A CIRCUIT DISASSEMBLY TECHNIQUE FOR SYNTHESIZING SYMBOLIC LAYOUTS FROM MASK DESCRIPTIONS

by

Bill Lin

Memorandum No. UCB/ERL M89/21

27 February 1989

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

A CIRCUIT DISASSEMBLY TECHNIQUE FOR SYNTHESIZING SYMBOLIC LAYOUTS FROM MASK DESCRIPTIONS

by

Bill Lin

Memorandum No. UCB/ERL M89/21

27 February 1989

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

Abstract

A new tool called a *circuit disassembler* has been developed to transform a mask-level layout into an equivalent symbolic layout. This technique has been implemented in a program called KAHLUA that can handle mask layout containing arbitrary Manhattan geometries. Circuits designed using physical layout systems can be disassembled automatically independent of the circuit technology into a symbolic environment. Once converted, the disassembled cells can be manipulated further by any existing symbolic design or verification tools. In particular, these cells can be automatically remapped for a new technology. Our formulation of the problem consists of two major stages: device extraction, and net decomposition. In the first stage, the transistors and contacts are extracted from the layout to form leaf cells. In the second stage a set of symbolic wires is derived from the remaining interconnect geometry.

Acknowledgments

I would like to acknowledge members of the Berkeley CAD group for their support during the course of this work – I wish to especially thank Dr. Ulrich Lauther for his assistance during his stay Spring semester of 1986 at Berkeley, Peter Moore for his early work with artwork analysis routines, Jeff Burns for his work with SPARCS and his patient help with tool integration, and Srinivas Devadas for reviewing this report and the many stimulating conversations we've had about various research topics.

I would like to thank my research advisor A. Richard Newton for suggesting the project and providing support throughout the course of this work, and I would like to thank Professor Carlo Sequin for being my second reader and for taking the time to read this report carefully. Both Professor Newton and Professor Sequin gave me many helpful suggestions for improving this report.

This research was supported in part by DARPA under contract N00039-83-C-0107, by the Hughes Aircraft Company, and by the Digital Equipment Corporation. Their support is gratefully acknowledged.

Contents

Chapter 1

Introduction

Full custom physical design systems have been used extensively for the layout of VLSI circuits (eg. [Ous81,Kel82a,Ous85]). Such systems represent the circuit as a collection of geometric objects (rectangles, polygons, etc.) at the mask layout level. A serious drawback of describing a design only at the mask-level is that once a cell has been designed, it is generally very time-consuming to modify the layout, even for minor changes in design rules. Many of the difficulties stem from the fact that even seemingly small changes to a mask layout can have disproportionately large effects. These effects occur because area is at a premium, so cells are designed with little room to spare. Hence, when anything in the cell moves or gets bigger, its neighboring information must be moved as well, so as to ensure that the layout rules are satisfied. Modifying mask layouts is a tedious and error-prone process. Simply performing a uniform "shrink" to accommodate new design rules is usually very area inefficient. But more importantly, it does not work in advanced technologies due to non-linear scaling effects.

Because of the large investment in design time involved in the design of highly optimized VLSI circuits, it is important that such effort is not wasted when design rules change. For example, it would be quite desirable to be able to reuse existing cell libraries for new process

technologies. For high-performance, high-volume, circuits (eg. advanced microprocessors) that are extremely expensive to develop, it is critical that such designs be able to migrate through a number of process improvements over their lifetime. In addition, because of the evolution of the ASIC market, multiple-source "foundries" have become essential for many companies. This requires that a component be retargetable for different fabrication processes. If all circuits are designed with technology remapping in mind at the outset, data structures can be used to aid in this process. However, there are many designs that have been developed at the mask-level that the designers would like to be able to map to new technologies.

One solution to technology independence is through the use of symbolic layout systems [Lar71,Cho77,Hsu78,Dun78,Wes81,Bur86,Boy87]. These symbolic layout systems promise to overcome some of the problems associated with technology dependence by maintaining explicit connectivity and structural information about the the circuit. These systems are usually accompanied by a compaction tool that allows the designer to be more concerned with actual design rather than detailed mask artwork. Because structural information is stored explicitly, symbolic layouts can easily be converted to final layout whenever the process layout rules change. In addition to the advantage of technology independence and design flexibility, there are many CAD applications that rely heavily on the existence of explicit connectivity information for carrying out their objectives. For example, the task of generating input for circuit simulation is simplified considerably when extracting from a symbolic format – the only task involved is the computation of parasitics since the connectivity information is stored explicitly.

Despite the advantages of symbolic layout, there is still considerable resistance in the industry to use symbolic design systems. In part, this is because most companies have already expended tremendous investments in the use of mask-level systems. Since mask layout systems have been used extensively in the past, there are many existing circuit layouts and library cells available in these environments. Particularly important are libraries of standard cells, datapath

slices, and module generator templates, which can potentially be re-used in new designs. However, these cells must be converted in order to fully integrate them into a symbolic design environment.

In this report, a new technique called *circuit disassembly* is introduced. This technique is used to synthesize equivalent symbolic layouts from mask-level descriptions and has been implemented in a technology-independent program called KAHLUA that can transform layouts containing arbitrary Manhattan geometries into a symbolic format. Once mapped into symbolic form, these cells can be parameterized for new technologies and can be manipulated by a variety of existing symbolic design and verification tools. In practical circuit examples, we have obtained good results within modest execution times. An additional feature of KAHLUA is its ability to disassemble hierarchical designs, as described in later chapters.

Our approach begins with a collection of mask-level polygons associated with the different mask layers. The actual meaning of the mask layers is defined by the user and thus is independent of the particular technology. From these mask geometries, transistors and contacts are derived to form leaf cells in the symbolic layout. The remaining interconnect geometries are then converted into symbolic wires for connecting the leaf instances. While this process is similar to net list and parasitic extraction, the final output of the analysis is a symbolic layout which can be inverted to re-generate the mask layout data. There are many problems associated with maintaining the geometric form of the original layout that arise during this transformation that are not present in the case of extraction. For example, to decompose a diffusion polygon, used to interconnect multiple transistors, general area routing techniques are used. The eventual result of the analysis is a complete symbolic design suitable for use in a symbolic design system without further manual modification.

The procedure described in this report is targeted for digital CMOS technologies. However, the algorithms presented can be extended to operate on other technologies.

The organization of this report is as follows. In the next chapter, an overview of symbolic layout is presented. Chapter 2 is a very important chapter because it describes the symbolic layout representation used for the circuit disassembly procedure. To a large extend, the premise of our work is set by the features and limitations of this representation.

In Chapter 3, a formulation of the circuit disassembly problem is presented, along with a discussion of some of the inherent difficulties. The problem of circuit disassembly is formulated as two separate but related problems, namely, device extraction and net decomposition. The purpose of the device extraction procedure is to extract transistor and contact information from the mask description in order to generate symbolic circuit elements. After the circuit elements have been created, the net decomposition procedure is invoked to derive symbolic wires from the remaining interconnect geometry that is necessary for interconnecting the circuit elements. Algorithms for solving these problems are also presented in this chapter.

In Chapter 4, the implementation of the program KAHLUA is described. In particular, KAHLUA uses extraction and routing techniques described in the two previous chapters to implement the circuit disassembly procedure. The application of these algorithms and special considerations specific to the circuit disassembly problem are presented. Finally, experimental results are presented on various examples.

An important advantage of symbolic layout is that certain CAD problems are simplified when the input description is already in the symbolic form. For example, the task of converting a layout description into an electrical network suitable for circuit simulation is greatly simplified starting from a symbolic representation since explicit circuit elements and connectivity are already known. A simple circuit extractor is described in Chapter 5.

The ideas and results presented in this report are summarized in Chapter 6. Suggestions for future work are also given.

Chapter 2

Symbolic Layout

The increasing complexity of Very Large Scale Integrated (VLSI) circuits has made the used of computer aided layout tools for capturing custom designs a necessity. Traditionally, custom layouts are generated using mask-level layout system. The user enters the design via a graphics editor for drawing boxes and polygons. These mask-level layout systems have been used extensively in the industry for the layout of VLSI circuits since the mid 1960's. For example, commercial systems from CALMA and Applicon have been in use since 1965.

Although mask-level layout systems are still widely used today, generating custom layouts using these systems is a tedious and error prone process. Once the artwork is captured, it is usually very time consuming to modify the layout even for minor changes. For major changes such as new design rules, it is usually infeasible to modify large mask-level designs manually.

The use of abstractions, or *symbols*, to represent components (eg. transistors and contacts) of a circuit can help reduce the complexity of the design process. In the symbolic layout design style, designers create full custom layouts using symbolic circuit elements and wires rather than actual mask geometry. Thus, some of pains associated with mask-level details are alleviated.

Symbolic design has been in use since the early 1970's [Larsen71]. Since then, the symbolic layout design style and methodology has evolved considerably. In most of these symbolic design systems, a *spacing* procedure is integrated for translating the symbolic layout to the mask-level. The purpose of the spacing tool is two fold. One is to generate a design rule correct mask description given a process technology. This capability simplifies the design task since the designer does not have to worry about design rule correctness when creating the layout, which can be quite tedious. The second purpose is to *compact* the design such that the layout is as small as possible without violating any design rules for the target IC process. This is accomplished by relocating the circuit elements so that unnecessary empty space is eliminated.

Over the years, a number of symbolic design systems have been developed. These systems can be classified by the way they represent the layout and their compaction method. In this chapter, various existing symbolic design systems are reviewed. In particular, their layout representation and compaction methods are examined. They can be classified into four categories: fixed grid systems, virtual grid systems, relative grid systems, and hierarchical systems. The algorithms presented in this report can be used to synthesize symbolic layouts in any of these representations. The hierarchical representation is the most general and complicated to synthesize. It will be described in much greater details at the latter part of this chapter. This representation sets the basis for the circuit disassembly procedure presented in this report.

2.1 Existing Symbolic Layout Systems

2.1.1 Cell Level Compaction

Earlier systems were based on fixed grids. These systems have been in use within industry since the early 1970's [Lar71]. The spacing in these systems is between grid lines rather than circuit elements. In fixed grid systems, the worst case design rule is used to set the spacing between grid lines uniformly. Significant area can be wasted as a result. The SLIC program from AMI [Gib76] and the MASKS system from Rockwell International [Lar78] are examples of these earlier systems.

Virtual grid systems were later introduced to address some of the limitations from the fixed grid approach. Like the fixed grid approach, symbolic circuit elements are placed at grid locations. In a virtual grid layout, the relative placement of the circuit elements is determined by their location on the input grid. This grid serves only to delineate the relative ordering of the circuit elements. The spacing between grid lines is dependent on the actual spacing required by the circuit elements on the grid lines. Different pairs of adjacent grid lines may have different spacing. Therefore, the spaced grid lines may not be uniformly distributed. The resulting compacted layout is more dense than could be attained with fixed-grid layout. One major disadvantage of virtual grid systems is that the circuit elements originally placed on a single grid line will remain aligned after compaction whether they need be or not. This restriction can potentially cost a notable amount of wasted area that could otherwise be removed via a more general compaction method. Hence, a layout density penalty may be incurred. The MULGA system [Wes81] developed at Bell Telephone Laboratories is an example of a virtual grid system. Another notable system is the VIVID system [Rog83] developed at the Microelectronic Center of North Carolina (MCNC). Even today, these and other reported systems are being used to design

integrated circuits.

Relative grid systems are yet more flexible than their virtual grid counterparts. In these systems, the circuit elements are not restricted to lie on some arbitrary grid lines. But rather, compaction strategies have been developed to exploit the additional empty space by allowing the circuit elements to slide pass each other. The final spacing is determined by spacing rules between circuit elements rather than grid lines. In principle, true relative grid systems should produce layouts with the densest areas.

One of the first systems to use the relative grid method is the FLOSS program from RCA [Cho77]. Another system that uses a relative grid model is the STICKS program developed at the Hewlett Package Company [Wil78]. This program compacts the symbolic layout by starting from one side of the layout and sequentially placing elements as far to the side as possible, given the positions of the previously placed elements and the spacing rules on different mask layers.

The CABBAGE system [Hsu78] developed at Berkeley used the critical path method (CPM) to compute the locations of the circuit elements. The CPM approach has the advantage that it can handle other more complex constraints than simple minimum distance rules (eg. maximum distance). A more recent relative grid system is the DASL system [Boy87] developed at Bell Communications Research. In the DASL system, the design is captured using a "virtual grid" like editor, but ignores the virtual grid lines during compaction. The internal data structure is a sparse matrix where explicit pointers are stored between adjacent circuit elements. Therefore, searching through the data structure is very fast. Like the STICKS system, the compaction procedure in DASL compacts a symbolic layout by starting from one end to the other while placing elements as close together as possible.

Other relative grid systems include the SLIP and SLIM systems described in [Dun78] and [Dun80], respectively. In these systems, critical path analysis, similar to the method used in CABBAGE, is used along with a local-compaction method for spacing the layout.

2.1.2 Hierarchical Compaction

There are two common ways of supporting hierarchy in symbolic design systems. The first is to treat flat design and hierarchical design as separate problems. These systems have a different representation for the *cell* level design and the *block* level design. The compaction at the cell level is also handled differently than at the block level. These compactors are typically referred to as *cell compactors* and *block assemblers*, respectively. At the cell level, circuit elements such as transistors and contacts are hard-wired into the system. Hence cell compactors can make use of this special information in the spacing procedure. However, these cell compactors cannot be used to compact *black boxes* along with cell level elements such as transistors. At the block level, the representation and compaction techniques are tailored for handling hierarchical objects, such as macro cells, rather than cell level circuit elements.

For example, in the MULGA system, a virtual grid compactor is used to optimize layouts at the cell level. At the block level, clusters of cells are assembled by abutting them together. These blocks are then forced to be *pitch-matched* to maintain connectivity.

An alternative paradigm is to treat cell level objects, such as transistors and contacts, in the same manner as hierarchical objects, such as macro cells. Thus, no distinction is made whether a design is flat or hierarchical. A generic abstraction can then be used to represent the design at all levels of the hierarchy. This is the approach taken by the PYTHON [Bal82] and SPARCS [Bur86] systems. These systems are based on a general constraint mechanism for representing spacing relationships between objects. These objects have a generic abstraction regardless of what the object is. Therefore, these systems can be used to represent and compact circuits other than digital CMOS. In the PYTHON system, the ideas of *protection frames* and *terminal frames* were introduced. The protection and terminal frames can be automatically generated using an abstraction tool [Moo82]. These frames are used to represent the interior of the object and the legal connection area, respectively. Relationships between objects can be specified in terms of

minimum constraint and *maximum constraint*. The compaction step is performed using a general critical path analysis procedure.

The system SPARCS is the successor to the PYTHON system. The fundamental representation and compaction method are very similar to PYTHON's. However, several new features were added. One is the ability to detect *overconstraints* quickly. Another is the ability to represent and handle *active constraints*. Active constraints can be used to represent as functions of other constraints. This can be used to maintain symmetry between objects during compaction. For example, symmetry may be necessary for compacting analog circuits.

2.2 A General Hierarchical Representation

The circuit disassembly procedure described in this report is intended to be used with the SPARCS symbolic design system. Therefore, it must support hierarchical designs in a general object oriented representation environment. However, the algorithms presented can be used to synthesize symbolic layouts in other symbolic representations. In this section, the symbolic layout representation used in the SPARCS system is described in greater details. In particular, the concepts of protection frames, terminal frames, and hierarchical objects are elaborated.

2.2.1 Protection Frames

Protection frames are used represent a simplified interface abstraction of a cell. In compaction, for example, protection frames may be used to abstract away some unnecessary details of a cell for compacting at the next level of the hierarchy. Protection frames may be defined as a single polygon or as a number of polygons corresponding to the mask layers. The only restriction is that the protection frames must contain all the geometry inside them. The geometry inside a protection frame is "protected" against interaction with other geometry on the same mask layer.

For example, the protection frame might simply be a single bounding box of the cell. Although it is easy and efficient to derive, a single bounding box does not provide a true representation of the internal geometry that it bounds. A more general model is to bound the geometry with bounding rectangular polygons on a per-layer bases. This is in fact what is done in PYTHON and SPARCS. The rectangular polygon approach appears to be a good tradeoff between simplicity and efficiency.

For example, the protection frames for a MOS transistor might be a polysilicon frame and a diffusion frame corresponding to the size of the active area.

2.2.2 Terminal Frames

The complement to protection frames is terminal frames. The terminal frames define the allowable area of interconnection for the cell object. One of the key differences between terminal frames and other simpler symbolic representation is that wire connections can be made anywhere along the terminal area rather than just a single point connection. However, the wire must terminate within the boundaries of the terminal frame. This added flexibility is most apparent in the case where the terminal frame is much wider than the width of the wire. In which case, the compaction program can take advantage of this by sliding the wire along the terminal area for maximum density. Another feature is the ability to connect more than one wire to the same terminal, which may be difficult to do in point-based terminal representations.

In the PYTHON and SPARCS system, the definitions of terminal frames have two restrictions. One is that at least one edge of the terminal frame must be coincident with a protection frame edge. The second restriction is that the terminal frames must be defined in terms of a single rectangle. At present, PYTHON and SPARCS only support single rectangle terminal frames.

The issue of complex terminal frames is a difficult one. Ideally, rectangular polygon terminal frames should be supported. One way to represent polygon terminal area is simply by several

non-overlapping rectangle terminals. For example, in Figure 2.1, the polygon terminal region is represented with three rectangle terminals. One disadvantage of this approach is that the wire connected to one of these terminals is only free to move within that particular terminal area. However, if the wire is free to move along the entire polygon terminal area, then potentially denser results can be achieved. Another problem with representing rectangular polygon terminal frames with only box terminals is that a wire cannot be connected to the entire polygon terminal area with a single connection. Therefore, the width of the wire is limited by the width of box terminal that it connects, as shown in Figure 2.2. In IC design, it is often desirable to connect a single wire the width of the polygon terminal area to the polygon terminal. However, this would require the representation of both polygon terminal frames and wires with "jagged" ends. This limitation makes representing non-linear transistors translated from mask layouts extremely difficult, as examined in greater details in Chapter 4. To be fair, the complex terminal frame problem is not unique to this symbolic layout representation. Most systems simply do not allow circuit elements with rectangular polygon terminals, or they have ad hoc ways of representing them with similar limitations.

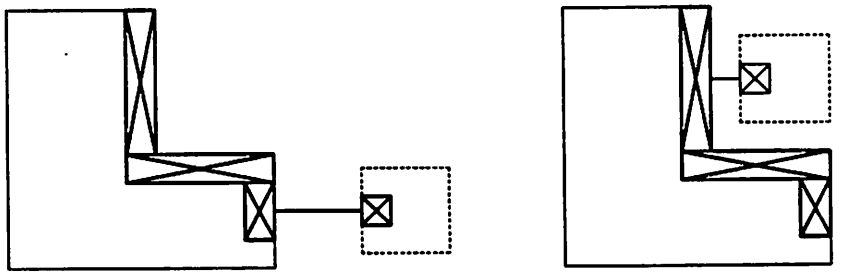


Figure 2.1: This figure shows the spacing savings possible resulting from the use of complex terminal frames. Potentially denser results can be obtained if a wire is free to move along the entire polygon terminal area rather than a box terminal area.

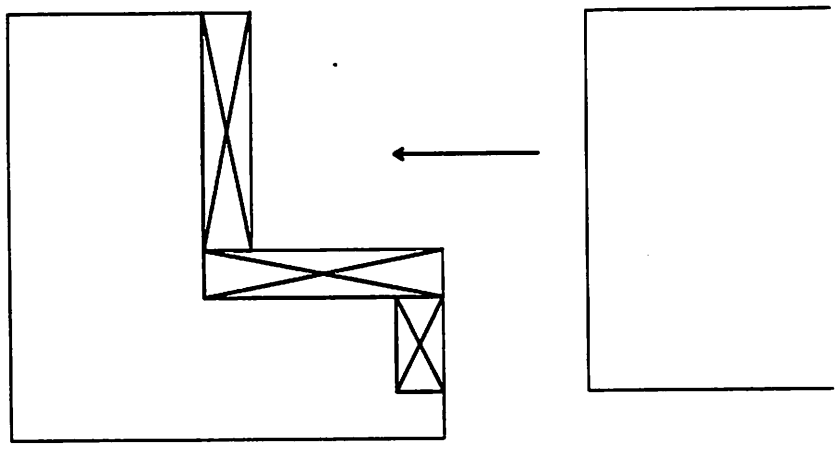


Figure 2.2: This figure shows the difficulty with connecting a wire to the entire polygon terminal area using only a single connection.

2.2.3 Hierarchical Objects

The concepts of protection and terminal frames combine together to form the basis of a general hierarchical representation. From the point of view of the compactor, objects are defined in terms of their protection and terminal frames, and hence, no distinction is made whether the object is a simple transistor or a complex macro cell. Thus, hierarchical objects are represented and treated in the same manner as cell level objects. Symbolic wires are used to interconnect these generic objects. During compaction, these objects are relocated to minimize area while satisfying a set of complex constraints. The symbolic wires are then readjusted accordingly.

Chapter 3

The Circuit Disassembly Problem

In this chapter, the problem of circuit disassembly is presented. The problem of circuit disassembly is defined in the next section and then a formulation of the circuit disassembly problem is presented in Section 3.2. In particular, the circuit disassembly problem can be cast in the form of other CAD problems, namely circuit extraction and routing. The elegance of the formulation stems from the synergy of these steps. The following two sections are dedicated to reviewing previously developed techniques for solving the extraction and the routing problems.

Finally, in the last section, the issues involved in circuit disassembly are presented. In particular, two major complexities are inherent. The first is the fact that there is no *one-to-one correspondence* between symbolic and mask layouts. Hence, some heuristics are required to make aesthetic decisions and approximations where necessary. Second, the problem is complicated by performance considerations. That is, it is crucial that the transformed symbolic layout have similar performance characteristics as its original mask counterpart.

3.1 Formulation of the Problem

The starting point of circuit disassembly is a mask-level description. In a mask description, the circuit is described by a set of associated polygons on the different mask layers. Other than this information, no other explicit information is given. In particular, circuit elements, like transistors and contacts, are not specified explicitly. These circuit elements can be inferred by performing geometric analysis on the relationships between the polygons. For example, simple MOS transistors can be inferred by the intersection of two mask layers, namely polysilicon and diffusion. In addition, connectivity information in a mask layout is also implicit.

In symbolic layout, on the other hand , the definition of a circuit is given in a considerably more explicit form. In particular, transistors and contacts are defined explicitly as symbolic circuit elements. For example, the location of a transistor, the size and shape of the active channel region, the location and definition of its device terminals are all specified directly. In addition, connectivity information is given. Not only mere connectivity, but the exact structure and topology of the interconnects are specified explicitly. This is accomplished by specifying the interconnections with individual symbolic wire segments.

The task of circuit disassembly is, therefore, to synthesize from a mask level description, where little information about the circuit structure is given, to a detailed symbolic layout with considerable amounts of added information.

While the problem of circuit disassembly *per se* is new, considerable work has been done in the area of circuit extraction [Fit82,Tar83,Gup83,Sco85]. The problem of circuit disassembly differs from that of circuit extraction because in the latter case only transistor dimensions and connectivity are considered. In particular, connectivity information alone is inadequate for producing a symbolic layout. A circuit disassembler must in addition derive a set of symbolic wires for connecting the symbolic circuit elements to form a complete circuit. These symbolic wires must correspond to the original topology of the mask layout.

In order to solve both the problems of extracting circuit elements and synthesizing symbolic wires, the circuit disassembly problem is formulated into two separate tasks. The first task is called *device extraction*. This procedure corresponds to the extraction of circuit elements from a mask layout. The second task is called *net decomposition*, which corresponds to the need to decompose interconnect geometry into symbolic wires. The former problem is related to circuit extraction while the latter is related to the routing problem.

3.2 Layout Extraction Techniques

The purpose of a layout extraction tool is to extract electrical information from a layout description for simulation and verification purposes. For example, the extracted output may be used for net list comparison between the layout and the original schematic. Another usage would be to perform a detailed circuit simulation using a program like SPICE [Nag73]. Among its tasks, an extractor must extract the transistors in the circuit and establish their connectivity.

The techniques developed for layout extraction can be applied to solve part of the circuit disassembly problem, namely the device extraction step in the disassembly process. The layout extraction problem requires several basic operations for analyzing VLSI artwork. These operations are commonly called Boolean mask operations. One of the most commonly used operation is the Boolean AND operation for finding the intersections of polygons. Another common operation is the MERGE operation. This is basically a variation on the AND operation where two polygons are considered *merged* if they belong to interacting mask layers and they intersect. Other operations such as OR and NOT are also variations of the AND operation.

For large designs, finding the intersections between all polygons in different mask sets (ie. layers) is a very time-consuming task. Over the years, several algorithms for finding intersections have been reported. In this section, some of these techniques are examined.

3.2.1 Bin Based Methods

The simplest method is to store the polygon in a linked list data structure. To perform an intersection operation, every polygon must be compared against every other polygon. Surprisingly, this method has been used in many systems. Although this method is simple to implemented, it is too slow for most applications. Therefore, more sophisticated data structures have been developed for storing and manipulating the geometry. One popular approach is to break up the layout area into bins. For example, the layout area can be divided into square bins [Ben80], as shown in Figure 3.1. Associated with each bin are the polygons that intersect it. Therefore, for most geometric operations with a polygon, the search is localized to the bins that it intersects. In practice, the search time is good. One problem with this scheme is that the efficiency is very much dependent on data. At the floorplan level of an IC layout, the objects vary substantially in size, ranging from a simple contact to an entire macro cell. If the bins are too small, cell instances can potentially intersect a number of bins. If they are too large, then they are little of use.

These problems lead to more sophisticated bin data structures based on rectangles rather than squares. In [Ous82], a data structure based on vertical and horizontal bins was described. This data structure allows for more flexibility in the bin sizes so that geometry with different shapes and sizes can be better supported. A variant to this data structure was proposed in [Kel84] called OSL (Orthogonal Scan-Lines). In OSL, the geometry is partitioned into non-overlapping, horizontal, rectangular bins termed *logs*, and non-overlapping, vertical, rectangular bins termed *poles*. To answer a rectangular range query, all bins that intersect the query rectangle are searched.

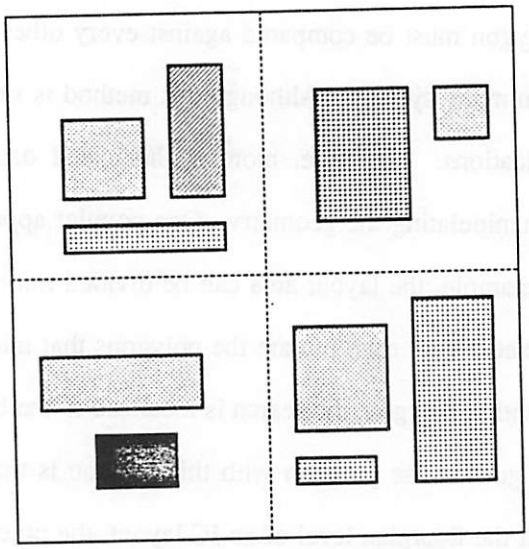


Figure 3.1: This Figure illustrates the binning technique. Basically, the layout area is divided into a number bins. The search is localized to only the relevant bins.

3.2.2 Tree Based Methods

As an alternative to rectangular bin based data structures, tree based techniques have been proposed. One such tree based data structure is called QUAD trees [Ked82]. It is based on the same idea as binning, but with some variations. The basic idea is to successively divide the layout area into quadsections. The root node in the QUAD tree is the entire layout. The root node branches into four children representing the quadsections. Each child node will in turn have four children nodes of its own until the lowest level of the tree which is the leaf level. The number of levels in the tree is typically demand-driven. For example, the algorithm may stop dividing a node into smaller nodes if the total geometry area in that node is less than some threshold amount. In Figure 3.2 the QUAD tree data structure is illustrated. Again this method works well in practice but is very dependent on data. Other tree based methods have been proposed. In [Lau80] and [Ben80], 4-d tress and k-d trees were proposed. They are based on similar ideas. The basic ideas is to maintain a balance tree so that geometry can be searched efficiently. Good results using

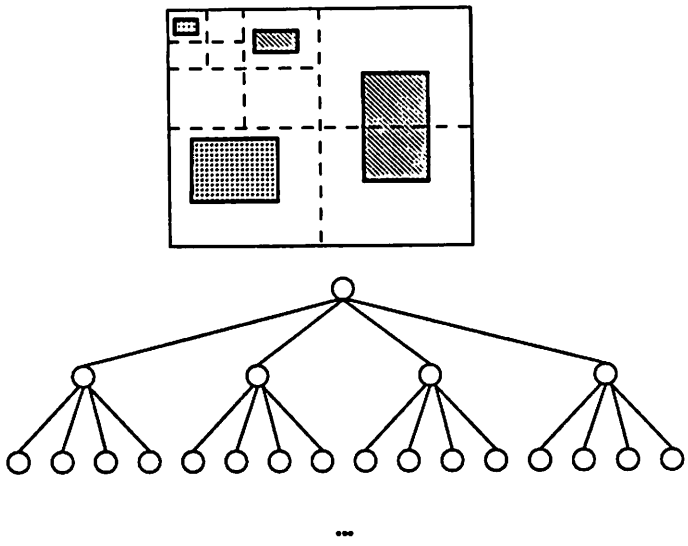


Figure 3.2: This figure illustrates the QUAD data structure. The layout area is successively divided into quadsections. The search is localized to the current node and all its children.

these data structures have been reported [Lau80].

3.2.3 Corner Stitching

Corner stitching is a data structure where the polygons are stored as non-overlapping rectangles called *tiles* [Ous84,Scot85]. It appears to be an efficient data structure for a variety of geometric search operations. A special property of corner stitching is that even empty space is represented explicitly by tiles whose type is "space". In Figure 3.3, a corner stitched plane is illustrated. The tiles in a plane are linked to their neighbors at their corners with four pointers, called *stitches*, two at each of the lower-left and upper-right corners. A unique feature of corner stitching is that adjacency information between neighboring tiles is stored explicitly. This information allows for very efficient searching operations. Thus, rather than comparing every polygon with every other polygon, the search is only limited to neighboring polygons. The basic searching algorithm involves traversing the corner stitch plane.

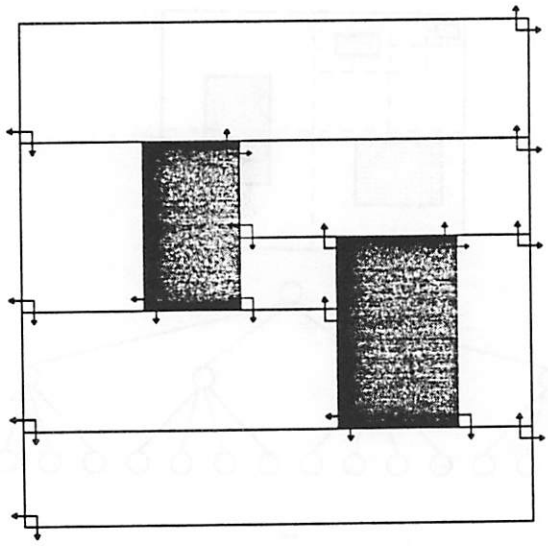


Figure 3.3: This figure shows a simple example of corner stitching. In corner stitching, polygons are represented as *tiles*, which are maximum horizontal strips. This tile plane data structure is efficient for various search operations since tile adjacencies are given explicitly as *stitches* on the corners of the tiles.

3.2.4 A Planesweep Algorithm

The algorithm [Bai77,Lau81] has been successfully used in many layout systems [Fit82,Tar83,Gup83] for extracting circuit devices from VLSI layouts. This technique also aims at reducing the complexity of the search operation by localizing the comparisons to neighboring geometry. The basic idea in this approach is to sweep a vertical scan line from one end of the layout to the other. In this planesweep motion, polygons are only analyzed when the scan line intersects them, thereby localizing the search to only those polygons with intersecting horizontal coordinates.

To simulate the planesweep notion, the data in the layout is first organized by sorting the edges of the polygons in the increasing horizontal coordinates. A sweeping action is then simulated by scanning through this data structure in its sorted order. In actual implementation, three data structures are constructed. The first one is called the STARTLIST. This is a list of all the

horizontal edges in the entire layout. These edges are sorted according to their starting horizontal coordinates. The second data structure is the ENDLIST. This is also a list of all the horizontal edges, but they are sorted according to their ending horizontal coordinates. These two data structures together represent the *plane* of the layout. The third data structure is the vertical scan line SCANLINE. The purpose of the vertical scan line structure is to store current horizontal edges as it sweeps through the plane. Initially, the vertical scan line is empty. At each significant horizontal location, all the edges on top of the STARTLIST that begin there are removed from the STARTLIST and inserted into the SCANLINE. At the same time, all the edges on top of the ENDLIST that end there are deleted from both the ENDLIST and the SCANLINE. This way, all the edges that have been passed are deleted from the STARTLIST and ENDLIST, and only the current edges are stored in SCANLINE.

A variation that is more efficient is to build the ENDLIST incrementally. Initially, the ENDLIST is empty. As new edges are inserted into the SCANLINE, they are also inserted into the ENDLIST. When an edge is no longer active in the current SCANLINE, it is removed from both the SCANLINE and the ENDLIST. Therefore, this approach is more efficient since only active edges are stored and consulted in the ENDLIST.

At each scan location, the edges on the current scan line are compared for analysis. This is accomplished by scanning the data structure SCANLINE from top to bottom. Hence, the SCANLINE data structure must be kept sorted. An efficient implementation of this algorithm was reported by Lauther in 1981 [Lau81]. Also, the planesweep technique has been successfully used in computer graphics since the mid 1970's [Ham77]. The planesweep method is illustrated in Figure 3.4.

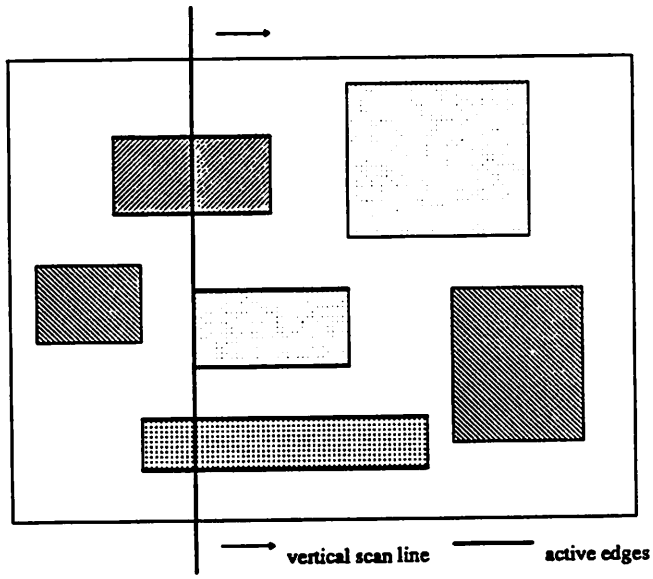


Figure 3.4: The planesweep method is illustrated in this figure. The basic idea in this approach is to sweep a vertical scan line from one end of the layout to the other. In this planesweep motion, polygons are only analyzed when the scan line intersects them, thereby localizing the search to only those polygons with intersecting horizontal coordinates.

3.3 Routing Algorithms

The purpose of routing is to implement the physical interconnect among a set of terminals that are electrically connected (ie. on the same net). Over the years, numerous routers have been developed to solve the different routing problems of maze routing, global routing, switchbox routing, and channel routing [Lee61,Hig79,Ree85,Shi86]. In the general case, a router must worry about a constrained routing region and obstacles within the region. For the purpose of solving the net decomposition problem in circuit disassembly, the router must be able to handle an arbitrary routing region, much like a maze routing problem. In this section, some applicable routing techniques are reviewed for solving this problem.

3.3.1 Wave Propagation Method

Lee's algorithm is perhaps the most widely used algorithm for solving maze routing or similar problems [Lee61]. The basic idea is to expand a wave front from a terminal until another terminal is found. The procedure begins by mapping the routing area and terminal locations onto a two-dimensional grid. In Figure 3.5, a simple routing problem is shown. Starting from a terminal location, a wave is propagated outwards until another terminal is reached. The algorithm then backtracks to find a feasible solution. This process is repeated until all the terminals are connected. In actual implementation, a queue is maintained to determine which terminals still require connection. This algorithm has a worst case time complexity of $O(p)$ for connecting two terminals, where p is the number of grid points in the two dimensional grid.

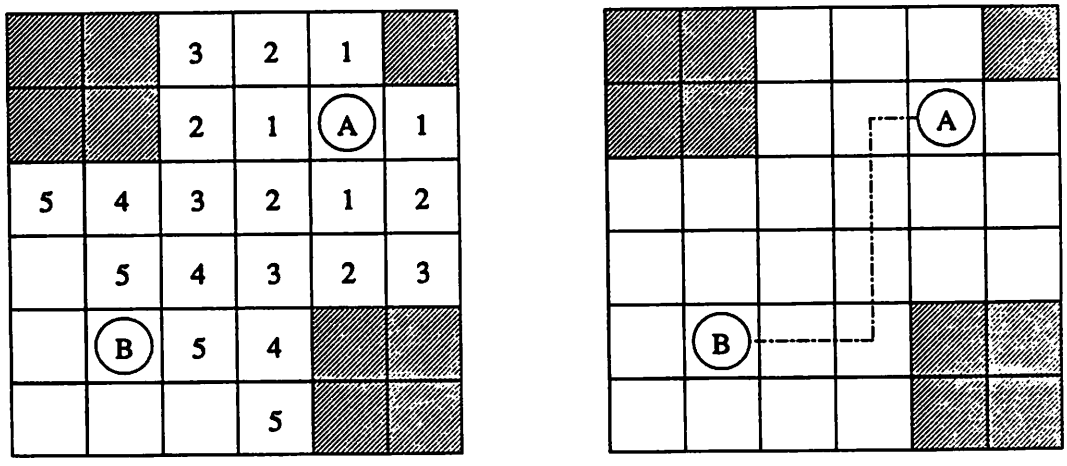


Figure 3.5: This figure is an example of a routing problem solved using the wave propagation algorithm developed by Lee [Lee61]. The routing problem is mapped onto a two-dimensional routing grid. A wave is propagated outwards from terminal "A" until another terminal, say "B" is reached. The shortest path can be found be backtracking, as shown in the gray area.

This algorithm is very attractive for several reasons. One is that it is very general. Any general area routing problem can be cast into this form. The two-dimensional routing grid can process irregular routing regions as well as obstacles. A nice feature of the algorithm is that it is guaranteed to find the shortest path if a path exists.

3.3.2 Line Search Method

The line search method [Hig79] is similar to Lee's wave propagation algorithm except that a *ray* is expanded rather than a wave front. In Figure 3.6, a pictorial description of the line search technique is given. Starting from a terminal, a ray is propagated in each of the four directions. When a boundary is encountered, the ray gets diffracted in the two orthogonal directions. A path is found when a ray intersects another terminal. The basic advantage of this technique over Lee's is two fold. First, since the algorithm is based on a continuous plane, there is theoretically no limit to the degree of precision that can be used to describe the coordinates of a routing problem. This is important for very large routing problems requiring a high degree of accuracy. The second advantage is that the algorithm only stores line segments. Therefore, only segments currently being defined need to be investigated, whereas conventional Lee style algorithms must examine minimal paths. In practice, modifications on Lee's algorithm can achieve similar efficiency. However, a disadvantage of Hightower's algorithm is that it is does not guarantee the shortest path.

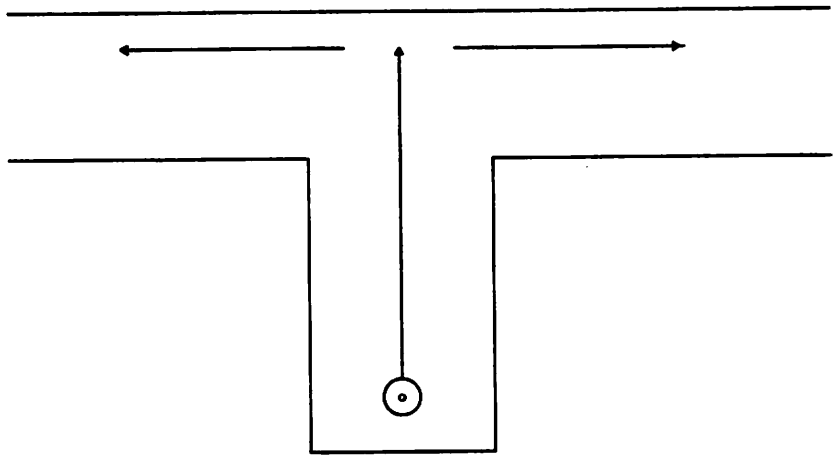


Figure 3.6: This figure gives a pictorial description of the line search routing method by Hightower [Hig79]. Starting from a terminal, say "O", a ray is propagated in each of the four directions. When a ray hits a boundary, the ray gets diffracted in the two orthogonal directions.

3.3.3 Pattern Matching

For a very limited number of unconstrained routing problems, the optimum Steiner tree solutions are already known. In particular, the number of solutions to routing problems with only two or three terminals is finite. Hence, a set of templates can be stored in a database for later lookups. These templates can be enumerated to find a solution. In practice, however, there are obstacles and other constraints making some of those solutions infeasible. In addition, quite often much more than three terminals must be routed together. Nonetheless, the template lookup technique can be quite useful. This is particularly useful when the routing problem can be cast in this simple form. If more than three terminals require connections, then they can be heuristically partitioned into smaller problems with three or less terminals.

3.4 Issues in Circuit Disassembly

The main difficulty in circuit disassembly is that there is no one-to-one correspondence between arbitrary polygons in a mask layout and the symbolic wires in a symbolic description. Therefore, they cannot simply be translated. Consider a small example in Figure 3.7. An interconnect region is shown that connects four terminal locations (eg. transistor terminals, or contacts). Given this geometric description, the disassembler must derive the four wires that connect the terminals as shown. To generate this kind of information optimally from a general polygon region with an arbitrary number of connection points is equivalent in complexity to a Steiner tree problem [Li84], which is known to be NP-complete [Li84]. Fortunately, we can apply known heuristic routing techniques to solve this problem, but perhaps suboptimally. For example, the Lee algorithm [Lee61] is guaranteed to find the shortest path between two terminals if one exists. The problem is further complicated by the fact that VLSI layouts are quite large. Thus, careful decomposition of the problem into manageable pieces is essential.

A related issue is performance considerations. Since it is not always possible to have a one-to-one mapping from a mask description to a symbolic layout, the two may not be identical. Consequently, their performance characteristics may also change. Therefore, some clever heuristics are required to make the final result behave as close as possible to its original counterpart.

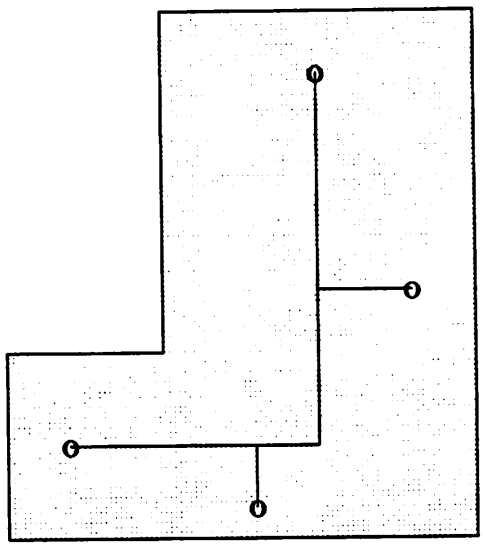


Figure 3.7: This figure illustrates some of the issues in circuit disassembly. The purpose of this figure is to show that there is no one-to-one correspondence between arbitrary polygons in a mask layout and the symbolic wires in a symbolic description. The polygon represents a typical interconnect mask region. The centerlines drawn represent desirable symbolic wires for the same connections.

Chapter 4

KAHLUA: A Circuit Disassembler

In this chapter, an implementation of the circuit disassembly technique introduced in the previous chapters is presented. The program, called KAHLUA, has been used to convert a variety of mask descriptions into symbolic layouts. The mask description is given to the program as a collection of boxes associated with different mask layers. This information is stored in an integrated object oriented database called OCT [Har86,Moo86]. The meaning of the layers and the geometry is abstracted from the programs in order to allow KAHLUA to be parameterized for different MOS technologies. The output of KAHLUA is a symbolic layout stored back into the OCT database to allow other symbolic analysis tools to manipulated the layout further. In particular, the symbolic layout generated can be processed by a spacing program such as SPARCS [Bur86,Bur87] for respacing the layout according to given design rules.

4.1 Overview

The program KAHLUA performs the following tasks. First, technology information is read to determine the meaning of the mask layers. The technology information can be parameterized to describe how different transistors and contacts are created. For example, in a typical CMOS technology, the NMOS transistor is usually determined by the intersection of the polysilicon layer and the N-diffusion layer. In addition, the technology information describes the minimum feature sizes for transistors, contacts, and interconnects. Second, the mask description is read from the OCT database, and the appropriate internal data structures of KAHLUA are constructed. Then, KAHLUA proceeds to the main procedures in the circuit disassembly process. The first one is the *device extraction* procedure. In this step, the transistors and contacts are extracted from the mask description to create symbolic circuit elements. A different circuit element is created for each unique transistor pattern and contact pattern. The second procedure is called *net decomposition*. In this step, the remaining mask geometry after device extraction are decomposed into symbolic wires. Particularly, the routing techniques, similar to those described in the previous chapter, are employed to synthesize the structure of the symbolic interconnects. Eventually, the disassembled symbolic layout is stored back into the OCT database for further processing.

In addition to the basic processing steps in the disassembly process, KAHLUA has an extra procedure for processing hierarchical designs. This procedure is used to handle hierarchical interactions between cells. Processing in KAHLUA begins at the lowest level of the hierarchy and moves upwards. The device extraction procedure is followed by the net decomposition procedure. The process is repeated through the hierarchy.

The basic algorithms and techniques for the disassembly procedures were described in the previous chapters. In the following sections, KAHLUA's implementations of these methods are described. Particularly, the device extraction step is described in Section 4.2. In Section 4.3, KAHLUA's implementation of the net decomposition procedure is presented. In Section 4.4, the

problem of disassembling hierarchically is addressed. Finally, experimental results are given in the last section.

4.2 Device Extraction

The device extraction procedure employs known circuit extraction techniques for synthesizing symbolic transistor and contact elements from the mask layout. In particular, a planesweep implementation is used for this process. While the problem of device extraction is similar to the standard circuit extraction problem, several complexities mentioned in the previous chapter are introduced that are not encountered in the circuit extraction case. The implementation of the device extraction step along with these complexities are described in the next two sections.

4.2.1 Extracting Transistors

KAHLUA recognizes MOS transistors. The active area of a transistor is assumed to be defined by the intersection of two mask layers. This active area derived represents the channel of the transistor. The regions adjacent to the channel area are assumed to implement the device terminals of the transistor. The device terminals of a transistor are the two gate terminals on both sides of the channel and the source and drain terminals. The particular mask layers involved are technology dependent. This information is abstracted from the extraction procedure so as to allow it to be parameterizable.

The primary operation in extracting the symbolic transistor elements is a Boolean mask operation for finding the intersections between two layers, namely, the AND operation. This operation is accomplished by using a planesweep technique described earlier. KAHLUA first finds the intersection of the layers involved to create the active transistor layers. Then, a separate

transistor element is created for each unique shape. In creating a symbolic transistor element, the regions adjacent to the active areas are examined to determine terminal implementations for the devices. For example, in a p-well CMOS process, the polysilicon regions adjacent to the active area represent the gate terminals, while the diffusion regions form the source and drain terminals.

In order to avoid creating unnecessary transistor elements, a hash table is kept with entries based on transistor dimensions and shapes. If a transistor element with the same features exists already, then that particular transistor element is used instead of creating a new element. Otherwise, a new transistor element is created and a new entry is entered into the hash table. Therefore, only unique transistor elements are created.

4.2.1.1 Representing Non-Linear Transistors

Although extracting transistors may appear to be quite straightforward, there are a number of complicated issues that must be addressed. In general, non-linear transistors are very difficult to represent in symbolic layouts. In fact, most symbolic layout systems either do not support them or have *ad hoc* ways for representing them. Thus far, no satisfactory representation has emerged. Consider the simple example shown in Figure 4.1. Here, we see two non-linear transistors adjacent to each other. The active area for each transistor can be determined using the planesweep method. Once the active area has been extracted, a symbolic element must be created to represent the transistor. The difficulty in creating the new transistor element is determining the correct terminal implementations for the source and drain terminals.

There are a number of possible alternatives. One alternative is to simply represent the terminal with a polygon rather than a rectangle, as shown in Figure 4.2(a). However, this would require connecting a symbolic wire to a jagged terminal. This implies that the ends of wires must also be jagged. The problem with this approach is that it is unclear how jagged wire ends and jagged terminals should be treated during spacing. It appears that representing jagged terminals

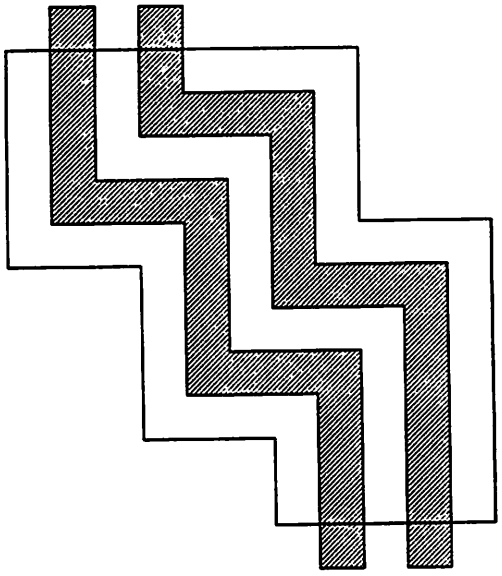


Figure 4.1: This figure shows two non-linear transistors adjacent to each other. In general, non-linear transistors are very difficult to represent in symbolic layouts.

and wire ends may not be possible since currently no symbolic layout system can support such a representation, and it is unclear whether jagged ends can be supported elegantly.

Another alternative is to represent the source and drain terminals with multiple rectangular terminals. For example, in Figure 4.2(b), the source terminal of the non-linear transistor shown is represented with five rectangular terminals. In this representation, a symbolic wire can connect to any one of these terminals. However, the width of the wire is constrained to be no larger than the width of the designated terminal. Therefore, this solution is unsatisfactory since typically the width of the wire is the same as the width of the channel area.

A variation on this approach is to use multiple wires to represent a single thick wire, as shown in Figure 4.2(c). However, this would require that a symbolic spacing program understands about merging parallel wires that are electrically connected. The spacing program SPARCS that we use does not support net merging of this type and therefore will force the parallel wires to be separated by the minimum design rule distance. One reason why a symbolic

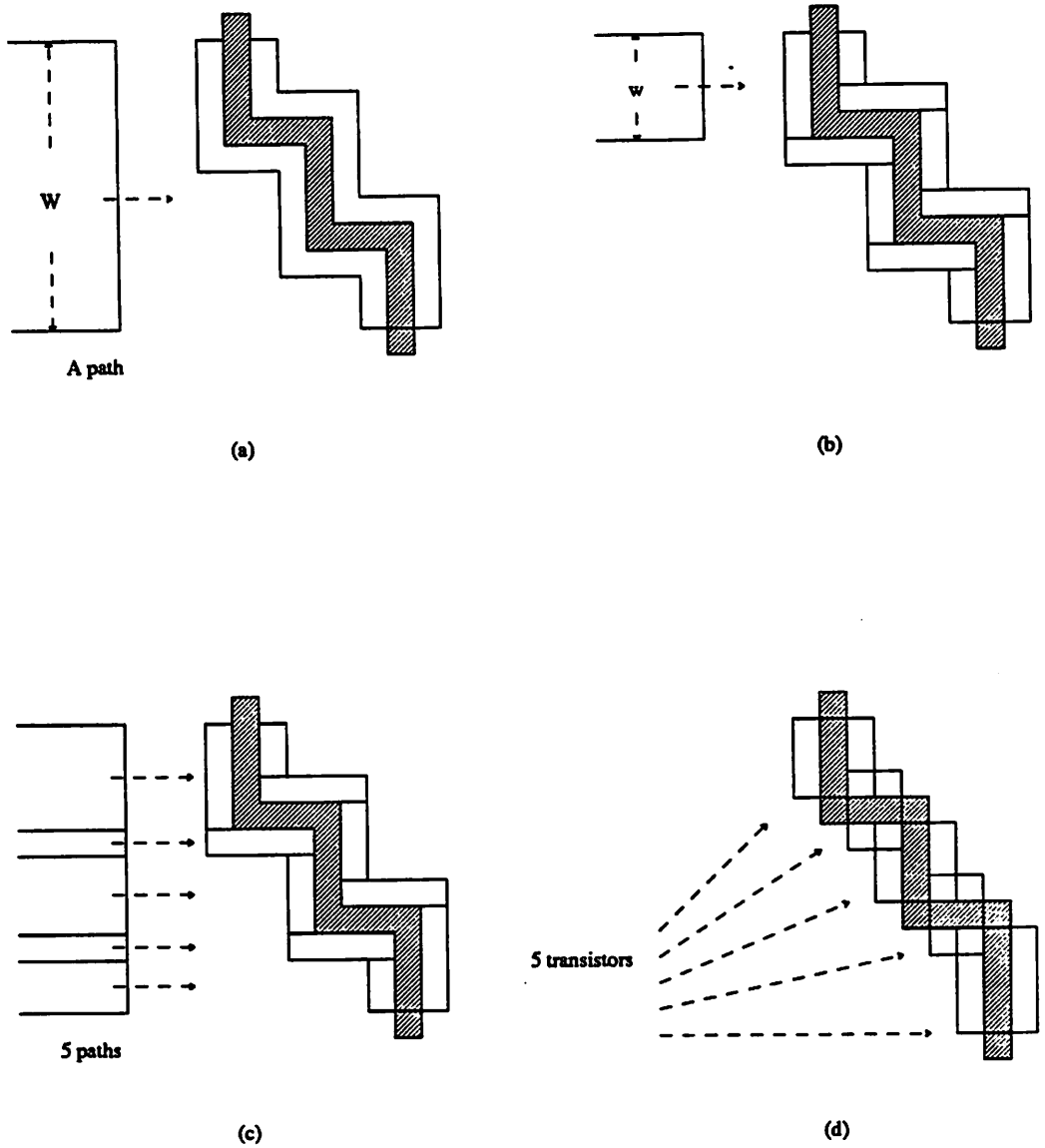


Figure 4.2: This figure shows some different ways of representing non-linear transistors. In part(a), the source and drain terminals are represented using polygons rather than rectangles. The problem with this approach is that wires with jagged ends cannot be represented. In part(b), the source and drain terminals are represented using multiple rectangular terminals. The problem here is that the width of the wire is limited by the width of the terminals. In part(c), thick wires are represented with multiple parallel wires. The problem with this approach is that a symbolic spacer may to force the wires apart. In part(d), the non-linear transistor is represented using multiple rectangular transistors. This approach is not desirable since the individual transistors may move apart during spacing.

spacing program should not support net merging of parallel wires is because merging nets is

sometimes dangerous, particular with bus lines where they have been designed to support certain current flow.

A fourth alternative is to represent each "bend" of a non-linear transistor with a separate transistor, as shown in Figure 4.2(d). Wires are then used to connect the proper device terminals. However, this approach does not seem to be feasible either. Again, as with the previous alternative, the width of the wire is constrained to be no larger than the width of the designated terminal. A more serious danger is related to the fact that the individual transistors can separate during spacing. Since they are only grouped together by wire connections, a spacing program may move them apart and result in a much wider transistor than expected. This is very dangerous since changing transistor sizes can significantly alter the behavior of the design.

4.2.1.2 KAHLUA's Representation of Non-Linear Transistors

The symbolic representation used in the SPARCS symbolic design system does not allow polygon terminal frames. Therefore, in extracting non-linear transistors, KAHLUA tries to represent the device terminals of non-linear transistors with a single box for each terminal. In this approach, the bounding box of the active area is first determined. Then the source and drain terminals are implemented with rectangles that extend from the bounding box, as shown Figure 4.3. This approach has the advantage that the symbolic wire can be as wide as the width of the channel area. Also, we do not have to worry about the "jagged end" problem described early. However, this approach does have one major drawback. The problem is that the symbolic element for a non-linear transistor is now larger, and thus some area penalty may be forfeited. In very dense mask layouts, there may not be any extra area to fit a larger transistor. Ultimately, the problem of representing non-linear transistors must be properly addressed.

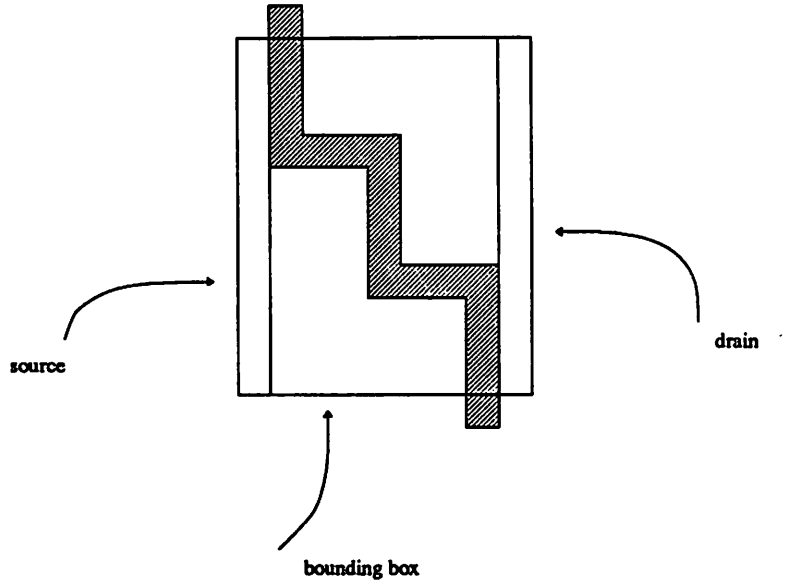


Figure 4.3: This figure shows KAHLUA's approach for representing non-linear transistors. In this approach, the bounding box of the active area is first determined. Then the source and drain terminals are implemented with rectangles that extends from the bounding box.

4.2.2 Extracting Contacts

Contacts are formed by intersecting contact cuts between two overlapping mask regions. In a symbolic layout, contact elements of different sizes can be represented. A larger size of contact element can be used to represent a collection of contact cuts that are adjacent to each other rather than having many small ones. KAHLUA exploits this feature by grouping clusters of contact cuts into larger contact elements. The technology information defining the interacting layers for contacts is parameterized.

By default, the grouping procedure is achieved by growing the cut area by half the minimum cut separation requirement and merging together any abutting or overlapping area. However, using the default merging distance may not work well enough. There may be cases where the contact cuts are separated by slightly more than the minimum cut separation. In these cases, it is still desirable to group them together. Therefore, in KAHLUA, the user can control

the merging distance by specifying it as a parameter. To prevent any illegal merging, the contact cuts that are grouped together are checked to make sure that they are indeed electrically connected. If a cluster of contact cuts are not all electrically connected, then only those that are electrically connected will be grouped together.

The grow and merge operations used for grouping contact cuts are also implemented using the planesweep algorithm. In creating a symbolic contact element, the two interacting layers along with the cluster of contact cuts are implemented. A rectangular terminal the size of the contact element is then defined to represent the connection point for the contact.

Before creating a new contact element, a hash table is first checked to see if a new element is needed. The hash table is organized with entries based on the dimensions of the contact elements and the number of cuts each contact contains. If a contact element with the same features exists already, then that particular element is used instead of creating a new one. Otherwise, a new contact element is created and a new entry is entered into the hash table. Therefore, only unique contact elements are created.

4.3 Net Decomposition

After device extraction, the active areas of transistor elements are subtracted from the original mask layout. The remaining geometry is assumed to be used for interconnection and is therefore decomposed into symbolic wires. These symbolic wires are used to interconnect the symbolic elements discovered in the device extraction phase. This is the task of the net decomposition procedure. In this section, we will first consider various possible alternatives for solving this problem and discuss their limitations. Next, we will describe KAHLUA's view of the problem which is equivalent to finding Steiner trees. The Steiner tree problem is solved using conventional routing methods with several minor enhancements. Then, we will present an overview on how

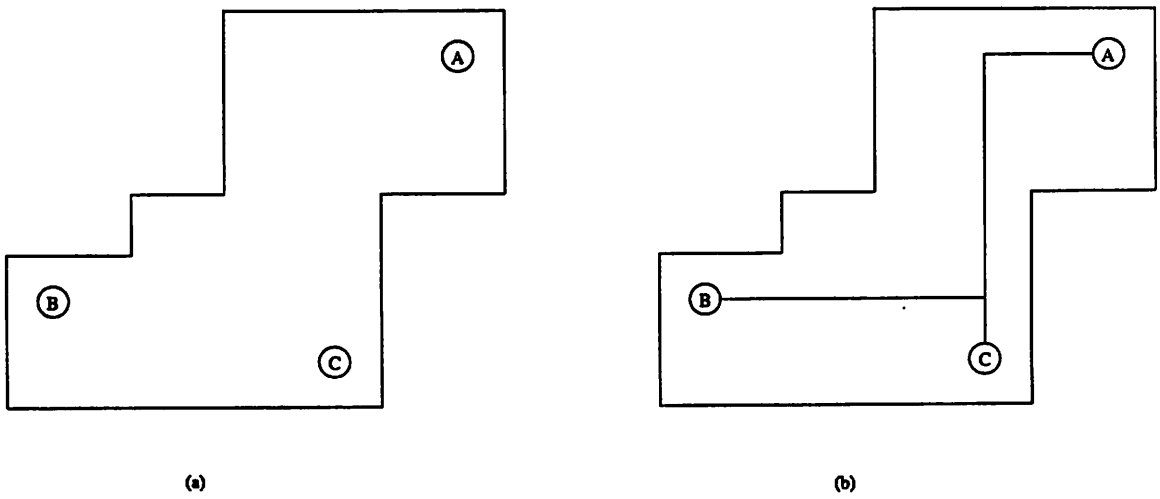
KAHLUA implements the net decomposition procedure. Finally, we will describe the various steps involved in solving the net decomposition problem.

4.3.1 Alternative Approaches to the Net Decomposition Problem

Before describing KAHLUA's approach to the problem, let us first consider several possible alternatives and examine their deficiencies. Consider a simple example shown in Figure 4.4. Here, the net decomposition procedure must derive a set of symbolic wires for connecting the three device terminals as shown. Ideally, the set of wires derived should resemble the topology of the original mask region while avoiding the introduction of any unnecessary symbolic wires or jogs. This is required partly for aesthetic reasons. Designers frequently want the converted designs to look as closely as possible to the original mask layout. But more importantly, it is usually critical to maintain similar performance characteristics to the corresponding mask design. Therefore, the choice of net composition method should support these criterions. Below, we will describe several possible alternative solutions and show why their drawbacks.

One naive approach would be to simply "shrink" the polygons in hopes of discovering some kind of centerlines. However, it should be obvious that this approach would not work since a uniformly reduced polygon will not give any additional hints about the topology of the interconnects than the original polygon. A more sophisticated method of shrinking might be to iteratively shrink parts of the polygon. In any case, even if centerlines can be discovered, they do not necessarily connect the terminals. Effectively, a routing procedure must follow to connect the terminals to the centerlines.

Another possible approach is to first decompose the polygons into rectangles, as shown in Figure 4.5. In this approach, we try to use the centerlines of the rectangles to represent symbolic wires. However, determining which direction of the rectangle to draw the centerline is very



(a) (b)

Figure 4.4: This figure illustrates a net decomposition example. The purpose of the net decomposition procedure is to derive a set of symbolic wires for representing a net region. In this example, symbolic wires must be derive for connecting the three terminals as shown.

difficult. This depends very much on what terminals need to be connected and the direction of current flow. Assuming that we can determine which centerlines to use, we still have the problem of joining the terminals and the centerlines together to form a valid set of symbolic wires. This is not obvious. In fact, to derive the solution shown in Figure 4.4, some kind of ad hoc routing method would need to be employed. In any case, no systematic solution based on this idea appears to be feasible.

In a third alternative, we can try to trace the contour of the polygons to determine the symbolic wires. In fact, this may be done during the device extraction step using the planesweep algorithm. At first glance, this appears to be an attractive solution. Consider the example shown in Figure 4.4. The basic idea here is to start at the leftmost terminal and sweep a vertical scan line across the polygon. Between the two parallel edges that the vertical scan line intersects, we try to trace a centerline. As the vertical scan line sweeps across the polygon, the separation between the parallel edges will change in distance. To follow the contour of the polygon, we can try to trace the path along the centers of these separations. When a terminal is encountered by the

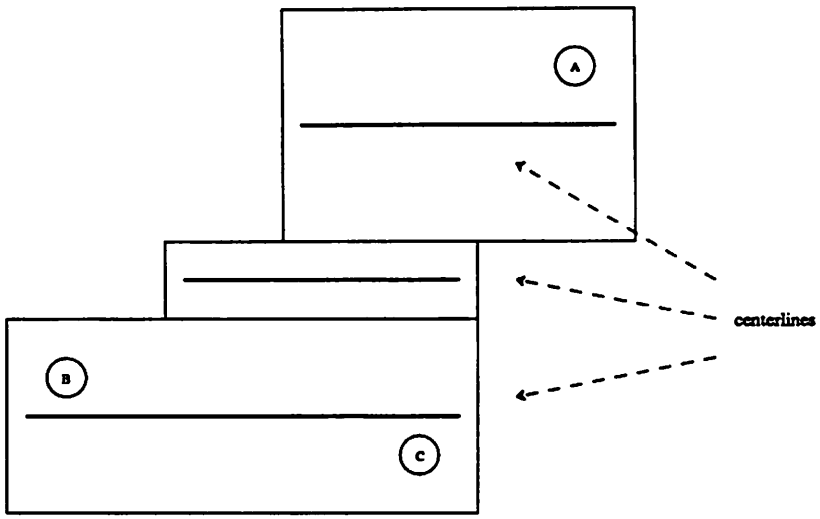


Figure 4.5: This figure illustrates the approach of first decomposing into rectangles and then trying to use the centerlines of the rectangles to represent the symbolic wires.

scan line, a path can be drawn from the current centerline to the terminal in order make a connection. The final solution for the example shown in Figure 4.4 is given in Figure 4.6. The advantage of this approach is that it works and it is quite efficient. Effectively, it is doing some kind of routing. However, the solution that it tends to produce is not very aesthetically pleasing. In particular, the solutions tend to have many unnecessary symbolic wires and jogs. Although a "cleanup" procedure could be employed, it was felt that a more systematic solution would be preferable. In fact, known efficient routing methods can be employed to solve this problem. This is indeed what KAHLUA tries to do, as described in the next section.

4.3.2 KAHLUA's Approach to the Net Decomposition Problem

KAHLUA tries to solve the net decomposition problem by using conventional routing methods with some minor enhancements. KAHLUA views the problem as equivalent to finding Steiner trees for representing the symbolic wires. The tree solution is used to connect the terminals of a

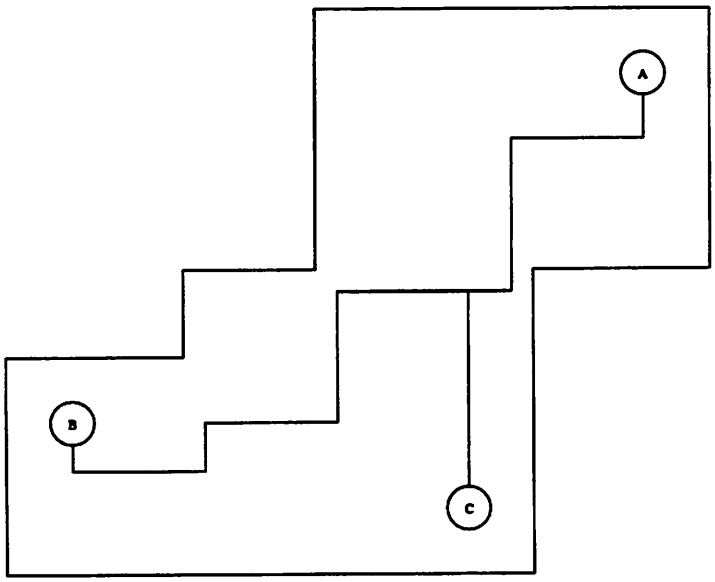


Figure 4.6: This figure illustrates the approach of tracing the contour of the polygon to discover symbolic wires.

given net while being constrained to the area defined by the original mask geometry. However, since VLSI layouts are often quite large, finding the Steiner trees for connecting all the terminals in the circuit can be very complex. Therefore, it is necessary to partition the problem into more manageable tasks.

In KAHLUA, the net decomposition procedure is implemented in two steps. The first step, called *global partitioning*, is used to partition the geometries on the different mask layers into smaller manageable regions. In this step, the partitioner also determines the terminals that each region intersects. The second step is a *routing* procedure for finding a Steiner tree solution for each partitioned region. The routing solution must be confined to the boundaries of the original mask geometry. The implementation of these two steps are described in the sequel.

4.3.2.1 Global Partitioning

A natural partition exists between the geometry on different mask layers. Interacting layers forming active circuit structures were already extracted in the device extraction step. The remaining interconnect layers only interact through contact vias. Therefore, the geometry on the different interconnect layers can be decomposed independently. This *planarization* of the problem simplifies considerably the net decomposition procedure. For each interconnect layer, KAHLUA further partitions the geometry by fracturing them into non-overlapping polygons. In some cases, these regions are quite small. As with other geometric operations, this fracturing step is consistently implemented using a planesweep algorithm.

For each partitioned region, the terminals that the area intersects are extracted. Each region along with the corresponding list of terminals defines a separate routing problem, which can be stated as follows:

> Given an arbitrary rectilinear region R, with corners defined by a set of vertices, V_i, for i=1,2,...,n, implement the connections specified by the terminal list T_j, for j=1,2,...,m. where T_j is contained within the boundaries of R.

4.3.2.2 The Routing Procedure

The primary objective of the routing procedure is to find a feasible rectilinear tree for connecting the terminals in a given region. This step is implemented using well-known routing algorithms. Although the task of finding symbolic wires is formulated as a routing problem, there are several subtle differences that introduce some complexities not encountered in the traditional problem. One of these differences is that the tree solution should resemble the topology of the original mask region while avoiding the creation of any unnecessary symbolic wires or jogs. This is

desirable for both aesthetic and performance reasons. Hence, special care needs to be taken when implementing the routing algorithm.

The first step in KAHLUA's routing procedure is to map the routing area and the terminal locations onto a two-dimensional grid. This mapping is extremely crucial to the quality of the final solution. Consider the routing region illustrated in Figure 4.7. A normal grid representing the coordinates of the routing area is shown in Figure 4.7a. Between two adjacent vertices, there are $n+1$ grid lines shown along the x axis, where n is the Euclidean distance between the two points. In the traditional routing problem, a path solution can potentially be along any one of those grid lines. However, we are interested in solutions where the paths ideally represent the *centerlines* of the routing region.

We can heuristically encourage such solutions by using the idea of a *virtual routing grid* instead of the standard coordinates. In a virtual routing grid, only significant x and y locations are constructed on the grid. In particular, the locations of terminals and the vertices of the routing

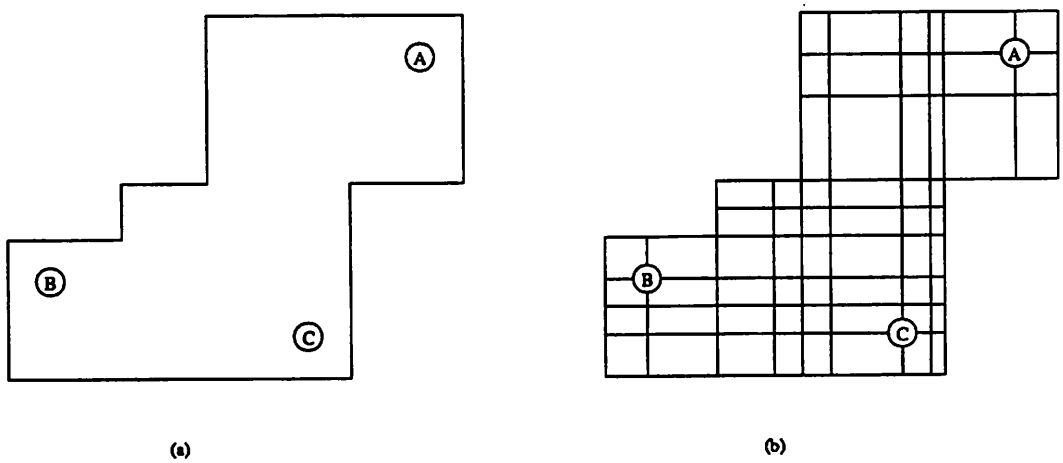


Figure 4.7: This figure illustrates the idea of a virtual routing grid. A normal grid representing the coordinates of the routing area is shown in Figure 4.7a. The corresponding virtual grid mapping is shown is Figure 4.7b. The virtual routing grid only contains significant coordinates, which enables heuristically better solutions and achieves higher efficiency.

region are considered. The polygon region is first decomposed into rectangles. For each rectangle, we consider the x and y coordinates for all four vertices as well as the center of the rectangle. Also, we add the coordinates of the terminals contained in the polygon. The corresponding virtual grid mapping for part(a) in Figure 4.7 is shown in part(b). These virtual coordinates are sufficient to guarantee that a Steiner tree solution can be found for connecting the terminals.

Another benefit of the virtual routing grid approach is that it avoids any unnecessary search. Since the complexity of the routing algorithm is proportional to the number of grid points, the run time on the reduced routing space is significantly improved. Moreover, the limited search space helps the router to avoid inserting any unnecessary jogs.

Once the virtual routing grid has been constructed, the next step is to find a routing solution. The routing region is defined by boundaries of the polygon. Therefore, the routing solution must be confined to the interior of the polygon. The routing approach taken by KAHLUA is primarily based on a single growth wave propagation technique developed for general area routing [Lee61]. This algorithm can be shown to have a worst case time complexity of $O(p)$ for connecting two points, where p is the number of grid points in a two-dimensional routing grid. In this method, we begin by starting at a terminal location. The basic idea is to expand a wave front from this terminal until another terminal is found. Next, the algorithm backtracks to find a feasible path. This path represents a simple tree for connecting the two terminals. Then, a wave is propagated from the current tree until yet another terminal is reached. This process is repeated until all the terminals are connected. The advantages of this method are its inherent generality and high quality results.

In addition to the wave propagation technique, a special template lookup router is also implemented in KAHLUA to efficiently solve the trivial routing problems. It attempts to route regions by matching the problem with a set of predefined lookup templates. For instance, in a two terminal case, a "L" shape route or a straight line route is tried first. In the three terminal

case, a "T" shape structure is attempted (see Figure 4.8).

In KAHLUA, only the four templates shown in Figure 4.8 are implemented. Empirically, a reasonable number of the routing regions can be solved using these simple templates. This is particularly the case for abutting symbolic elements where only a simple wire is needed. The lookup table is organized with entries defining the number of terminals that each template supports. In each template, the routing solution is stored in terms of edges and their relative position. KAHLUA tries to match the pattern to the current routing problem and determines if the pattern can fit within the boundaries of the polygon. For more complicated problems that cannot be solved using this simple technique, the general routing strategy is used.

As a final step, after a tree solution is obtained, actual symbolic wires must be generated. Ideally, the paths are located relatively in the center of the region. For each branch in the tree, a symbolic wire is generated. A symbolic wire is defined to be a segment between two points with a given width. The width of each wire is determined by examining the perimeter of the routing region and selecting the maximum possible width that satisfies the constraints enforced by the region boundary. An example illustrated in Figure 4.9 demonstrates this problem. In addition, jogs are inserted corresponding to each Steiner node in the tree. As a final step, global connectivity between all electrically connected symbolic wires and circuit elements must be established. The routing step just described will create a set of net clusters each corresponding to a different partitioned routing region. The connectivity between the net clusters can be determined by merg-

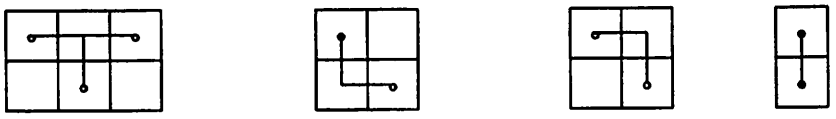


Figure 4.8: This figure illustrates some predefine routing templates for simple cases. A quick lookup strategy is used to solve trivial routing problems.

ing multiple net clusters connected to the same terminals.

4.4 Disassembling Hierarchical Designs

In VLSI layouts, the design is typically captured hierarchically. For example, in a microprocessor, the design is usually defined in terms of macro blocks such as data paths and control paths. These macro blocks are in turn defined in terms of smaller components such as functional units and registers, and these components are in turn defined in terms of yet smaller components like logic gates, and so on. Therefore, using hierarchy is a natural way of capturing a design. In fact, most VLSI designs are highly regular in nature, containing many instances of the same cell. For example, data paths are typically two dimensional arrays with only a few unique cells replicated many times. Thus, it is often possible to improve the performance of a CAD tool significantly by

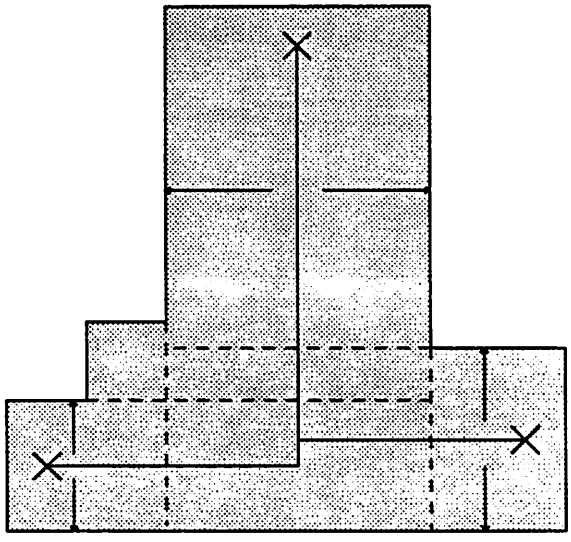


Figure 4.9: This figure shows how wire widths are determined after their centerlines have been derived. The basic idea is to make them as wide as possible while not violating the boundary constraints.

processing each cell definition only once. In addition to the savings in computation such an approach offers, it is important for circuit disassembly to maintain the hierarchy during the conversion process so that the cells can be used separately once they have been disassembled. Moreover, it is usually desirable to maintain the structure of the original layout as it was intended. In Figure 4.10, a hierarchical example is shown.

In a mask layout, connections between cells are defined by their overlap or by creating pieces of geometry for connecting them. In a symbolic layout, *formal terminals* are defined for a cell so that it can be used in a hierarchical design [Bal82,Kel84]. Formal terminals can be thought of as connection ports for the cell. To use a cell in a hierarchical design, connections to the cell

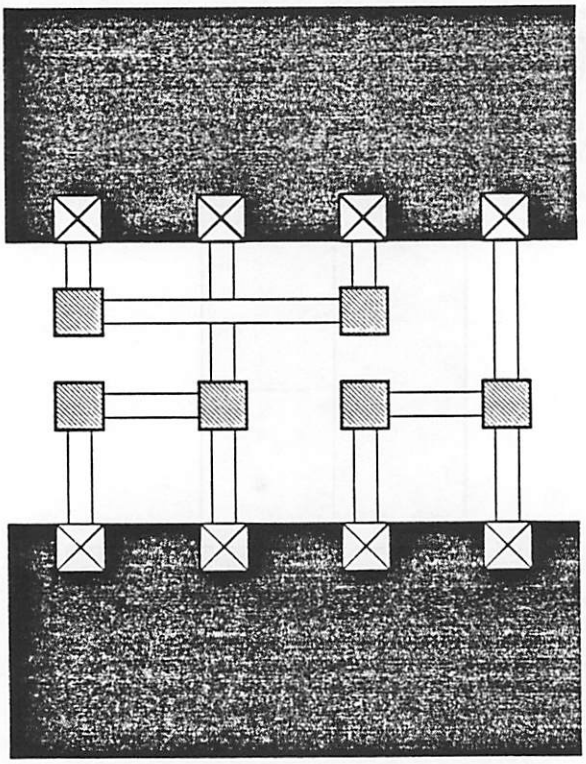


Figure 4.10: In this figure, a hierarchical example is shown. Both hierarchical objects and cell level circuit elements are represented using the same abstraction.

must be made only to the formal terminals. In most symbolic layout systems, these formal terminals are required to lie on the border of the cell. This restriction simplifies the spacing problem considerably. In general, KAHLUA expects the user to define the formal terminals for the cells. Once defined, KAHLUA treats these terminals like any other terminal. Thus in a hierarchical design, KAHLUA will generate symbolic wires for interconnecting these terminals. However, if the formal terminals are not specified, then the user can optionally ask KAHLUA to synthesize them.

In general, the problem of synthesizing formal terminals from a mask-level design is extremely difficult. The problem is difficult partly because what constitutes a formal terminal for a cell depends very much on the usage of the cell. Particularly, a formal terminal is needed for every overlap or abutment that occurs between interacting cells. Since connections between cells in a mask layout are done by overlapping geometry, the locations of formal terminals can be arbitrary, even somewhere in the center of the cell.

Therefore, KAHLUA does not try to synthesize *all* formal terminals. Instead, it will only heuristically examine the border area for possible implementations. Hence, it does not guarantee that all the necessary formal terminals will be discovered. For the ones missing, the user can interactively define them. The basic idea is to search around the boundary of the cell to determine what pieces of geometry are potential candidates for implementing formal terminals. This is done by creating a frame around the border of the cell. Each rectangle bordering the boundary is checked for its aspect ratio. If the shorter side of the rectangle is bordering the boundary, then KAHLUA will assume that a formal terminal is needed there, as shown in Figure 4.11. For example, the ends of the power rails shown in Figure 4.11 would be considered as formal terminals.

For processing hierarchically, KAHLUA recursively disassembles the hierarchy by depth-first traversal. This is accomplished by first disassembling each subcell and then registering all

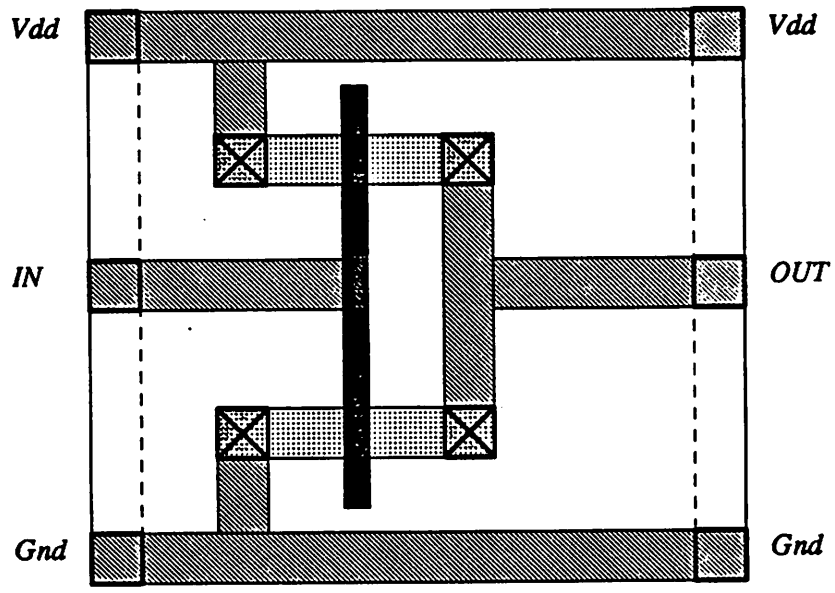


Figure 4.11: This figure illustrates the problem of synthesizing formal terminals. Formal terminals must be generated for symbolic layout conversion. The basic idea is to search the boundaries of the cell for likely candidates.

the connection terminals of the subcells at the parent level so that KAHLUA can generate symbolic wires for connecting them. In addition, the implementations of these terminals are added to the parent level as possible routing area. Experimental results show that hierarchical disassembly significantly reduces CPU expenditure, as illustrated in the next section.

4.5 Experimental Results

In this section, performance measurements for the program KAHLUA are presented. A set of disassembled examples before and after spacing are shown. These examples are used to illustrate that the spaced layouts are able to maintain reasonable density as compared with the original highly-tuned mask layout on most of the examples tested. This is important since many designers still do not trust symbolic design systems to produce dense layouts. Next, a set of examples to

Before and After Spacing							
	width	unspaced height	area	width	spaced height	area	area ratio
Dff	154	158	24,332	128	144	18,432	0.76
DT540	93	170	15,810	97	178	17,226	1.09
DT509H	663	241	159,783	665	238	158,270	0.99
DT522	427	236	100,772	421	238	100,198	0.99

Table 4.1: This table presents results for symbolic layouts before and after spacing. The symbolic layouts were generated from mask descriptions using KAHLUA. The purpose of this experiment was to determine the area penalty, if any, that would be incurred as a result of spacing. On the examples tested, the size of the spaced layout is about the same as the original mask layout, as shown in column eight. Note that it is not expected that the spaced layout will be much smaller than the original layout since the original was highly tuned by an experienced designer.

evaluate the performance of KAHLUA on flat disassembly and hierarchical are described. These examples illustrate the improvements obtained by exploiting hierarchy.

Results on other examples are summarized in Table 4.1. The name of the circuit, the size measurements for each example before the spacing and after are given. On most of the examples tested, the size of the spaced layout is about the same as the original layout, as shown in last column of the table.

The execution times for flat disassembly on some typical library cells are given in Table 4.2. The name of th cell, the size of the cell and the corresponding execution times are given. These examples were drawn from an industrial cell library. Typical disassembly time is about 30 CPU-seconds.

If the mask layout was originally captured hierarchically, KAHLUA can take advantage of this hierarchy to reduce computing requirements. The performance of KAHLUA on a set of hierarchical examples is presented in Table 4.3. The execution times for both disassembling hierarchically and flat are given. The speed improvements are dependent on the nature of the design examples. These examples demonstrate that performance improvements can be achieved

Cell Level Designs					
cell	#fets	#contacts	#connectors	#wires	cpu-time
DT509H	40	70	227	362	40.0
DT509L	40	78	122	261	33.2
DT510H	41	77	136	277	37.5
DT510L	41	77	135	275	36.0
DT511HE	24	48	76	158	14.9
DT511HO	24	47	71	152	13.8
DT511LE	24	49	76	160	14.8
DT511LO	24	47	71	152	14.0
DT518H	32	65	106	233	20.7
DT519H	32	61	111	236	21.5
DT519L	32	62	109	235	21.5
DT522	24	51	135	239	15.3
DT540	4	15	35	57	2.6

Table 4.2: This table presents the flat execution times of the circuit disassembly process using KAHLUA on a number of industrial examples. These results were collected on a DEC VAX 8650 running ULTRIX. The area figures are reported in λ and the times are in cpu-seconds. The time figures also include all database related processing.

Hierarchical Designs						
cell	#fets	#wires	#instances/ #cells	hierarchical cpu-time	flat cpu-time	speed ratio
hex1	532	4,767	38	700	9,244.3	0.08
hex2	392	3,080	28	151	3,306.4	0.04
hex3	392	3,571	28	495.6	4,996.3	0.10
hex4	210	1,860	15	123.3	1,341.2	0.09
hex5	168	1,494	12	83.4	923.1	0.09

Table 4.3: This table presents the execution times of the circuit disassembly process using KAHLUA on hierarchical examples. Both hierarchical execution times and flat execution times were measured. These results were collected on a DEC VAX 8650 running ULTRIX. The area figures are reported in λ and the times are in cpu-seconds. The time figures also include all database related processing.

by exploiting hierarchy.

All the CPU times given in this section are on a DEC VAX 8650 running ULTRIX. The area figures are reported in square λ and the times are in CPU seconds. The times expended also

include all database related processing.

Chapter 5

Circuit Extraction For Symbolic Layout

The problem of circuit extraction is to convert a layout description into an electrical network of transistors, resistors, capacitors, and interconnecting wires, suitable for use by a variety of electrical simulation and verification tools.

The problem of extracting from a symbolic layout rather then from a mask-level layout is considerably simplified because connectivity and transistor locations are already known. The remaining task is to derive the parasitic resistance and capacitance in the layout. Hence, the execution time required to extract from a symbolic layout is faster than from a mask-level layout. In fact, the savings in execution time can be significant since most mask-level layout extractors typically spend over 50% of the time extracting connectivity and transistor information. This is one of the major advantages of symbolic layout. Since circuit extraction is frequently done during the verification cycle of the design, execution speed is of critical importance.

In this chapter, a simple circuit extractor called OCTEXTRACT written for use on symbolic layouts represented in the OCT database is described. The input to OCTEXTRACT is a symbolic layout rather than a mask-level description. The purpose of OCTEXTRACT is twofold. First, it demonstrates how the problem of circuit extraction is simplified when the starting layout description is symbolic rather than mask-level. Second, it provides a reasonably powerful, but simple, extractor that can execute quickly. The electrical network generated by OCTEXTRACT is stored back into the OCT database. Another translation program called OCTEXT2SPICE is used to convert the OCT database format into the SPICE format for simulation purposes [Nag73].

The remainder of this section is organized as follows. First, different circuit models for various simulation and verification tasks will be examined (ie. what information is present in the circuit description produced by an extractor). Second, the extraction procedure is described. Finally, examples are presented and performance measurements are given.

5.1 Circuit Models

The purpose of a circuit extractor is to extract pertinent information from a layout necessary for simulation and verification. For example, one possible application is to use the extracted connectivity of a layout and compare it with an independently drawn schematic to verify that the two circuit descriptions are the same [Spi85]. For logic verification, the extracted transistor net list can be simulated to ensure that the circuit functions as intended [Bry81,Ter83]. For performance analysis, a timing analyzer can be used to compute the delays through a circuit to determine whether it meets the performance requirements [Ous83]. Finally, for an in-depth circuit analysis, a detailed simulator can be used to analyze the waveforms propagating through the circuit [Nag73,New83].

The amount of information that an extractor should produce depends on the simulation or verification task that needs to be performed. In the case of logic simulation or net list comparison, where only a transistor net list is required, the information is adequately contained in a symbolic layout itself, and thus no additional extraction is required. In the case of a timing analysis tool or a detailed waveform simulation tool, resistance and capacitance of interconnecting wires must also be obtained.

Two basic circuit models can be used to represent an electrical network of transistors, resistors, and capacitors. One approach is to describe the circuit as a detailed RC network where a resistance and a capacitance value is computed for each individual symbolic wire in the layout, as shown in Figure 5.1. For example, this detailed model is used in [Bas83,McC84]. The advantage of this approach is accuracy, but it is unnecessarily bulky for some verification requirements.

A simpler approach is to lump each group of symbolic wires that are electrically connected into a single electrical *node*. Each node includes a "lumped" parasitic resistance and a lumped parasitic capacitance to ground, as shown in Figure 5.2. This simple lumped model has been used by a number of extractors, such as MEXTRA [Fit82] and MAGIC [Sco85], and simulation

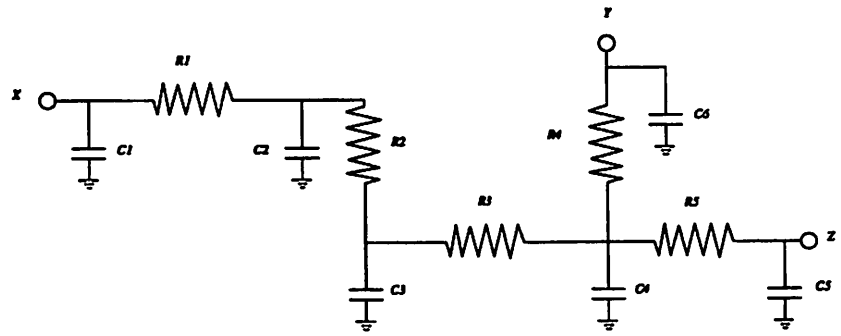


Figure 5.1: This Figure illustrates the detailed RC network model. A resistance and a capacitance value is computed for each individual wire segment associated with a net. This approach has the advantage of accuracy, but is unnecessarily bulky for some verification requirements.

tools, such as CRYSTAL [Ous83] and ESIM [Ter83]. Though quicker simulation and more concise representation is possible using the lumped model, accuracy may be forfeited.

Both models are useful depending on verification needs. The tradeoff between the two models is accuracy versus storage size and simulation time. Therefore, the extractor described in this chapter can optionally produce either circuit model.

5.2 Information Extracted

In this section, the extraction procedure, which includes extracting transistors, resistance, and substrate capacitance from a symbolic layout stored in OCT, is described. In general, the task of circuit extraction is a complicated problem.

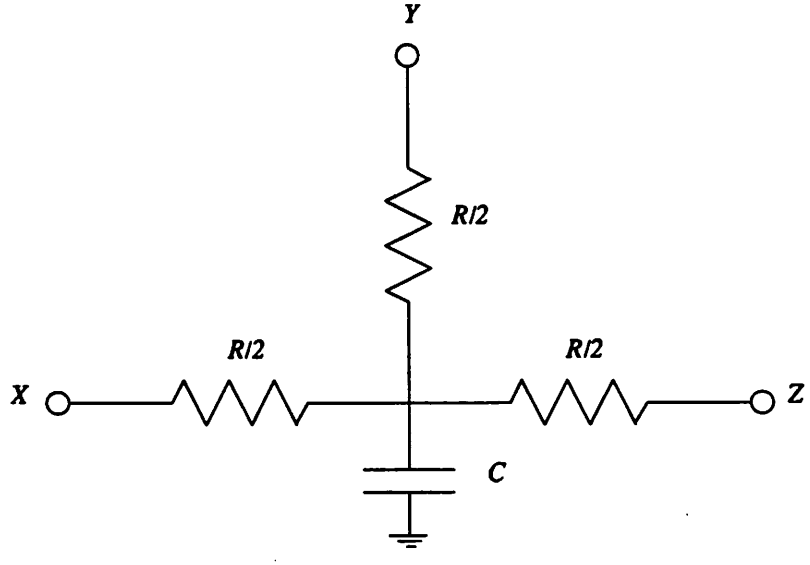


Figure 5.2: This Figure illustrates the simplified lumped RC model. In this model, all the resistance and capacitance values on a given net are lumped together into a single lumped parasitic resistance and a single lumped parasitic capacitance. Though quicker simulation and more concise representation is possible using the lumped model, some accuracy is forfeited.

Since transistors are already represented in the symbolic layout, they are simply translated to transistor elements in the extracted output with widths and lengths in actual microns (as oppose to database units). For the "non-coupling" parasitics, OCTEXTRACT computes a parasitic capacitance to ground and a resistance value for each symbolic wire in the layout. Thus, a highly detailed RC network is produced. If the user desires the simpler lumped model, then the parasitic values for a group of wires in a single electrical node is accumulated.

For substrate capacitance, OCTEXTRACT computes both the area capacitance and the perimeter capacitance. Each type of mask material has a different capacitance to ground per unit area and a perimeter capacitance per unit length. Resistance is computed using the length to width ratio of a wire and multiplying it to the sheet resistance value associated with the corresponding mask layer.

Cell Level Designs			
cell	#fets	#wires	cpu-time
DT509H	40	367	12.0
DT510H	41	277	8.5
DT518H	32	233	7.1
DT522	24	239	7.3
DT540	4	57	2.0

Table 5.1: This table presents the execution times of the symbolic extraction process using OCTEXTRACT on a number of cell level designs. These results were extracted on a DEC VAX 8650 running ULTRIX. The times are in cpu-seconds. The time figures also include all database related processing.

5.3 Examples

In this section, performance measurements for the program OCTEXTRACT are given. The times presented are CPU seconds collected on DEC VAX 8650 running ULTRIX. The reported times include transistor conversion, parasitic resistance extraction, substrate capacitance extraction, and all database related processing.

Several cell level design examples were extracted and the results are summarized in Table 5.1. The name of the circuit, the number of transistors and symbolic wires in the circuit, and the execution time required for each example are given. Since transistors and connectivity information are explicitly represented in a symbolic layout, most of the time is spent on extracting parasitics. On average, the execution time was about 32 wires per CPU second.

If the symbolic design was captured hierarchically, OCTEXTRACT can take advantage of this hierarchy to reduce computing requirements. In Table 5.2, several hierarchical design examples are presented. The name of the circuit, the number of transistors and symbolic wires in the circuit, the ratio between the number of instances to the number of unique cells, and the CPU

Hierarchical Designs						
cell	**#fets**	**#wires**	**#instances/ #cells**	**hierarchical cpu-time**	**flat cpu-time**	**speed ratio**
hex1	532	4,767	38	27.4	104.7	0.26
hex2	392	3,080	28	14.0	63.1	0.22
hex3	392	3,571	28	22.6	73.7	0.31
hex4	210	1,360	15	12.1	40.2	0.30
hex5	168	1,494	12	10.3	30.7	0.34

Table 5.2: This table presents the execution times of the symbolic extraction process using OCTEXTRACT on hierarchical designs. Both hierarchical execution times and flat execution times were measured. These results were collected on a DEC VAX 8650 running ULTRIX. The times are in cpu-seconds. The time figures also include all database related processing.

times for hierarchical and flat extraction are given. The speed ratio of extracting hierarchically versus flat is given in the last column. Note that the speed improvement is dependent on the nature of the experimental circuit. The purpose of these examples is merely to demonstrate that performance improvement can be achieve by exploiting hierarchy.

Chapter 6

Conclusions

In this report, a new technique called circuit disassembly that is used to convert mask-level descriptions into the equivalent symbolic layouts has been presented. Among its contributions, this technique provides an important link between mask level design systems and symbolic layout systems. Now layouts generated from one environment can be converted to the other. In addition, once converted into a symbolic form, the layout can be further manipulated by other symbolic layout and analysis tools. In particular, a compaction program, such as SPARCS, can be invoked to respace the layout according to new design rules.

The circuit disassembly problem was formulated as two separate but related problems, namely, device extraction and net decomposition. These two topics were the main focus of the report. The first was the problem of extracting devices from mask layouts. The device extraction procedure is performed on the layout using known geometric manipulation algorithms. These algorithms have been previously applied to circuit extraction and design rule checking problems. A detailed discussion on these techniques was given in Chapter 3. Of the algorithms, the planesweep method was chosen and implemented in KAHLUA. This technique allows for flexible mask operations such as finding intersections of layers, as well as GROW and MERGE

operations. In addition, the computation complexity is equal or better than other known algorithms for performing complex mask operations.

The second topic addressed was the problem of net decomposition. Net decomposition is performed on interconnect mask regions to decompose them into symbolic wires. The problem is similar to finding a Steiner tree for interconnecting terminals in a given constrained region. Hence, known routing strategies were employed to solve the problem. Several general routing procedures were examined in Chapter 3. The method implemented in KAHLUA is a variation on the well known Lee Algorithm [Lee61] based on single growth wave propagation. Though similar, net decomposition is different from the traditional routing problem in that the objective functions are not entirely the same. The differences stem from the need to maintain performance characteristics when converting from a mask layout to a symbolic layout. These considerations are taken into account by heuristically optimizing for solutions that most resemble the original mask regions as well as avoiding any unnecessary wires and jogs. In addition, a template lookup technique was used to solve simple routing problems. Since VLSI layouts are often very large, partitioning the problem into smaller manageable tasks is a necessity. Therefore, a global partitioning procedure was implemented to perform that role.

Several areas remain for future research on the circuit disassembly problem. In this report, a formulation of the circuit disassembly problem based on the synergy of known extraction and routing algorithms has been described. In particular, a framework for circuit disassembly has been developed. With this framework, different extraction and routing techniques other than the ones implemented in KAHLUA can be explored. For example, a fast algorithm for circuit extraction based on the corner stitching data structure [Ous84] was described in [Sco85]. This technique has the potential of increasing the efficiency of the device extraction step.

For the net decomposition problem, other routing schemes such as the line search algorithm described in [Hig79] can be applied. Using the corner stitching representation, the routing algo-

rithm can be implemented on the same data structure as the extraction procedure, and hence potential savings in execution time and storage requirements can be achieved, by not creating different data structures for each operation. Also, it would be interesting to see if the net decomposition problem can be solved in conjunction with the device extraction step. In particular, perhaps symbolic wires can be derived during the planesweep procedure as interconnect geometry are examined.

Another problem that requires further examination is the disassembly of complex circuit structures. KAHLUA has been designed for converting standard CMOS layouts. In particular, it currently only understands MOS transistors and basic contacts. For example, other creative structures such as MOS capacitors, contacts stacked on top of gates, and bipolar/ECL components are not currently recognized by KAHLUA. To identify and extract these structures, the device extraction algorithm may be extended by incorporating special "rules". In general, identifying such structures in free-form manual designs is a complex pattern recognition problem, and KAHLUA does not solve this problem today.

Another area that offers interesting research is the representation of rectangular polygon terminal frames. Although the planesweep method described in this report is capable of recognizing them, representing rectangular polygon terminals frames in a symbolic form is still an issue. Currently, terminal frames must be represented with boxes. This limitation makes representing non-linear transistors translated from mask layouts extremely difficult. Besides being inefficient at times, it is impossible in some cases to represent non-linear transistors without rectangular polygon terminal frames. Developing a robust representation along with a compaction procedure that can support and exploit complex rectangular polygon terminal frames represents a major engineering challenge in this research area.

In addition, the problem of disassembling hierarchical designs still represent some interesting challenges. In particular, the problem of synthesizing formal terminals for hierarchical

objects such as macro cells should be further explored. This is still relatively an open problem. If formal terminals are not already specified in the database, KAHLUA currently only attempts to find terminals along the cell borders. A more uniform treatment of this problem is desirable.

Appendix

Manual Pages

Following are the UNIX manual pages for the programs developed for this Masters Thesis. They are as follows:

- **kahlua:** A hierarchical circuit disassembler as described in the text of this report. It accepts an OCT mask layout and synthesizes an appropriate OCT symbolic layout.
- **octextract:** Circuit extraction program for symbolic layouts as described in the text of this report. It generates input descriptions to circuit simulators for debugging and verification.
- **octext2spice:** OCT database format to spice input format translator.

NAME

kahlua – A Hierarchical Circuit Disassembler

SYNOPSIS

kahlua [options] cell[:view]

DESCRIPTION

Kahlua is a hierarchical circuit disassembler program for synthesizing symbolic layouts from mask level descriptions. Its input is an oct cell of viewtype *physical*, and its output is another oct cell of viewtype *symbolic*. To represent the necessary transistors, contacts, and connectors in the symbolic layout, a set of physical leaf cells is created. In addition, a statistics file is created at the end of execution that summarizes the program's output characteristics. Once a mask layout is disassembled by *kahlua*, it can be used with a rich collection of available symbolic design and verification tools.

The command line options are:

-o cell[:view]
Specifies an alternate output. The default output cell name is inherited from the input cell. The default output view name is *symbolic*.

-l leafview
Specifies a different view name for the leaf cells. The default view name for the leaf cells is *physical*.

-t techfile
Specifies an alternate technology file for design rules (see below for default).

-e expand
Expands the design rules by *expand* factor.

-p directory
Specifies an alternate directory to store the leaf cells. The default directory is *<outcell>/<outview>*.

-a Specifies a maximum aspect ratio for the paths.

-s statname
Specifies an alternate statistics file. The default is a file named *kahlua.stats* located in the *primitive directory* (see above).

-g Ignores explicit formal terminals.

-m Forces all path widths to be the minimum allowable by the design rules.

-v Enables verbose debugging output.

FILES

~cad/lib/technology/<technology>/design_rules	technology file
<outcell>/physical/contents	input facet
<outcell>/symbolic/contents	output facet
<outcell>/symbolic	primitive directory

SEE ALSO

sparcs(1), octextract(1), octext2spice(1)
oct symbolic policy document

AUTHOR

Bill Lin

NAME

octextract – A Hierarchical Circuit Extractor for Symbolic Layouts

SYNOPSIS

octextract [options] cell[:view]

DESCRIPTION

Octextract is a hierarchical circuit extracting program for symbolic layouts generated in the OCT/VEM environment. Its input is an oct cell of viewtype *symbolic*, and its output is an extracted electrical network stored back into OCT.

If no options are specified, octextract will expect the input cell to have the view name "symbolic" and will produce an output with the same cell name but with the view name "extracted".

The default extraction mode is to produce a simplified electrical network using the lumped RC circuit model. A more accurate, but also more bulky, circuit description using the detail RC network model could be generated using the -d option. In addition, the user can specify resistance and capacitance thresholds to screen out negligible parasitic values.

The command line options are:

-o cell[:view]
Specifies an alternate output. The default output cell name is inherited from the input cell. The default output view name is *extracted.*

-t techfile
Specifies an alternate technology file for design rules (see below for default).

-e expand
Expands the design rules by *expand* factor.

-p directory
Specifies an alternate directory for the library of primitive electrical elements (see below for default).

-r threshold
Sets the resistive threshold to be *threshold* ohms. Only resistive values greater than or equal to this amount will be represented in the extracted outout.

-c threshold
Sets the capacitance threshold to be *threshold* farads. Only capacitance values greater than or equal to this amount will be represented in the extracted outout.

-d Specifies a detailed extraction using the detailed RC network model. The default mode is to produce a simplified circuit using the lumped model.

-v Enables verbose debugging output.

FILES

~cad/lib/technology/<technology>/octextract.tech	technology file
~cad/lib/technology/<technology>/octextract	primitives directory
<outcell>/symbolic/contents	input facet
<outcell>/extracted/contents	output facet

SEE ALSO

kahlua(1), octext2spice(1), spice(1)
oct symbolic policy document

AUTHOR

Bill Lin

NAME

octext2spice – A tool for converting from oct format to spice format

SYNOPSIS

octext2spice [-o filename] cell[:view]

DESCRIPTION

Octext2spice is a tool for converting an extracted output from the oct database into a spice format. The extracted output is assumed to have been generated by the circuit extraction problem *octextract*. The spice node numbers corresponding to the formal terminals of the cell are documented in the spice output. The default output file name is the input cell name appended with a *.spice* suffix. The user can use the *-o* option to specify a different output.

SEE ALSO

octextract(1), spice(1), *oct symbolic policy document*

AUTHOR

Bill Lin

Bibliography

[Bai75] H.S. Baird and Y.E. Cho, "An Artwork Design Verification System", *Proceedings on the ACM IEEE 12th Design Automation Conference*, pp.414-420, 1975.

[Bai77] H.S. Baird, "Fast Algorithms for LSI Artwork Analysis", *Proceedings on the ACM IEEE 14th Design Automation Conference*, pp.303-311, June 1977.

[Bal82] M.W. Bales, "Layout Rule Spacing of Symbolic Integrated Circuit Artwork", ERL Memo No. UCB/ERL M82/72, University of California, Berkeley, December 1982.

[Bas83] J.D. Bastian and C.E. Huang and M. Ellement and L.P. McNamee and P.J. Fowler, "Symbolic Parasitic Extractor for Circuit Simulation (SPECS)", *Proceedings of the ACM IEEE 20th Design Automation Conference*, pp.346-352, 1983.

[Ben80] J. Bentley, D. Haken, and R. Hon, "Fast Geometric Algorithms for VLSI Tasks", *ACM Computing Surveys*, IEEE COMPCON, 1980.

[Ben79] J.L. Bentley and J.H. Friedman, "A Survey of Algorithms and Data Structures for Range Searching", *ACM Computing Surveys*, Vol 11, No.4, 1979.

[Boy87] D.G. Boyer, "Split Grid Compaction for a Virtual Grid Symbolic Design System", *IEEE ICCAN Conference*, November 1987.

[Bry81] R.E. Bryant, "MOSSIM: A Switch-Level Simulation for MOS LSI", *Proceedings on the ACM IEEE 18th Design Automation Conference*, 1981, pp.786-790.

[Bur83] M. Burstein and R. Pelavin, "Hierarchical Wire Routing", *IEEE Trans. on Computer-Aided Design*, Vol CAD-2, No.4, pp.223-234, October 1983.

[Bur86] J.L. Burns and A.R. Newton, "SPARCS: A New Constraint-Based IC Symbolic Layout Spacer" *Proceedings on the Custom Integrated Circuit Conference*, May 1986.

[Bur87] J.L. Burns and A.R. Newton, "Efficient Constraint Generation for Hierarchical Compaction", *Proceedings on the IEEE International Conference on Computer Design"*, October 1987.

[Cho77] Y.E. Cho, A.J. Korenjak, and D.E. Stockton, "FLOSS: An Approach to Automated Layout for High-Volume Designs", *Proceedings of the 14th Design Automation Conference*, June 1977.

[Dun78] A.E. Dunlop, "SLIP: Symbolic Layout of Integreated Circuits with Compaction", *Computer-Aided Design*, Vol. 10, No. 6, pp. 387-391, November 1978.

[Dun80] A.E. Dunlop, "SLIM: The Translation of Symbolic Layouts into Mask Data", *Proceedings of the 17th Design Automation Conference*, pp. 595-602, June 1980.

[Fit82] D.T. Fitzpatrick, "MEXTRA: A Manhattan Circuit Extractor", ERL Memo M82/42, Electronics Research Laboratory, University of California, Berkeley, January 1982.

[Gib76] D. Gibson and S. Nance, "SLIC: Symbolic Layout of Inteegrated Circuits", *Proceedings of the 13th Design Automation Conference*, June 1976.

[Gre86] J.W. Greene, "Layout-to-Layout Compaction for Technology Conversion", *VLSI Systems Design*, pp.46-51, November 1986.

[Gup83] A. Gupta, "ACE: A Circuit Extractor", *Proceedings on the ACM IEEE 20th Design Automation Conference*, pp.721-725, 1983.

[Ham77] G. Hamlin, C.W. Gear, "Raster-Scan Hidden Surface Algorithm Techniques", *Computer Graphics (Proceedings on Siggraph 77)*, Vol. 11, No. 2, pp.206-213, Summer 1977.

[Har86] D. Harrison, P. Moore, R. Spickelmier, A.R. Newton, "Data Management and Graphics Editing in the Berkeley Design Environment" *Proceedings on the IEEE International Conference on Computer-Aided Design*, Nov. 1986.

[Har86b] D. Harrison, "Vem User's Guide" *Internal Memorandum*, University of California, Berkeley, 1982.

[Hig69] D. Hightower, "A Solution to the Line Routing Problem on the Continuous Plane", *Proceedings on the ACM IEEE Design Automation Workshop*, pp.1-24, 1969.

[Hsu78] M.Y. Hsueh, "Symbolic Layout and Compaction of Integrated Circuits", ERL Memo No. UCB/ERL M79/80, University of California, Berkeley, December 1979.

[Ked82] G. Kedem, "The Quad-CIF Tree: A Data Structure for Hierarchical On-Line Algorithms", *Proceedings on the ACM IEEE 19th Design Automation Conference*, pp.352-357, June 1982.

[Kel82a] K.H. Keller and A.R. Newton "KIC2: A Low-Cost, Interactive Editor for Integrated Circuit Design" *Proceedings on the Spring COMPCON Conference*, pp.305-306, 1982.

[Kel82b] K.H. Keller and A.R. Newton, "A Symbolic Design System for Integrated Circuits", *Proceedings on the ACM IEEE 19th Design Automation Conference*, June 1982.

[Kin84] C. Kingsley, "A Hierarchical, Error Tolerant Compactor", *Proceedings on the ACM IEEE 21st Design Automation Conference*, 1984, pp.136-132.

[Lar71] R.P. Larsen, "Computer-Aided Preliminary Layout Design of Customized MOS Arrays", *IEEE Transactions on Computers*, Vol C-20, No.5, pp.512-523, May 1971.

[Lar78] R.P. Larsen, "Versatile Mask Generation Techniques for Custom Microelectronics Devices", *Proceedings of the 15th Design Automation Conference*, June 1978.

[Lau80] U. Lauther, "A Data Structure for Gridless Routing", *Proceedings on the ACM IEEE 17th Design Automation Conference*, June 1980.

[Lau81] U. Lauther, "An O(N log N) Algorithm for Boolean Mask Operations" *Proceedings on the ACM IEEE 18th Design Automation Conference*, June 1981.

[Lee61] C.Y. Lee, "An Algorithm for Path Connection and its Applications", *IRE Trans. on Electronic Computers*, Vol EC-10, pp.346-365, September 1961.

[Li84] J.T. Li snd M. Marek-Sadowska, "Global Routing for Gate Array", *IEEE Transactions on CAD*, Vol CAD-3, October 1984.

[Lin87] B. Lin and A.R. Newton, "KAHLUA: A Hierarchical Circuit Disassembler", *Proceedings on the ACM IEEE 24th Design Automation Conference*, June 1987.

[McC84] S.P. McCormick, "EXCL: A Circuit Extractor for IC Designs", *Proceedings on the ACM IEEE 21st Design Automation Conference"*, pp.624-628, 1984.

[Mea80] C. Mead and L. Conway, *Introduction to VLSI System*, Addison-Wesley Publication, 1980.

[Moo82] P. Moore, "Fang User's Guide" *Internal Memorandum*, University of California, Berkeley 1982.

[Moo86] P. Moore, "The General Structure of OCT" *Internal Memorandum*, University of California, Berkeley, January 1986.

[Nag73] L.W. Nagel and D.O. Pederson, "SPICE: A Simulation Program with Integrated Circuit Emphasis", *Electronics Research Laboratory University of California*, Memorandum No. UCB/ERL M382, April 1973.

[New83] A.R. Newton and A.L. Sangiovanni-Vincentelli, "Relaxation - Based Circuit Simulation", *IEEE Transaction on ED*, Vol ED-30, No.9, September 1983, pp.1184-1207.

[Oct88] "Octtools Version 2.0", *University of California, Berkeley*, 1988.

[Ous81] J.K. Ousterhout, "Caesar: An Interactive Editor for VLSI Layouts" *VLSI Design*, Vol.II, No.4, 4th Qtr, pp.34-38, 1981.

[Ous82] J.K. Ousterhout and D. Ungar, "Measurements of a VLSI Design", *Proceedings of the 19th Design Automation Conference*, June 1982.

[Ous83] J.K. Ousterhout, "Crystal: A Timing Analyzer for NMOS VLSI Circuits", *3rd Caltech Conference on VLSI*, March 1983, pp.57-70.

[Ous84] J.K. Ousterhout, "Corner Stitching: A Data-Structuring Technique for VLSI Layout Tools", *IEEE Trans on Computer-Aided Design*, Vol CAD-3, No.1, January 1984.

[Ous85] J.K. Ousterhout, C.T. Hamachi, R.N. Mayo, W.S. Scott, and G.S. Taylor, "The Magic VLSI Layout System", *IEEE Design and Test of Computers* 2, 1 (February 1985), pp.19-30.

[Ree85] J. Reed, A. Sangiovanni-Vincentelli, and M. Santomauro, "A New Symbolic Channel Router: YACR2", *IEEE Trans. on Computer-Aided Design*, Vol CAD-4, No.3, July 1985, pp.208-219.

[Rog85] Rogersa and Rosenberge and Daniels, "MCNC's Vertically Integrated Symbolic Design System", Proceedings on the ACM IEEE 22nd Design Automation Conference, 1985, pp.62-68.

[Rub74] F. Rubin, "The Lee Connection Algorithm", *IEEE Trans. on Computers*, Vol. C-23, pp.907-914, 1974.

[Sco85] W.S. Scott, "Compaction and Circuit Extraction in the MAGIC IC Layout System", Report No. UCB/Computer Science Dpt. 86/269, Computer Science Division (EECS), University of California, Berkeley, November 1985.

[Shi86] H. Shin and A. Sangiovanni-Vincentelli, "MIGHTY: A 'Rip-Up and Reroute' Detailed Router" *Proceedings on the IEEE International Conference on Computer-Aided Design*, November 1986.

[Spi85] R.L. Spickelmier and A.R. Newton", "Connectivity Verification Using a Rule-Based Approach", *Proceedings on the ICCAD*, 1985, pp.190-192.

[Sza76] L. Szanto, "Network Recognition of a MOS Integrated Circuit from its Masks", *Tesla Electronics*, pp.67-75, September 1976.

[Tan87] D. Tan and N. Weste, "Virtual Grid Symbolic Layout 1987", *Proceedings on the IEEE International Conference on Computer Design*, 1987, pp.192-196.

[Tar83] G. Tarolli and W.J. Herman, "Hierarchical Circuit Extraction with Detailed Parasitic Capacitance", *Proceedings on the ACM IEEE 20th Design Automation Conference"*, pp.337-345, 1983.

[Ter83] C. Terman, "ESIM Reference Manual" *Internal Memorandum*, UCB/MIT, 1983.

[Wes81] N. Weste, "Virtual Grid Symbolic Layout", *Proceedings on the ACM IEEE 18th Design Automation Conference*", pp.225-233, 1981.

[Wil78] J. Williams, "STICKS: A Graphical Compiler for High-Level LSI Design", *AFIPS Conference*, Vol.47, 1978, pp.289-295.

[Yam72] M. Yamin, "Derivation of All Figures Formed by the Intersection of Generalized Polygons", *The Bell System Technical Journal*, Vol. 51, No. 7, pp.1595-1610, September 1972.