SYMBOLIC LAYOUT AND COMPACTION OF INTEGRATED CIRCUITS

by

Min-Yu Hsueh

Memorandum No. UCB/ERL M79/80

10 December 1979

COVER

SYMBOLIC LAYOUT AND COMPACTION OF INTEGRATED CIRCUITS

by

Min-Yu Hsueh

Memorandum No. UCB/ERL M79/80

10 December 1979

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

Symbolic Layout and Compaction of Integrated Circuits

Min-Yu Hsueh

ABSTRACT

A series of computationally efficient algorithms and methods has been developed for the computer-aided generation of layouts of integrated circuit building blocks. The layouts that are generated and compacted are based on symbolic layout plans drawn by the user with an interactive graphics editor. The layout compaction process minimizes the area of a layout by selectively reducing the spacing between elements in the layout plan to the minimum required spacing specified by geometric layout design rules and user-defined constraints. The compaction algorithms do not attempt to make drastic changes, such as interchanging the location of elements, to the topology of the layout plan. The preservation of the general topology makes the compaction algorithms efficient and gives the user control over the layout compaction process.

The principal algorithm used for compacting the layout is derived from the rectangle dissection method by Tutte, et al. Layout compaction and placement methods developed by other researchers are described also for the purpose of comparison.

A layout generation system is implemented based on the aforementioned layout compaction algorithms. The system is written in FORTRAN and Ratfor on a minicomputer and can generate compact layouts for circuits of reasonable size in a few minutes. Because of the almost immediate feedback from the layout system, the user can improve the layout quickly by modifying the topology he has chosen or by trying other topologies. The power of such a cooperation between the user and the computer is illustrated by

several examples. In particular, one example shows that a layout generated with this system is more compact and regular than an equivalent layout done by hand for a high production-volume microprocessor type of circuit.

The speed of the compaction process is limited by the analysis of design-rule requirements among the elements in the symbolic layout plan. Statistics collected from experiments show that the compaction time increases as the 1.4th power of the number of elements in the layout plan. A hierarchical layout approach may be used with the layout system to keep the layout generation time reasonable. Because the compaction algorithms are capable of handling user-defined constraints, the size and shape of compacted building blocks can be fixed with such constraints and used in the same manner as other elements for the hierarchical construction of a layout.

Acknowledgements

I would like to thank all those who have contributed to the development of the CABBAGE system. The following is a short history of the progress of the CABBAGE project and the persons mentioned deserve special thanks.

Professor D. O. Pederson has provided me with invaluable opportunities and guidance that greatly contributed to the overall development of the CABBAGE project. These opportunities include the many summer work positions in the electronic instrument industry, where I was given relevant and interesting projects to work on and have learned the many aspects of the design, construction, and application of integrated circuits. In particular, as a part of my Research Assistantship supported by the Hewllet-Packard Co., I was able to join a team led by K. Eldredge at the Loveland, Colorado, facility of H-P in the summer of 1977 and participate in the development a Large-Scale Integrated circuit chip. The layout experience from that summer, especially the use of layout plans for both the chip-level and the cell-level layout generation, motivated the development of the layout compaction and generation system described in this thesis.

The many excellent courses on integrated circuit design taught at Berkeley provided me with the necessary background for developing the IC layout system described in this report. Among them, courses on layout methods and techniques, and in particular the special seminar arranged by Dean E. S. Kuh and taught by Professor D. A. Mlynski, have been particularly relevant.

The generous equipment grant from Hewlett-Packard Co., administered by M. Brooksby, has made the implementation of the layout system a pleasant exercise. I had the personal use of a graphics terminal with a virtu-

ally dedicated minicomputer and many peripherals. The many operating system improvements and versatile utility programs written by E. Cohen have also immensely improved the environment for program development. In fact, this report was first written on the H-P minicomputer with the interactive text editor implemented by Mr. Cohen. The text was subsequently transferred to the VAX 11/780 computer of the Computer Science Division for generating the final output.

Finally, this report has been read by Professors D. O. Pederson, A. R. Newton, R. Taylor and Dr. S. Tsukiyama. Their criticisms and comments, as well as the discussions I had with Drs. W. McCalla, A. E. Dunlop and W. Heller, and Mr. J. Williams, have improved the quality of this report. The many drawings done by Y. Y. Wang also make the descriptions in the text clearer.

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

A circuit layout is an artwork consisting of detailed device geometry and locations and configurations of interconnection, from which masks for fabricating Large-Scale Integrated (LSI) circuit chips can be made. The circuit layout phase has been one of the major bottlenecks in the development of LSI circuit chips. Layout by hand-drawing of microprocessor-type LSI chips typically takes a man-year to complete. It is such a time-consuming process because physical entities (high-density mask patterns) must be synthesized from abstract structures (circuit schematics or logic equations) at this phase of the design. In addition, with ordinary layout methods, errors such as inconsistent translation and layout design-rule violations are hard to detect while the layout is being generated. As a result, the initial layout may have to be modified several times before the circuit can be manufactured successfully.

Computer-aided layout generation techniques available or under development today can be categorized as shown in the chart in Figure 1.1. It must be noted that the use of interactive graphics editors for the purpose of input and modification of data for most of these layout systems is implied in that chart.

The hierarchical layout construction approach consisting of chip planning, building-block placement and interconnection, and the construction of building blocks has been a popular way for developing high-density, high production-volume LSI circuit chips. Here, a chip is divided into functional

areas (such as a control logic area), and these areas are further divided into building blocks (such as a control sequencer) and their constituent cells (such as a register). With such a division the design and layout of the chip can be carried out at the individual levels of the hierarchy with reduced complexity and improved organization. At all levels of the chip design, most designers find it advantageous to sketch a rough topological organization of the layout before generating the actual geometric layout. In particular, at the cell or the simple building-block level, such a topological planning involves the conversion of the circuit schematic diagram into a topological layout plan with an improved placement of circuit elements and routing of interconnection lines to reduce, for example, the number of line crossovers. The use of this type of topological layout plan to guide the generation of the actual layout is commonly referred to as the symbolic layout approach. (The symbolic topological layout plan also is referred to as the "stick diagram" of a circuit [1, 18].) Figure 1.2(a) shows the circuit schematic diagram of a Metal-Oxide-Silicon (MOS) buffer circuit and one of its topological layout plans drawn in the symbolic form. The symbolic layout plan has an improved topology compared to the original circuit schematic and makes it possible to produce a dense and well organized final layout, as in Figure 1.2(b). (This can be compared with the final layout as in Figure 1.2(c), which is generated directly from the circuit schematic diagram. Both final layouts are generated with the programs developed in this research project.)

This report introduces a symbolic layout method which converts a loose symbolic topological layout plan of a cell or a building block into a compact and correct geometric layout with the aid of a computer. This symbolic layout approach can improve the productivity of a layout designer greatly because

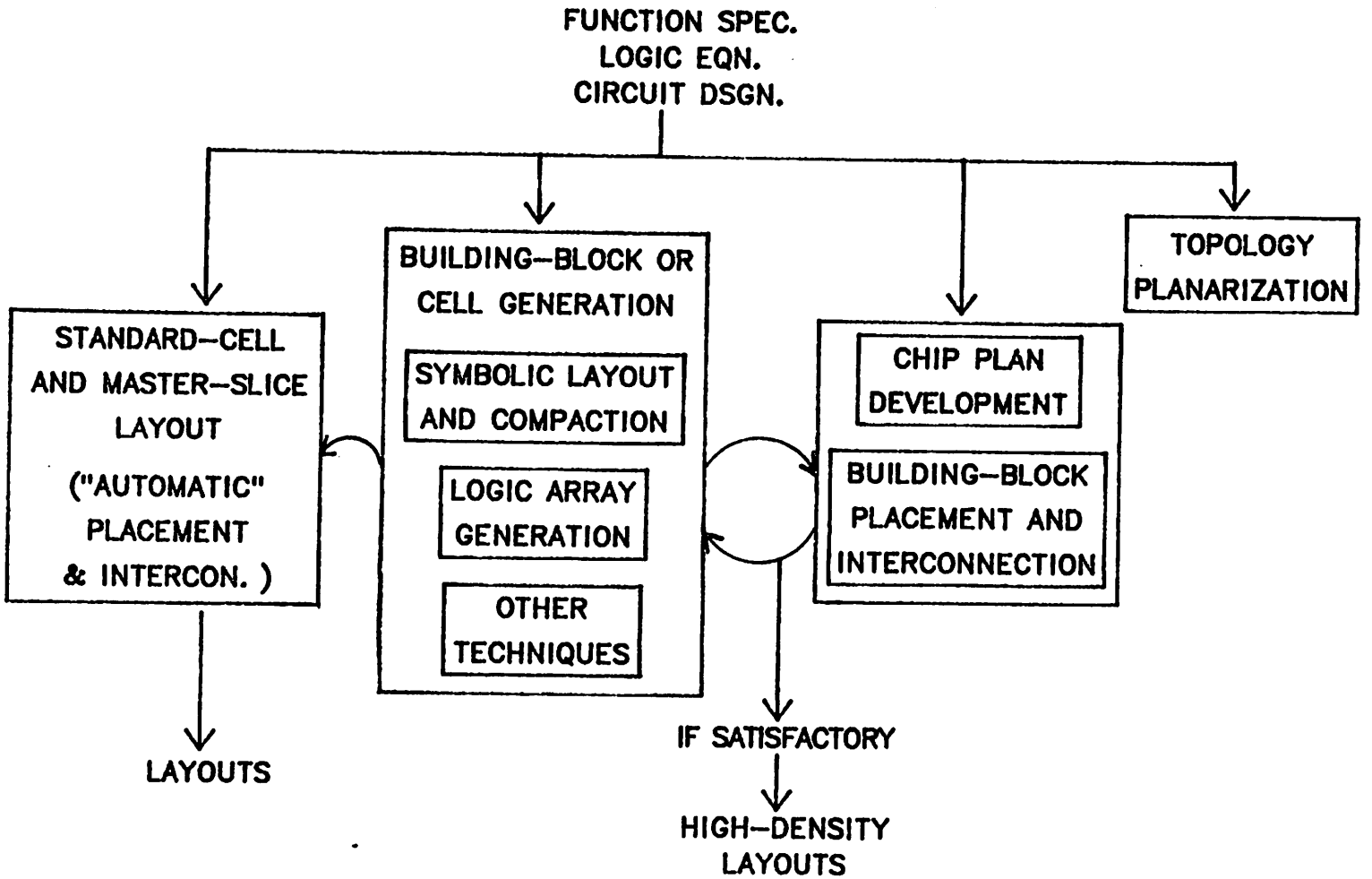


Figure 1.1 Computer–based tools for IC layout.

a) it is often possible to obtain a better geometric layout if topological constraints and details are thought out in advance.

b) most layout designers take a comparatively short time to develop a good layout topology and consume much more time to generate the actual geometric layout.

c) geometric layout design rules are enforced automatically, so that little effort by the user is required to ensure the correctness of the layout geometry.

d) with fast turn-around, the user can experiment with many different layout topologies from which the best topology can be selected for the final layout.

In effect, the symbolic layout method developed in this research project reduces the cell layout process to that of topology planning. The designer can concentrate on developing good topological layout plans, while the computer program performs the tedious task of generating the detailed geometric layout meeting all design-rule requirements.

The layout approach described in this report is just one of a large number of layout methods being developed for the present and the future LSI circuit layout needs. In Chapter 2 the representative system realizations of the computer-aided layout techniques shown in Figure 1.1 are described briefly.

Chapter 3 describes several representative algorithms for computer-aided layout generation and methods related to or having similar functions as those developed in this research project. With the background material

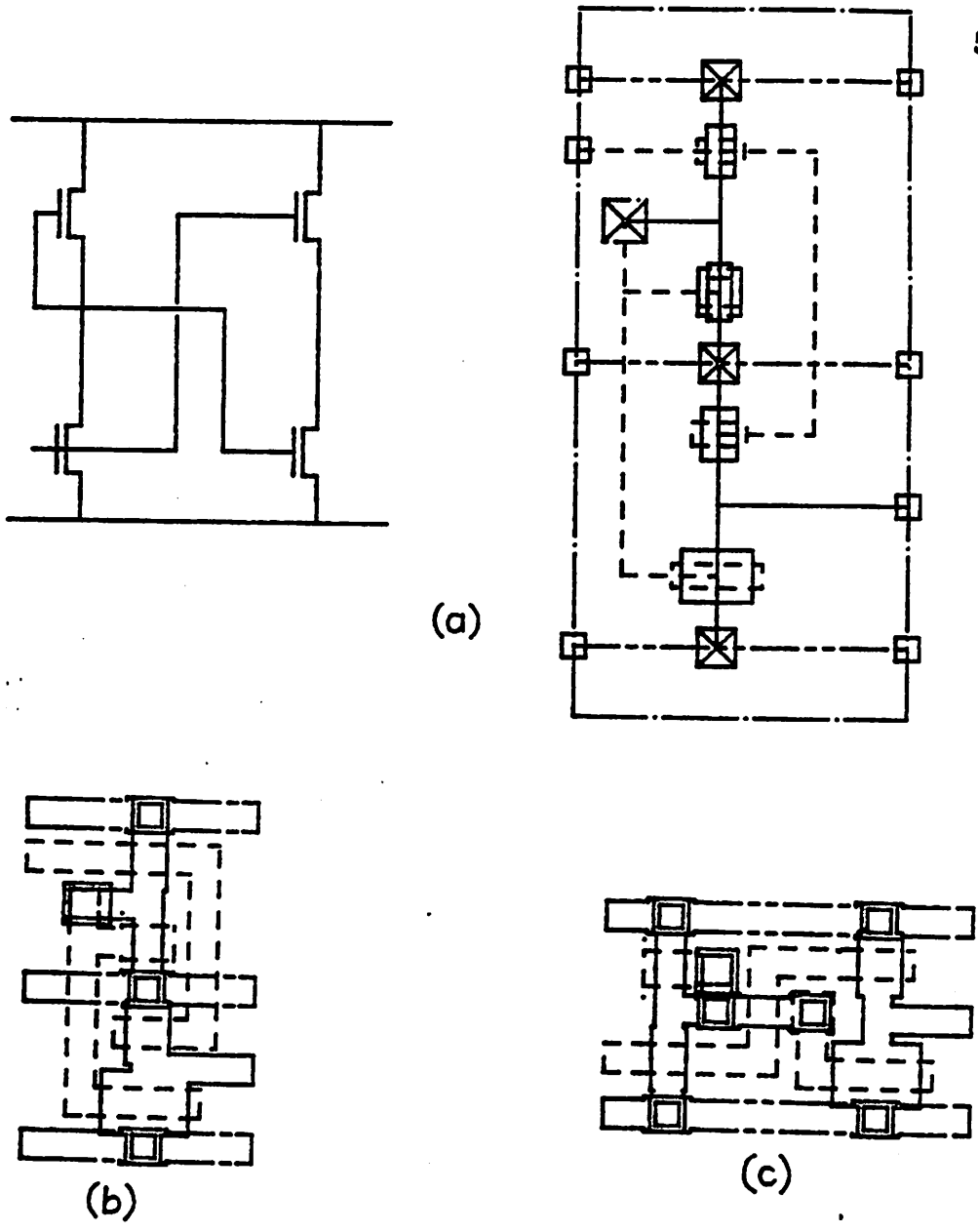


Figure 1.2(a) The circuit schematic and symbolic topological layout plan of a buffer.
(b) The compacted layout generated from the layout plan.
(c) The compacted layout generated from the circuit schematic is 8% larger.

introduced in Chapter 3, Chapter 4 presents an overview of the symbolic layout system CABBAGE (Computer-Aided Building-Block Artwork Generator and Editor) that resulted from this research project. The algorithms and techniques used in this system are also described. These algorithms are developed with the emphasis of computation efficiency, good man-computer interaction, and applicability to hierarchical chip construction, in addition to realizing the usual advantages of the symbolic layout approach.

The implementation of these algorithms in CABBAGE is the subject of Chapter 5. Most of the detailed descriptions of the accompanying data structures are placed in appendices. Chapter 6 includes several layout compaction examples and the corresponding run time and memory usage statistics obtained from the CABBAGE system. In addition to the expected gain in layout productivity, one of the examples shows that the CABBAGE system can be used to generate layouts as compact as those done by hand for microprocessor-type LSI circuits.

At the present time, the CABBAGE system can perform layout generation for silicon-gate MOS circuits only. Chapter 7 summarizes the possible future extensions to CABBAGE necessary for handling other types of circuits. Several features which were excluded from the scope of the present research project are also described in that chapter as possible future research topics.

CHAPTER 2

LSI CIRCUIT LAYOUT METHODS

2.1. Introduction

Traditionally, most circuit layouts have been done by hand in the hope to achieve ultimate area efficiency and to lower the fabrication cost of individual circuit chips. However, layout by hand is both a tedious and an error-prone task. Repeated modifications and reworks necessary to correct layout errors often lengthen the total layout time substantially.

Errors in a hand-produced layout generally come in two categories. The first involves the erroneous translation of a circuit function into a geometric layout. This type of error often occurs in the form of missing or extra elements and interconnections. The second type of error involves the so-called design-rule violations. Geometric layout design rules are summaries of chip production tolerances derived from photolithographic limits, mask alignment, etching tolerances, oxidation and impurity diffusion effects, and device physics limitations. (A more detailed description of typical geometric layout design rules is included in Appendix 4.) Violations of the design rules, such as placing two distinct elements too close to each other, are often the reason for drastically lowered chip production yield. These two types of errors usually are detected after a hand-drawn layout is completed. (Often the errors are detected with the aid of design-rule check and circuit-to-layout consistency check programs.)

Digital LSI circuits containing over 20,000 devices are common today, and the device count has been doubling every two years. With such a growing

chip complexity, it has become even more important to be able to generate error-free layouts efficiently. Further, in order to manage the ever-increasing complexity effectively, chips should be well-organized for the easy addition of future improvements. Most modern computer-aided layout methods are designed to prevent the aforementioned errors from occurring in the first place. For example, the standard-cell layout method [2, 3] accepts logic specifications as input and produces layout with standard forms almost automatically. The symbolic layout method described in this report accepts a topological layout specification in symbolic form and produces an area-efficient layout free of design-rule violations automatically. Because the layout is specified with simpler, higher-level descriptions, both these methods make it easy to modify the layout quickly.

The following sections describe some of the most popular layout methods used for LSI circuit design today, as well as some of the most effective computer-aided layout methods under development.

2.2. The Traditional Layout Method

To date, for the lack of better tools, most LSI circuits have been laid out by hand. Typically, at the beginning of the layout phase, the designer estimates the building-block size, shape, and interconnection requirements and creates a chip-level organization plan (chip plan) that summarizes the block placement and the interconnection routing schemes. Based on this chip plan, the detailed layout for each building-block is drawn by hand. The drawings are subsequently digitized and saved in computer-readable formats. Programs that check layout design-rule violations, and circuit-to-layout consistency and performance can then be used to detect errors in the layout.

Correction of some of these errors may require a substantial rework of the layout because the original layout is normally tightly packed. Frequently, several passes through the design loop are necessary before a circuit can be manufactured successfully.

The use of interactive graphics systems has been very popular in the hope of improving the productivity of this traditional layout scheme [4]. These systems are useful for drawing, editing and archiving layouts, but provide little for the efficient generation of correct layouts. In fact, generating layouts with interactive graphics systems may be likened with generating documentations with interactive text editors lacking text-formating capabilities. Both tools are incomplete in that they require the user pay attention to distracting details, such as the spacing requirements between elements in the case of the layout generation and the length of sentences and the pagination in the case of writting a documentation. With such interactive graphics systems the basic design flow still loops around the editing and the checking steps.

The traditional hand or interactive graphics-aided layout approach has become increasingly ineffective for both present and future layout needs. Over the years a number of computer-based layout methods have been under development to tackle the geometric complexity and to best utilize the system potentials of LSI circuits. Due to the complexity of the layout problem, most methods attempt to improve just one aspect of the layout process. For example, the symbolic layout approach is very useful for generating detailed cell-level layouts but becomes less effective when dealing with the planning of the overall chip organization. It is most profitable to incorporate all the computer-based tools shown in Figure 1.1 in a general, hierarchical layout

system to maximize the strength of the individual techniques. Despite apparent differences in the emphasis, the principles behind the majority of these layout methods are similar, namely:

a) the use of higher-level representations to simplify the user interface. Rather than letting the user manipulate the layout at the geometric shape level, function specification or circuit configuration is used as the starting point of the layout. Thus, the initial layout specification and the subsequent modification procedures may be significantly simplified.

b) the introduction of standardization. These methods call for the use of regular layout forms or existing cell structures whenever possible, either in the form of an array or in a hierarchical fashion. For both the array-oriented and the hierarchical layout approaches, the standardization typically requires the use of cells and building-blocks of similar and compatible size, shape, and structure (such as the arrangement of power buses in the buiding blocks).

These computer-based layout methods are described in the following sections.

2.3. The Standard-Form Layout Approaches

The standard-form layout methods are aimed at the automatic translation of logic designs into circuit layouts. Thus, in principle, the layout phase can be significantly shortened with a standard-form layout method. Typically, a standard-form layout system employs a pre-defined layout structure

into which logic equations can be mapped directly. Three such layout structures are described below.

2.3.1. The Standard-Cell Layout Method

The standard-cell layout method is a way to automatically place and interconnect a large variety of standardized cell layouts based on circuit interconnection information [2, 3]. The standard cells typically perform logic gate- and register-level functions. They are drawn carefully so that their heights are identical and their widths integral multiples of a common grid size. An example of a standard cell is shown in Figure 2.1(a). In addition, the signal and power lines into each cell are put on grid positions or at predetermined locations. Thus, standard cells can be placed in rows and interconnected through the intervening routing channels in a uniform fashion, as shown by the example in Figure 2.1(b).

Because of the similarity in layout configuration between the standard-cell chip and the printed circuit board (PCB), many methods related to popular PCB placement and wire-routing schemes are used by standard-cell layout systems. In the cell placement phase, each standard cell is assigned to a row based on its connectivity with other cells in the circuit. , The channel width needed to wire up the cells in a given row is considered also in assigning a cell to a particular row. The location of a cell is further optimized with intrarow manipulations, such as reflection and pair-wise interchange. If a group of interconnected cells are assigned to different cell rows, feed-through cells must be inserted into the intervening rows to accommodate inter-channel routing. Channel routing is then carried out to complete the interconnection. In the routing phase, the goal is to keep the channel width

as close to the theoretical lower bound as possible while ensuring routing completion.

Computer-based standard-cell layout systems can generate error-free LSI circuit layouts with minimal human intervention (to complete a few difficult wire routes the computer program failed to handle) in a matter of a few hours. The resulting layouts, however, are generally large and loose. The large layout size makes the circuit costly to produce. (The circuit production yield is higher than what can be expected based on the usual inverse relationship between the chip size and the yield. The higher than usual yield may be attributed to the fact that defects in the silicon substrate are less likely to be covered by active areas in a loose layout.) The looseness of the layout degrades the circuit performance because power and some signal lines are often required to extend over long distances. Despite these drawbacks, the standard-cell layout approach has been very popular with system manufacturers because of its effectiveness to produce system components (LSI circuits) quickly. The added circuit production cost and the lost circuit performance of standard-cell-type layouts may be compensated by the overall system-level design.

2.3.2. The Master-Slice Layout Method

The master-slice layout approach [5] also follows the principle of repeated use of standardized components in a predefined manner to implement logic designs directly. In this case, frequently used cell-level circuits are arranged and prefabricated on LSI chips, called the master slices. Routing programs are then used to interconnect the cells on one or more mask layers to customize a master slice for particular applications. A schematic

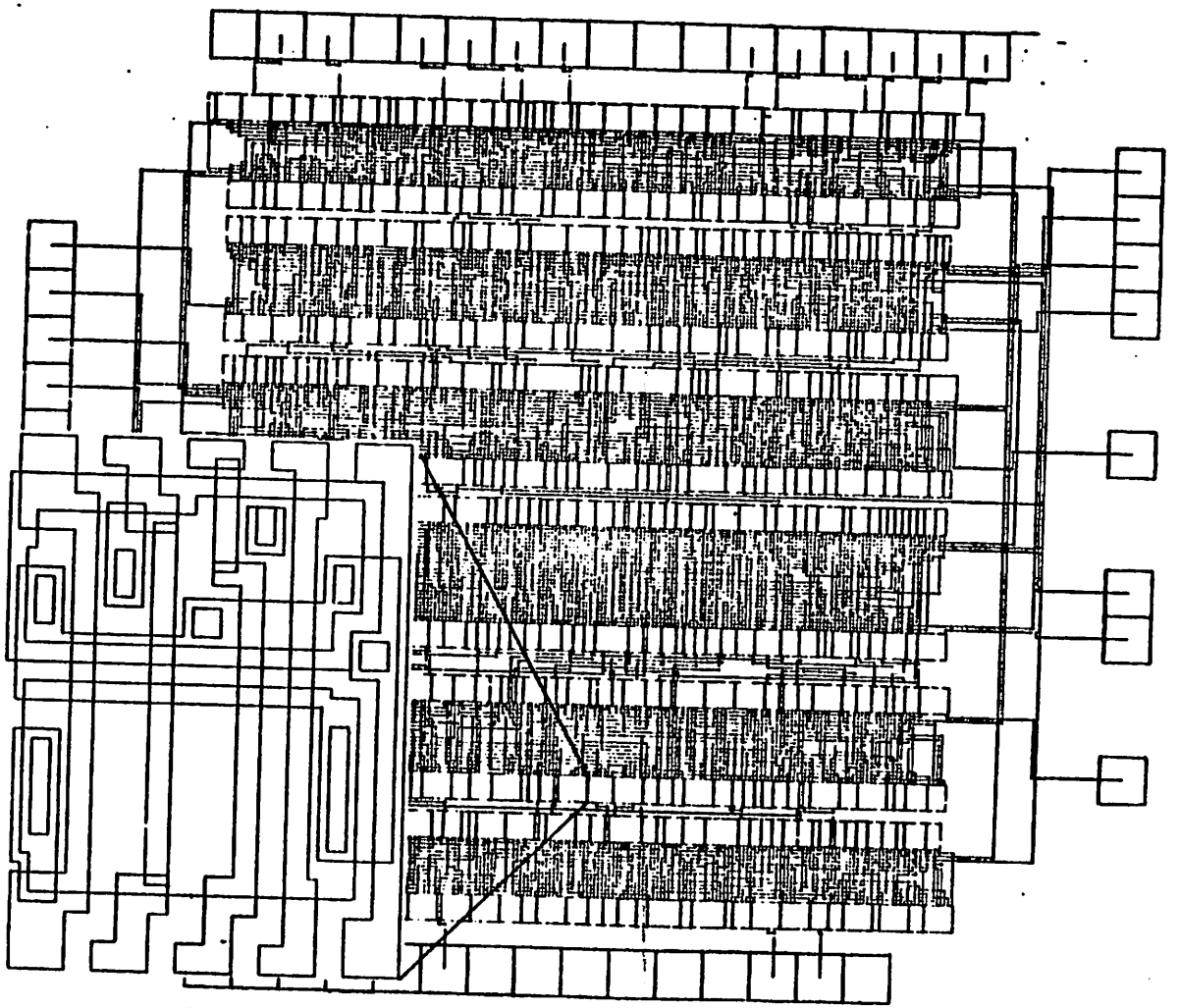


Figure 2.1(a) A standard cell and (b) a standard–cell layout

diagram of a master slice is shown in Figure 2.2. This layout approach has been heavily used by main-frame computer manufacturers, who prefer to keep a relatively small inventory of component types (master slices) capable of covering most of the circuit needs of their computers.

Both the standard-cell and the master-slice layout methods attempt to implement the classical random logic design directly. As such, they must support any number of cells used by logic designers. Since each cell must be laid out and checked, the initial overhead of constructing the cell library can be quite significant. The third standard layout form avoids this problem by employing a memory-like structure, which remembers logic equations as bit patterns.

2.3.3. The Programmable Logic Array as a Standard Layout Form

A Programmable Logic Array (PLA) consists of a regular array of transistor sites so arranged that desired logical AND-OR-NOT or OR-AND-NOT functions can be obtained if transistors are placed in the proper sites. Furthermore, finite-state machines can be implemented with PLA's by adding internal or external feed-back and state-registering mechanisms. Figure 2.3 shows the implementation of two different logic equations with the same PLA structure.

The regularity of the PLA structure makes it ideal for automatic generation from logic specifications. However, since the size of a PLA grows with the number of prime implicants (the product terms appearing in the output of an AND-OR-NOT PLA), a straightforward translation of a complex logic specification into the PLA form often results in a large PLA layout with sparsely planted transistors. The long interconnects in such a large PLA may

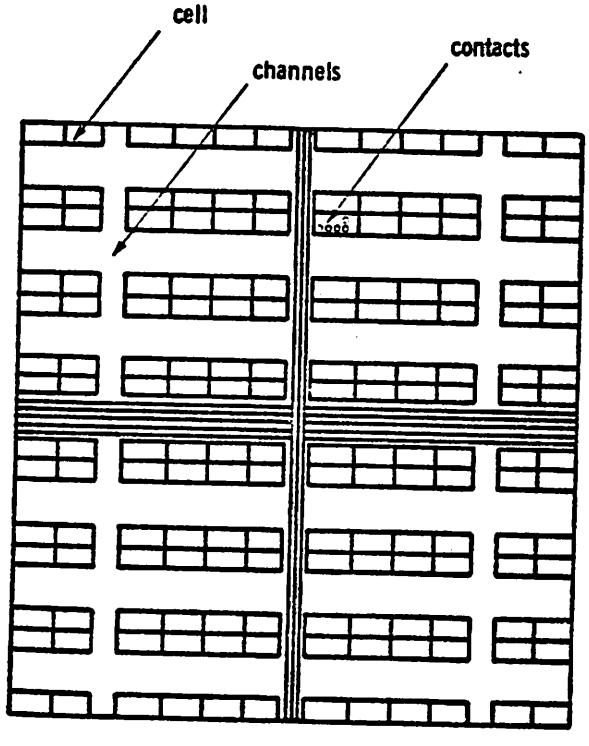


Figure 2.2 A schematic showing the organization of a master slice.

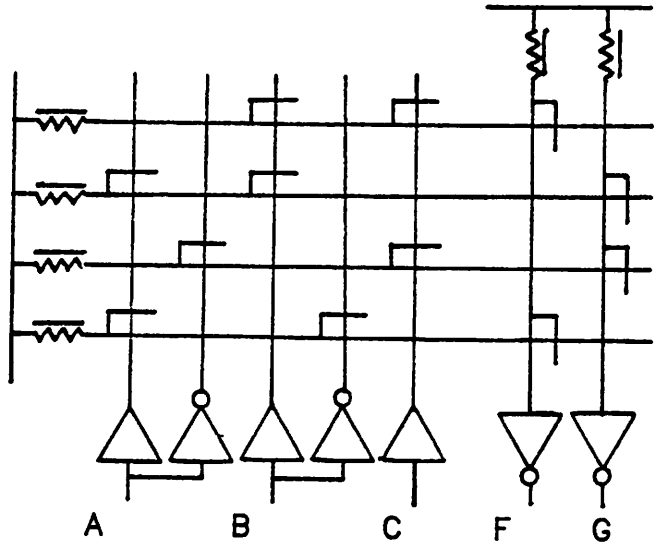
significantly degrade the electrical performance.

The problem with the low density has been tackled with logic equation minimization schemes. The classical approach to the PLA minimization problem uses a process in which all prime implicants are generated first, and a minimal set is then selected to cover the outputs. Because of the computational complexity, this formal procedure is impractical for most real-world design problems. A heuristic approach capable of minimizing function specifications consisting of a few hundred implicants has been reported by Hong et al. [6]. Recently, a new PLA generation technique has been proposed which implements complex functions with a hierarchy of small PLA's [7]. This new scheme may realize significant area reduction and is compatible with a top-down hierarchical design methodology.

2.4. The Hierarchical Layout Approach

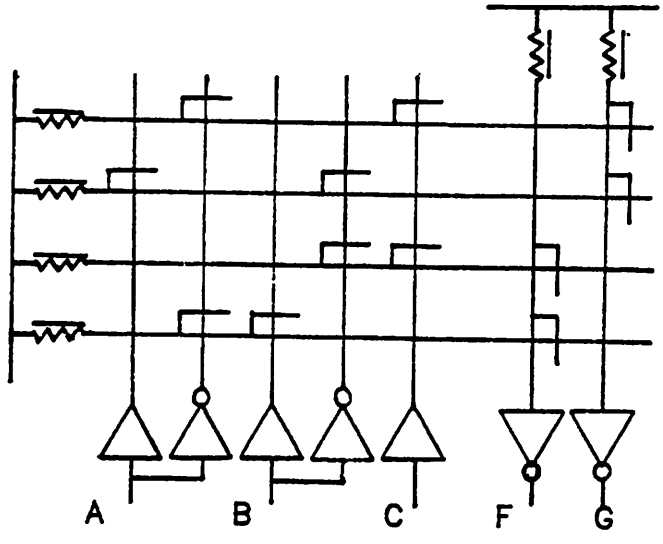
The hierarchical layout approach calls for the structured implementation of a layout to manage the complexity of the layout problem. At each level of the hierarchy the layout schemes best suited for the implementation of individual system functions may be selected to perform the layout operation. In contrast to standard-form layout methods, where the layout process is effectively eliminated from the design cycle, the hierarchical approach stresses the fact that layout requirements may be used to advantage as an integral part of system design considerations. A detailed presentation on the structured design methodology is given by Mead and Conway [1].

The traditional hand layout method that begins with the chip planning process is in fact a hierarchical layout method. As described in Section 2.2, this hierarchical scheme begins with the preparation of a chip organization



$F = (A+B')(B+C)$

$G = (A'+C)(A+B)$



$F = (A'+B)(B'+C)$

$G = (A+B')(A'+C)$

Figure 2.3 The logic function of a PLA is determined by the locations of the transistors.

plan that partitions the layout into functional areas and constituent building blocks. With the chip plan as a guide, each building-block is then constructed with cells and devices in a hierarchical manner. The building-blocks are subsequently merged, and fine adjustments are made to both the blocks and the chip organization plan until these blocks are bound together snugly. At the present time, an integral system of computer aids does not exist for this type of layout approach. However, components for such a system have been the subject of an increasing number of research and development projects. The two major types of components are those for the chip-level design, such as the chip planning and the building-block placement and interconnection techniques, and those for the block-level design, such as the symbolic layout method. The organization of these tools into a layout system is depicted previously in Figure 1.1.

2.4.1. Chip-Plan Development and Layout of Building-Blocks

Building-block layout methods deal with the problem of placing and interconnecting blocks to form a complete chip layout [8, 10, 11]. The individual building blocks are usually laid out as random logic, PLA, or collection of standard-cells and may have arbitrary size, shape, and locations of connection points. As such, the building-block layout problem is quite complex and most work in this area uses separate placement and interconnection phases to generate the final solution. Because of this separation, layouts generated with building-block layout programs tend to be larger than what can be produced by hand. However, with the development of new algorithms that handle placement and routing simultaneously [12] and the complexity of typical LSI circuits increasing beyond the level of 50 building blocks and 100 buses, the computer-based building-block layout methods have received growing

acceptance.

The chip planning problem encompasses not only the generation of an approximate block placement and interconnection routing scheme, but the estimation of block size and wiring requirements as well. Although there has been some work on the subject of predicting wiring space requirements [13], a formula for estimating the block size is not yet available.

2.4.2. The Symbolic Layout Method

The symbolic layout method provides a way to convert automatically cell or building-block topological specifications in symbolic form to actual geometric layouts. Thus, the symbolic layout method reduces the tedious and error-prone layout process to a simple operation of topology planning. Since layout design-rules are enforced by the conversion program, the resulting layout is normally free of design-rule violations.

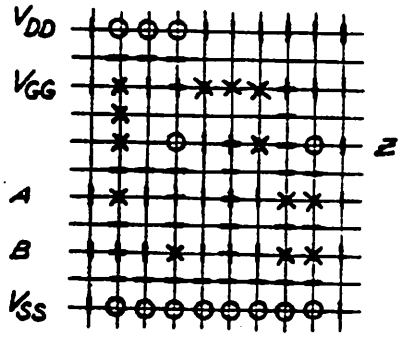
There are two types of symbolic layout methods. The fixed-grid (or coarse-grid) method [14, 15] requires the layout plan be drawn on equally spaced grids. Here the grid spacings are determined on the basis of a set of dominant layout design-rules. The conversion program operates simply to replace each symbol in the user input with its actual geometric structure. Figure 2.4 shows an example of a symbolic layout plan and the resulting layout from a fixed-grid symbolic layout system.

The fixed-grid symbolic layout technique has been used successfully by a number of companies in the electronics industry. It has proven to be an effective method for generating reasonably compact layouts in a short time. Note, however, that the compactness of the resulting layout depends on two factors: First, the original user input in the symbolic form has to be very

compact as the program only performs symbol-to-geometry conversion. Second, since the design-rules used to determine the common grid spacing normally do not give rise to identical minimum pitch (the sum of the minimum line width and spacing) on all mask layers, layers with smaller minimum pitch are made sparser in order to match the larger minimum pitch of other layers. Alternatively, several grid spacings can be used for different mask layers to avoid this low-density problem at the expense of increased system complexity.

The second type of symbolic layout method avoids these difficulties by performing geometry modifications, such as compaction and expansion of elements, and limited topological changes, such as introduction of jogs, on loose symbolic layout plans or other types of layout representations drawn by the user [16, 17, 18, 19, 20]. This type of method is termed the relative-grid method since the loose layout plan indicates only the relative placement and interconnection of element symbols with respect to other symbols. Figure 2.5 gives an example of a layout generated by the relative-grid method developed in this research project. (The different mask layers generated by CABBAGE are represented as follows: diffusion is in solid line or green, polysilicon is in dashed line or red, metal is in dash-dotted lines or blue, and a fourth line type or black is used for all other mask layer or symbols.)

The relative-grid method is more versatile because it lets the user ignore design-rule requirements almost completely. Thus the user can spend most of the design time wisely on developing good layout topologies based on his experience, cleverness, and common sense. (The design strategy for the latch-driver circuit in Section 6.2.1. is an example of this type of heuristic approach to the topological layout planning problem.) In principle, it is also



SYMBOLIC VERSION

TOPOLOGICAL VERSION

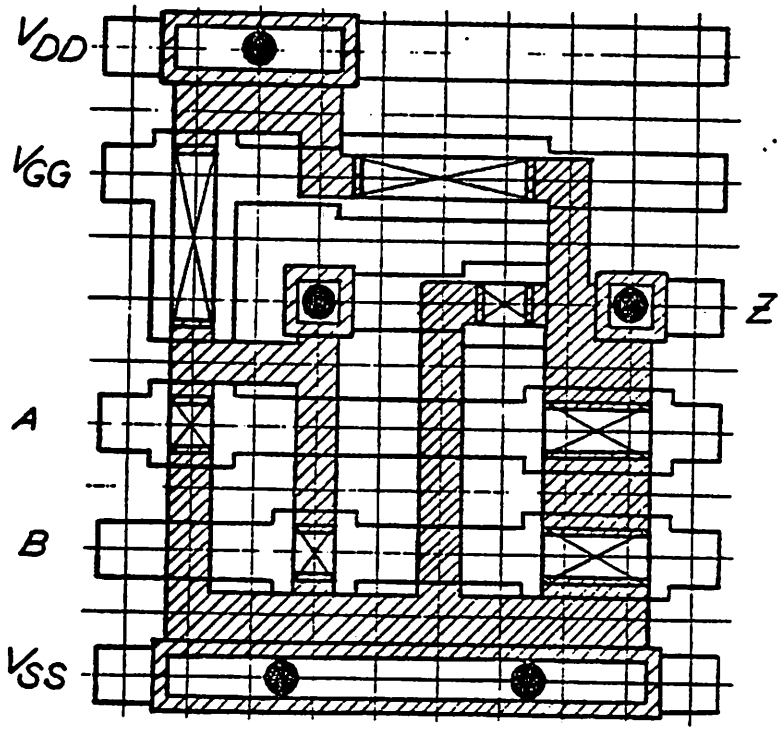


Figure 2.4 An example of a fixed–grid symbolic layout.

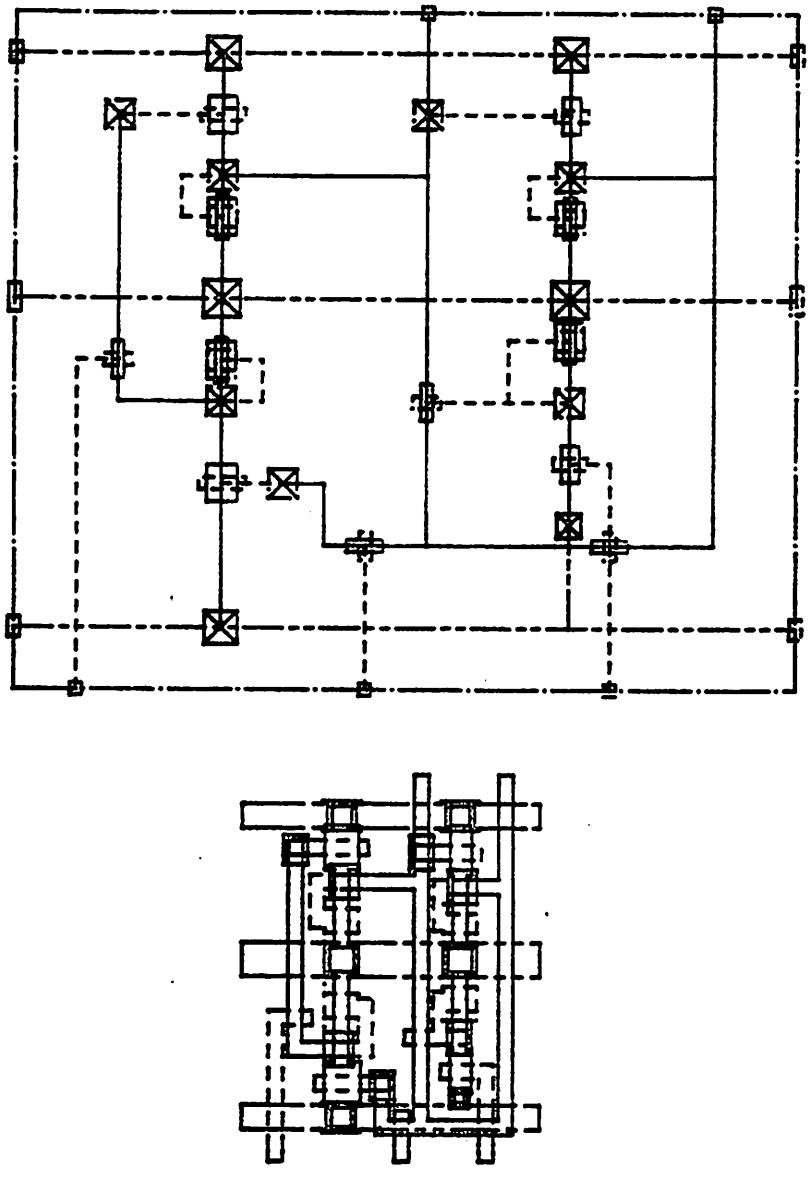


Figure 2.5 The symbolic layout plan and the compact layout of a T–flip flop.

possible to generate topological layout plans directly from circuit connectivity information with the aid of rigorous graph planarization algorithms [21, 22] and assignment techniques [23]. Since the actual geometric layout of typical cells and simple building blocks can be generated with relative-grid symbolic layout systems (such as CABBAGE) in a matter of a few minutes on minicomputers, the user can receive almost immediate feedback on the topology he has devised.

The versatility of the relative-grid symbolic layout approach does require much more intelligence on the part of the computer program. The next two chapters introduce a number of algorithms and related methods used in the aforementioned relative-grid layout compaction programs.

CHAPTER 3

ALGORITHMS FOR LAYOUT GENERATION

3.1. Introduction

A relative-grid symbolic layout plan, such as the one in Figure 2.5, provides the designer with an efficient means for expressing the desired layout topology, from which an actual geometric layout meeting all design-rule requirements can be generated automatically by a computer program. Provided that a good initial topological specification is given, a geometric layout of high quality usually can be generated by simply compressing the symbols close to one another.

In this report, the generation of an actual layout from a symbolic layout plan is handled as a global building-block placement problem. The geometric shapes associated with the symbols are placed according to the planned topology as close to one another as is permitted by design-rule requirements and other fixed constraints. A similar approach is used by Ivannikov and Sipchuk [20]. In fact, the placement method used in this research project is equivalent to the well-known Critical Path Method (CPM) for task scheduling [24]. (The symbolic structures can be viewed as individual tasks, and their locations may be calculated as if they were the starting time of the tasks.) The CPM approach is also used by Cho et al. [16] in an independent effort to solve the problem of building-block LSI circuit layout problem. Other formulations for the symbolic layout compaction problem include the compression ridge method used by Akers et al. [25] and Dunlop [17], and the localized

placement method used by Williams [18]. All these layout generation methods can stretch as well as compress symbols in order to generate layouts without design-rule violations. For simplicity, the generation of such tightly packed geometric layouts is commonly termed the layout compaction operation.

This chapter presents a survey of a number of placement algorithms which are related to the algorithms and methods developed in this project. In addition, the symbolic layout compaction techniques used by other researchers are described for the purpose of comparison.

3.2. Building-Block Placement Algorithms

Building-block placement algorithms are used to solve the problem of tightly fitting blocks of dissimilar size and shape on a chip, while observing connection and proximity requirements. Some of the ideas in building-block placement algorithms are applicable to the layout generation from relative-grid symbolic layout plans, since the actual geometric shapes of the symbols may be considered as blocks.

There are a number of placement techniques [26] which have been successfully used by printed circuit board layout systems to place standardized integrated circuit packages on a board. But most of these become less effective when applied to the building-block-oriented LSI circuit layout environment because of the widely varied size and shape of typical building-blocks. A group of more successful building-block placement methods employ variations of the rectangle dissection method developed by Tutte et al. [27], as detailed below.

3.2.1. Rectangle Dissection-Type Algorithms

Tutte's method was originally developed to dissect a rectangle into a finite number of non-overlapping squares. This problem is solved by mapping the desired organization of the resulting squares into a single source, single sink, planar directed graph (polar graph). Further, the polar graph is viewed as a resistive network, where the height-to-width ratios of individual areas are treated as branch resistances. (Thus unit resistances are used for dissecting a rectangle into squares.) A current equaling the width of the rectangle is fed into the source node to develop voltages. The voltage across and the current through each branch are considered as the actual values of the height and the width, respectively, of the corresponding square. Figure 3.1 provides an example of this scheme. This method also can be applied to dissect a rectangle into many non-overlapping smaller rectangles by using their aspect ratios as the branch resistances.

Since minimum area requirements of the constituent rectangles can not be expressed explicitly in the original dissection formulation, modified forms of Tutte's method are used by most building-block placement algorithms employing the dissection approach. In one scheme, the power dissipation (area) of each branch (constituent rectangle) is used in place of the resistance (aspect ratio) to fix the minimum available area [28]. However, a dissection solution is not guaranteed for any arbitrary combination of power dissipations. A more popular approach is to optimize the maximum cutset width and the longest path length of the polar graph subject to meeting the lateral length constraint of the constituent rectangles [29, 30]. Since polar graphs representing the vertical and the horizontal relations are the dual of each other [27], an equivalent scheme is to optimize the longest path lengths

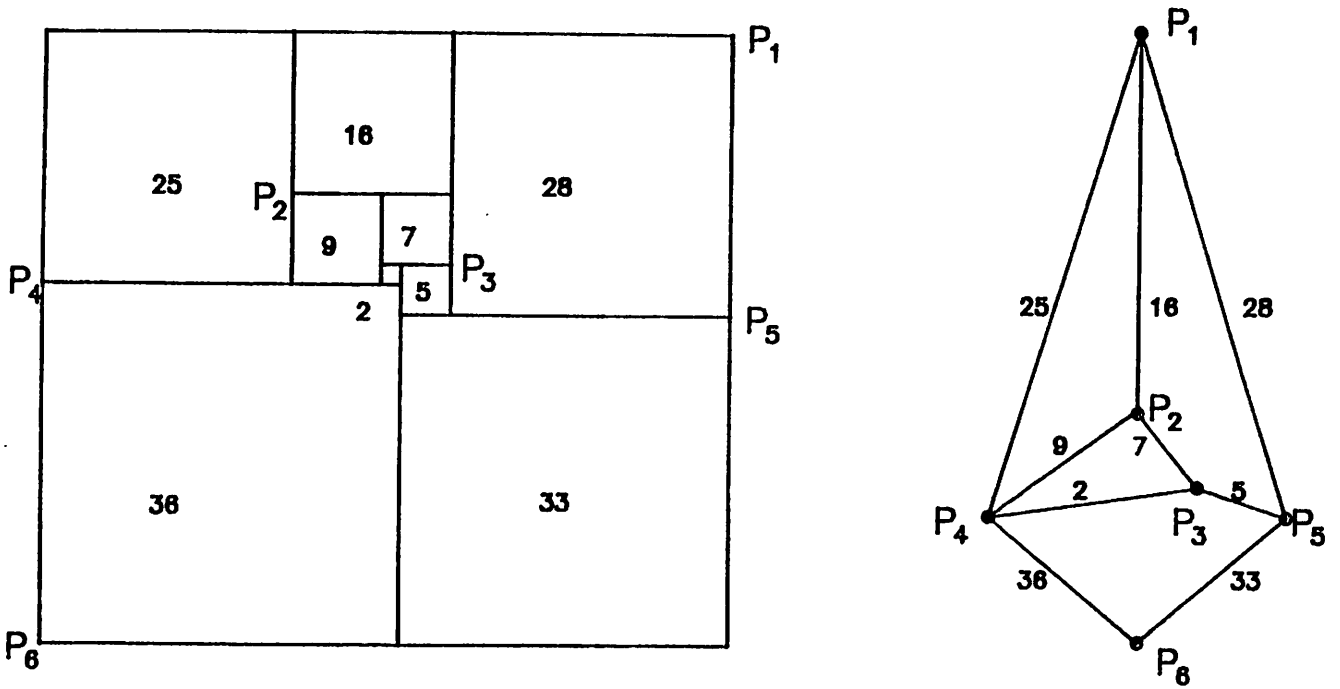


Figure 3.1 An example of a polar graph and the corresponding dissection result.

in the dual graphs representing the respective horizontal and vertical structural relationships, as shown in Figure 3.2 [31, 8, 10].

Building-block layout systems employing the placement methods described above typically carry out the interconnection routing in a separate phase. It is interesting to note that polar graph-based solution schemes again may be used here to determine the arrangement of routing channels [9, 10].

3.2.2. A Chip Topology Planning Algorithm

A chip topology plan is an organizational plan indicating how building-blocks can be packed together and optimally interconnected, assuming their approximate sizes and shapes have been determined.

Recently, Heller proposed a method which uses weighted dual graphs of an initial relation-graph as chip plans [12]. In Heller's algorithm, the initial relation-graph is an undirected graph whose weighted nodes and branches represent building-block areas and numbers of interconnection lines between building-blocks, respectively. Additional nodes representing "global wiring areas" are placed at the crossover points of branches to planarize the initial relation-graph. A suitable chip topology plan is then selected from the rectangularized weighted dual graphs of the initial relation-graph. These dual graphs are constructed from the initial graph such that a) the areas enclosed by branches are sufficient to accommodate the corresponding building-blocks, and b) the branches are long enough to accommodate the total width of interconnection lines crossing them. As such, it is possible to take into account placement and interconnection constraints simultaneously. An example from the paper by Heller [12] is shown in Figure 3.3.

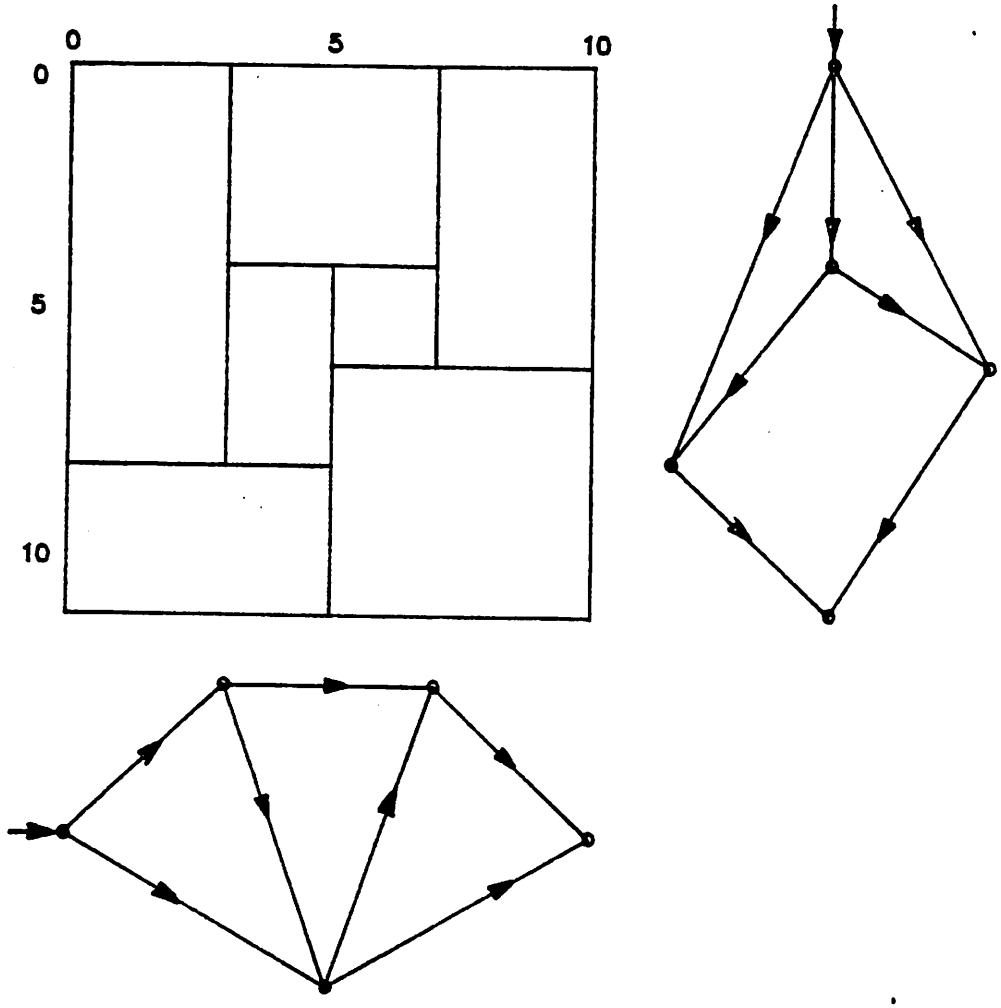


Fig. 3.2 Graph representations of the horizontal and vertical structures of the dissected rectangle.

3.3. Symbolic Layout Compaction Algorithms

Symbolic layout compaction algorithms are used to generate correct and area-efficient layouts based on the connection, adjacency, and spacing requirements of constituent elements. The symbolic layout compaction problem has been dealt with from many angles. Due to its complexity, however, all published algorithms in this area perform the compaction of the horizontal and vertical dimensions of a layout separately. Also, the original topology of the layout plan is kept more or less unchanged throughout the compaction process. Table 3.1 summarizes the statistics obtained from some of the existing layout compaction programs to provide an overview of the capabilities of layout compaction methods as a whole. This table is not intended as a chart of comparison since all methods have different emphases and are implemented with dissimilar equipments. In particular, the size of a circuit is measured differently by the four systems as the number of elements (in SLIP) or elements and lines (in CABBAGE), the number of intersections (in STICKS), or the number of rectangle edges (in FLOSS). Area penalties of the compacted layouts over the hand-generated equivalent solutions are available for several of the examples and are listed in the last column of the table as percentage increases in area. Brief descriptions of the compaction algorithms developed by other researchers are given in the following subsections.

3.3.1. The Compression Ridge Method

In 1970, Akers, et al., described a method to compact a loose layout by removing excess space from it successively [25]. Specifically, bands of continuous excess area, called compression ridges, are developed and removed

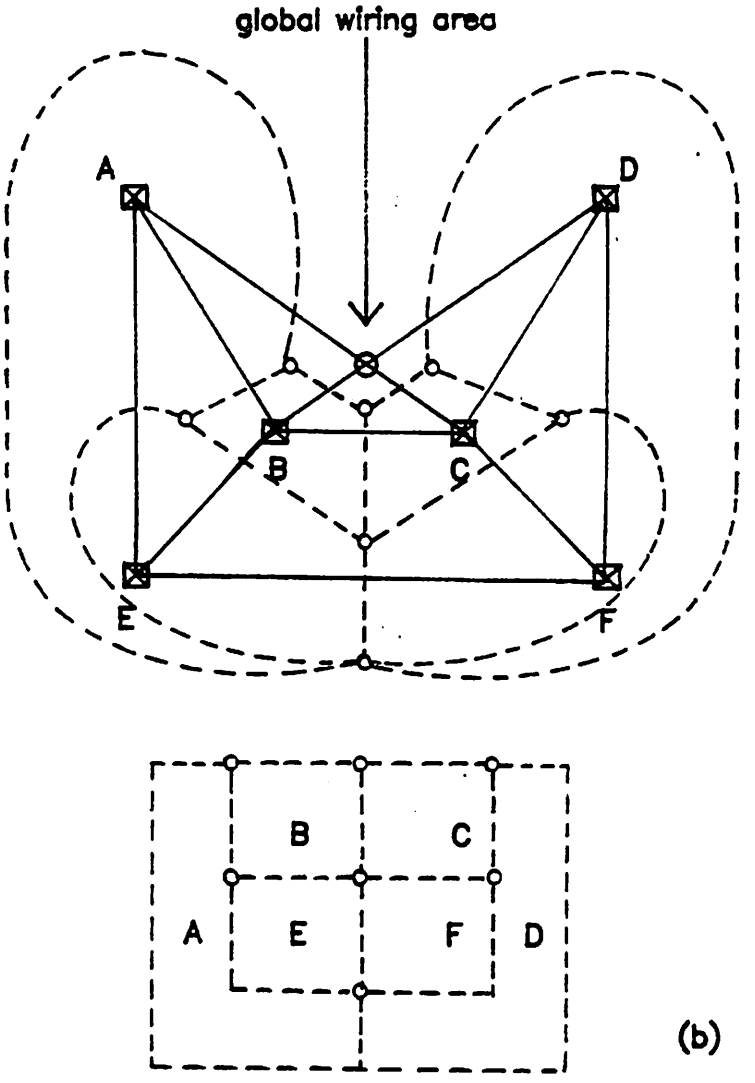


Figure 3.3(a) A planarized relation graph
and its dual (in dashed lines).
(b) The rectangularized dual graph
may be used as a chip plan.

SYSTEM	COMPUTER	EXAMPLE	CKT. SIZE	RUN TIME (seconds)	MEMORY USAGE	AREA INCRS.
CABBAGE	HP 1000 E RTE IV	T–FLIP FLOP	87	21	3200	
		5 T–FF's IN A ROW	407	191	11400	
		LATCH–DRIVER	290	220	8800	–2%
FLOSS	IBM 370/168	A	1200	29		+59%
		B	4000	100		
STICKS	HP 3000	3–INPUT GATE	20	20		
		C	3000	50 hrs		
SLIP	PDP 11/70 UNIX	INVERTER	14	28	CONSTANT	0%
		5–INPUT NAND	28	44	"	–15%
		D–FLIP FLOP	52	215	"	+30%

Table 3.1 Statistics of Layout Compaction Systems

in succession from the entire span of the layout, as shown in the example in Figure 3.4. Also shown in the example is the use of shear lines to join adjacent excess areas to achieve a higher layout density.

This algorithm was first implemented with a fixed-grid type data structure. Symbols and excess areas were contained in equally spaced grids such that uniform compression ridges can be developed easily. Recently, Dunlop [17, 32] has taken this approach one step further by partitioning the area of compaction and applying nonuniform compression ridges to relative-grid symbolic layout plans. Here the initial layout plan is first expanded, if necessary, so that symbol-to-symbol spacing is at least equal to the minimum design-rule requirement. Tightly connected elements in the expanded layout plan may be put into clusters via a partitioning process. Thus, the compaction of the overall layout may be carried out at two levels: the global compaction involves the reduction of spacing among the clusters and the local compaction performs the more elaborate compaction of the internal structure of each cluster. At both levels the width of each compression ridge is determined based on the spacing requirement of the individual clusters or symbols encountered by the extending compression ridge. The spacing requirement is calculated using design-rule analysis techniques in a working window, which is two maximum clearances higher than the present cluster or symbol and spans across the entire layout in the direction perpendicular to the present ridge. In addition, the trade-off between compactions in the x-direction and the y-direction may be considered during the development of a compression ridge for the local compaction.

A hierarchical layout scheme may be incorporated into the compression ridge method by representing the lower-level cells as forbidden regions,

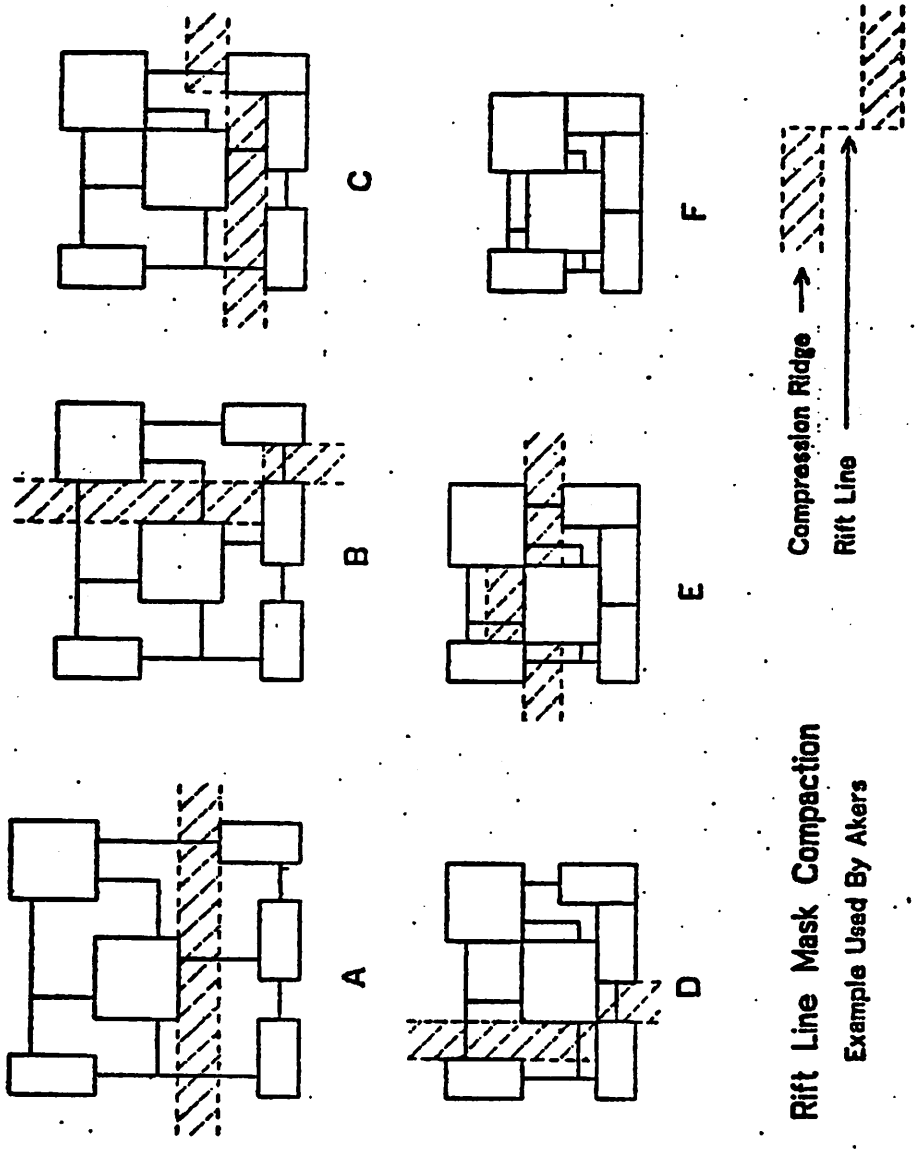


Figure 3.4 An example illustrating the compaction ridge method.

where compression ridges are not allowed to enter.

The main drawback of the compression ridge method is in its inability to determine the optimal location and ordering for introducing compression ridges. Frequently, an incomplete compression ridge must be discarded because all its possible shear lines have been exhausted. This trial-and-error scheme is especially time-consuming when compression ridges are developed based on elaborate design-rule checking techniques.

3.3.2. The Localized Placement Method

Instead of performing layout compaction by iteratively improving the compactness of a large and loose layout, as with the compression ridge method, the compaction problem may be treated as one in which each symbol is placed in sequence to construct a compact layout. With this latter compaction method, the location of a symbol is determined based on the locations of its neighbors which are connected to or may have potential conflict with the symbol. For this reason, such a layout construction method may be termed the localized placement method.

The localized placement method has been used in the program STICKS [18]. As an example of the data manipulation scheme used in STICKS, Figure 3.5 shows a graph representation for the compaction of three horizontal bars [33]. Each bar is assigned to a separate group represented by a node. (In this method, a group generally contains a rigid piece of symbolic structure.) Design-rule analyses are done for neighboring symbolic structures, which may be in contact or have potential conflict with one another. Based on the result of the design-rule analysis, a pair of forward-pointing and backward-pointing branches is inserted between the affected nodes. The same design-

rule analysis provides information on the required spacing, which are used as branch weights. The construction of a compact layout begins with the placement of one of the nodes at the boundary of the layout, say node a in Figure 3.5. Since node a is connected to nodes b and c in this example, the location of a affects those of b and c. Thus nodes b and c are considered next, not necessarily in any particular order. Suppose that node c is considered next, the branch (cb) causes the location of node b to be adjusted, which in turn affects the location of node c through (bc). Such adjustments are carried out until all branches are considered.

Note that if parallel paths exist in such a graph, only the longest of them determines the final location of each node in the path. Thus the manipulation of any branch not in the longest path only degrades the efficiency of this method. The longest path (or the critical path) principle is used to advantage by the compaction algorithms used in the program FLOSS [16] and the CABBAGE system (described in this report).

3.3.3. The Critical Path Method

Cho et al. described a building-block layout compaction program, FLOSS, in 1977 [16]. The use of the critical path principle in that program is announced recently [34].

In FLOSS, the edges of elements (building blocks and interconnection lines) are mapped into nodes and the size and spacing requirements are mapped into branches, as shown in Figure 3.6(a). The critical path of the resulting graph is then determined to give the locations of each of the element edges. An example of the critical path algorithm (also known as the longest path algorithm) can be found in Section 4.4.2. of this report, as well

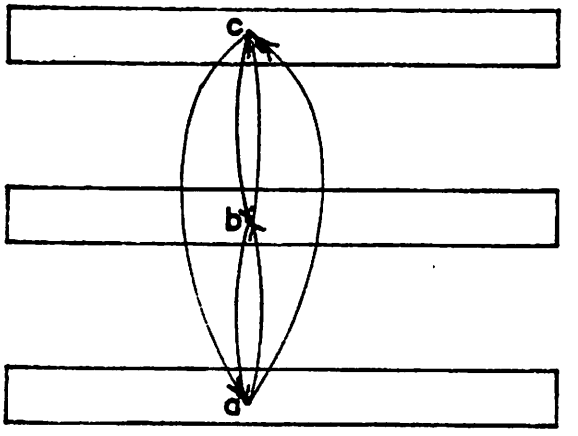


Figure 3.5 The graph representation
used in the STICKS program.

as textbooks on the subject of operations research [24].

The spacing requirements are generally lower-bound-type constraints of the form

$$i - j >= d,$$

which indicates that location i must be at least a distance d beyond location j. Such a constraint can be represented by a single branch with weight d from the node at location j to that at location i, as those shown in dotted lines in Figure 3.6(a). However, the element sizes are usually fixed and must be represented with fixed constraints of the form

$$i - j = d.$$

Fixed constraints can be mapped into the graph as a pair of forward- and backward-pointing branches having the same weight d, as those in solid line in Figure 3.6(a). If over-constraining conditions exist due to input errors, as shown in the example in Figure 3.6(b), the sum of branch weights around some of the loops introduced by the branch pairs may become positive. (The loop abcdefa in Figure 3.6(b) is such a positive cycle and indicates an over-constrained situation.) The particular critical path method used in FLOSS cannot detect this type of input error until the critical path is extended beyond a predetermined limit by looping through positive cycles repeatedly. In contrast, the graph representation used in the CABBAGE system does not generate such cycles and allows over-constraining situations to be detected easily. The graph representation and other techniques used in CABBAGE are the subject of the next chapter.

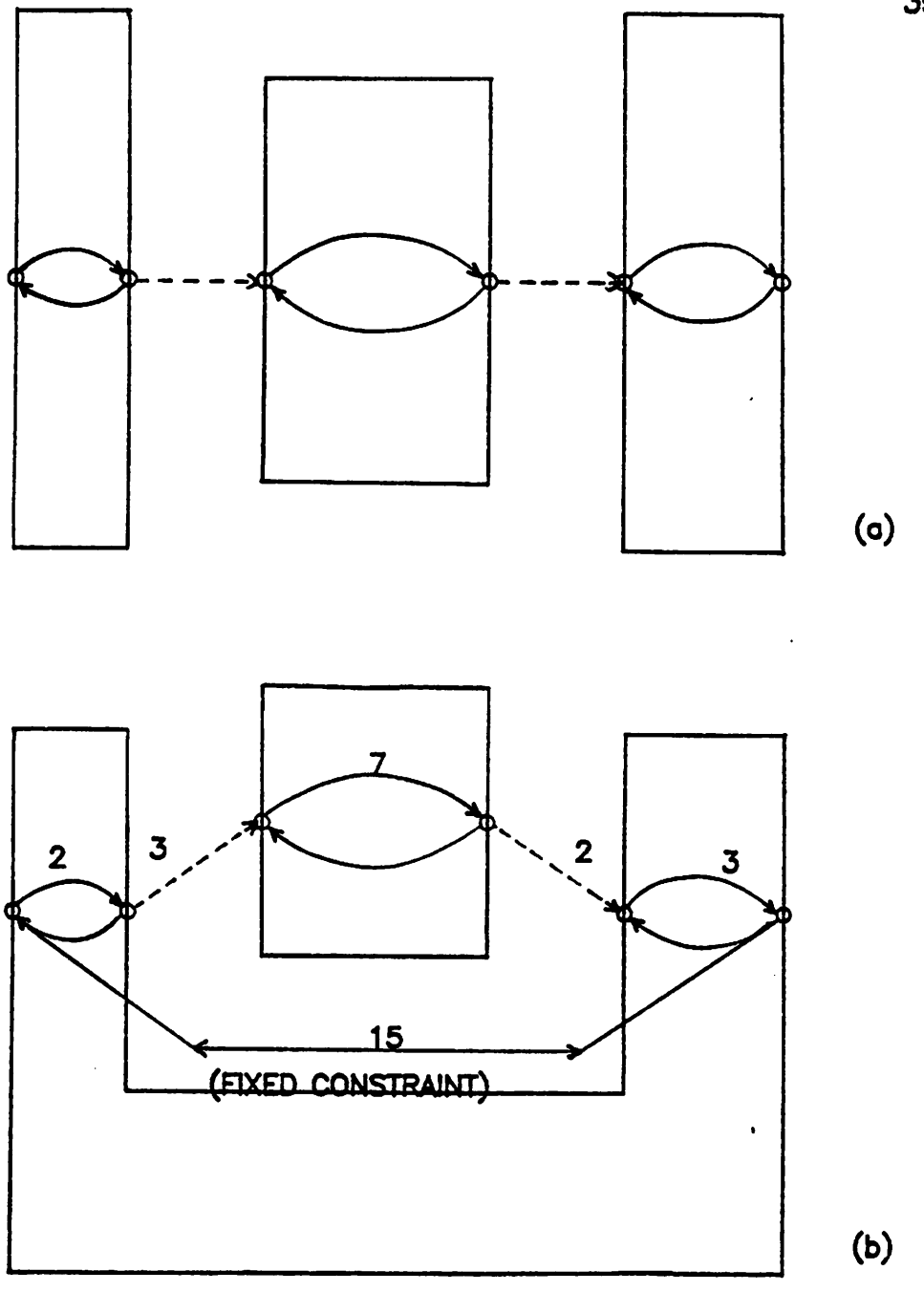


Figure 3.6(a) The graph representation used in the FLOSS program.
(b) An over-constrained structure.

CHAPTER 4

THE CABBAGE LAYOUT SYSTEM

4.1. Introduction

The CABBAGE layout system is a relative-grid symbolic layout system for generating LSI circuit building-block layouts. It is developed with the following objectives:

a) High computation efficiency. The layout system must generate a compact and correct geometric layout from a symbolic input as fast as possible to keep the designer actively in the design loop. The turn-around time should be on the order of a minute or less so that the user can receive almost continuous feedback on the layout topology he has chosen.

b) Good man-machine interaction. In this type of computer-aided design environment, the user must be able to obtain easily a layout whose configuration is similar to what he has in mind. Thus, in addition to giving the user continuous feedback, the design system must provide the user with means to guide the progress during the layout compaction process and to influence the compaction result.

c) Hierarchical layout capability. The hierarchical implementation of a layout not only works well with a structured LSI circuit design strategy but also helps

to maintain the short turn-around time that is essential to a good interactive design environment. In general, algorithms used in CABBAGE have an order of complexity in the range between n log(n) (for sorting) and $n^{1.5}$ (for design-rule analysis), where n is the total number of features (rectangles) in the layout. As such, better computation time can be achieved if larger building-blocks are made up with devices and lower-level building-blocks, where the representation of lower-level building-blocks is greatly simplified.

In the following sections an overview of the CABBAGE system is presented. The design considerations and algorithms for achieving the above objectives are also described.

4.2. An Overview of the System

At the present time, the CABBAGE system consists of two programs: The program GRLIC (Graphics Routines for Laying-out Integrated Circuits) is an interactive graphics editor for generating the initial symbolic layout plan and manipulating interim compaction results at the symbolic level. GRLIC allows the user to perform most of the essential graphics editing functions, such as adding, deleting, and copying symbols and files. It supports a set of layout symbols whose size and orientation can be varied with commands issued by the user. Thus, most of minimum size, overlap, and enclosure design-rules are enforced by the input processor of GRLIC as default minimum values and in the form of standard symbol structures. The second program, PRSLI

(Packing a Relative-grid Symbolic Layout of an IC), is the layout compactor that generates the correct geometric layout. It analyzes layout design-rule requirements and constraining conditions for each symbol in the symbolic layout plan and compacts the layout in horizontal and vertical directions as much as is allowed by these requirements. The user can also request PRSLI to put in jog points automatically after the initial compaction step and perform the compaction operation once more to achieve a possibly more area-efficient layout. The compaction and the automatic jog-point introduction algorithms are described in detail in later sections.

There are other more drastic topological modifications, such as rotation and mirroring of symbols or a complete reorganization of the layout plan, which the user may wish to perform after seeing the initial compaction result. Thus the two programs are designed to be used alternately to construct, compact, and modify the layout and its corresponding topological plan until the user is satisfied with the result. The communication between these programs is carried out through a simple disk file. This file contains only the most basic information: a layout plan is stored as a collection of symbols (devices, lines, constraints, etc.) represented by their individual Symbol Description Blocks (SDB's). The layout symbols are described in the SDB's by their individual symbol type, orientation, center location and size. A precise data organization for the SDB is shown in Appendix 1.

The use of a common data representation for the symbols allows each program to employ the best internal data structure for its particular function. For example, GRLIC treats the layout plan as a set of properly located individual symbols and keeps no information regarding their connectivity and adjacency. Since the symbols do not interact with one another, the data

structure used by GRLIC is one that can be grown or shrunk easily to maximize the editing efficiency. In contrast, the layout compactor PRSLI must first develop connectivity information so that symbols can be properly separated or merged during the compaction operation. Here the connectivity information is in the form of electrical node numbers. (Note that input to circuit simulation programs [35, 37, 36] may be derived easily from the node number information and symbol type and size specifications.) In addition, for purposes of geometric layout design-rule analysis and displaying the complete geometric layout, PRSLI interprets properly connected symbols as rectangular polygons. Thus, at any time during the layout process, the user can view his layout in both the symbolic form with GRLIC, and the geometric layout form with PRSLI.

The separate program and simple common interface approach makes the CABBAGE system easily extensible. Potential future additions to the CABBAGE system, such as electrical performance simulation, function-to-layout consistency check, and mask pattern generator tape-making programs, can be implemented as individual program modules that derive necessary information from the common disk file.

4.3. The Drawing Rules for the Layout Plan

Before describing the main body of the compaction procedure, it is useful to introduce two rules for drawing the symbolic layout plan that greatly simplify the compaction procedure itself.

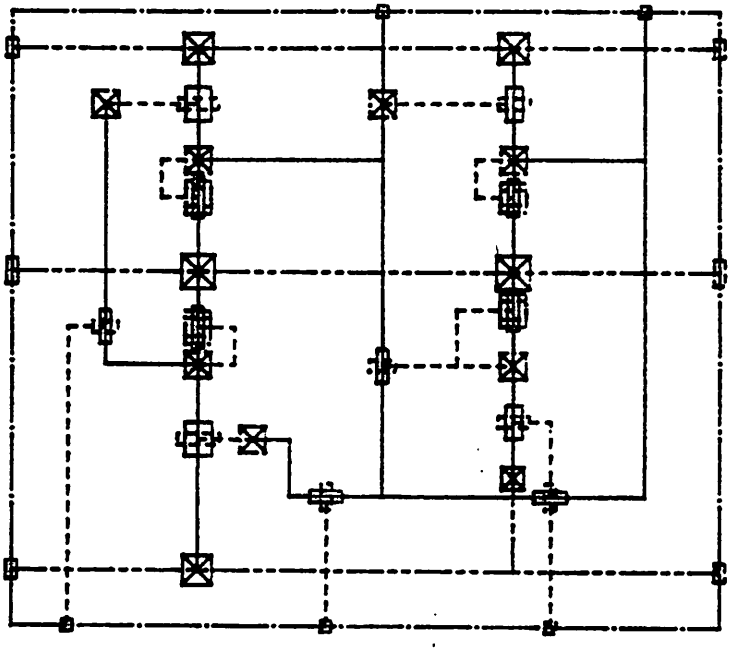
At the present, a symbolic layout plan used in CABBAGE is made up of two types of elements: lines and point structures. The point structure is a layout entity or a circuit device. For example, transistors and contacts are

all considered point structures in the CABBAGE system. The line element is the usual interconnection line between circuit elements.

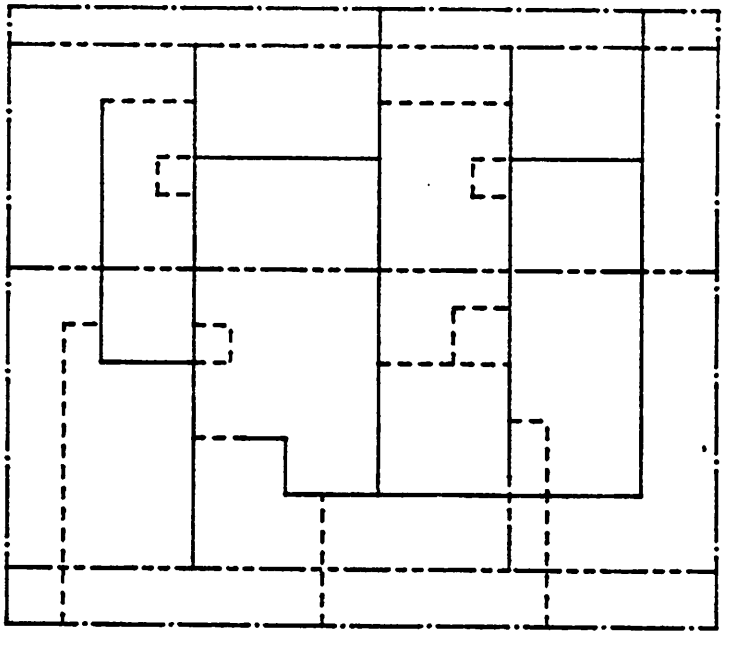
In terms of layout compaction, the characteristics of the line element differs from that of the point structure element in two aspects: The line element can be stretched and shrunk, while the size of a point structure element must stay fixed. Also, a point structure, such as a transistor, may require more than one electrical node to describe its connectivity in the circuit.

In order to simplify the implementation of the stretch and shrink capabilities of line elements, the CABBAGE system requires all lines in the symbolic layout plan be properly terminated. A line may be terminated by an orthogonal line or a special point structure, called the terminal element, at the boundary of the layout plan. In fact, in the absence of point structure elements, the remaining layout plan still is a connected network as shown in Figure 4.1. With such a connected network, the end points of any line element can be derived readily from the center location of connecting point structures and orthogonal line elements. Thus, this rule on the proper termination of line elements provides the essential mechanism for transmitting changes made in one direction to the orthogonal direction.

In conjunction with this first drawing rule, the second rule requires that all point structures be located at the intersections or end points of lines so as to simplify the automatic assignment of node numbers. As such, the point structure elements serve to identify the types of line intersections, both in the data structure for the compaction operation and in the graphical symbolic layout plan. In addition, the actual sizes of point structure elements are used in the symbolic layout plan as a visual aid to the user. This practice



(a)



(b)

Figure 4.1(a) A symbolic layout plan of a T--flip flop.
(b) Lines remain connected when point structures are removed from the layout.

helps prevent the user from piling up elements and simplifies the logic in the compactor for distinguishing the connectivity of the individual elements. Abbreviated names are used in place of graphical symbols to display point structures in symbolic layout plans in other layout compaction systems [17, 18].

4.4. The Layout Compaction Algorithm

There is usually a certain amount of unused space or dead area in a "compact" layout. The dead areas exist because geometric structures surrounding them may be held apart by larger structures in other parts of the layout. The compaction algorithm in PRSLI uses this fact to advantage; it finds and improves only the limiting or the most-constraining symbolic structures which directly affect the compactness of a layout. Other (non-constraining) symbols are placed next based on the location of the constraining symbols to generate a compact geometric layout.

A technique analogous to that used in polar graph-based building-block placement can be used here to determine the most-constraining structures. As described in Section 3.2.1. [8, 31], the building-block placement can be optimized by manipulating the longest paths, which correspond to the most-constraining building blocks, in the polar graphs describing the desired horizontal and vertical placement relations. In PRSLI, two similar, but simplified graphs are used for the sole purpose of determining the longest paths or the most-constraining structures in the horizontal and the vertical directions.

4.4.1. The Graph Representation

A major difference between the relative-grid symbolic layout compaction problem and the building-block placement problem is the way the connectivity of a layout is treated. The connectivity of the symbolic layout plan must be kept unchanged throughout the compaction operation. For simplicity, this may be done by preserving the topology of the layout plan. In contrast, the objective of performing building-block placement is to devise a good topology whereby the blocks can be tightly packed. Since building blocks usually are linked up in a separate routing phase, the connectivity among blocks is not as strong a factor in determining the adjacency as in the case of layout compaction.

As a result, a proper grouping of symbols into "building blocks" is essential in order to map a symbolic layout plan to a polar graph-like representation. If each symbolic element is considered as a block by itself, as in the example in Figure 4.2(a), it is a relatively complicated matter to ensure that connected elements do not fall apart as a result of the compaction operation, as in the example in Figure 4.2(b). To prevent this unwanted splitting from occurring, PRSLI considers all topologically connected elements sharing the same vertical or horizontal center line as belonging to a single block and moves them as a group during the compaction operation. An example of such a grouping of elements is shown in Figure 4.2(c). Note that elements having different electrical potentials but the same horizontal or vertical location may be put in the same group.

In the graph representation of the symbolic layout plan the groups are mapped into nodes and the separation requirements between groups are mapped into branches. In particular, the weight of each branch is equal to

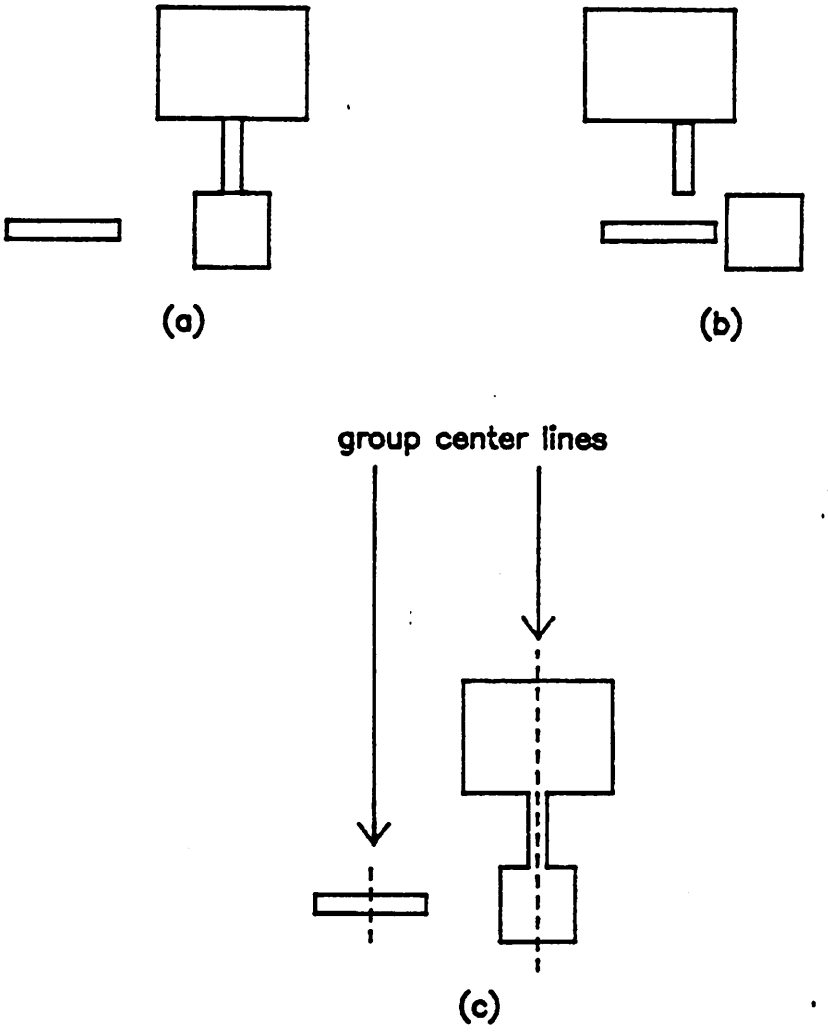


Figure 4.2 An example of a compaction in the horizontal direction. If each element (rectangle) is treated as an individual entity, as in (a), the compaction process may break apart connected elements, as in (b). The grouping shown in (c) holds elements together during the compaction.

the maximum sum of a) the half-widths of any two directly adjacent elements in the two groups separated by the branch and, b) the corresponding spacing requirement between those elements. An example of this mapping is shown in Figure 4.3. Note that this mapping is different from that used in the Tutte rectangle dissection method but achieves the same effect as the said mapping. In the rectangle dissection method, boundaries and individual areas of constituent rectangles are mapped to nodes and branches, respectively, in a polar graph. With the Tutte-type mapping, rectangles can be made to share a common boundary line if their corresponding branches are connected to the same node. Similarly, with the mapping used in PRSLI, unoccupied spaces are made to share a common group of symbolic elements, thereby providing the necessary separation area around that group.

The horizontal and vertical structural relations of a symbolic layout plan are represented with separate graphs in PRSLI. For simplicity in constructing the graphs, the horizontal groups are not included in the vertical graph, and vice versa. The two graphs are no longer dual because of this simplification of the grouping process.

4.4.2. Finding the Longest Path

The graphs obtained with the mapping described in the previous subsection are planar, directed, and noncyclic graphs. Moreover, since branches are put in with a top-down, left-to-right sequential design-rule analysis, the predecessor-descendant relationship of each node can be obtained easily. A detailed description of the design-rule analysis process is given in a Section 4.5.

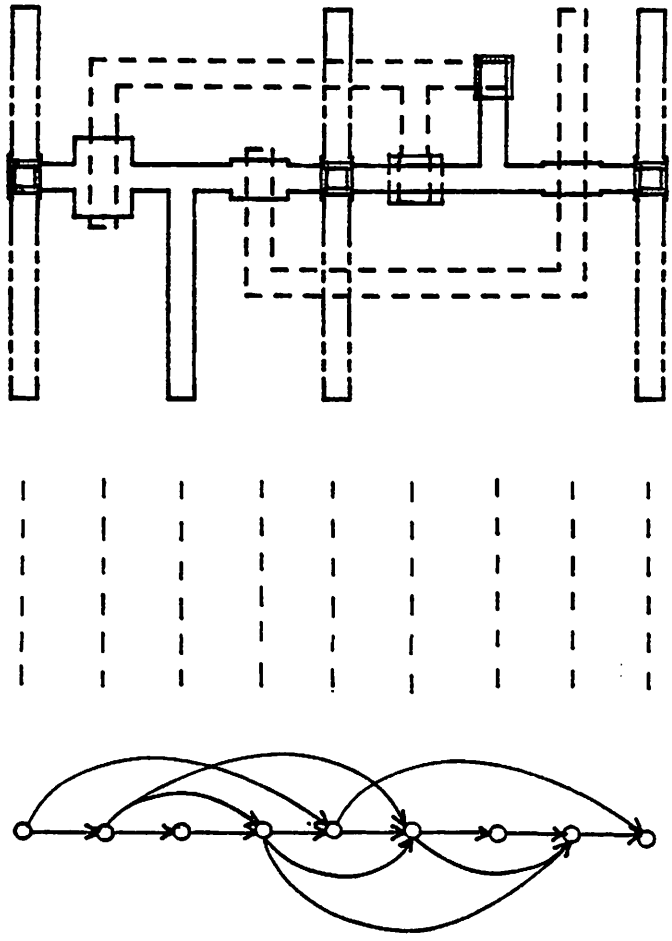


Figure 4.3 The graph representation of the layout in the horizontal direction.

It is a simple matter to determine the longest path through such planar directed graphs with known descendants for each node. (The determination of the longest path through such a graph is known as the critical path method for analyzing the project network in the field of operations research [24].) Informally, the longest path to a given node is taken as the maximum of

a) the present path length to the node, and

b) the sum of the weight of the branch between the present node and one of its predecessor nodes, and the longest path length to that predecessor node.

A formal description of the longest path algorithm is shown below with a self-explanatory programming language. (This language follows the general direction of the Ratfor language [38]. The rules of the language are a) simple operations are separated with a semicolon and a block of operations is enclosed in a pair of braces, and b) the assignment of i to j is written as j = i.) Before the presentation of the algorithm, however, a few functions and arrays used in the algorithm must be defined. (Here the word "function" is used loosely and should not be viewed rigidly as that used in some formal computer programming languages.) Let the graph be described by three functions **nd**(i), **dnode**(i,j), and **bw**(i,k). The function **nd**(i) gives the number of descendant nodes of node i. The function **dnode**(i,j) gives the node name or number of the j-th first-generation descendant of node i. (Note that the descendants of a node need not be arranged in any particular order.) The function **bw**(i,k) gives the weight of the branch between nodes i and k. In addition, an array np and a first-in first-out queue is used to facilitate the longest path calculation. The individual elements of np, np[k], are initialized

to the number of predecessor nodes of node k. The longest path length to node i, lp[i], is then obtained as follows:

```
initialize lp[i] to zero for all nodes;
put nodes with zero predecessor into a queue;
while the queue is not empty {
    take the first node in the queue and call it n;
    for each of its nd(n) descendants {
        get the node name k = dnode(n,j) of a descendant;
        lp[k] = max( lp[k], lp[n] + bw(n,k) );
        np[k] = np[k] - 1;
        if np[k] is equal to zero then
            add node k to the end of the queue;
        }
    pop the queue (thus deleting node n from it);
    }
```

The path length lp[i] to each node is used as the center location of groups in the direction of compaction. As a result, locations of groups not in the longest paths tend to be biased towards the first group. (Note that special topological properties devised by the user, such as the symmetry in the placement of similar transistors, in a symbolic layout plan cannot be preserved because of the bias applied by this compaction strategy. Further research must be carried out to incorporate different compaction strategies so that the user has more control over the compaction process.) In PRSLI, a force-directed placement technique [26] is applied to these non-constraining groups to pull them away from the first group and to put approximately equal unoccupied space on their two sides. The same force-directed place-

ment principle forms the basis for the automatic introduction of jog points.

4.4.3. The Automatic Introduction of Jog Points

Jog points are locations at which a line can be bent and then continued a short distance away. Such a bending may result in a more area-efficient layout as shown by the example in Figure 4.4. (A higher area efficiency can not be guaranteed unconditionally since the bent line must occupy some space in the direction of the bend.)

Although jog points may be put into the symbolic layout plan manually at any stage of the compaction operation, the longest path information derived during the initial layout compaction operation may be used to determine all possible jog points at once. Specifically, jog points may be included in a group where the longest path goes into and comes out from two different intervals of the group. In essence, the two segments of the longest path may be viewed as force vectors which exert a torque on the group. The introduction of a jog point allows the group to be torn apart by the torque.

In the actual implementation, the intervals over which the maximum separation requirement exists between two groups are recorded during the design-rule analysis and the record is accessible to the descendant group. (Such intervals are shown in dark lines for the example in Figure 4.5) With such a setup, the automatic jog point insertion algorithm can be described formally below. In the description, the function **sep**(i, j) gives the center-to-center spacing between groups i and j after the compaction operation. The function **minsep**(i, j) gives the minimum required center-to-center spacing between groups i and j. The function **mxint**(i, j) facilitates the inspection of the record containing the intervals over which the maximum

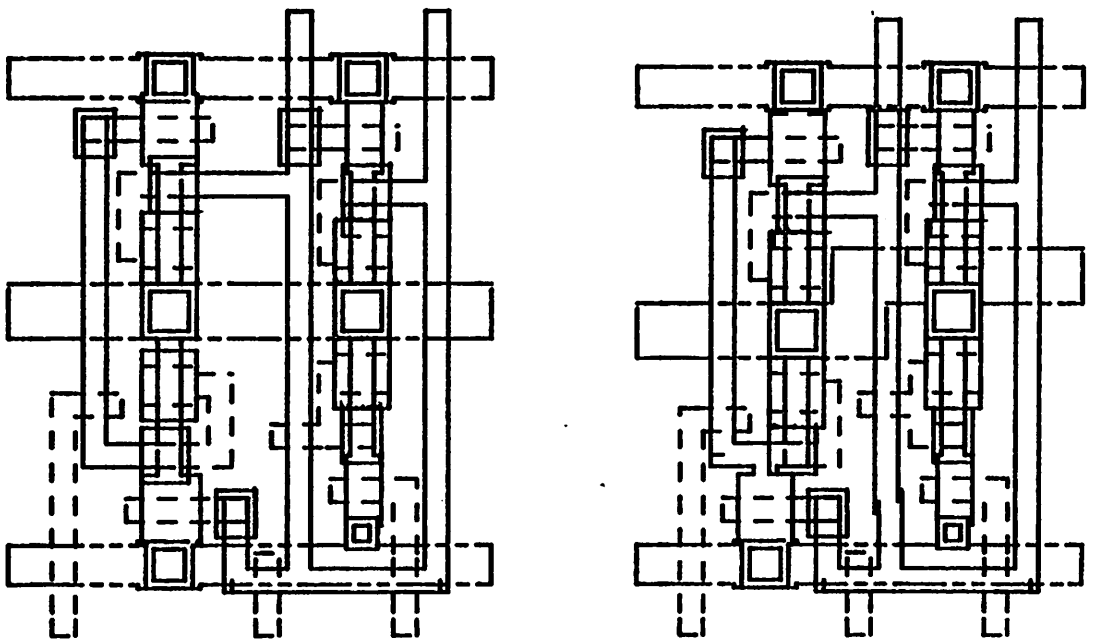


Figure 4.4 The T–flip flop before and after jog generation.
The new version on the right is 9% smaller.

spacing exists between the predecessor group i and the descendant group j. Thus, two groups are in the longest path if **sep**(i, j) equals **minsep**(i, j).

```
for each group i in the direction of compaction {
    for each of its descendant groups j {
        if sep( i, j ) is equal to minsep( i, j ) then {
            for each descendant group k of group j {
                if sep( j, k ) is equal to minsep( j, k ) and
                  there are intervals in group j not covered by
                  either mxint( i, j ) or mxint( j, k ) and
                  the covered intervals above and below them
                  are from different groups, then
                  put a jog point in the uncovered interval
                }
            }
        }
    }
```

The inclusion of a jog point involves the splitting of the group into two parts at the point of the jog and the addition of a line element at that point connecting the new group to the original group. The actual jogs are brought out with a subsequent compaction in the direction of interest. Thus, jogs may not appear at all the potential jog points found with the above algorithm.

4.5. The Enforcement of Design Rules

Geometric layout design rules specify the permissible structural relationship among elements used in a circuit layout. These rules are derived on the bases of production tolerances and basic device physics limits to ensure

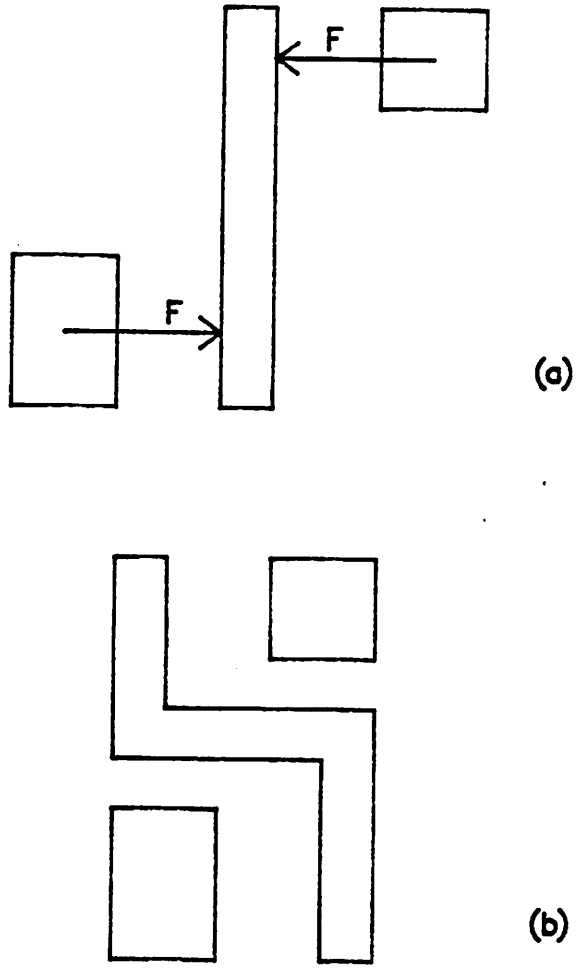


Figure 4.5 The torque applied by neighboring groups, as shown in (a), bends the middle group, as shown in (b).

the successful fabrication of a circuit.

Design rules are enforced by the CABBAGE system automatically to ensure the efficient generation of a layout free of geometric errors. (A summary of the typical types of design rules used in the industry and their implementation in the CABBAGE system is included in Appendix 4.) For purposes of presentation in this section, most of the common design rules may be categorized as follows:

a) the minimum element sizes, as in Figure 4.6(a). This type of rule is used to make sure that, for example, an element will not be eliminated even with severe over-etching.

b) the minimum overlap and enclosure of geometric features in an element, as in Figure 4.6(b). These rules are used to guarantee that, for example, even with the worst misalignment of different mask layers, features in an element will touch one another as intended.

c) the minimum spacing between elements, as in Figure 4.6(c). These rules give the minimum required clearance beyond which distinct elements will not touch even with the worst misalignment.

Most of the design rules in the first two categories are enforced by GRLIC, at the time the symbolic layout plan is drawn, as the default minimum size or the standard structure of an element. For example, when the user requests a simple transistor symbol, a structure consisting of a section of polysilicon intersecting a section of diffusion is automatically supplied by

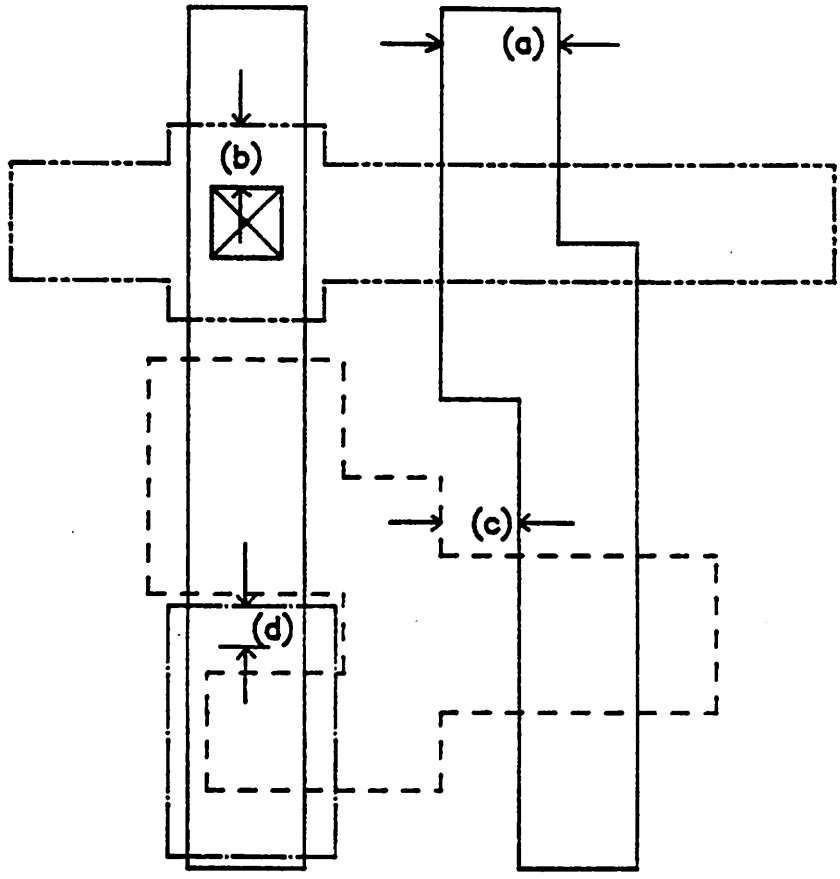


Figure 4.6(a) The minimum size of a diffusion line.
(b) The minimum metal overlap of a contact window.
(c) The minimum spacing between a polysilicon and a diffusion line.
(d) The extra window overlap of the buried contact.

GRLIC. Moreover, the overlapping area of this structure (the active channel area of the transistor) is constrained to be at least the minimum area required by design rules.

Design rules that are functions of a given topology must be applied dynamically by PRSLI during the compaction operation. These rules include all the spacing rules, as well as a few rules governing the overlap requirements of a contact area. These latter rules typically call for an increase in contact coverage in the direction of the contacting line to ensure the proper electrical connectivity, as in the example in Figure 4.6(d). Since the direction of the contacting line can change as a result of the compaction operation, such rules on extra overlap can be applied only at the end of each compaction operation.

It is interesting to note that the minimum size requirement described in the first category above may be viewed as a minimum (internal) spacing requirement between the two edges of an element. This viewpoint allows the minimum size and spacing rules be treated in a unified manner. However, PRSLI treats the two separately for two reasons. First, with the present implementation, treating the size of an element as a spacing requirement makes the size variable and increases the complexity of the force-directed placement scheme described in Section 4.4.2. Second, it is often undesirable to enlarge the size of an element above the minimum required value for such enlargements may increase line capacitances and reduce the drive capabilities of transistors. Special constraints may be used to limit such enlargements at the expense of increased program complexity. (The optimization of the size and drive capability of a transistor with respect to the available space is considered by Ivannikov and Sipchuk [20].)

Spacing design-rules are analyzed between neighboring groups by PRSLI. Specifically, PRSLI determines the enclosing neighboring groups above or to the right of a given (primary) group, and proceeds to analyze the spacing requirements between the primary group and its neighbors. For each constituent element in the primary group, its edges facing the neighboring group is determined and checked against those of the neighboring group mask by mask, as shown in Figure 4.7. For neighboring but distinct groups, a branch is inserted between the nodes representing the two groups. The maximum sum of element half widths and the appropriate spacing design-rules between the groups are used as the weight of the branch. For two groups that can be merged, PRSLI inserts a branch with zero weight between the nodes representing these groups. Although such a branch could be left out to allow the mergeable groups more freedom of movement, it becomes quite involved to redetermine the enclosing neighborhood for groups examined prior to the current primary group in the event that the mergeable neighbor moves past the primary group. For this reason, the zero-weight branch is used as a compromise.

Because the spacing design-rule analysis is done at the lowest level of mask shapes, proper diagonal-spacing requirements can be calculated easily. With diagonal-spacing requirements included in the graph representation, a symbolic layout may be compacted to meet all design-rule requirements, in theory, in just one horizontal and one vertical compactions. However, the application of extra overlap rules after the spacing-rule analysis makes it necessary to carry out more than the minimum number of compaction operations to obtain correct layouts in most cases. The potential design-rule violations introduced by the extra overlaps may be eliminated with additional compaction operations after a layout has been substantially compacted and

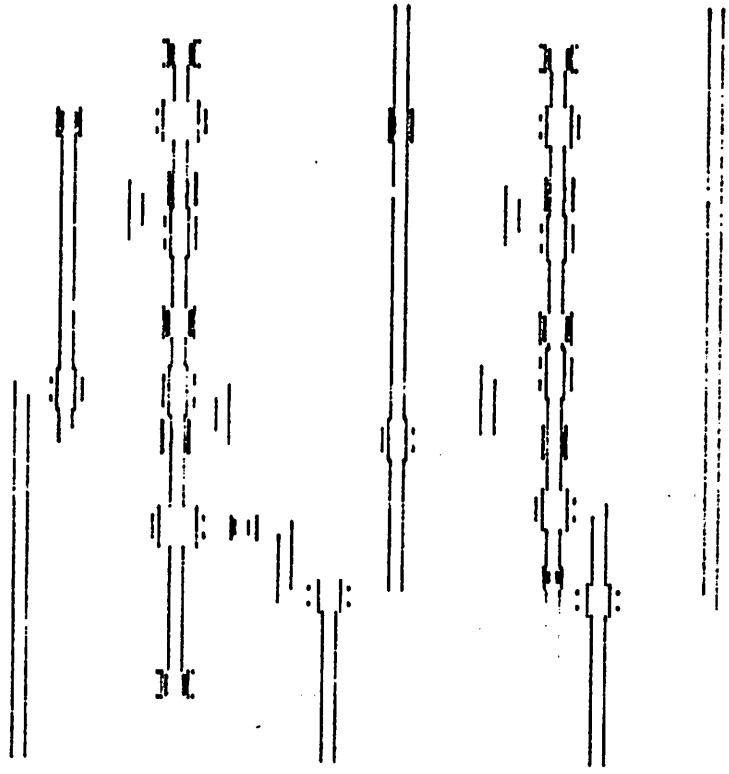


Figure 4.7 Vertical edge components of the T–flip flop in Fig. 2.5.

directions of contacting lines stablized. An example requiring this type of iteration is given in Figure 4.8.

Iterations must be carried out until all design-rule requirements are met. Thus the iterative process may be terminated as soon as the spacing requirements calculated at the present iteration are not in conflict with the center-to-center distances of the corresponding groups determined at the previous iteration. Note that a more compact layout may be produced with additional iterations if not all locations of groups remain the same during two consecutive compactions in a given direction. Thus, in addition to reporting the correctness of the solution of the previous compaction operation, PRSLI reports the movement of the groups during the present compaction operation to let the user determine if he would proceed further.

In order to perform the design-rule analysis, PRSLI must develop electrical connectivity information for all elements to determine mergeability. Further, it must convert the symbolic representation of groups into actual mask shapes for the detailed design-rule analysis. The next two subsections describes how this information is derived.

4.5.1. Determination of Electrical Connectivity

PRSLI represents electrical connections in the form of electrical node numbers. Every element is assigned a node number (three in the case of a transistor element) at the beginning of the compaction operation according to its electrical connection. The global connectivity information provided by the node numbers is used throughout the entire compaction process: the node numbers are used to determine whether two elements may be merged and the possible partial coverage (overlap) of an element by another element

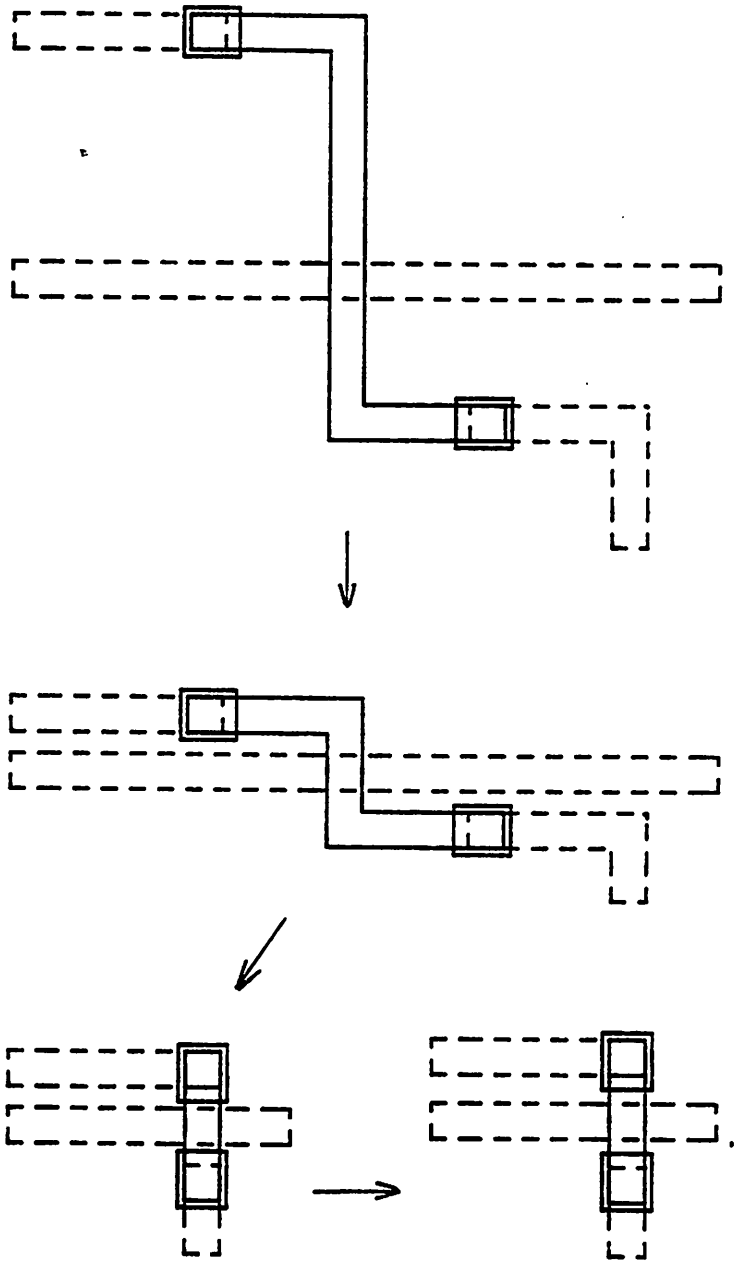


Figure 4.8 An example showing the need for iteration for the directional enlargement of buried contact windows.

remotely connected to it. (Two elements are remotely connected if they are connected through other elements sharing the same electric potential with them, as in Figure 4.9.)

Node numbers may be assigned with a path-finding type of algorithm [39]. For simplicity, assume each element is tied to only one node. The node numbering algorithm used in PRSLI begins by assigning a distinct, monotonically increasing temporary node number to each element sequentially. The elements are then examined in the same order: the (temporary) node number of the present element is compared with that of the element directly connected to it. All elements with the larger (temporary) node number are then assigned the smaller (temporary) node number. This node equivalencing operation terminates when all elements have been examined.

4.5.2. Conversion of Symbols into Actual Geometric Shapes

When a group of symbols is compared with its neighboring groups to determine the necessary spacing requirements, the comparison must be done at the detailed mask shape level to guarantee that the single branch weight used to separate the two groups reflects the true spacing requirement. PRSLI performs this comparison by converting the symbolic representation of the two groups in question into edge segments placed at a half of an element width away from the group centers. Figure 4.7 shows the edge segments when the primary group on the left side is compared to one of its enclosing neighbors to the right.

Two factors must be considered in developing the edge segments of a group of symbols. First, since the interconnection configuration of elements may change from one compaction operation to the next, the edge segments

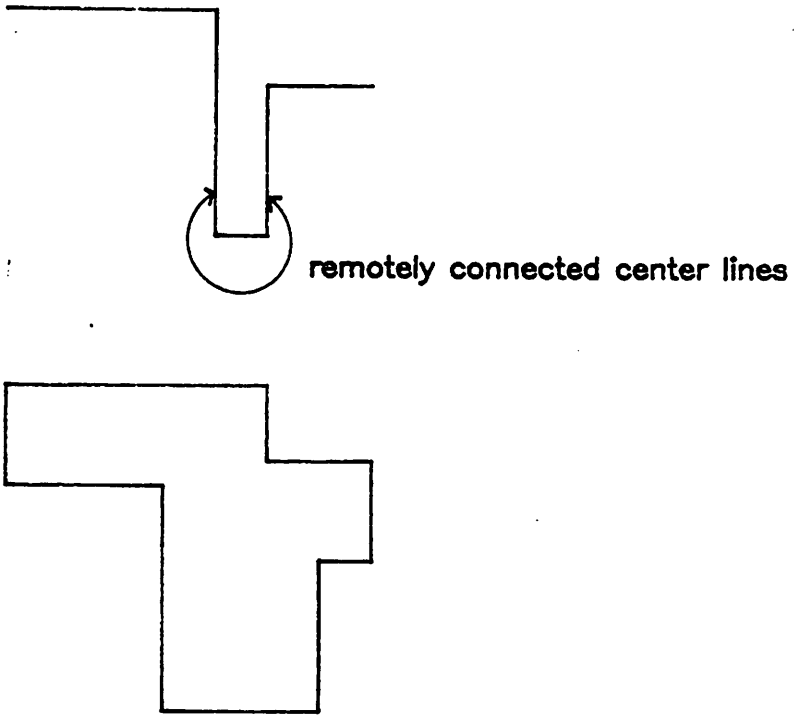


Figure 4.9 Remotely connected center lines may merge into one polygon if they are wide enough.

must be developed for each compaction operation to reflect the most current configuration. Consider, for example, the structures (connected areas) in Figure 4.10(a). As shown in the figure, several of the lines in the center structure must be extended so that they form smooth joints with their connecting lines. It is clear from Figure 4.10(b) that, after a compaction operation in the horizontal direction, the extended segments in that structure are no longer the same as the extended segments before the compaction. Second, for the proper operation of the compaction process, it is not sufficient simply to disregard or to remove hidden segments of the elements. Hidden segments may reappear as a result of the compaction. Consider again the movement of the center structure during the horizontal compaction of the example shown in Figure 4.10(a). It is necessary to remove the hidden segments of the two vertical lines, as shown in Figure 4.10(c), for the vertical line on the top to move past the vertical line below it. At the same time, however, the "removed" segment of the lower vertical line must be examined against the left edge of rectangle R. Had rectangle R been placed in a lower position, as shown in Figure 4.10(d), such an examination would place a corner-to-corner spacing requirement between the rectangle and the upper corner (the "removed" segment) of the lower vertical line. (Similarly, the "removed" segment of the upper vertical line must be examined against rectangle L.) Because of such a spacing constraint, the compacted result in Figure 4.10(d) is different from that in Figure 4.10(b) Thus it is necessary to record the covering segment of a hidden segment. This knowledge allows the compactor to discriminate the neighboring segments of a hidden segment and to skip the examination of the covering segments.

In all, four types of designations of edge segments are needed to describe the outline of a given group correctly. The normal and the gap

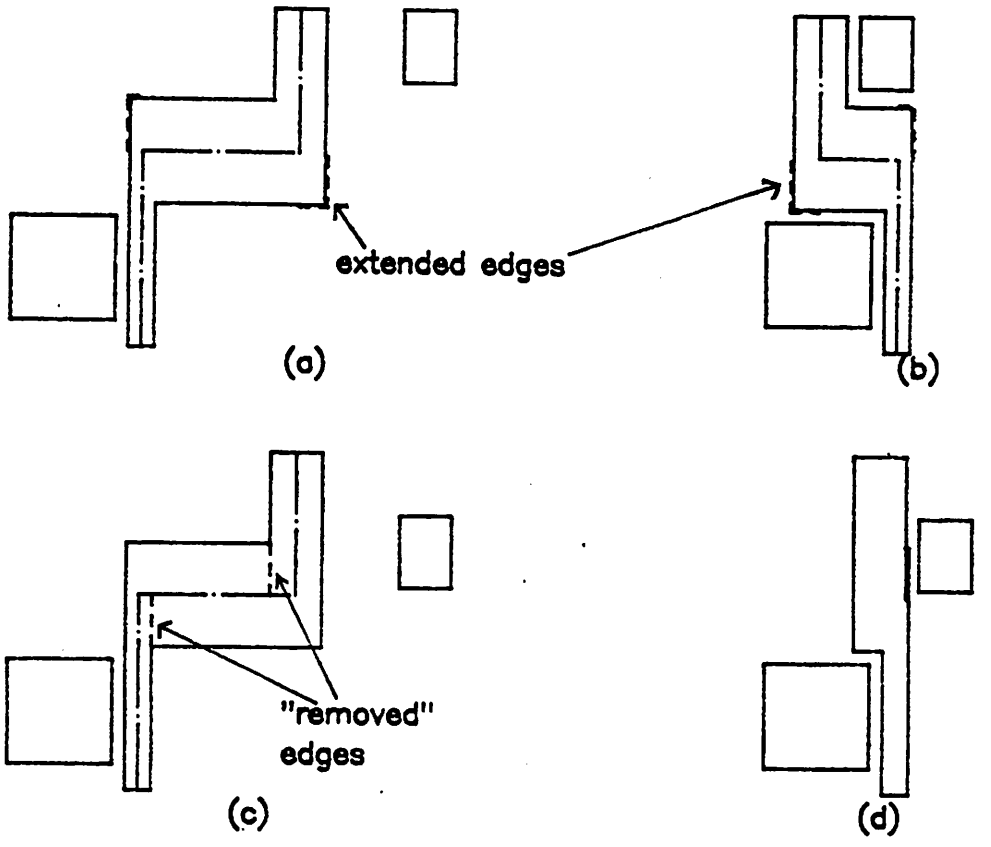


Figure 4.10 Edge type designations for the rectangular polygon representation of center lines.

segments are used to describe the presence and the absence, respectively, of a line segment in the individual symbols. A temporarily covered segment is one that may reappear as a result of the compaction, as described in the previous paragraph. A fully covered segment, such as the vertical edges of the transistor in Figure 4.11, always stays hidden and needs not be examined against any other edge. (Note that the design-rule analysis routine must detect an initially fully covered group and anchor it to its neighboring groups to prevent the corresponding node from floating in the graph representation. An initially fully covered group may result when the length of a line is shrunk to zero due to the merger of its connecting groups in the orthogonal direction.)

The edge segment designations are used not only for design-rule analysis purposes but also for plotting the compaction results in rectangular polygon form. The normal edges from each of the four sides of a group are drawn to form a complete outline of the group.

4.5.3. The Table of Spacing Design-Rules

Since design-rules governing the spacing between elements are used repeatedly throughout the compaction process, PRSLI organizes these rules in the form of a table to facilitate the determination of the spacing requirements. The table used by PRSLI is shown in Table 4.1.

Note that the entries of the table contain not only the usual mask levels but also some compound mask levels, such as the level containing the active channel area of a transistor. The compound mask levels are included in the edge segments, just as the regular mask levels, when PRSLI performs the design-rule analysis. For example, when a transistor symbol is converted

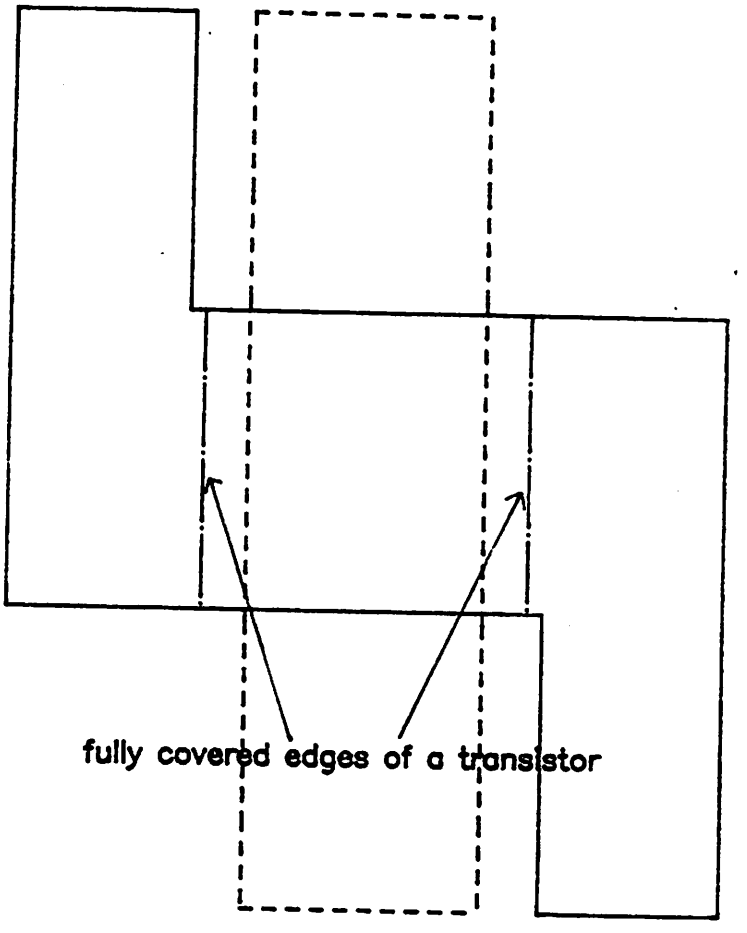


Figure 4.11 The vertical edges of the transistor are covered fully by the connecting diffusion line.

into its geometric equivalent, the corresponding edge segments contain a drain-to-source diffusion edge, a polysilicon gate edge, and an active channel edge. Thus, the rule for separating a neighboring diffusion line and the active channel of the transistor can be applied readily.

The ability to use compound mask levels to simplify the design-rule analysis is just one of the many advantages of the symbolic layout technique over the conventional hand-drawn or interactive graphics-aided layout schemes. In the latter schemes, the components of a circuit element, such as the channel and the gate of a transistor, typically are stored separately as components on different mask levels in the computer data base. With such a separation, the design-rule check program must reconstruct the transistor from components on different mask levels (by logic AND, OR and other operations) in order to inspect the compound mask levels.

At the present, the Table of Spacing Rules does not contain more involved rules such as the one requiring extra spacing between two diffusion lines if there are contact windows near the edge of each of the diffusion lines. (An example of such a configuration is shown in Figure 4.12.) However, these types of rules may be added easily by expanding the table to include additional compound mask levels, such as a special diffusion-edge type with nearby contact windows.

4.5.4. The Computation Complexity of the Design-Rule Analysis

PRSLI employs a computationally efficient design-rule analysis scheme which examines only the enclosing neighbors to the right or above a given group. Since all groups are sorted by their individual center locations before each compaction operation, all groups can be covered properly as the

DIF	PLY	MET	RNX	ACT	CNT	BUR	IMP	TRM	
5	2	X	5	X	6	4	X	X	DIFFUSION
	5	X	5	X	6	4	X	X	POLYSILICON
		6	X	X	X	X	X	X	METAL
			5	5	5	5	5	5	RUNX
				5	5	4	4	X	ACTIVE AREA
					10	2	X	X	CONTACT WINDOW
						4	X	X	BURRIED CONT.
							X	X	IMPLANT REGION
								X	TERMINAL POINT

Table 4.1 Table of spacing rules.

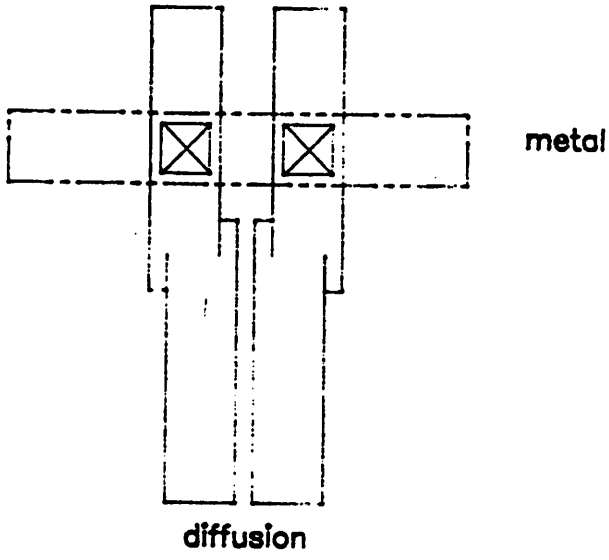
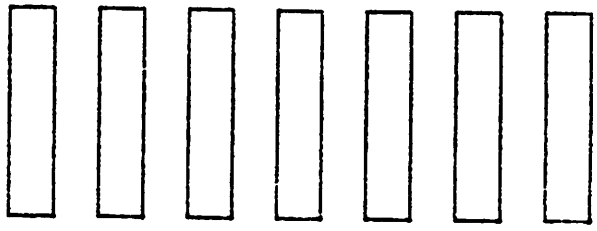


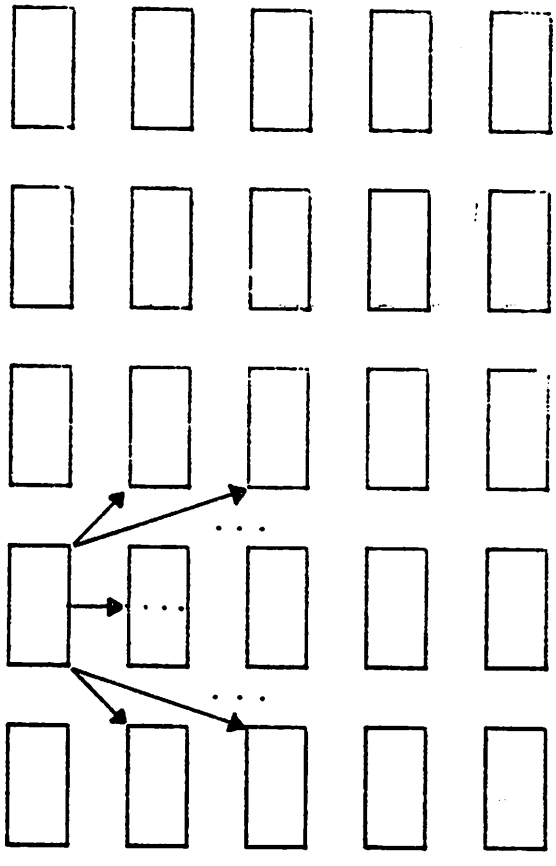
Figure 4.12 The spacing between diffusion lines is widened in the vicinity of contacts.

design-rule analysis routine sweeps across the layout in the direction of the compaction.

It is clear that, in the best case, the run-time would be linear for this design-rule analysis technique. An example which gives the best run time is shown in Figure 4.13(a). Assume that all bars are on the same mask level and the minimum spacing between any two shapes on that level is greater than zero. The run time for design-rule analysis in the horizontal direction is linear since each bar can be enclosed by its immediate neighbor to the right. However, the same type of configuration can be used to generate the worst run time: Assume that all bars have the same electrical potential and are mergeable. Since a neighboring bar may be merged with the primary bar, the neighbor's neighbor must be examined against the primary bar as well. Further, since the merger takes place only after spacing requirements for all bars are determined and included in the graph representing the structural relationship of the bars, the design-rule analysis must be carried out for each bar against the rest of the bars to its right. Thus, for a row of m bars, $(m - i)$ bars must be examined for the i-th bar in the row. In all, $(m^2 - m)/2$ examinations are necessary and the run time is proportional to m^2. (Alternatively, the same conclusion may be reached by assuming that all bars are on different mask levels and there is no spacing requirement between any two elements on different mask levels. Since the mask levels of neighboring bars are not known a priori, and the design-rule analysis is carried out for all masks simultaneously, the compaction in the horizontal direction requires the design-rule analysis routine to examine all bars to the right of each given bar to ensure proper enclosure.)



(a)



(b)

Figure 4.13 Examples for analyzing the complexity of the method used for design–rule analysis.

A stack of the parallel bars (Figure 4.13(b)) is generally a reasonable model for most circuit layout configurations. In particular, assuming that there are **m** rows of parallel bars with **m** bars in each row, and all bars in the same row are on different mask levels or mergeable, the run time of the design-rule analysis routine for horizontal compaction is proportional to $(3\mathbf{m}^3 - 5\mathbf{m}^2 + 2\mathbf{m})/2$. (Briefly, for each bar all the neighbors to the right, in the same row as well as one row above and below it, must be examined for possible side and diagonal enclosure.) Letting the total number of groups (bars) in the layout model be **n**, where $\mathbf{n} = \mathbf{m}^2$, it is apparent that the design-rule analysis technique has an order of complexity of $\mathbf{n}^{1.5}$ in the worst case.

Statistics taken from PRSLI for real circuit layout examples show the number of groups examined by the design-rule analysis routine increases approximately as $\mathbf{n}^{1.2}$, where **n** is the total number of elements in the layout. The exponent may be made somewhat lower if the construction of the enclosing fence is implemented more elaborately. Details of the program implementation and the gathering of run-time statistics are presented in Chapters 5 and 6, respectively.

4.6. Handling Fixed Constraints

Thus far it has been assumed that all lines in a layout plan behave like rubber bands, whose lengths may be extended or shrunk to suit the design-rule requirements. However, in many cases it is useful to be able to fix the length of a line or the spacing between elements. For example, a maximum length may be imposed on a line for circuit performance reasons. Similarly, two elements may be set a fixed distance apart so that their relative topol-

ogy does not change.

The most general form of a fixed constraint between two groups at locations i and j (i >= j) may be written as

$$l <= i - j <= u,$$

where l and u stand for the lower and the upper bound, respectively. Thus, for example, a rigid fixed constraint can be specified by setting l equal to u. These constraints are included in the graph representation of the layout plan as branch weights that force the distance between two nodes to meet the constraining requirements.

Although the distance between two groups may be specified with a fixed constraint, the design-rule requirements between them still must be determined because the fixed constraint specified by the user may be in conflict with other (design-rule-induced) constraints in the layout. Thus, the lower bounds are used first with the longest path algorithm to determine the new location for each group. After all new locations are determined, the upper bounds are checked against the separations between the corresponding groups. A conflict is detected if the separation between two groups is greater than the corresponding upper bound. Such conflicts are reported to the user for correction.

4.7. Macrocells and the Hierarchical Build-up of a Layout

Layouts are often generated hierarchically to save time and storage requirements and to reduce organizational complexities. With a CABBAGE-like layout system, the hierarchical layout process involves the build-up of larger circuit layouts with previously compacted cells or user-defined black boxes. For this application, it is sufficient to know only the peripheral struc-

ture of a compacted cell, provided the user does not attempt to alter its internal structure. Here, the proper peripheral representation of a compacted cell is termed a macrocell.

There are four factors that must be considered in generating a macrocell. First, the peripheral structure must fully enclose the rest of the macrocell to keep out neighboring elements. This requirement also implies that the peripheral representation should reflect the edges of a macrocell accurately to let neighboring elements fit snugly around its border. The present data structure used in CABBAGE makes it necessary to represent the peripheral structure with element symbols, which can be treated by the program uniformly as any other element. Second, the macrocell must be fixed in size and shape throughout the compaction process. Here, fixed constraints can be used to set the required distances between peripheral elements. Third, the macrocell must preserve electrical connectivity. Since the mergeability of elements are determined based on electrical node numbers during the compaction process, proper node numbers are essential to the area-efficiency achievable by the compactor. Because macrocells are represented as collections of peripheral symbolic elements, it becomes necessary to retain internal contacts to preserve the proper electrical connectivity at the edge of the macrocell.

It should be noted that, with some changes to the data structure used in the present implementation of CABBAGE, macrocells may be represented more naturally as individual elements rather than collections of their constituent elements. In particular, the point structures used in CABBAGE at the present, such as transistors, are macrocells in reduced form; they are macrocells with only one connection point at the element center. The indivi-

dual element representation of a macrocell would reduce the storage and computation requirements substantially to realize the full advantage of the hierarchical layout approach.

The last factor of consideration concerns the use of macrocells in a general hierarchical layout environment. Here it is a common practice to build up larger building-blocks from macrocells directly. For such applications, area-efficiency and chip organization considerations make it desirable to match the connector locations of the connecting macrocells. The match may be achieved by using fixed constraints to set the relative locations of connectors based on estimates of the final connector locations of the connecting cell. However, the compaction process for both macrocells often must be carried out iteratively to finalize the connector location.

The use of stretchable macrocells for the purpose of direct hook-up has been proposed by Johansen [40]. Here, certain elements of a compacted cell (macrocell) are made to follow the movement of some specified connectors of that cell. With such a scheme, an interactive graphics editor may be used to split and to hook up compacted cells, forcing connectors in smaller cells to stretch and to match the connectors of larger cells. The area-efficiency lost in the individual stretched cells usually can be compensated by the overall area reduction resulting from the more regular interconnection pattern among macrocells.

The use of stretchable macrocells amplifies the usefulness of the multi-algorithm approach to computer-aided layout generation, as depicted previously in Figure 1.1. The layout compactor is very efficient at packing each macrocell into a small area and correcting all violations of design-rules. The stretching of macrocells to uniform dimensions can be performed readily

with an interactive graphics editor as a final touch-up.

4.8. Preferential Compaction

The problems of the direct hook-up of macrocells are not limited to those of matching connector locations. The shape of each macrocell must be optimized also to achieve the overall regularity and performance requirements at the chip level. Since rectangles have been the most popular shape for macrocells and building-blocks, the shape optimization is often reduced to that of optimizing the aspect ratio of rectangles. In particular, the user of a layout compaction system should have control over the length of at least one dimension of the rectangular bounding box of the compacted cell. For example, the user may want to specify an upper bound for that dimension so as to be able to fit the resulting cell into a limited space left on the chip.

The separate horizontal and vertical compaction operations used in PRSLI (as well as in all other published work on the topic of layout compaction [16, 17, 18]) do not provide mechanisms for observing this type of aspect ratio requirement. In PRSLI, however, it is possible to make compaction in one direction more preferable than compaction in the other direction. Specifically, since enclosing neighbors within one maximum spacing of the two end points of a primary group are examined during the design-rule analysis phase for possible corner-to-corner spacing requirements, a particular direction may be made less preferable for compaction by extending the safety zones at the two ends beyond one maximum spacing. In effect, the extended safety zones prevent the premature movement by neighboring groups into the (unused) areas protected by these extended zones. Thus, the neighboring groups in the perpendicular direction can be given the priority

for claiming the usage of these preserved areas to achieve a possibly higher layout density in that direction.

CHAPTER 5

DESCRIPTION OF PROGRAM STRUCTURE

5.1. Introduction

The purpose of this chapter is to describe how the major algorithms introduced in Chapter 4 are linked up in the CABBAGE system. All the algorithms presented in Chapter 4 are complete in principle and operational by themselves. However, necessary links must be placed at appropriate locations and sometimes compromises must be made for the algorithms to work together efficiently in a computer program.

The organization of the GRLIC (interactive graphics editor) program is quite straightforward. The GRLIC program is written primarily to fulfill the needs for putting data into the PRSLI (layout compaction and generation) program. For completeness, however, a brief description of the GRLIC program is included in Section 5.2 below. In contrast, the PRSLI program is more involved and is described in detail in Section 5.3. The description of the PRSLI program is partitioned based on the program overlays.

5.2. The GRLIC Program

The GRLIC program is an interactive graphics editor for generating and modifying symbolic layout plans. It is driven by the commands issued by the user. (A list of commands accepted by the GRLIC program is included in the User's Guide to the CABBAGE System in Appendix 5.) If the command involves graphics operations, such as drawing or erasing a symbol, the PRGRF (Process Graphics Commands) routine is used to receive the intended location of

the symbol and to drive the pertinent routines containing the knowledge about the shape and structure of the symbol. Commands involving no graphics operations are processed by the PRCMD (Process Command) routine, which collects the necessary parameters and drives the appropriate routines for performing the command. A listing of the routines used in the GRLIC program is in Appendix 8.

GRLIC treats a symbolic layout as a collection of symbols and keeps no information regarding their interconnection and adjacency relationship. Each symbol is kept in a Symbol Description Block (SDB) which is a copy of the SDB for that symbol in the disk file. The SDB's in the program memory are stored in a linear array (the LSG array) and are sorted by their center locations for the ease of later references. The data structure used in GRLIC is described in Appendix 2. (It is to be note that symbols for fixed constraints and electrical elements, such as lines and transistors, are all stored in the LSG array as SDB's in a uniform fashion.)

5.3. The PRSLI Program

The control flow of the PRSLI program can be divided into three phases. The initial setup phase involves the determination of the interconnection relationship of the elements. With the knowledge of the interconnection relationship, the program enters the compaction phase to perform the design-rule analysis and the group placement. Finally, the result of the compaction is displayed in the output phase. Routines for the first and the last phases are grouped into two separate program overlays. In order to match the length of these two overlays, the routines used in the compaction phase are divided according to their functions and are put into three overlays for

design-rule analysis, element placement, and jog generation.

5.3.1. The Setup Overlay

The setup phase begins with the reading of Symbol Description Blocks from the file containing the symbolic layout. The SDB's for electrical elements are stored in a linear array (the LSG array in the data structure described in Appendix 3). The SDB's for fixed constraints are read in and stored temporarily in a separate array.

The connectivity of the elements in the LSG array is examined next. The interconnection relationship of the elements is represented in PRSLI in the form of electrical node numbers, which are appended to the SDB's of the individual elements as described in Appendix 3.

The simplifications of drawings allowed by GRLIC that affects the connectivity of the elements must be removed at this stage for the determination of the proper electrical connectivity. For example, for the ease of drawing, GRLIC allows the user to draw a continuous diffusion line through the channel area of a transistor symbol. However, the two ends of the channel of a transistor usually are connected to different electrical potentials. As a result, such a continuous diffusion line must be located and split into two segments (as two new SDB's) at the center of the intersecting transistor. Similarly, GRLIC allows the user to terminate any type of line with a terminal element and hence the line type corresponding to the terminal element must be determined also at this stage. (In fact, the same mechanisms for determining the terminal type may be used to determine the type of the contact necessary for connecting two intersecting lines or to detect illegal crossovers of lines. Due to an oversight, however, such capabilities are not imple-

mented in the CABBAGE system at the present time.)

As in the case of the GRLIC program, the SDB's are sorted by their center locations and grouped together based on their topological connectivity. (Two symbols are topologically connected if they touch each other. Further, they are put in the same group if they are topologically connected and have the same center location in the x or the y direction.) The primary purpose for grouping topologically connected symbols is to form clusters of symbols that stay together during the compaction operation. But the grouping is also useful as an approximation to the actual electrical connectivity. After the preparatory work has been performed, the node-numbering routine can proceed to assign electrical node numbers to each element. The node number of each element is appended to its SDB and backpointers to SDB's having the same node number are grouped and stored in the NLST array.

The topologically connected groups are represented by the LREF blocks in the data structure. The LREF blocks have backpointers (through the LIST array) to their constituent symbol description blocks and are sorted by the center locations of the groups they represent. Further, since the topologically connected groups are mapped into nodes in the longest path algorithm, the precedence relationship of the nodes derived from the design-rule analysis are recorded in the LREF blocks.

5.3.2. The Design-Rule Analysis Overlay

The first and the most time-consuming task to be performed in the compaction phase is the design-rule analysis. The result generated by the design-rule analysis is a graph describing the precedence relationship of groups.

The design-rule analysis overlay examines the sorted LREF blocks sequentially, beginning with that representing the left-most or the lowest group for the compaction in the horizontal or the vertical direction, respectively. Each primary group is examined against the neighboring groups to the right or above it to determine the spacing requirements between them. In order to perform an accurate spacing analysis, the edge structure of the primary group viewed from the right or the top must be developed and checked against the left or the bottom view of the edge structure of a neighboring group. The array for the edge structures contains detailed descriptions for each edge of the elements in a group. The descriptions are used mainly by the subroutine SPACE to determine the spacing requirement and the mergeability of two opposing edges and include the contributing element, the edge type (described in Section 4.5.2), the electrical node number, the distance from the center of the group, and the mask level of an edge. At the present, the edge structures are developed whenever needed, with the subroutine TRACE, to conserve memory at the expense of increased run time.

The fence within which the design-rule analysis must be carried out for a primary group is built during the design-rule analysis and is recorded in the edge structure array. In particular, the edge type is changed to "covered" (defined in Section 4.5.2) once an edge of the primary group is covered fully by a neighboring group. (The change of an edge type is shown graphically for the simple example in Figure 5.1. The primary group on the left is gradually covered by its neighbors to the right when the design-rule analysis is performed in the horizontal direction.) Since design-rule analysis is not needed for a covered edge, the design-rule analysis for a primary group is terminated when all edges in that group become covered. This scheme for building up a fence is somewhat crude but the fence is guaranteed to enclose

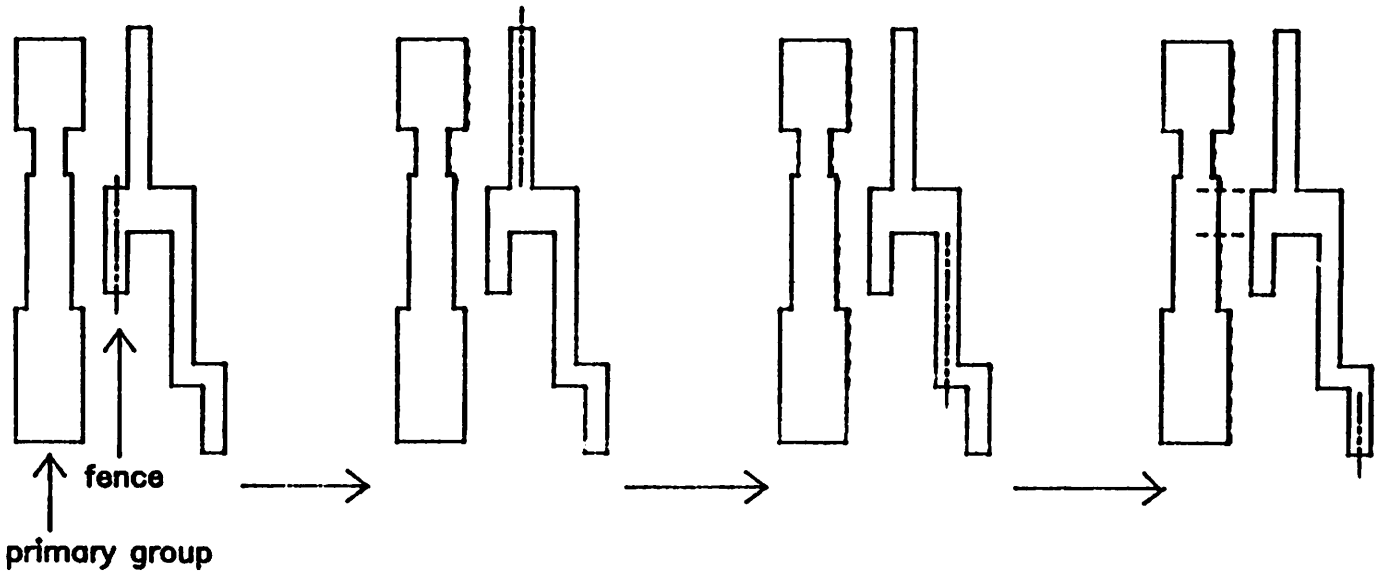


Figure 5.1 The progress of the covering process for the primary group.
The covered edges of the primary group are emphasized with dashes.

all neighboring groups visible by the primary group. It is crude because the fence may be too conservative: for lack of a more elaborate recording scheme, each segment is covered starting with its two ends and any fence covering only the middle portion of a segment therefore must be discarded, as in the case of the left-most fence in Figure 5.1. The exclusion of the left-most fence leaves a small portion of the primary group uncovered even after the fences from the other three neighbors have been considered. Thus, with the present recording scheme, the difficulty of finding a suitable fence rises with the length of the corresponding primary segment.

The precedence relationship and the spacing requirements among groups developed by the design-rule analysis routine are used to construct a graph representing the horizontal or the vertical structure of the layout plan. The nodes of the graph are represented by the individual LREF blocks, and the branches emitting from a node is stored in the LF (List of Followers) array accessible by the corresponding LREF block.

5.3.3. The Group Placement Overlay

The longest path through the graph built by the design-rule analysis overlay is determined with the algorithms described in Sections 4.4.2 and 4.6 to guide the placement of groups. For groups not on the longest path, the unused space around them are distributed based on their predecessor-to-descendent ratios. Here, such a ratio approximates the ratio of the attraction forces on either side of the less-constrained group. In most cases, such a force-directed distribution of space has the effect of keeping connected groups from slipping away from each other, as shown in Figure 5.2, as a result of the bias applied by the longest path algorithm. However, since the

predecessor and descendant counts of a group also include groups unconnected to it, the predecessor-to-descendent ratio is only a simple approximation and may not give the best placement.

The spacing requirements resulting from the longest path calculation is compared with the compaction solution obtained in the previous iteration. The compaction operation in the present direction may be terminated (by the user) if the solution obtained in the previous iteration meets all design-rule requirements and occupies the same amount of area as the present solution. Otherwise the generation of the new solution must be continued: the line elements not in the direction of compaction must be modified (shrunk or expanded) to reflect the changes in geometry produced by the compaction. In addition, groups that merged together as a result of the compaction must be joint and represented by a single LREF block.

5.3.4. The Jog Generation Overlay

The jog generation overlay determines all intervals on both sides of a group where the spacing between the group and its neighbors is equal to the minimum required spacing. These intervals are collected and coalesced in a temporary array (the MCST array). The current group is split at locations not covered by these intervals, provided the intervals above and below the selected locations are contributed by different sides of the group. The groups resulting from this split are then linked up with lines that may be stretched during the subsequent compaction operation to form bent lines or jogs. Most of the routines used in the design-rule analysis overlay are used here for the purpose of determining the minimum spacing intervals. Several routines used in the setup overlay for adding new elements and forming new

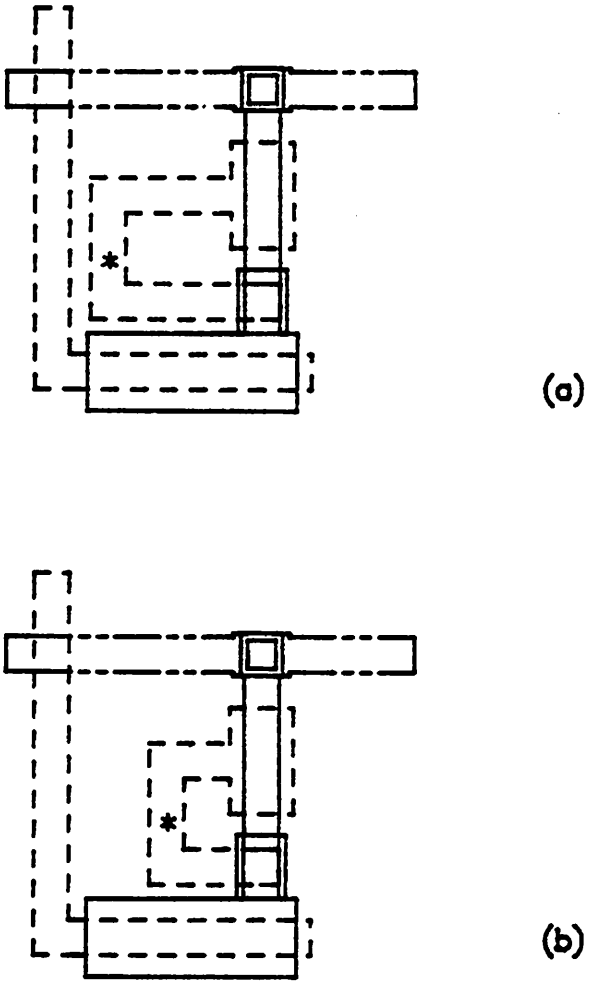


Figure 5.2(a) The line marked with * slips to the left as a result of the longest path calculation.

(b) Attractive forces applied by the structure on the right reposition the line.

groups are used also for splitting groups and adding linking lines.

5.3.5. The Display Overlay

The result of the compaction is displayed in rectangular polygon form. Here the routines for developing edge views are used again to supply the edge segments of the four sides of a group. Thus the result displayed is exactly what the design-rule analysis routine will see in the subsequent compaction operation.

CHAPTER 6

THE PERFORMANCE OF THE CABBAGE SYSTEM

6.1. Introduction

The performance of the CABBAGE system may be evaluated from two perspectives. First, the efficiency of the layout obtained from the CABBAGE system must be compared with those obtained by other means, such as the hand layout approach. Here the efficiency of a layout is evaluated mainly on the basis of the overall size. Other performance and quality factors of a layout, such as the power-delay product, the long-term reliability of a particular configuration, etc., either require more involved measurements or are less well-defined. As such, these latter factors are not considered in this chapter formally. Second, the computer resources required by the CABBAGE system must be examined in terms of the run time and the memory usage.

For the purpose of evaluating the efficiency of the solution, hand-drawn layouts of functional blocks used in microprocessor-type LSI circuits are compared with their equivalent layouts obtained from the CABBAGE system. Such comparisons show that, as the user of the CABBAGE system keeps improving the layout topology, the final solution obtained from the CABBAGE system is not only more regular but also smaller than that done by hand. The details of the iterations this author went through with the CABBAGE system to layout a functional block used in a microprocessor circuit are described in a separate section below. (The layout for that block is approximately 2% smaller than the equivalent layout done by hand.)

For the evaluation of computer resource requirements, statistics on run time and memory usages are collected and compared with the predicted trends. Detailed results included in separate sections show that, with the present implementation, the compaction time increases approximately as the 1.4 power and the memory usage increases linearly with the number of elements in the layout.

6.2. An Example: The Latch-Driver Block

Figure 6.1(a) shows the circuit schematic diagram of a latch-driver block used in a microprocessor. The corresponding hand-drawn layout used in that microprocessor is shown in Figure 6.1(b). This layout is representative of the dense and area-efficient layouts commonly used in the industry and hence is a realistic example for examining the efficiency achievable with the CABBAGE system.

A close look at the layout in Figure 6.1(b), however, reveals that it is unstructured and its topology is less than optimum: The gate location and interconnection diagram (Figure 6.1(c)) of this hand-drawn layout shows the gates are placed randomly and the signal lines often wander deep into the interior of the layout and occupy the intervening space unnecessarily. Thus, a new and regular layout topology is developed to generate an equivalent layout with the CABBAGE system.

6.2.1. Strategies for Developing a Better Layout Topology

Because the topology of MOS transistors consists of simply the intersection of a diffusion and a polysilicon line, signal lines are generally more space-consuming than the devices they connect to in a layout of an MOS cir-

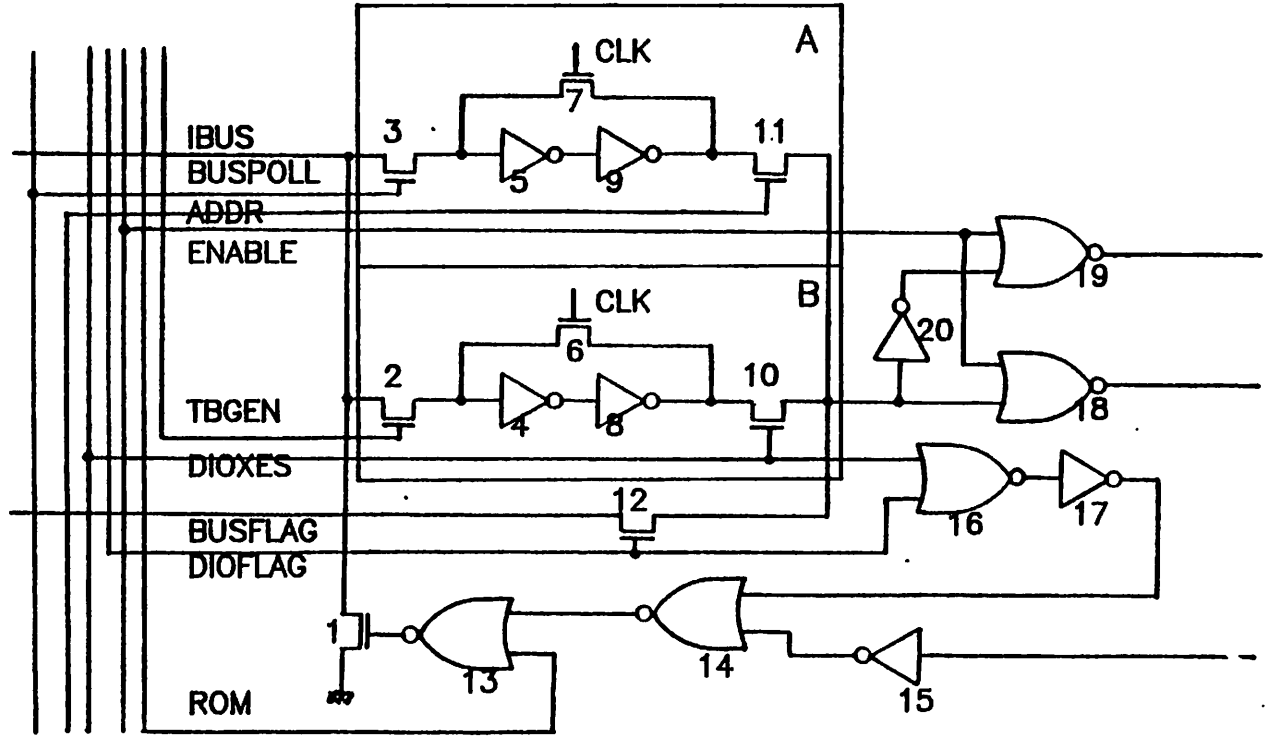


Figure 6.1(a) The circuit schematic diagram of the latch–driver block.

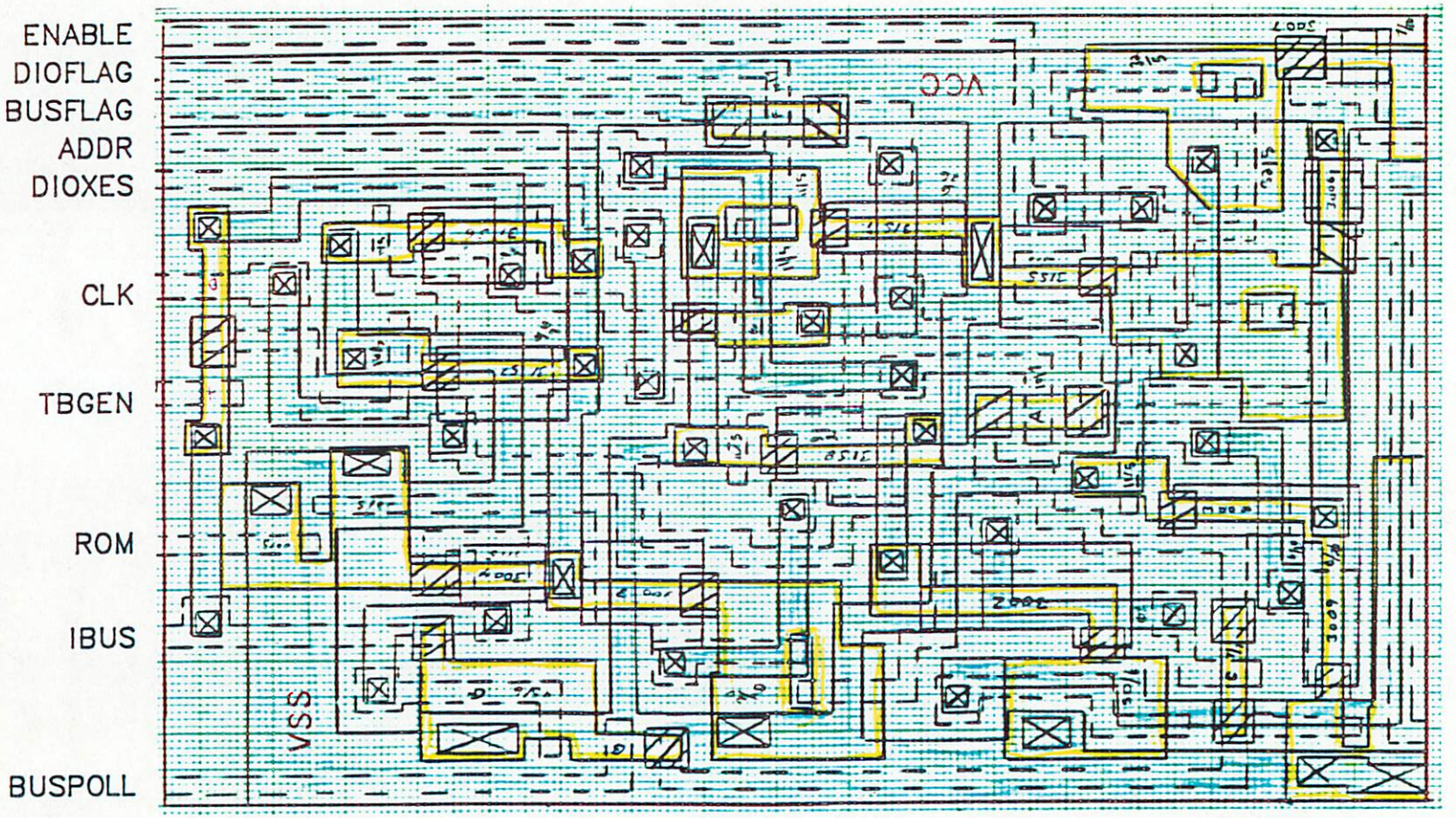


Figure 6.1(b) The hand-drawn layout of the latch-driver block.

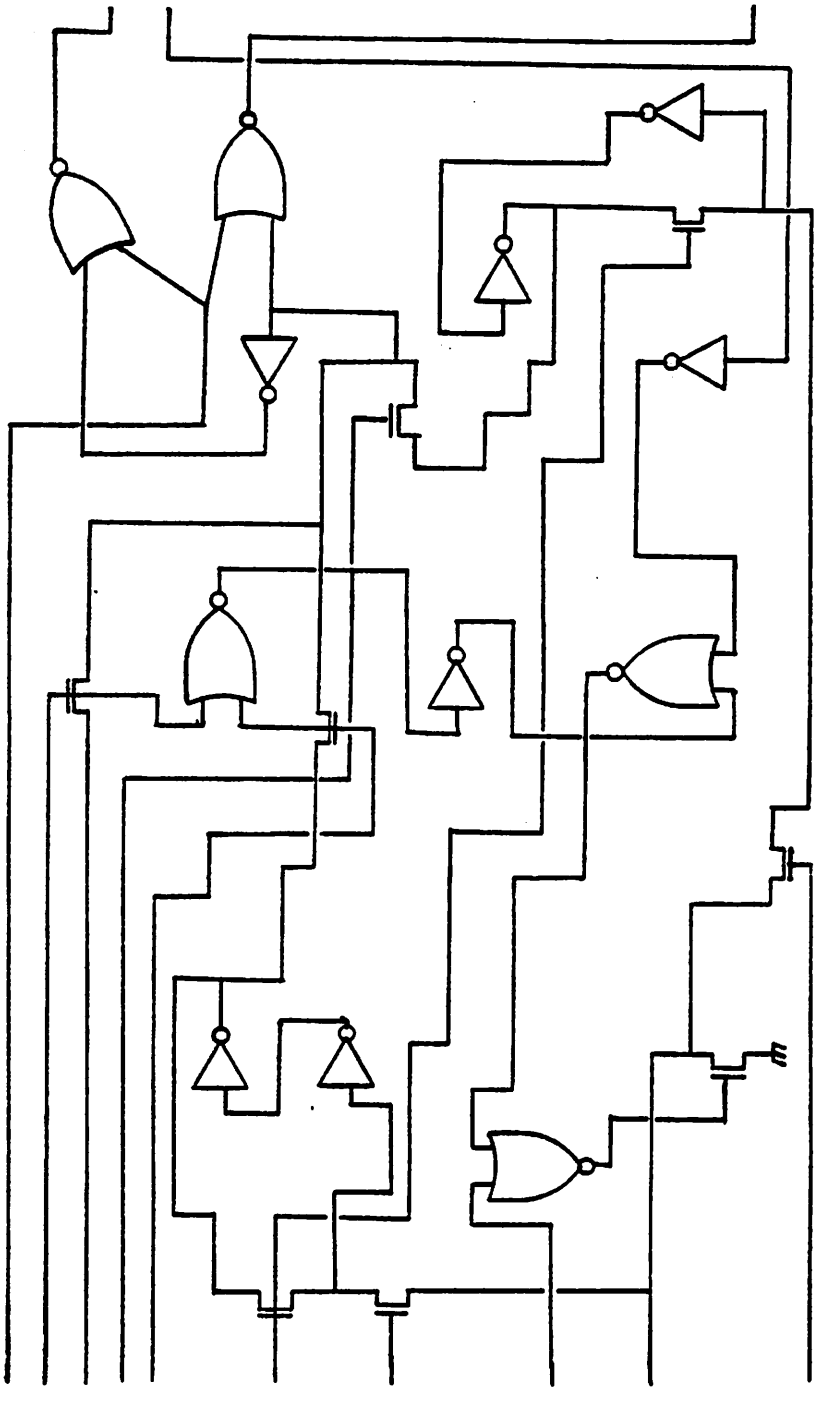


Figure 6.1(c) The gate placement and interconnection diagram of the hand--drawn layout.

cuit. Thus, the first priority in developing a better layout topology for the circuit in Figure 6.1(a) would be to keep signal lines as short as possible. As a direct result of this decision on routing, the placement of gates becomes more or less determined: The two data latches (Latches A and B in Figure 6.1(a)) have the most connections to the outside and must be placed near one edge of the layout. In particular, the IBUS and the CLK lines are the two signal lines with the most connection to the latches and should be kept short to save space and lower their loading capacitance. Similarly, the driver section (Gates 18, 19 and 20 in Figure 6.1(a)) must be located at the opposite edge of the layout as most of its inputs are connected to internal nodes and its outputs are directed outward. The pull-down logic (Gates 1, 13, 14, 15, 16 and 17 in Figure 6.1(a)) is thus placed in the middle as it has connection lines from both edges and some interior nodes. The placement and interconnection diagram of this new layout topology is shown in Figure 6.2(a).

6.2.2. The Construction of the Layout

The actual topological layout plan is developed and modified in three stages to minimize the complexity of the layout problem. During each stage of the construction, new elements are added on top of the existing layout and some elements in the existing layout are adjusted in the hope to achieve a snug fit. In addition, the area used in later layout stages are estimated and reserved.

In the first stage the two latches and Gate 1 of the pull-down logic are laid out, as shown in Figure 6.2(b). This part of the circuit is chosen as the starting point because it has possibly the most complex topology and the most connections to the outside.

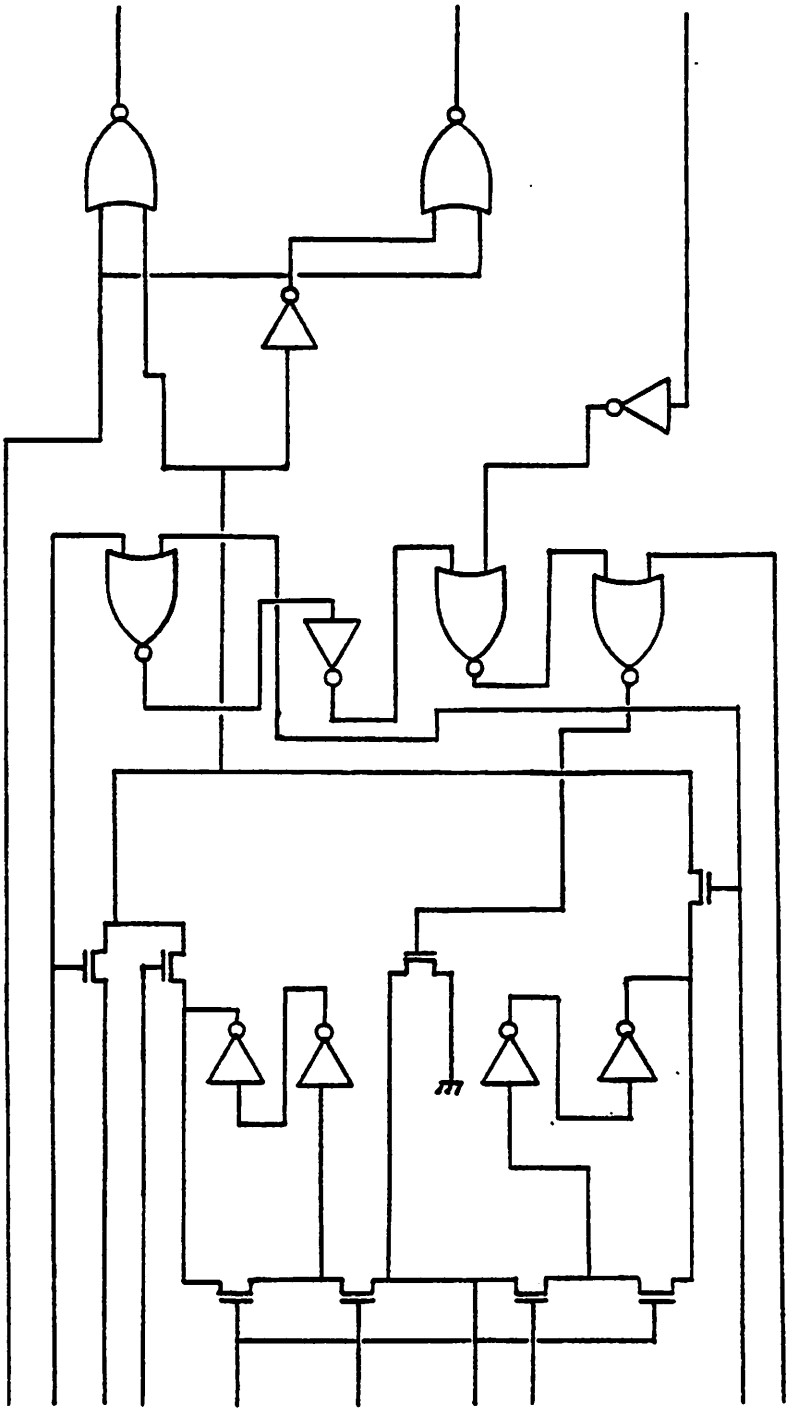


Figure 6.2(a) The new gate placement and interconnection diagram.

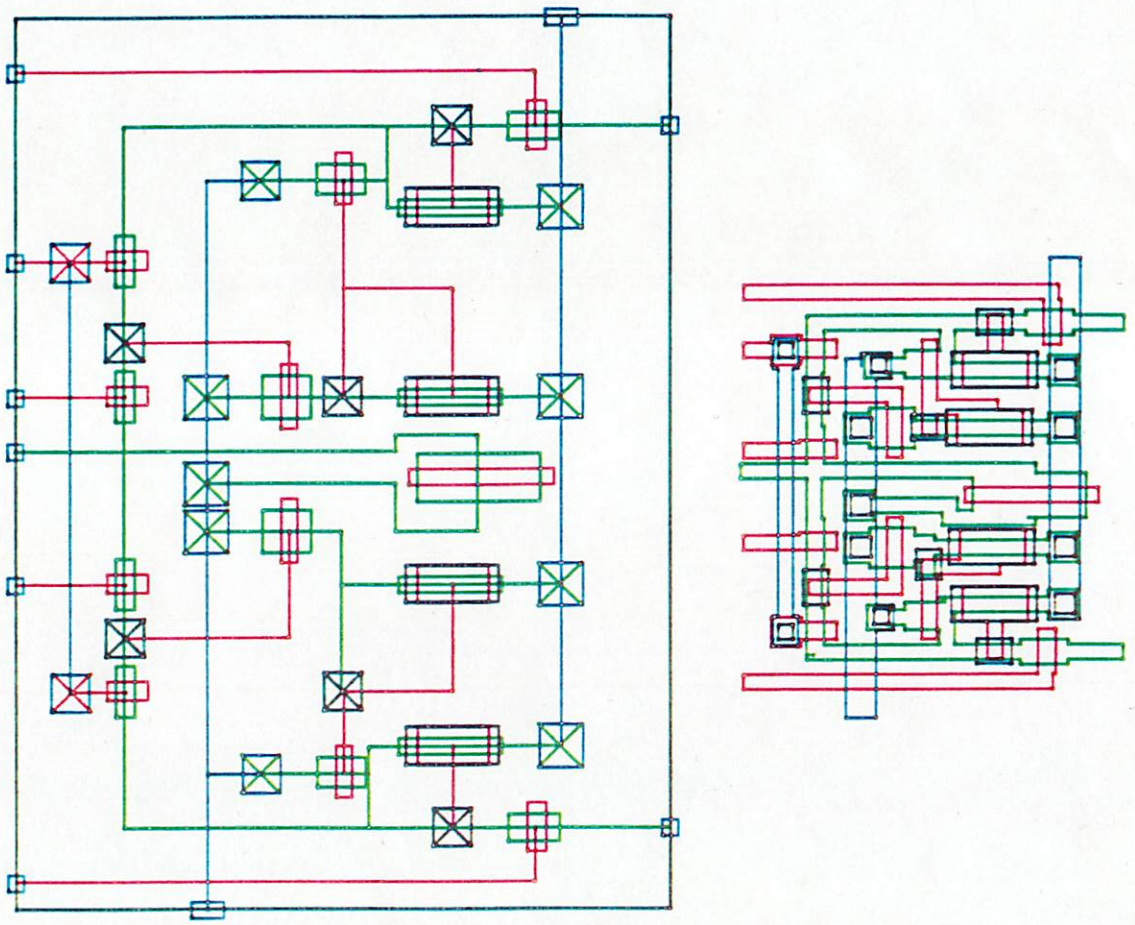


Figure 6.2(b) The symbolic layout plan and the compacted layout of the two latches in the latch–driver block.

Because it is desired to generate a layout whose aspect ratio is comparable to the hand-drawn layout in Figure 6.1(b), every effort has been made to keep this first-stage layout short (in the vertical direction) to accommodate future additions of signal lines to other parts of the layout. Typically, such efforts involve the alteration of the directions of transistors and the modification of interconnection configurations (introduction of jogs) in the most constraining structures in the interim layout obtained from the PRSLI program. (Compare, for example, the layout of the first stage in the final layout in Figure 6.2(d) with the original layout of the first stage in Figure 6.2(b).)

The development of the gate placement and interconnection plan and the actual topological layout plan for the first stage of the latch-driver example took the author an afternoon to complete. Once the first stage is completed, the rest of the layout is appended relatively easily. A major portion of the pull-down logic is added that evening in stage two of the layout. (The interim result is shown in Figure 6.2(c).) The final layout in Figure 6.2(d) is completed the following morning with the addition of the driver section. The total layout time is on the order of eight hours, with nearly 90% of the time spent on developing and improving the layout topology. (The compaction of the final layout in Figure 6.2(d) takes less than three minutes of computer time on the H-P 1000 Series E minicomputer.)

6.2.3. A Comparison of Area-Efficiency

In order to make the comparison of the area-efficiency fair and realistic, the layout done with the CABBAGE system follows most of the rules used in the hand-drawn layout. In particular, the two layouts use the same family of

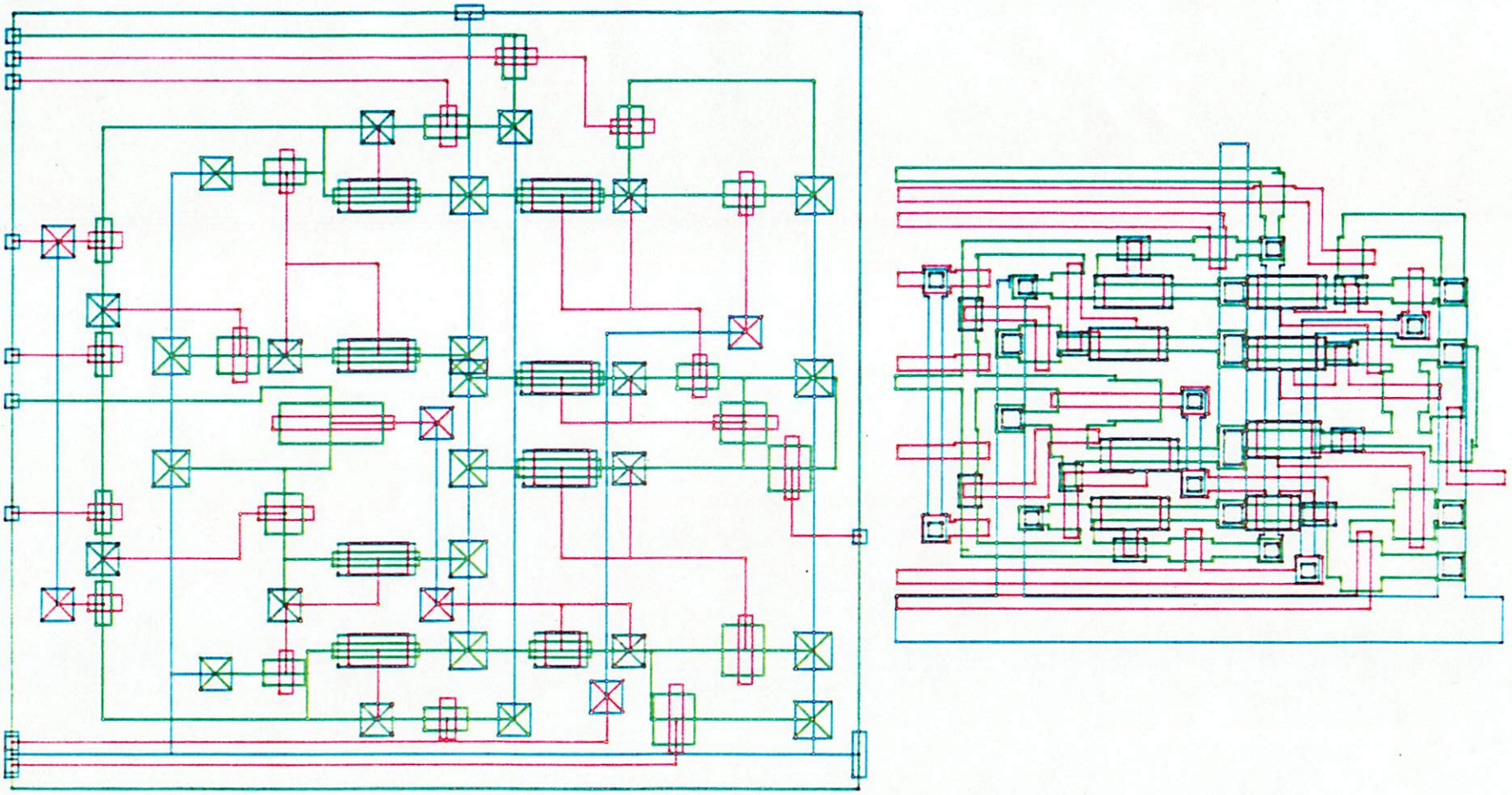


Figure 6.2(c) The symbolic layout plan and the compacted layout of the latches and the pull-down logic in the latch-driver block.

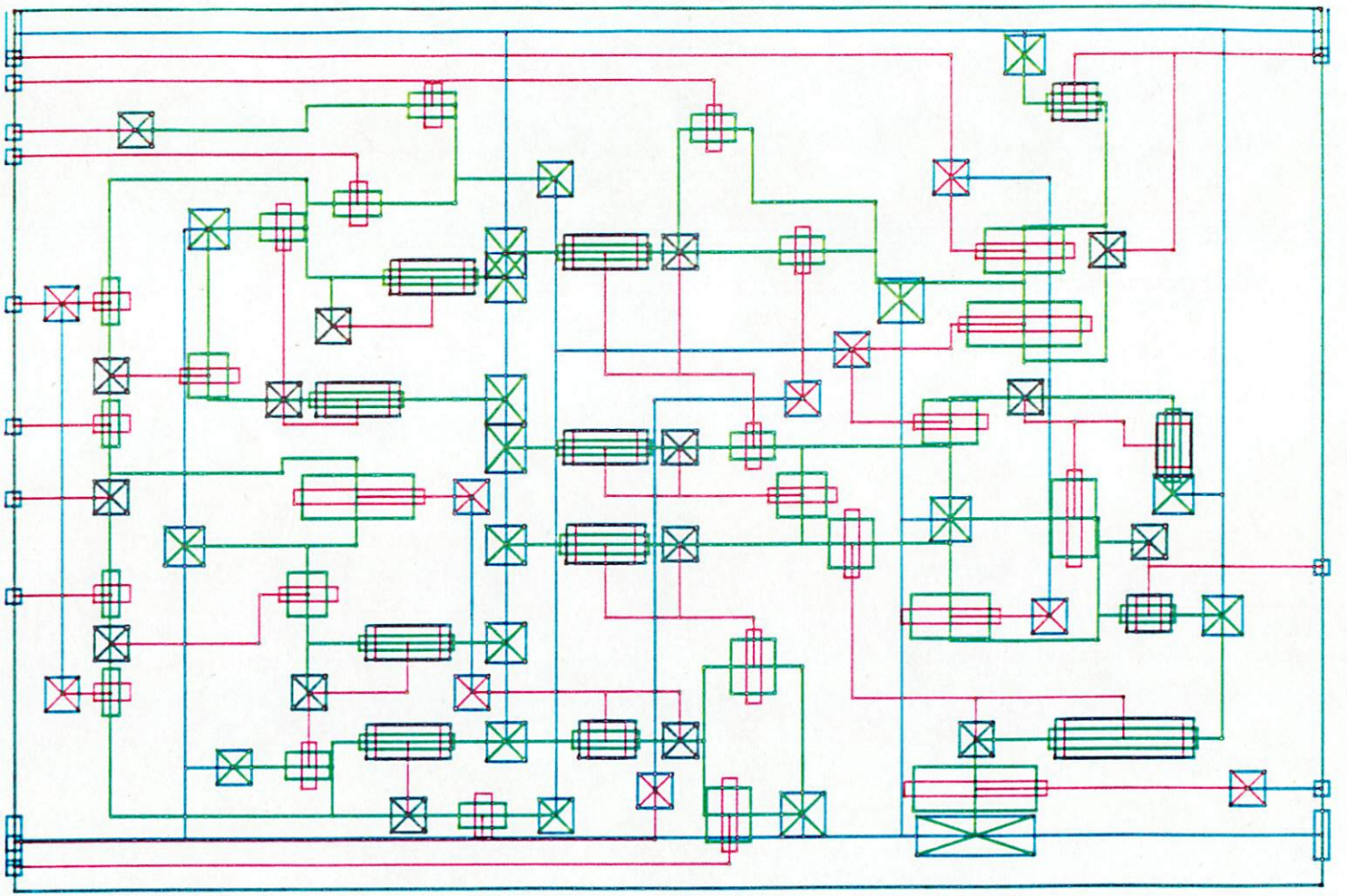


Figure 6.2(d) The symbolic layout plan of the complete latch–driver block.

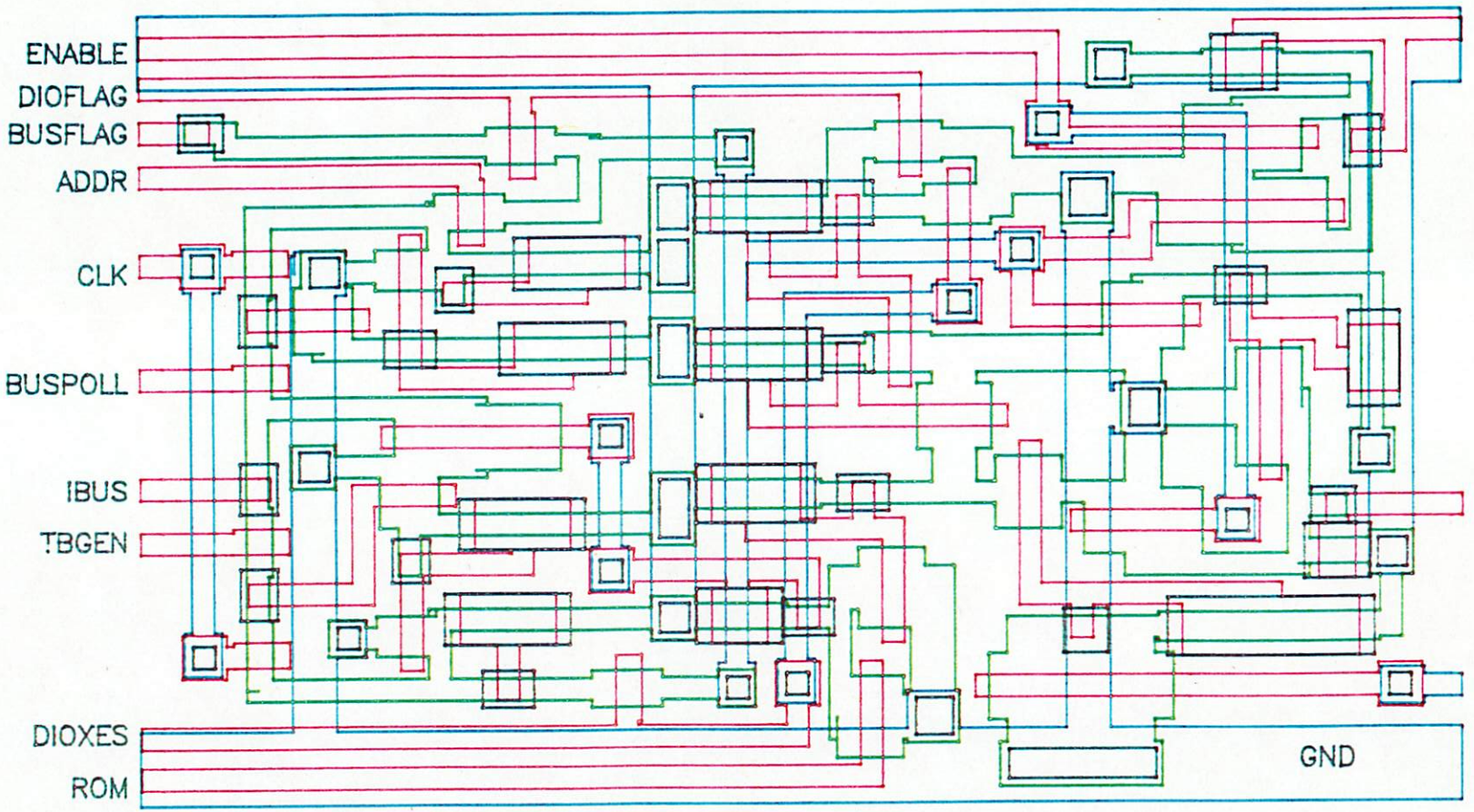


Figure 6.2(d) (continued) The final layout of the latch–driver block generated with CABBAGE.

geometric layout design rules and both have 20-micron wide power lines and minimum-width polysilicon signal lines. The major differences of the elements used in these two layouts are:

a) 6-micron-wide polysilicon lines are used in the layout generated with CABBAGE in place of the minimum-width 5-micron-wide polysilicon lines used in the hand-drawn layout. CABBAGE uses 6 microns as the minimum line width in the interest of keeping the minimum line widths uniform among all line types.

b) The minimum channel length of driver transistors is 6 microns in the layout generated with CABBAGE instead of 5 microns used in the hand-drawn layout. It is assumed that, in converting a 6-micron channel to a 5-micron channel, the area saving gained with the shorter channel length would be sufficient to compensate the 1-micron extra gate extension in the perpendicular direction for minimizing the edge-rounding effect on short-channel transistors. (The edge-rounding effect is described in Appendix 4.)

c) Only transistors with straight channels are used in the layout generated with the CABBAGE system. At the present, CABBAGE does not have element models for transistors with interdigitated or S-, L-, or U-shaped channels.

d) The order of signal lines at both edges is different for the two layouts. The order is assumed interchange-

able since all signal lines are hooked to external bus lines running in the vertical direction.

Even though both differences a) and c) above tend to degrade the compactness of the layout generated with the CABBAGE system, the resulting layout is a little over 2% smaller than the hand-drawn layout in Figure 6.1(b). (The size of the hand-drawn layout is 220 by 350 microns; The size of the layout generated with the CABBAGE system is 215 by 350 microns, with a peripheral margin comparable to that in the hand-drawn layout.) The quality of the layout generated with the CABBAGE system is comparable, if not superior, to that of the hand-drawn layout; the channel area of a transistor is defined better with a straight gate than with any other gate shapes. In addition, the sizes of contacts in the layout generated with CABBAGE are similar to those used in the hand-drawn layout. (Some of the contacts in the layout generated with CABBAGE may be made even larger if there were a simpler way to express off-center contacts in the topological layout plan.)

The area-efficiency realized with the CABBAGE system may be attributed to two factors: First, as a result of the regularity of the layout topology devised by the author, elements with similar sizes and shapes are located close to each other and occupy contiguous, rectangular regions. Area savings may be achieved more easily with butting rectangular regions than with more complicated polygonal regions. Second, the almost immediate feedback from the CABBAGE system makes it possible for the user to compare topological changes he has made. Moreover, the changes can be made with little effort on the part of the user. Thus the user and the compactor work together to improve the congested areas of a layout. It is felt that, in most cases, such a cooperation makes use of the capabilities of both the human

user and the computer to a fuller extend than what is possible with automatic operations alone.

6.2.4. Comments on the Layout Time

Experienced layout designers typically spend several days to construct by hand layouts of the complexity of the latch-driver block described in this section. The designer typically builds up the layout based on just one topological plan because modifications of a finished portion of the layout can be quite time-consuming. In contrast, most of the time spent by a designer working with the CABBAGE system is for the improvement of layout topologies. In fact, reasonably good layout topologies typically can be found after three to five major changes to the original topologies in approximately one hour of intensive work with the CABBAGE system. Layouts 10% to 30% larger than the hand-drawn layout can be generated with these modified or improved topologies. The extra areas can be compacted out with further minor modifications to the layout topology. These final touch-ups typically involve moving an element around a neighboring element to trade the space in the vertical direction with the space in the horizontal direction, and vice versa. Clearly, the development of an elaborate preferential compaction scheme that performs the trading of space for the user would improve the layout time with the CABBAGE system appreciably.

Note that the layout generated by hand must be examined for design-rule violations. The correction of such violations in a dense layout may take as much time as that used in producing the original layout. Thus, additional savings in layout time is likely as layouts generated with the CABBAGE system are guaranteed to be free of design-rule violations.

6.3. The Use of Macrocells

The layout of the two latches (Figure 6.2(b)) generated in the previous section is a useful entity for many applications. In this section, the latch-driver block is laid out again in which the layout of the two latches is used as a ready-made macrocell. Although the push-pull output driver consisting of the two larger NOR gates and their signal inverter (Gates 18, 19 and 20) is also useful for many applications, it is not made into a macrocell in this example for considerations of the density of the layout. However, the two large NOR gates are included in a separate stage when the layout of the rest of the circuit is finalized and made into a new, larger macrocell.

In the conversion of the layout into a macrocell, the symbolic structures near the periphery of the layout are preserved and their locations are fixed with respect to one another by user-defined constraints. Figure 6.3(a) shows the structure of the macrocell for the layout of the two latches. Since no element is to be added to the left of the two latches in this particular example, the left edge of the macrocell may be represented simply by the border of the overall layout.

The symbolic layout plan for the pull-down logic is done separately and is added to the symbolic representation of the latch macrocell as shown in Figure 6.3(b). The separate construction of the symbolic layout plan prevents the creation of a crowded layout plan whose overall size is influenced heavily by the size of the macrocells used in it. It is difficult to obtain good compaction results with crowded layout plans since the elements in such plans have less freedom of movement. Conversely, large and sparse layout plans can be hooked to small macrocells easily with bent interconnection lines. Note that the gates in the pull-down section are arranged in a

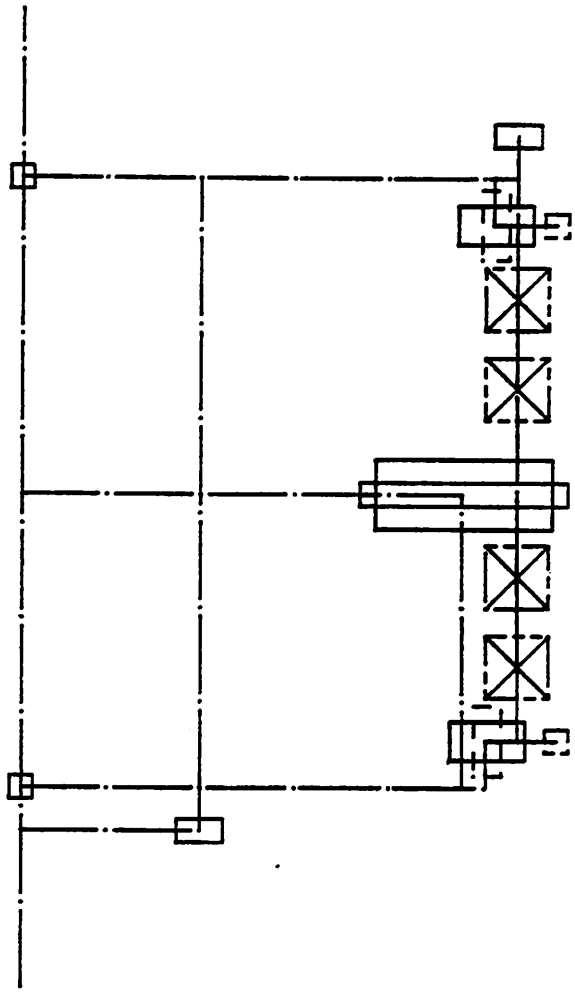


Figure 6.3(a) The macrocell for the latch circuit in Fig. 6.2(b). Fixed constraints are shown in dash–dotted line.

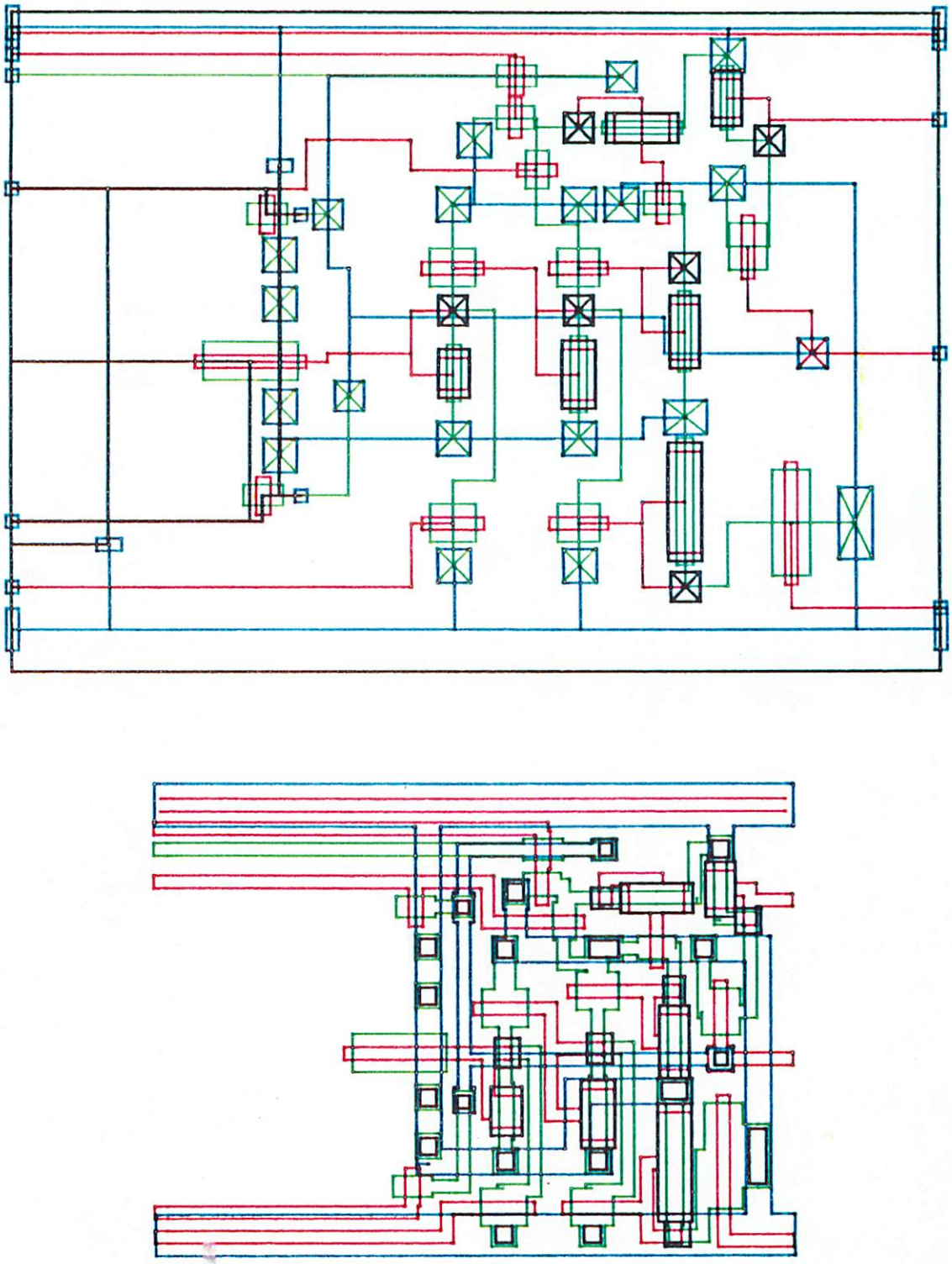


Figure 6.3(b) The layout of the pull–down logic and the latch macrocell. Fixed constraints are shown in black.

configuration different from that used in the layout generated in the previous section. The compacted layout consisting of the latches and the pull-down logic is made into a second macrocell. The NOR gates at the output are added to this second macrocell for the generation of the final layout, as shown in Figure 6.3(c). The compacted final layout is 223 by 350 microns in size or 4% larger than the equivalent layout generated with CABBAGE in the previous section.

The total layout time for the pull-down logic and the output driver is of the order of three hours. The maximum compaction run time is reduced to approximately one quarter of the maximum compaction run time required previously for the compaction of the entire latch-driver block and the memory usage is reduced by one half. The significant reduction in computer resource requirements and the acceptable area penalty indicate the usefulness of the hierarchical construction approach for the symbolic layout method.

6.4. Run Time and Memory Usage

The accumulative counts obtained from the program counter of the computer provide a first order breakdown of the time spent by each routine in the program. Such counts are taken and indicate that nearly 50% of the time is spent for design-rule analysis. The design-rule analysis consists of the two major tasks of (a) developing the combined edges for each group and (b) determining the spacing between groups. The complexity of these two tasks is observed experimentally with the compaction of the T-flip flop shown in Figure 2.5. The T-flip flop is duplicated from two to five times in the horizontal direction. The compaction time, the number of groups for which the

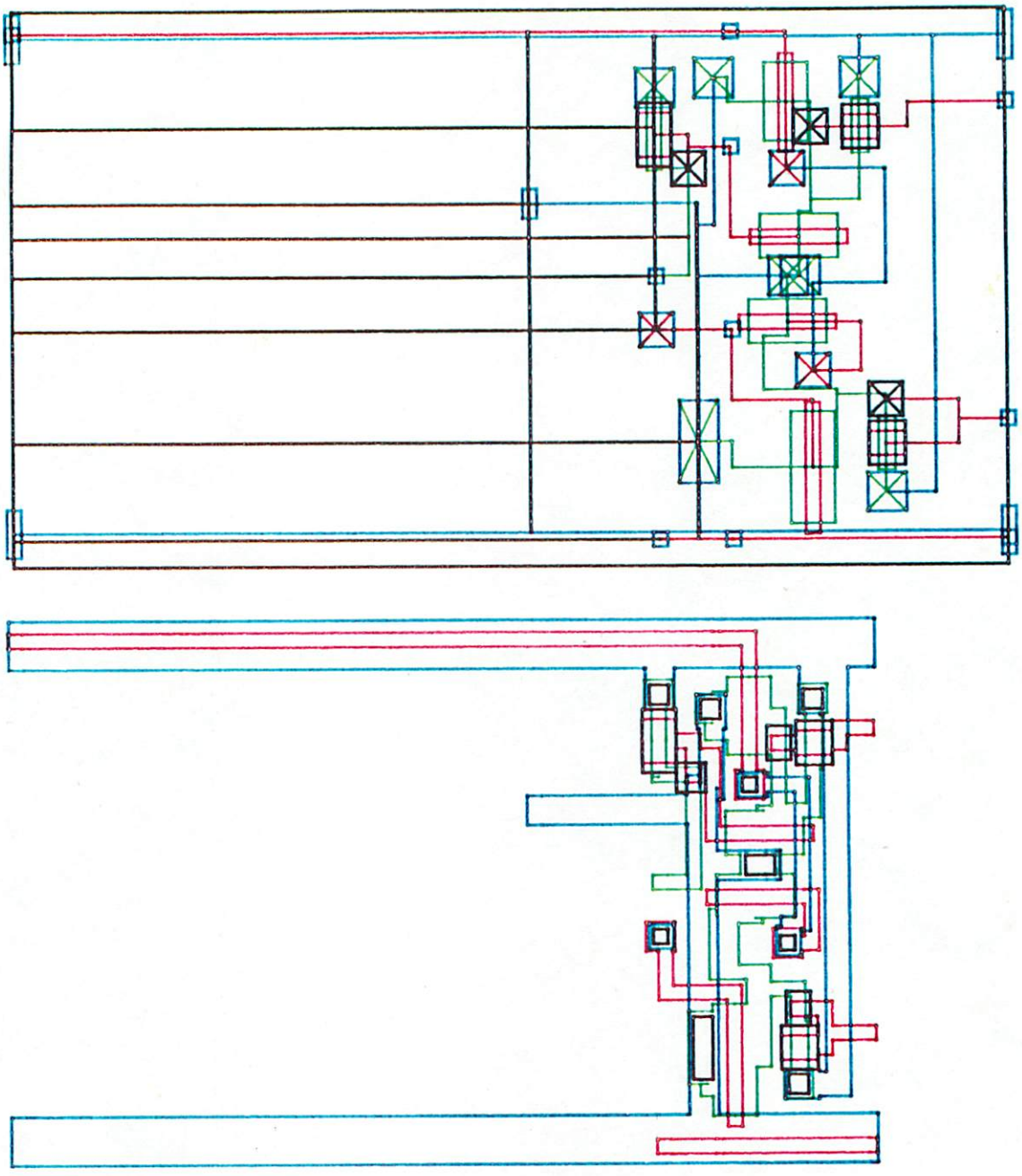


Figure 6.3(c) The layout of the output driver and and the macrocell generated in the previous step.

combined edges are developed and the number of groups spaced for the five similar but increasingly complex circuits are summarized in Figure 6.4. The top curve in that graph is proportional to the 1.5 power of the number of elements in the circuit and represents the upper bound for the compaction time. The actual time used for (a complete vertical and horizontal) compaction is shown by the curve below it and is proportional to the 1.4 power of the number of elements in the circuit. The number of groups for which the combined edges are developed is shown by the triangles and increases as the 1.2 power of the number of elements in the layout. Finally, the number of groups spaced is shown by the lowest curve and increases almost linearly with the number of elements.

Note that the compaction time cited in the figure are those for a vertical and a horizontal compaction operation. The number of iterations required for making a layout free of design-rule violations depends heavily on the topology of the circuit. For example, the T-flip flops require four iterations (two horizontal and two vertical) to meet all design-rule requirements, while the latch-driver block in Figure 6.2(a) requires six iterations.

The memory usage increases approximately linearly. The balk of the memory is used for recording the elements and groups and is thus almost directly proportional to the number of elements.

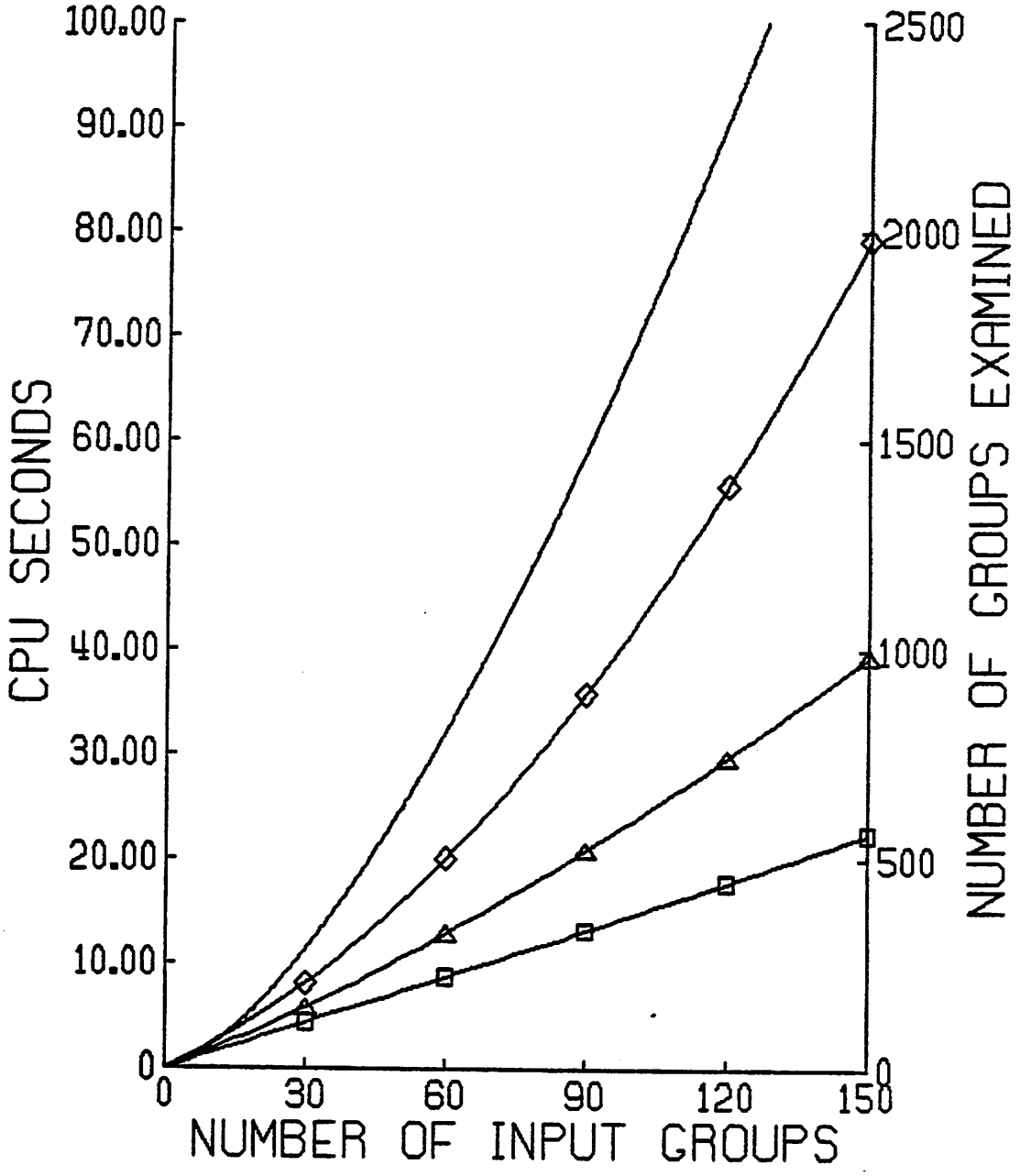


Figure 6.4 The execution time of the compaction operation.

CHAPTER 7

CONCLUSIONS

The complexity of LSI circuits makes it necessary to carry out the layout process hierarchically for the construction of compact, well-organized and error-free layouts. The symbolic layout compaction method described in this report is most useful for the efficient generation of the actual cells and building-blocks at the lowest level of the hierarchy. The generation of actual layouts from symbolic descriptions allows the user to direct his attention to the development of good layout topologies that lead to geometrically simpler, more structured and more compact layouts. The compaction program is used only as an aid to the user and performs the repetitive and tedious tasks of determining and comparing geometric constraints among the symbolic elements. It provides the user with an almost immediate feedback in the form of a compact actual layout of the layout topology the user has chosen. The power of such a man-computer cooperation is illustrated clearly by the layout of the latch-driver example in Chapter 6.

The fundamental algorithm for layout compaction used in the layout generation system described here is a rectangle compaction algorithm related to the rectangle dissection method developed by Tutte et al. [27]. A directed graph summarizing the adjacencies and constraints among elements in the symbolic layout plan is used to provide the compaction algorithm with a global view of the topological structure of the layout plan in the direction of the compaction operation. Based on this graph, the compaction program can determine the most advantageous location for placing each ele-

ment to construct a compact geometric layout. The compaction process is carried out iteratively in the horizontal and the vertical directions until the layout is free of violations of layout design-rules and the user is satisfied with the overall size and organization of the layout. The relative rigidity of the layout plan in the two directions of compaction may be altered to force the desired aspect ratio for the building block generated by the compaction program. Further research must be carried out to incorporate methods for the evaluation and adjustment of the interaction between the two directions of compaction and for the preservation of the symmetry of a layout.

For algorithmic simplicity and in the interest of providing the user with control over the compaction results, the compaction program does not attempt to make drastic changes to the layout topology specified by the user. The automatic jog introduction is the only topological modification performed by the compaction program and is controlled by the outcome of the previous layout compaction operations. The computer-aided generation of good layout topologies is best implemented with a separate program that tackles the problem with both rigorous mathematical methods, such as the graph planarization methods [21, 22], and heuristic approaches to topology planning such as the strategies described in Chapter 6.

At all stages during the layout generation and compaction process, the user has access to the most recent layout topology with an interactive graphics editor. Thus, the user may guide the progress of the layout compaction process by modifying the interim results and by restricting the movement of certain elements with user-defined constraints. Further, the complete circuit interconnection and composition information is available from the compaction program throughout the compaction process. This information can

be used to derive the necessary input for circuit simulation programs for the examination of the electrical performance of the particular geometric layout generated by the layout compaction program.

At the present time, the compaction program handles only the generation of n-channel polysilicon-gate MOS transistor circuit layouts. This limitation arises from the lack of design-rule analysis capabilities for other integrated circuit fabrication process families. (The design-rule analysis is essential to the construction of the graph containing the adjacency and constraint information.) The element-based design-rule analysis technique has been found useful and general for the polysilicon-gate MOS process supported by the compaction program. Preliminary results from experiments indicate that this design-rule analysis technique can be applied to other MOS process families. The application of this technique to bipolar junction transistor processes must be determined. For all these families of design rules, a translator must be developed to convert the rules specified by the process developer into forms readily usable by an element-based rule analyzer. The grouping of rules according to their level of importance and complexity should be considered also for the more efficient hierarchical analysis of design rules.

The generalization of the design-rule analyzer is only a part of the overall generalization necessary to make the layout compaction and generation system developed here more versatile. The present data structure and supporting algorithms, especially those related to the extraction and construction of the composite edge of a group of symbols, must be improved to accommodate general elements that have arbitrary shape and interconnection points. The ability to incorporate such general elements allows the

efficient construction of new elements by the user and the more natural handling of macrocells for the hierarchical construction of a layout.

Finally, it is significant that the entire layout generation system runs on a minicomputer. The short computation time required and the moderate circuit size handled by the layout generation system indicate the usefulness of such a system as a dedicated designer's workbench. The small amount of computation resources required by this system may be built into an intelligent terminal for the generation of LSI circuit building blocks. The hardware and software needs for incorporating other layout tools, such as the building-block placement and routing programs, in such a workbench should be examined for the construction of an integral and powerful system for LSI circuit layout generation.

APPENDIX 1

THE COMMON FILE

The data organization of the common disk file is shown in the attached chart. The element an individual data block represents is specified by the number stored in the element type entry. The numbers and their corresponding elements can be found in the string substitution file of the PRSLI program in Appendix 9. The element orientation equals 0 if it is placed horizontally and equals 2 if it is placed vertically. The x center and y center entries are used to store the center location of an element. The two entries after each of the center location entries are used to store the distances from the center location to the left (or bottom) and right (or top) edges. Finally, the name entry is used to store a pointer to the name field. The element name is not used at the present.

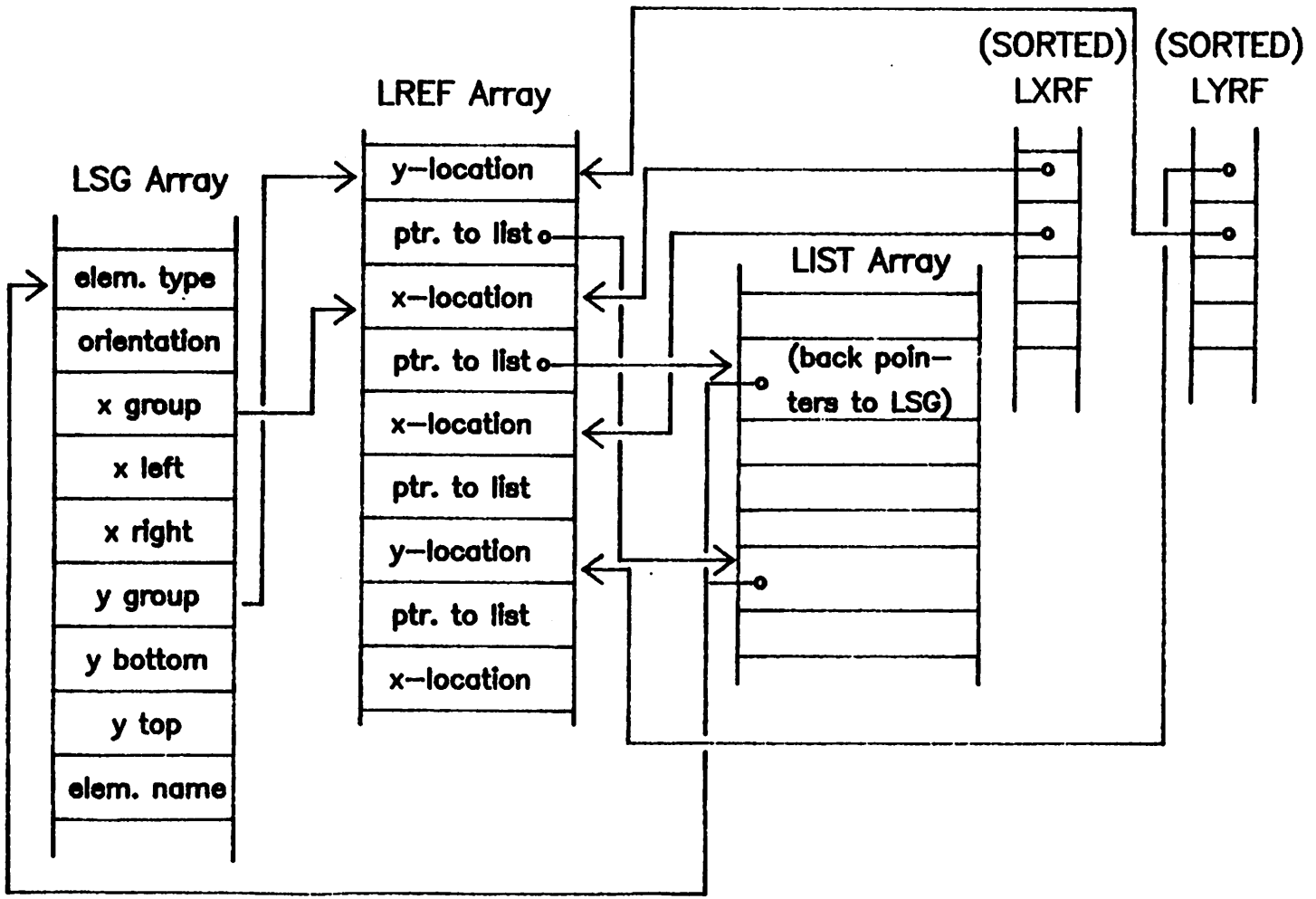
DATA ORGANIZATION USED IN THE DISK FILE

element type
orientation
x center location
x left (offset)
x right (offset)
y center location
y bottom (offset)
y top (offset)
name (pointer)

APPENDIX 2

THE DATA STRUCTURE OF GRLIC

The data structure used in the GRLIC program is shown in the attached chart. The LSG array contains symbol description blocks (SDB's) which have a similar organization as that used in the common file. Note that the x and y center locations are now substituted by pointers to the LREF array. The LREF array is used to store element center locations and pointers to the LIST array in which backpointers to the SDB's are stored. The center locations in the LREF array are sorted and the sorted pointers to these locations are stored in the LXRF (for x center locations) and LYRF (for y center locations) arrays.



The Data Structure of GRLIC

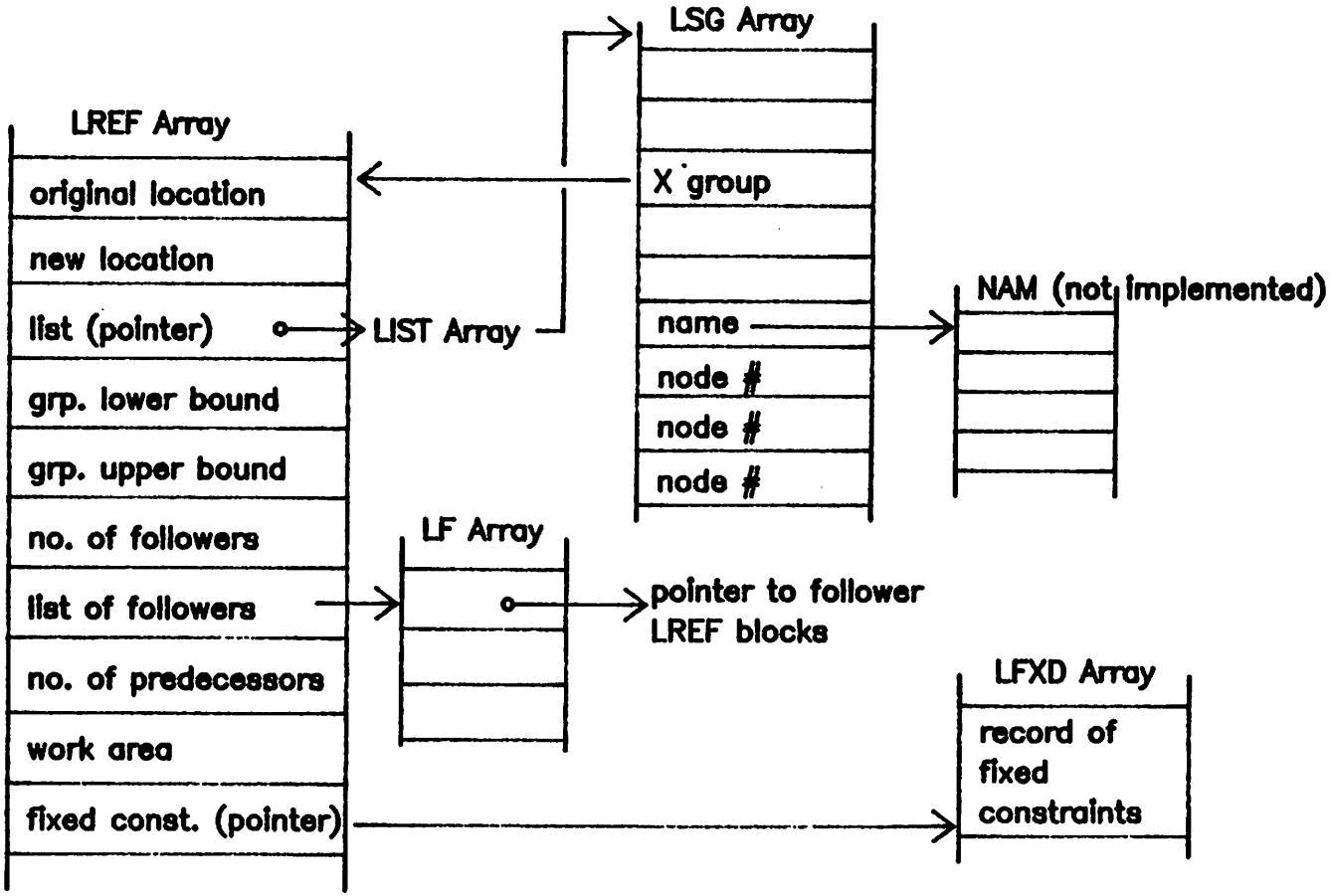
A2.2

APPENDIX 3

THE DATA STRUCTURE OF PRSLI

All the arrays used in the data structure of GRLIC are used in PRSLI. As such, only the additional arrays and the augmented LREF array are shown in the attached chart. The use of each entry in most of the arrays are documented in the string substitution file for the PRSLI program in Appendix 9.

The LREF array is augmented to include information for design-rule analysis (such as the high and low ends of the group) and for longest path calculation (such as the new group location, the number of followers, etc.). Three working arrays (NWST, NBST, ISTK) not shown in the chart are used for the development of composite edges of the primary and the neighboring groups. A fourth working array (MCST) is used to store the "forces" on the primary group for the purpose of jog introduction.



Data Structure of PRSLI (Partial)

APPENDIX 4

COMMON DESIGN RULES AND THEIR IMPLEMENTATION IN CABBAGE

1. Introduction

A major portion of a set of geometric layout design rules for a polysilicon-gate MOS circuit fabrication process commonly used in the industry is implemented in the CABBAGE system. The detailed derivation and explanation for rules similar to those used in CABBAGE may be found in the book by Alquist [41]. The purpose of this appendix is to indicate the actual or possible implementation in CABBAGE of most of the rules in that design-rule set.

2. Forms of Design Rules Suitable for CABBAGE

Because CABBAGE treats a layout as a collection of elements (interconnection lines, transistors, contacts, etc.) rather than rectangles on separate mask levels, rules associated with elements can be implemented readily. In addition, since elements are grouped together, rules affecting related elements (either in the same group or in neighboring groups that are inspected for mutual constraints during the design-rule analysis) can be included easily.

It is indeed very natural and general to implement design rules at the element level. Most design rules arise from considerations of the process tolerances and physical limitations in the construction of elements, such as contacts. Unfortunately, the rules in most design rule books are arranged according to the mask levels they affect and thus separating a rule regarding a single element into many parts. As a result, the rules stated in a typical

design-rule book must be reorganized for the CABBAGE system. The following sections describe the rules used in CABBAGE based on their corresponding elements.

3. Rules Regarding Interconnection Lines

The minimum line widths and spacings are the most fundamental rules for the interconnection lines (diffusion, polysilicon, and metal lines). The minimum line widths are enforced at the time the lines are drawn. The minimum line spacing rules are stored in the Table of Spacing Rules and used in the manner described in Chapter 4.

In some cases the minimum line spacing is increased in the presence of an interfering geometry. The interfering geometry comes in two major types. First, spacings between diffusion lines must be widened if ion-implant or contact windows are located near the edges of the diffusion lines. (Figure 4.12 in the main text shows an example of this type of requirement.) This type of exception arises mainly as a result of the widening of the diffusion line by the extra processes associated with the interfering implant or contact area. This type of design rules are not implemented in CABBAGE at the present time but can be included easily as rules for composite elements. Since a contact or an implant area located near the edge of a diffusion line normally is put in the same symbol group as the diffusion line, the presence of such interfering geometry can be detected easily. The diffusion line with such interfering geometry may be treated as a composite element and the wider spacing requirement may be used as the minimum spacing requirement between it and a regular diffusion line.

The second type of exceptions to spacing rules involves the reflection of light during the photolithographic process by the surface steps created by previous processing operations. Since the reflection of light by the steps causes an effective widening of the patterns on the mask, the separations between mask patterns must be increased to compensate such reflection effects. In the particular polysilicon-gate MOS circuit fabrication process supported by CABBAGE, such reflection problems occur when either the polysilicon or the metal mask is placed over a diffusion region. The rules regarding the reflection effects for the polysilicon mask can be taken care of very easily on an element basis, since an element (a transistor, a buried contact, etc.) is created whenever a polysilicon line crosses a diffusion region. Unfortunately, since there is no spacing requirement between a metal region and a diffusion region, the reflection rules for the metal mask cannot be implemented efficiently with the data structure used in CABBAGE at the present time. (The compactor of the CABBAGE system, PRSLI, allows mergeable regions or regions without mutual spacing requirements the freedom of movement by not keeping track of their relative movement. Thus, it is difficult for PRSLI to determine if a metal region is on top of a diffusion region.)

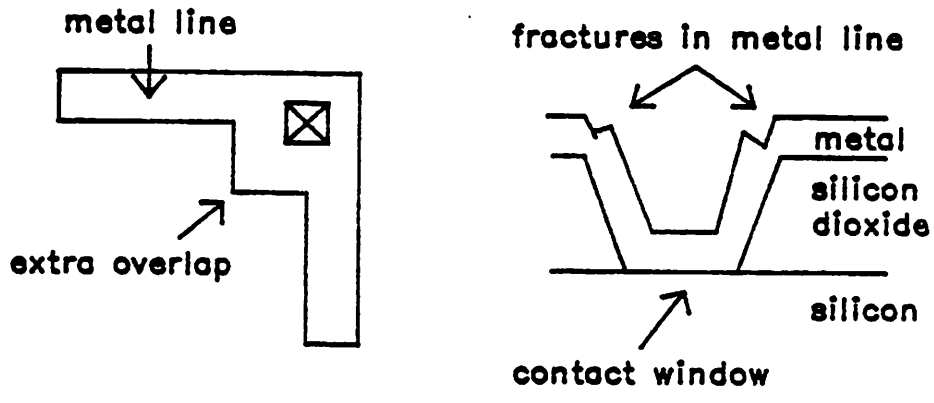
4. Rules Regarding Contacts

Contacts are elements for joining regions or lines on different mask layers. Thus a contact window must provide a sufficient contacting surface and must be covered by the material it joins. Such coverage requirements are implemented as the size of a contact element and are enforced at the time the contact element is put in by the user. The size of a contact element consists of the size of the contact window and the extra margin around the

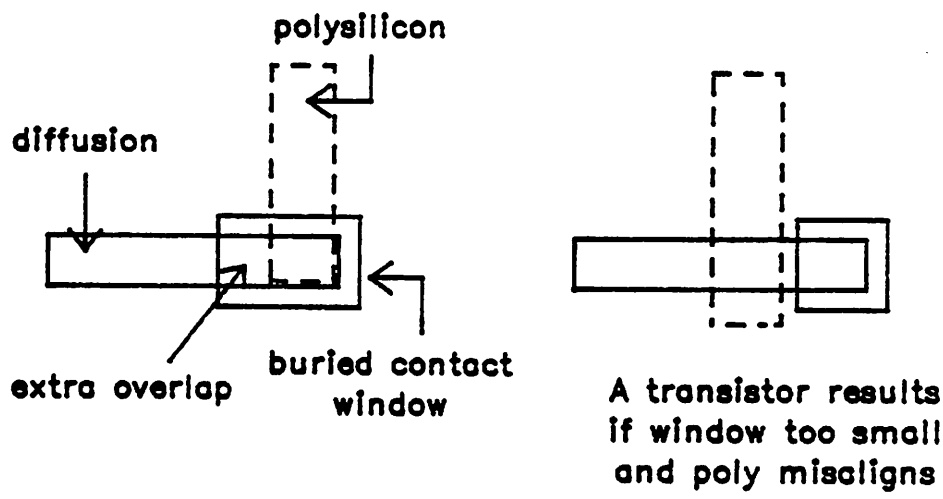
periphery of the contact window that must be covered by each of the two materials coming into contact. (In the case of the buried contact joining the diffusion and the polysilicon regions, the contact window is larger than the lines it joins. Such a larger window is treated in the same manner by CABBAGE as the other types of windows.)

The size of a contact is increased directionally for two kinds of situations. First, for a metal-to-diffusion or metal-to-polysilicon contact, the size of the metal cover must be enlarged in the direction of the metal line for the successful coverage of the surface step created when the contact window is opened on the thick (0.3 to 1 micron) insulation material, as shown in Figure A4.1(a). Second, for a polysilicon-to-diffusion buried contact, the size of the contact window must be enlarged in the direction of the diffusion line to avoid the creation of an unwanted transistor in the case of the worst possible misalignment, as shown in Figure A4.1(b). Both of these directional enlargements of the contact element are handled by a routine in PRSLI which figures out the direction of the relevant lines joined by the contact at the moment that contact element is referred to (displayed or analyzed against other elements).

The spacing requirements associated with a contact element are stored in the Table of Spacing Rules. In addition to the covering materials (diffusion, polysilicon and metal), the contact windows are listed in the table also for implementing rules concerning the spacing between it and other components of an element.



(a)



(b)

Figure A4.1 Directional Enlargement of Contacts

5. Rules Regarding Transistors

A polysilicon-gate MOS transistor is defined by the intersection of a diffusion line and a polysilicon line. As such, the minimum lengths of the sides of the area of the intersection (transistor channel) and the margins of diffusion and polysilicon lines on the sides of the intersection are important parameters for defining a transistor. All these parameters are included as the size of a transistor in the CABBAGE system and are enforced at the time a transistor is specified by the user at the interactive editing phase. In addition, for an ion-implanted transistor the ion-implant window is included as a part of the transistor symbol.

The margin at the end of a polysilicon gate line is often increased as the width of the polysilicon line (the length of the transistor channel) reduces. Such an increase is needed to prevent the edge-rounding effect, as shown in Figure A4.2, and can be handled easily by CABBAGE at the interactive editing phase.

The spacing requirements associated with a transistor are stored in the Table of Spacing Rules and consist of rules governing the spacing of diffusion and polysilicon lines as well as the active channel area. The integrity of the active channel area is important to the performance of the transistor and hence many elements must be placed a certain distance away from the active channel. The ion-implant window is also included in the Table of Spacing Rules for ion-implanted transistors.

6. Rules Regarding the Combination of Regions

Elements or their components at the same electric potential and located within a certain distance of each other often can be joined together.

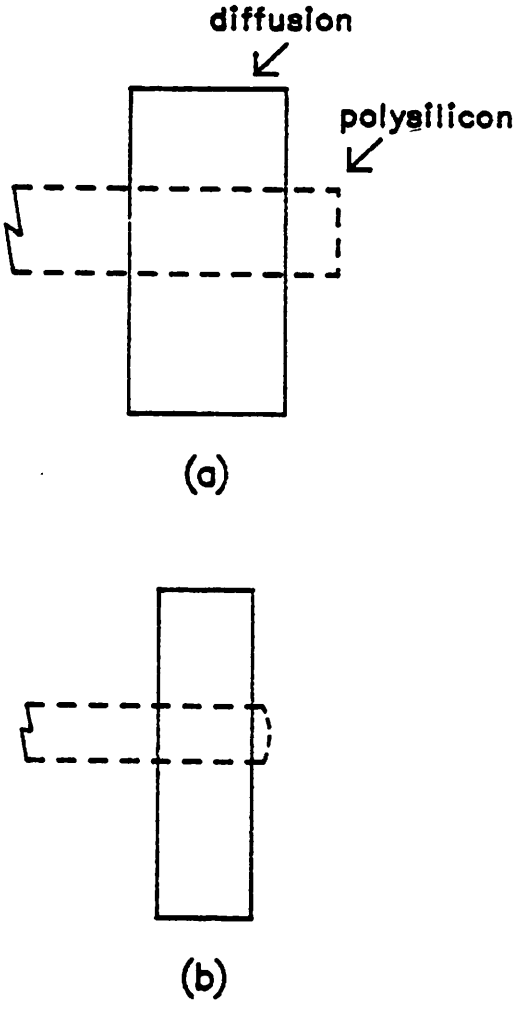


Figure A4.2 The Edge–Rounding Effect.
(a) A Transistor as Drawn on the Mask, and
(b) The Actual Transistor Built. The Poly–silicon is misaligned to the left.

In a hand-layout environment, such conditions for combination are assumed to be obvious to the layout designer and are not elaborated in design-rule books. (In fact, the only combination rule listed in the design-rule book used by CABBAGE is one that encourages the combination of ion-implant windows located within 4 microns of each other.) At the present time, CABBAGE does not attempt to combine elements unless the elements come into contact with each other as a result of the compaction. The lack of the capability to combine nearby elements is a direct result of the previously mentioned policy of not keeping track of unconstrained elements. The nearby and distinct contact windows in the upper left portion of the final layout generated with CABBAGE in Figure 6.2(d) provide an example of this type of peculiarity of the CABBAGE system. Fortunately, the combination of related regions may be performed with a postprocessor as the action of combining the regions normally do not cause design-rule violations in other parts of the layout. The same postprocessor may be designed to smooth edges of a region by removing small indents in the original output from the CABBAGE system.

University of California
College of Engineering
Department of Electrical Engineering
and Computer Sciences

A USER'S GUIDE TO THE CABBAGE SYSTEM
Version 1A - 20 Oct 79

M. Y. Hsueh

1. Introduction

The CABBAGE (Computer-Aided Building-Block Artwork Generator and Editor) system is a symbolic layout system with automatic layout compaction capabilities. It is implemented on the H-P 1000 E-series 16-bit word minicomputer, under the RTE-IVa operating system, and uses the H-P 2648A graphics terminal as the primary user interface. The graphics output can be plotted on the H-P 9872A plotter also.

CABBAGE consists of two programs: GRLIC (Graphics Routines for Laying-out Integrated Circuits) is the interactive graphics editor. It is used to draw symbolic layout plans or "stick diagrams". PRSLI (Packing a Relative-grid Symbolic Layout of an IC) is the layout compactor that generates correct and area-efficient geometric layouts from the symbolic layout input. Note that PRSLI performs both compaction and expansion to generate compact layouts meeting all design-rule requirements. Results obtained from PRSLI may be modified with GRLIC at the symbolic level for further compaction operations.

2. System Limitations

CABBAGE is developed as a layout aid for designing (digital) LSI circuit building-blocks. At present, it can be used to design n-channel polysilicon-gate MOS circuit layouts only. The element and layout design-rule types supported by CABBAGE are similar to those used by the electronics industry in 1977, though the value of each design-rule can be changed with minor modifications to a few routines of the system. A list of elements and design-rule types and their values is given in Appendix 1.

PRSLI does not attempt to perform drastic modifications, such as interchanging elements, on the symbolic layout topology specified by the user. Thus it is essential that the user specifies a reasonable topology as the starting point for PRSLI. (In other words, the user should not expect PRSLI to rearrange, for example, a row of 100 connected transistors into an array of 10 by 10 transistors.)

Lines, point structures (transistors, contacts, etc.), and user-specified fixed constraints are all stored as individual elements. The present version of the CABBAGE system is capable of handling approximately 360 such elements.

3. The GRLIC Program

GRLIC operates in two modes: Commands are accepted in the command mode, which is indicated by the appearance of the prompt sign "GRLIC > ". The use of a drawing command or a command for which the location of the graphics cursor must be supplied transfers the system to the drawing mode. In the drawing mode, the graphics cursor can be moved on a user-defined grid with the ^, @, [, and : keys for the upward, left, right, and downward

movement, respectively. (Note that these four keys are adjacent and and are located at the upper right corner of the alphanumeric keyboard of the H-P 2648A terminal.) The graphics cursor can also be moved with the local cursor-move keys on the terminal keyboard. The system returns to the command mode if any other key is depressed.

The drawing surface is 32,000 units high by 32,000 units wide. One drawing unit is equated to one micrometer (micron) in this version of the CABBAGE system. The graphics screen of the H-P 2648A is used as a window, where the lower left hand corner of the screen is called the window origin. The window can be positioned anywhere on the drawing surface by setting its origin with the ORIG command. Also, the window can be shifted one half of a frame automatically if the user moves the graphics cursor to the window boundary and hits the proper cursor-move key twice. For example, if the cursor is at the top of the screen and the ^ key is depressed twice, the window will be moved up by half the screen height.

The GRLIC command processor accepts command input in free-field format; items in a command usually are separated by one or more spaces. A null command (one consisting of a single carriage return alone) repeats the previous command. The action of any command can be aborted by hitting the "DEL" key, and the system returns to the command mode immediately. The UNDO command can be used to cancel the effect of certain commands issued immediately before it.

The following is a summary of the commands recognized by GRLIC. Due to space restrictions, this summary does not include all possible usages and functions of the commands. Detailed descriptions of these commands are presented in a later subsection.

In the summary, anything inside a pair of brackets, [and], is optional. A "Y" in the U-column indicates the command can be undone.

Summary of Commands

COMMAND	U	MEANING
Q[UIT]		Exit from GRLIC
G[ET] namr		Get file "namr" from disk
S[AVE] namr		Save file "namr" on disk
MER[G] namr [x] [y]	Y	Get file "namr" from disk and merge it at (x,y)
GR[ID] [n]		Set grid spacing to n
O[RIG] [x] [y]		Set window origin at (x,y)
I[DEN]		Show the cursor location
PL[OT] [m] [LU] [t]		Plot mask level m on display device LU with line type t
E[RAS] [name]	Y	Erase the name-ed element
U[NDO]	Y	Undo the last command
D[IFF] [name] [w] [l]	Y	Draw diffusion line; set line width to w, and length to l
P[OLY] [name] [w] [l]	Y	Draw polysilicon line
M[ETA] [name] [w] [l]	Y	Draw metal line
R[UNX] [name] [w] [l]	Y	Draw line
F[IXD] [name] [w] [l]	Y	Draw fixed constraint
T[RAN] [name] [w] [l]	Y	Draw transistor symbol, with channel width w and length l
L[OAD] [name] [w] [l]	Y	Draw implanted load symbol
C[ONT] [name] [w] [l]	Y	Draw contact symbol
TE[RM] [name] [w] [l]	Y	Draw terminal symbol

3.1. Description of Commands

QUIT: The QUIT command allows the user to exit from GRLIC, if the "last change saved" flag is set to true. The "last change saved" flag is set to false if the data is altered by the action of a command, such as adding or deleting an element. The flag is set to true by a) the SAVE command which saves the data on a disk file, b) the GET command which reads in new data from the disk, or c) the QUIT command. Thus, two consecutive QUITs force GRLIC to terminate unconditionally.

GET: The GET command retrieves a file from the disk if the "last change saved" flag is set to true. It has the general form of

GET namr

The file name, namr, must follow the general rule for naming RTE File Manager files.

SAVE: The SAVE command saves a file onto the disk. It has the same general form and follows the same naming rules as the GET command.

MERG: The MERG command merges a file into the current display. At the present, the origin of a file is taken to be at x=0, and y=0. The MERG command allows the user to place the file origin anywhere on the drawing surface, so as to shift the content of the merged file to a new location. The relocation of the file origin can be specified in two ways. The command

MERG namr

transfers the program into the drawing mode to read the cursor loca-

tion. This cursor location is then used as the relocation origin for merging the file namr. The second way to specify the relocation origin is to use the command

MERG namr x y

where the location of the file origin is specified by x and y. Signed x or y specifies incremental relocation; that is, +x or -x means adding or subtracting x from the current window origin (see the ORIG command) and using the result as the x-location of the file origin. Similar operations apply to +y or -y. The default values for x and y are 0.

GRID: The general form of this command is

GRID n

It sets the grid spacing to the absolute value of the integer n. A uniform grid spacing is used if n is a positive integer. If n is a negative integer, a non-uniform grid that is snapped to the center locations of elements is superimposed on the uniform grid. This latter grid system is useful for editing symbolic representations of compacted layouts. The minimum default value of n is 10 in the case of a uniform grid.

ORIG: This command is used to set the window origin. The window origin can be set at the location of the graphics cursor with the command

ORIG

Note that this form of the command transfers the program into the drawing mode to read the cursor location. The second form of the command,

ORIG x y

can be used to set the window origin at location (x,y). Again, signed x or y is treated as an increment from the current window origin, as in the case of the MERG command.

IDEN: This command prints out the cursor location and the window origin. It transfers the program into the drawing mode and displays the information when the drawing mode is properly terminated.

PLOT: This command is used to plot the graphics data on the H-P 2648A or the H-P 9872A. The complete command is

PLOT msk LU ltyp pnspd xshift yshift xmax ymax xmin ymin

where the parameters are all integers. The mask level to be plotted is specified by the first parameter, msk. If msk equals 0 or is unspecified, all mask levels are plotted on the H-P 2648A graphics terminal. The plot is made on the device whose logic unit number is LU. In this version of the program, LU number 9 specifies the H-P 9872A four-color plotter; all other LU numbers send the output to the H-P 2648A graphics terminal. The plot may be done with a particular line type or pen color, as specified by the third parameter, ltyp. The numbers associated with mask levels, default line types and plotter pen numbers are listed in Appendix 2.

The rest of the parameters are used mainly to aid the generation of pen-plots on the H-P 9872A plotter. The plotting speed in centimeters per second is set by pnspd. The default value for pnspd is 5 centimeters per second. The xshift and yshift parameter are multiplying factors which can be used to shift the plot by integral multiples of the plot window in the horizontal and the vertical directions, respectively.

For example, two copies of a layout plan may be drawn side by side by setting the xshift parameter to 0 for the first copy, and 1 for the second copy. (The user must make sure that the actual plotting surface is wide enough to accommodate two copies of the layout plan in the horizontal direction for this example.) The last four parameters, xmax, ymax, xmin, and ymin, specifies the upper right and lower left corners of the desired plot window. The default plot window is the current display window.

ERAS: The erase command deletes an element from the current data. The element to be erased may be specified by its name with the command

ERAS name

The element may be selected with the cursor also, by using the command

ERAS

This second form of the command transfers the program into the drawing mode and reads the cursor location when that mode is properly terminated. The cursor must be placed on the center line of the element to be erased. Although any key other than the cursor-move keys may be used to terminate the ERAS command, the H and the V keys may be used to specify the deletion of a horizontal and a vertical element, respectively.

UNDO: This command undoes the last command, provided that command is undoable. It is ignored if the last command cannot be undone.

DIFF: This is the command for drawing a diffusion line. It has the form

DIFF name w l

where name specifies the name of the element, w specifies its width, and l its maximum length. All three parameters are optional. If the line width w is not specified, the minimum diffusion line width required by the layout design-rule is used as the default value. (See Appendix 1.) If the maximum line length l is not specified, the line length is taken to be variable, and takes the difference of its two end points as its initial length.

The location of a line is specified through digitization. The DIFF command transfers the program into the drawing mode and allows the user to put down end points continuously in that mode. The drawing mode may be terminated by depressing any non-cursor-move key twice.

POLY: This is the command for drawing polysilicon lines. All comments about the DIFF command apply here.

META: This is the command for drawing metal lines. All comments about the DIFF command apply here.

RUNX: This is the command for drawing general-purpose lines. All comments about the DIFF command apply here. This line type is used by the author to outline the border of a building-block.

FIXD: This command allows the user to put down fixed constraints in a layout plan to fix the distance between any two elements. The comments about the DIFF command apply, with two exceptions: A fixed con-

straint does not have a width. However, for uniformity in the command format, an arbitrary width must be specified. In addition, the length of such a constraint is always fixed. (The length is neither the maximum length nor the initial length of a variable line.)

TRAN: This is the command for drawing a transistor symbol. It has the general form

TRAN name w l

where name specifies the name of the transistor, w specifies its channel width, and l its channel length. Again, all parameters are optional. If omitted, the channel width and length are defaulted to the minimum values required by layout design-rules.

The center location of the transistor is specified by the cursor location in the drawing mode. A vertical transistor (one whose channel is vertically oriented) can be specified by terminating the drawing mode with the V key. Similarly, a horizontal transistor is obtained by terminating with the H key.

LOAD: This is the command for drawing an implanted load device symbol. All comments about the TRAN command apply.

CONT: This is the command for drawing contact symbols. The present version provides the user with three types of contact symbols: a) the diffusion-to-metal contact, b) the polysilicon-to-metal contact, and c) the diffusion-to-polysilicon buried contact. The size and name of all three symbol types are specified with the command

CONT name w l

For diffusion or polysilicon to metal contacts, w and l are the width and the height of the contact window. For the buried contact, w and l specify the width and the height of the area where the polysilicon overlaps the diffusion. The type of a contact is specified at the termination of the drawing mode for the contact command; A diffusion to metal contact is drawn if the D key is depressed to terminate the drawing mode. Similarly, the P or the B key may be used to draw a polysilicon to metal or a buried contact respectively. Because contacts may be expanded in the direction of current flow, as required by many LSI circuit layout design-rules, GRLIC always displays the maximum possible contact size instead of the nominal size specified by the user.

TERM: All line elements must be terminated for the proper operation of the compactor. A line may be terminated by another line or by a "terminal" symbol, which is specified by the command

TERM name w l

The terminal width and height are specified by w and l, respectively. For example, a terminal with w = 6 (microns) and l = 20 (microns) may be used to terminate a 20 (microns) wide horizontal line element. Similarly, a 9 (microns) wide vertical line may be terminated by a terminal with w = 9 and l = 6.

3.2. Drawing Rules

A few rules must be observed in drawing a symbolic layout plan so that it can be compacted properly by the compactor PRSLI:

Rule 1: All line elements must be terminated. A line may be terminated by another perpendicular line or with a terminal symbol. In particular, a line element should not be terminated by any other point structure, such as a transistor, alone.

Rule 2: All point structures other than the terminal symbol must be placed at the intersection of lines. The function of a point structure is to indicate the type of the intersection where it is located.

Rule 3: The center locations of any two point structures must not overlap. In general, the user should save a reasonable amount of space around each element in the initial symbolic input. This may save the user from redoing the entire input when it is desired to modify a part of the symbolic input to achieve a better compaction result. Any violation of minimum spacing design-rules will be corrected by the compactor, provided the elements in question do not overlap each other.

Rule 4: A border must be drawn around the symbolic layout plan. The line type RUNX may be used to draw the border. A terminal symbol must be placed at the intersection where a line terminates on the border.

4. The PRSLI Program

Symbolic layout plans drawn with GRLIC can be compacted and expanded with PRSLI to generate area-efficient and correct geometric layouts.

A brief description of the internal operation of PRSLI is presented here to give the user some insights into the way a particular compaction result is obtained by the PRSLI program. PRSLI performs compaction in the horizontal and vertical directions separately. It takes a topologically connected

column or row of elements as a group and proceeds to determine the maximum spacing requirements between neighboring groups. The calculation of spacing requirements is done in a manner similar to those used in modern design-rule check programs. The maximum spacing requirements are subsequently analyzed to derive a set of groups whose combined spacing requirements gives the lower bound on the overall length of the layout in the direction of compaction. These constraining groups are used as "seeds" to determine the proper locations of non-constraining groups.

As a result of the compaction scheme used in PRSLI, connected elements on the same center line location will remain connected, on the same center line, after compaction. Diagonally adjacent elements will also observe design-rule requirements properly.

In theory, a layout plan may be compacted to meet all design-rule requirements after just two compactions; once in the horizontal direction, and once in the vertical direction. However, the implementation of certain design-rules, such as the enlargement of contact covers in the direction of current flow, makes it necessary to carry out the compaction operation repeatedly until the placement of groups in the layout plan meets all design-rule requirements and user-defined constraints. When this occurs, the layout is said to be "legal" and the compactor would display a message to this effect on the terminal. Since Groups still may move in a legal layout and such movements may result in an even more compact layout, the user is allow to let the compactor continue the compaction process for a legal layout. The layout is said to have "converged" if all of its groups have converged to their respective final locations and no more movement of groups is possible. (The correctness of a layout cannot be guaranteed if it is not legal in

both the horizontal and the vertical directions in consecutive compactions.)

Description of Commands

The command processor in PRSLI is very simple and honors only the exact commands. The use of an abbreviated command may result in erroneous operation.

QUIT: QUIT lets the user to exit from PRSLI.

GET namr: This command is used to get a file by the name namr from the disk. The FMGR file naming convention is used for namr.

SAV namr: This command is used to save a file by the name namr onto the disk. The FMGR file naming convention is used for namr.

PL: This command is used for displaying the layout on the terminal or the plotter. It has the same form as the PLOT command described in Section 3.1.

HO: This command initiates a compaction operation in the horizontal direction.

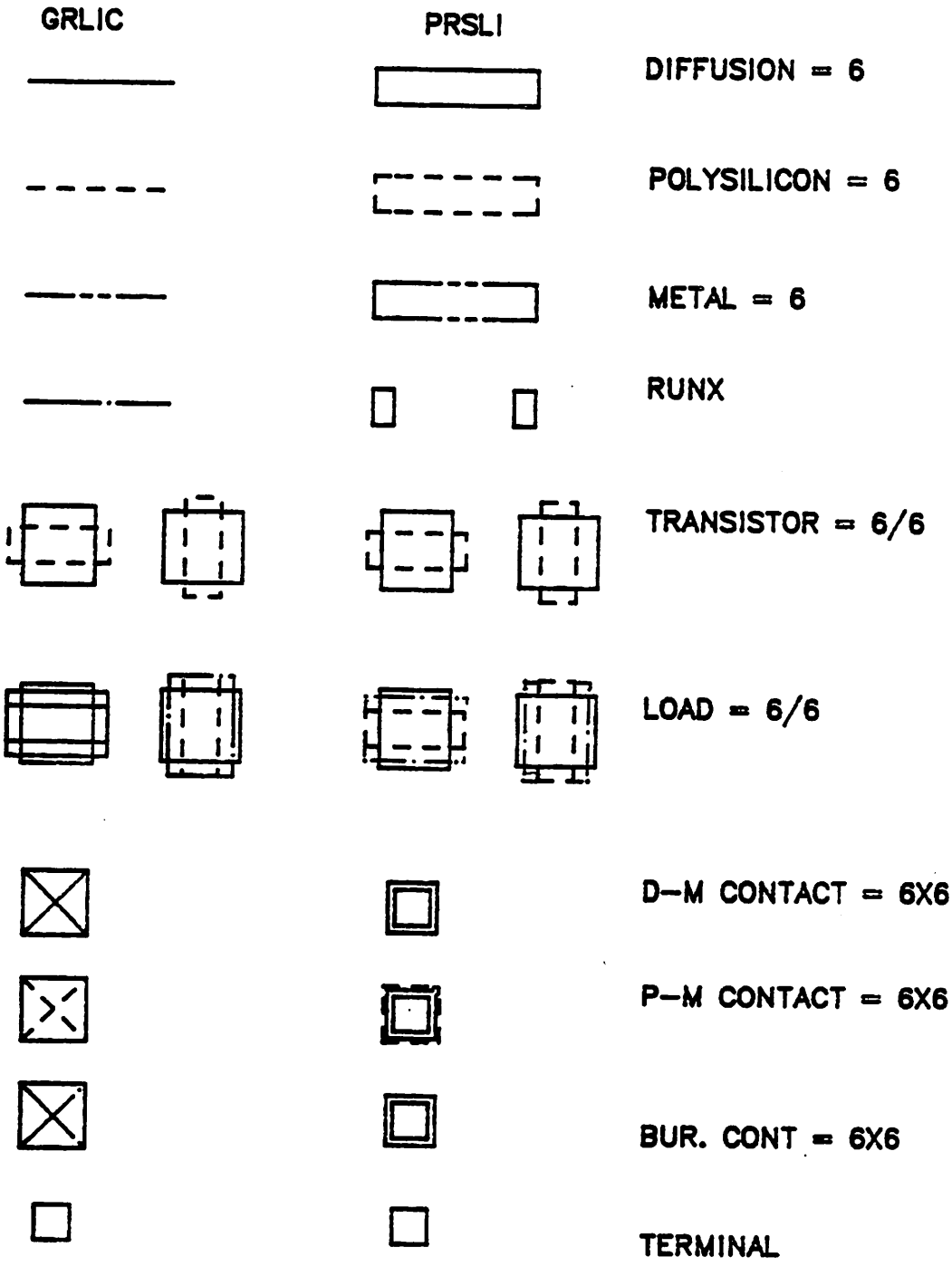
VE: This command initiates a compaction operation in the vertical direction.

GO: This command allows the placement operation to continue for a legal but not converged layout.

JO: This command is used for the automatic introduction of jog points in the direction of the last compaction operation.

Appendix 1
Elements and Their Design-Rules

1. Elements and Their Minimum Sizes



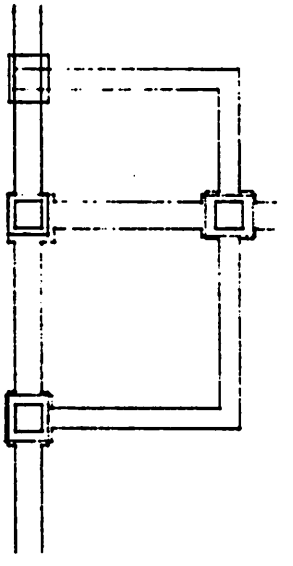
2. Table of Spacing Design-Rules

Spacing rules are stored in an upper triangular matrix (including the diagonal) as follows: (An X indicates that the two mask layers do not have a mutual spacing requirement.)

DIF	PLY	MET	RNX	ACT	CNT	BUR	IMP	TRM
5	2	X	5	X	6	4	X	X
	5	X	5	X	6	4	X	X
		6	X	X	X	X	X	X
			5	5	5	5	5	5
				5	6	4	4	X
					10	2	X	X
						4	X	X
							X	X
								X

3. Other Design-Rules

The metal cover on a diffusion-to-metal or a polysilicon-to-metal contact is enlarged in the direction of the metal line. The buried contact mask is enlarged in the direction of the diffusion line covered by it. These enlargements are shown in the following examples.



Appendix 2

Table of Mask Representations

Mask Level	Line Type	Pen No.
1: diffusion	solid	pen 1
2: polysilicon	dotted	pen 2
3: metal	dash-dot	pen 3
4: a reserved line	dashed	pen 4
5: active area	solid	pen 4
6: contact window	solid	pen 4
7: buried contact	solid	pen 4
8: implant area	dashed	pen 4
9: an additional mask	solid	pen 4

NOTES:

(a) For design-rule analysis purposes, nonphysical mask layers, such as the layer containing active areas of transistors, are included in the Table also.

(b) The different line types are used on the H-P 2648A graphics terminal only. The different pen numbers refer to the pen used for plotting the layout on the H-P 9872A four-color plotter.

APPENDIX 6

COMPUTER-AIDED LAYOUT OF LSI CIRCUIT BUILDING-BLOCKS

A reprint of the first publication on the CABBAGE system.

Proceedings of 1979 ISCAS

Computer-Aided Layout of LSI Circuit Building-Blocks†

Min-Yu Hsueh and Donald O. Pederson

Dept. of Electrical Engineering and Computer Sciences
University of California, Berkeley, CA 94720

Abstract

This paper describes the organization of and the compaction algorithm used in the program CABBAGE. CABBAGE can generate a compact actual layout from layout schematics, which are similar to circuit schematic diagrams, but with improved topology for layout purposes. The compacted layout can be used in the hierarchical build-up of a complete LSI circuit layout. The compaction operation is based on finding the longest path through a graph which represents the size and spacing requirements of the features in the layout schematic diagram. The program is written in FORTRAN and is implemented on a mimcomputer. Circuits of the complexity of a T flip-flop require approximately 6 seconds to compact. The maximum circuit size that can be handled by the program is limited only by the data storage available on the computer.

1. Introduction

Building-blocks of an LSI digital circuit are logical partitions of an overall chip function; they can vary in size and complexity from that of a major function, such as a complete control section, down to a logic element level, such as gates. At present most of the building-blocks implemented in random logic form are laid-out manually or with interactive graphics tools, such as commercially-available graphics processing systems. With either of these layout methods, most layout designers find it advantageous to convert the circuit schematic diagram into a layout schematic diagram which has an improved placement of elements and interconnection lines and a reduced number of crossovers before generating a compact actual layout meeting all design-rule specifications. An example of a layout schematic diagram is shown in Fig. 1. In other words, the layout operation is carried out in two steps: the *topology* of the layout is optimized before the exact *geometrical* requirements are imposed.

While most designers can generate with ease efficient layout schematics, the actual layout process is a tedious and error-prone task that few enjoy. This paper describes the organization and the compaction algorithm of the computer program CABBAGE (Computer-Aided Building-Block Artwork GEnerator), which generates correct and compact layout drawings from layout schematic diagrams input by the user.

2. An Overview of CABBAGE

CABBAGE consists of an interactive graphics input processor and a compactor. The graphics input processor allows the user to draw layout schematics interactively on a refresh CRT terminal. It supports a set of basic elements, such as different line types, transistors, contacts, etc., as well as the capability to merge into the existing drawing predefined "library building-block" files. Since the user must either specify the size (width and/or length) of basic elements or use their default values, design-rules specifying minimum size requirements are enforced at this input phase.

The compactor operates on the user input to determine the exact location of each of the elements, based on the elements' size and spacing requirements. Since the particular compaction algorithm used here detects the most congested areas in the layout schematic diagram before calculating element locations, further improvement on the resulting layout is carried out only in these congestion areas to achieve computation efficiency. The compacted layout also can be converted back to the schematic form so that the user can make manual improvements on the intermediate result.

At the present time, CABBAGE is written specifically for the silicon gate N-MOS logic family. However, the compaction algorithm used in it is quite general, as the algorithm is concerned with only the compaction of rectangles.

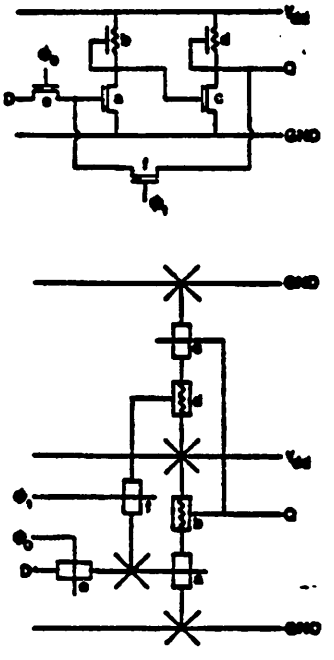


Fig. 1 Layout schematic diagram and its corresponding circuit schematic diagram (top drawing) of a D-type latch.

† This work is sponsored in part by Hewlett-Packard Co., Palo Alto, CA.

3. The Compaction Algorithm

A number of layout schematic diagram compaction methods have been developed and implemented recently [1,2,3]. The detail of the compaction algorithm used in the program described in [1] has not been published. The other programs use either an exhaustive search method or a localized trial-and-error approach by confining the compaction process to a small portion of the overall layout. Such methods are inefficient and require approximately n^2 operations, where n is the number of features (elements and line segments) to be compacted [3].

In contrast, the compaction method described in this paper is based on the most-constraining structure principle. The most-constraining structure in a layout is a group of features which ultimately limits the size of the layout in a given dimension. Thus, modification (reducing the length) of such a structure is most likely to improve the compactness of a layout rapidly. The most-constraining structure can be found if the user's layout schematic diagram is converted into a directed graph to determine its longest path in a given dimension. The use of the longest path principle in layout automation is not new; among other applications it has been used to determine the placement and routing of building-block-based integrated circuits [4, 5] and the placement of hybrid circuit elements [6].

In the following the computer-based compaction process is described with the aid of the T flip-flop example shown in Fig. 2a. One of the N-MOS circuit implementations of this T flip-flop is shown in the layout schematic diagram form in Fig. 2b. This layout schematic describes the desired relative placement and the actual interconnection of transistors and line segments.

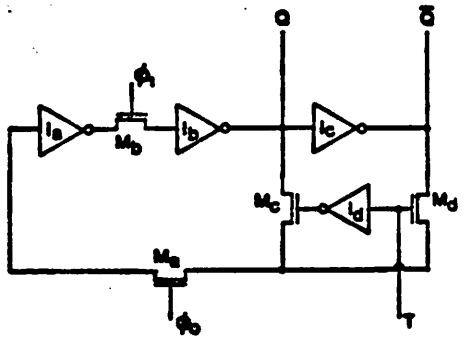


Fig. 2a Circuit schematic diagram of a T-type flip-flop.

Compaction Procedure

Step 0: Layout features in the user input are sorted according to their minimum X and Y values. Node numbers (or path names) are assigned to features to indicate their electrical connections. These operations are done primarily in preparation for an efficient design-rule checking in the following steps. Note, however, a side benefit of being able to generate node numbers is that the same schematic input can be used to describe circuit connectivity to existing circuit and timing simulation programs.

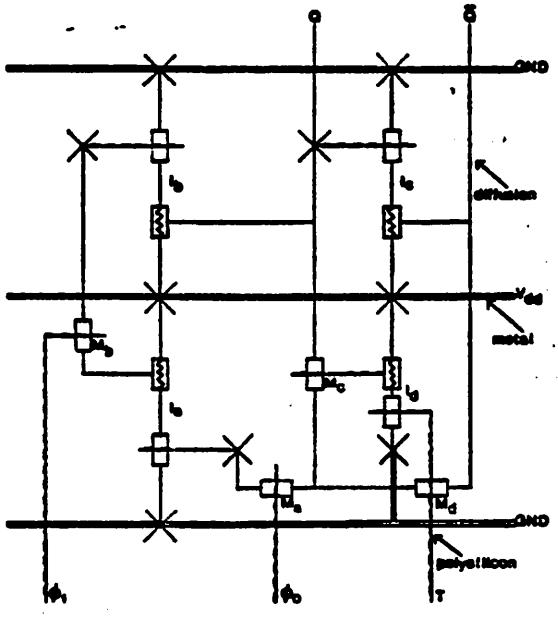


Fig. 2b Layout schematic diagram of the circuit shown in Fig. 2a.

Step 1: The layout schematic diagram is separated into horizontal and vertical portions. The resulting horizontal portion contains all vertically-connected features since these features are to be compacted horizontally. Similarly, the vertical portion contains all horizontally-connected features. This operation decouples movements of the features due to the horizontal compaction from those due to the vertical compaction, and vice versa. It significantly simplifies the required data structure and computational effort at the expense of the ability to compact in a general direction, such as diagonal compaction.

Step 2: For each portion obtained in Step 1, features connected on the same center line are grouped together. Such a grouping operation prevents connected features from drifting apart during the compaction. For example, in Fig. 2b, inverters I_a and I_b and their associated vertical line segments and contacts are all considered to be in the same group.

Step 3: Spacing design-rule requirements are now added to the portion under consideration. This is done using techniques similar to those employed in traditional design-rule checking programs [7]. Consider, for example, the ϕ_1 polysilicon line (the left-most feature) in the horizontal portion of Fig. 2b. Spacing design-rules are applied to calculate the required separation between group center-lines of the ϕ_1 line and the groups bordering it on the right; these bordering groups being a) the ϕ_0 line, b) the transistors forming the inverter I_a, and c) the transfer transistor M_b and its connection line. Since spacing is measured from center line to center line, line widths and element sizes are automatically included in the above calculation. Line lengths are assumed to be variable, unless specified otherwise. At the end of the calculation, only the largest required separation is retained for each pair of groups.

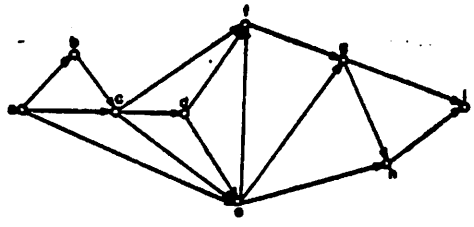


Fig. 3 Graph representation of the relation among the feature groups of the horizontal portion of the layout schematic diagram in Fig. 2b.

Step 4: After the design-rule requirements are added, a graph can be generated to represent the relations among the individual groups. Fig. 3 shows such a graph for the horizontal portion of the example in Fig. 2b. The nodes of the graph represent the individual groups and the branches the separation requirements between groups. The maximum separation requirements calculated in Step 3 are used as branch weights.

Step 5: The longest path is determined through the graph obtained in the last step. Fig. 4 shows the data structure used to calculate the longest path through the horizontal graph of the present example. The path length to a node is the maximum of

a) the existing path length and

b) the path length from its predecessor currently under consideration.

After each path length calculation the number of unprocessed predecessors of a given node is reduced by one. The longest path search operation completes when all nodes have zero unprocessed predecessor.

node	number of unprocessed predecessors	link	list of followers
a	0	—	b, c, e
b	1	—	c
c	2	—	d, e, f
d	1	—	e, f
e	3	—	f, g, h
f	3	—	g
g	2	—	h, i
h	2	—	i
i	2	—	SS

Fig. 4 Data structure used for calculating the longest path through the graph in Fig. 3.

Step 6: The path length to each node indicates the exact location of the associated features. The resulting movement in the compaction direction necessitates a new spacing design-rule calculation in the perpendicular direction. Compaction is carried out in this perpendicular direction next. Thus the compaction process iterates until no new design-rule requirement calculation is necessary.

Fig. 5a shows the result obtained after the user input is compacted in the horizontal direction. Fig. 5b shows the result of a vertical compaction performed on Fig. 5a.

The layout generated with the above procedure is correct but not necessarily the most compact. The size of the layout can be reduced further by modifying the composition of the longest path. In fact, break points in the longest path are good places to generate jogs. Since a break point occurs in a group when the feature limited by a previous group differs from the feature limiting a following group, the latter feature can be moved closer to the previous group in order to move in the following group. Fig. 5c shows the result after jog generation has been performed on Fig. 5b. Other layout improvement methods such as rotating and mirroring features are being developed. All such methods use information derived from the longest path to achieve computation efficiency.

4. Program Performance

CABBAGE is implemented on an HP-1000 Series E 16-bit word minicomputer with an HP-2648A graphics terminal as the primary user interface. The code sections for the interactive input processor and the compactor are both written in FORTRAN, and each of them is approximately 24K words long when compiled. The data area required to store the T flip-flop (Fig. 2) is approximately 2K words long. (The maximum main memory available for data is 4K words. In addition, there are a 32K-word block-transfer type fast memory and a 50-megabyte disk available for data storage.) The computation time for the same T flip-flop is approximately 1.5 seconds per compaction or jog generation. The major portion of the computation time is spent in design-rule checking to determine separation between groups. Since modern design-rule checking techniques typically have a complexity of $n^{1.5}$, where n is the number of features in the layout, the performance of the compactor will degrade slightly for larger circuits.

5. Summary

An algorithm is described in this paper which can be used to generate compact LSI circuit building-block layout from the user's layout schematic diagrams. The use of the layout schematic input allows the user to provide the computer program with a good head-start. The tedious layout work is performed by the program which also ensures the correctness of the resulting layout. Further improvements of the layout are carried out automatically or through manual modifications. In both cases, the most-constraining structure derived by the program are used as a guideline.

With the approach used in the program CABBAGE, the traditional, "after-the-fact" design rule checking, circuit connectivity extraction, and capacitance calculation are all integrated in a forward path. This path starts with the layout or circuit schematic diagram input, and terminates with a complete layout and a set of input ready for use in circuit and timing simulation programs. High-density LSI circuits thus can be built up efficiently from building-blocks of increasing complexity.

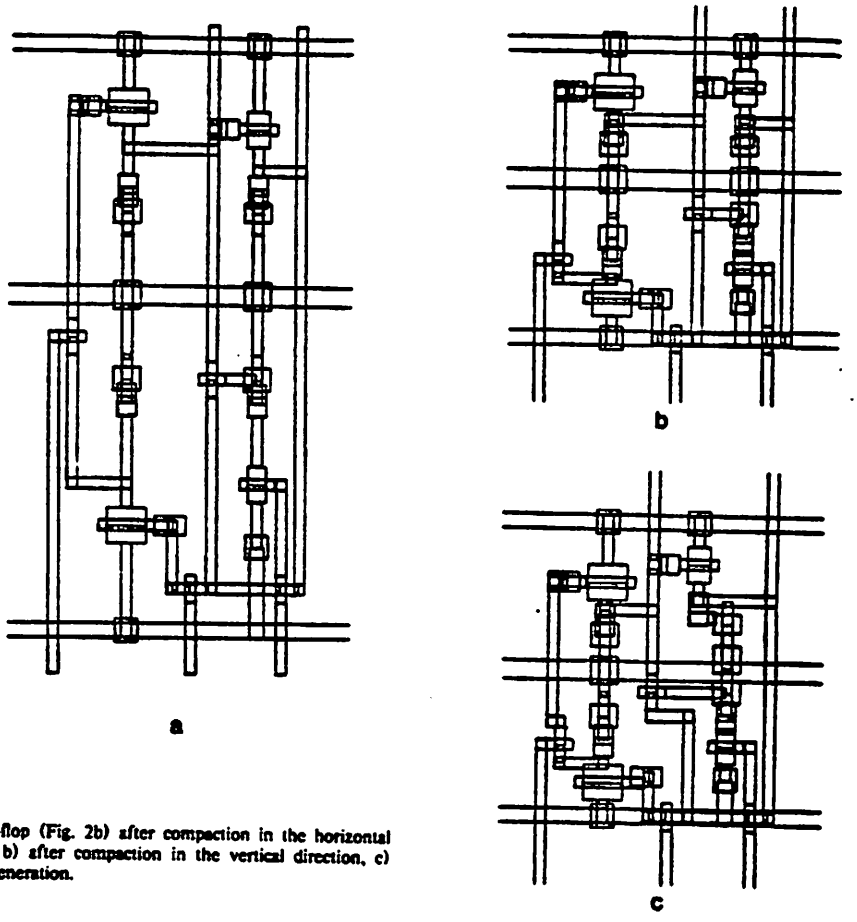


Fig. 5 a) T flip-flop (Fig. 2b) after compaction in the horizontal direction, b) after compaction in the vertical direction, c) after jog generation.

References:

[1] 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 Conferences, June 1977, pp. 138-141.

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

[3] A. E. Dunlop, "SLIP: Symbolic Layout of Integrated Circuits with Compaction," Computer-Aided Design, vol. 10, no. 6, Nov. 1978, pp. 387-391

[4] K. Kani, H. Kawanishi, and A. Kishimoto, "ROBIN: A Building-Block LSI Routing Program," Proceedings of the 1976 International Symposium on Circuits and Systems, pp. 658-661.

[5] B. T. Preas and C. W. Gwyn, "Methods for Hierarchical Automatic Layout of Custom LSI Circuit Masks," Proceedings of the 15th Design Automation Conferences, June 1978, pp. 206-212.

[6] K. Zibert, and R. Saal, "On Computer-Aided Hybrid Circuit Layout," Proceedings of the 1974 International Symposium on Circuits and Systems, pp. 314-318.

[7] H. S. Baird, "Fast Algorithms for LSI Artwork Analysis," Proceedings of the 14th Design Automation Conferences, June 1977, pp. 303-311.

APPENDIX 7

MASKS FOR THE LATCH-DRIVER BLOCK

The following is a set of six mask layers for the latch-driver example shown in Figure 6.2(d) in the main text.

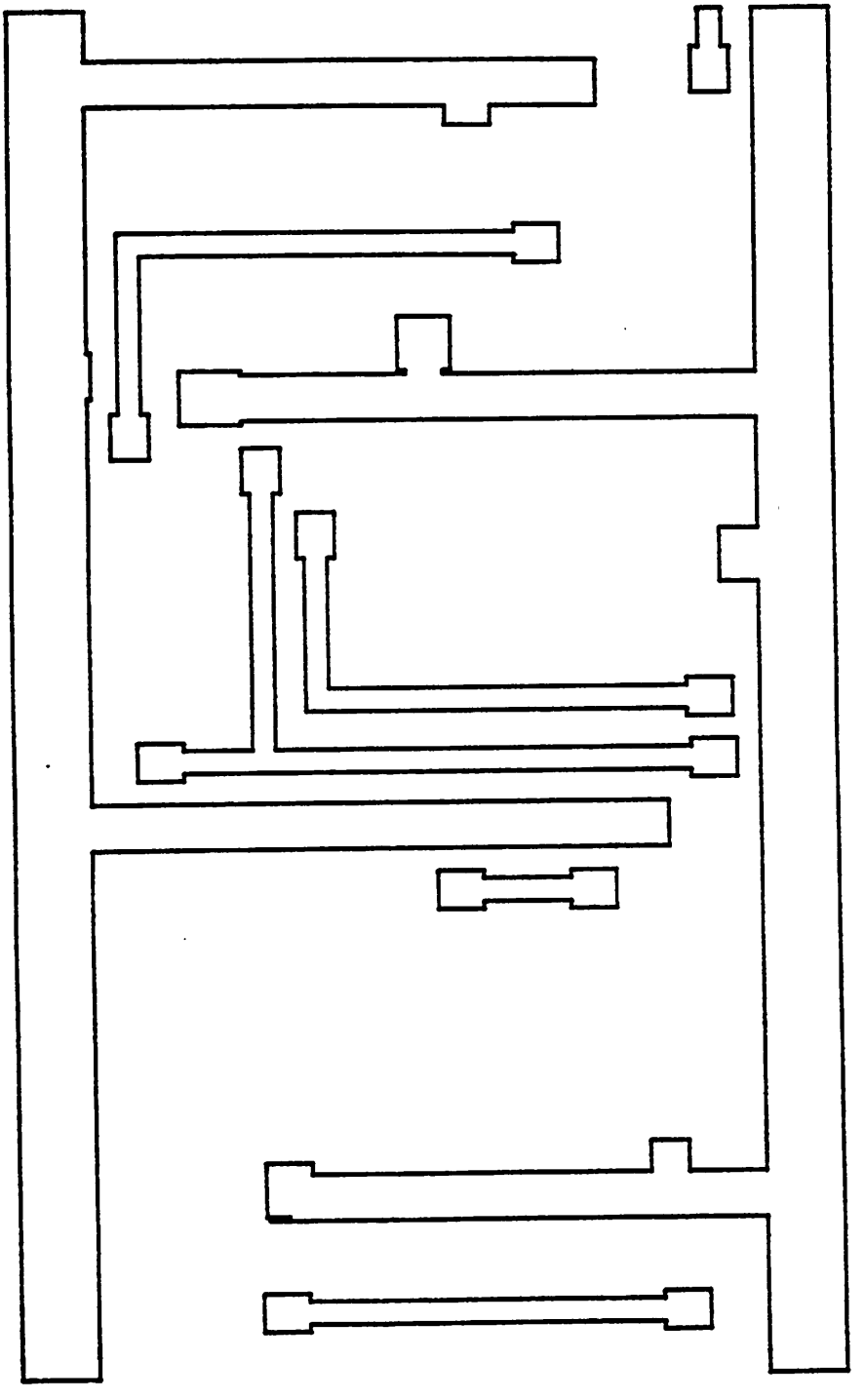


Figure A7(c) Metal mask

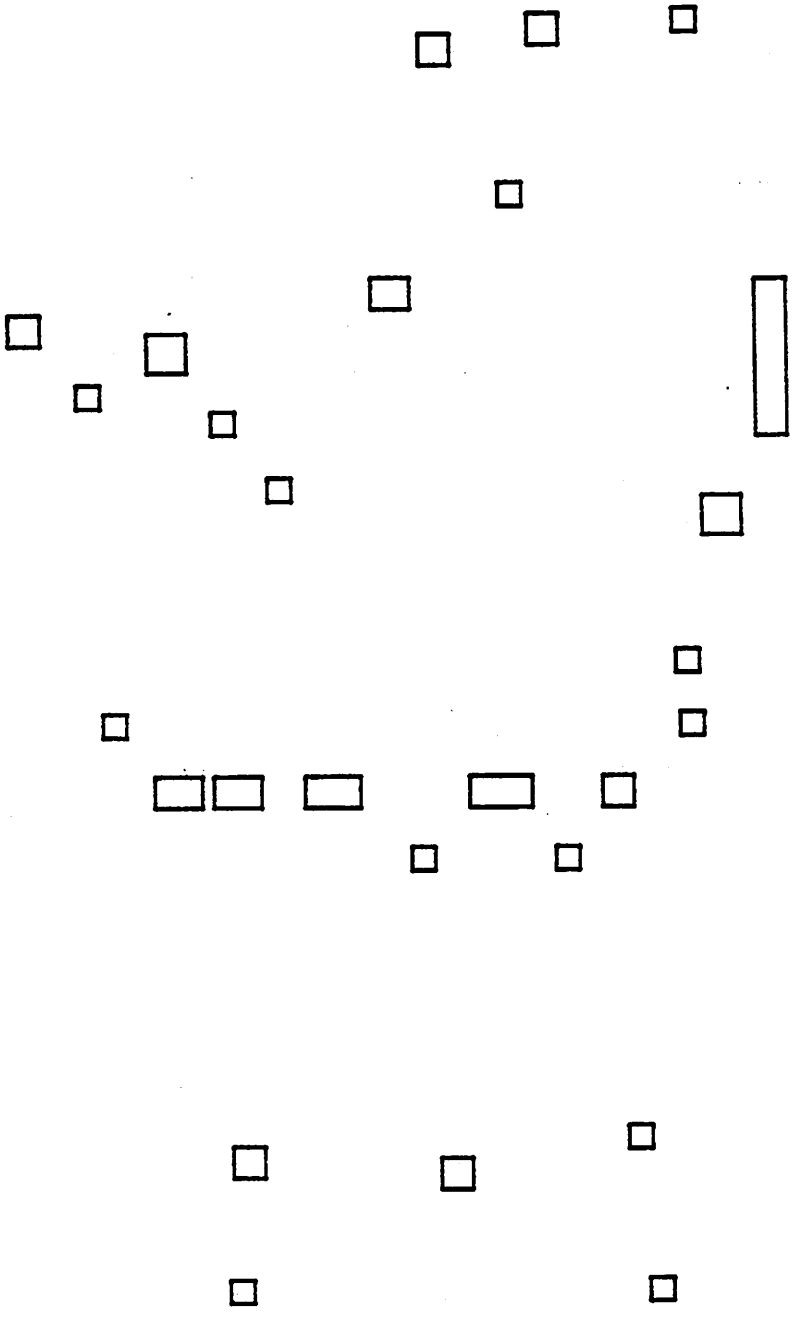


Figure A7(d) Contact window mask

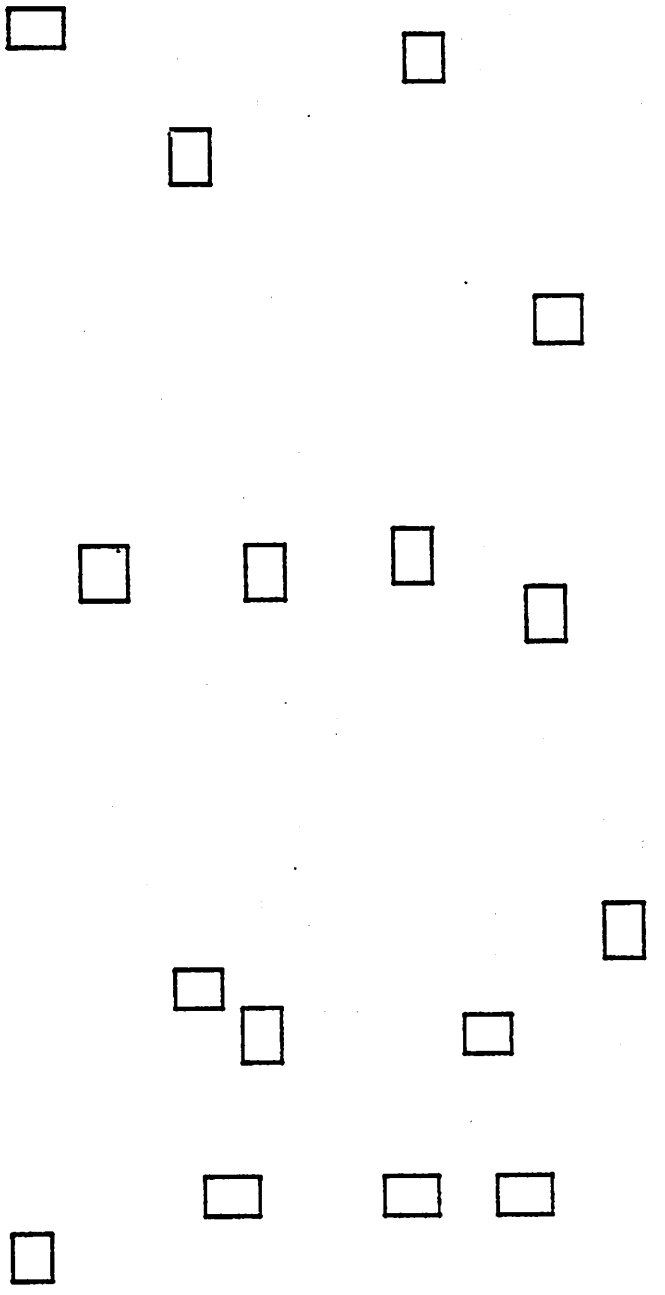


Figure A7(e) Buried contact mask

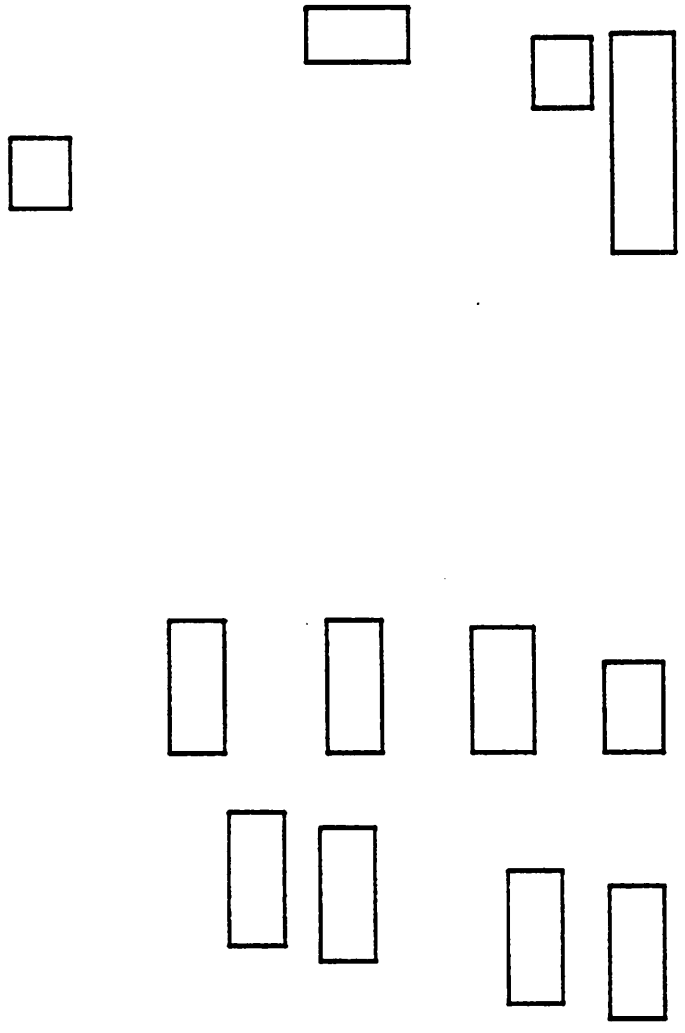


Figure A7(f) Ion-implant mask

APPENDEXES 8 and 9.

PROGRAM LISTINGS

Updated versions of the CABBAGE (GRLIC and PRSLI) program may be obtained from Doris R. Simpson, ERL Publications Office, 433 Cory Hall, University of California, Berkeley, CA 94720.

REFERENCES

[1] C. Mead and L. Conway, *Introduction to VLSI Systems*, Addison-Wesley, 1979.

[2] G. Persky, D. N. Deutsch and D. G. Schweikert, "LTX - A Minicomputer-Based System for Automatic LSI Layout," *Journal of Design Automation and Fault- Tolerant Computing*, Vol. 1, No. 3, May 1977, pp. 217-255.

[3] A. Feller, "Automatic Layout of Low-Cost Quick-Turnaround Random-Logic Custom LSI Devices," *Proc. 13th Design Automation Conference*, June 1976, pp. 79-85.

[4] There are many papers in this area. For example, there are three papers describing interactive graphics editors for IC mask generation in Session 9 of the 1978 Design Automation Conference. *Proc. 15th Design Automation Conference*, June 1978, pp. 182-192, 199-205.

[5] R. J. Blumberg and S. Brenner, "A 1500-Gate, Random-Logic, Large-Scale Integrated (LSI) Masterslice," *IEEE Journal of Solid- State Circuits*, Vol. SC-14, No. 5, Oct. 1979, pp. 818-822.

[6] S. J. Hong, R. G. Cain and D. L. Ostapko, "MINI: A Heuristic Approach for Logic Minimization," *IBM J. Res. Develop.*, Vol. 18, No. 5, Sept. 1974, pp. 443-458.

[7] R. Ayers, "Silicon Compilation - A Hierarchical Use of PLAs," *Proc. 16th Design Automation Conference*, June 1979, pp. 314-326.

[8] U. Lauther, "A Min-cut Placement Algorithm for General Cell Assemblies Based on a Graph Representation," *Proc. 16th Design Automation Conference*, June 1979, pp. 1-10.

[9] K. Kani, H. Kawanishi and A. Kishimoto, "Robin: A Building-Block LSI Routing Program," *Proc. 1976 IEEE International Symposium on Circuits and Systems*, pp. 658-661.

[10] B. T. Preas and C. W. Gwyn, "General Hierarchical Automatic Layout of Custom VLSI Circuit Masks," *Journal of Design Automation and Fault- Tolerant Computing*, Vol. 3, No. 1, Jan. 1979, pp. 41-58.

[11] K. J. Loosemore, "I.C. Layout - The Automatic Approach," *Proc. 1979 European Solid- State Circuits Conference*, pp. 48-50.

[12] W. R. Heller, "An Algorithm for Chip Planning," *SSP Memo #2806*, Computer Science Department, California Institute of Technology, May 1979.

[13] W. R. Heller, W. F. Mikhail and W. E. Donath, "Prediction of Wiring Space Requirements for LSI," *Proc. 14th Design Automation Conference*, June 1977, pp. 32-40.

[14] D. Gibson and S. Nance, "SLIC - Symbolic Layout of Integrated Circuits," *Proc. 13th Design Automation Conference*, June 1976, pp. 434-440.

[15] R. P. Larson, "Versatile Mask Generation Techniques for Custom Microelectronic Devices," *Proc. 15th Design Automation Conference*, June 1978, pp. 193-198.

[16] Y. E. Cho, A. J. Korenjak and D. E. Stockton, "FLOSS: An Approach to Automated Layout for High-Volume Designs," *Proc. 14th Design Automation Conference*, June 1977, pp. 138-141.

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

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

[19] M. Y. Hsueh and D. O. Pederson, "Computer-Aided Layout of LSI Circuit Building-Blocks," *Proc. 1979 IEEE International Symposium on Circuits and Systems*, pp. 474-477. (This article is included in Appendix 6.)

[20] A. D. Ivannikov and P. P. Sipchuk, "Computer-Aided Design of MOS Integrated Circuit Layout," *Proc. IEE - CADMECCS*, Brighton, England, July 1979, pp. 47-51.

[21] J. Hopcroft and R. E. Tarjan, "Efficient Planarity Testing," *J. ACM*, Vol. 21, No. 4, 1974, pp. 549-568.

[22] W. L. Engl, D. A. Mlynski and P. Pernards, "Computer-Aided Topological Design for Integrated Circuits," *IEEE Transaction on Circuit Theory*, Vol. CT-20, No. 6, Nov. 1973, pp. 717-725.

[23] M. A. Breuer (Ed.), *Design Automation of Digital Systems, Vol. 1*, Prentice-Hall, 1972, Chapter 3.

[24] A. Thesen, *Computer Methods in Operations Research*, Academic Press,

1978, Chapter V.

[25] S. B. Akers, J. M. Geyer and D. L. Roberts, "IC Mask Layout with a Single Conductor Layer," *Proc. 7th Design Automation Conference*, June 1970, pp. 7-16.

[26] M. Hanan, P. K. Wolff and B. J. Agule, "Some Experimental Results on Placement Techniques," *Proc. 13th Design Automation Conference*, June 1976, pp. 214-224.

[27] R. L. Brooks, C. A. B. Smith, A. H. Stone and W. T. Tutte, "The Dissection of Rectangles into Squares," *Duke Math. Journal*, Vol. 7, 1940, pp. 312-340.

[28] K. D. Brinkmann and D. A. Mlynski, "Computer-Aided Chip Minimization for IC Layout," *Proc. 1976 IEEE International Symposium on Circuits and Systems*, pp. 650-653.

[29] T. Ohtsuki, N. Sugiyama and H. Kawanishi, "An Optimization Technique for Integrated Circuit Layout Design," *Proc. International Conference on Circuit and System Theory*, Kyoto, Sept. 1970, pp. 67-68.

[30] R. H. J. M. Otten and M. C. van Lier, "Automatic IC Layout: The Geometry of the Islands," *Proc. 1975 IEEE International Symposium on Circuits and Systems*, pp. 231-234.

[31] K. Zibert and R. Saal, "On Computer-Aided Hybrid Circuit Layout," *Proc. 1974 IEEE International Symposium on Circuits and Systems*, pp. 314-318.

[32] A. E. Dunlop, *Integrated Circuit Mask Compaction*, Ph. D. Thesis, Carnegie-Mellon University, Oct. 1979.

[33] J. D. Williams, Private Communications, Sept. 1979.

[34] R. Aurbach, Oral Presentation of the Methods Used in FLOSS, Design Automation Workshop, East Lansing, Michigan, Oct. 1979.

[35] L. Nagel, "SPICE2: A Computer Program to Simulate Semiconductor Circuits," *Electronics Research Laboratory Report No. ERL-M520*, University of California, Berkeley, May 1975.

[36] A. R. Newton, "Techniques for the Simulation of Large-Scale Integrated Circuits," *IEEE Transaction on Circuits and Systems*, Vol. CAS-26, No. 9, Sept. 1979, pp. 741-749.

[37] B. R. Chawla, H. K. Gummel and P. Kozak, "MOTIS - An MOS Timing Simulator," *IEEE Transaction on Circuits and Systems*, Vol. CAS-22, No. 12, Dec. 1975, pp. 901-909.

[38] B. W. Kernighan and P. J. Plauger, *Software Tools*, Addison-Wesley, 1976.

[39] A. V. Aho, J. E. Hopcroft and J. D. Ullman, *The Design and Analysis of Computer Algorithms*, Addison-Wesley, 1974, Chapter 5.

[40] D. Johansen, "Bristle Blocks: A Silicon Compiler," *Proc. 16th Design Automation Conference*, June 1979, pp. 310-313.

[41] N. Alquist, Manuscript, Intel Corp.