The HILL System: A Design Environment for the Hierarchical Specification, Compaction, and Simulation of Integrated Circuit Layouts

Thomas Lengauer*  Kurt Mehlhorn*

Fachbereich 1o, University of the Saarland
6600 Saarbrücken, West Germany

ABSTRACT

We present a CAD system for hierarchical symbolic layout design. Circuit and layout are specified by means of a set of hierarchically defined stick diagrams. A flexible cell concept provides the designer with ample topological freedom to tune previously defined cells to the environment in which they are used. At the same time a rigid maintenance of the hierarchy ensures that the designer does not lose control over the expanded layout while working on it hierarchically. At each level in the design hierarchy three layout modes (composition, placement, graphical) are available that cater to a wide variety of design needs. A switch-level simulator helps test out circuits and a compactor ensures the correctness of the produced mask data. The system guides the design process with extensive consistency checks both on the circuit and the layout, and communicates with the designer through graphics I/O both on the symbolic and mask level. A prototype of the system is running. The version described here is currently being implemented.

1. INTRODUCTION

The HILL (Hierarchical Layout Language) system is currently under development at the University of the Saarland. The development of HILL is one of several research projects that are funded by Deutsche Forschungsgemeinschaft, Grant SFB 124, and are under way at the universities in Kaiserslautern and Saarbrücken, West Germany. The research projects include the development of CAD systems for VLSI circuit development, such as HILL and CADIC as well as work on parallel machine architectures and their software.

HILL aims at supporting symbolic layout design on the stick diagram level. For this purpose the HILL system provides a hierarchical layout specification language in conjunction with a graphics editor, a switch level MOS simulator, and a compactor.

The simulator is based on a simple MOS model that includes no timing. Transistors are modeled as switches that can be in one of three states: 0, 1, and X modeling off, on, and undefined. In [MNN82] three things are proved:

1. The correctness of the simulation w.r.t. the model
2. The relationship between the model and the reality of RC-networks
3. The efficiency of the simulation

The simulator runs in time O(n) on a large class of networks, where n is the number of transistors in the network that change state. Experimental evidence shows that 0.2 msec are spent per transistor on a SIEMENS 7760.

An overview over the simulation model and algorithms used in the HILL simulator is given in [LM83].

Since the simulator simulates the electrical network underlying a symbolic layout it has no access to delay information. Rather, the simulator uses a unit delay assumption. Of course, this assumption is quite problematic and may bear no relation to the actual delays. In particular, networks may pass the simulator which will then not work in reality and vice versa. Recently, a partial solution to this problem was found. In the refined simulation model arbitrary delays are assigned to gates and wires and the behavior of the circuit must be independent of the associated delays. Thus a circuit exhibits a certain I/O-behavior in the new model if this behavior is exhibited no matter what the delays of the gates and wires are. A refined simulation algorithm was shown to be correct with respect to the refined model. The complexity of the refined algorithm is linear in the number of switches of the network. Details can be found in [N].
The compacter is based on the graph-theoretic approach for gridless compaction pursued by [CABBAGE, FLOSS, STICKS]. The algorithms it uses are described in [L82a, L83]. It allows previously unachievable topological flexibility. Specifically it can switch the positions of adjacent circuit structures that lie on the same mask layer, if no design rules are violated during this process, and if doing so reduces the area of the layout. The compaction algorithm is proved to run in \( O(n \log n) \) time, and is expected to perform very well in practice.

A less efficient and more complicated compacter has been implemented as part of [HILL82], a predecessor of the HILL system. Nevertheless, even in HILL82 compaction is quite fast. For example, the adder using the carry chain below consists of \( (126, 270) \) transistors for \((8, 16)\) bit numbers, and is compacted in about \((21.9, 45.5)\) seconds on a SIEMENS 7760.

This paper cannot go into the details of the simulator and the compacter. Here we intend to give an overview of the specification language used in HILL. It is described in detail in [HILL83]. The implementation of HILL83 is under way. A predecessor [HILL82] is running and is being used successfully by us and students. HILL82 has a less flexible cell concept, a less efficient (running time and quality of output) compacter and a less convenient user interface. (Compilation from HILL to PASCAL had to be done by hand.)

Figure 1 gives an overview of the HILL system.

2. OVERVIEW OF THE HILL SPECIFICATION LANGUAGE

HILL is a tool for single chip development. The main focus of HILL is layout generation and verification. HILL aims at supporting the designer who has a comprehensive global image of his circuit. HILL provides a convenient way of describing a layout symbolically either by a HILL program or during an interactive session at the graphics terminal. Even though the layout is specified in a symbolic manner the designer has many means of exerting direct influence on the quality of the resulting mask data.

HILL is a system which combines convenient circuit description with efficiency of the implementation. We aim for efficiency in three respects: human design time, chip area and delay, and computational resources.

HUMAN RESOURCES: In HILL, integrated circuits are described at the level of stick diagrams enhanced by extensive means for structuring a design hierarchically. We have chosen the level of stick diagrams because on the one hand it still allows the designer to express his insights about the topology of the circuit and on the other hand it frees the designer from the tedious and error-prone task of specifying his circuit at the mask level. The stick diagram level has been used successfully in systems like [CABBAGE, STICKS, MULGA].

Even though it certainly is no good practice to specify a whole large-scale circuit with one giant stick diagram, if enhanced by extensive means for hierarchical structuring, especially with a powerful cell concept, stick diagrams become a convenient symbolic representation of even large scale layouts. In the HILL report [HILL83] this thesis is exemplified by a number of examples. We will give a small example below.

Most existing stick diagram systems are totally or almost totally graphics oriented. Graphics is indeed an ideal tool for describing small irregular parts of a layout. The graphics editor in HILL is similar to existing ones and is not described here. It is currently being implemented. Graphics has its limitations when it comes to building up hierarchies because it can support only limited mechanisms for structuring layouts. Essentially, graphics can only support composition and primitive forms of iteration (duplication).

However, graphics cannot support regularity in its full generality. A (large) object is regular if it allows for a short description. Only a universal programming language can support regularity in its full sense. Therefore HILL is designed as a PASCAL extension which interacts gracefully with a graphics editor. In particular, HILL supports recursion, the full power of iteration, and parameter passing. Recursion and iteration are central to (software) algorithm design, and they are already proven to be powerful concepts in hardware design (e.g. [GV82, MC80 Chapter 8]). Recursion corresponds to trees, iteration to array-like structures, and programming in general allows to specify general regular networks.

A good example, although too long to be included here, is the multiplier described in [LM83]. This multiplier is based on the divide-and-conquer method using three multiplications of numbers of half length. The layout given there is regular, however, the regularity can certainly not be captured within a pure graphics system. Rather, powerful descriptive tools, such as recursion, iteration, and parameter passing are needed to capture the regularity.
Chip area and delay: Experience with existing systems [CABBAGE, STICKS, MULGA, HILL82] suggests that automatic compaction can yield small layouts which come close to hand-compacted layouts. The stick diagram level is close enough to the silicon to allow the designer to incorporate performance aspects into his specification, and the compacter supports chip performance with his knowledge of the fabrication process. Finally, the ease of circuit description and the feasibility of the algorithms used in the system allow the designer to try several approaches to his circuit and select the one he likes best.

Computational resources: Most existing systems have definite shortcomings in this respect. In most cases, only sketchy theoretical analyses of the running time are given, often algorithmic constraints enter the system only tacitly. Computational experience with the systems suggests that the running time is highly non-linear. For example, it is reported that CABBAGE takes time $O(n^{1.2})$ to compact a circuit with $n$ transistors and wires. In HILL82 we improved upon this and implemented an $O(n \log n)$ algorithm. A more elegant and flexible $O(n \log n)$ algorithm is described in [L82a, L83] and will be used in HILL83.

However, even this algorithm will not do for large scale circuits, because of its $O(n)$ space requirement. The solution to this problem is to compact hierarchically. See [L82b].

The main structuring device in HILL is that of a "cell". A cell is the specification of a subcircuit of the chip to be designed. This subcircuit will in general function as a module in the chip that communicates with its surroundings through relatively few connections (pins) and performs a specific subfunction of the chip function. It is rectangular in shape with the pins arranged on its boundary. It is very much reminiscent of a function in a sequential programming language like PASCAL. As procedure parameters form the (up to side effects exclusive) interface between the function and its call environment, so the pins of a cell facilitate the interface between the cell and the circuitry around its location of placement on the chip. The only way to contact to a cell is through one of its pins. Like procedures cells can be compiled separately and defined externally. For "instantiating" a cell only a description of its rectangular boundary, its so-called "template" has to be given. The template contains no electrical or topological information about any of the inner workings of the cell. However, it contains both electrical and topological information about the cell boundary. Pins can be related to each other electrically in the template, and they have to be "placed" in order to specify the order in which they appear around the cell boundary.

In addition to specifying in which order the pins occur on each side of the cell, HILL also requires to specify the relative positions of pins on opposite sides of a cell. This is done by giving two partial orders, one for the x-direction and one for the y-direction. The system checks whether the symbolic layout given for a cell can be distorted such that it conforms to any linear extension of the partial orders. An example of this is given in the next section. The distortion mechanism implemented in HILL83 is one-dimensional for reasons of efficiency, and therefore one of the partial orders mentioned above has to be a linear order. Even with this limitation HILL cells are quite flexible and hence adapt easily to changes in the environment of their placement. On the other hand, the cell template captures enough information about the topology of a cell for the designer to keep track of the relative placement of his circuit components while stepping through the design hierarchy. We believe that this is a must if the designer wants to keep control of his layout while working on it hierarchically.

3. THE HILL SPECIFICATION OF A RECURSIVE CARRY LOOK-AHEAD ADDER

We now give an example that highlights many of the features of the HILL specification language. We design the carry chain for an adder with complete carry look-ahead according to the construction given in [GV82] which is particularly well suited for NMOS technologies. The fabrication process for which we design the circuit is described in [MC80]. This circuit accepts the two numbers to be added not in binary notation $a = a_{n-1} \ldots a_0$, $b = b_{n-1} \ldots b_0$ but encoded as two strings of carry generate bits $g = g_{n-1} \ldots g_0$ and carry propagate bits $p = p_{n-1} \ldots p_0$. Here $g_i = a_i \land b_i$, $p_i = a_i \lor b_i$, $i = 0, \ldots, n-1$

The bit pairs $(g_i, p_i)$, $i = 0, \ldots, n-1$ are then used to compute the carry bits $c_i$ out of each bit position $i = 0, \ldots, n-1$. Here an associative operation $\circ$ on bit pairs is used that is defined as follows:

$$(g, p) \circ (g', p') = (gg', pp')$$

The carry chain computes the carry bit $c_i = a_i b_i \lor a_i c_{i-1} \lor b_i c_{i-1} \lor i_0, \ldots, n-1$ ($c_{-1} = 0$) out of each bit position recursively.
using the o-operation. Specifically, it computes

\( (G_1, P_1) = (g_1, P_1) o (g_{i-1}, P_{i-1}) o \ldots o (g_0, P_0) \) i=0, \ldots, n-1

Note that \( G_1 = c_1 \).

The circuit for the carry chain is defined recursively. Its bottom level element is the basic cell, depicted in Figure 2.

[Figure 2]

Note that in0, in1, and out denote pairs of pins. The cell performs the following function.

\[
\text{out} = \text{in1} \odot \text{in0} \\
\text{cout} = \text{cin} \\
\text{clout} = \text{in0.g} \lor (\text{cin} \land \text{in0.p})
\]

Thus, if we set cin=0 the basic cell computes the carry bits of the sum of two 2-bit numbers and provides them at the pins clout, and out.g.

For \( n=2^{i} \), \( i>1 \), the carry chain \( \text{chain}(i) \) is built up out of two carry chains \( \text{chain}(i-1) \) for \( n/2 \)-bit numbers according to Figure 3.

[Figure 3]

It is shown in [GV82] that this construction indeed computes the carry bit \( c_i \) and provides it at the pin data[i+1].out for \( i=n-1 \) and over.g for \( i=n-1 \).

Actually the above construction does not lead to the fastest carry chain possible. As [GV82] point out, inverting the output signals at each stage in the hierarchy allows to omit some gates which makes the circuit faster and smaller. We chose to exemplify the HILL system with this suboptimal layout, because we do not want to obscure the issues at hand through an overly complex example.

We will now describe the cells that make up our circuit in HILL. We need three types of cells. The first cell has the name op and implements the o-operation. Its definition in HILL looks as follows:

```hills
aggregate cbits = record g,p:poly end; 
ps = record vd:metal sig=vdd; 
        gd:metal sig=gnd 
        end; 

cell op(z:int); 
temp pins out,inb,inp:cbits; 
goutl,poutl:metal; 
l,rp:ps; 
order implicit:ver; 
top:out; 
bottom:in; 
rightr.vd at 2,inp,r.gd,inp.g; 
left:goutl,l.vd,l.gd at 4,poutl
```

The above text only describes the rectangular boundary of the cell. Its interior, i.e., its layout is input with the interactive graphics editor. This results in Figure 4.

[Figure 4]

It can be easily checked that this cell realizes the following function:

\[
goutl = \text{out.g} = \text{inb.g} \lor (\text{inp.p} \land \text{inp.g}) \\
\text{out.p} = \text{inp.p} \land \text{inp.p}
\]

The cell has an integer parameter \( z \). This parameter determines the strength of the transistors in the cell, which has to be different in different levels of the recursive hierarchy of the chip specification (see [GV82]).

The specification of cell op is preceded by an aggregate-statement that forms groups of pins. Here corresponding carry generate and carry propagate lines are paired up, and a power supply bus is defined.

We now describe the specification of cell op. The header containing the numerical parameter is followed by a so-called template enclosed in the parentheses temp and pmct. The template gives information about the rectangular boundary of the cell op. It is followed by a definition of the layout of the interior of the cell. For graphically defined cells the symbol external is substituted for this part of the specification.

The template lists all pins of the cell in the pins section. Each pin has a sort that identifies the layers it exists upon and can be accompanied by some specification of the electrical signal it carries, e.g., gnd or vdd. Each pin also has a location on the boundary of the cell. Thus each pin is assigned a side of the template and the ordering of the pins on each side is determined by enumeration. Pins on opposite sides can also be related to each other w.r.t. their positions. This can happen in two ways. In the above example the pins on the vertical sides (the y-pins) are paired up implicitly by assigning coordinate values to them. These values are assigned according to the sequence of enumeration except for explicit changes (e.g. r.vd is assigned the value 2). Thus pins l.vd and r.vd as well as pins l.gd and r.gd are paired, i.e., they receive the same y-coordinate in each symbolic layout of the cell. The locations of the x-pins of the cell op are not constrained w.r.t. each other. This reflects the fact that out.g
and out.p are free to slide along the top side to the position required by the specific surroundings in which the cell is placed. Only limited use is made of this freedom in our example. When using cell op we could have always aligned out.p with in.p and out.g with in.g. However, the freedom is carried out to the boundary of the cell containing the carry chain, where it may be used in a placement of the whole carry chain circuit higher up in a chip specification hierarchy.

This flexibility is a critical part of the cell concept underlying HILL. It allows for detailed information about the topology of the cell layout as we step through the chip specification hierarchy.

We use cell op to define cell basic. Cell basic can also (and should) be defined graphically, but we will specify it with a HILL program to exemplify the placement mode provided in HILL.

cell basic (z: int);
temp pins out, in0, in1: cbits;
cin, cOout, clout: poly;
lb, rb: ps;
order implicit: ver;
top: out, cin;
bottom: in0, clout, in1, cOout;
left: lb;
right: rb;
constraints begin out.p leftof clout;
end
in above cOout
end

pmet layout extend order top: out, cin after 3;
bottom: in0, clout, in1, cOout;

var ystretch: ylinelist;
component r1, r2: op(z) extend order top: out; bottom: in;

begin makegrid (0,0,2,2,0,2,0,1,1,4);
ystretch := ally/(0.4/) conc ally/(6.7/);
place r1 on (allx/(0.3/),ystretch);
place r2 on (allx/(7.10/),ystretch);
route in1.g to r1.inb.g;
route in0.g to r2.inb.g;
route in0.p to r2.inb.p;
route r1.r.vd to r2.r.vd;
route r1.r.gd to r2.r.gd;
route r2.r.vd to rb.vd;
route r2.r.gd to rb.gd;
route cOout to cin;
route clout go up layer metal
plated. Only go right to r2.out1;
route r1.inr.g go right 1 layer metal
go down 2 go right
to xline(in0.g);
route r1.inr.p go right 1 go down 2
to xline(in0.g);
route r2.inr.p go right 1 layer metal
go down 2 go right
to xline(r2.out1);
route r2.inr.g to xline(cOout);

We now describe, how the HILL system turns the above specification into a cell as depicted in Figure 5

[Figure 5]

Let us begin with the template. Again the y-pins are matched up against each other, but this time the x-pins on the top and bottom sides are also partially constrained. As can be seen from the figure cin must have the same x-coordinate as cOout and out.p cannot come to lie to the right of clout. This is stated in the constraints section of the template. If this specification were not given the processing of the cell basic would terminate with an error, because the system checks whether all the freedom among pins allowed by the template can be realized by the cell layout. This is important since this freedom may be exploited higher up in the chip hierarchy. In this way we assure that the layouts specified hierarchically really resemble the planar chip layout after all cells are expanded. Thus the designer does not get trapped into forming faulty images of what his layout looks like after expansion.

The constraints on the pins allow several layouts of the cell interior. These layouts can be transformed into each other by a process called non-uniform stretching. In order to keep this transformation feasible HILL restricts cells to allowing this freedom of pin placement only in one dimension. In the other dimension the pins have to be ordered linearly; no freedom of relative placement exists.

The layout specification following the template gives an example layout for the cell. This layout is assembled on a square grid. Here pins always come to lie on grid points and wires always come to lie on grid lines. First the pins are placed on grid points by the same mechanism that is used for matching up pins on opposite side of a template. This happens in the section following the keyword extend order.

After processing this section the system provides a grid for placement, as in Figure 7

[Figure 7]

Now the components to be placed on the grid are declared. They are two rectangles r1 and r2 of type op(z), with their pins matched as given by the following extend order section. (A consistency check is made here). Then the placement begins. First additional grid lines are created to expand the grid to the required size. (This can
also happen dynamically during the placement.) This yields an enlarged grid, as in Figure 8.

[Figure 8]

Then rectangles r1 and r2 are placed by specifying the gridlines, that their templates are mapped onto, for x- and y-direction. These gridlines may be adjacent (as in the x-direction for this example), or they may be non-adjacent thus resulting in cell stretching before placement (as in the y-direction of this example). A quite elaborate routing statement allows to route wires flexibly and to create contacts between layers implicitly where needed.

Note, that in r1 pin gout1 is not connected to anything and in r2 pins out.g and out.p are not connected to anything. HILL has a pin-forgetting mechanism that can be evoked explicitly or, as in this case, automatically. It causes pins of cells to be suppressed and it is also possible to reduce the layout structure inside the cell by removing all components that only served to compute the signals on pins that we suppressed. In this case this causes the interior layout of cells r1 and r2 to differ, even though r1 and r2 are of the same type. Figure 6 shows the layout of cell basic with r1 and r2 expanded.

[Figure 6]

Finally we have to recursively put the cells of type basic together. This is done differently from the specification of cell basic. We use composition mode here. In composition mode no grid is used, but existing cells are copied and plugged together in x-direction using the hor-operator or in y-direction using the ver-operator. For this operation to succeed the corresponding sides of the cells have to match.

The recursive specification looks as follows:

cell chain (n: int);
  temp pins over: cbits;
  cin: poly;
  data: array [0..2^n-1] of record in: cbits; out: poly end
  ps: array [1..n] of ps;
  order implicit: ver;
  top: over, cin;
  bottom: data reverse;
  left: ps;
  right: ps;
constraints
begin
  above data[0].out;
  over.p leeftof data[2^n1].outend

pmet composed
begin if n=2 then basic(n) ver
  (chain(n-1) hor chain(n-1))
else
  chain := basic(n)
end

Figure 9 gives an expanded sticks layout for the 8-bit carry chain and Figure 10 shows the compacted mask data.

[Figure 9]

[Figure 10].

4. ADDITIONAL FEATURES OF THE HILL SPECIFICATION LANGUAGE

The HILL specification language entails a number of features that have not been used in the above example. These will be described in the following.

4.1 MAINTENANCE OF ELECTRICAL INFORMATION

HILL has the concept of an electrical signal. Electrical signals are associated with points in the circuit and grouped into classes by the link-statement to form electrical networks. The designer can build up completely or in part the electrical network he wants to realize with a layout. The HILL system encompasses tools for checking whether the network realized by the layout is the same as the intended network. HILL is also designed to be augmented with automatic layout routines that realize an intended electrical network with an automatically generated layout. Only a PLA generator exists at this point. Global signal names allow to implement naming conventions for signals that pervade the whole specification hierarchy (like power supply and clock signals).

4.2 CONTROL OVER THE PROPORTIONS OF THE MASK LAYOUT

The cell concept used in HILL ensures complete control over the topology of the symbolic layout, even when specifying a chip hierarchically. However, control over the mask layout is only indirect, since an automatic compacter determines the final proportions of the mask layout, and the sizes of cells in the mask layout are not known when specifying a chip hierarchically in a top-down fashion. However, floor planning and performance optimization of a layout require direct influence on a subset of the mask data. For this purpose HILL provides the keep statement. The keep statement formulates explicit constraints that have to be obeyed during compaction. The existence of the keep statement also allows to include pre-compacted cells into
a symbolic layout specification. This is the basis for compatibility with cell libraries generated with other layout systems and for hierarchical compaction to be implemented in the future.

4.3 TOP-DOWN DESIGN

HILL supports top-down design of chips in several ways. One of them has been mentioned in 4.2. Another is the strict separation of the template and the interior layout of a cell. This allows to use subcells in a hierarchical specification, even if they have not been designed yet. The detailed information given in a template serves as a guide both to the user of the subcell and to the designer who has to fill in the layout of the subcell independently. A fill statement is used in HILL to fill the empty template with a layout that makes up the interior of the cell. However, for floor planning not all templates have to be filled for the chip to be processed.

4.4 LAYOUT MODES

For each cell the designer can choose one of three modes (composition, placement, graphical) to design his cell. The graphical mode will generally be used if no algorithmic concepts enter the design. How should the designer decide between placement mode and composition mode? As a rule placement mode is more verbose than composition mode and thus composition mode will be preferred for simple cell designs that the designer feels sure about. However, composition mode allows only rudimentary error handling since no run time names are associated with subcells and no graphic snapshots can be given. On the other hand placement mode provides the designer with extensive debugging tools. Each component of the cell has a unique run time name and snapshots of partially assembled layouts are possible at any time. These advantages may prompt the designer to use placement mode even for array-like designs that are eligible to be specified with composition mode.

5. RELATION TO OTHER WORK

For the purpose of relating HILL to other work we give a quick taxonomy of existing layout systems:

1. Layout Mode

a) Operational Layout: The layout topology resp. mask geometry is here specified explicitly by referring to a planar universe, i.e., a grid or some other coordinate system [CABBAGE, STICKS, POOH, Electric, Astra, LUCIFER, SLIM, MULGA, VIVID]

b) Constrained Layout: Only local constraints are given on locations of layout components. This set of constraints has to be resolved into a consistent layout [i, ALI2].

c) Functional Layout: The layout is constructed essentially by abutting cells. [Sticks&Stones, LUBRICK, POOH].

The graphical and placement mode in HILL are operational, and the composition mode is functional. For graphical systems the operational mode is preferable because of its incremental nature and easy error handling capabilities. For textual systems the constrained and functional mode are more elegant. However, the constrained mode has serious drawbacks when it comes to error handling.

Some systems provide different layout mechanisms for different levels in the design hierarchy [LUCIFER/LUBRICK,ASTRA, MULGA/MULGA2]. In HILL all layout mechanisms are available at all levels in the design hierarchy.

2. Level of Description

a) Geometric Systems: Into this group fall many systems that are currently termed sticks systems. In fact, they handle truly geometric i.e., distance information and only present the designer with some symbolic notation. In terms of such systems compaction means the transformation of one geometric layout to another small layout that is correct w.r.t. the design rules. [CABBAGE, STICKS, POOH, Electric, ALI2, Sticks&Stones, LUCIFER]

b) Topological Systems: Here the layout is given in a truly topological fashion, i.e., no concept of distance is inferred, not even for provisional purposes. This point is minor if a system is totally graphics based [ASTRA], but it becomes essential when the layout system also allows for textual input. Then either constrained layout mode is used [ALI2]. Compaction in this context means resolution of the constraint system to yield a small layout. Or the circuit is laid out on a virtual grid [MULGA, VIVID, HILL]. Then compaction is a mapping from the topological into the geometric domain. In MULGA the virtual grid survives until after compaction, in HILL it is removed before. This enables HILL to do a better job of compaction, however, the compaction algorithms of HILL are more sophisticated than those of MULGA.

3. Maintenance of the Design Hierarchy

a) Leaf Cell Design Only: [CABBAGE, SLIM]
b) General Cell Concept: Cells have no specified shape. [I, LUCIFER, Electric].

c) Rectangular Cell Concept: Cells must not overlap; pins are located on the cell boundary [MULGA, ASTRA, ALI2]

d) Non-rectangular Box Concept: Cells can have other than rectangular shapes [STICKS, LEBRICK]

e) Rectangular Cells, Moving Pins: This is the concept of HILL. One might be tempted to put ALI2 in this class as well, but in ALI2 pins can only move by compaction, and not by explicit topological transformations.

6. CONCLUSION

The HILL specification language is a powerful tool for describing planar circuit layouts. It has a cell concept that is flexible, yet keeps track of all the relevant topological information that has to be transferred across cell boundaries. This enables the designer to go through with a truly hierarchical design. HILL allows to choose between three modes of layout description that cater to different design styles and tasks. It provides the full mechanism of a high level language to allow the algorithmic description of layouts that are regular, yet non-trivial. Furthermore it separates electrical, topological, and geometrical information, which results in excellent interfacing of the different design phases (circuit, layout).

Supplemented with an efficient switch level simulator and a powerful compactor we believe that the HILL system serves well for the hierarchical specification of symbolic layouts.

7. ACKNOWLEDGEMENTS

The HILL prototype system was implemented within a student project course. Major parts of the code were written by J. Doenhart, H. Lehser, S. Nähler, M. Nowak, H.G. Osthof, M. Porzberg, C. Storb, and P. Vogelgesang.

8. REFERENCES

[ALI2]

[Astra]

[Cabbage]

[Electric]

[Floss]

[GV82]

[HILL82]

[HILL83]

(1)

[L82a]

[L82b]

[L83]

[Lm83]
T. Lengauer, K. Mehlhorn, "VLSI Complexity, Efficient VLSI Algorithms, and the HILL

[LUBRICK]

[LUCIFER]

[MC80]

[MNN82]

[MULGA]

[MULGA2]

[N]

[PASCAL]

[POOH]

[SLIM]

[STICKS]

[Sticks&Stones]

[VIVID]