CELL GENERATION AND TWO-DIMENSIONAL FOLDING FOR VLSI LAYOUT

by

Dong-Min Xu

Memorandum No. UCB/ERL M90/52

12 June 1990

CELL GENERATION AND TWO-DIMENSIONAL FOLDING FOR VLSI LAYOUT

by

Dong-Min Su

Memorandum No. UCB/ERL M90/52

12 June 1990

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

CELL GENERATION AND TWO-DIMENSIONAL FOLDING FOR VLSI LAYOUT

by

Dong-Min Su

Memorandum No. UCB/ERL M90/52

12 June 1990

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

Cell Generation and Two-Dimensional Folding for VLSI Layout

Dong-Min Xu

Abstract

Most research on VLSI cell generation has been concentrated on linear architectures and one-dimensional optimizing algorithms. In this report, a quadratic architecture, two-dimensional folding-based architecture (2FBA), is proposed. To minimize the area of this new architecture, a two-dimensional folding (or array optimization) algorithm is developed by using a deterministic strategy. The experimental results indicate that the algorithm, like simulated annealing, has the property of climbing out of the local optimums. However, it requires much less computing time.

Based on the new architecture and algorithm, a layout program, PASTORALE, is developed. In order to deal with the practical constraints, a new cost function is used in an improved two-dimensional folding algorithm. Also, the swap operation is added to this improved algorithm.

Some 2FBA layout results are illustrated in this report. The future work is also discussed.

ACKNOWLEDGEMENTS

I would like to express my deepest thanks to Professor Ernest S. Kuh for his support, guidance and encouragement. I would also like to thank Professor Yun-Kang Chen for his guidance in every aspect of my research.

I appreciate Professor Ping Keung Ko's valuable advice. I am grateful to Chuck Kring and Michael Jackson for many technical suggestions. I enjoyed working with all the students and visitors in Professor Ernest S. Kuh's group.

Special thanks are due to Ms. Tahani Sticpewich for providing me a great deal of convenience during my stay in ERL, EECS, University of California, Berkeley.

The work described here was supported by the National Science Foundation.

Chapter 1

1. Introduction

1.1. Module Generation:

A random logic cell is defined as an irregular structure of basic components, such as transistors and gates. Being widely varied in complexity and constraints, the cells cannot be kept in a library without expensive maintenance. Random cell (or module) generators are therefore becoming important in VLSI layout design.

From the silicon compilation point of view [1], the cell generator is a part of the cell compiler, which translates a cell's behavioral description into mask layout. It generates layout from a structural description, a circuit or logic schematic, instead of from a Boolean equation. Cell generation, even from a circuit schematic, is difficult. Traditionally, this problem has been treated by using some different layout architectures.

A layout architecture (or style) is defined by a set of global layout rules to satisfy some constraints on cell size, shape, and I/O pin positions. Global rules specify topological relationships between objects (transistors, contacts and wires), orientation of objects, and layer assignment. The constraints could be the restriction of all vertical wires to the polysilicon layer and all horizontal wires to the metal layer, or the restriction of all P devices to one row and all N devices to another parallel row, etc.

Layout architectures can be viewed as linear (or one-dimensional) architecture or as quadratic (or two-dimensional) architecture. In linear architecture, the components, such as gates and transistors, are placed in a linear array with connections among components. As the circuit complexity increases, the size of the array expands mainly in one direction. The consideration of optimization is also focused on one direction, which results in the one-dimensional assignment problem. In quadratic architecture, however, the components are connected in a more flexible way. The arrays expand in

both horizontal and vertical directions. The area can be reduced by using two-dimensional optimization algorithms. We can thus avoid the long, narrow cell shapes that occur in linear architectures for large circuits.

In recent years, most attention has been concentrated on linear architectures and one-dimensional optimizing algorithms. The systematic formulation and theoretical description of the one-dimensional problem have been well understood for many years. However, quadratic architecture and its theoretical description still need fundamental study. In the following sections, we review several different layout architectures. Some of them are linear architectures; others are quadratic.

1.2. Layout Architectures:

1.2.1. Weinberger Array:

The Weinberger array [2] was the first attempt to stylize the layout implementation of multilevel combinatorial logic. A circuit consisting only of NOR (or NAND) gates is converted into a one-dimensional array of NMOS gates, one gate in each

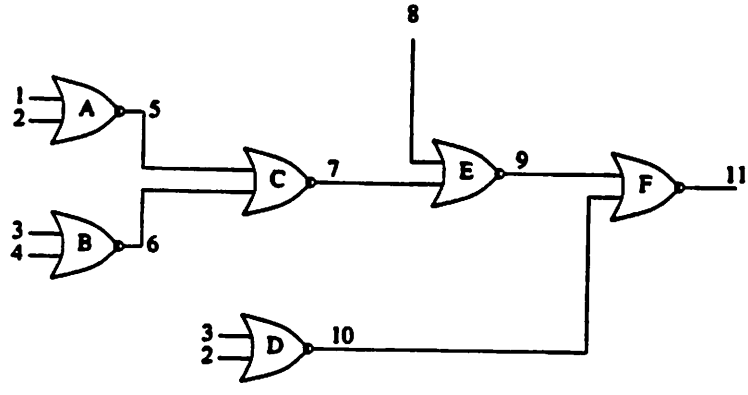


Figure 1

column. Each column consists of two vertical metal wires. One is connected to the pullup transistor and serves as the gate output port, while the other is connected to the ground power line. In a real application, two neighboring gates share a common ground wire. The input signals to the gates are a set of polysilicon rows which are routed horizontally across all gates. A transistor is formed by the intersection of a diffusion segment between the output and ground lines, and a vertical polysilicon extension from a horizontal polysilicon row. A NOR gate schematic and its equivalent Weinberger layout are shown in figure 1 and figure 2, respectively.

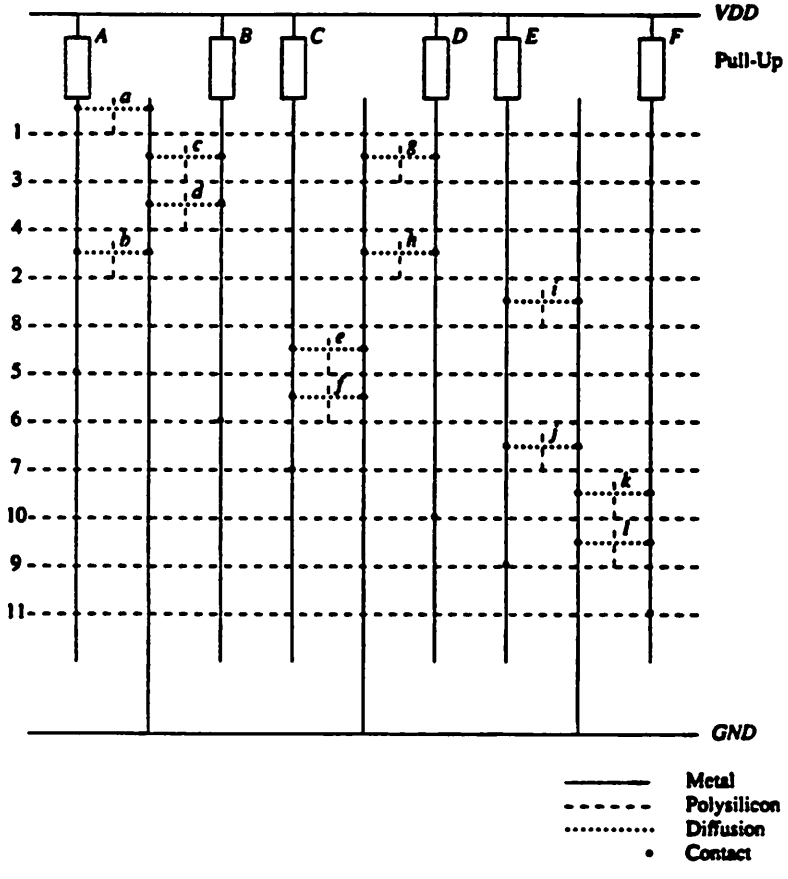


Figure 2

The size of a Weinberger array layout is proportional to the product of the number of gates and the number of nets. Since each gate occupies a single column and

each net occupies a single row, this approach generates a sparse array, although it is simple. In practice, the number of rows can be reduced by having two or more nets share one row, as shown in figure 3. This net assignment problem can be solved by using the left-edge algorithm [3], which can yield an optimal solution. Although the left-edge algorithm is easier to deal with, it does not solve the layout problem since different gate placements result in different net assignments. Finding an optimal gate placement is a one-dimensional assignment (or linear placement) problem, which has been proved to be NP-complete [4].

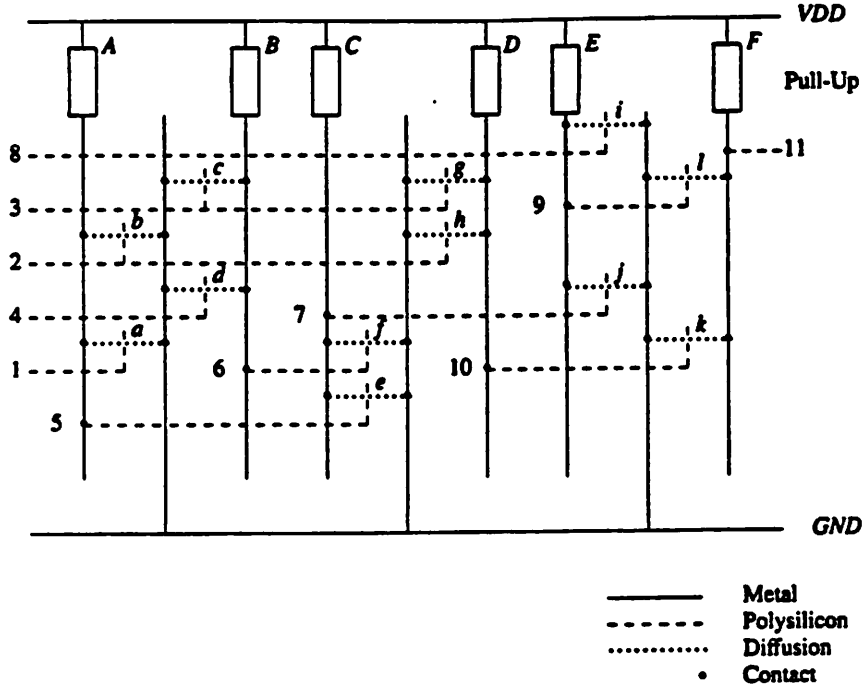


Figure 3

1.2.2. Gate Matrix Layout:

Gate matrix layout is also a linear architecture, which was first introduced by Lopez and Law [5] of Bell Laboratories for the layout of custom CMOS circuits. It is

very similar to the Weinberger array style. Since it uses a simple and regular structure, it is easy to complete a random logic design automatically [6]. The theory behind gate matrix layout was first given by Ohtsuki, etc. in [7].

Gate matrix is a layout style for CMOS circuits. An example is shown in figure 4. In this layout, the P transistors are placed in the top half of the matrix and the N transistors in the lower half, and all the vertical polysilicon gates are placed in columns. The interconnections among transistors are made by horizontal metal lines. A net is defined as a series of metal lines which connect transistors in the same row. The size of a gate matrix is proportional to the product of the number of columns and rows. To minimize a gate matrix, the number of rows can be reduced by placing more than one nonoverlapped net on a row. The number of required rows depends heavily on the column ordering. After the column ordering is decided, the number of rows becomes fixed. Finding an optimal column ordering is a one-dimensional assignment problem.

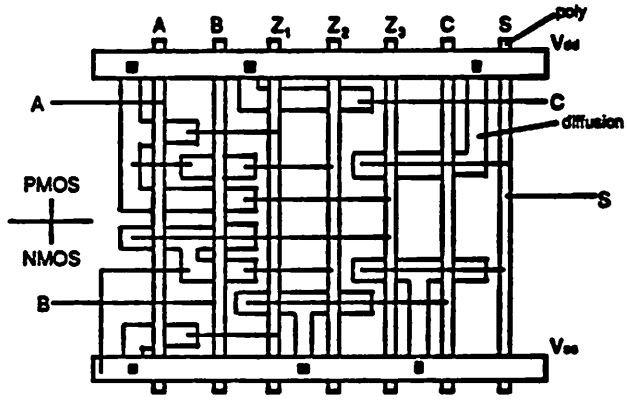


Figure 4

1.2.3. PLA and SLA:

Programmable Logic Arrays (PLAs) are perhaps the most popular structures for the implementation of a bi-level logic function. Most modern VLSI microprocessors

include large PLAs to implement the data path control and a variety of smaller PLAs for controlling other activities on the chip. A PLA can map a set of Boolean functions in canonical, bi-level sum-of-product form directly into a geometrical structure [8]. It is very convenient to implement the logic synthesis from Boolean equations to the mask level automatically. The whole procedure of PLA logic synthesis can be divided into logic-level (or functional) optimization, topological optimization and layout optimization [9].

A PLA consists of an AND-plane and an OR-plane, shown in figure 5. For every input variable in the Boolean equations, there is an input signal to the AND-plane. The AND-plane produces product terms by performing the AND operation on the selected set of input signals. The OR-plane generates output signals by performing an OR operation on the product terms fed by the AND-plane.

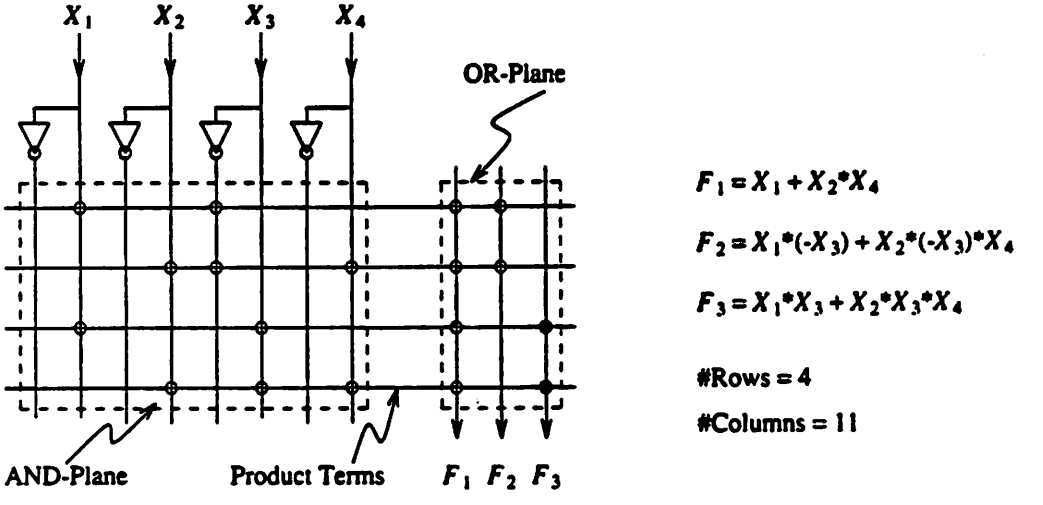


Figure 5

The area of a PLA is proportional to the product of the number of columns and rows used, where the number of rows equals the number of product terms, and the number of columns equals the sum of the number of output signals and twice the

number of input signals. A PLA layout can be compacted by using logic minimization to reduce the number of rows and folding to reduce the number of columns.

PLA folding allows two or more signals to share a single row or column and thus reduce the total number of rows or columns. Figure 6 shows the folding result of figure 5.

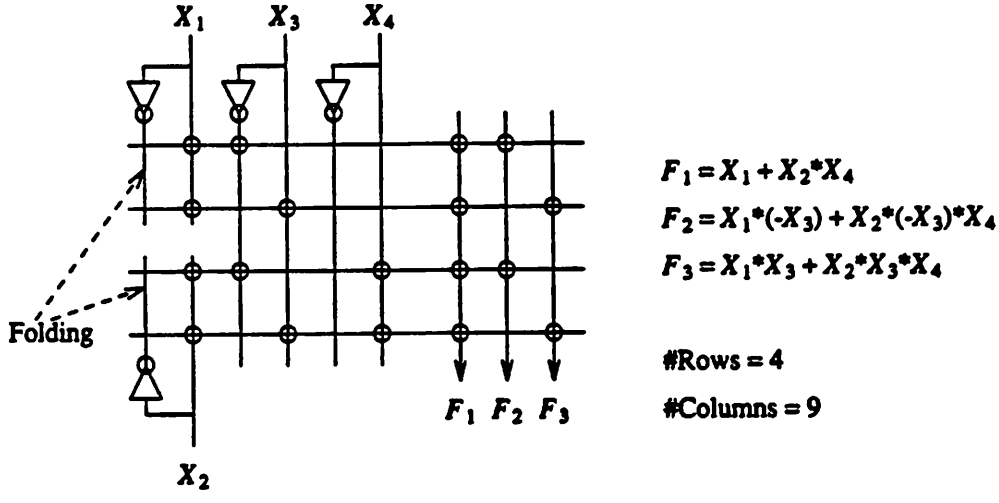


Figure 6

A PLA folding is said to be a simple folding when the maximum number of input, output or product terms in each column or row is less than two. In multiple folding, more than two input, output , or product terms are collapsed into the space of a single term. Early work on PLA folding theory and implementation was carried out in [10]. In [11], the optimal PLA folding problem was shown to be NP-complete.

A PLA multiple column or row folding can be considered to be a one-dimensional assignment problem [12]. If a PLA is folded for both columns and rows simultaneously [13], it becomes a two-dimensional folding problem.

A storage/logic array (SLA) [14][15] is an extension of the PLA architecture. It was first introduced in 1975 [16]. In contrast to the PLA architecture, an SLA mixes

AND and OR operation on a single plane (see figure 7). It allows higher level components such as flip-flop and inverters to be placed at grid points, while a PLA allows only transistors to be placed at grid points.

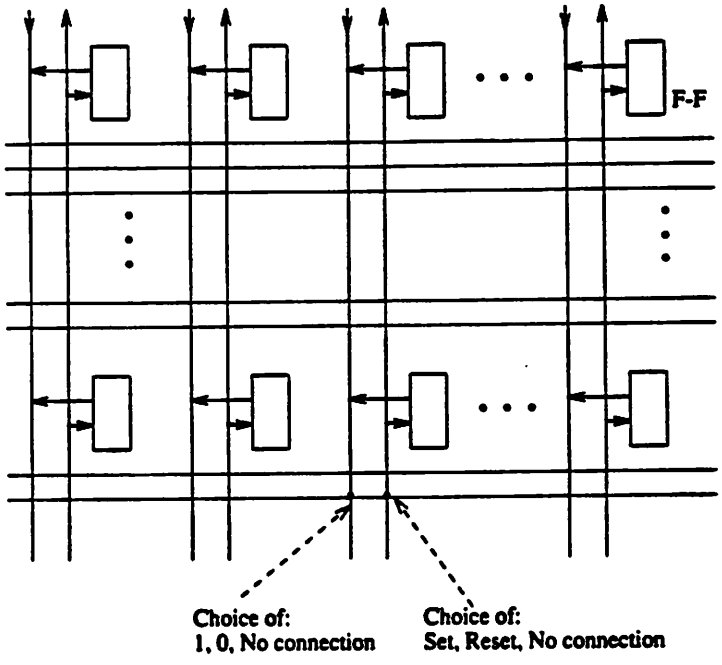


Figure 7

Since the cells in the SLA are of widely differing sizes (one may be ten times the area of the other), two-dimensional SLA multiple folding is even more difficult than PLA compaction.

1.2.4. Multi-level Matrix (MLM) Architecture:

A multilevel matrix (MLM) [17] is a two-dimensional structure like a PLA but supporting multistage logic circuits. It is a hybrid structure, with characteristics of both gate matrix and Weinberger array structure. In an MLM, more than one gate can

occupy a column (like a gate matrix), and more than one signal can occupy a row (like a Weinberger array). Figure 8 shows an example of an MLM.

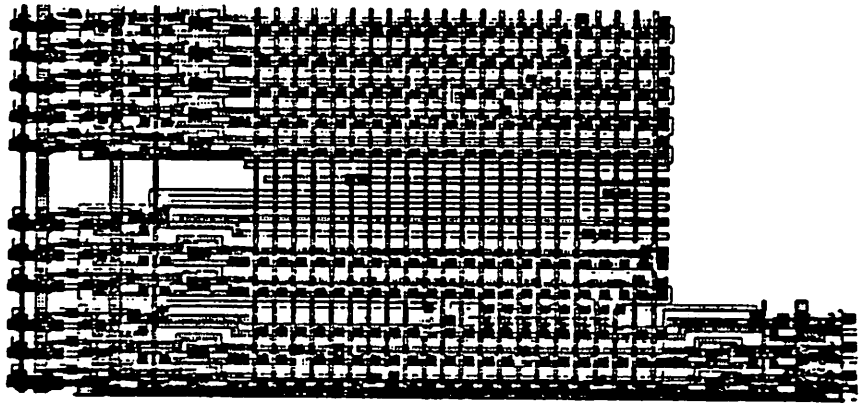


Figure 8

Like a PLA, an MLM can be folded for both rows and columns. Multiple folding can be used for intermediate inputs without any penalty. In [18][19], two different algorithms, TWIST and GENIE, were proposed for MLM folding.

1.2.5. Metal-Metal Matrix (M^3) Architecture:

The Metal-Metal Matrix (M^3) was proposed by Kang[20] for high-speed VLSI circuits in single-poly and double-metal CMOS technology. Unlike gate matrix layout, it employs maximal use of metal interconnections while restricting delay-consuming polysilicon. The most important motivation of M^3 layout is to avoid the RC delay.

Figure 9 illustrates the basic M^3 structure. All the signals run vertically with the second-layer (upper) metal lines forming the signal columns. Interspersed between these metal lines are diffusion columns. The first-layer (lower) metal lines are used to horizontally connect the signal columns to the gates of MOSFETs, to interconnect the sources and drains of MOSFETs, and to run power and ground busses.

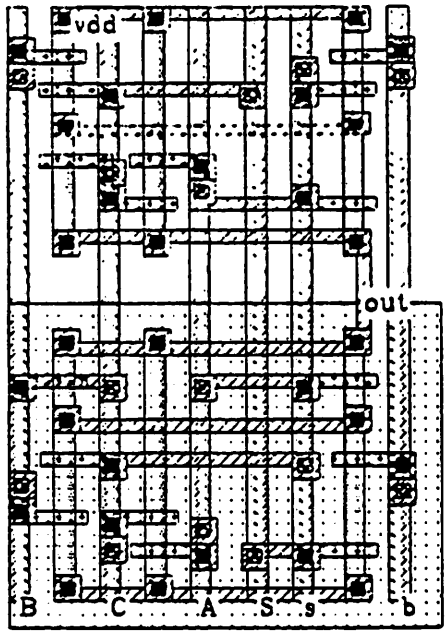


Figure 9

A symbolic layout system, iSILVER, using the M^3 layout style has been reported[21].

In [22], a M^3 generator was reported. It tries to merge some transistor diffusions by reordering the rows of a node incidence matrix A_a. The signals, including diffusion strips, are then rearranged so as to reduce the number of horizontal tracks in M^3 [23].

1.2.6. Doubled Folded Transistors Matrix Layout:

A new two-dimensional folding cell generator [24] was proposed recently. This method is more flexible than the gate matrix layout style and is suitable for NMOS one-layer metal technology. It does not require transistors with the same gate signal to be on the same poly strip so that both vertical nets and horizontal poly strips can be folded. This two-dimensional folding technology makes it possible for the aspect ratio to be adjusted with a large range. Also, the chip area is more compact than it is in gate matrix layout.

In this layout style, a circuit is represented by a net list. The target of the layout is to assign the signal nets to columns and the transistors to rows without overlapping them. The signal nets are implemented in metal, while the transistors use diffusion and polysilicon. Figure 10 is an example of this layout style.

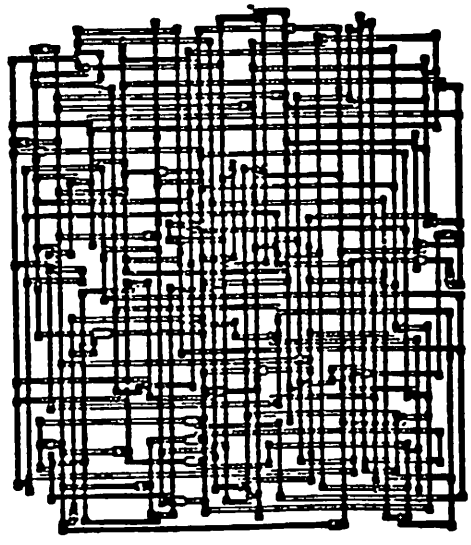


Figure 10

1.3. Optimization Problem:

The dimensionality of a specific layout architecture refers to the number of directions in which a module may grow. In a linear array, the module grows in one dimension with respect to the number of components (Weinberger array) or the number of nets (gate matrix). In a quadratic architecture, the modules grow in both directions. Based on this idea, the Weinberger array, gate matrix and PLA (where folding is allowed in one dimension) are linear arrays, while the Multi-level Matrix (MLM) and PLA (where folding is allowed in two dimensions) architectures are quadratic arrays.

The linear array is usually formulated as a one-dimensional assignment (or linear placement, gate sequencing) problem. It can be stated as follows: Given a set of modules, numbered 1,2, ..., m and a set of nets n_1, n_2, ..., n_k, together with a net list which specifies the connection pattern. A net corresponds to a horizontal wire which interconnects modules assigned to the net. The assignment can be considered to be a permutation $\pi = \{ \pi_1, \pi_2, ..., \pi_m \}$ of modules such that module π_i is placed in the i-th position on a row. The most commonly used objective function for minimization is the total wire length. However, a different criterion is often used in the Weinberger array and gate matrix layout: the number of tracks.

The quadratic array can be formulated as an array optimization problem. The array optimization problem can be stated as follows: Given a set of cells with varying sizes, each cell is connected to a horizontal and a vertical net. To reduce the area of the array, several vertical nets are allowed to share a column and several horizontal nets to share a row. The objective of the optimization is to find a folding result such that the folded array has minimum area.

In this report, the one-dimensional assignment problem is also called the one-dimensional folding problem, and the array optimization problem is also referred to as the two-dimensional folding problem.

1.4. Organization of The Report:

In chapter 2, a new quadratic architecture, 2-dimensional Folding Based Architecture (2FBA) is introduced. Then, a 2-dimensional folding (or array optimization) algorithm is presented in chapter 3. Chapter 4 describes a new module generator program, PASTORALE, which uses 2FBA architecture. Some experimental results are included in chapter 4. Chapter 5 gives some ideas for future work.

Chapter 2

2. A 2-D Folding Based Architecture: 2FBA

This chapter introduces a new architecture which is more flexible than the ones previously discussed. In this layout architecture, the horizontal nets are usually implemented by the second (upper) layer metal lines for gate signals. The horizontal connections can also be made by the short polysilicon segments, if the adjacent transistors are closely spaced. The vertical nets consist of transistors which are connected by the first (lower) layer metal lines. A vertical net may contain more than one transistor. Sometimes, the interconnection among transistors can be diffusion segments. From another point of view, the horizontal and vertical nets are used to form two-dimensional grids on a tile plate. Transistors sit on the grids.

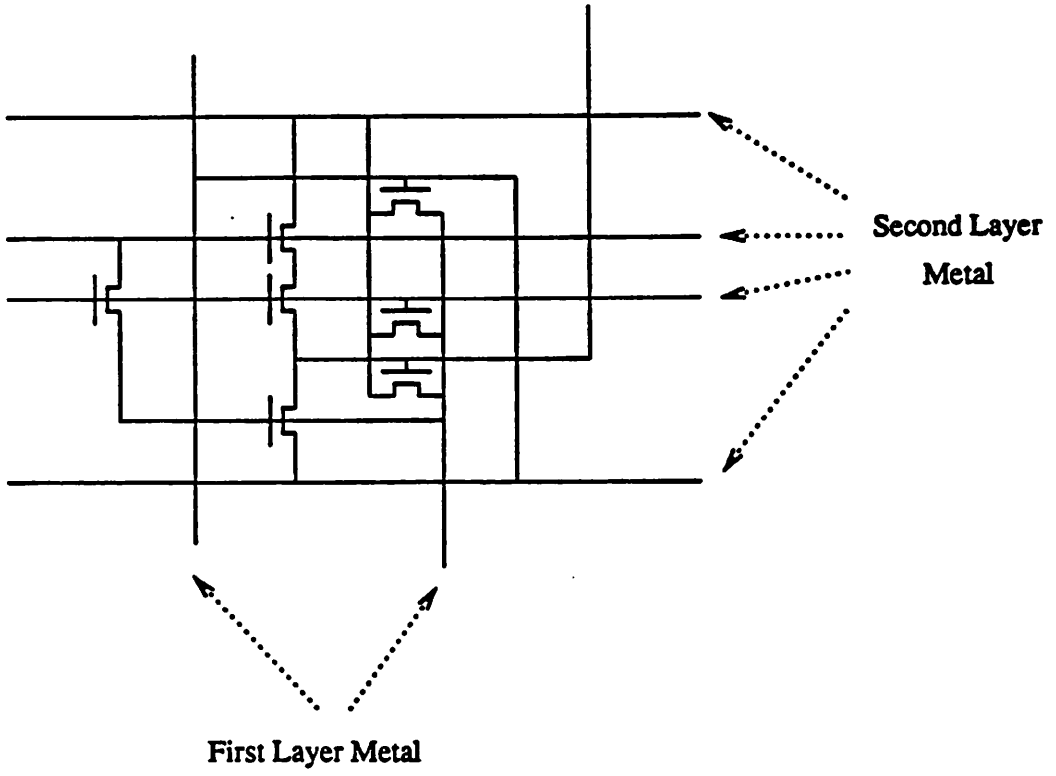


Figure 11

Figure 11 illustrates the basic structure of 2FBA architecture. The primary motivation for this kind of architecture is to be able to minimize the area by using two-dimensional folding (or array optimization) algorithms. To do so, we require both horizontal and vertical nets to be rearrangable, as long as the connectivity is maintained.

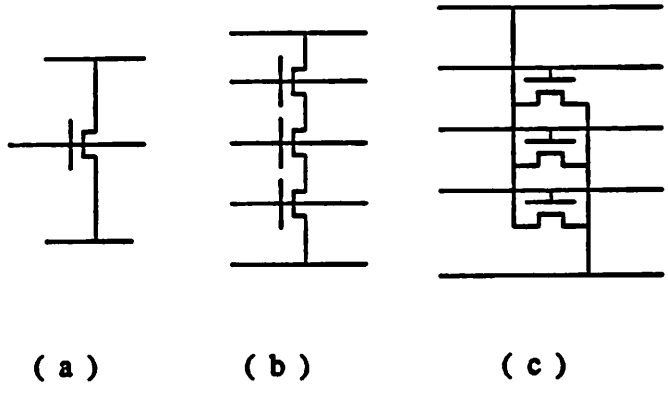


Figure 12

To build a 2FBA architecture according to a given circuit schematic, three basic

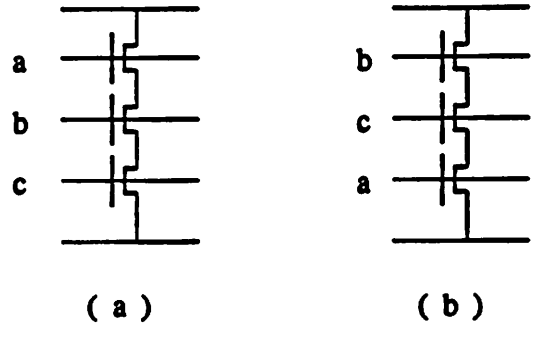


Figure 13

vertical nets are necessary. They are shown in figure 12. Figure 12(a) indicates a single transistor, while figure 12(b) and figure 12(c) represent a group of serial and parallel transistors, respectively. The vertical nets in figure 12(a), figure 12(b) and figure

12(c) are called, respectively, the single transistor nets, serial transistor nets and parallel transistor nets.

In a static circuit, the ordering of series-connected transistors is logically immaterial. For the example in figure 13, the series-connected transistors can be reordered.

As the ordering of the horizontal nets changes, figure 12(a) and figure 12(b) have some variations. They are shown in figure 14. However, the configurations in figure 12(a) and figure 12(b) are preferred.

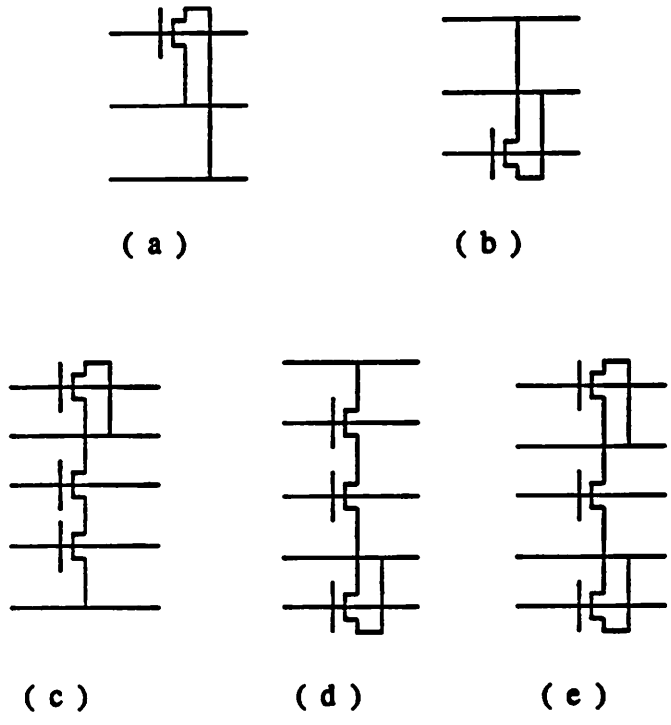


Figure 14

Figure 15 shows a transistor in detail. To connect a transistor's polysilicon with the second layer metal line, we have to connect it to a small segment of the first layer metal by means of a via.

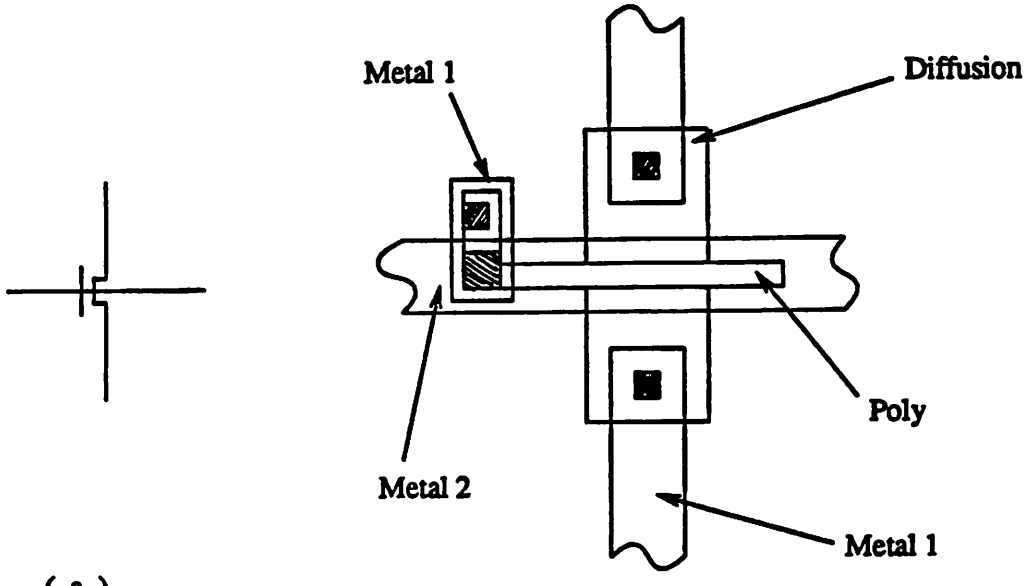


Figure 15

The mask-level appearances of the three basic vertical nets are plotted in figure 16.

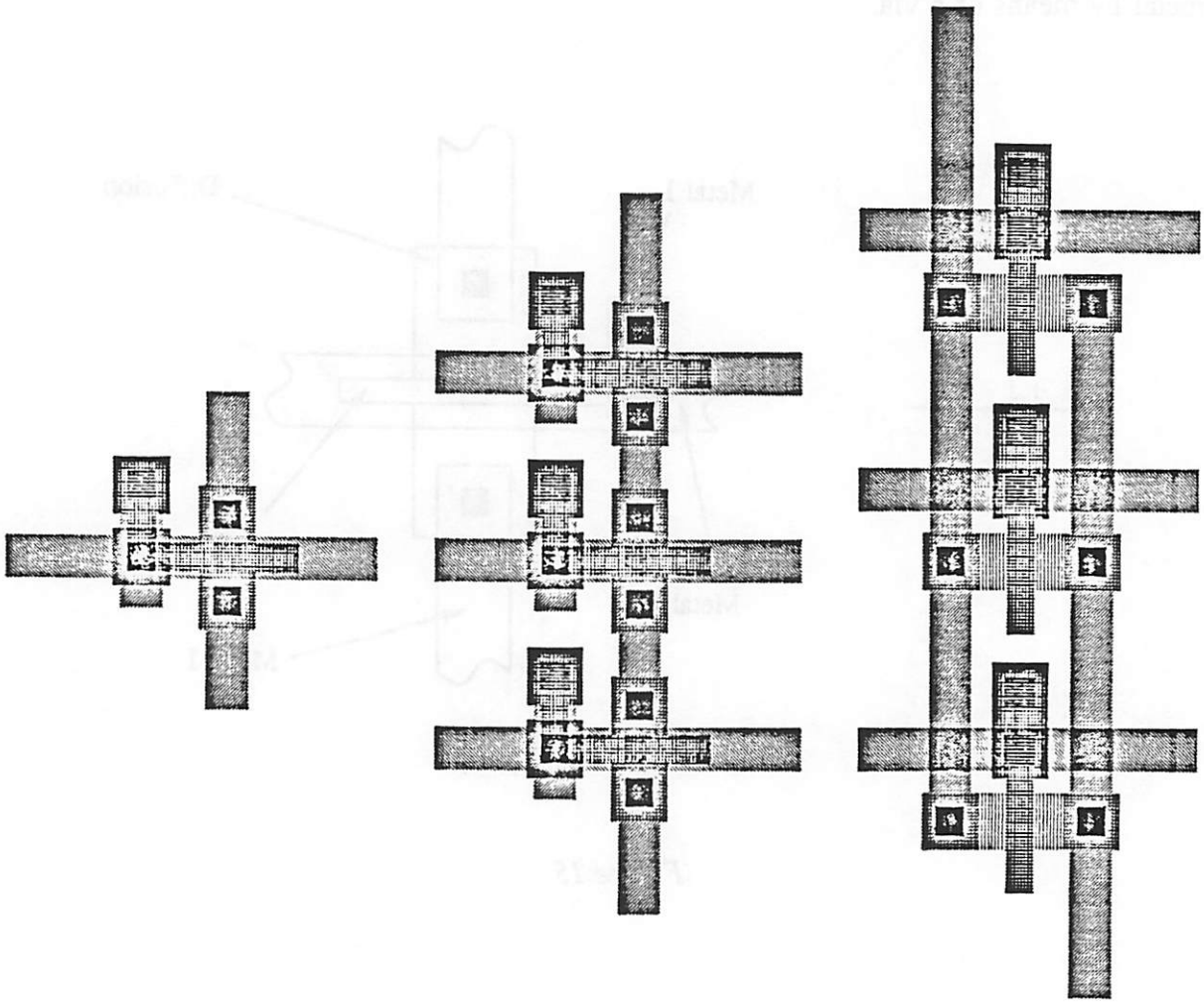


Figure 16

The layout of an ADDER, shown in figure 17, is realized by using the 2FBA architecture.

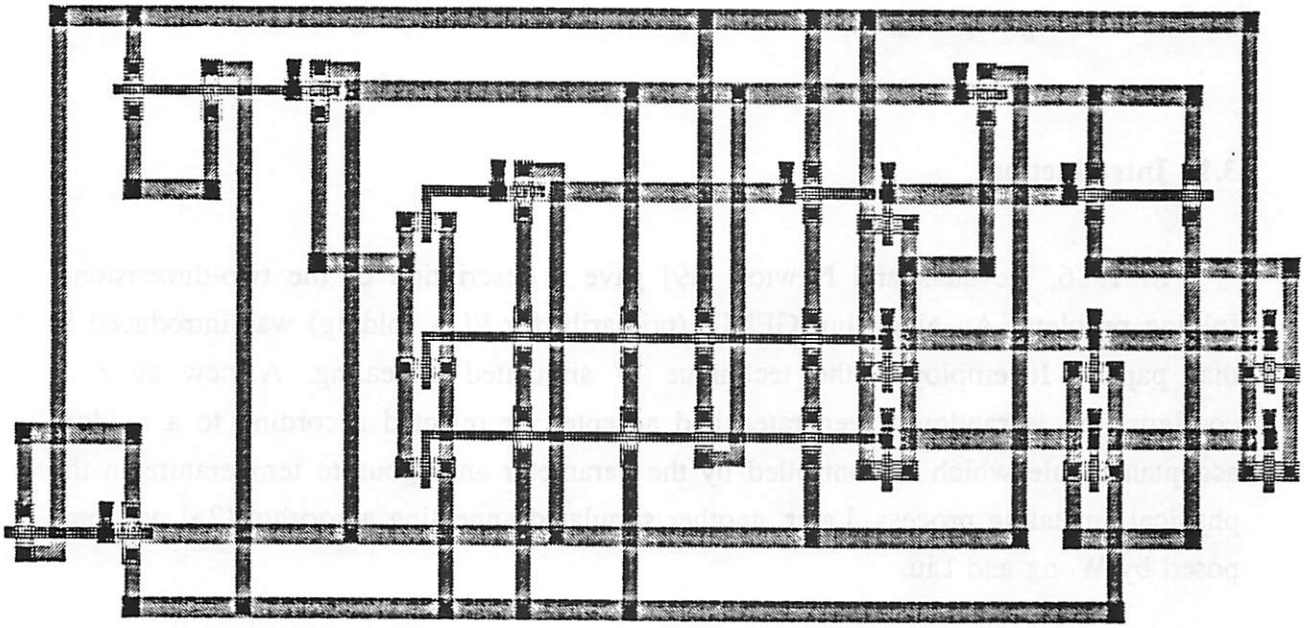


Figure 17

Chapter 3

3. 2-D Folding Algorithm:

3.1. Introduction:

In 1986, Devadas and Newton [19] gave a description of the two-dimensional folding problem. An algorithm GENIE (primarily for PLA folding) was introduced in that paper. It employed the technique of simulated annealing. A new state or configuration is randomly generated and accepted or rejected according to a random acceptance rule which is controlled by the parameter analogous to temperature in the physical annealing process. Later, another simulated annealing algorithm [25] was proposed by Wong and Liu.

A naive solution to the two-dimensional folding problem, is to divide it into two 1-dimensional assignment problems. For example, we first rearrange the ordering of vertical nets so as to minimize the number of horizontal tracks and the length of horizontal nets. After the horizontal track assignment, the ordering of horizontal tracks can also be changed to minimize the number of vertical tracks and the length of vertical nets. Because the ordering of vertical nets has been fixed at this time, the vertical track assignment is extremely constrained. So this naive method can not succeed in general.

In this chapter, we propose a new algorithm for the two-dimensional folding problem. Instead of using simulated annealing technology, we use a deterministic strategy to solve this problem. Usually, when a horizontal (or vertical) net is moved to a new position, the length of the attached vertical (or horizontal) nets will be changed. If the length of those attached vertical (or horizontal) nets increases, the move is called a negative move. If it decreases, the move is called a positive move. For each negative move of a horizontal (or vertical) net, we set an upper bound to control the increase of

the length of the attached vertical (or horizontal) nets. The length increase is not allowed to be more than this upper bound.

For a set of nets, the algorithm attempts to find a number of positive moves. If those moves result in an improvement to the total length of the nets, the moves are accepted. If not, the algorithm will try to find another set of moves which include both positive and negative moves. The algorithm allows the length of the nets to increase subject to a chosen upper bound. This process continues until the objective function can not be improved any further.

We illustrate the algorithm by means of the curves from an example in [25]. The curves consist of the number of moves from 1219 to 11498. For each move, we mark the total length of nets and the number of tracks in Y coordinates. The results tell us that our algorithm, like simulated annealing, has the property of climbing out a local optimum. Unlike the simulated annealing approach, it requires much less computing time. We have tested some published examples. The results are shown later.

3.2. Two Dimensional Folding Problem:

An unfolded two-dimensional example is illustrated in figure 18(a). For simplicity, we assume that all the cells are of the same size. In figure 18(a), the cells are connected by a set of horizontal and vertical nets. Each net consists of a number of cells. We are supposed to rearrange the horizontal and vertical nets without disturbing the cell connectivity. This is accomplished by folding horizontal and vertical nets for the purpose of minimizing the area usage. Figure 18(b) is the folded result of figure 18(a). In addition, the two-dimensional folding problem attempts to achieve a specific aspect ratio of the resulting rectangle which encloses the cells.

3.3. Definitions:

Let H = { h_1, h_2, ..., h_m } denote a set of horizontal nets, and V = { v_1, v_2, ..., v_n } a set of vertical nets. Let h_i^k denote a horizontal net h_i which is on the horizontal track

k and, similarly, let v_j^l denote a vertical net v_j on vertical track l. Let $c_{k,l}$ denote a cell

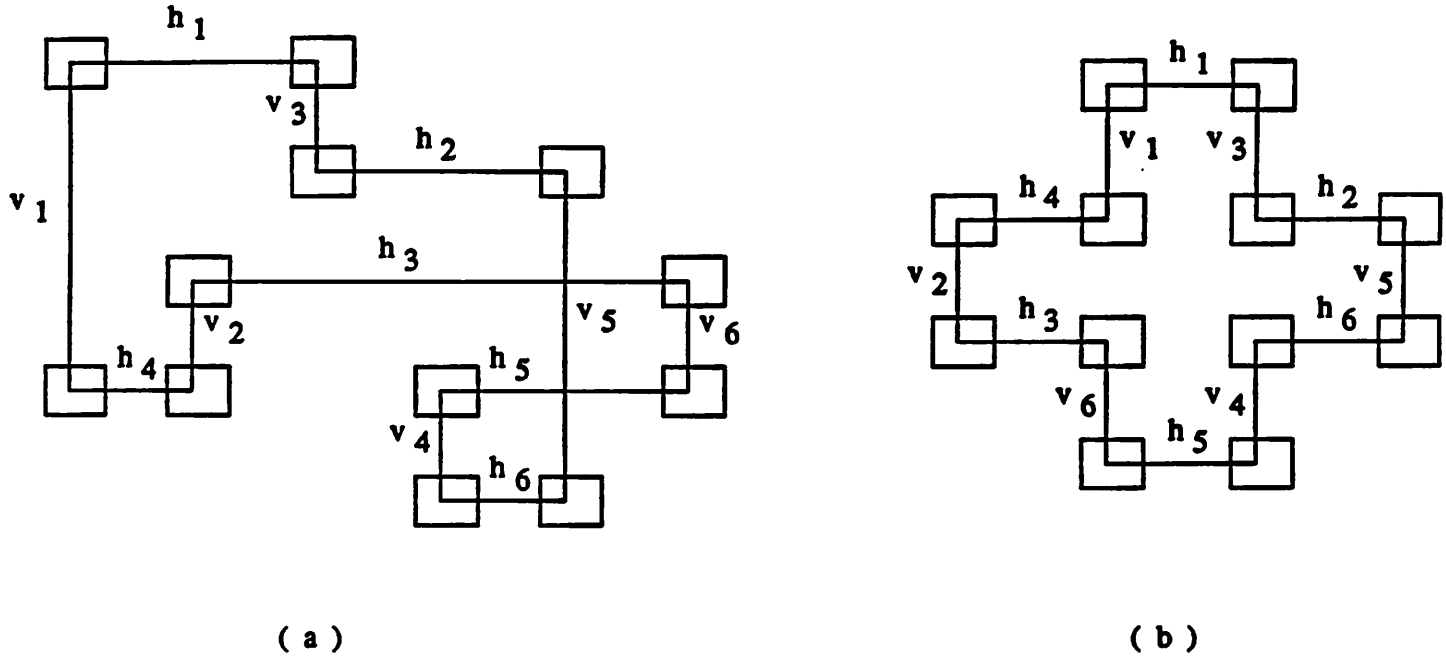


Figure 18

at the cross point of horizontal track k and vertical track l (see figure 19). A net can be described by a set of cells which are connected to the net. Thus,

$$h_i^k = \{\ c_{k,l} \mid_{\text{for all } l,\ c_{k,l} \text{ connects with net } h_i^k}\ \}$$

$$v_j^l = \{\ c_{k,l} \mid_{\text{for all } k,\ c_{k,l} \text{ connects with net } v_j^l}\ \}$$

An interval of a horizontal net h_i^k can be defined as:

$$I(h_i^k) = [\ \min\{\ l \mid_{c_{k,l} \in h_i^k}\ \},\ \max\{\ l \mid_{c_{k,l} \in h_i^k}\ \}\ ]$$

Similarly, for a vertical net,

$$I(v_j^l) = [\ \min\{\ k \mid_{c_{k,l} \in v_j^l}\ \},\ \max\{\ k \mid_{c_{k,l} \in v_j^l}\ \}\ ]$$

The length of a horizontal or vertical net can be defined as,

$$|I(h_i^k)| = \max \{ 1 \mid_{c_{k,j} \in h_i^k} \} - \min \{ 1 \mid_{c_{k,j} \in h_i^k} \}$$

$$|I(v_j^l)| = \max \{ k \mid_{c_{k,j} \in v_j^l} \} - \min \{ k \mid_{c_{k,j} \in v_j^l} \}$$

For a set of nets A = { a_1, a_2, ..., a_p }, the total length of those nets is thus:

$$|I(A)| = \sum_{i=1}^{p} |I(a_i)|$$

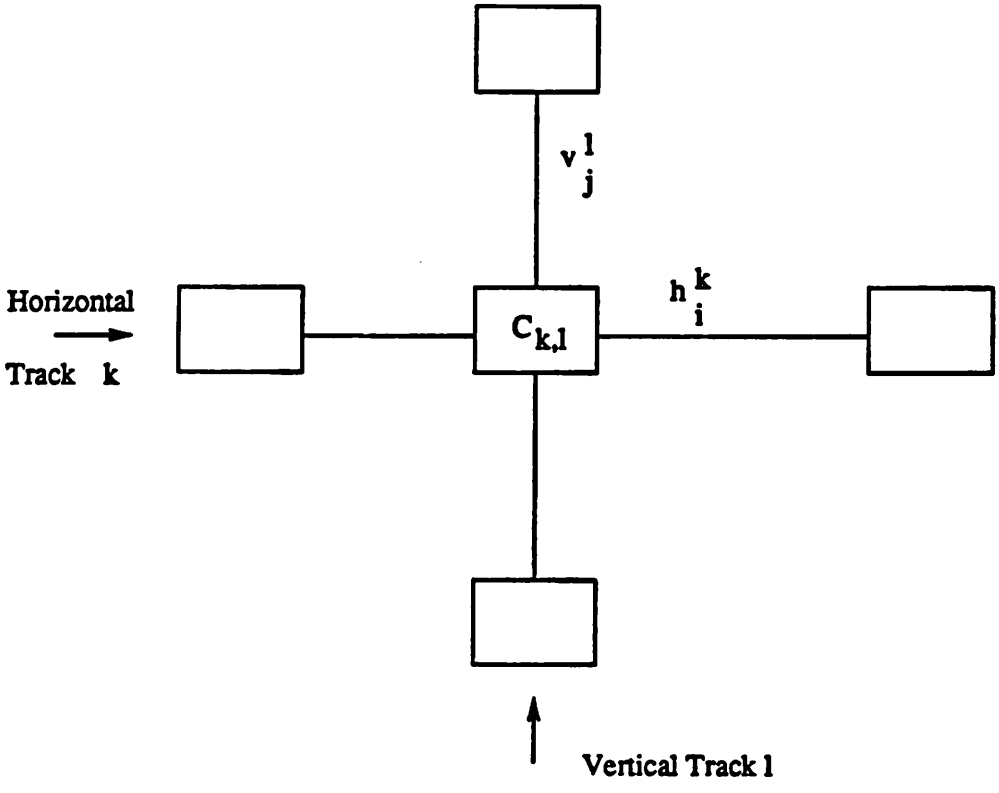


Figure 19

If two horizontal (or vertical) nets $h_{i_1}^k$ and $h_{i_2}^k$ (or $v_{j_1}^l$ and $v_{j_2}^l$) are on the same track, they are described as "overlapping" if and only if the interval $I(h_{i_1}^k)$ overlaps with the interval $I(h_{i_2}^k)$.

For the example in figure 18(a), H = { h_1, h_2, h_3, h_4, h_5, h_6 }, V = { v_1, v_2, v_3, v_4, v_5, v_6 }, and the set of cells is { $c_{1,1}$, $c_{1,3}$, $c_{2,3}$, $c_{2,5}$, $c_{3,2}$, $c_{3,6}$, $c_{4,1}$, $c_{4,2}$, $c_{4,4}$, $c_{4,6}$, $c_{5,4}$, $c_{5,5}$ }. The above definitions can be illustrated by using net h_4, h_5, v_1 and v_2.

According to the definition, net h_4, h_5, v_1 and v_2 can be represented as,

$$h_4^4 = \{ c_{4,1}, c_{4,2} \} \qquad h_5^4 = \{ c_{4,4}, c_{4,6} \}$$
$$v_1^1 = \{ c_{1,1}, c_{4,1} \} \qquad v_2^2 = \{ c_{3,2}, c_{4,2} \}$$

The intervals of these nets are,

$$I(h_4^4) = [\ \min(1, 2), \max(1, 2)\ ] = [1, 2]$$
$$I(h_5^4) = [\ \min(4, 6), \max(4, 6)\ ] = [4, 6]$$
$$I(v_1^1) = [\ \min(1, 4), \max(1, 4)\ ] = [1, 4]$$
$$I(v_2^2) = [\ \min(3, 4), \max(3, 4)\ ] = [3, 4]$$

Assume A = { h_4^4, h_5^4, v_1^1, v_2^2 }, we obtain

$$|I(A)| = |I(h_4^4)| + |I(h_5^4)| + |I(v_1^1)| + |I(v_2^2)|$$
$$= (2 - 1) + (6 - 4) + (4 - 1) + (4 - 3) = 7$$

The horizontal nets h_4^4 and h_5^4 are on the same track, as shown in figure 18(a). Their intervals [1, 2] and [4, 6] do not overlap. So, net h_4^4 does not overlap with net h_5^4.

If a horizontal net h_i^k connects with a vertical net v_j^l , $h_i^k \cap v_j^l \neq \phi$, the vertical (or horizontal) net v_j^l (or h_i^k) is said to be related to the horizontal net h_i^k (or v_j^l). In figure 18(a), the horizontal net h_4^4 connects with the vertical net v_2^2 at cell $c_{4,2}$. Thus, we get

$$h_4^4 \cap v_2^2 = \{ c_{4,1}, c_{4,2} \} \cap \{ c_{3,2}, c_{4,2} \} = \{ c_{4,2} \} \neq \phi$$

The vertical net v_2^2 is said to be related to net h_4^4 and vice versa.

For a horizontal (or vertical) net h_i^k (or v_j^l), all the related vertical (or horizontal) nets can be represented by

$$R(h_i^k) = \{\ v_j^l \mid_{\text{for all } v_j^l,\ h_i^k \cap v_j^l \neq \phi}\ \}$$

$$R(v_j^l) = \{\ h_i^k \mid_{\text{for all } h_i^k,\ v_j^l \cap h_i^k \neq \phi}\ \}$$

In figure 18(a), $R(h_4^4) = \{\ v_1^1, v_2^2\ \}$.

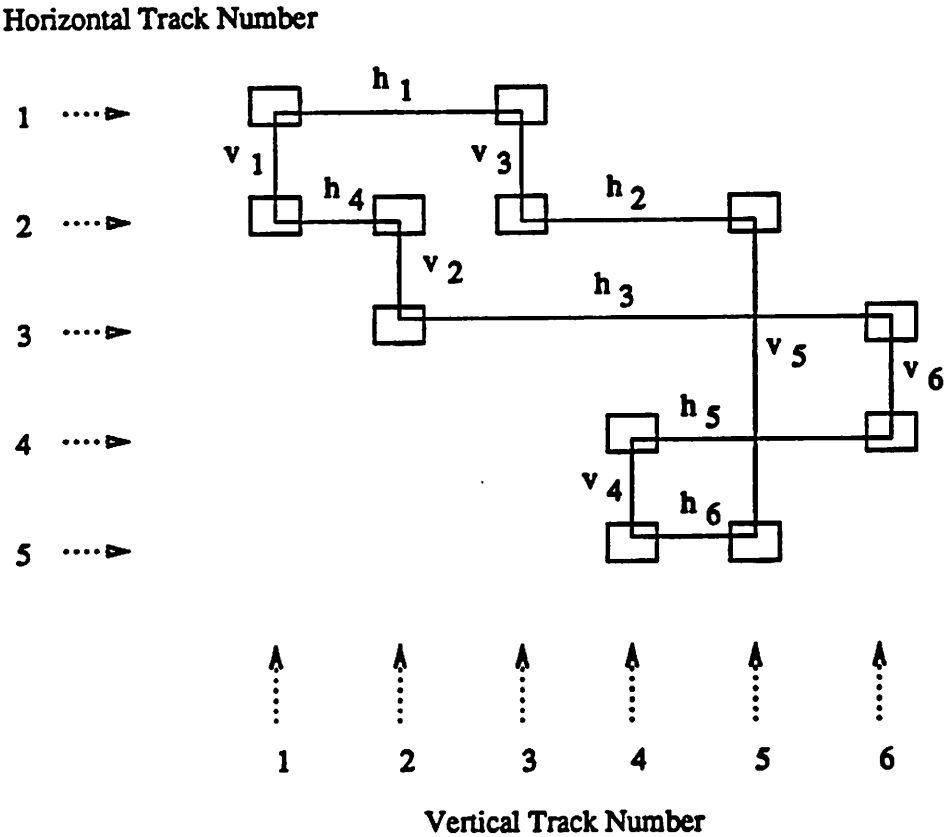


Figure 20

If a horizontal net $h_i^{k_1}$ is moved from track k_1 to track k_2, all the related vertical nets in $R(h_i^{k_1})$ will be changed. Thus, the length of those related nets may be changed. If a horizontal net $h_i^{k_1}$ is moved from track k_1 to track k_2, the change of the total length of its related nets is defined as follows:

$$| \Delta R(h_i^{k_1}) |_{k_1}^{k_2} = |I(R(h_i^{k_2}))| - |I(R(h_i^{k_1}))|$$

If $| \Delta R(h_i^{k_1}) |_{k_1}^{k_2} \leq 0$, the move is called a positive move. If it is greater than 0, it is said to be a negative move.

For the example in figure 18(a), if the horizontal net h_4^4 is moved from track 4 to track 2 (see figure 20), we get

$$\begin{aligned} | \Delta R(h_4^4) |_4^2 &= |I(R(h_4^2))| - |I(R(h_4^4))| \\ &= [(2 - 1) + (3 - 2)] - [(4 - 1) + (4 - 3)] = -2 \end{aligned}$$

Thus, the move is positive. It means the move reduces the total length of the related vertical nets by 2.

Similarly,

$$| \Delta R(v_j^{k_1}) |_{k_1}^{k_2} = |I(R(v_j^{k_2}))| - |I(R(v_j^{k_1}))|$$

3.4. Algorithm:

We use the wire length as the objective function for minimization. The number of tracks, as a subordinate objective, is not allowed to increase during the minimization. This is because we may have many opportunities to put more nets on a track if the lengths of those nets are short. This strategy can obtain better results than just concentrating on minimizing the number of tracks.

The algorithm is divided into horizontal phases and vertical phases. In the horizontal phase, a set of horizontal nets is moved, while in the vertical phase, a set of vertical nets is moved. A move of a horizontal (or vertical) net $h_i^{k_1}$ (or $v_j^{k_1}$) from track k_1 to track k_2 is accepted if the following conditions are satisfied:

(1) $h_i^{k_1}$ (or $v_j^{k_1}$) does not overlap with the horizontal (or vertical) nets on track k_2;

(2) after the move, all the related vertical (or horizontal) nets in $R(h_i^{k_1})$ (or $R(v_j^{k_1})$) do not overlap with the other vertical (or horizontal) nets; and

(3) $|\Delta R(h_i^{k_1})|_{k_1}^{k_2} \le C$ (or $|\Delta R(v_j^{k_1})|_{k_1}^{k_2} \le C$), $C \ge 0$.

By comparing the current and the required aspect ratios of a cell, the algorithm can decide which phase is going to be the next one. If the cell is wide, it enters the vertical phase and tries to reduce the length of horizontal nets and the number of vertical tracks. If the cell is slender it will pick the horizontal phase and attempt to reduce the length of vertical nets and the number of horizontal tracks. The algorithm will keep choosing the horizontal or vertical phase to fold the given problem, until the result cannot be improved.

Let $H = \{ h_1, h_2, ..., h_m \}$ be the set of all horizontal nets and $V = \{ v_1, v_2, ..., v_n \}$ be the set of all vertical nets. In a horizontal (or vertical) phase, we select a set of horizontal (or vertical) nets $P \subseteq H$ (or $Q \subseteq V$) and attempt to reduce the length of vertical (or horizontal) nets and the number of horizontal (or vertical) tracks by moving these nets to better positions. If the length of the nets cannot be reduced, some negative moves are then accepted.

$$P = \{ p_1, p_2, ..., p_r \} \quad (r \le m)$$
$$Q = \{ q_1, q_2, ..., q_s \} \quad (s \le n)$$

For a horizontal (or vertical) net $p_i \in P$ $(1 \le i \le r)$ (or $q_i \in Q$ $(1 \le i \le s)$), a number of moves can satisfy the above three conditions. In the algorithm, we choose one of the moves arbitrarily. After all the nets in P (or Q) are moved, we calculate the change in length of the nets:

$$AR = \sum_{p_i^{k_1} \in P} |\Delta R(p_i^{k_1})|_{k_1}^{k_2}$$

$$\left(\text{ or } AR = \sum_{q_i^{k_1} \in Q} |\Delta R(q_i^{k_1})|_{k_1}^{k_2} \right)$$

If $AR \le C$ ($C \ge 0$, at the beginning of a phase, C is set to be 0), this set of moves is accepted. Otherwise, we increase C slightly and try another set of moves for these

nets. This means that some negative moves are allowed. If the moves are still rejected, we increase C again, until C reaches an upper bound UB. In the case of C = UB, we consider this phase a failure. After several continuous failed phases, the algorithm stops.

```
Algorithm 2D_FOLDING
 {
        do {
                get required aspect ratio: reqAspectRatio;
                calculate actual aspect ratio: realAspectRatio;
                if ( realAspectRatio >= reqAspectRatio )
                        Phase( vertical );
                else
                        Phase( horizontal );
        } while ( movesAccepted );
 }

 Phase( P ) {
 {
        C = 0;
        do {
                AR = 0;
                for ( p_i ε P ) {
                        if ( |ΔR(p_i^{k1})|_{k1}^{k2} <= C ) {
                                move p_i ε P from track k1 to k2;
                                AR += |ΔR(p_i^{k1})|_{k1}^{k2};
                        }
                } /* for */
                if ( C++ > UB ) return( movesRejected );
        } while ( AR > C )
        return( movesAccepted );
 }
```

Figure 21 shows the searching approach of this algorithm. In figure 21, the X and Y axes indicate the move steps and the total length of nets, respectively. The solid dot at the left side indicates the total length of nets at step k.

We define the number of moves in each phase as the phase size. For a set of moves in a phase, a number of nets are selected to be moved. At the beginning, C is

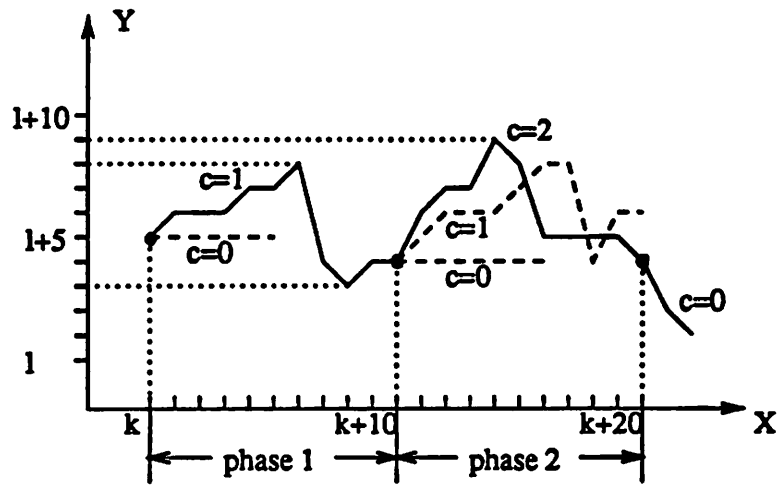


Figure 21

set to be 0. This means that each move of the selected nets should result in a decrease of either the total length of nets or no change. Comparing the ending point with the beginning point of the phase, if the length increase of the nets is less than C, this phase is completed. Otherwise, the phase is not completed, and C will then be set to 1. Thus, the total length of nets is allowed to be increased by 1 at each step of the move. After a set of moves, the algorithm will compare the last point of the phase with the beginning point of the phase again. If the length increase is not less than C, this phase is still not completed. we increase C to 2. The algorithm will continue until either C reaches UB or the phase is completed. If C reaches UB, we consider this phase a failure. After several continuous failed phases, the algorithm stops. In figure 21, the solid curve shows a set of possible moves in two phases. The broken lines in each phase indicate that the moves are not accepted. So, phase 1 is not completed when

C=0. And, phase 2 is not completed when C=0 and C=1. Phase 1 and phase 2 are completed when C equals 1 and 2, respectively.

According to the above description, we know that the algorithm is going to find a set of positive moves at the beginning of a phase. The positive moves are greedy. However, if the number of positive moves is less than the phase size (i.e., the total number of moves specified), the algorithm will not give up. Instead, it introduces some negative moves. By choosing moves this way, the algorithm is no longer greedy and the searching approach resembles that of the simulated annealing algorithms.

To illustrate the property of climbing out of a local minimum, we trace all the moves of this algorithm when it is applied to an example in [25]. In this example, the total length of the nets is 1920. The horizontal and vertical tracks are 56 and 81, respectively. Figure 22 shows two curves which are obtained when UB equals 0. Each of the curves consists of 1219 points, and each point corresponds to a move. The upper flat curve (called the track curve) indicates the change of the sum of horizontal and vertical tracks. The lower curve (called the wire length curve) describes the change of the total length of the nets. The algorithm obtained a result with the horizontal tracks, vertical tracks and the length of the nets equal 31, 44, and 974, respectively. Figure 23 shows the curves when UB equals to 2. The curves of UB=4 and UB=8 are shown in figure 24 and figure 25, respectively. The detailed results related to these curves are listed in table 1.

Table 1

UB	horizontal tracks	vertical tracks	wire length	number of moves
0	31	44	974	1219
2	28	36	890	3583
4	26	33	908	8426
8	26	32	882	11498

As shown in figures 22 through 25, the wire length curves become more rugged as the upper bound UB is increased. When UB equals 0, the curve in figure 22 behaves in a greedy way. When UB is greater than 0, the algorithm exhibits a property which is similar to that of the simulated annealing algorithms: it climbs up and down and goes through a lot of local minimums. The algorithm traces all the local minimum

solutions and keeps the minimum one as the final solution. Notice that because the algorithm does not allow the increase of tracks, the track curves keep descending.

A special feature of this algorithm is that it embeds a greedy searching into a global searching strategy. This accounts for its speed, which makes it much more attractive than algorithms based on simulated annealing. In the beginning, the algorithm reduces the total length of nets and the number of tracks very quickly.

In this algorithm, a couple of beginning phases can be completed when C equals 0. As the density of the nets increases, the negative moves are introduced gradually.

Then the result begins to oscillate. From this point on, improvement to the total length

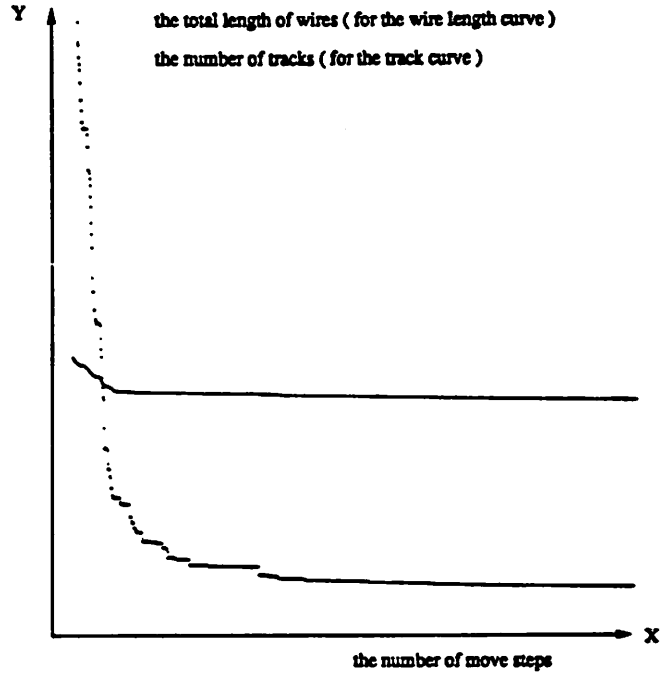


Figure 22

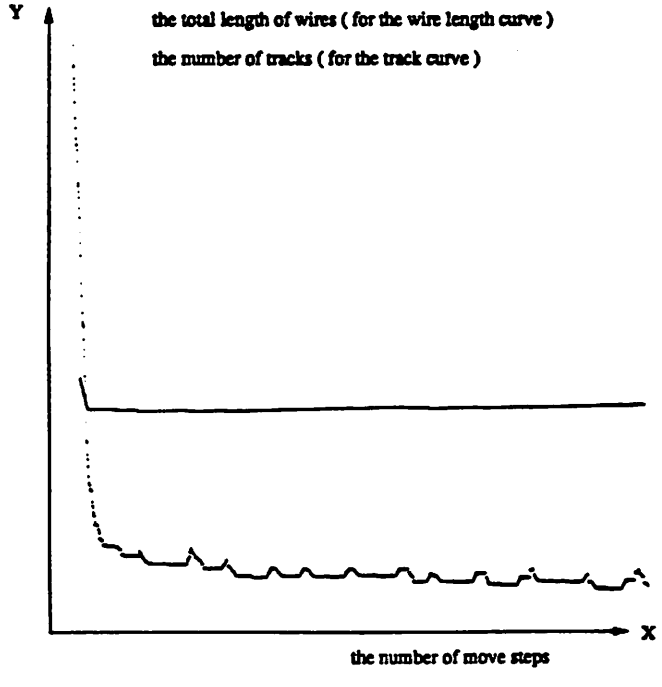


Figure 23

of nets and the number of tracks becomes difficult.

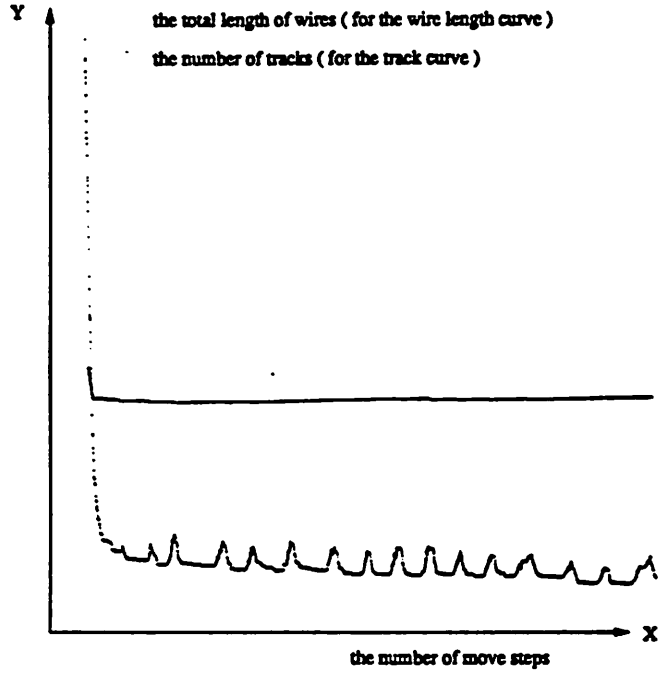


Figure 24

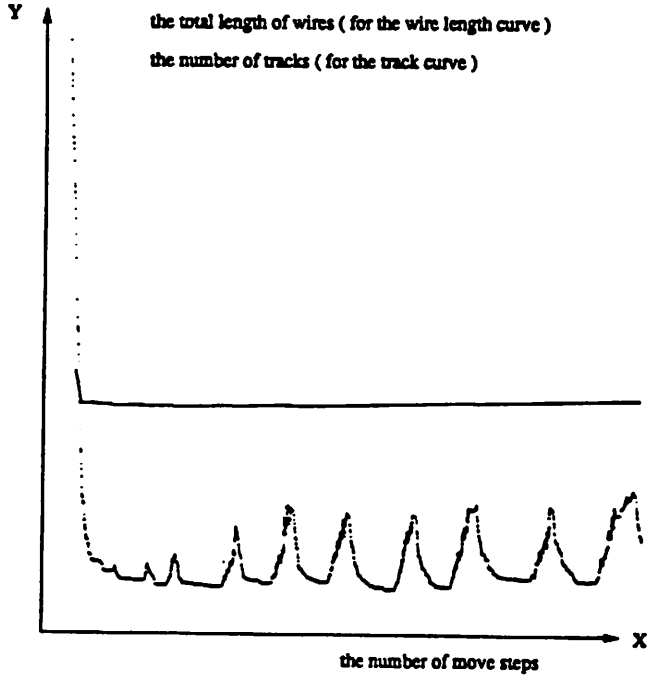


Figure 25

3.5. Conclusion:

The algorithm has been programmed in C and implemented on a DEC 3100. We have tested three examples in [25] and [19]. For all three examples, we set UB to be 8. The results are shown in figure 26, figure 27 and figure 28. The detailed comparison is listed in table 2.

Table 2

Examples	example 1		example 2		example 3	
	[19]	ours	[19]	ours	[25]	ours
horizontal tracks	18	14	32	32	24	25
vertical tracks	19	13	52	50	30	32
time (seconds)	-	3.3	-	4.9	720	66.2

In this chapter, a new algorithm for optimizing the two-dimensional folding problem was proposed. It divides all the moves of nets into a number of phases. The moves in each phase can be either positive or negative. The algorithm prefers the positive moves in each phase, so it improves the solution very fast at the beginning of calculation. Since negative moves are also allowed in each phase, the algorithm has the

property of climbing out of local minimums. It resembles the simulated

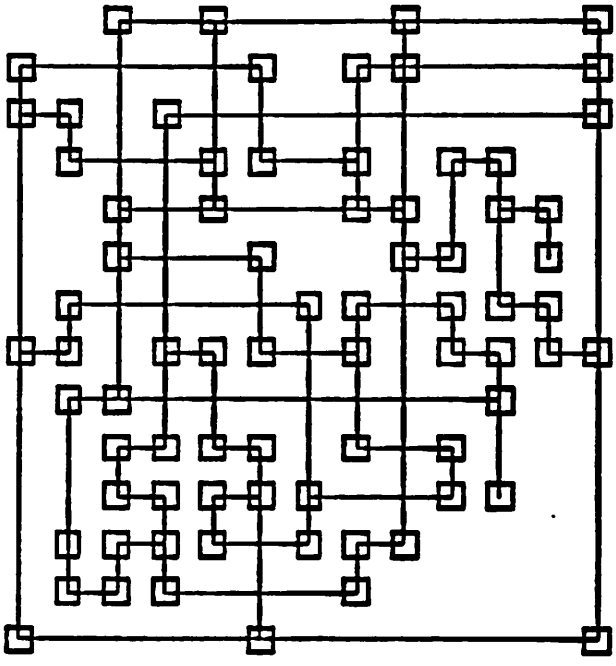


Figure 26

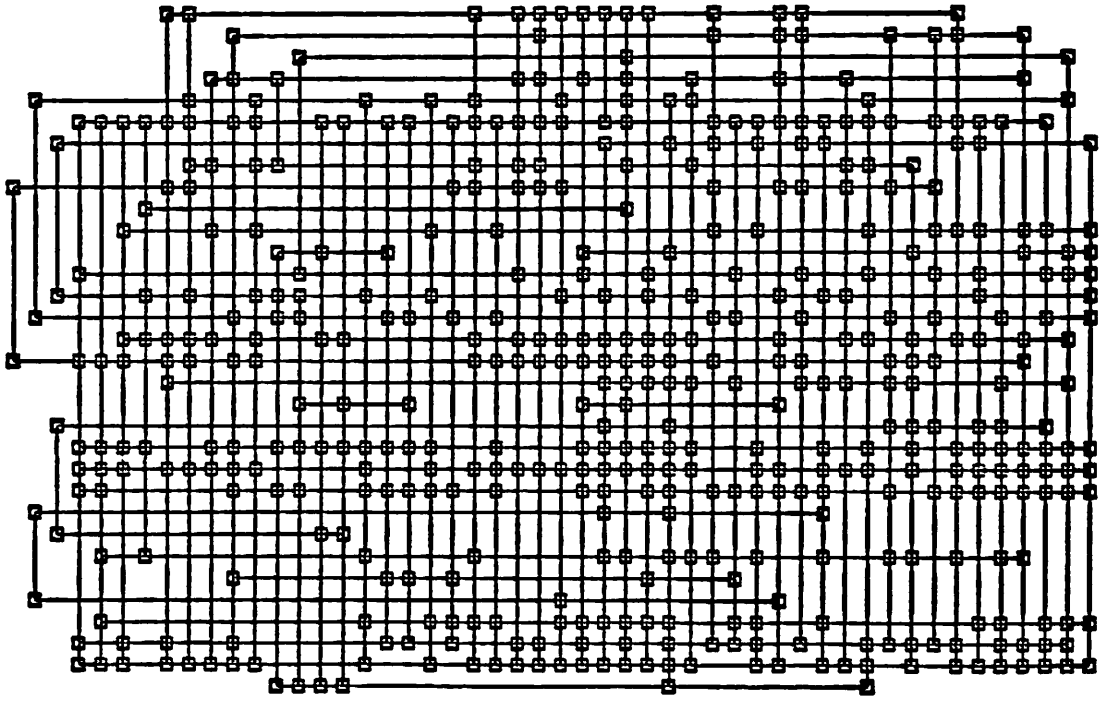


Figure 27

annealing approach in this respect, but, it is much faster.

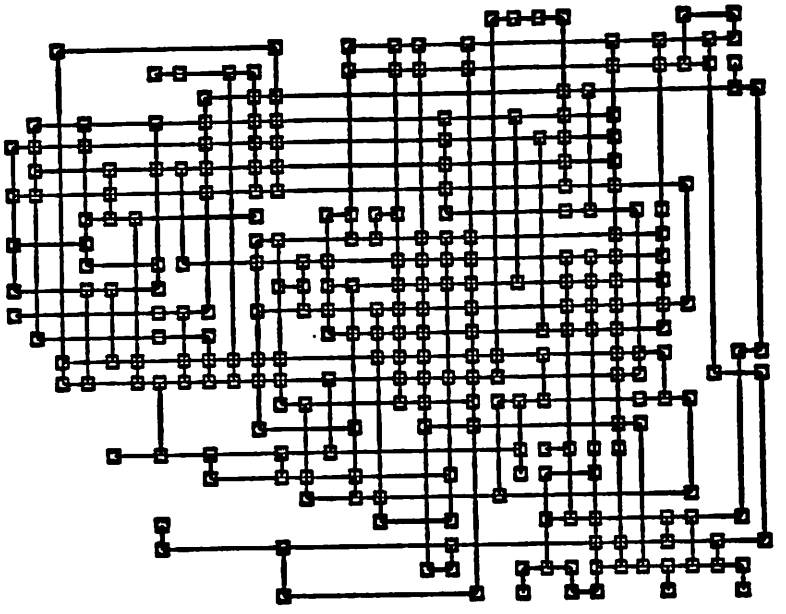


Figure 28

Chapter 4

4. 2DFA Layout Package:

4.1. Introduction:

A module generator program, PASTORALE, has been developed by utilizing the 2DFA architecture. Its inputs are the net lists, which specify the interconnection relations among transistors. Its outputs are CIF files. It can also produce the output files in PostScript description. PASTORALE allows users to specify the time limit in the input files. If the two-dimensional folding algorithm is still doing the iterative improvement when the time limit is reached, PASTORALE will stop the folding processes automatically and begin some post processes.

PASTORALE can accept a positive real number given by the user as the constraint of the aspect ratio. There is no limit to this real number, But, the final aspect ratio is up to the initial architecture and the flexibility of the two-dimensional folding algorithm. The program tries to achieve a layout result with an aspect ratio approaching this given number. It is sometimes difficult to satisfy the aspect ratio constraint precisely.

PASTORALE reads inputs from the given input files. If there is something wrong with the input format, it will report the error type and the error position, and then stop. As we mentioned in chapter 2, the 2DFA layout style has three basic vertical nets: the single transistor net, the serial transistor net and the parallel transistor net. In order to build an initial 2DFA architecture from a given circuit description, PASTORALE should find the serial and parallel connected transistor groups first. The vertical nets are then constructed. The horizontal nets are established by relating each of the nets to a node in the given circuit. The aspect ratio of the initial layout depends on the given circuit schematic.

To reduce the area of the initial layout, the two-dimensional folding algorithm, as discussed in chapter 3, is applied. In chapter 3, the cost function includes the wire length and the number of tracks. In this chapter, a more practical cost function will be used. Based on it, we propose an improved algorithm.

In chapter 3, the algorithm was devoted to move operations only. A move operation is defined as moving a net from one position to another, while maintaining the connectivity. A given two-dimensional folding problem can be completed by using a sequence of the move operations.

In the improved algorithm, we will consider both move operations and swap operations. A swap operation is defined as one which switches the positions of two different nets, while keeping the connectivity unchanged. The improved algorithm achieves better results.

4.2. Improved Algorithm:

4.2.1. Swap Operation:

Considering the example in figure 29(a), we find that no nets can be moved if the number of tracks is not allowed to increase. So the move operation cannot reduce the number of tracks any more. In this example, if we swap the vertical nets v_2 and v_5, figure 29(b) is obtained. By moving up horizontal net h_5, the number of horizontal tracks can be reduced from five to four. This example indicates that better results can be achieved if we take advantage of both the move operation and the swap operation in the algorithm.

Our improved algorithm begins the swap operation after the move operation is done in each phase.

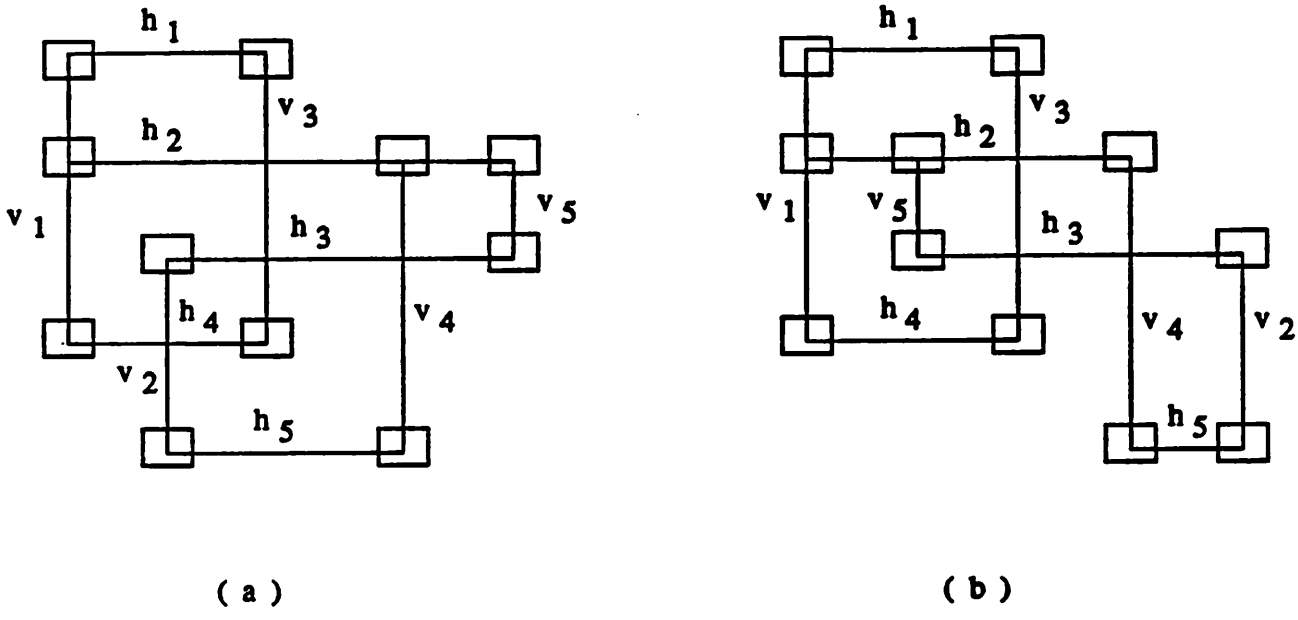


Figure 29

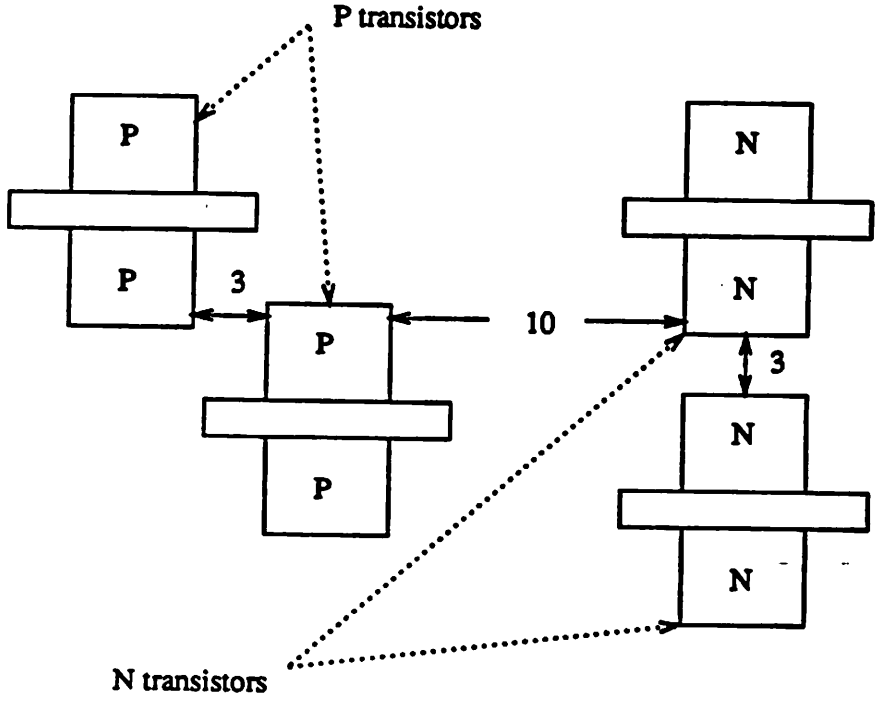


Figure 30

4.2.2. Cost Function:

In chapter 3, the cost function included two factors: the length of nets and the number of tracks. In a real layout, additional factors should be considered.

The 2DFA architecture deals with CMOS circuits. According to the design rules, the space between the same type (P or N) of transistor is much smaller than the space between different types of transistor. In SCMOS technology (a kind of VLSI design rule used at UC Berkeley), the space between the same type of diffusion is 3λ, while the space between different types of diffusion is 10λ. This is shown in figure 30.

To minimize the layout area, it is necessary to cluster the same type of device together, so as to reduce the space consumed by adjacency of different types of transistor. If we cannot avoid the adjacency of different types of transistor, we may take advantage of the space in between by allowing a metal line to go through it, as shown in figure 31.

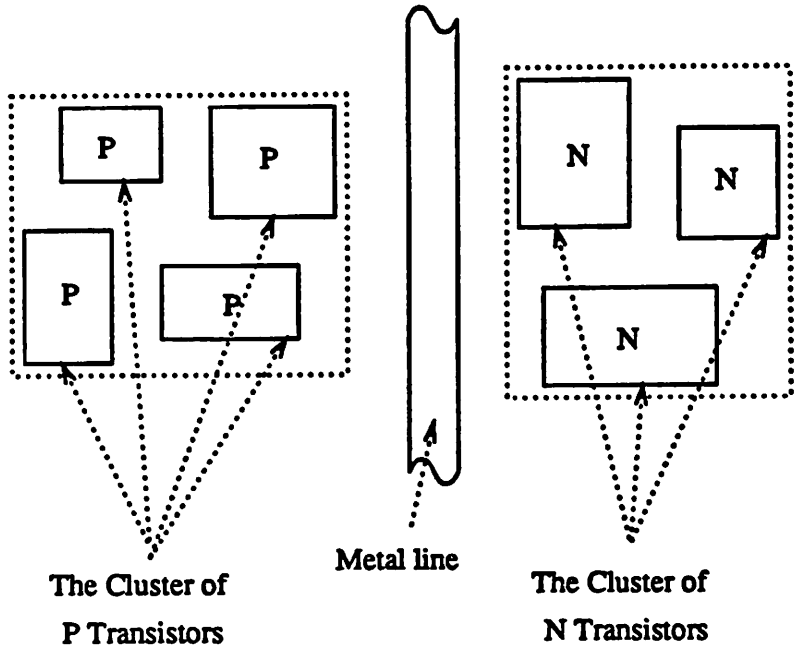


Figure 31

In the cost function, we define *pnPenalty* as the number of adjacent P and N transistors. It should be minimized.

During the layout, the interconnections among transistors are usually implemented by metal lines. To do so, a number of contacts and vias are required. However, when they are close enough, we can sometimes abut the transistors together through poly or diffusion segments, as shown in figure 32.

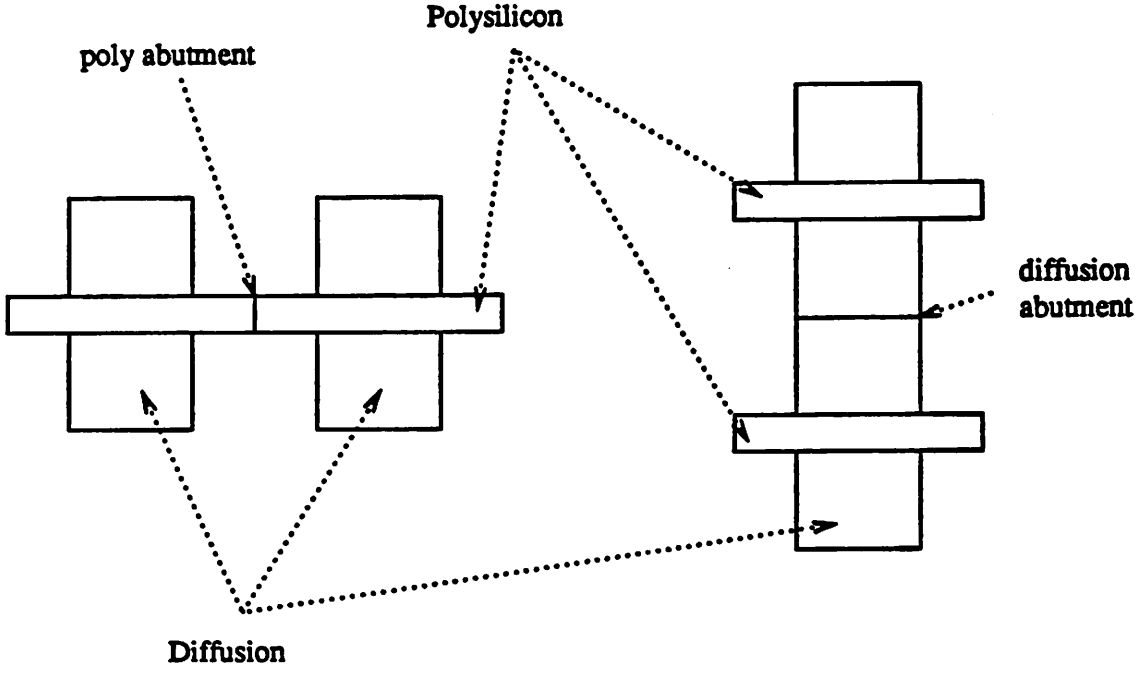


Figure 32

We use *abutGain* to represent the number of abutments in a layout. Thus, *-abutGain* can be considered as the penalty.

As described in chapter 2, three fundamental vertical nets are used to construct a 2DFA layout. Each of the vertical nets connects with the horizontal lines through two vias and one or more transistors. If we put one via on the top, another one on the bottom, and leave all the transistors in the middle, the vertical interconnection of this net can be made by a number of straight lines, instead of the bended lines (see figure 14). So we prefer this kind of permutation with transistors in the middle and vias in the top

and bottom. If one or two vias are put in the middle, a further penalty is generated. We use *midPenalty* to represent the number of bended vertical nets.

The new cost function is defined as follows:

$$cost = k_1 \times netLength + k_2 \times tracks + k_3 \times pnPenalty$$
$$- k_4 \times abutGain + k_5 \times midPenalty$$

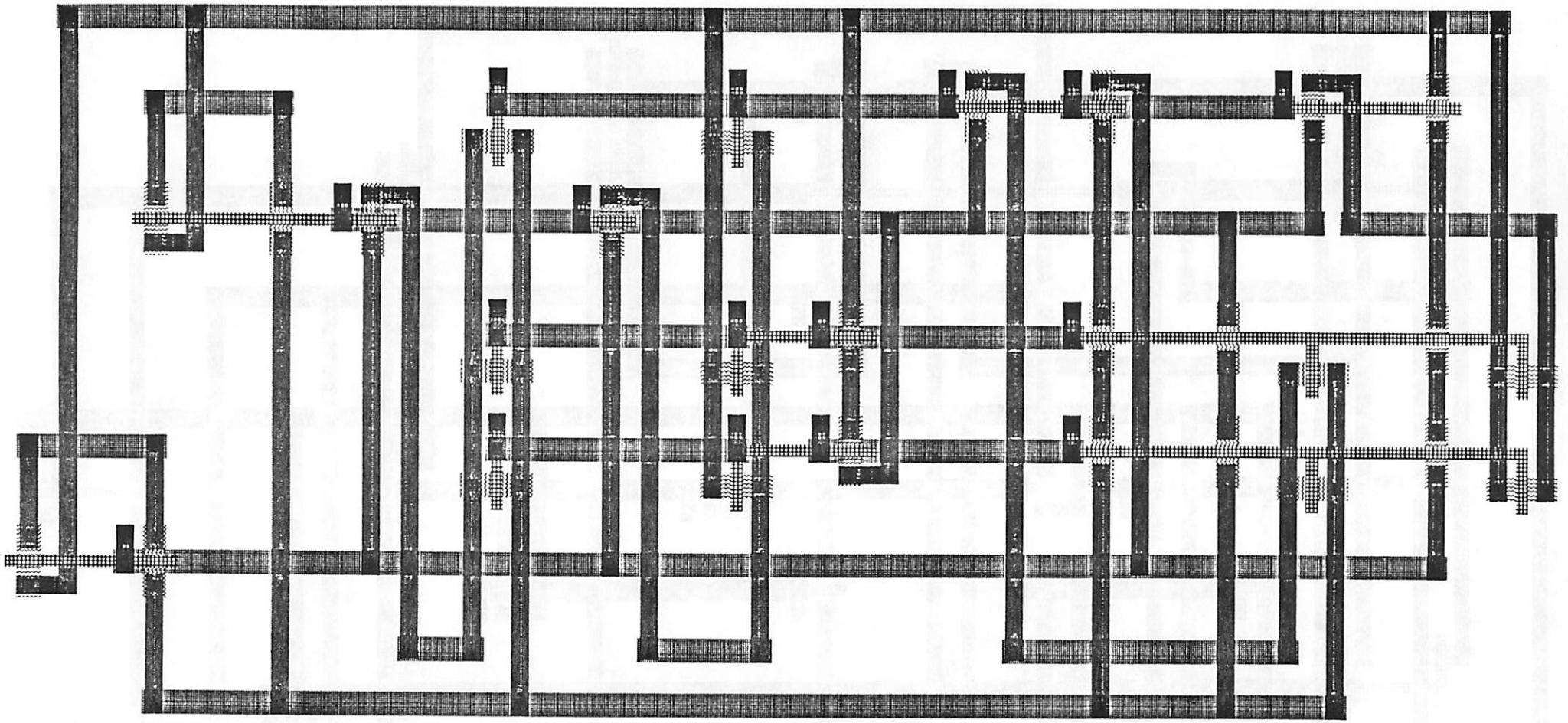
where, *netLength* is the total length of horizontal and vertical nets, and *tracks* is the number of horizontal and vertical tracks.

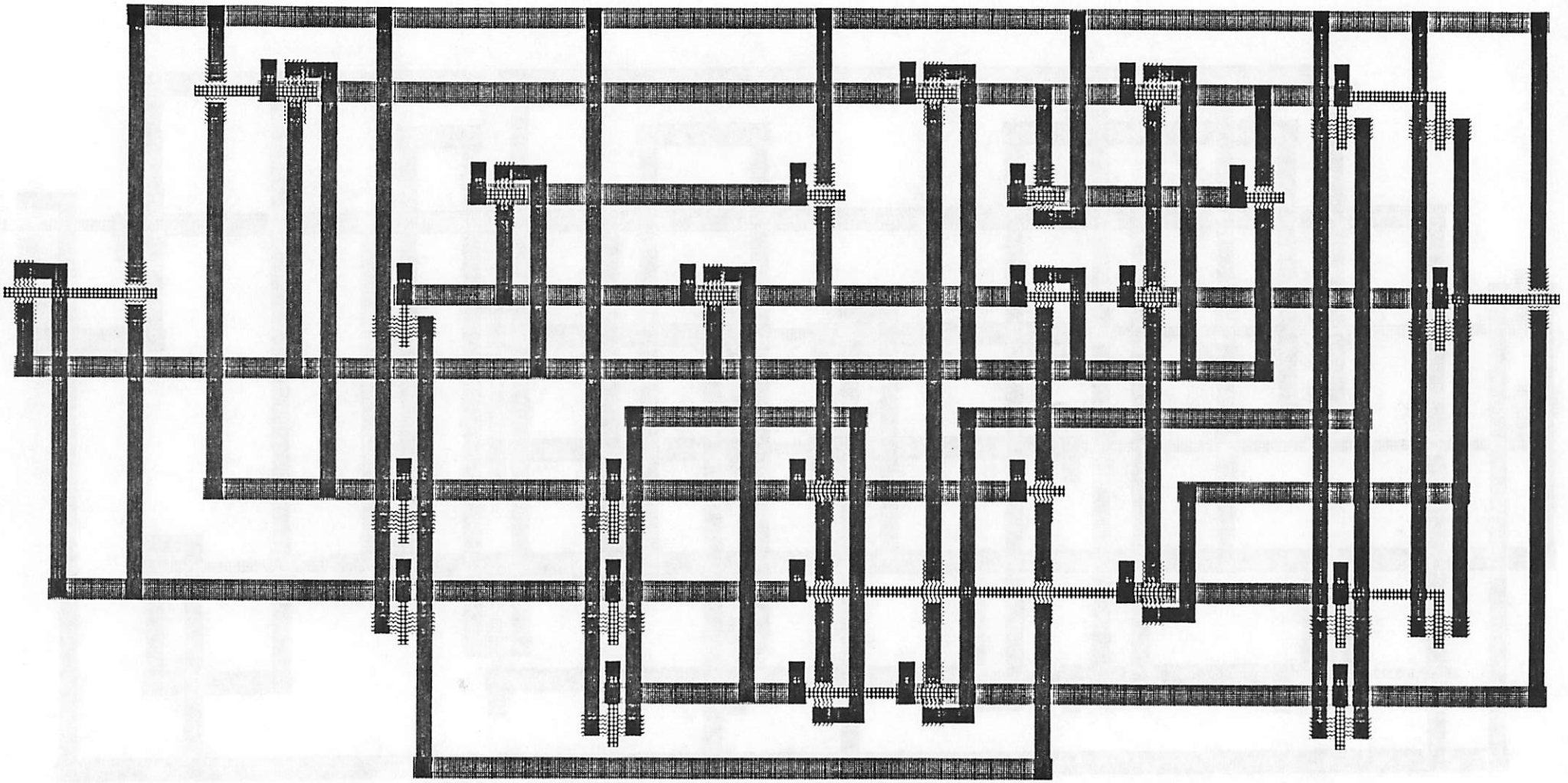
4.3. Results:

PASTORALE has been implemented in C on a DEC 3100. It takes the circuit description at transistor level as its input. Users can put the constraint of the aspect ratio and time limit constraints into the input files. In the final layout, the horizontal and vertical nets are put on equally spaced grids which are separated in a worst case distance according to the design rules. However, unless we apply compaction, the final results are not as good as those of gate matrix layout. Currently, the compaction package is not integrated into PASTORALE.

We will present some results in the following pages.

Adder





flip - flop

Chapter 5

5. Future Work:

5.1. Theoretical Problem:

The two-dimensional folding (or array optimization) problem was proposed several years ago [19]. It has not been solved precisely. Some more work should be done, for example, on a theoretical formulation.

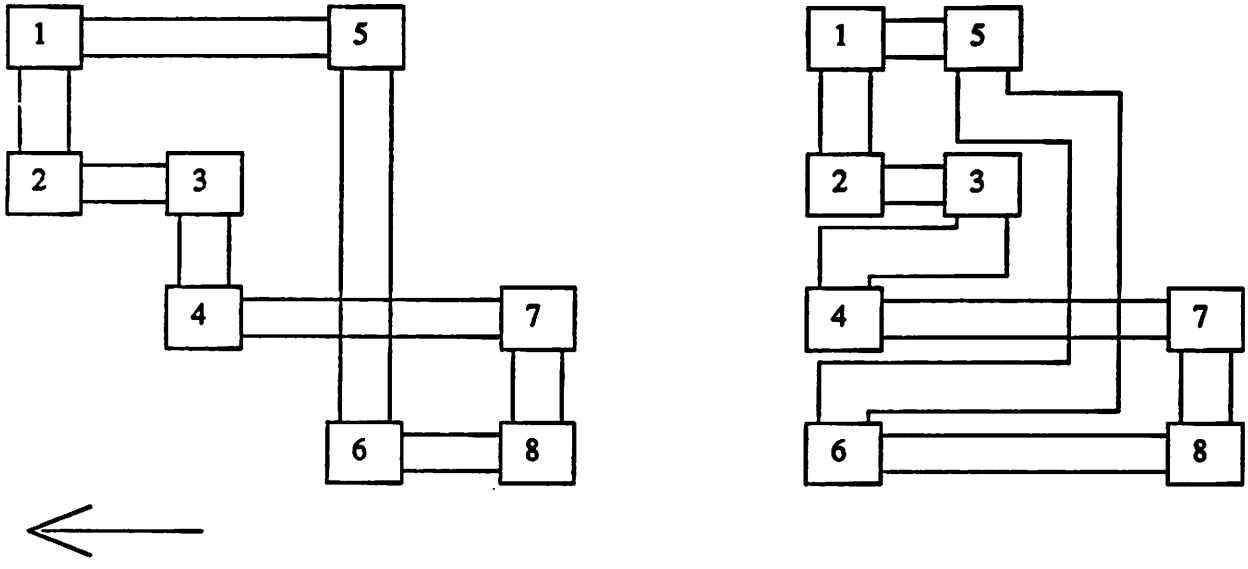
5.2. Practical Constraints:

For a module generator, it should be able to accept the user's specified constraints, such as the position of I/O pins, the aspect ratio, the ordering of I/O pins, etc. In order to optimize the performance of the chip, the timing problem should also be considered. PASTORALE can handle the aspect ratio constraint; The other constraints should be considered.

5.3. Coupling of Folding and Compaction:

In a sense, PASTORALE is still under development. The next version will include the compaction program. According to traditional compaction methodology, the objective of compaction is to minimize the space between adjacent objects, as long as the design rules are not violated. The insertion of jogs may be necessary to achieve a more compact layout. During compaction, the topological structure of the objects is not changed: The objects are moved around without running into one another.

Figure 33(a) gives an example of eight objects together with horizontal and vertical nets. Considering one-dimensional compaction in the horizontal direction, we can shift a number of objects to the left by introducing a couple of jogs. The result of the compaction is illustrated in figure 33(b). The objects 7 and 8 can not be shifted to the left any further.



Compacting Direction

Compacted Result

(a)

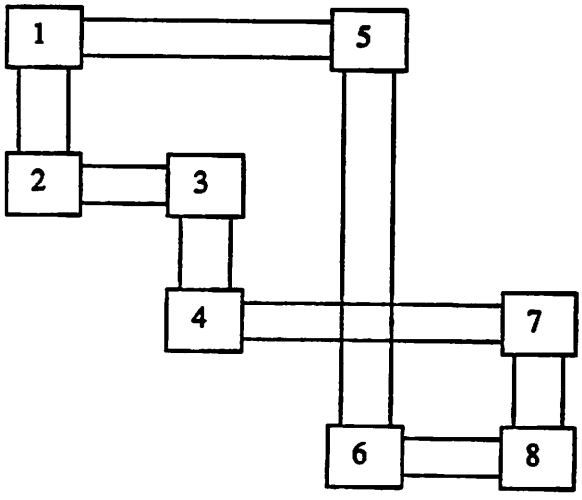
(b)

Figure 33

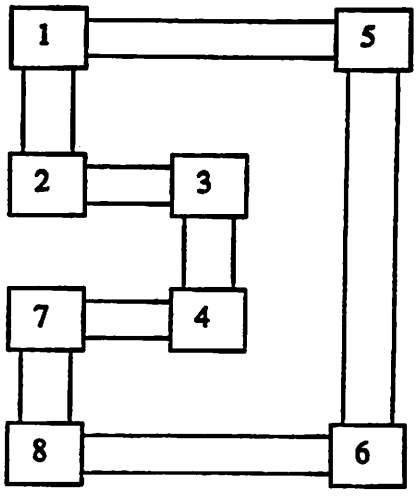
Notice that we are now dealing with an architecture which allows the folding of vertical nets. A new problem emerges: Is it possible to obtain a more compact result by coupling the folding process with the compaction process? Let us take a look at the example in figure 34. Figure 34(a) is the same problem as that in figure 33(a). Before the compaction, we fold the object 7 and 8 to the left side, as shown in figure 34(b). The topological structure is changed. Then we apply compaction to figure 34(b). Finally, a more compact result is obtained (see figure 34(c)).

The above example indicates that a more compact result is achieved if the topological structure is allowed to change during compaction. This represents a more generalized compaction. In 2FBA architecture, this concept can be seen as a combination

of folding and traditional compaction. To handle this problem, an efficient data structure and algorithm need be designed.

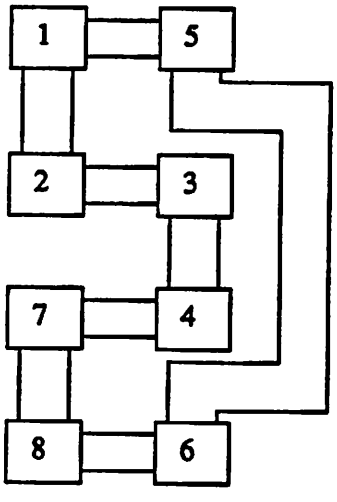


(a)



After Folding

(b)



Compaction

Result of Coupling Folding and Compaction

(c)

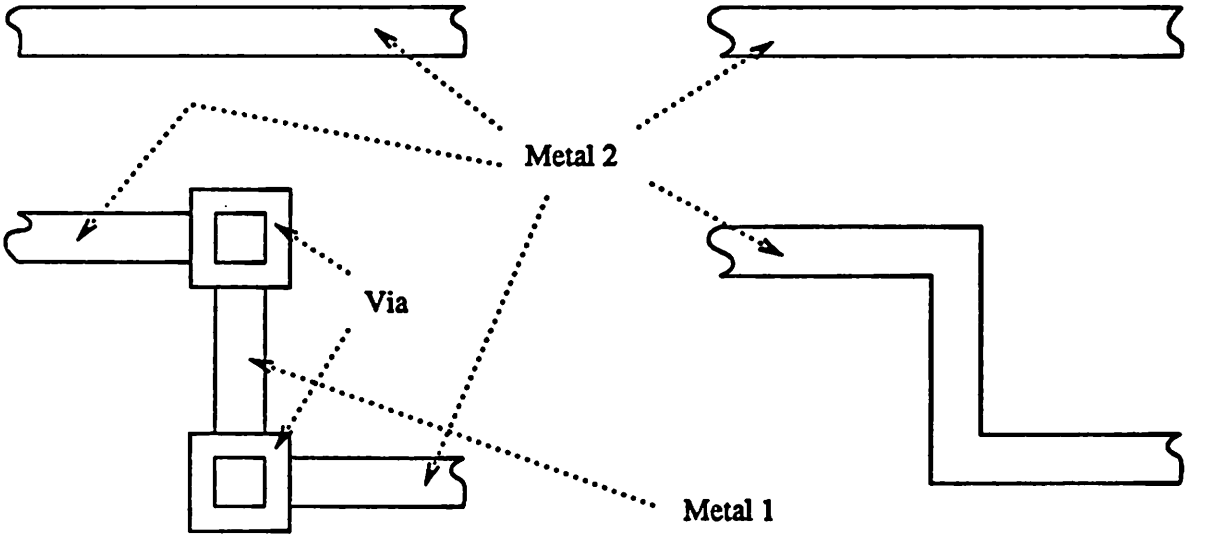
Figure 34

The concept of this generalized compaction can also be extended to the two-dimensional case.

5.4. Net Splitting and Jog Insertion:

According to the way we build the initial 2FBA architecture, the layout usually consists of long horizontal nets and short vertical nets. Since a lot of short vertical nets are connected to a small number of long horizontal nets, it is very difficult to fold the vertical nets. So the aspect ratio cannot be well-controlled.

To handle this problem, we can either split the horizontal nets or insert jogs into the horizontal nets, as shown in figure 35. After the horizontal long nets are removed, better results can be achieved.



Split A Horizontal Net

(a)

Insert Jogs into A Horizontal Net

(b)

Figure 35

5.5. Layer Change and Via Minimization:

In 2FBA architecture, the first metal layer is restricted to the horizontal connections, and the second metal layer to the vertical connections. This rigid rule produces a lot of vias for the interconnections between metal 1 and metal 2. If we allow some horizontal nets on the first metal layer, both the number of tracks and the number of vias may be reduced. In figure 36, the big rectangles represent transistors, and the small rectangles represent vias. In figure 36(a), all the horizontal nets are the second layer metal lines. If we implement net 1, net 2, net 4 and net 7 on the first layer metal lines, the number of vias will be reduced by ten, see figure 36(b). Also, net 7 can be moved up on top of net 5. To do so, the number of tracks is reduced by one.

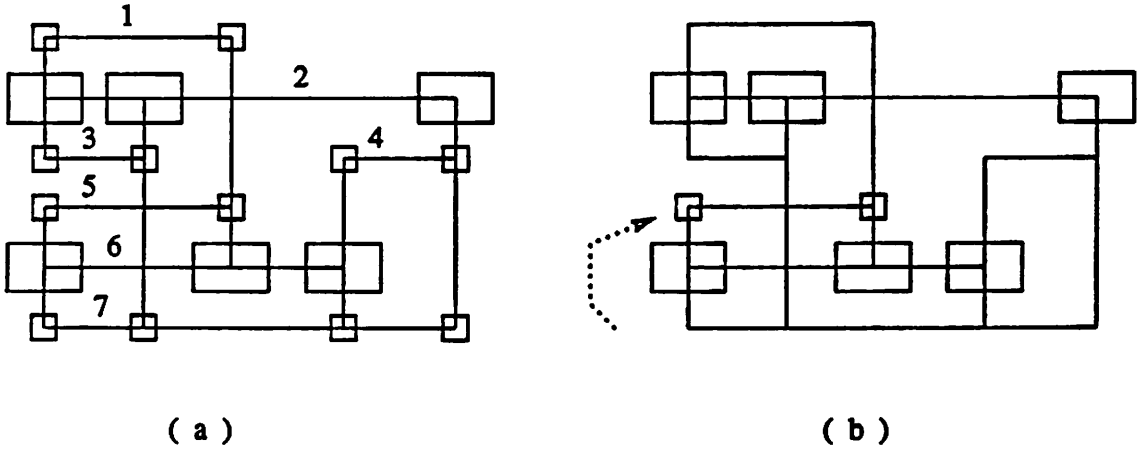


Figure 36

The layer change problem is related to the via minimization problem. In 2FBA layout, the via minimization problem can be formulated as follows: Given a bounding box with some rectangles in it, a set of nets can be accessed from the boundary of both the bounding-box and the rectangles (see figure 37), the objective of the via minimization is to find the interconnection topology and the assignment for all nets so as to minimize the number of vias used.

This is the constrained via minimization problem.

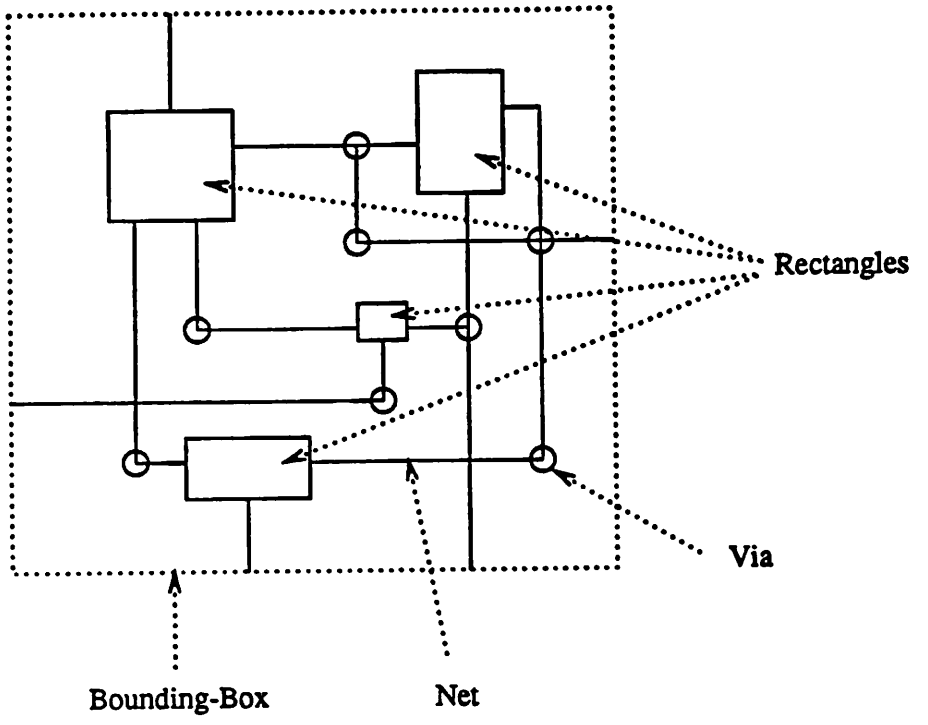


Figure 37

5.6. Improvement on 2FBA architecture:

In 2FBA architecture, a vertical net consists of a couple of transistors. In many cases, a vertical net includes just one transistor, so the initial layouts are not very compact. The area optimization depends heavily on the two-dimensional folding algorithm.

We are now considering an improved architecture which may produce a more compact layout. The basic architecture is illustrated in figure 38. In figure 38, the fundamental components are a number of gate matrices, which are represented by the rectangles. The PG nets on top and bottom can be accessed by each gate matrix. The broken lines in each gate matrix represent polysilicon stripes. These polysilicon stripes can be connected through the second layer metal lines (see the horizontal solid lines). The interconnections in each gate matrix can be implemented by first layer metal lines and short diffusion segments. Since the gate matrix layout style is very efficient for

small circuits, this new architecture may result in more compact layouts.

The two-dimensional folding algorithm can be applied to this new architecture.

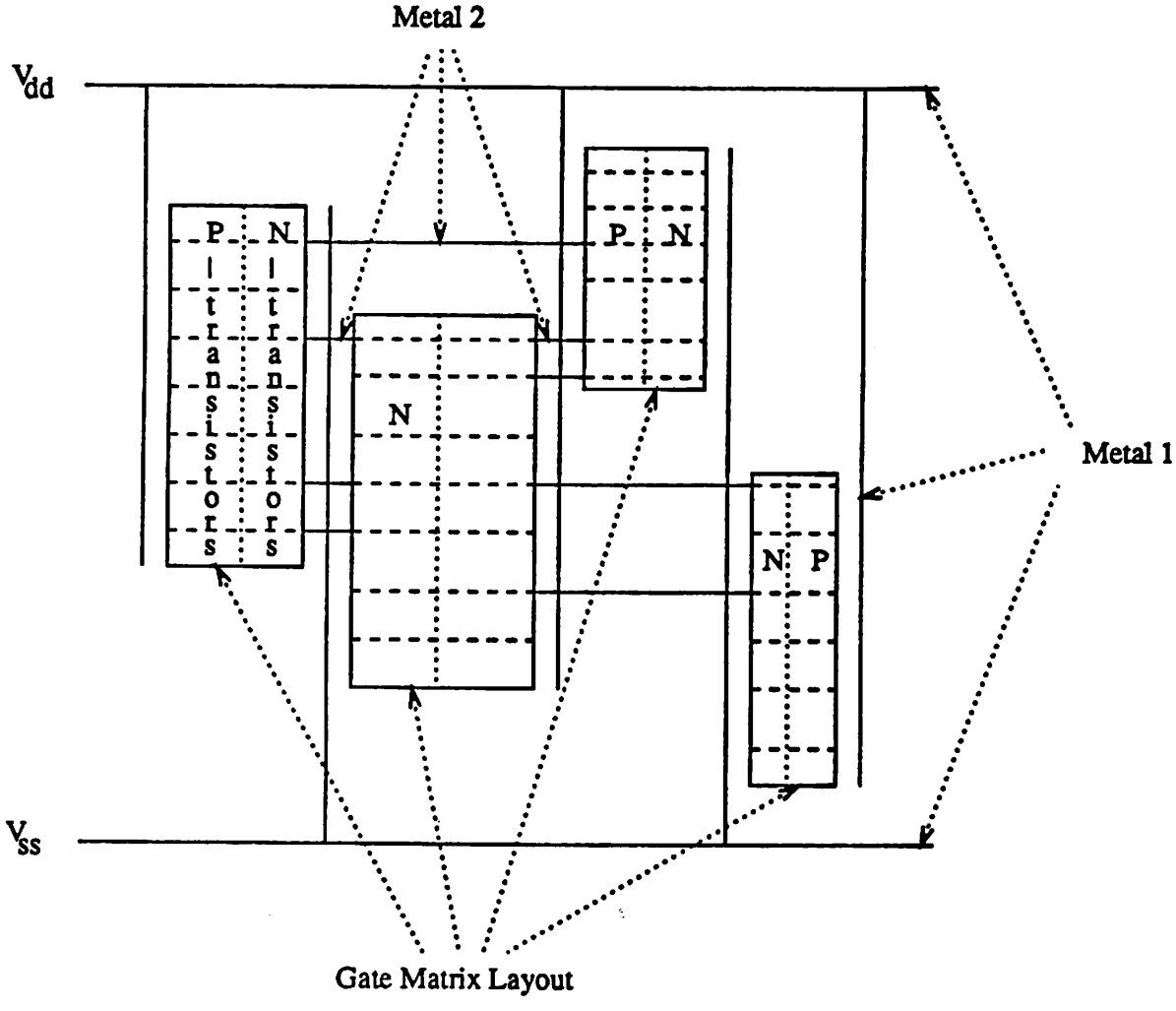


Figure 38

Chapter 6

6. Conclusion:

In this report, a new quadratic architecture, two-dimensional folding-based architecture (2FBA), is introduced. In this architecture, the horizontal nets are implemented on the second layer metal lines for the gate signals. The vertical nets consist of transistors which are connected by the first layer metal lines. The transistors sit on the grids which are formed by the horizontal and vertical nets.

To minimize the area of this new architecture, a new two-dimensional folding (or array optimization) algorithm is proposed. Instead of using simulated annealing technology, we use a deterministic strategy to solve this problem. The experimental results indicate that our algorithm, like simulated annealing, has the property of climbing out of the local optimums. However, it requires much less computing time. Some published examples were tested, with satisfying results.

Based on the new architecture and the new two-dimensional folding algorithm, a layout package, PASTORALE, is developed. In order to get better layout results, a more practical cost function is considered in an improved two-dimensional folding algorithm. Also, the swap operation is included in this improved algorithm.

Some 2FBA layout results are illustrated in this report. Since the horizontal and vertical nets are put on the equally spaced grids which are separated in a worst case distance, a compaction program is necessary. We intend to integrate a compaction package into PASTORALE. Some more future work is also discussed in the report.

REFERENCES

[1] Daniel D. Gajski, ed, "Silicon Compilation", the United States of America: Addison-Wesley Publishing Company, Inc. 1988, pp 9-27.

[2] A. Weinberger, "Large-Scale Integration of MOS Complex Logic: A Layout Method", IEEE Journal of Solid-State Circuits, vol. SC-2, no. 4, December 1967, pp 182-190

[3] A. Hashimoto, and J. Stevens, "Wire Routing by Optimizing Channel Assignment within Large Apertures", Proc. 8th Design Automation Conference, 1971, pp 155-169.

[4] T. Kashiwabara, and T. Fujisawa, "NP-completeness of the problem of finding a minimum-clique-number interval graph containing a given graph as a subgraph", Proc. 1979 IEEE ISCAS, pp 657-659.

[5] A.D Lopez, and H-F. S. Law, "A Dense Gate Matrix Layout Method for MOS VLSI", IEEE Trans. on Elec. Devices, vol. ED-27, no. 8, Aug. 1980, pp 1671-1675.

[6] O. Wing, "Automated Gate Matrix Layout", Proc. 1982 IEEE ISCAS, Rome, Italy, pp 681-685.

[7] T. Ohtsuki, H. Mori, E.S. Kuh, T. Kashiwabara, and T. Fujisawa, "One-dimensional Logic Gate Assignment and Interval Graph", IEEE Trans. Circuits Syst., vol. CAS-26, Sept. 1979, pp 675-684.

[8] H. Fleisher, and L.I. Maissel, "An Introduction to Array Logic", IBM Journal of Research and Development, vol. 19, no. 2, March 1975, pp 98-109.

[9] E. Horbst, "VLSI '85", Elsevier Science Publishers B.V., North-Holland, 1986, pp 33-48.

[10] G.O. Hachtel, A.R. Newton, and A. Sangiovanni-Vincentelli, "An Algorithm for optimal PLA folding", IEEE Trans. on CAD of Integrated Circuits and Systems, vol. CAD-1, no. 2, April 1982, pp 63-77.

[11] M. Luby, U. Vanzirani, V. Vazirani, and A. Sangiovanni-Vincentelli, "Some Theoretical Results on the Optimal PLA Folding Problem", Proc. IEEE International Conf. on Circuits and Computers, New York, NY, October 1982, pp 165-170.

[12] Q. Yu, and O. Wing, "Interval-Graph-Based PLA Folding", Proc. IEEE ISCAS, 1985, pp 1463-1466.

[13] G.D. Micheli, and A. Sangiovanni-Vincentelli, "Multiple Constrained Folding of Programmable Logic Arrays: Theory and Applications", IEEE Trans. on CAD, vol CAD-2, no. 3, July 1983, pp 151-167.

[14] C.K.C. Leung, S.S. Patil, and H. Ravindra, "The storage/Logic Array (SLA) Approach to IC Design", VLSI Design, January 1984, pp 54-61.

[15] K.F. Smith, T.M. Carter, and C.E. Hunt, "Structured Logic Design of Integrated Circuits Using the Storage/Logic Array (SLA)", IEEE Trans. on Electron Devices, April 1982, pp 765-776.

[16] S. Patil, "An asynchronous Logic Array", Project MAC Tech. Memo TM-62, May 1975.

[17] M. Hofmann, "Automated Synthesis of Multi-level Combinational Logic in CMOS Technology", Ph.D. dissertation, University of California, 1985.

[18] M. Hofmann, and A.R. Newton, "A Domino CMOS Logic Synthesis System", Proc. 1985 ISCAS, Kyoto, Japan, June 1985. pp --.

[19] S. Devadas, and A.R. Newton, "Topological Optimization of Multiple-Level Array Logic", IEEE Trans. on CAD, vol CAD-6, no.6, Nov. 1987, pp 915-941.

[20] S.M. Kang, "Metal-Metal Matrix (M^3) for High-Speed MOS VLSI Layout", IEEE Trans. on CAD, vol cad-6, no. 5, Sept. 1987, pp886-891.

[21] P.Gee , I.N. Hajj and S.M Kang, "iSILVER: A Symbolic Layout Generator for MOS Circuits", to be published.

[22] P. Gee, M.Y. Wu, S.M. Kang, and I.N. Hajj, "Metal-Metal Matrix (M^3) CMOS Cell Generator with Compaction", 1987 IEEE Int. Conf. on CAD, Nov. 1987, pp184-187.

[23] P. Gee, M.Y. Wu, S.M. Kang, and I.N. Hajj, "A Metal-Metal Matrix Cell Generator for Multi-level Metal MOS Technology", Integration, the VLSI Journal 9 (1990), vol. 9, no. 1, Feb, 1990, pp 25-47.

[24] L.P.P.P V Ginneken, J.T.J. Eijndhoven, J.A.H.C.M. Brouwers, "Doubly Folded Transistor Matrix Layout", 1988 IEEE Int. Conf. on CAD (Santa Clara, CA), Nov, 1988, pp134-137.

[25] D.F. Wang, and C.L. Liu, "Array Optimization for VLSI Synthesis", Proc. 1987 IEEE DA Conference, pp 537-543.