GM: A NEW GATE MATRIX LAYOUT SYSTEM

by

Dong-Min Xu

Memorandum No. UCB/ERL M89/81

27 June 1989

GM: A NEW GATE MATRIX LAYOUT SYSTEM

by

Dong-Min Xu

Memorandum No. UCB/ERL M89/81

27 June 1989

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

GM: A NEW GATE MATRIX LAYOUT SYSTEM

by

Dong-Min Xu

Memorandum No. UCB/ERL M89/81

27 June 1989

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

GM: A New Gate Matrix Layout System

Dong-Min Xu

Abstract

The random module generators are necessary in VLSI layout phase. GM is a new random module generator for CMOS circuits. It is based on the gate matrix layout style which has received considerable attention over the past few years. GM divided the whole layout procedure into two one-dimensional assignment problems. Some new algorithms are developed, and the results are promising.

GM also considered a lot of practical constraints such as the pre-assigned I/O pins, oversized transistors and critical nets. In order to improve the area usage and the aspect ratio, it applied the horizontal compaction to the gate matrix. Considerable improvement over each results has been obtained.

ACKNOWLEDGEMENTS

I am much indebted to Professor Ernest S. Kuh for his support, guidance and encouragement. I would like to express my deepest gratitude to Professor Yun-Kang Chen who enlightened my interests in VLSI layout. His guidance is much appreciated.

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. Gate Matrix Layout

A random logic module is defined as an irregular structure of basic components, such as transistors and gates. Being widely varied in complexity and type, the modules can not be kept in a library. So the random module generators are necessary in VLSI layout phase. In building block layout -- a cell based design style, the cells with certain size are in fact the results of the random module generators.

Gate matrix layout was first introduced by Lopez and Law [1] of Bell Laboratory for the layout of custom CMOS circuits. It is very similar to the Weinberger array [2] style which were studied in [3][16]. Since it uses a simple and regular structure, it is easy to complete a random logic automatically. Several module generators [5][6], based on this layout style, have been seen.

Gate matrix layout is a style for CMOS circuits. An example is shown in fig.1. In this layout, all the vertical polysilicon gates are placed in columns. All the transistors connected to the same gate signal are constructed along the same column. The interconnections among transistors are made by horizontal metal lines. A net is defined as a series of metal lines and transistors which are connected on a 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 should be reduced since the number of columns is already fixed. Because a row can be shared by more than one unoverlapped

nets, the number of rows heavily depends on the column ordering.

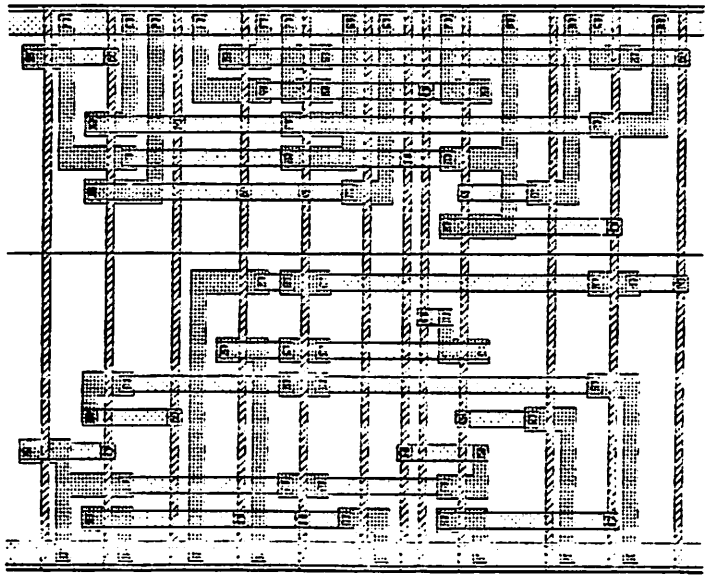


Figure 1

1.2. Previous Work

In 1982, Wing proposed automatic gate matrix layout [7]. He solved this problem by using a graph theoretic method which had been well considered in [3]. In this method, the nets are treated as intervals. A gate matrix is represented as an interval graph G(V,E), where V is the set of vertices which correspond to nets and E is the set of edges which connect vertices whose nets overlap in one or more columns. Wing transformed the gate matrix optimization problem into the problem of finding an interval graph from a general graph such that the maximum clique size of the interval graph is minimum. This problem is actually a one-dimensional assignment (or linear placement) problem which has been proved to be NP-complete [8].

Since this is NP-complete problem, most of published algorithms attacked this problem with heuristic strategies. All the algorithms can be classified into two categories: the constructive algorithms and the iterative improvement algorithms.

The constructive algorithms can be divided into two major parts. The first part aims at finding a good gate arrangement such that the total number of required tracks

is minimum. Ohtsuki et. al. [3] used a graph theoretic approach and first formulated this problem as an augmentation of a graph. By an augmentation, they mean a set of edges which are added to a graph such that the resulting graph becomes an interval graph. A augmentation which leads to a least clique number is what we want. Unfortunately, this problem turns out to be a NP-complete [8]. Instead, they proposed a polynomial-time algorithm for finding a minimal augmentation, where an augmentation is minimal if no proper subset of it is an augmentation. Another linear minimal augmentation algorithm [9] by using PQ-trees was proposed in 1981.

Base on the above idea, Wing [7] represented the connection graph by a dominant versus vertex matrix [10] and obtain an interval graph by reconstructing the matrix in such a way that its rows are minimally filled with ones so that it has the consecutive ones property. By using this method, Li proposed a family of heuristic algorithms [11]. We also published an algorithm [12] which improved Li's idea by using a two stages strategy to get rid of the greedy property. The results were much better than the previous ones.

Some other heuristic algorithms [13][14][15] were also proposed to attack this problem. Deo [21] formulated this problem as a dynamic programming problem and used a set of heuristic approaches to deal with the problem. Hwang, et. al. [22][23] used a modified min-net-cut heuristic in conjunction with dynamic net lists (D-nets) to get a at worst a log-n approximation algorithm.

The second part of the constructive algorithms attacked the one-dimensional problem by searching for an optimal "net ordering", rather than an optimal gate ordering. The corresponding gate ordering is then constructed according to the obtained net ordering. Asano [16][17] first proposed this method. Huang and Wing [18] gave a modified Asano's algorithm. The considerable improvement was reported.

There existed some other algorithms which can be called iterative improvement algorithms. These algorithms got an initial result first and then used heuristic approaches to improve it. Leong [19] gave an iterative improvement algorithm. He employed the technique of simulated annealing and used a cost function that minimized both the layout area and the total wire length. Yoshizawa proposed another heuristic algorithm [20] which used a clustering method to get an initial gate ordering

and then defined two exchange operations to do the further iterative improvement.

In addition to the above algorithms, some other improvements were completed to the gate matrix layout. In gate matrix, a group of series-connected transistors can be placed in either one or two rows [5], if the position of the transistors are exchangerable. To allow such an exchange, a special net representation is used for series-connected transistors. In [22][23], the series-connected transistors are treated as so-called "D-net", which allows exchanging of transistors in a "D-net".

In [5], two nets were merged into one when they have a MOSFET in common and overlap at a single gate column. Shu [24] showed a more compact layout which can be obtained by merging two nets so that they shared common track spaces even when they do not have a MOSFET in common. In fig.2, Shu's result, shown in fig.2(b) was much better than the usual methods, shown in fig.2(a).

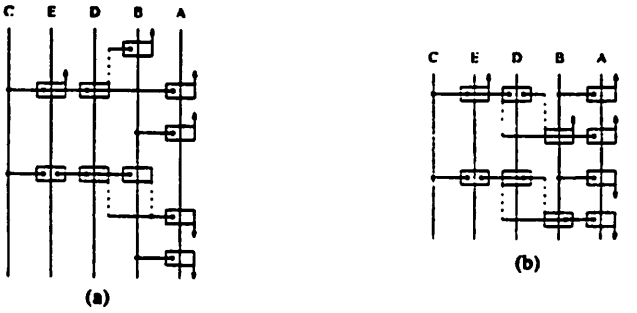


Figure 2

From the experimental statistics, the larger the circuit, the lower its density in a gate matrix layout. In addition, as the circuit size gets increased, the aspect ratio of the layout may become unacceptable. Huang proposed a heuristic gate matrix partioning algorithm [25] which divides a gate matrix into several blocks. The interconnection among blocks are implemented by gates which are placed on the same column such

that no extra routing area is needed.

Recently, a new two-dimensional folding cell generator [26] was proposed. It is based on gate matrix, but more flexible. It is suitable for NMOS circuits. In order to allow the folding of both the vertical signal nets and the horizontal poly strips, it does not require the transistors with the same gate signal to be on the same poly strip. The results were more compact and the aspect ratio was well controlled.

From the above discussion, we know that most of the authors focused on the one-dimensional algorithm, rather than implementing a layout system. Here in this report, we present a new gate matrix layout system which is called GM. A lot of attentions was put on both the algorithms and some detail improvements to the gate matrix layout.

1.3. Organization

In chapter 2, we give an overview of the GM program, and introduce our gate arrangement algorithm and its results. The considerations for series-connected transistors and the horizontal nets merging are included in chapter 3. The net assignment which is found to be an extended one-dimensional assignment problem is presented in chapter 4. Chapter 5 shows some layout results of the GM program. The conclusions are given in chapter 6.

Chapter 2

2. Gate Arrangement

2.1. GM Overview

GM is a gate matrix layout program which can treat the placement and interconnections of transistors at the same time. It uses the circuit description files (at transistor level) as its inputs. The outputs are given in CIF files. The program uses KIC command to draw the layout results on the screen.

In gate matrix layout style, all the vertical poly strips are usually considered to be equally spaced. Our program, however, does not keep the polys equally spaced. Instead, it applies the horizontal compaction to the gate matrix such that the polys are placed as close as possible if they do not violate the design rules (see fig.1). This process improves both the area usage and the aspect ratio a lot.

Another important feature of GM is that it allows users to do interactive improvement on the layout results. If the users consider some critical nets are too long, what they need to do just points these nets out and put the weight they want on them. GM will then automatically reroute these critical nets in the shortest possible distance.

GM can also handle some practical constraints such as oversized transistors and pre-assigned I/O pins. It allows the users to decide a certain range for a I/O pin's position or specify a special order for I/O pins. The program can automatically check if there exist conflicts among these pre-assigned I/O pins. Currently, the I/O pins constraints can be exerted to the vertical polys of the gate matrix.

GM divides the gate matrix layout procedures into two steps: gate arrangement and net assignment. The gate arrangement tries to determine a good gate sequence so

as to minimize the number of tracks and the total length of nets. Here, we use a modified algorithm of our original algorithm [12]. The net assignment is used to assign nets to tracks such that the total length of the vertical diffusion runs, and the intersections of the vertical diffusion runs over transistors are minimal. Our recent work shows that the net assignment problem is also a one-dimensional assignment problem. But it is a little bit different from the original one-dimensional assignment problem. We called it an extended one-dimensional problem. Based on our formulation, a heuristic algorithm was proposed. The results are satisfying.

2.2. Introduction

In gate matrix layout, shown in fig.3, all P transistors are placed in the top half of the matrix and all N transistors in the lower half. The column of the matrix are polysilicon stripes (also called gates) which are not always equally spaced. A gate serves two functions: first, as a transistor gate when diffusion areas placed on each side of the stripes; second, as a conductor which connects two gates of each complementary pair of transistor. The interconnection among transistors are made by segments of horizontal metal lines (called nets) which are placed in the rows of the matrix. A transistor is formed when a horizontal diffusion intersects with a vertical poly.

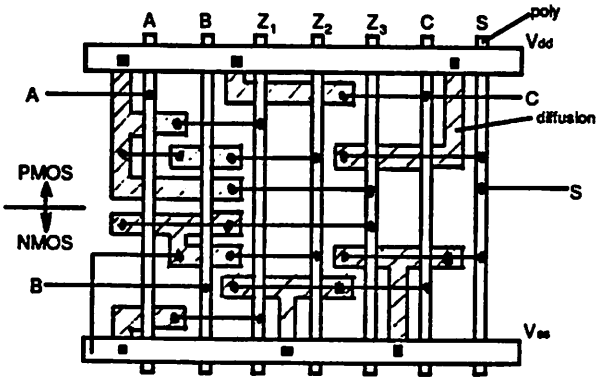


Figure 3

In gate matrix layout, we usually divide the layout procedure into two steps: first, we determine an optimal gate sequence, so as to minimize the number of tracks and

the total length of nets. Second, we assign nets to tracks.

In [5], Wing, Huang and Wang formulated the above two steps in terms of two function , f and h. Function f is a one-dimensional assignment problem that assigns the gates to the columns of the matrix. Function h is the function that puts the nets on to the rows of the matrix. For function h, Wing, Huang and Wang [5][18] mainly focused on "realiable" vertical diffusion runs, but they did not present a systematic procedure.

In this chapter, a new gate matrix layout program is introduced. we propose an improved algorithm based on our earlier work[12] that takes into account practical constraints, such as I/O nets, I/O gates and critical nets.

2.3. Improved Gate Arrangement Algorithm

2.3.1. Gate Arrangement Problem

The gate arrangement problem can be symbolicly represented by Fig.4, where the horizontal lines indicate the nets and the vertical lines the gates.

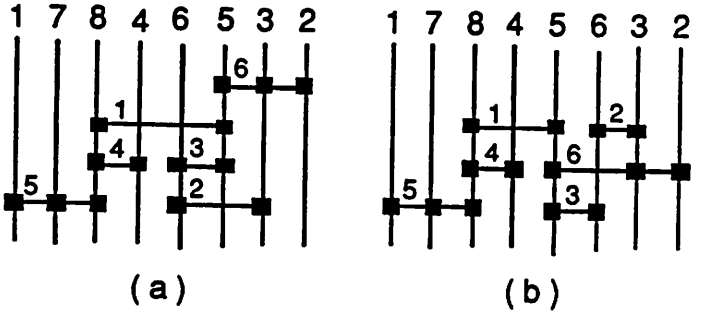


Figure 4

If we rearrange the order of the columns in Fig4(a), Fig4(b) is obtained. The number of tracks is reduced from four to three. Thus, the gate arrangement problem is to find an optimal permutation of columns, so as to minimize the number of horizontal tracks and the total length of nets.

2.3.2. I/O Constraints

In gate matrix layout, an I/O pin can be placed at the left or the right boundary through metal nets, called I/O net. It can also be placed at the top or the bottom boundary through polysilicon gates, called I/O gate. Usually, the position of I/O nets and gates, and the specific order of I/O nets and gates are decided before a gate matrix layout is generated.

2.3.3. Critical nets constraints

In order to optimize the performance of the layout results, we should consider the delay problem in gate matrix layout. By a critical net, we mean a part of the net path which influences the performance in a critical way. It should be laid in a shorter distance.

2.3.4. Improved algorithm

Here, we propose an improved algorithm over that in our previous paper [12]. Both the I/O constraints and critical net constraints are considered. We define a vertex versus gate matrix (for short, v.g. matrix) as follow:

Def 1: A v.g. matrix A=(a_{ij}) is a $m \times n$ matrix, where m is the number of nets, n the number of gates. It is defined by

$$a_{ij} \begin{cases} =1 & \text{if } \textit{net } i \textit{ connects gate } j \text{ and } \textit{this element is not on a critical path} \\ >1 & \text{if } \textit{net } i \textit{ connects gate } j \text{ and } \textit{this element is on a critical path} \\ =0 & \textit{Otherwise} \end{cases}$$

In the above definition, if $a_{ij} > 1$, the value of a_{ij} indicates the weight of the critical part of a net.

In order to demonstrate the definition clearly, an example is illustrated below:

$$A_1 = \begin{bmatrix} 0\,0\,0\,0\,1\,0\,0\,1 \\ 0\,0\,1\,0\,0\,1\,0\,0 \\ 0\,0\,0\,0\,1\,1\,0\,0 \\ 0\,0\,0\,1\,0\,0\,0\,1 \\ 1\,0\,0\,0\,0\,0\,1\,1 \\ 0\,1\,2\,0\,2\,0\,0\,0 \end{bmatrix}$$

From the above matrix A_1, we know that net 6 is a critical net. So, column 3 and column 5 should be put as closely together as possible.

A v.g. matrix has the consecutive non-zero element property, i.e., the non-zero elements in each row occur in consecutive positions. We call the v.g. matrix with consecutive non-zero element property realizable. If a v.g. matrix is not realizable, one simply changes the order of columns, fills each rows with ones (these ones are called fillins) and makes the resultant filled matrix A' realizable. Thus, A' realizes the connection relations of matrix A.

In our original algorithm [12], we reconstruct matrix A' from a v.g. matrix A column by column, so as to realize the given connection relations of A. The algorithm includes two stages. In the first stage, it divides the candidate gates (i.e. unplaced gates) into the "selectable gate" set (for short, s.g. set) and "unselectable gate" set (for short, u.g. set). The division is made in such a way that the nets connecting to the u.g set have as little incident relation to the current nets (i.e. the nets being considered) as possible. Then, the next gate will be selected from the s.g. set only. In the second stage, the algorithm will make further choice from the s.g. set to obtain the best selected gate. The main objective in this stage is to minimize the total length of nets. In order to achieve good results, the subsequent selection contains several steps and considers factors such as the number of fillins, starting nets and ending nets, etc.

In this improved algorithm, we also keep the above two stages strategy. First, we decide the selectable gate set with exactly the same method as [12]. Second, we make further choice in selectable gate set with the consideration of I/O constraints and critical path constraints. Also in this paper, we introduce a concept of dynamic constraint graph. The initial dynamic constraint graph is the very connection graph. All incident nets in this graph can not be put on the same track. When we reconstruct matrix A' from matrix A column by column, the dynamic constraint graph will be changed. When the matrix A' is finally formed, the dynamic constraint graph becomes an

interval graph. The clique number of this interval graph is the number of tracks occupied by all nets.

In [12], we use a two-stage method to choose an initial column. The obtained initial column has a little incidence relation to the other nets. We found that this method made most of the nets to be heaped in the center of the gate matrix. The layout results are influenced a lot by the initial column. In the improved algorithm, we use a new method to choose an initial column. Our heuristic idea is that the more the short nets, connected to the initial column, the better is the result. This strategy will cause many short nets placed at the beginning of the layout. So the distribution of the nets is more reasonable than it was before.

For the above matrix A_1, assume the I/O constraints are as follow: net 5 and net 6 are placed at the left and right boundary, respectively (we put two fictitious columns outside the matrix A_1' to indicate the constraints of these two I/O nets). Column 4 and column 7 are placed at position 4 and 6 in new formed matrix A_1'. As indicated earlier, net 6 is a critical net. So, before we perform rearrangement of the v.g. matrix A_1, we already have:

$$A_1' = \begin{array}{c} \\ 0 \\ 0 \\ 0 \\ 0 \\ \rightarrow 1 \\ 0 \end{array} \begin{array}{c} \begin{array}{cccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \end{array} \\ \left[\begin{array}{cccccccc} & & & 0 & 0 & & & \\ & & & 0 & 0 & & & \\ & & & 0 & 0 & & & \\ & & & 1 & 0 & & & \\ & & & 0 & 1 & & & \\ & & & 0 & 0 & & & \end{array}\right] \end{array} \begin{array}{l} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \rightarrow \text{I/O net} \end{array}$$

The algorithm is to construct A' from A column by column. The columns are placed from left to right in A'. For the partially formed matrix A', the candidate gate set C is defined by gates which have not been fixed. Assume that the selectable gate set s.g. has been obtained (s.g. $\subseteq$ C). For $j \in s.g.$, we define:

fillin(j): the number of fillins which makes A' maintain the consecutive non-zero element property when column j is catenated to A'.

end(j): the number of ending nets when column j is put in A'. For the above matrix A_1, assume the partially formed matrix A_1' is as follow:

$$A_1' = \begin{matrix} 1 & 8 & 2 & 4 \end{matrix} \begin{bmatrix} 0 & 1 & * & * \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & * & 1 \\ 1 & 1 & * & * \\ 0 & 0 & 1 & * \end{bmatrix}$$

When column 4 is put in position 4, only net 4 is laid out. So we have end(4)=1;

distance(j): the distance between the current position and the first non-zero element of the prefixed column (I/O gate). Assume a net in the partially formed matrix A' is:

$$(0\,0\,0\,0\,0\,\underset{\wedge}{\_}\,\_\,\_\,\_\,1)$$

In this case, if the current position (at column 6) is occupied by 1, the next three space will certainly be filled with fillins or ones. The number of these spaces, three, is called distance. It is clear that the shorter the distance(j), the better are the results.

constraint(j): the number of new added edges in the dynamic constraint graph, when column j is catenated to A'. This will be illustrated as follow: for the above given matrix A_1, assume the partially formed matrix A' is:

$$A_1' = \begin{matrix} 1 & 8 & 2 \end{matrix} \begin{bmatrix} 0 & 1 & * \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 1 & * \\ 1 & 1 & * \\ 0 & 0 & 1 \end{bmatrix}$$

Before column 2 is put in A_1', the constraint graph is the connection graph, shown in fig5.(a).

When column 2 is put in A', it will lead into some new constraints, indicated by broken lines in fig5.(b). In this case, constraint(2)=2 (2 is the number of broken lines). If we really put column 2 in A' after the column choice, the constraint graph will be changed by adding these two edges. We call this new constraint graph, the dynamic

constraint graph.

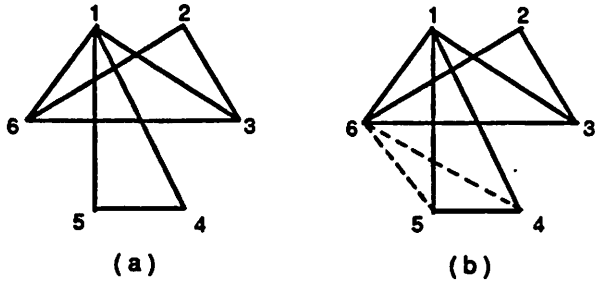


Figure 5

start(j): the number of new starting nets when column j is put in the current position.

For the above matrix A_1, assume the partially formed matrix A_1' is as follow:

$$A_1' = \begin{matrix} \\ 0\,1 \\ 0\,0 \\ 0\,0 \\ 0\,1 \\ 1\,1 \\ 0\,0 \end{matrix} \begin{matrix} 1\,8 \\ \left[\begin{matrix} \\ \\ \\ \\ \\ \end{matrix}\right. \end{matrix}$$

When column 8 is put in position 2, two nets, e.g. net 1 and net 4, are introduced. So we have start(8)=2.

critical(j): the product of the number of fillins among critical part of nets and its weight. For example, in matrix A_1', when column 7 is put in position 6, the critical(7) = 1×2 = 2.

$$A_1' = \begin{matrix} \\ 0\,1\,*\,*\,1\,0\,0\,0 \\ 0\,0\,0\,0\,0\,0\,1\,1 \\ 0\,0\,0\,0\,1\,*\,*\,1 \\ 0\,1\,*\,1\,0\,0\,0\,0 \\ 1\,1\,*\,*\,*\,1\,0\,0 \\ 0\,0\,1\,*\,2\,*\,2\,* \end{matrix} \begin{matrix} 1\,8\,2\,4\,5\,7\,3\,6 \\ \left[\begin{matrix} \\ \\ \\ \\ \\ \end{matrix}\right. \end{matrix}$$

We define a function α(j) as follow:

$$\alpha(j) = critical(j) + constraint(j) + fillin(j) - end(j)$$

For all gates in the s.g. set, we choose the best one, $j \in s.g.$, by the following steps:

(1) Select the column with the smallest $\alpha(j)$, if more than one.

(2) Select the column with the smallest distance(j), if more than one.

(3) Select the column with the smallest start(j), if more than one.

(4) Select a column arbitrarily.

Now, we construct A_1' from A_1 step by step:

step 1: I/O net 5 is placed at the left boundary of the matrix A_1'. From fig.5(a), the adjacent nets of net 5 are net 1 and net 4. According to the strategy of [12], the s.g. set {1,4,7,8} is obtained. Because column 4 and column 7 have been prefixed, the s.g. set is modified to be {1,8}. Then the function $\alpha(j)$ is calculated as follow:

column	1	8
fillin	0	0
end	0	0
distance	4	6
constraint	0	0
start	0	2
critical	0	0
α	0	0

According to our choice strategy, column 1 is picked up in this step.

step 2: In this step, s.g.={8}. So only column 8 is chosen for catenating to A_1'.

step 3: In this step, s.g.={2,5}. So we have:

column	2	5
fillin	3	2
end	0	1
distance	0	0
constraint	2	4
start	1	2
critical	0	0
α	5	5

According to selection method, column 2 is chosen.

The remaining columns chosen by using the above method, are shown in table 1.

Table 1

step	u.g. set	s.g. set	chosen column	order of column
1	2,3,4,5,6,7	1,8	1	1,
2	2,3,4,5,6,7	8	8	1,8,
3	3,4,6,7	2,5	2	1,8,2,
4			4(I/O gate)	1,8,2,4,
5	7	3,5,6	5	1,8,2,4,5,
6			7(I/O gate)	1,8,2,4,5,7,
7		3,6	3	1,8,2,4,5,7,3,
8		6	6	1,8,2,4,5,7,3,6,

The merit of this algorithm is that:

(1) it considers the I/O net and gate constraints reasonably.

(2) it can treat critical net constraint so that the circuit performance can be improved.

(3) it does the gate assignment more wisely, because of the introduction of the concept of dynamic constraint graph.

2.3.5. Results

We have implemented our improved algorithm. Several examples are tested in comparison with the results of [18] in table 2. The results are satisfying.

Table 2

examples		No.1		No.2		No.3	4-B ALU		No.4
		N	N&P	N	N&P		N	N&P	
# of transistors		50	118	232	462	306	158	316	76
# of gates		33	33	140	211	71	70	70	28
# of nets		48	87	199	370	131	84	214	36
Lower Bound		14	27	16	30	19	11	22	5
Improved Gate Matrix Layout(H&W)	# of rows	14	28	32	73	40	28	58	10
Our Algorithm	# of rows	15	30	29	62	36	23	58	10

Chapter 3

3. Practical Considerations

3.1. Serial Transistors

In a static circuit, the order of a series-connected transistors is logically immaterial. We consider a group of series-connected transistors as a serial net which is not connected with other transistors before the gate arrangement. After the gate arrangement is done, this net should be connected with the specified transistors. For a serial transistors group in fig.6(a), it can be treated as a serial net, shown in fig.6(b). (Here, n_1 and n_2 are the outside nets to which the serial transistors should connect). After the gate arrangement is done, the gate ordering may be as shown in fig.6(c). Fig.6(c) corresponds to the circuit, shown in fig.6(d). By comparing fig.6(a) and fig.6(d), we know that the ordering of serial transistors are changed. This means the gate of the serial transistors can be reordered as one wishes. The main problem which we face is how to connect serial transistors to the outside nets after the gate arrangement is done.

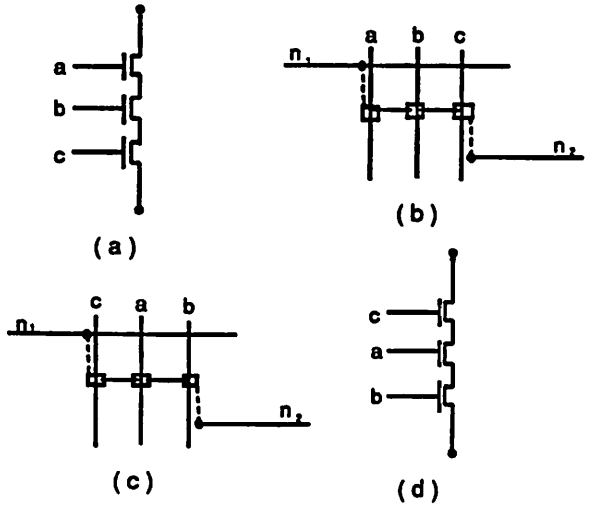


Figure 6

For each serial transistors group, two outside nets are necessary. For each outside net, we have to stretch it to the left by **l** or the right by **r** (see fig.7), so as to connect it with the end of the serial net. For both outside nets, we can calculate l_1, r_1 of net n_1 and l_2, r_2 of net n_2 easily. We choose a connection which has a minimum stretch for the outside nets.

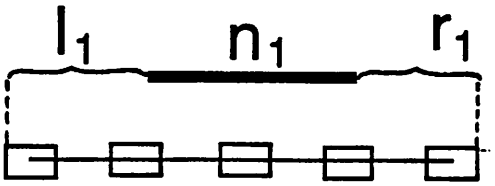


Figure 7

For the example in fig.8(a), $l_1 + r_2 = 0$, $l_2 + r_1 = 3$, So we choose min($l_1 + r_2$, $l_2 + r_1$) = $l_1 + r_2 = 0$, which is shown in fig.8(b).

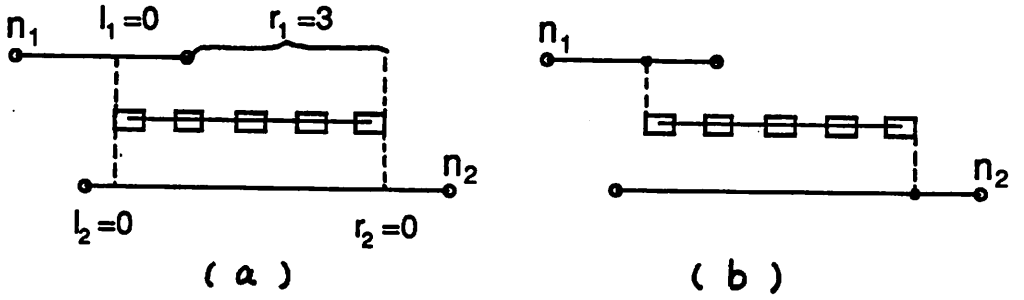


Figure 8

3.2. Nets Merging

After the gate sequence is obtained from the gate arrangement algorithm, our program will do the net merging, as shown below. There are three kinds of nets which

can be merged. They are shown in fig.9(a). The merged results are shown in fig.9(b).

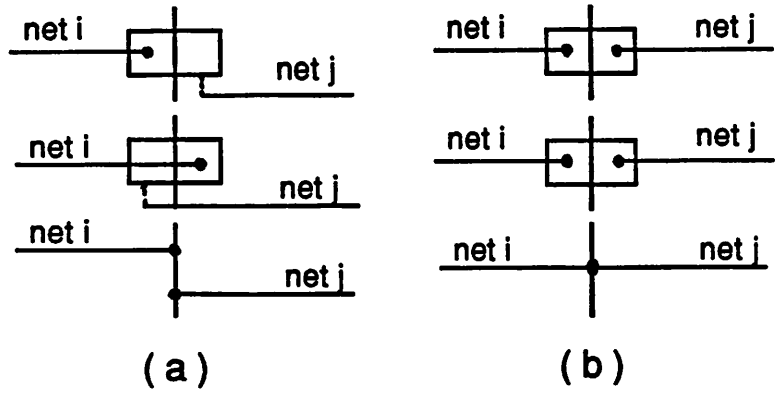


Figure 9

3.3. Adjustment of Vertical Diffusion Runs:

For an initial constructed gate matrix, the distribution of its vertical diffusion runs may not be reasonable. So our program will balance the vertical diffusion runs at the two sides of a gate. For the example shown in fig.10(a), all the four vertical diffusion runs are placed at the right side of the gate. This may be difficult sometimes for the net assignment algorithm to be realized without the violation of the design rules. In order to avoid this tough case, we try to make the number of vertical diffusion runs at each side of the gate be nearly same. The new distribution of the vertical diffusion runs for the example is shown in fig.10(b).

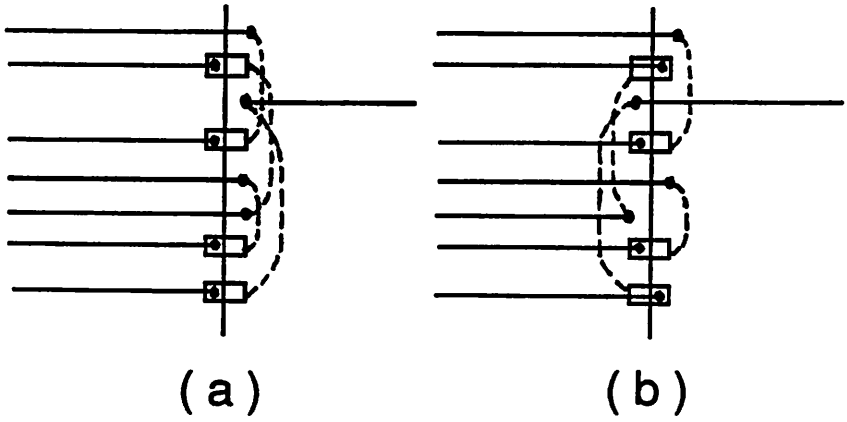


Figure 10

Chapter 4

4. Net Assignment

4.1. Introduction

After gate arrangement, net merging, adjacentment of vertical diffusion runs, and the treatment of the serial transistors, the net assignment algorithm is necessary for assigning nets to tracks. Since the number of tracks have been determined at this stage, the net assignment algorithms can not reduce the number of tracks anymore.

The gate arrangement problem has been proved to be NP-complete. The net assignment problem still needs more considerations.

For the net assignment problem, if the short vertical diffusion runs in gate matrix are not considered, the left-edge-first algorithm[4] can find a net assignment with minimum number of tracks which is determined by the gate assignment algorithm.

In practical cases, however, we have to consider the vertical diffusion runs. Each vertical diffusion run is attached to a gate. In order to improve the performance of the circuit, the length of the vertical diffusion runs should be routed as short as possible. An upper boundary for the maximum length of vertical diffusion runs is necessary.

In [5], Wing and Huang gave a concept of a realizable net assignment. They considered that a net assignment was unrealizable if a transistor is located on the path of a vertical diffusion run. In this case, the enlargement for an extra space was necessary, shown in fig.11, so as to make the layout realizable. Huang and Wing developed a zone-net assignment algorithm[18]. A zone is decided in such a way that any two nets in each zone can not be put on the same track, namely, the nets in a zone overlap one another. Paper [29] gave a well description about this zone decision. The algorithm in [18] completed net assignment zone by zone from left to right. For the net

assignment in each zone, Huang and Wing utilized a heuristic method.

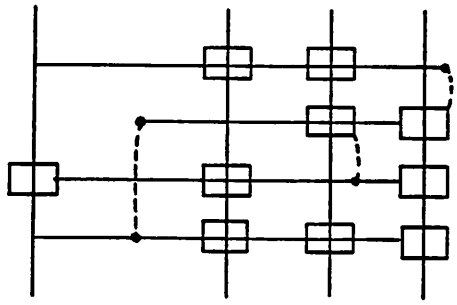


Figure 11

Based on the formulation in [5], Rim and Nakajima proved that the net assignment problem is NP-complete[28]. And then a heuristic algorithm was proposed.

In this chapter, we will present a new formulation for the net assignment problem. It is quite different from the formulation in [5]. Based on our new formulation, we find that the net assignment problem is something like one-dimensional assignment problem. But it is a little bit different from the original one-dimensional assignment problem. So we call it an extended one-dimensional assignment problem.

We have developed an algorithm for this extended 1-d assignment problem. The results are promising.

4.2. Formulation:

After the gate assignment is done in gate matrix layout, a new gate sequence is obtained. So, the number of tracks becomes fixed, and the net assignment is next to be done. Now, we define the unrealizable net assignment as follow:

Def 1: For all vertical diffusion runs at the same side of a gate, if any two of them intersect each other, we say that the net assignment is unrealizable.

In fig.12, the vertical diffusion runs v2 and v3 have to intersect each other. So this net assignment is unrealizable.

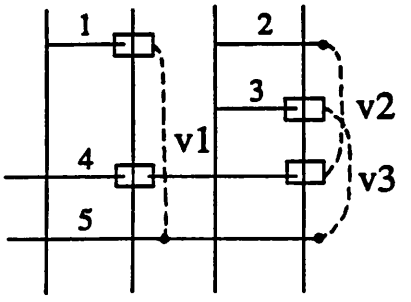


Figure 12

In net assignment, we also take into account another factor which is the total length of vertical diffusion runs.

Def 2: For a given gate matrix layout, we define the total length of vertical diffusion runs to be the sum of the length of all vertical diffusion runs.

In fig.12, the length of vertical diffusion run v1 is 4-1=3, and the length of v2 and v3 are 3-1=2 and 4-2=2, respectively. So the total length of vertical diffusion runs is 3+2+2=7.

In net assignment, if a vertical diffusion run intersects with a transistor, an enlargement should be done, shown in fig.11. So those intersections have to be limited so as to reduce the area usage.

Def 3: For a given gate matrix layout, we define the conflict number to be the sum of intersections of vertical diffusion runs over transistors in the matrix.

For the example shown in fig.12, the vertical diffusion run v1 intersects with one transistor. V2 and v3 intersect with one transistor each. Thus, the conflict number is 3.

Now, we give our new formulations as follow:

Net Assignment Problem: For a given gate order of a gate matrix, the number of tracks is fixed. The objective of the net assignment is to put all nets on to tracks so as to minimize the total length of vertical diffusion runs and the conflict number while keeping the track number unchanged. The final layout must be realizable.

4.3. Extended One-Dimensional Assignment Problem:

4.3.1. One-Dimensional Assignment Problem:

One-dimensional assignment problem (or linear placement, gate sequencing problem) can be formulated as follow: Given a set of modules, numbered as 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 ith position on a row. The most commonly used objective function for minimization is the total wire length. However, a different criterion is used in one-dimensional logic gate array and gate matrix layout. The number of tracks has to be minimized.

4.3.2. Extended One-Dimensional Assignment Problem:

For the extended 1-d assignment problem, the modules are defined to be the vertical strips which have different vertical ranges. The strips are considered having no

width. Fig.13 shows several strip modules.

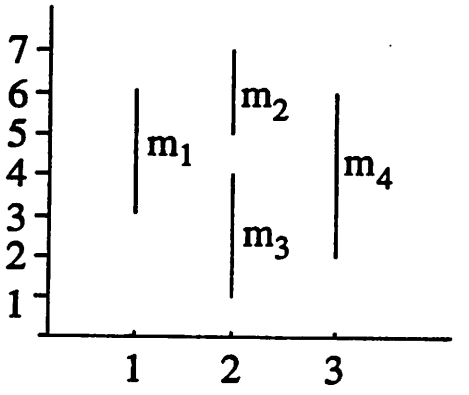


Figure 13

The vertical range of a strip module can be measured by an interval (x_1,x_2). For example, the interval of strip module m_1 is (3,6).

Several strip modules can be put on to the same column, if they do not overlap. We consider each column, with at least one strip module on it, as a track. The strip modules in fig.13 occupy three tracks (or columns).

The strip modules can be connected by nets. A net can be used to connect strip modules in horizontal direction. The jogs are not allowed for each net. So it is obvious that the strip modules can not be connected by a net, if they are on the same track (or column). Fig.14 shows an example of a connection pattern specified by several nets.

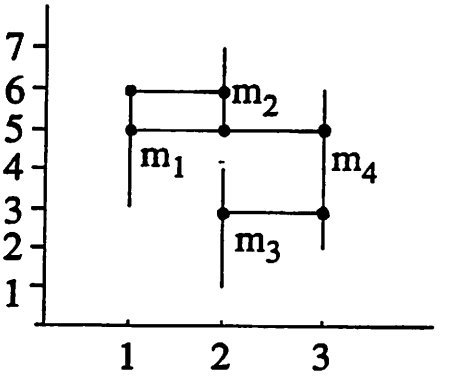


Figure 14

The EXtend One-Dimensional Assignment Problem (EXODAP) can be stated as follow: Given a number of strip modules which have different ranges, and a set of nets which specifies the connection pattern. The extended 1-d assignment problem can be considered to be an arrangement of strip modules such that the number of tracks (or columns) occupied by strip modules and the total length of nets are minimized.

4.3.3. Extended 1-D Assignment Algorithm:

This EXtended One-Dimensional Assignment Algorithm (EXODAA) will assign all strip modules to tracks (or columns) one by one from left to right. The track which is being treated is called current track. Assume

(a) M is a set of all strip modules connected by a set of nets.

(b) FM is a subset of M whose modules have been fixed on to tracks (or columns).

(c) UFM is a subset of M whose modules have not been fixed on to tracks (or columns).

(d) CFM is a set of strip modules which have been fixed on the current track.

If we compare strip module $m_i \varepsilon UFM$ with all strip module $m_j \varepsilon CFM$, the algorithm will decide a selectable module set SM from UFM in such a way that the strip module $m_k \varepsilon SM$ does not overlap with all $m_j \varepsilon CFM$. It is obvious that any strip module $m_k \varepsilon SM$ can be assigned on to the current track (or column) without leading to the overlap of modules. The set SM is a subset of UFM.

```
Algorithm EXODAA
  {
    i=1;
    while ( UFM != ϕ ) {
        get SM by comparing strip modules in UFM with
        strip modules on the current track i;
        if ( SM == ϕ ) {
            i++;  SM = UFM;
```

```
            }
            for all strip modules in SM, select the best strip module r∈SM so as to
            minimize the total length of nets, and put it on to the current track i;
            UFM = UFM - {r};
        } /* while */
    }
```

Now we illustrate the algorithm step by step for fig.14. At the beginning, we choose an initial module m_1 and put it on track i (i=1). So we have M = { m_1, m_2, m_3, m_4 }, FM = { m_1 }, and UFM = { m_2, m_3, m_4 }. Since the strip module m_1 overlaps with all strip modules in UFM, SM is empty. Then we do next track, namely set i=2. According to the algorithm, we have SM = UFM. If module m_2 is put on the track 2, we get FM = { m_1, m_2 }, UFM = { m_3, m_4 }, CFM = { m_2 }, and SM = { m_3 }. The strip module m_3 can be put on to the same track as m_2 is on. The final strip module m_4 has to be put on track 3 (i=3).

In the extended 1-d assignment problem, if we directly use left-edge-first algorithm to assign strip modules on to tracks, the length of nets may be quite long. The algorithm EXODAA can survey a lot of choices. So it can surely get much better results.

For the extended 1-d assignment problem, if we consider each strip modules as an interval, an interval graph then can be obtained. The left-edge-first algorithm can get an assignment result with the number of tracks to be the clique number of this interval graph. It is easy to prove that the number of tracks, which the EXODAA algorithm needed, is also this clique number.

4.4. Net Assignment:

In gate matrix layout, the objective of the net assignment is to minimize

(a) the total length of vertical diffusion runs, and

(b) the conflict number

such that the net assignment is realizable.

According to the above discussions, if we consider each net in gate matrix as a strip module and each vertical diffusion run in gate matrix as a net in EXODAP, then, the net assignment problem can be transformed into the extended 1-d assignment problem.

Fig.15(a) is a net assignment problem. It can be easily transformed into the extended 1-d assignment problem, shown in fig.15(b)

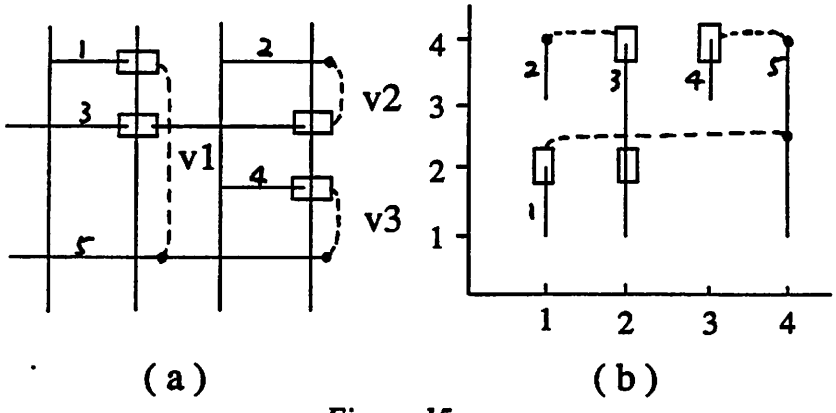


Figure 15

Our net assignment algorithm is a modified version of the algorithm EXODAA. It assigns nets to tracks column by column from left to right. The track which is being treated is called the current track. Assume

(a) N is the set of all nets,

(b) FN is a subset of N whose nets have been fixed on to the tracks,

(c) UFN is a subset of N whose nets have not been fixed on to the tracks yet,

(d) CFN is a set of nets which have been fixed on the current track.

With exactly same method as we used in EXODAA, we can get a selectable net set SN from UFN. Any net $n_k \varepsilon SN$ can be assigned on to the current track without leading to the overlap of nets. In order to choose the best net $n_r \varepsilon SN$ more wisely at current

step, we introduce some functions as follows:

v_danger(k): during the net assignment, we must find a realizable layout. For the example shown in fig.15, assume FN={5}, UFN={1,2,3,4}, the current track 2 is empty. So we have SN={1,2,3,4}. The vertical diffusion v2 and v3 can not intersect each other, otherwise the layout is unrealizable. So if net 5 is fixed on track 1, net 2 and net 3 can not be placed on to a track until net 4 is placed. In this case, we say that net 4 is a dangerous net, caused by v3. For each net, several vertical diffusion runs, like v3, may be connected to it. We define the dangerous number of a net to be the number of this kind of vertical diffusion runs. V_danger(k) is the dangerous number of net k. In above case, the dangerous vertical diffusion run of net 4 is v3. So we have v_danger(4)=1.

In order to make the layout to be realizable, some nets should be exempt from SN. So the algorithm must modify SN. In the above case, net 2 and net 3 are needed to be exempt from SN in current step. So we have SN={1,4}.

v_ending(k): In current step, if net k has been put on to the current track, the number of ended vertical diffusion runs on net k is v_ending(k). In the above example, v_ending(1)=1 and v_ending(4)=1.

v_conflict(k): when net k is put on to the current track, some vertical diffusion runs may go over the transistors on this net. The number of these vertical diffusion runs is v_conflict(k).

v_fillin(k): when net k is placed on to the current track, the number of unended vertical diffusion runs is v_fillin(k). For the above example, v_fillin(1)=0 and v_fillin(4)=0.

For all nets in SN set, the algorithm chooses the best one $n_r \varepsilon SN$ by the following steps:

(1) Select the net with the biggest v_danger(k), if more than one.

(2) Select the net with the biggest v_ending(k), if more than one.

(3) Select the net with the smallest v_conflict(k), if more than one.

(4) Select the net with the smallest v_fillin(k), if more than one.

(5) Select a net arbitrarily.

```
Algorithm NET_ASSIGN
  {
    i=1;
    while ( UFN != ϕ ) {
        get SN set by comparing nets in UFN with nets on the current track i;
        if ( SN == ϕ ) {
              i++;  SN = UFN;
              }
        get rid of the unrealizable layout by modifying SN;
        if ( SN == ϕ ) {
              printf("Net assignment can't be completed.");
              stop;
              }
        else {
              for ( net k∈SN ) {
                 calculate v_danger(k), v_ending(k), v_conflict(k), v_fillin(k);
                 } /* for */
              choose the best net r∈SN and put it on to the current track i;
              UFN = UFN - {r};
             }
        } /* while */
  }
```

Table3 illustrates the NET_ASSIGN algorithm step by step for fig.15.

Table 3

UFN set	current track	CFN set	SN set	modified SN set	net-->track
1,2,3,4,5	1		1,2,3,4,5	1,2,3,4,5	5-->1(initial choice)
1,2,3,4	2		1,2,3,4	1,4	4-->2
1,2,3	2	4	1	1	1-->2
2,3	3		2,3	2,3	2-->3
3	4		3	3	3-->4

From the above description, we may say that the net assignment problem can be considered as an extended 1-d assignment problem. In fact, the net assignment algorithm NET_ASSIGN not only rearranges the sequence of the nets, but also puts some nets which do not overlap one another, on the same track.

5. Conclusion:

The algorithm has been programmed in C and implemented on VAX780. Two examples from [5][27] were tested. Our net assignment result did not require any enlargement for example in [3], as shown in fig.16, while two enlargements were needed in [5]. For this example, the total length of vertical diffusion runs in [28] is 25,

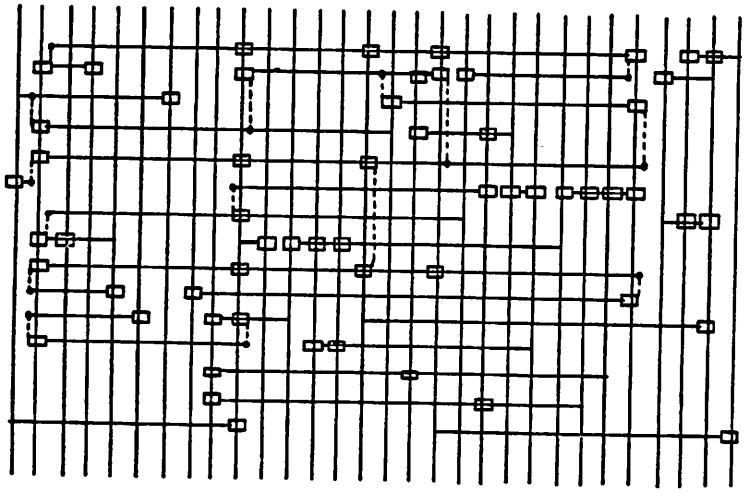


Figure 16

while we use 22. For the example in [27], the total length of vertical diffusion runs we

needed was 24, shown in fig.17. It was better than the result in [27]. All the detail comparisons are shown in table 4.

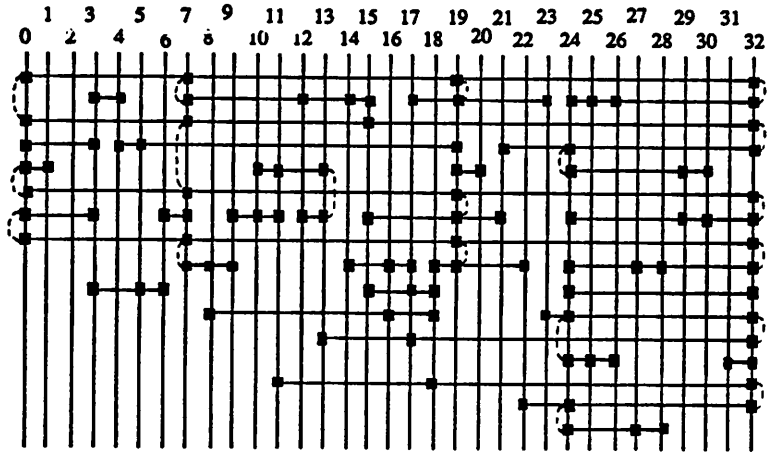


Figure 17

Table 4

Examples		H&W	R&N	Ours
Example1[5]	enlargement	2	0	0
	total length of vrun	23	25	22
Example2[27]	enlargement	0		0
	total length of vrun	28		24

This chapter presents a new formulation for net assignment problem in gate matrix layout. It also defines a new problem, called extended 1-d assignment problem which is something like 1-d assignment problem. But it is not the traditional 1-d assignment problem anyway. The paper proposes a heuristic algorithm to solve this extended 1-d assignment problem. The algorithm is applied to the net assignment problem. The time complexity of this algorithm is bounded by $O(m \times n^2)$, where n is the number of nets, m is the sum of the number of vertical diffusion runs and the number of gates.

Chapter 5

Results

Our gate matrix layout program (GM) has been implemented in C on VAX780. It can work in both direct mode and interactive mode. The interactive mode is capable of making some critical nets shorter by pointing out the list of the critical nets. GM will improve the length of the critical nets iteratively until users are satisfied. Fig.18 shows a layout result of a circuit with 118 transistors. The horizontal dark lines are critical nets. The iterative improvement to fig.18 achieved the reduction of the area

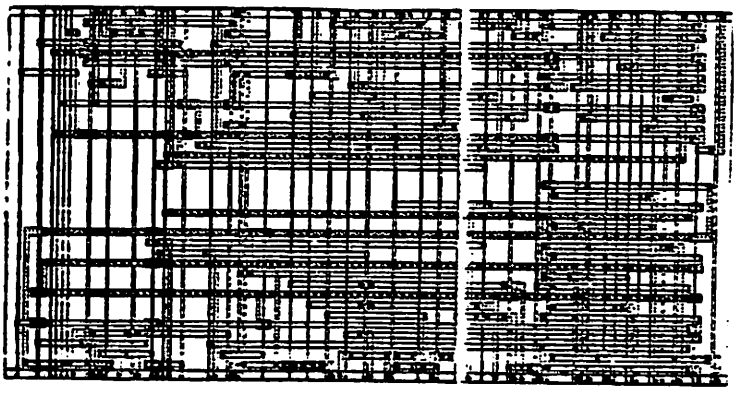


Figure 18

usage, track number and total length of critical nets by 2.1%, 2.9% and 7.1%, respectively, shown in fig.19.

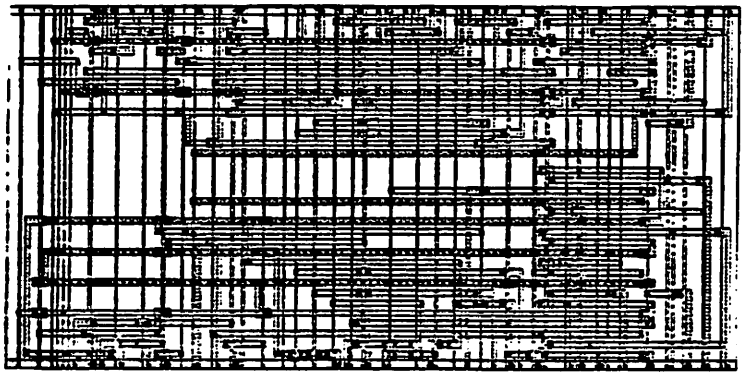


Figure 19

For a gate matrix, P/G nets are usually put at the top and the bottom, shown in

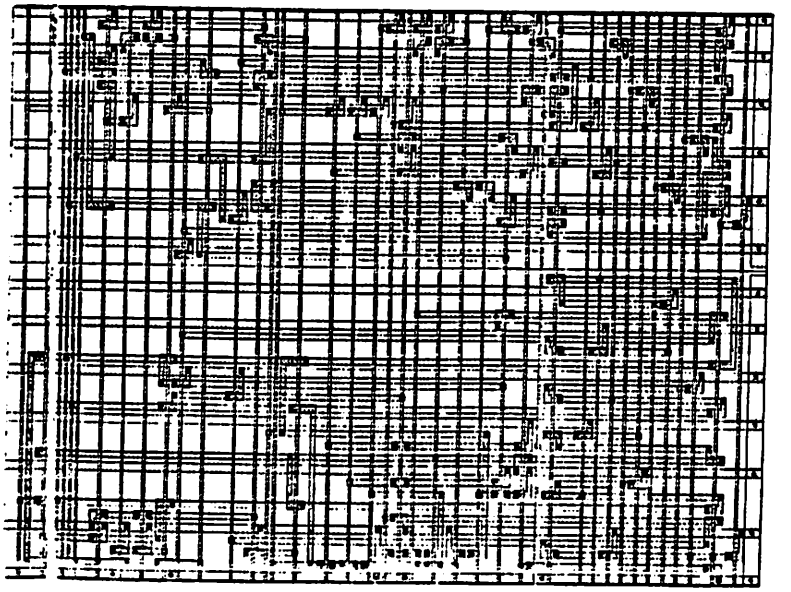


Figure 20

fig.18. The transistors are connected to P/G nets by vertical diffusion runs. If a transistor is far away from the P/G nets, a long vertical diffusion runs is necessary. The

circuits performance suffers a lot from the long vertical diffusion runs. GM program puts some P/G nets into gate matrix such that a limit to the maximum length of vertical diffusion runs is set. Fig.20 shows an improvement to fig.18. All the P/G nets in gate matrix are connected by second layer metal lines which run at the right edge of the gate matrix.

The program takes a circuit description at transistor level as its input data. For the four examples obtained from Wing and Huang, the results are shown in table 5. Our net assignment algorithm was able to complete all these examples while the overlaps of vertical diffusion runs over transistors did not take place.

Table 5

examples		adder	alu	itt1	itt2
# of transistors	P	14	158	59	242
	N	14	158	59	232
# of gates		7	70	41	162
# of nets	P	8	84	39	175
	N	8	105	46	196
net assignment		done	done	done	done
# of tracks		8	57	33	79

We also tested eight other practical circuits. GM went through all these examples. The results are shown in Table 6.

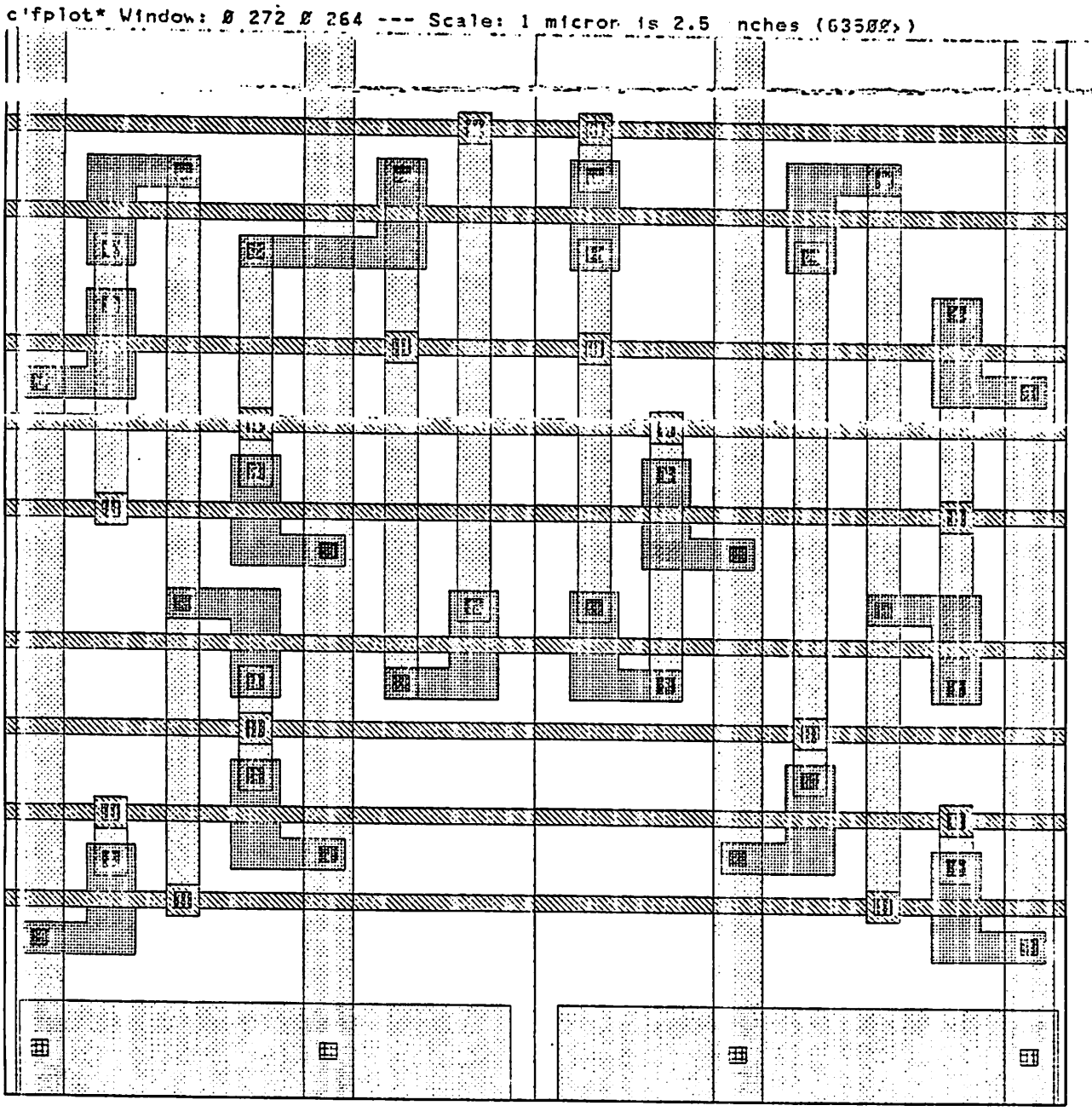
Table 6

examples		1bAdd	2bAdd1	2bAdd2	4bAdd	8bCmp	8bIdCmp	addCarry	decoder
# of transistors	P	19	36	38	103	50	59	69	17
	N	19	36	38	103	50	59	69	17
# of gates		15	26	24	60	35	44	41	12
# of nets	P	15	23	25	67	35	44	32	12
	N	19	30	30	77	27	36	50	16
net assignment		done	done	done	done	done	done	done	done
# of tracks		14	15	18	34	21	21	25	14

Here, we present some results in following pages:

Decoder

flip - flop



flip - flop

cifplot* Window: Ø 482 Ø 372 --- Scale: 1 micron is 1.4 inches (.0556")

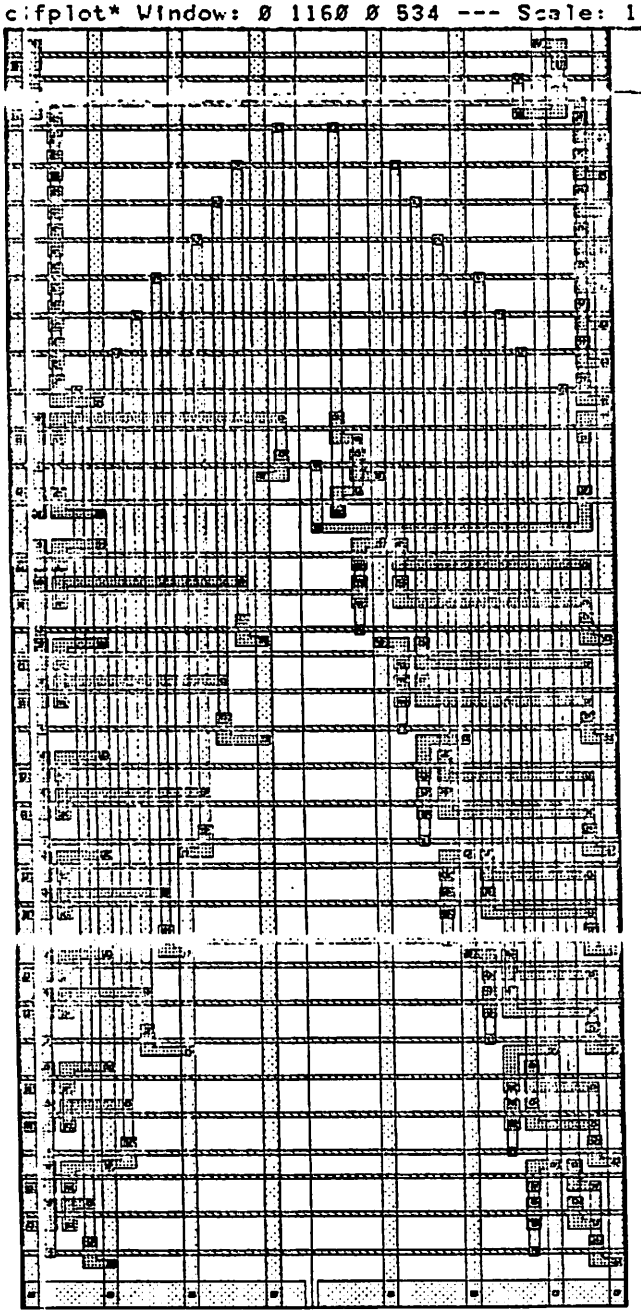
16 Adder

c fplot* Window: 0 772 0 408 --- Scale: 1 micron is 1 inches (25400x)

26 Adder 1

2b Adder 2

cifplot* Window: Ø 1160 Ø 534 --- Scale: 1 micron is Ø.6 inches (1524(x)

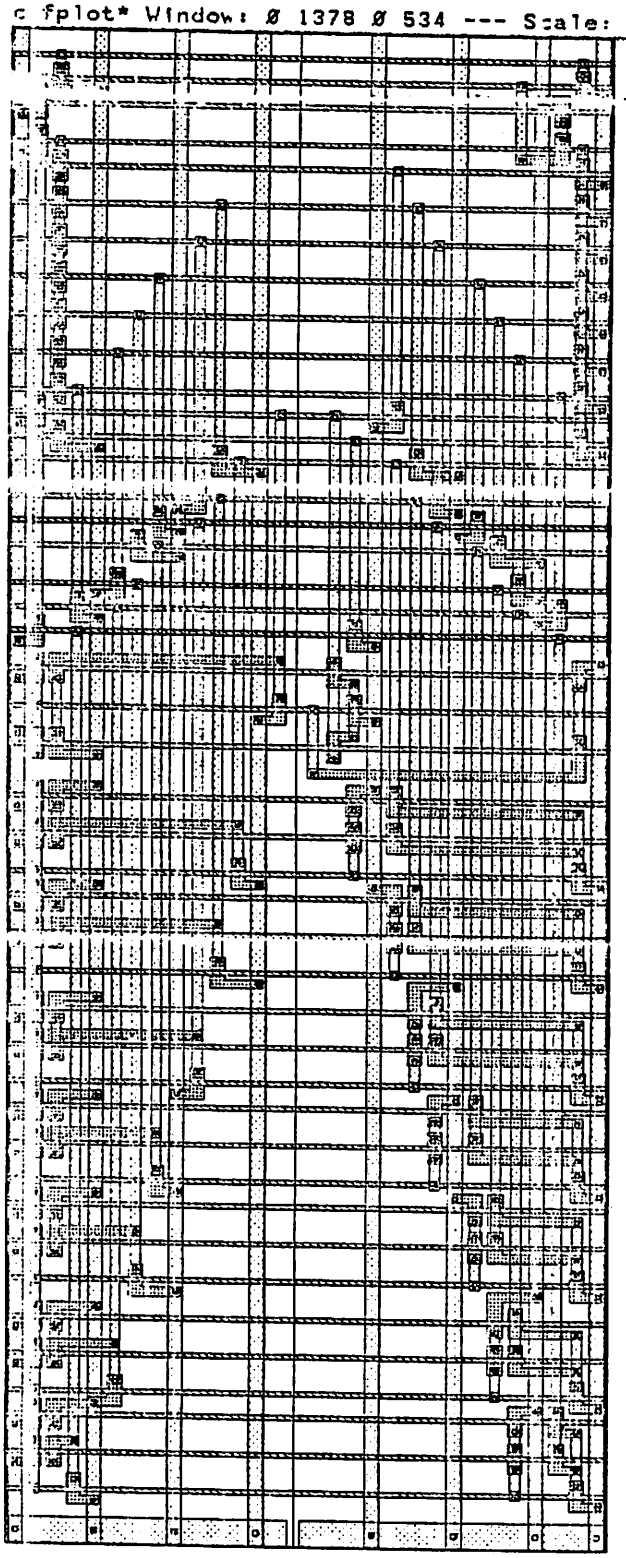


86 Cmp

c.fplot* Window: 0 1674 0 858 --- Scale: 1 micron is 0.5 inches (12700x)

4b Adder

c fplot* Window: Ø 1378 Ø 534 --- Scale: 1 micron is Ø.6 inches (1524fx)

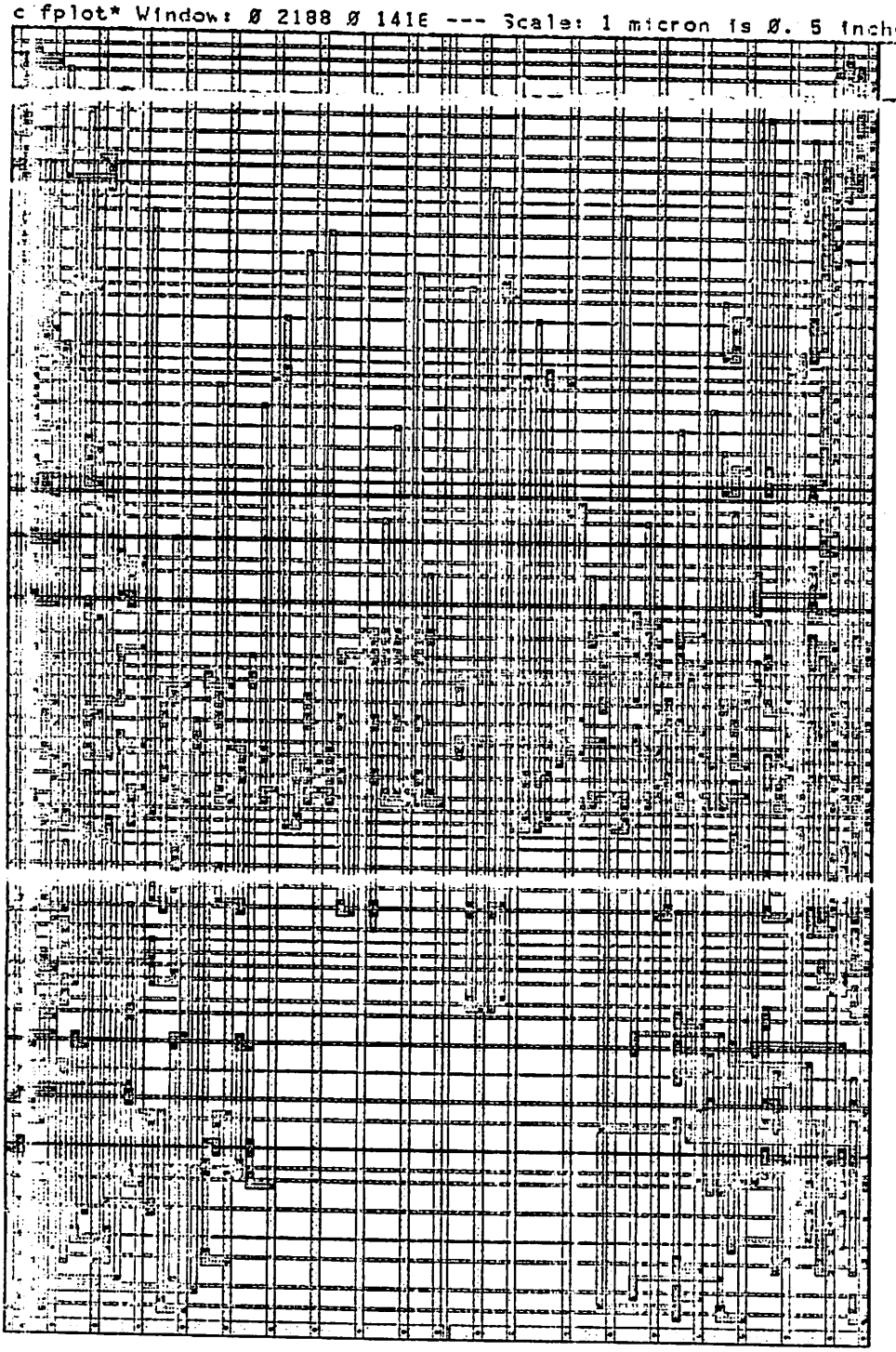


26Id Cwp

c:fplot* Window: 0 1120 0 642 --- Scale 1 micron is 0.6 inches (15240x)

add Carry

c fplot* Window: Ø 2188 Ø 1418 --- Scale: 1 micron is Ø. 5 inches (888Øx)



Alu

Chapter 6

Conclusions

In this report, a new gate matrix layout program GM is introduced. It can handle practical constraints such as oversized transistors and pre-assigned I/O pins. It allows users to do iterative improvement to the layout results, so as to make the critical nets be re-laid in the shortest possible distance. In order to improve the area usage and the aspect ratio, GM does not require the poly strips be equally spaced. Instead, it applies the horizontal compaction to the gate matrix such that the polys are placed as close as possible.

In this report, a modified gate arrangement algorithm to [12] was developed as a part of GM package. It can consider some practical constraints, such as the I/O constraints and critical net constraints. In addition, we formulate the net assignment problem as an extended one-dimensional assignment problem and propose a new net assignment algorithm. The tested results show that our algorithm has achieved good results.

REFERENCES

[1] 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.

[2] A. Weinberger, "Large-Scale Integration of MOS Complex Logic: A Layout Method", IEEE Journal of Solid-State Circuits, vol. SC-2, (1967), pp 182-190

[3] 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.

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

[5] O. Wing, Shuo Huang and Rui Wang, "Gate Matrix Layout", IEEE Trans. on CAD, vol. CAD-4, no. 3, July 1985, pp 220-231.

[6] M. Ishii, "GMMG -- A System for Generating CMOS Gate Matrices", report, Aug. 1st, 1983.

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

[8] 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.

[9] T. Ohtsuki, H. Mori, T. Kashiwabra, and T. Fujisawa, "On Minimal Augmentation of a Graph to Obtain an Interval Graph", Journal of Computer and System Sciences, vol.22, 1981, pp 60-97.

[10] D.R. Fulkerson and O.A. Gross, "Incidence Matrices and Interval Graph", Pacific J. Math, no. 3, 1965, pp 835-855.

[11] J.T. Li, "Algorithm for Gate Matrix Layout", Proc. 1983 IEEE ISCAS, Newport Beach, California, 1983, pp 1013-1016.

[12] D.M. Xu, Y.K. Chen, E.S. Kuh, and Z.J. Li, "A New Algorithm for Gate Matrix Layout", Proc. 1987 IEEE ISCAS, pp 288-291.

[13] T. Fujii, H. Horikawa, T. Kikuno and N. Yoshida, "A Heuristic Algorithm for Gate Assignment in One-Dimensional Array Approach", IEEE Trans. on CAD, vol. CAD-6, no. 2, March 1987, pp 159-164.

[14] O. Wing, "Interval-Graph-Based Circuits Layout", Proc. 1983 IEEE ICCAD, Santa Clara, California, Sept. 1983, pp 84-85.

[15] K. Nakatani, T. Fujii, T. Kikuno, and N. Yoshida, "A heuristic algorithm for gate matrix layout", IEEE ICCAD, Santa Clara, California, Nov. 1986, pp 324-327.

[16] T. Asano and K. Tanaka, "A Gate Placement Algorithm for One-Dimensional Arrays", Journal of Information Processing, vol. 1, no. 1, 1978, pp 47-52.

[17] T. Asano, "An Optimum Gate Placement Algorithm for MOS One-Dimensional Array", Journal of Digital Systems, vol. VI, pp 1-27, 1981.

[18] S. Huang, and O. Wing, "Improved Gate Matrix Layout", IEEE ICCAD, Santa Clara, California, Nov. 1986, pp 320-323.

[19] H.W. Leong, "A New Algorithm for Gate Matrix Layout", IEEE ICCAD, Santa Clara, California, Nov. 1986, pp 316-319.

[20] H. Yoshizawa, H. Kawanishi and K. Kani, "A Heuristic Procedure for Ordering MOS Arrays", Proc. 1975 DAC, pp 384-393.

[21] N. Deo, M.S. Krishnamoorthy, and M.A. Langston, "Exact and Approximate Solutions for the Gate Matrix Layout", IEEE Trans. on CAD, Jan. 1987.

[22] D.K Huang, W.K. Fuchs, and S.M. Kang, "An Efficient Approach to Gate Matrix Layout", IEEE ICCAD, Santa Clara, California, Nov. 1986, pp 312-315.

[23] D.K Huang, W.K. Fuchs, and S.M. Kang, "An Efficient Approach to Gate Matrix Layout", IEEE Trans. on CAD, vol. CAD-6, no. 5, Sept, 1987, pp 802-809.

[24] W. Shu, M.Y. Wu, and S.M. Kang, "Improved Net Merging Method for Gate Matrix Layout" IEEE Trans. on CAD, col.7, no. 9, Sept 1988, pp 947-951.

[25] S. Huang, and O. Wing, "Gate Matrix Partitioning", IEEE ICCAD, Santa Clara, California, Nov. 1988, pp 130-133.

[26] L. Ginneken, J. Eijndhoven, J. Browwers, "Doubly Folded Transistor Matrix Layout", IEEE ICCAD, Santa Clara, California, Nov. 1988, pp 134-137.

[27] S. Huang, and O. Wing, "Improved Gate Matrix Layout", Report, Columbia University.

[28] S. Rim, and K. Nakajima, "Net Assignment in Gate Matrix Layout", Proc. 1988 IEEE ISCAS, pp731-734.

[29] T. Yoshimura and E.S. Kuh, "Efficient Algorithms for Channel Routing", IEEE Trans. on CAD, vol. cad-1, no. 1, January 1982, pp25-35.

NAME

gm - A gate matrix layout generator program

SYNOPSIS

gm [-i] [host:display] inputfile [> outputfile]

DESCRIPTION

Gm is a program that performs the layout of CMOS circuits in gate matrix style. This layout style can treat placement and interconnection of transistors at the same time. The program is divided into two steps: gate assignment and net assignment. The gate assignment determines an optimal gate sequence, so as to minimize the number of tracks and the total length of nets. The net assignment assigns nets to tracks such that the total length of vertical diffusion runs, and the intersections of vertical diffusion runs over transistors are minimal.

Gm allows users to do interactive improvement on the layout result. This improvement can make some critical nets be laid in the shortest possible distance.

Gm takes a circuit description at transistor level as its input data. Its output results are given in CIF files. The program places the polysilicons of the gate matrix as close as possible if they do not violate the design rules. So the polysilicons of the final layout are not equally spaced.

INPUT/OUTPUT

A Circuit Description (CD) file is composed of a sequence of characters in a limited character set. The file contains input card, output card, gate sequence card, net sequence card, subcircuit cards and main circuit card. Each card is composed of sentances. The sentances are separated with simicolons.

The CD file is described using the standard notation proposed by Niklaus Wirth: production rules use equals = to relate identifiers to expressions, vertical bar | for or, and double quotes " " around terminal characters; curly brackets { } indicate repetition any number of times including zero; square brackets [ ] indicate optional factors (i.e. zero or one repetition); rules are terminated by period.

```
cdFile       = { subcircuit } [ input ] [ output ]
               [ gateSequence ] [ netSequence ] mainCircuit.
subcircuit   = subStart { { blank } { transistor } { subCall }
               semi } subEnd.
transistor   = t_name { blank } int { blank } int { blank }
               int { blank } int [ { blank } int ] type semi.
subCall      = s_name { blank } { int { blank } } name semi.
```

```
subStart      = "subckt" { blank } name { blank }
                { int { blank } } semi.
subEnd        = "ends" semi.
type          = "n" | "N" | "p" | "P" .
input         = inStart { { blank } pin { blank } } inEnd.
pin           = name { blank } int { blank } orientation
                [ { blank } int { blank } int ].
inStart       = "input" semi.
inEnd         = "endi" semi.
orientation   = "t" | "T" | "b" | "B" | "l" | "L" | "r" | "R" .
output        = outStart { { blank } pin { blank } } outEnd.
outStart      = "output" semi.
outEnd        = "endo" semi.
gateSequence  = gsStart { name { blank } } semi gsEnd.
gsStart       = "g_order" semi.
gsEnd         = "endg" semi.
netSequence   = nsStart { name { blank } } semi nsEnd.
nsStart       = "n_order" semi.
nsEnd         = "endn" semi.
mainCircuit   = mStart { { blank } { transistor } { subCall }
                semi } mEnd.
mStart        = "mainckt" semi.
mEnd          = "end."
blank         = " " | "        " .
semi          = { blank }  ";" { blank }.
digit         = "0" | "1" | "2" | ... ... | "9".
int           = [ "+" | "-" ] digit { digit }.
char          = "A" | "B" | ... ... | "Z" | "a" | "b" | ...
                ... | "z" .
name          = char { { char } { digit } }.
t_name        = "m" name.
s_name        = "x" name.
```

Gm considers upper characters exactly same as lower characters. Comments which are quoted within '{' and '}', can be inserted in any positions in CD files.

In CD files, subcircuit and main circuit are composed of transistors and subcircuit calls. A transistor is given in the following form:

```
NAME  DRAIN#  GATE#  SOURCE#  SUBSTRATE#  [ size ] TYPE;
```

For example, the following line represents a N-channel transistor, whose drain connected to the circuit node # 2, gate connected to node # 1, source connected to Vss, and the substrate connected to Vss. Its name is mtr (the first letter of transistor's name should be "m", and the first letter of subcircuit call should be "x").

```
mtr   2   1   0   0   n ;
```

Node number 0 has a special meaning. When it is used in the N part of CMOS circuit, it represents Vss. When it is used in the P part, it represents Vdd.

The input and output card are composed of "pin" sentances. The "pin" sentance starts with the port name, followed by circuit node number, orientation and a range. The orientation can be one of top, bottom, left, or right, by using 't', 'b', 'l' or 'r' in this position. For example, if a pin is placed at the top of the gate matrix, ranging from 7th column to 10th colomn, it is represented by:

```
pc   15   t   7   10;
```

For all I/O pins contained in input/output card, gm can place them according to a certain sequence which is given by users. An example of gate sequence card is given as follow. It indicates that the pin "pa" should be placed at the left of "pb", and "pb" at the left of "pc". (The net sequence card "n_order" is still under development).

```
g_order;
pa  pb  pc;
endg;
```

The outputs of gm are CIF file (.cif), transistor list, gate assignment result and other stuffs. The CIF file can be displayed by "xkic" command on a graphic terminal. Also it can be ploted by "cifplot" on the Versatec electrostatic printer. The transistor list describes the connection of transistors of entire circuit. Each transistor is listed with the format which is similar to the description in the CD file.

An example of CD file "eor" is shown as below:

```
{ This is an EOR circuit. }

subckt pass 4 1 2 3;
mn 3 1 4 0 N;
mp 3 2 4 0 P;
ends;

SUBCKT inv 2 1;
mn 2 1 0 0 N;
mp 2 1 0 0 p;
ENDS;

input;
pa 2 t 1 1 ;
pb 1 b 3 4;
endi;
```

```
output;
ps 4 1;
endo;

g_order;
pa pb;
endg;


mainckt;
xinv 3 1 inv;
xpass 4 3 1 2 pass;
mp3 4 2 1 0 p;
mn3 4 2 3 0 n;
end.
```

The output of "eor" is shown below:

```
the number of P_transistors is 3
the number of N_transistors is 3
-----P_transistors-----
name   D   G   S   B   type  width   t_number
xinv_mp   3   1   0   0   p   0   2
xpass_mp   2   1   4   0   p   0   4
mp3   4   2   1   0   p   0   5
-----N_transistors-----
name   D   G   S   B   type  width   t_number
xinv_mn   3   1   0   0   n   0   1
xpass_mn   2   3   4   0   n   0   3
mn3   4   2   3   0   n   0   6
The number of gates is 4
The number of P nets is 4
The number of N nets is 3
fillins=7
The number of tracks needed in P part is 4
The number of tracks needed in N part is 3
The total number of tracks is 7
The sequence of gates
2 3 0 1
gate arrangement time is 0.016667 seconds.
0 nets are merged in P part
0 nets are merged in N part
Real number of P nets is 4.
Real number of N nets is 3.
Adjustment of vertical diffusion runs in P part
Adjustment of vertical diffusion runs in N part
P nets assignment
P part finally height is 4
N nets assignment
N part finally height is 3
```

```
The net assignment is completed
Area usage: 174*112=19488
This layout needs 0.250000 seconds
sum time is 0.433333 seconds
```

For this example, the output of CIF data are stored in "eor.cif" file.

OPTIONS

-i is used to select an iterative and interactive mode. For the obtained layout result, gm can make some critical nets become shorter if the users consider these nets are too long. Gm first asks if you satify with the layout result. If the answer is no, it will query the user for further information. In this step, what the users need to do, only point out the critical nets and their weights (>=1). After all the critical nets are given, the users have to answer "end". Then a new improved layout will be produced. Also the reduction (or increase) of the area, track number and total length of critical nets are given by gm. This improvement will keep on going until the users stop it.

An example of this interactive mode is shown as below:

```
Do you satify with your layout result(y/n)? n
Please point out the nets you want to modify.
input track number (#/'end'): 4
all P nets on track 4 are listed below:
10 54
which net (#/'q')? 10
weight: 2
which net (#/'q')? q
all N nets on track 4 are listed below:
77
which net (#/'q')? 77
weight: 2
which net (#/'q')? q
input track number (#/'end'): 6
all P nets on track 6 are listed below:
40 18 46 68
which net (#/'q')? q
all N nets on track 6 are listed below:
33
which net (#/'q')? 33
weight: 2
which net (#/'q')? q
input track number (#/'end'): end
```

host:display
Normally, gm gets the host and display number from the environment variable "DISPLAY". One can, however specify them explicitly. The host specifies which machine to draw the layout result on, and the display argument specifies the display number. For example, "petrus:1" draws the layout result on display one on the machine "petrus".

DIAGNOSTICS
The input routine gives out error messages in case of wrong specification of the CD files.

SEE ALSO

FILES

/users/dongmin/GM/gm	gate matrix layout program.
/users/dongmin/GM/manual	the manual of "gm".
/users/dongmin/GM/CD/*.in	some CD (input) files.
/users/dongmin/GM/OUT/*.out	the output of above CD files.
/users/dongmin/GM/CIF/*.cif	the CIF data of above CD files.

All the above files are on the machine "petrus".

AUTHER
Dong-Min Xu

BUGS
send bug reports to "dongmin@esvax"