模拟集成电路二维 Stack 生成及模块合并算法*

刘 锐[1,2], 董社勤[2], 洪先龙[2], 龙 迪[2], 顾 钧[3]

[1](中国科学院 软件研究所,北京 100080)

[2](清华大学 计算机科学与技术系,北京 100084)

[3](香港科技大学 计算机系,香港)

Two-Dimensional Stack Generation and Block Merging Algorithms for Analog VLSI

LIU Rui[1,2], DONG She-Qin[2], HONG Xian-Long[2], LONG Di[2], GU Jun[3]

[1](Institute of Software, The Chinese Academy of Sciences, Beijing 100080, China)

[2](Department of Computer Science and Technology, Tsinghua University, Beijing 100084, China)

[3](Department of Computer Science, Science and Technology University of Hong Kong, Hong Kong, China)

+ Corresponding author: E-mail: rliu_ac@hotmail.com, http://www.iscas.ac.cn

Received 2002-10-18; Accepted 2003-07-16

Liu R, Dong SQ, Hong XL, Long D, Gu J. Two-Dimensional stack generation and block merging algorithms for analog VLSI. *Journal of Software*, 2004,15(5):641~649.

http://www.jos.org.cn/1000-9825/15/641.htm

Abstract: In analog VLSI design, 2-dimensional symmetry stack and block merging are critical for mismatch minimization and parasitic control. In this paper, algorithms for analog VLSI 2-dimensional symmetry stack and block merging are described. Several theoretical results are obtained by studying symmetric Eulerian graph and symmetric Eulerian trail. Based on them, an $O(n)$ algorithm for dummy transistor insertion, symmetric Eulerian trail construction and 2-dimensional symmetry stack construction is developed. The generated stacks are 2-dimensional symmetric and common-centroid. A block merging algorithm is described, which is essentially independent of the

* Supported by the National Nature Science Foundation of China under Grant Nos.90307005 and 60121120706 (国家自然科学基金); the National Nature Science Foundation of China and Research Grants Council of Hong Kong joint Project under Grant No.60218004 (国家自然基金与香港研究资助局联合资助); the National Natural Science Foundation of USA (NSF) under Grant No.CCR-0096383 (美国国家自然科学基金); the National High-Tech Research and Development Plan of China under Grant No.2002AA1Z1460 (国家高技术研究发展计划(863))

LIU Rui was born in 1974. He is a Ph.D. candidate at the Institute of Software, Chinese Academy of Sciences. His research areas are algorithm and analog VLSI CAD. **DONG She-Qin** was born in 1964. He is an associate professor at the Department of Computer Science and Technology, Tsinghua University. His research areas are optimization algorithm and VLSI physical design. **Hong Xian-Long** was born in 1940. He is a professor and doctoral supervisor at the Department of Computer Science and Technology, Tsinghua University. His research areas are algorithm and CAD system. **LONG Di** was born in 1980. He is PhD candidate at the Department of Computer Science and Technology, Tsinghua University. His research interests focus on analog layout automation. **GU Jun** is with the Department of Electrical and Computer Engineering, University of Calgary, Alberta T2N 1N4, Canada, and is visiting the Department of Computer Science, Hong Kong University of Science and Technology, Hong Kong, China. His research areas are SAT problem, optimization algorithm and VLSI CAD.

topological representation. Formula for calculating the maximum block merging distance is given. Experimental results show the effectiveness of the algorithms.

Key words: analog VLSI; 2-dimensional stack; block merging; symmetric Eulerian graph

摘 要: 在模拟集成电路设计中,关于 X 轴和 Y 轴同时对称的 Stack,以及模块之间的合并,对于增加器件之间的匹配和控制寄生是至关重要的.描述了模拟集成电路二轴对称 Stack 生成算法和模块合并算法.通过对于对称欧拉图和对称欧拉路径的研究,得出了多项理论结果.在此基础上,提出了时间复杂度为 $O(n)$的伪器件插入算法、对称欧拉路径构造算法和二轴对称 Stack 生成算法.生成的 Stack,不但关于 X 轴和 Y轴对称,而且具有公共质心(common-centroid)的结构.还描述了模块合并算法,给出了计算最大合并距离的公式.该算法本质上是独立于任何拓扑表示的.实验结果验证了算法的有效性.

关键词: 模拟集成电路;二维 Stack;模块合并;对称欧拉路径

中图法分类号: TP302 **文献标识码**: A

1 Introduction

Analog blocks typically constitute only a small fraction of components on mixed-signal ICs and emerging systems-on-a-chip (SoC) designs. But due to the increasing levels of integration available in silicon technology and the growing requirement for digital systems to communicate with the continuous-valued external world, there is a growing need for quality of analog integrated circuits. Although many efforts have been made in this field, the analog layout is still a hard and time consuming task which has a considerable impact on circuit performance. Asymmetries and device mismatch can easily upset the critical precision of component, and together with the parasitics associated with the interconnections they can introduce intolerable performance degradation. Device merging, i.e. placing devices such that diffusion geometry is shared between electrically connected devices, is a very important technology for analog VLSI layout to dramatically reduce the parasites as well as area occupation. Merging a series transistor is called *stacking*.

Many researches have been done to explore the device merging optimizations during the analog layout in the past years. KOAN/ANAGRAM II[1~3] keeps the macro cell style, and merges the common area when the nodes are exactly placed together during the simulated annealing. This kind of approach not only puts a heavy burden on the simulated annealing, but also can not achieve the interdigitated and common-centroid structure[4], which is often used by manual layout. The stack generator in Ref.[5] generates optimum stacks that satisfy the performance constraints, using a path partitioning algorithm. However, because it attempts to enumerate all optimal stacks, runtime can be extremely sensitive to the size of the problem. Symmetry and matching constraints can greatly prune the search, but the basic algorithm has exponential time complexity. Reference [6] gives an approach using simulated annealing to randomly generate a stack. An algorithm with $O(n)$ is present in Ref.[7], which can generate optimal stack without symmetry constraints, or an approximate solution under symmetry constraints. Based on the previous approach, Ref.[8] proposes parasitic and mismatch models for the basic analog circuit blocks such as differential pair, current mirror, and cascade.

However, there exist two problems. First, they can only generate a 1-D stack but not a 2-axial symmetry stack. 1-D stacks are sometimes slim and long, which is not desirable during placement. Furthermore, Refs.[9,10] have shown that the 2-axial symmetry minimizes the mismatch and cancels out press, process and thermal gradients in every direction. So, a 2-axial stack is more desirable. Second, during the stacking, only transistors with similar channel width can be generated in one stack. The different stacks can not be merged during placement.

During placement, several non-slicing topological representations can be used, which include Sequence

Pair[11,12], BSG[13,14], O-tree[15], and Corner Block List (CBL)[16], etc. Among these representations, some analog VLSI constraints such as matching and symmetry[17,18] and other useful constraints[19~21] have been implemented.

In this paper, we first present an $O(n)$ algorithm to generate 2-axial symmetry stacks. Several stacks with different ratios are generated for one group of transistors. These stacks can be chosen during the placement. Then, an algorithm for block merging is given, which can be used to merge stacks with different channel width transistors. The block merging algorithm is implemented on Sequence Pair, but it is essentially independent of any topology representation. This paper is organized as follows. The 2-axial symmetry stack generation algorithms are described in Section 2. Section 3 presents the block merging algorithms. Section 4 gives the experimental results and finally Section 5 concludes the paper.

2 Stack Generation Algorithms

2.1 Basic stacking strategy

Circuit schematic should be modeled in a format suitable for a graph algorithm to solve the layout problem effectively. Our strategy is similar to that introduced in Refs.[1,7]:

(1) Divide the circuit into *partitions* with respect to the device type and bias node (body node in MOS transistors).

(2) Perform device *folding*: split large transistors into smaller parallel transistors.

(3) Perform further partitioning to reduce the variation on the module widths in a partition, only the transistor with the same width is in the same partition. If there are self symmetry devices, further partition must apply to isolate the self symmetry devices from other devices because only the partition that is fully symmetric and has not self-symmetry edge can generate common-centroid 2-axial symmetry stacks.

(4) *Generate* stacks that implement each partition.

In analog CMOS circuits, as in digital standard-library leaf-cells, only transistors of the same type (e.g. NMOS), which share a common well, can be stacked (i.e. their common diffusion nodes can be merged in the layout to minimize the diffusion area). In addition, in analog circuits, it is fairly common to have transistors of the same type which require distinct body potentials, for example, to optimize the noise performance. Such transistors have their own isolated wells and cannot be stacked with other transistors of the same type. Therefore in the first step, we put such transistors in different partitions. We also allow the designer to specify explicitly to have two or more transistors in the same stack.

In the second step, large transistors are folded into fingers to minimize the diffusion capacitances as well as to balance the aspect ratio of the resulting module. This can be done either automatically[3] or manually by the designer. It is important to note that, in this stacking strategy, transistor folding is done *a priori*. The stack generation algorithm is given the fixed-width modules as input – it does not dynamically fold transistors. This is in contrast to tools such as KOAN[1~3], in which the overall optimization loop treats stacking, folding and placement simultaneously.

In the third step, the partitions are examined again to account for variations in module widths. Only the transistors with the same width can be put in the same partition. Furthermore, the symmetrical constraints are examined. The result of partition should be fully symmetrical and the self symmetrical parts are isolated.

In the fourth step, the 2-dimensional stack generation algorithms (Subsections 2.2 and 2.3) operate on each circuit partition separately. Several stacks of different ratios are generated for each partition. Choosing one of the stacks with different ratios for the same partition is added to move the set of the simulated anneal during placement.

2.2 Finding symmetric Eulerian trail

For the partition $G=(V,E)$ that is fully symmetry, the first step to generate the 2-axial symmetry stacks is to construct the symmetry Eulerian graph by adding dummy edges. At most one self symmetric dummy edge can be added. Based on this symmetry Eulerian graph, symmetric Eulerian trail can be built. Now, we give some definitions.

Definition 1. A *self symmetric vertex* is one on the symmetric axle whose symmetric vertex is itself (Fig.1(a)).

Definition 2. A *self symmetric edge* is an edge to which the two vertices associated are symmetric to each other. And this edge crosses the symmetric axle and whose symmetric edge is itself (Fig.1(a)).

Definition 3. A *symmetric Eulerian trail* $(v_1,v_2,\ldots,v_n)$ is a trail symmetrically covering the graph, i.e., in the trail, v_i and v_{n+1-i} are symmetric to each other.

From Definition 3, we can easily get the following lemma that is useful during the Eulerian trail construction.

Lemma 1. Given the *Symmetric Eulerian trail* $(v_1,v_2,\ldots,v_n)$, if n is an odd number, there are $n-1$ edges in the trail and there is no self symmetric edge. If n is an even number, $(v_{n/2+1},v_{n/2+2})$ is a self symmetric edge.

Now we show how to construct a symmetric Eulerian graph based on the fact that a symmetric Eulerian trail can be built. We first find all the vertices with odd degree. The *degree* of a vertex is the number of edges incident to it. For each odd degree vertex, there must be a symmetric vertex that is also odd degree. The following Lemma 2 shows self symmetric vertex must be of even degree. Therefore, we only need to find out all the odd degree vertices on one side of the symmetric axle. All and only their symmetric vertices are also of odd degree on the other side of the symmetric axle.

Lemma 2. Given a symmetry graph $G=(V,E)$ and there is no *self symmetric edge*, if there are some self symmetric vertices, their degree must be even.

Proof. The proof is straightforward. Since there is no self symmetric edge, each edge incident to the self symmetric vertex must have a symmetric one incident to the same vertex. So, the degree of self symmetry vertices must be even. □

If the number of odd degree vertices on each side of the symmetry axle is even, two methods can be applied.

Method-1: After connecting these odd degree vertices by adding dummy edges pair by pair on each side, there will be no odd degree vertex left in the partition (Fig.1(b)). Because all vertices are even degree and each edge on one side of the symmetric axle has a symmetric edge on the other side, a symmetry Eulerian trail can be constructed symmetrically and synchronously on both sides of the axle. And there are even edges in the trail.

Method-2: Another method to connect these odd degree vertices is as follows. We connect these vertices pair by pair just as the previous method does except the last two pairs. For the last two pairs of odd degree vertices, we connect an odd degree vertex with its symmetric one by a self symmetric dummy edge and leave the other odd degree symmetry pair unconnected (Fig.1(c)). Let the self symmetry edge be $(v_{n/2+1},v_{n/2+2})$ in the symmetric Eulerian trail. Then, the two vertices $v_{n/2+1}$ and $v_{n/2+2}$ are used as the start vertex on each side of the axle to construct the symmetric Eulerian trail. There are odd edges in the trail.

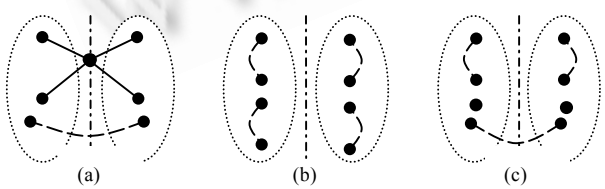


Fig.1 (a) Self symmetric vertex and self symmetric edge; (b) All the odd degree vertices are connected pair by pair; (c) Two odd degree vertices leaved not connected.

Because previous two methods can generate trails with different number of edges, we generate both trails to be used by future stack generation.

On the other hand, if the number of odd

degree vertices on each side of symmetric axle is odd, there are also two methods which can be applied.

Method-3: After connecting them pair by pair, there will be one odd degree vertex leaved unconnected on each side of the symmetric axle, which is symmetric to each other (Fig.2(a)). The following theorem guarantees that there must exist a self symmetric vertex. Therefore, we use the self symmetric vertex as the start vertex to construct symmetric Eulerian trail symmetrically and synchronously on both sides of the symmetrical axle. The trail has even edges.

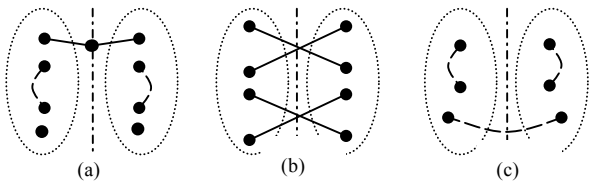


Fig.2 (a) One odd degree vertex leaved unconnected on each side; (b) Edges crossing symmetry axis; (c) Connecting the last odd degree vertex with its symmetric vertex by a self symmetric dummy edge.

Theorem 1. Given a symmetric graph $G=(V,E)$, if there is no *self symmetric edge* and the number of odd degree vertices on each side of the symmetric axis is odd, there exists at least one *self symmetric vertex*.

Proof. Suppose for the sake of contradiction that there is one vertex that is not *self symmetric*, i.e. each vertex has a symmetric one on the other side of symmetry axis.

Since there is neither self symmetric edge nor self symmetric vertex, we can separate the graph into two sub graphs by removing the edges that cross the symmetry axis (Fig.2(b)). While removing one pair of the crossing edges, two vertices' degrees change their parity from odd (even) degree to even (odd) degree. Therefore, the parity of the odd degree vertex number on each side of symmetric axle keeps unchanged. Then, in each sub graph, the number of odd degree vertices is still odd, contradicting the corollary of Eulerian theorem that the number of odd degree vertices is even. □

Method-4: After connecting them pair by pair as done previously, we connect the last odd degree vertex with its symmetric vertex by a self symmetric dummy edge (Fig.2(c)). Let the self symmetry edge be $(v_{n/2+1},v_{n/2+2})$ in the symmetric Eulerian trail, then the two vertices $v_{n/2+1}$ and $v_{n/2+2}$ are used as the start vertex on each side of the axle to construct the symmetric Eulerian trail. There are odd edges in the trail.

The following procedure gives an explicit description of the previous processes.

Procedure BuildSymmetricEulerianTrail
Begin
Find all odd degree vertices on one side of the axle;
If the number of odd degree vertices is even
// symmetric Eulerian Trail-1 *has even edges*
Use method-1 to construct symmetric Eulerian Trail-1;
// symmetric Eulerian Trail-2 *has odd edges*
Use method-2 to construct symmetric Eulerian Trail-2;
Else if the number of odd degree vertices is odd
// symmetric Eulerian Trail-1 *has even edges*
Use method-3 to construct symmetric Eulerian Trail-1;
// symmetric Eulerian Trail-2 *has odd edges*
Use method-4 to construct symmetric Eulerian Trail-2;
End;

2.3 2D stack generation

We construct several common centroid stacks with different ratios for each partition. These stacks can be selected by simulated annealing placement algorithm. The upper and lower bounds of the stack ratios can be specified by the user.

In the following procedure, one row stack is constructed first and then two row stacks and so on, until the ratio limits are reached. The trail (trail-1 or trail-2) that can be divided exactly by the row number is used to construct the stack.

Procedure ConstructStack
Begin
row=1;
length-1=edge number of trail-1;
length-2=edge number of trail-2;
while stack ratios do not reach the limits
    if row is even and length-1 can be divided exactly by row.
Construct stack as Fig.3(a) using Trail-1;
    else if row is odd
        if length-2 can be divided exactly by row
Construct stack as Fig.3(b) using Trail-2;
        if length-1 can be divided exactly by row
Construct stack as Fig.3(c) using Trail-1;
row++;
End.

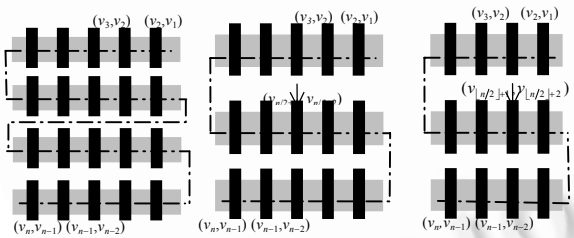


Fig.3 The dash-dotted line represents the Eulerian trail. (a) Even row stack; (b) Odd row odd column stack; (c) Odd row and even column stack.

From the procedure and definition of the symmetric Eulerian trail we can see the centroid of the symmetric devices is at the center of the stack. The only possible self symmetric edge $(v_{n/2+1},v_{n/2+2})$ appears at the center of the middle row (Fig.3(b)).

There are still some other tricks we have used in our algorithms. Edges including dummy edges incident to the same two vertices are mutually interchangeable. We do not decide which one is chosen during constructing the trail. Decision is postponed to the stack generation. During stack generation, selection of the dummy edge or real device edge is decided dynamically based on its position in the stack. If the current position is the end of a stack row, a dummy one is preferred[22].

3 Block Merging Algorithms

Block merging is to further explore the possibility of geometric share between different stacks during placement. It reduces not only the area but also the parasites.

The block merging has following two main steps which occur during the packing.

(1) For each module all the merging candidate modules are found.

(2) The maximum merging distance is calculated, according to which the module position is adjusted.

In first step, because each module can only be merged to its left and bottom, only that left or below module a is a merging candidate for it. For module a and arbitrary module b,

if $X(a)<X(b)+Width(b)$ and $X(b)<X(a)+Width(a)$, b is a's bottom merging candidate.

if $Y(a)<Y(b)+Height(b)$ and $Y(b)<Y(a)+Height(a)$, b is a's left merging candidate.

In second step, the maximum merging distance is the length that the module can horizontally (vertically) merge to the left (bottom). Each module can be represented by two rectangles (Fig.4). One is the merging area m, and the other is the solid area$s$ that can not be merged. w_1 and w_4 are widths of the solid areas. w_2 and w_3 are widths of the merging areas. d_1 is the distance between the merging areas. d_2 is the distance between the solid areas. d_3 and d_4 are distances between the merging area and solid area. When module-2 is horizontally merged to left, if the two areas never touch, the distance between them is set to be infinite, for example, d_2 in Fig.4(b) is infinite. Figure 4 shows the case of a horizontal merging to the left. The vertical merging is similar.

$$d_1=\begin{cases}X(m_2)-X(m_1)+Width(m_2) & \text{If } Y(m_2)<Y(m_1)+Height(m_1) \text{ and } Y(m_1)<Y(m_2)+Height(m_2)\\ +\infty & \text{Others}\end{cases}$$

$$d_2=\begin{cases}X(s_2)-X(s_1)+Width(s_2) & \text{If } Y(s_2)<Y(s_1)+Height(s_1) \text{ and } Y(s_1)<Y(s_2)+Height(s_2)\\ +\infty & \text{Others}\end{cases}$$

$$d_3=\begin{cases}X(m_2)-X(s_1)+Width(m_2) & \text{If } Y(m_2)<Y(s_1)+Height(s_1) \text{ and } Y(s_1)<Y(m_2)+Height(m_2)\\ +\infty & \text{Others}\end{cases}$$

$$d_4=\begin{cases}X(s_2)-X(m_1)+Width(s_2) & \text{If } Y(s_2)<Y(m_1)+Height(m_1) \text{ and } Y(m_1)<Y(s_2)+Height(s_2)\\ +\infty & \text{Others}\end{cases}$$

Theorem 2. The maximum merging distance is

$$D_m=\min(\min(w_2,w_3)+d_1,d_2,d_3,d_4) \quad (1)$$

Proof. According to Fig.5, we can justify the theorem for each situation. For Fig.4(a), the merged result is shown in Fig.5(a) and the maximum merging distance is w_2+d_1 when $w_2<w_3$, or w_3+d_1 when $w_2>w_3$. Figure 5(b) is the result of Fig.4(b) and the merging distance is w_2+d_1 when $w_2<w_3$, or w_3+d_1, when $w_2>w_3$. Figure 5(c) is the result of Fig.4(c) and the merging distance is d_3, i.e. the blocks can only be compacted, but can not be merged. There are still other similar situations. They can all be enumerated and the correction of the theorem can be verified. □

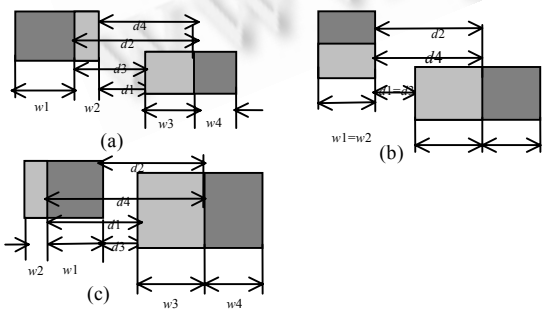


Fig.4 The light rectangle in each module represents the merging area. The dark rectangles represent the solid areas

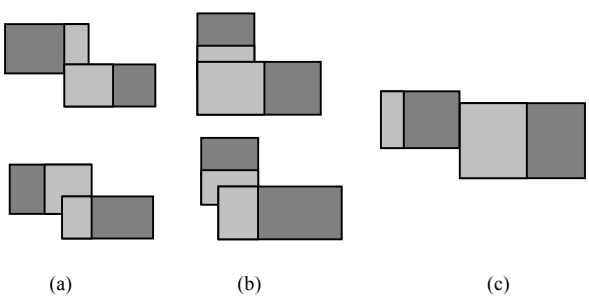


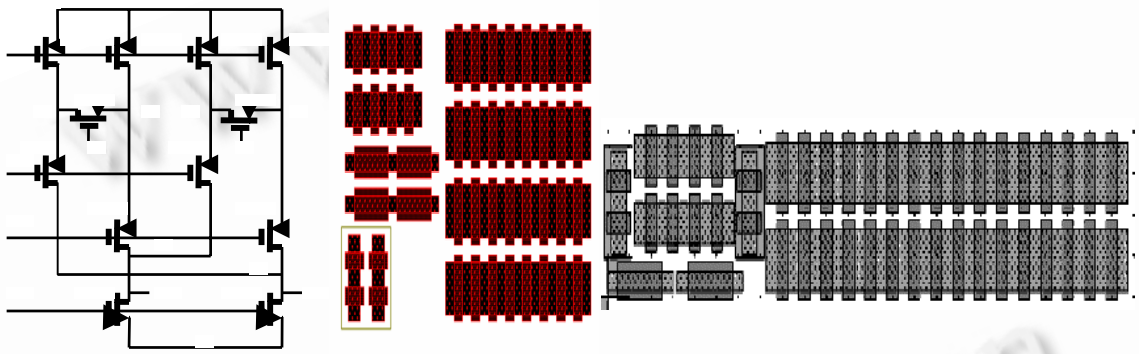
Fig.5 The merged results of Fig.4

Our algorithms are implemented based on sequence pair. However, they are essentially independent of any topological representations. Thai is to say, it can be fairly easy to migrate our algorithms to any other topological representations.

4 Results

Figure 6(a) shows a multiplier circuit[23] which has been used as a benchmark in several previous researches[1~3,7]. The number of stacks generated is 4. All these stacks are 2-axial symmetric and common-centroid (Fig.6(b)). In Fig.6(c), layout with only 1D stacks is given as a comparable result, which is also generated by our tool. We can see the stacks are slim and long, and the satisfaction of the symmetry constraints depends on the placement.

The test cases for the block merging algorithm are based on the MCNC benchmark, and the merging areas are designated randomly. Figure 6 and Table 1 give the results of the test case based on ami33 and ami49. The merged areas have been circled in Fig.7. From Table 1, we can see the area usages have been dramatically increased while the time consuming is still in the acceptable scope.



(a) The Mult circuit (b) Generated stacks (c) The layout with only 1D stacks

Fig.6 Experiment results

Table 1 Experimental results of test cases

Test case	Without merging			With merging		
	Area	Usage (%)	Time (s)	Area	Usage (%)	Time (s)
Ami33	1 245 972	92.82	44.23	1 157 870	99.88	80.17
Ami49	37 239 020	95.18	82.12	36 862 560	96.16	176.56

5 Conclusions

In this paper, we propose algorithms to generate 2-axial symmetric stacks in linear time. Several stacks with different ratios are generated for one group of transistors. These stacks are chosen during the placement. Then, the algorithm for block merging is given, which can be used to merge stacks with different channel width transistors during placement. The algorithms have been proved to be efficient by the experimental results.

References:

[1] Cohn JM, Garrod DJ, Rutenbar RA, Carley LR. KOAN/ANAGRAM II: New tools for device-level analog placement and routing. IEEE Journal of Solid State Circuits, 1991,26(3):330~342.

[2] Cohn J, Garrod D, Rutenbar R, Carley LR. Analog Device-Level Layout Generation. Norwell: Kluwer, 1994.

[3] Garrod D, Rutenbar R, Carley LR. Automatic layout of custom analog cells in ANAGRAM. In: Proc. of the ACM/IEEE Int. Conf. Computer-Aided Design (ICCAD). Nov. 1988. 544~547.

[4] Gatti U, Maloberti F, Liberali V. Full stacked layout of analog cells. In: Proc. of the IEEE Int'l Symp. on Circuits and Systems. 1989. 1123~1126.

[5] Malavasi E, Pandini D. Optimum CMOS stack generation with analog constraints. IEEE Trans. on Computer-Aided Design, 1995,14:107~122.

[6] Arsintescu GB, Spanoche SA. Global generation of MOS transistor stacks. IEEE EuroDAC, 1996.

[7] Basaran A, Rutenbar RA. An $O(n)$ algorithm for transistor stacking with performance constraints. In: Proc. of the IEEE/ACM DAC. 1996. 221~226.

[8] Zeng X, Li MY, Zhao WQ, Tang PS, Zhou D. Parasitic and mismatch modeling for optimal stack generation. In: Proc. of the IEEE ISCAS 2000, Geneva, Switzerland, May 23-31, 2000.

[9] Bastos J, Steyaert M, Graindourze B, Sansen W. Matching of MOS transistors with different layout styles. In: Proc. of the IEEE Int'l Conf. on Microelectronic Test Structures. 1996. 17~18.

[10] Bastos J, Steyaert M, Graindourze B, Sansen W. Influence of die bonding on MOS transistor matching. In: Proc. of the IEEE Int'l Conf. on Microelectronic Test Structures. 1996. 17~31.

[11] Murata H, Fujiyoshi K, Nakatake S, Kajitani Y. Rectangle-Packing-Based module placement. In: Proc. of the IEEE Int'l Conf. Computer-Aided Design. 1995. 472~479.

[12] Murata H, Fujiyoshi K, Nakatake S, Kajitani Y. VLSI module placement based on rectangle-packing by the sequence pair. IEEE Trans. Computer-Aided Design, 1996,15(11):1518~1524.

[13] Nakatake S, Fujiyoshi K, Murata H, Kajitani Y. Bounded-Slicing structure for module placement. Technical Report, Institute of Electronics, Information, and Communication Engineers (IEICE). 1994. 19~24.

[14] Nakatake S, Fujiyoshi K, Murata H, Kajitani Y. Module placement on BSG-structure and IC layout applications. In: Proc. of the IEEE Int'l Conf. Computer-Aided Design, 1996. 484~491.

[15] Guo PN, Cheng CK, Yoshimura T. An *O*-tree representation of non-slicing floorplan and its applications. In: Proc. of the 36th ACM/IEEE Design Automation Conf. 1999. 268~273.

[16] Hong XL, Huang G, *et al*. Corner block list: An effective and efficient topological representation of non-slicing floorplan. In: Proc. of the IEEE Int'l Conf. Computer-Aided Design 2000. 2000.

[17] Balasa F, Lampaert K. Symmetry within the sequence-pair representation in the context of placement for analog design. IEEE Trans. on Computer-Aided Design and System, 2000,19(7).

[18] Pang YX, Balasa F, Lampaert K, Cheng CK. Block placement with symmetry constraints based on the *O*-tree non-slicing representation. In: Proc. of the IEEEDAC 2000. 2000. 464~467.

[19] Liu R, Hong LX, Dong SQ, Gu J. VLSI/PCB placement with predefined coordinate alignment constraint based on sequence pair. In: Proc. of the IEEE ASICON 2001. 2001. 167~170.

[20] Liu R, Hong XL, Dong SQ, Gu J, Cheng CK. Module placement with boundary constraints using *O*-tree representation. In: Proc. of the IEEE ISCAS 2002. 2002. 871~874.

[21] Liu R, Hong XL, Dong SQ, Gu J. Block placement with predefined coordinate alignment constraint using sequence pair representation. Journal of Software, 2003,14(8):1418~1424 (in English with Chinese abstract). http://www.jos.org.cn/1000-9825/14/1418.htm

[22] Ismail M, Fiez TS. Analog VLSI: Signal and Information Processing. New York: McGraw-Hill, 1994.

[23] Basaran B, Rutenbar RA, Carley LR. Latchup-Ware placement and parasitic-bounded routing of custom analog cells. In: Proc. of the IEEE/ACM Int'l Conf. on Computer-Aided Design. 1993. 415~421.

附中文参考文献:

[21] 刘锐,洪先龙,董社勤,顾钧基于序列对表示的对齐约束模块布局算法.软件学报,2003,14(8):1418~1424. http://www.jos.org.cn/1000-9825/14/1418.htm