xN routes into 2xN routes connecting the corresponding descendent supercells (preserving separation of different net parts in the strip): duplicating all 1xN edges in the 2xN strip and adding orthogonal edges for every occupied column in the boundary-splitting step yields a connected (though not loop-free) 2xN subgraph of the lattice graph for every separate problem.

Labeling techniques make a retrieval and detouring-check possible by searching these graphs after "spreading" them on an array representing the lattice graph. (Exactly the same efficient technique should be used in the reconnection routing step.)

The datastructuring that is used in the routing phase is illustrated in figure 12.6.

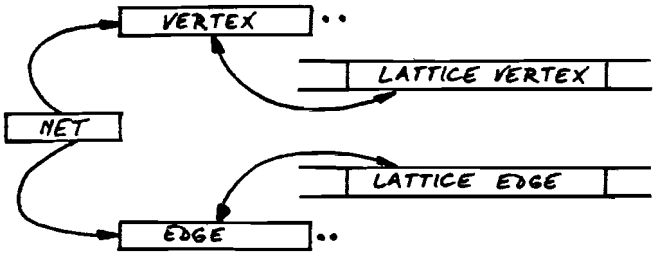


figure 12.6

```
begin
 for all 2xN strips to be routed
  do
   for all blocks
    do
     collect terminal vertices and pseudo-terminal vertices per net;
     collect route edges per net (removing tracks);
    od;

   repeat
    if imbedded rerouting
     then
      compute costs from presently-consummated capacities;
     else
      reset capacities and costs;
    fi;

    for all nets involved (in some order)
     do
      spread all vertices and edges on the 2xN lattice;
      label all connected vertices;
      for all separate net parts (separately labeled subgraphs)
       do
        compute detouring constraints;
        remove all edges and non-(pseudo-)terminal vertices;
        if imbedded rerouting
         then
          update capacities and costs;
        fi;
        find minimal interconnection tree;
        add all new edges and label new vertices;
        update capacities and costs;
       od;
      remove all vertices and edges from the 2xN lattice
      (undoing labeling);
     od;

   until no more rerouting;

   for all nets involved
    do
     remove vertices;
     remove edges (creating corresponding tracks per boundary);
    od;
  od;
end;
```

Conclusions

At first sight a simultaneous placement and global routing component for a flexible gate array layout system seemed to be possible by starting from the algorithms proposed by Burstein [4]. Therefore the initial intention of the graduation work presented here was directed at a further improvement of the placement aspect of the Burstein procedure by using more sophisticated min-cut partitioning heuristics.

However, it turned out that the ostensible generalities of the gate array modelling on which this procedure is based, plus some additional design prerequisites, are not generally applicable for gate arrays. That this dissonance is fundamental, has been shown in this report by an extensive reorientation on the essentials of placement and global routing for gate arrays. The consequences of this fact for the basic strategies has been shown for both placement and global routing individually, as well as in conjunction.

The most important problem for a simultaneous placement and global routing methodology is the fact that a strict hierarchy of (qua admissability criteria) independent subproblems evolving straightforwardly as in the Burstein-case (albeit with a variable-level transition to a placement fixation), is impossible in principle. This requires far-reaching fundamental modifications of the original strategy.

New adaptations and extensions have been discussed in this report, necessarily covering all of the entire algorithmic conglomerate of placement, min-cut partitioning, rearrangement, global routing and reconnection. The implications of the much more complicated modelling, algorithms and datastructuring have prevented a completion of the program implementation in the scope of the graduation period that is reported on here.

Acknowledgement

The author would like to thank the members of the Automatic System Design group for a year of pleasant cooperation. Special thanks are due to prof.dr.-ing. Jess for coaching this graduation period.

References

[1] Aho A.V., M.R. Garey, F.K. Hwang, "Rectilinear Steiner Trees: Efficient Special Case Algorithms", Networks, vol. 7, 1977, pp. 37-58.

[2] Breuer M.A., "Min-Cut Placement", Journal of Design Automation and Fault-Tolerant Computing, vol. 1, no. 4, Oct. 1977, pp 343-362.

[3] Burstein M., "Analysis or a Network Partitioning Technique", Proc. 15th Int. Symp. on Circuits and Systems, ISCAS 1982, pp. 477-480.

[4] Burstein M., S.J. Hong, R. Pelavin, "Hierarchical VLSI Layout: Simultaneous Placement and Wiring of Gate Arrays", VLSI 83: VLSI Design of Digital Systems, Proc. IFIP Int. Conf. on Very Large Scale Integration, Trondheim, 16-19 Aug. 1983, F. Anceau and E.J. Aas (eds.), pp. 45-60.

[5] Burstein M., R. Pelavin, "Hierarchical Wire Routing", IEEE Trans. on Computer-Aided Design, vol. CAD-2, no. 4, Oct. 1983, pp. 223-234.

[6] Burstein M., R. Pelavin, "Hierarchical Channel Router", Integration, vol. 1, 1983, pp. 21-38.

[7] Burstein M., R. Pelavin, "Hierarchical Channel Router", Proc. 20th Design Automation Conference, 1983,pp. 591-597.

[8] Burstein M., R. Pelavin, "Hierarchical Channel Router", Computer-Aided Design, vol. 16, 1984, pp. 216-224.

[9] Chen J.Z., W.B. Chin, T.-S. Jen, J. Hutt, "A High-Density Bipolar Logic Masterslice for Small Systems", IBM. Journal of Research and Development, vol. 25, no. 3, May 1981, pp. 142-151.

[10] Corrigan L.R., "A Placement Capability Based on Partitioning", Proc. 16th Design Automation Conference, June 1979, pp 406-413.

[11] Deutsch D.N., P. Glick, "An Over-the-Cell Router", Proc. 17th Design Automation Conference, 1980, pp. 32-39.

[12] DeWilde P. (ed.), "The Integrated Circuit Design Book", Delft University Press, Delft, 1986.

[13] Fiduccia C.M., R.M. Mattheyses, "A Linear-Time Heuristic for Improving Network Partitions", Proc. 19th Design Automation Conference, 1982, pp. 175-181.

[14] Hightower D.W., F.G. Alexander, "A Mature I2L/STL Gate Array Layout System", Proc. COMPCON Spring 1980, pp. 149-155.

[15] Hopcroft J.E., R.M. Karp, "An $n^{5/2}$ Algorithm for Maximum Matching in Bipartite Graphs", SIAM Journal of Computing, vol. 2, no. 4, 1972, pp. 224-231.

[16] Jess J.A.G., R.J. Jongen, P.A.C.M. Nuyten, J.C. Bu, "A Gate Array Design System Adaptive to Many Technologies", Proc. IEEE Int. Conf. on Computer Design, 1984, pp. 338-343.

[17] Kernighan B.W., S. Lin, "An Efficient Heuristic Procedure for Partitioning Graphs", Bell Syst. Techn. Journal, vol. 49, Feb. 1970, pp. 291-307.

[18] Kirkpatrick S., C.D. Gelatt Jr., M.P. Vecchi, "Optimization by Simulated Annealing", Science, vol. 220, no. 4598, 13 May 1983.

[19] Krishnamurthy B., "An Improved Min-Cut Algorithm For Partitioning VLSI Networks", IEEE Trans. on Computers, vol. C-33, no. 5, May 1984, pp. 438-446.

[20] Lee D.T., S.J. Hong, C.K. Wong, "Number of Vias: A Control Parameter for Global Wiring of High-Density Chips", IBM Journal of Research and Development, vol. 25, no. 4, 1981, pp. 261-271.

[21] Lee C.Y., "An Algorithm for Path Connections and Its Applications", IRE Trans. on Electronic Computers, vol. EC-10, no. 3, Sept. 1961, pp. 346-365.

[22] Morrison C.R., G.D. Hachtel, "A Preliminary Report on Generalized Hierarchical Channel and Switch-Box Routing", Proc. 26th Midwest Symposium on Circuit and Systems, Aug. 1984, pp. 369-373.

[23] Nuyten P.A.C.M., "Hierarchical Wire Routing Of Gate Arrays", master thesis, Eindhoven University of Technology, Department of Electrical Engineering, Feb. 1985.

[24] Nuyten P.A.C.M., idem, revised version, June 1985.

[25] Papadimitriou Ch., K.H. Steiglitz, "Combinatorial Optimization, Algorithms and Complexity", Prentice-Hall, 1982.

[26] Schweikert D.G., B.W. Kernighan, "A Proper Model for the Partitioning of Electrical Circuits", Proc. 9th Design Automation Workshop, June 1979, pp. 57-62.

[27] Shiraishi H., F. Hirose, "Efficient Placement and Routing for Master-Slice LSI", Proc. 17th Design Automation Conference, June 1980, pp. 458-464.

[28] Slenter A.G.J., "Local Routing of Gate Arrays", master thesis, Eindhoven University of Technology, Department of Electrical Engineering, Oct. 1985.

[29] AMI Uncommited Logic Array design manual.

[30] Texas Instruments, Low Power Schottky Logic Arrays, TAL user design guide.

[31] EXAR semi-custom product guide.

[32] Twente University of Technology, gate array design manual.