RISCE — A Reduced Instruction Set Circuit Extractor for Hierarchical VLSI Layout Verification

Volker Henkel, Ulrich Golze

Institut für Theoretische Informatik
Technische Universität Braunschweig
D–3300 Braunschweig, West Germany

Abstract

We present a circuit extractor preserving the hierarchical layout structure isomorphically. As opposed to existing extractors, our approach permits all cell overlaps which are electrically meaningful. New mask operations based on stretched geometries handle topologically open and closed areas. A reduced set of model independent instructions is introduced for device recognition. An existing implementation is discussed.

1 Introduction

A hierarchical netlist extracted from a full custom layout is called *isomorphic* (to the layout) if the hierarchical layout structure is preserved. An isomorphic hierarchy simplifies subsequent hierarchical netlist comparisons and mixed mode simulations significantly. Overlaps between cells and subcells are called *interactions*. Interactions are helpful for designers, e.g., when wiring inner terminals of subcells. For extractors producing isomorphic netlists, however, they are difficult because they may create or modify or remove devices as well as connections [1]. (Imagine a diffusion wire in a subcell crossing a poly wire in a cell.) Interactions can *change* the electrical meaning (*semantic*) of subcells already extracted. If uncompensatable [2] in the netlist, the change should be forbidden. Therefore, most systems simply restrict interactions to touches or small overlaps with subcell shapes [3,4,5]. Layer specific protection frames are used in [6,7,8] to define regions of forbidden interactions. These systems either restrict the region or the type of possible interactions considerably, or they require large and time consuming protection frames.

An *abstract* is a simplification of a cell reduced to data that is only required for subsequent processing of calling cells. Technology rules guide the derivation of the abstract from the cell [7]. During extraction, subcell abstracts *replace* the subcells. They consist of *interface geometries* (for connections to subcells) and other geometries. Of course, larger abstracts (with more interface nodes in the corresponding circuit) would allow more interactions, but would increase the extraction time as well. Therefore, it is inefficient to define protection frames by abstracts.

How can we detect forbidden interactions without the concept of protection frames? As a first approach, a new mask semantic was used by the MAGIC system [2]. Here gates are separate elements, and thus *all* poly and diffusion overlaps are illegal and can be detected by a design rule check. But for usual mask oriented layouts or for more detailed parameter extractions, we need additional interaction violation checks. They are realized by our so called *interaction rule check (IRC)*. On the layout level, its interaction rules will verify the equality of the flattened extracted netlist and the netlist extracted from the flattened layout.

2 Overview

For each cell, RISCE performs two independent tasks as shown in Figure 1: First, the netlist is extracted ignoring all interactions. Secondly, the IRC verifies this netlist to mark all points of the layout where the extraction was incorrect. Both tasks are completely controlled by a technology description specified by the user.

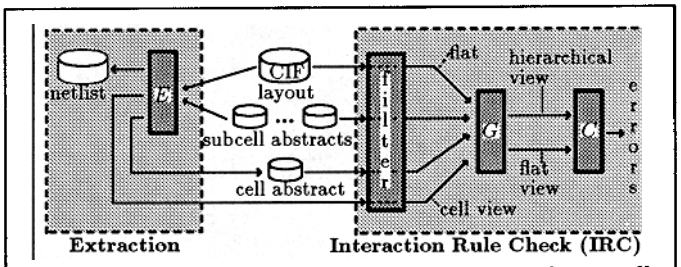


Figure 1: Data flow for extraction and IRC of one cell. Extraction rules (**E**) transform cell geometries and subcell abstracts into a new cell abstract. Within regions of interaction, generation rules (**G**) generate the flat and hierarchical views to be compared for errors (**C**).

To identify forbidden interactions, geometries represent the full semantic (even for detailed features). This is achieved by a new technology description without built–in device models based on several new concepts:

- The usual mask operations combining areas are extended to process even lines and borders of geometries. They support geometries of topologically *open* and *closed* sets.
- Beyond geometry processing, relations between geometries are derived simultaneously.
- These basic operations are compiled from *extraction rules* of the model independent technology description. Resulting geometries of the *cell view* always represent the semantic of the cell just extracted.
- We will model more accurate parasitics (e.g. resistances) by *splitting* geometries.

• With the exception of formal interface nodes which are determined during the abstract generation, abstracts are always extracted in the calling cells. This local layout flattening permits a context dependent extraction of subcells in critical regions.

• Mask operations generate and compare *flat* and *hierarchical views* of semantic within small filtered layout regions.

This paper first introduces the basic operations of the reduced instruction set. An example of a technology description for flat extraction will show the power of these operations. We will extend the description to hierarchical extraction, and finally, we give details of the interaction rule check (IRC). We conclude with first results on an implementation.

3 The Instruction Set

The reduced instruction set consists of mask operations, border operations and semantic operations.

3.1 Mask and Border Operations

Usual mask operations based on regular closed sets [9] are less sufficient for extraction as they do not handle lines. But touching geometries as well as geometry borders can strongly influence the electrical semantic(e.g., a guard ring touching a well). Hence, we once overstretch all given layout geometries by a small constant. The *usual* mask operations now generate over- and under*stretched* sets uniquely representing topologically closed and open sets, respectively. Figure 2 illustrates how lines are represented as small stripes.

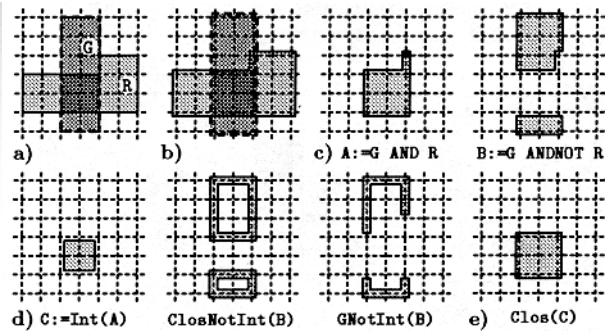


Figure 2: Mask operations: a) Given layers `G` and `R` before and b) after overstretching. c) Results of usual regularized mask operations simulating open and closed parts. d) Additional border operations. e) `Clos(Int(A))` yields the usual regular closure of `A`. With the exception of a), all geometries are over- or understretched.

The usual mask operations `AND`, `OR`, `ANDNOT` and `XOR` can 'simulate' the unregularized set theoretical operations for intersection, union, difference and comparison. If e.g. two closed geometries touch each other, their intersection is nonempty. In our 'simulation' both geometries are overstretched; they overlap each other and their regular closed intersection is no longer empty.

In addition, there are *border operations* `Int`(L) and `Clos`(L) simulating the topological interior and closure of a layer L. Moreover, geometries of a layer are *nonoverlapping* except for border operations which yield overlaps of streched geometries as shown in Figure 3.

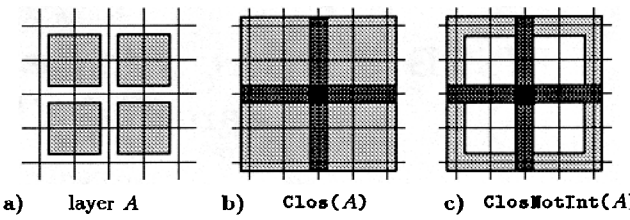


Figure 3: Overlaps with border operations: a) Given layer L with four open geometries. b) Closure of L with overlaps at geometry 'borders'. c) There are further border operations. Notice that '`Clos`(L) `ANDNOT Int`(L)' would yield a nonoverlapping layer.

3.2 Semantic Operations

In order to deduce semantic relations between geometries, each geometry refers to one *object*. All information required for the netlist are stored in objects and references between objects. The structure of objects is discussed in chapter 4; they are composed of *attributes* for numerical values and of *dependencies* for sets of references. Three basic methods process these objects:

• *Connectivity* for one layer as well as for two layers. The operation

'`TYPE`(t) L'

applied to one layer L creates objects (of type t); pathwise connected geometries always refer to the same object (see 'region numbering' in [10], but also some special cases in Figure 4). With the operation

'`CONNECT` $L1$ $L2$'

applied to two layers $L1$ and $L2$, overlapping geometries refer to the same object.

• *Measurement* of geometries for one layer. Measuring results (e.g. area, perimeter or border length) are accumulated in attributes of the corresponding object. Measurements are performed by

'`AREA`(a) L', '`PERIM`(a) L', '`BORDER`(a) L'

for attribute a on layer L.

• Deriving *relations* for two layers. Similarly to '`CONNECT`'

'`RELATE`(d) $L1$ $L2$'

operates on overlapping geometries with one object referring to a second one. Such relations are 'accumulated' as reference sets in the dependency d of the object.

Geometries of a newly created layer do not refer to objects as long as no `TYPE`-operation is performed on this layer. As a special feature in border operations, the objects of input geometries are passed-through to corresponding output geometries in such a way that regions with overlaps of output geometries may refer to several objects.

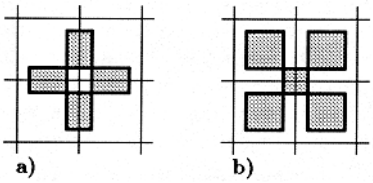


Figure 4: Special cases with connectivity on one layer: a) These four lines are unconnected. b) Four open boxes connected with the central point.

As each object represents the equivalence class of all pathwise connected objects, connectivity has to merge objects. In this case, corresponding attributes are added and corresponding dependencies are united.

4 The Model Independent Technology Description

We now introduce the technology description for a more convenient way of programming the RISCE. Hierarchical features are given in chapter 5. The model independency [11] is achieved in two steps. First, the instructions of chapter 3 transform geometries into objects and references. Then, these data structures are collected into devices. Both steps are specified in structural and operational parts of our technology description:

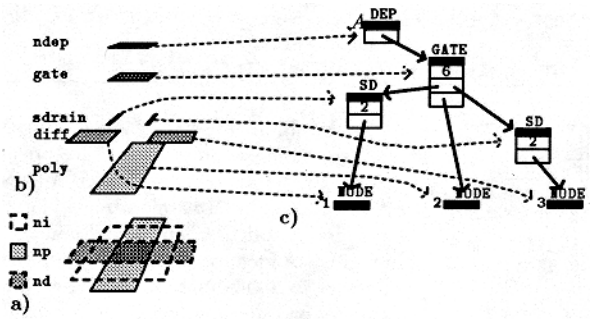
- A 'semantic data model' specifies the object structures with user defined *types*. Objects have attributes and dependencies to objects of other types. Cycles between types are forbidden. Device models are compounded by several types. Figure 5 shows with the help of the declaration of a simple MOS–model, how references should 'connect' geometries and objects.

- *Extraction rules* are extended assignments with an expression over layers, mask and border operations. As usual, they create new layers. Furthermore, they specify the object generation in such a way that for every newly created layer, the rule determines the object type, initiates the deduction of layer connectivity and allows measurements. Interlayer connectivity is formed between the newly created layer and all layers marked by an optional operator '≡' as a high priority prefix. Analogously, an operator '∼xyz' forms a relation for the dependency **xyz** (see Figure 6). Extraction rules are compiled into operations of the instruction set. A '≡' operator, for example, yields a **CONNECT**–operation, and a '∼xyz'–operator yields a **RELATE(xyz)**–operation.

- For the second step, device structures are specified with their parameters and pins.

- The output rules are object oriented. For a given object, all attributes of its (indirectly) referenced objects are available for computing the device parameters.

For a more detailed extraction, we will now extend the simple example of Figure 5 and Figure 6. We model resistances as *star nodes* [2] with **NODE**–objects containing attributes. In the second step, these attributes are used to compute a resistance parameter for each node. Special *cuts* are used as devices for interconnections of nodes:




```
TYPE NODE :=();
TYPE SD   :=( channel_width,
                    NODE: sd );
TYPE GATE :=( area, SD: sd_pin,
                    NODE: g_pin );
TYPE DEP  :=( GATE: g );

MOS 1 2 3 ndep W=2 L=3
```

Figure 5: Example for a MOS–model. a) On the given layers implant, poly and diffusion, a transistor is formed. **b)** Layers generated newly, their geometries refer to objects. **c)** Objects extracted to compound the transistor. The **GATE**–object consists of an attribute for the gate area and of two dependencies. **sdrain**–geometries have the length '**2**', measured to model the **channel_width** of the transistor. **d)** Type declarations in the technology description to support the objects in c). **e)** The netlist device collected from c).

```
TYPE NODE  := ( area, perimeter );
TYPE CUT   := ( NODE: toNode );
NODE poly  ( area:=AREA, perimeter:=PERIMETER ) np;
```

On larger nets, such approximations are of course unsatisfactory. Without going into details, we outline the *geometry splitting* for a more accurate resistance approximation. Created by algorithms from [12] or simply specified by the designer, a layer **split** contains splitting lines at **poly** branches. This layer is used to prevent a complete layer connectivity for poly in the following rules:

```
(1)  NODE poly   := np;
(2)  GATE gate   ( perimeter:=PERIMETER )
                 := nd AND ∼g_pin poly;

(3)  NODE diff   := nd ANDNOT gate;
(4)  SD   sdrain (channel_width:=PERIMETER )
                 := ∼sd Clos( diff ) AND ∼sd_pin gate;
```

a)

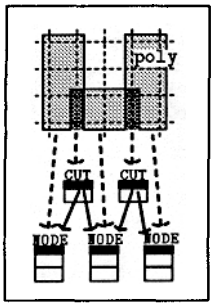
```
(1')  poly   := np;              TYPE(  NODE )          poly;
(2')  gate   := nd AND poly;     TYPE(  GATE )          gate;
                                 PERI(  perimeter )     gate;
                                 RELATE( g_pin )  gate  poly;
(3')  diff   := nd ANDNOT gate;  TYPE(  NODE )          diff;
(4')  xxx    := Clos(diff);
      sdrain := xxx AND gate;    TYPE(  SD )            sdrain;
                                 PERI(  channel_width ) sdrain;
                                 RELATE( sd )     sdrain xxx;
                                 RELATE( sd_pin )  gate  sdrain;
```

b)

Figure 6: Extraction rules. a) Rules for the MOS–model of Figure 5. Rule (1) enforces layer connectivity of **poly** by generating **NODE**–objects. To model the gate pin, the operator '∼g_s' in rule (2) forms a relation by references from **GATE**–objects to **NODE**–objects. Measurements are performed with **PERI**. The closure operation in (4) overlaps **diff**–geometries with **gate**–geometries in order to model source and drain pins.

b) Operations of the instruction set compiled from a). On the left hand, there are mask and border operations. On the right hand, there are semantic operations. The 'assignment' in line (1') creates a nonoverlapping layer. All semantic operations process the result layer of the rule. In (4') the direction of the relation follows from the semantic data model in Figure 5d: Notice that '∼**sd**'–references start at SD–objects of **sdrain**–geometries, whereas '∼**sd_pin**'–references refer to SD–objects.

```
NODE poly  ( area:=AREA, perimeter:=PERIMETER )
           := np ANDNOT split;
CUT  cpoly := ~toNode Clos(poly) AND split;
```



In the figure to the left, two splitting lines subdivide a poly wire. The unconnected poly–fragments will refer to separate objects. The Clos operation will overlap them at splitting lines (see Figure 3b). In regions of such overlaps, the CUT–objects will refer to NODE–objects. In this way, resistances on one layer are interconnected. Such splittings can even be applied to capacitances. With the help of our border operations, we can consider detailed side wall effects. For further details on measurements, coupling capacitances and algorithms see [13]. Netlists containing star nodes and cuts are easily convertable into usual SPICE–netlists.

5 The Hierarchical Extraction

Hierarchical extraction requires additional mechanisms for creating and using abstracts. It operates in a bottom–up manner. For *incremental* verification, cells are processed independently of their calling context. In this chapter, we ignore interactions. Flat abstracts replacing subcells reduce the hierarchical extraction to a flat one. Given all subcell abstracts of a cell and, moreover, given all cell geometries on the actual level of the hierarchy, we create a new cell abstract independently of the cell context. A opposed to layers of the given layout being always topologically closed, abstracts contain layers of stretched geometries. The abstract layers are defined explicitly in the technology description. Two main problems are to be solved for every flat cell extraction:

- Deriving the formal circuit interface: After extraction, all abstract layers are scanned to collect the set of objects to which the geometries refer. The circuit interface nodes are determined by this set. In the technology description, this is controlled by listing the names of abstract layers.
- Deriving actual subcircuit interfaces: Before extraction, the layer abc[subcellabstract] corresponding to the abstract layer abc contains the (interface) geometries of all subcells. Such subcell abstract layers are not nonoverlapping if interface geometries of different subcells have overlaps. A built–in procedure attaches objects to [subcellabstract]–layers. A priori, their geometries refer to different objects at overlaps. As specified in the technology description, connectivity has to merge these objects in order to interconnect circuit pins.

With extensions of capabilities introduced first in [7] (see Figure 7), we can select interconnected geometries. This simplifies the resistance extraction in a hierarchy because the abstract could lift, for example, short poly wires to the calling cell where they are extracted. Therefore, the usual substitution of formal interface nodes into actual ones is extended even to the substitution of node *parameters*, e.g., the resistance parameter of a star node.

Each cell is only extracted outside of its interface geometries. Interface geometries are extracted in calling cells if the context of the cell is known. Therefore, a *root cell call* must be extracted yielding the parasitics of the root interface. Neither an abstract nor a circuit declaration is generated in this case.

```
(0) ABSTRACT   poly_pins, diff_pins, metal_pins;

(1) NODE poly  := np OR ≡ poly_pins[SUBCELLABSTRACT];
    GATE gate  ( perimeter:=PERIMETER) := nd AND ~g_pin poly;
    NODE diff  := ( nd OR ≡ diff_pins[SUBCELLABSTRACT] )
                                              ANDNOT gate;
    NODE metal := nm OR ≡ metal_pins[SUBCELLABSTRACT];
    NODE mc    := ≡ metal AND nc;
    NODE mcp   := ≡ mc AND ≡ poly;
    NODE mcd   := ≡ mc AND ≡ diff;

(2) SET L      := LABEL(diff) or LABEL(metal) or LABEL(poly);
(3) NODE poly_pins  := ≡ SELECT(poly,L);
    NODE diff_pins  := ≡ SELECT(diff,L);
(4) NODE metal_pins := ≡ SELECT(metal,L) AND nms;
```

Figure 7: Declaration and use of abstracts. Layers np, nm, nd, and nc are given layers of the layout. In addition, the declarations of Figure 5 are assumed. Declaration (0) lists the names of abstract layers.
Initially, interface geometries in poly_pins[SUBCELLABSTRACT] refer to NODE–objects. Rule (1) constructs layer connectivity with all 'visible' poly–geometries. Notice that np contains only geometries of the cell itself.
In two steps a simple abstract is defined with labeled geometries. First, to select 'nodes' labeled on several layers, a set L of objects is formed in (2). Geometries referring to these objects are selected. Then, rule (3) generates the new cell abstract layer with objects reflecting formal interface nodes.

6 The Verification of Interaction Rules

The hierarchical extraction always works on the *cell view* of a layout. This means for every cell that only its cell geometries on the actual level and its subcell abstracts on the next deeper level are visible. As we saw in the first chapters, interactions with geometries extracted earlier or with unvisible subcell geometries could influence semantic changes which are not represented in the netlist:

- Connections within one layer as well as between layers could be broken.
- New combinations of layers could modify or remove devices.
- Overlaps of subcell devices could generate their multiple representation in corresponding subcircuits.
- Internal nodes of subcells could be used for connections or pins.

In order to identify such interaction violations as errors, the IRC compares after the extraction of a cell its hierarchical view with its flat view (see Figure 1). The *flat view* of a cell is based on flattened cell geometries from which mask operations deduce the shape of devices and connections as they should be extracted, i.e., as they will appear on the physical chip. This means that the flat view acts as a correct reference view for the cell. Furthermore, we assume that subcells are already verified. Then we can use the flat view of the cell and the flat views of its subcells in order to verify the hierarchical extraction.

The *hierarchical view* of a cell is the collection of flat views of its subcells and of the cell view. It represents the semantic actually extracted. Of course, this comparison is only in critical regions of interaction necessary. Therefore, a filter reduces many geometries before these views are generated. Such filters are well known for hierarchical design rule checking [14,15]. Algorithms and performance results for filters are given in [16].

To control the IRC by the technology description, *interaction rules* define the generation and comparison of views (see Figure 8):

• *Generation rules.* For flat views, these rules are repeated once for each subcell and generate nonoverlapping layers. Hereafter, their views are collected implicitly with the cell view into layers with postfix '[HIER]'. These layers are not nonoverlapping. Next, the rules are repeated once more to generate the flat view of the cell in nonoverlapping layers with postfix '[FLAT]'.

```
a) MASK poly_int    := poly;
   MASK diff_int    := diff;
   ...
b) VIEW( MASK poly      := np;
         MASK gate      := diff AND poly;
         MASK diff      := nd ANDNOT gate;
         MASK nenh      := gate ANDNOT ni;
         MASK diff_int  := diff  ANDNOT diff_pins;
         MASK poly_int  := poly  ANDNOT poly_pins;...
       )
(1) ERROR "gate" gate[HIER] UNIQUE_XOR gate[FLAT];
(2) ERROR "diff" diff[HIER] XOR diff[FLAT];
(3) ERROR "nenh" nenh[HIER] XOR nenh[FLAT];
(4) ERROR "poly  connectivity" UNIQUE( poly_int[HIER] );
(5) ERROR "diff  connectivity" UNIQUE( diff_int[HIER] );...
```

Figure 8: Interaction rules for Figure 7. a): Additional extraction rules for the cell view. Hore, both layers are copied, as they are required for section b) with two different names. **b)**: Generation rules for flat views. Comparison rules are numbered. Multiple gate overlaps are checked in (1). Rule (2) verifies horizontal connections. Rule (3) detects changes of transistor types. (4) identifies touches and overlaps with internal nodes.

• *Comparison rules* are supported by mask operations. For every flat view device, there must be exactly one hierarchical view device. Since a comparison with XOR would ignore cases of overlaps inside a hierarchical view layer, special operations UNIQUE_XOR and UNIQUE detect overlaps in [HIER]–layers (see Figure 9).

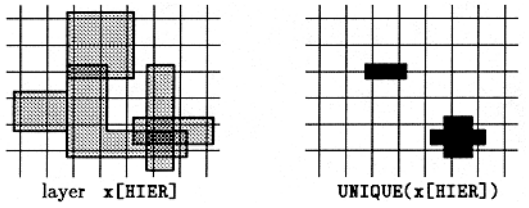


Figure 9: Operation UNIQUE used for overlap detection. Due to the stretched geometries, even touching geometries yield small overlaps if closed.

Figure 10 shows interaction errors generated by the rules of Figure 5 and Figure 7. Assume that cell L has no interface, i.e. its abstract layers poly_int and diff_int are empty. As no geometries are lifted to the calling cell F, all layers of the cell view are empty, too. The flat view of F contains four transistors and both diffusion lines are broken twice. But only two circuit calls for L are extracted. Therefore, the hierarchical view of F contains twice the flat view of L with both transistors extracted. Comparison rules will detect the difference and will generate interaction errors. Even if the designer would label both poly and diffusion lines in L, these interactions were not correctly extractable, because the connectivity for diffusion is extracted already in L, but broken in F. With modified rules, the designer has to protect the diffusion area against connectivity to overcome this error.

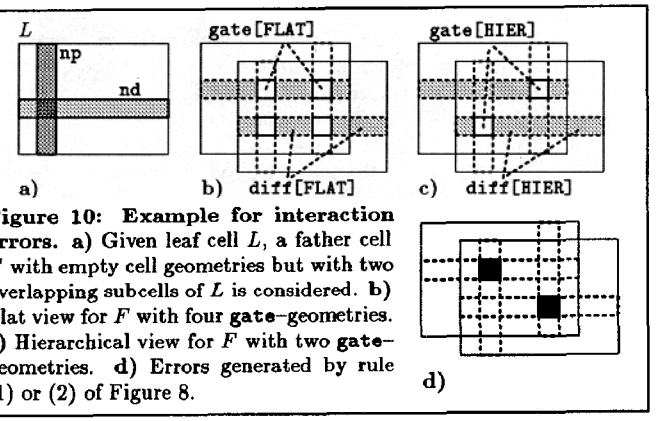


Figure 10: Example for interaction errors. a) Given leaf cell L, a father cell F with empty cell geometries but with two overlapping subcells of L is considered. **b)** Flat view for F with four **gate**–geometries. **c)** Hierarchical view for F with two **gate**–geometries. **d)** Errors generated by rule (1) or (2) of Figure 8.

With a second example, we demonstrate the power of open and closed geometries. As shown in rule (4) of Figure 7 with the metal interface and an additional given layer **nms** for masking, only parts of geometries are specified as an interface. This means, only the external part (the interface) is lifted in the hierarchy. On extraction, internal parts of subcells are invisible. As touching errors are detected only between closed parts (see Figure 11), no wrong errors occur between internal and external parts. In this example the interaction error can be removed by defining a larger interface for L.

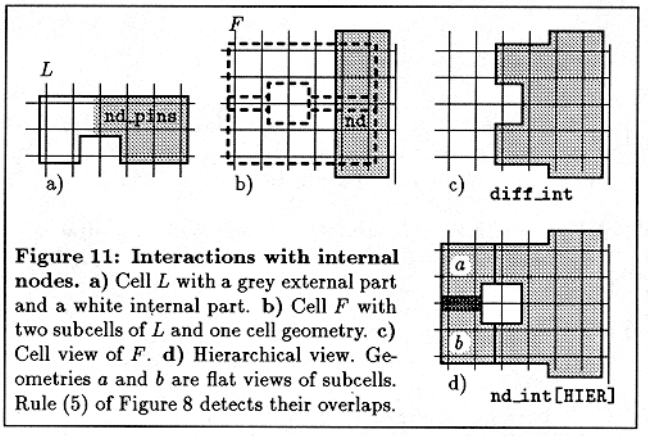


Figure 11: Interactions with internal nodes. a) Cell L with a grey external part and a white internal part. **b)** Cell F with two subcells of L and one cell geometry. **c)** Cell view of F. **d)** Hierarchical view. Geometries a and b are flat views of subcells. Rule (5) of Figure 8 detects their overlaps.

7 A System Implementation

Our implementation of the hierarchical incremental extractor RISCE including the IRC is integrated together with other tools in a design environment [17]. It operates under VAX/VMS and has 30000 lines of 'C' code. Geometries generated can be projected into the original layout together with their node or device numbers as well as error messages. The new concept of stretched geometries simplifies the implementation of mask operations drastically because we need not take care of special cases at geometry borders. Furthermore, we reduce the manifold and complicated device extraction procedures to a small but powerful set of instructions working with a user defined data structure. In comparing flat and hierarchical views of the layout semantic, we detect all interaction violations. Contrary to other extractors, a final verification by a flat extraction pass is unnecessary in our case.

Extraction speed strongly depends on the amount of details required, on the regularity and hierarchy structure of the layout and on the size of abstracts as well as on the area of interactions in the IRC. Of course, the filter is a crucical component if subcells have massive overlaps and must be flattened completely. Nonincremental extraction of an NMOS–chip with typical regularity and 50000 transistors in the flattened version requires less than 10 minutes on a VAX 11/750. An IRC requires two minutes only. Of course, incremental mode is faster. Experiences with our way of hierarchy handling and view comparison seem to indicate that designers learn to organize better structured designs.

References

[1] A. Bootehsaz, R.A. Cottrell, "A Technology Independent Approach to Hierarchical IC Layout Extraction.", *Proc. 23nd Design Automation Conference*, 1986, pp. 425–431.

[2] W.S. Scott, J.K. Ousterhout, "Magic's Circuit Extractor", *IEEE Design and Test of Computers*, Vol. 3, Feb. 1986, pp. 24–34.

[3] T.J. Wagner, "Hierarchical Layout Verification", *IEEE Design and Test of Computers*, Vol. 2, Feb. 1985, pp. 31–37.

[4] G.M. Tarolli, W.J. Herman, "Hierarchical Circuit Extraction with Detailed Parasitic Capacitance", *Proc. 20th Design Automation Conference"*, 1983, pp. 337–345.

[5] S.N. Stevens, S. McCabe, "An Integrated Approach for Verification of VLSI Mask Artwork", *IEEE Journal of Solid-State Circuits*, 1983, Vol. SC-20, Apr. 1985, pp. 501–509.

[6] K.H. Keller, A.R. Newton, S. Ellis, "A Symbolic Design System for Integrated Circuits", *Proc. 19th Design Automation Conference*, 1982, pp. 460–466.

[7] L. Scheffer, "Hierarchical Analysis of IC Artwork with User Defined Abstraction Rules", *Proc. 22nd Design Automation Conference*, 1985, pp. 293–298.

[8] Y. Wong, "Hierarchical Circuit Verification", *Proc. 22nd Design Automation Conference*, 1985, pp. 695–701.

[9] R.B. Tilove, "Set Membership Classification: A Unified Approach to Geometric Intersection Problems", *IEEE Transaction on Computers*, Vol. C-29, No. 10, 1980, pp. 874–883.

[10] T.G. Szymanski, C.J. Van Wyk, "Space Efficient Algorithms for VLSI Artwork Analysis", *Proc. 20th Design Automation Conference*, 1983, pp. 734–739.

[11] K.-C. Chu, Y.E. Lien, "Technology Tracking for VLSI Layout Design Tools", *Proc. 22nd Design Automation Conference*, 1985, pp. 279–285.

[12] M. Horowitz, R.W. Dutton, "Resistance Extraction from Mask Layout Data", *IEEE Transaction on Computer-Aided Design*, Vol. CAD-2, No. 3, 1983, pp. 145–150.

[13] V. Henkel, "Zur hierarchischen Parameter–Extraktion beim Entwurf integrierter Schaltkreise", Ph.D. thesis, Technische Universität Braunschweig, West Germany, 1988.

[14] T. Whitney, "A Hierarchical Design Analysis Front End", *Proc. of VLSI 81*, Edinburgh, 1981, pp. 217–227.

[15] J. Hannken–Illjes, U. Golze, "Ein hierarchischer inkrementeller Designrulechecker", *Informationstechnik it*, Vol. 28, No. 3, 1986, pp. 132–138.

[16] N. Hedenstierna, K.O. Jeppson, "New Algorithms for Increased Efficiency in Hierarchical Design Rule Checking" *Integration*, Vol. 5, No. 3-4, 1987, pp. 319–336.

[17] U. Golze, J. Hannken-Illjes, V. Henkel, E. Moser, "VLSI-Entwurfsumgebung KICBOX", *Handbuch der modernen Datenverarbeitung 127*, 1986, pp. 62–77.