IDS : an interactive design system for integrated circuits

THE-RC 50403

Eindhoven University of Technology

Computing Centre Note 11

IDS: AN INTERACTIVE DESIGN SYSTEM FOR INTEGRATED CIRCUITS

der Woude

October 1982

1. Introduction

In this report an interactive design system for integrated circuits (IDS) is outlined. The system is presently under construction within a software project in which participate the researchgroup ES of the department of EE and the computing centre of the THE. The following persons are active within this project:

J. Jess	(ES)
K. Delhij	(ES, student for master degree)
L.R.A. Kessener	(Comp.Centre)
Th. van Ooyen	(ES, student for master degree)
X. Timmermans	(ES, student for master degree)
M. van der Woude	(Comp.Centre)

The design system has to be as portable as possible between B7700 and PDP11T60, it has to be usable for IC-design for the planned 'IC-production unit' at the THE. The system is to be part of the 'NELSIS'-system [1] in which co-operate the Dutch Universities of Technology of Delft, Eindhoven and Twente. The designs produced with this system have to be portable to facilities of the N.V. Philips, preferable at each stage of the design. Therefore a number of its requirements are given. Further it is intended to co-operate within this project with Philips Physical Laboratory, so a number of the described requirements are the result from discussions with Philips.

Within the NELSIS-project it has been agreed to define in the near future standardized design description languages. We distinguish:

- Network Description Language (NDL)
- Symbolic layout description language (SLDL)
- Geometric layout description language (GLDL)

These languages, still to be defined, will serve as input languages for IDS, besides interactive input with graphics terminals.

At the THE within the group ES there is a project going, in which a network simulator is designed based on Piecewise-Linear Modelling [2]. The aim is a mixed-mode mixed-level simulator, which can simulate integrated circuits from circuit level to logic level and to behavioural level. This PWL simulator will become in future part of IDS. However it will not be discussed here; in this report we will concentrate on the design and verification of I.C. layouts.

2. Design Description Languages

A global sketch of the design process of an integrated circuit is given in fig. 1. Roughly four stages can be distinguished. In general the design process will be iterative, both top down and bottom up. The four global stages are:

- behavioural specification
  Usually this is not done in a formal way. To formalize this behavioural specification one needs a "Behavioural Specification Language" (BDL). Existing examples are: ISPS [3], behavioural section of HDL [4]. From the behavioural specification, formalized or not it has to be possible:
  - to derive functional tests
  - to verify simulation results

- network definition
  The network is formally described in a Network Description Language (NDL). Existing examples of a NDL are SPICE, HDL and TEGAS. From the NDL description of a network it has to be possible:
  - to simulate the circuit
  - to design testpatterns
  - to design and verify the layout

- layout design
  The layout design will be distinguished into two levels:
  - symbolic layout design ("STICKS-level") described in SLDL (Symbolic Layout Design Language)
  - geometric layout design (mask-level) described in GLDL (Geometric Layout Design Language)

  An example of an existing SLDL is ICDL [5], examples of existing GLDL's are CIF [6] and CIRCUITMASK [7]. The latter is used within the Philips concern. From SLDL there is a translation possible to GLDL, but not conversely, since in GLDL more exotic geometries of layout components are permitted.

We will concentrate in this report on SLDL and GLDL, discuss relations with NDL and leave BDL for the future. Also we will not pay attention to testing, testability and test pattern generation though these subjects have to be carried in mind during all stages of the design. The following requirements are desired for NDL, SLDL and GLDL:

- agreed standards
- hierarchical structuring and separation
- same hierarchical structure for one design as far as possible

The benefits are:

- design sharing due to portability of textfiles
- facility sharing
- design editing with texteditors is possible
- simple design verification by mapping hierarchical

structures onto each other
- if domain overlap of different cells is restricted to a limited number of special cases layout verification with the circuit description and with the designrules can be done per compound cell, i.e. the required computing resources increase linear with the length of the description.

Finally standard design languages are a necessary but not sufficient condition for portable software.

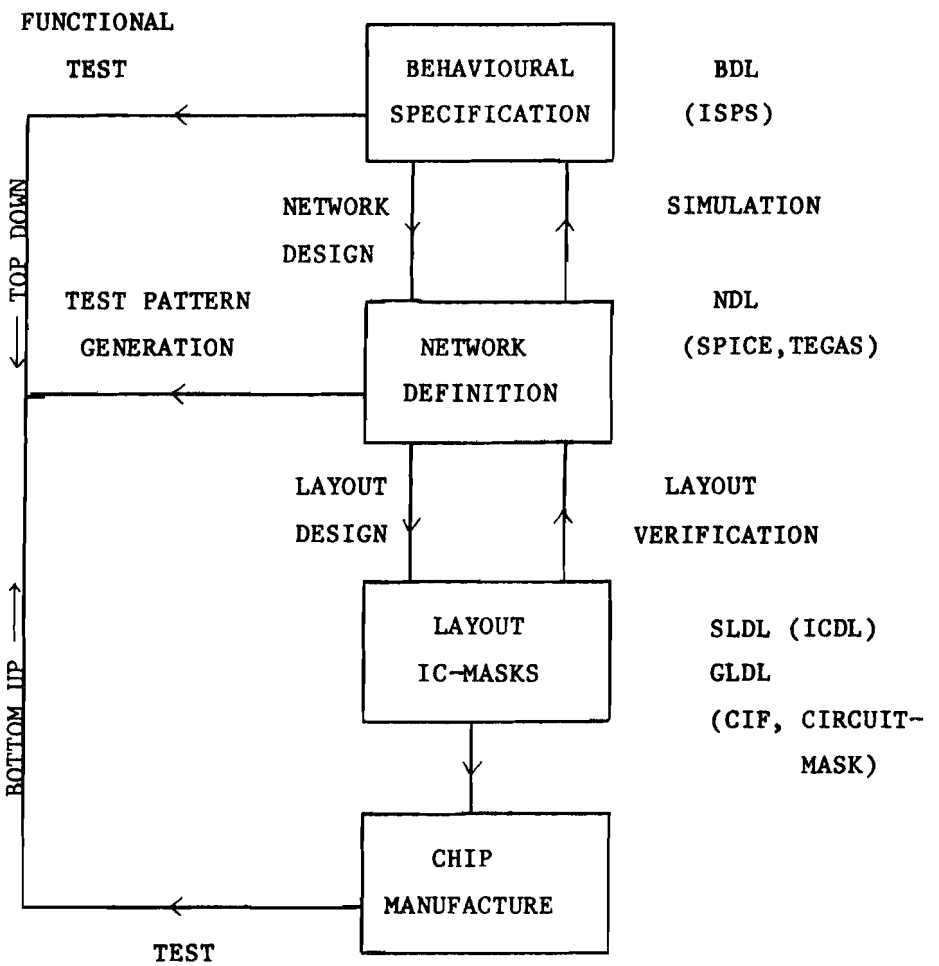

Fig. 1 Design process of an integrated circuit.

3. Hierarchical Design Data Structure.

We have seen in the previous section that it is advantageous to make designs of integrated circuits in a hierarchical way. In fact, when circuits are very large, it may be the only way of desigr within reasonable time and within reasonable use of (computer ard human) resources. However we have to pay for these advantages by:

- some loss in efficiency of chip area, due to necessary domain separation [8]
- restricted freedom in design
- more conservative design rules especially at domain boundaries

In fig. 2 we give a schematic example of a chip layout with a hierarchical structure. In fig 3. the graph representing this hierarchical structure is given. A (directed) edge in this graph corresponds with an instant call of a cell, a node corresponcs with a compound or leaf cell definition.

Note:The graph is not a tree and it is acyclic.

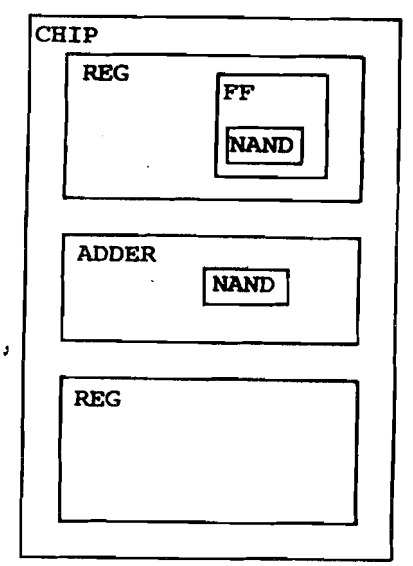

Fig. 2 Example of hierarchical chip layout.

It will be clear that it is possible to have the same hierarchical structure for a design in NDL, SLDL and GLDL. However since the amount of detail increases when the design proceeds, it may be useful or necessary in practise to have more than one SLDL-version of one NDL compound, or to have more than one GLDL-version of one SLDL compound. It even can occur that hierarchies differ in the lowest hierarchical echelons. However this has implications for network verification. To store descriptions into a database we can use the same structure for NDL, SLDL and GLDL. As details may differ we will describe in section 5 the internal datastructure for SLDL.

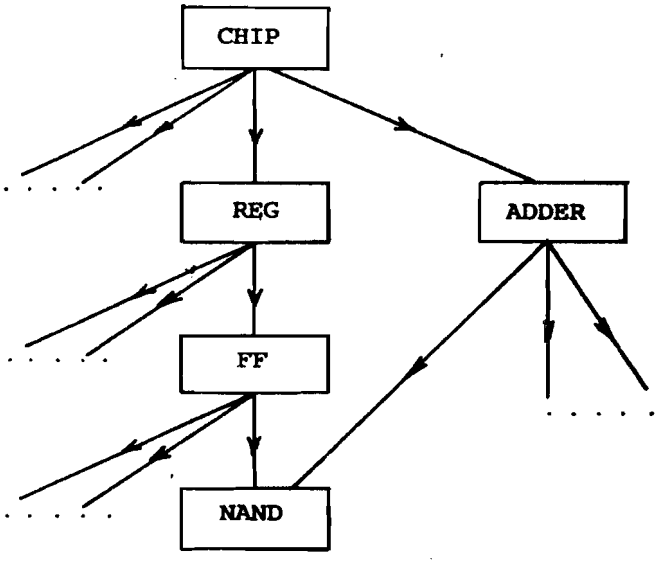

Fig. 3 Graph representing hierarchical structure of chip of fig. 2.

4. The symbolic layout editor of IDS

The purpose of symbolic layout design is to deliver the designer from the evil of superfluous detail. Therefore the geometry in a symbolic layout design environment is represented in a symbolic way. The advantages of symbolic design are well-known:

- design rule check simplified by the use of symbolic design rules
- incremental design rule check is possible
- computer-tools such as compactor and router are easy made available
- increase in design speed, in [14] Black and Hardage report a reduction in design time of about a factor 10

These advantages have to be paid for by some loss in area efficiency due to some restrictions in design freedom. However by making building blocks defined in GLDL accessable, this loss can be overcome to a great extent. It is expected that the only feasible way to design layouts of VLSI size will be by symbolic means.

It is important that the symbolic pictures look like the geometric ones. The designer must be able to predict from the symbolic layout how the ultimate mask layout will look like. Various ways of symbolic drawing have been proposed [9], [10], [11] with topological or virtual grids. However as in [12], and [13] we prefer a geometric grid for representing symbolic layouts, to meet the above mentioned requirement of predictability. One grid unit (lambda) will be a characteristic minimal distance of the design rules, e.g. a half minimal linewidth.

The basic building bricks or leave cells in the symbolic layout environment are called STICKs. We distinguish the following types of STICKs:

- Linepieces of various types, e.g. metal, polysilicon and diffusion. Since a linepiece is only in one layer and is always rectangular it can be defined by instantiation. All other STICK cells except for text strings have to be defined before they can be placed.
- Vias to go from one layer to another.
- Contacts or pins of different layers to indicate locations on the boundaries of compound cells were connections to the outside world can be made.
- Shared buses to denote areas where compound cells may overlap.
- Depletion type MOS transistors.
- Enhancement type MOS transistors.
- Other elements to be defined when necessary, e.g. bipolar devices.
- Text strings to be plotted as comment on screen or mask.

Each of these basic building bricks is represented on the screen in a symbolic way. In fig. 4 some STICK-representations are given.

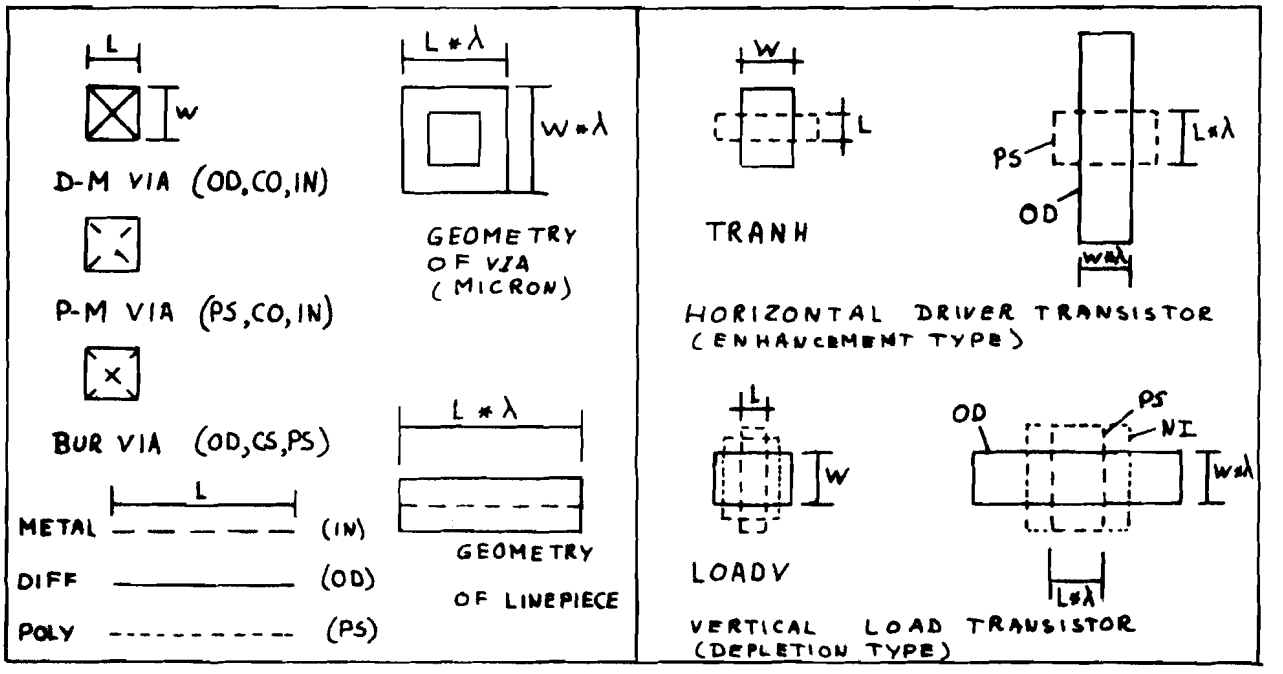

Fig. 4 Symbols and layout of some STICK cells.

Note:

- A linepiece is indicated by its centerline, the physical dimensions can be obtained by blowing it up (see fig. 4) by a distance of half its width. If no color graphics is available distinction of linetypes can be made by various ways of dashing. The linewidth cannot be read from such a symbolic representation.
- The symbols of via, pin and transistor indicate their characteristic length and width. The exact geometric representation may be obtained either by specifying it via a GLDL description or in a default way by the editor, see below.

As a composite building brick we use the (compound) cell or shortly compound. Such a cell is built up of other cells and STICKs. It may be represented on different hierarchical levels. The hierarchical level will be indicated by a figure called level number, ranging from 0 upwards. The higher this level number the more detail. If a compound is drawn on level 0 then its surrounding rectangle, its pins and its name are drawn. On level 1 in addition its STICKs and all instantiated compound cells are drawn on level 0, and the name is suppressed, etc. When a compound cell defined within GLDL is instantiated, its GLDL name (which may differ from its SLDL name), surrounding rectangle and pins (names and locations) have to be provided to the editor. Of course there is only one level of drawing possible in this case.

The composition of symbolic layouts is gouverned by a set of symbolic design rules. To keep the design hierarchical we have to take care that interaction of STICK and compound cells occurs only in an allowed way, i.e. by means of pins and shared buses [17]. The following symbolic design rules are proposed:

- Cell boundaries except shared buses are separated from masks by a distance lambda, such that 2 * lambda is sufficient separation between figures of any layer. Only at pins and shared buses, linepieces of the outside world may overlap the cell boundary, these linepieces have to approach the cell perpendicular to its boundary. The overlap in case of pins has to be half the width of the pin, in case of shared buses overlap may be only on the shared buses.
- Separation of STICKs is gouverned by the physical design rules. A table with design rules has to be available within the editor.
- Linepieces may be connected to each other by complete abutment or by complete overlap at at most one of the ends (see fig. 5)

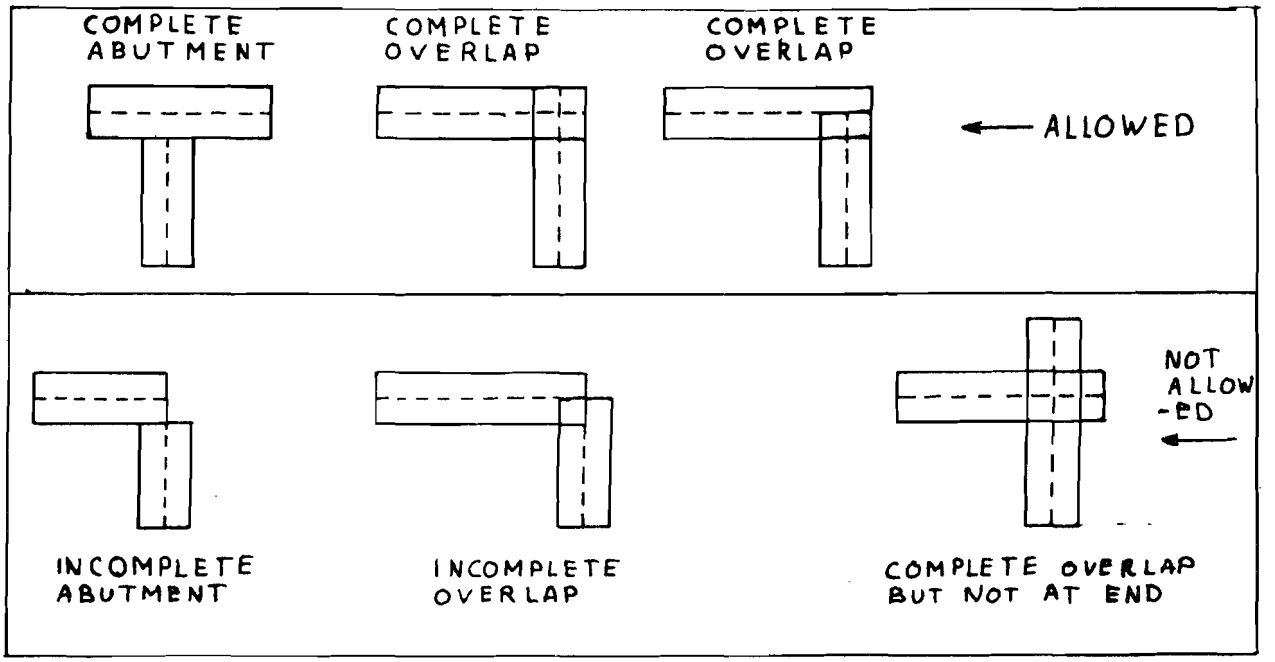

Fig. 5 Interconnection of linepieces.

There are two ways of giving the mask information of a STICK cell. One way is by having it generated by the symbolic editor. In this case the characteristic geometry, together with the set of design rules is used to derive the layout. Another way is to provide a description of the STICK layout in GLDL. In this case the GLDL name, the type and the surrounding rectangle of the STICK have to be provided to the editor. We will refer to these two cases respectively by "default STICK" and "GLDL STICK".

The editor has a number of graphics operations available:

- Input via menu (commands and numbers), keyboard (names), and digitizer.
- Manipulation commands such as modify, place, delete and undo.
- Drawing commands with hierarchical level number, zoom-in and zoom-out facilities.
- Expansion of any compound cell in a drawing to one hierarchical level deeper.
- Unpack i.e. replace a compound cell instance in the database by its contents.
- Translate an internal symbolic description of a design to a GLDL description
- Plot a drawing on one of the available plotters.

To make these operations possible all design data have to be stored in a hierarchical design database which has to be well organized in order to enable:

- quick access, necessary for interactive work
- future extensions easy to be implemented
- compilation of a design described in SLDL to the internal database.

In the following section we give an outline of the proposed database.

5. Internal Datastructure of the Symbolic Design-Database.

Basically the internal datastructure is a mapping of the graph representing the hierarchical logic structure (see e.g. fig. 3) of the design. We represent this mapping by a bidirectionally chained list of variable length records, stored in a file called STRUCTURE. Each record contains a sequence of words of at least 32 bits or 4 characters. Besides that we maintain two directories and three lists:

- CELDIR containing a directory of compound cells and STICK definitions
- NAMELIST containing names of cell instances, nets and pins
- NDLLIST containing names of NDL cells, these names may differ from their SLDL counterparts
- GLDLLIST containing names of GLDL cells, these names may differ from their SLDL counterparts

Any record in STRUCTURE contains:

<integer> denoting length of record in words
<integer> denoting type of record
<pointer> to previous record
<pointer> to following record
<body>

The <body> contains data depending on the type of the record and is described below for each type:

type 1: begin compound cell definition

<body>: <pointer> to end compound cell record
<pointer> to compound name in CELDIR
<pointer> to start of pin section
<pointer> to start of net section
<pointer> to start of instant call section
<4* nill> reserved for future use

If one section is empty the corresponding <pointer> has the value nill. If there is only a GLDL description the pin section should be filled properly. If the pins are explicitly defined within GLDL then this is no problem. So it follows as a requirement for GLDL that it has to be possible to indicate pins explicitly.

```
type 2: begin of net cell definition

<body>: <pointer> to end net cell record
        <pointer> to netname in NAMELIST

        <pointer> to start of pin section of that net
        <pointer> to start of net description

type 3: end compound, net or STICK cell

<body>: empty

type 4: end data structure

<body>: empty

type 5: call of stick or compound cell
<body>: <position>
        <pointer> to directory record in CELDIR
        <pointer> to cell definition record in STRUCTURE
        <pointer> to instant name in NAMELIST
        <trafo>
        <rep info>

type 6: linepiece record

<body>: <layer>
        <width>   of linepiece
        <startpoint>
        <endpoint>
        <trafo>
        <rep info>

type 7: text record

<body>: <position>
        <layer>
        <number of chars>
        <char height>
        <string> containing text to be displayed
```

The types with numbers of 20 and higher indicate definitions of STICK cells. The <body> of a STICK cell is of the form:

```
<body>: <pointer> to end STICK cell record
        <pointer> to name of STICK in CELDIR
        <layer>   indicating layers of the STICK in use
        <pointer> to start of pin section
```

The following types of STICK definitions are distinguished:

type 20: pin definition
type 30: shared bus definition
type 40: via definition
type 50: depletion transistor definition
type 60: enhancement transistor definition

The additional directories and lists are in general much smaller than STRUCTURE. They are built up as sequential files with fixed length records. The contents of the records are respectively:

- CELDIR

  <name> denoting the name of the compound cell
  <box> surrounding rectangle
  <pointer> to the cell definition in STRUCTURE
  <integer> denoting the number of its instances
  <pointer> to the cell-name in NDLLIST
  <pointer> to the cell-name in GLDLLIST
  <word> denoting the design status of the cell
  (to indicate e.g. GLDL description present, NDL description present, etc.)

- NAMELIST

  <name> of cell instance or net

- NDLLIST (GLDLLIST)

  <name> of NDL (GLDL) cell

6. Compaction

One tool that may be useful in combination with a symbolic layout editor is a compactor. A compactor can shift the elements of a symbolic layout together in either X or Y direction such that the layout dimension in that direction becomes minimal without violation of design rules.

Various methods for compaction exist, the methods depend heavily on the symbolic representation. E.g. in the virtual grid compaction method which is used in the MULGA system [5], the minimum distance between each two adjacent virtual grid lines is determined and so the (symbolic) virtual grid layout is converted to a geometric grid layout. A similar method has been used by Williams [9]. In our case the compaction is started with a geometric grid, but still we could use the same methods. However if it is also allowed to shift only a part of the elements on one grid line a more optimal result may be reachec. We will consider the following two methods:

- The constraint graph method (Hseuh [15]).
- Shear line or Compression ridge compaction (Akers [16], Dunlop [10]).

In the constraint graph method elements are partitioned into groups. E.g. with horizontal compaction two elements are in one group if they are connected by a vertical linepiece. Groups may shift with respect to each other, during horizontal compaction horizontal linepieces are allowed to change length. The minimum distance between two groups is determined from the design rules. These minimum distances are assigned to edges of the constraint graph, the nodes of which are the groups. The direction of the edges corresponds with the position of the groups, e.g. if group 1 is situated at the right of group 2 the corresponding edge is directed from 2 to 1.

The minimal (compacted) geometry can be found by calculating longest paths in the constraint graph. So each group i is placed a distance to the right of group 0, given by the longest path in the constrained graph from node 0 to node i (it is assumed that node 0 is the fixed left side of the compound to be compacted).

The constraint graph method is of complexity n**2, where n is the number of layout elements since each element has to be checked with all other elements for grouping and constraint determination.

Note:

- The results after compaction can be re-edited because before and after compaction we have a symbolic layout with geometric information. When compaction is conversion from symbolic to geometric layout re-editing with the symbolic editor is not possible.

- The first step of the compaction process can be usec as design rule check.

This compaction method may be improved by jog insertion which can be done interactively or automatically. A problem with this method is that only constraints of the "minimal distance" type can be handled. In practice there are also a lot of constraints of type "distance between minimum and maximum". E.g. a via connecting two wires of different layers. These minimax constraints occur also when layouts on higher hierarchical levels have to be compacted. These problems have partially been solved by Mosteller [13], however no algorithms have been published.

The Shear line compaction methods compacts layouts by searching for "compression ridges" which may be clipped out. When fully automatic the method is time consuming [10] due to the problem of "dead alleys" which give rise to backtracking. It may be useful to use a combination of both methods with manual interaction to overcome long response times.

A problem related to compaction is "pitchmatching". It is often necessery that cells pin each other by abutment. If the pitches of the pins on each of the cells differ one would like stretching or compacting cells such that pins abut. It has to be examined if an operation "stretch" can be defined which is in some sense the inverse of compact. A method of pitchmatching has been implemented in the MLLGA system [5]. The topics of compaction, stretching and pitch matching remain subjects for future research.

7. Layout Verification

Since human beings are error prone it is likely that layouts designed with a symbolic editor contain errors. Tools should be available for layout verification. The layout can be verified with respect to:

- design rules
- circuit description
- additional errors.

If all requirements of hierarchical separation have been satisfied design rule check and circuit verification may be done compound cell wise. By verifying the symbolic design rules compound cell wise (see section 5) it is ensured that most physical design rules are satisfied [8]. This verification may be done in an "incremental" way. Design rule check may also be done by the compactor.

For circuit verification it may be necessary to indicate in the database logic equivalence of (sets of) circuit elements, and of (sets of) signals. This and other extensions can be made to the database by including new sections in the definitions of compound cells. There is a relation between circuit verification and the NDL language and database in which the circuit is described. This is a subject for further research.

Additional errors which cannot be found by circuit and design rule verification may occur. Examples are:

- errors on powerline interconnect,
- errors in the network definition, especially those that cannot be found by simulation. Some of these errors may be detected by test pattern generation (e.g. stuck-at faults that cannot be propagated to the output).

All the problems mentioned above concerning layoyt verification have to be examined and solved. We believe that the struture of the symbolic design database is sufficiently flexable to store the required data.

8. References

1. P. Dewilde, J. A. G. Jess, "NELSIS, a co-operative system for large integrated circuit design",

2. h.M.G. van Bokhoven, "Piecewise-Linear Modelling and Analysis", Thesis Eindhoven University of Technology, 1981.

3. S.M.R. Barbacci, "Instruction Set Processor Specification (ISPS): The Notation and its Applications", IEEE Trans. on Computers, Vol C-30, No 1, pp 24-40, January 1981.

4. "The Hardware Description Language HDL", Report number 020-002, 12 February 1982, Texas Instruments, Dallas, Texas.

5. Weste, N. "MULGA - An Interactive Symbolic Layout System for the Design of Integrated Circuits", Bell Systems Technical Journal, Vol 60, No 6, July-Aug. '81, pp 823-857.

6. C.A. Mead and L.A. Conway, "Introduction to VLSI Systems", Addison Wesley, 1980.

7. CIRCUITMASK USER MANUAL Integrated Circuit Design Automation Department, Mullard Southampton.

8. C. Niessen, "The Role of CAD tools in VLSI Design Methodology", ESSCIRC '81 Digest of Technical Papers, pp 75-86, Freiburg, Sept. 22-24, '81.

9. J. D. Williams, "STICKS - A Graphical Compiler for High Level LSI Design", AFIPS Conference Proceedings, Vol 47, 1978, pp 289-295.

10. A. E. Dunlop, "SLIM - The Translation of Symbolic Layout into Mask Data", Proc. 17th Design Automation Conference, pp 595-602, 1980.

11. N. Weste, "Virtual Grid Symbolic Layout", Proc. 19th Design Automation Conference, pp 225-233, 1981.

12. D. F. Bracken and N. J. Mc Calla, "An Interactive Graphics System for Structured Design of Integrated Circuits", Hewlett Packard Journal, June 1981, pp 18-25.

13. R. C. Mosteller, "REST A Leaf Cell Design System", Silicon Structures Project Report 4317, California Institute of Technology, Pasadena, California.

14. K. M. Black and P. K. Hardage, "Advanced Symbolic Artwork Preparation (ASAP)", Hewlett Packard Journal, June 1981, pp 8-10.

15. M. Y. Hsueh, "Symbolic Layout and Compaction of Integrated Circuits", Ph. D. Thesis University of California, Berkeley, UCB/ERL M79/80 Memo 1979.

16. S. B. Akers, J. M. Geyer and D. L. Roberts, "IC Mask Layout with a Single Conductor Layer", Proc. 7th Design Automation Workshop, San Fransisco, pp 7-11, 1970.

17. M. Tucker and L. Scheffer, "A Constrained Design Methodology for VLSI", VLSI DESIGN, pp 60-65, May/June 1982.