# Symmetry Detection for Automatic Analog-Layout Recycling*

Youcef Bourai and C.-J. Richard Shi
Department of Electrical Engineering, Box 352500
University of Washington
Seattle, WA 98195-2500, USA

***Abstract:*** *Layout symmetry is used to minimize the impact of mismatch on the performance of analog circuits. In this paper, an efficient algorithm is presented to detect automatically the mask layout symmetry. It consists of identifying signal nets, isolating circuit devices and detecting their symmetry, and finally synthesizing the layout symmetry. Combined with layout compaction with symmetry constraints, this technique provides a methodology for automatic analog-layout recycling.*

## 1. INTRODUCTION

Analog and mixed analog/digital (mixed-signal) circuits represent a significant portion of today's application specific integrated circuits (ASIC) industry. Tight performance needs, fast technology innovations and strict time-to-market requirements lead to a constant evolution of the ASIC market. To be successfully in the market place, multiple-source "foundries" have become essential for many companies. This requires that a layout design be retargetable for different fabrication processes. The same issue arises in the emerging Intellectual Properties (IP)-based design as how to reuse physical layouts for digital cell libraries and analog building blocks.

*Recycling* layout resources accumulated for old fabrication-processes is important in cutting the design cycle time. It is generally very time-consuming to modify a mask layout even for minor changes in the design rules. Several automatic techniques have been proposed to recycle the digital portion of a mixed-signal-circuit layout. They are either based on explicit symbolic-layout extraction followed by compaction [4,8], or by transforming a given layout into a layout description format expressed in parameters associated with shapes, sizes, and locations, and then to re-synthesize an optimal layout based on the new set of design rules [10].

Unfortunately, these techniques are not directly applicable to recycle the analog portion of a mixed-signal-circuit layout. Different from digital layouts, to achieve high performance requirements, analog layouts are designed to satisfy certain symmetry and matching constraints.

The methodology we propose to automatic analog-layout recycling consists of layout symmetry extraction and layout compaction with symmetry constraints. The latter problem has been resolved in [2,6]. In this paper, we address how to extract the circuit symmetry from a mask layout.

*. This work was sponsored by Rockwell Semiconductor Systems, NSF/industry Center for Digital-Analog Integrated Circuits (CDADIC), and by US Defense Advanced Research Projects (DARPA) under grant number F33615-96-1-5601 from the United States Air Force, Wright Laboratory, Manufacturing Technology Directorate.

## 2. OVERVIEW OF THE PROPOSED METHOD

To minimize electrical offsets and noise, certain devices of an analog circuit are placed symmetrically with respect to an axis. Skillful analog designers may insert several symmetry axes. This is illustrated in Figure 1.

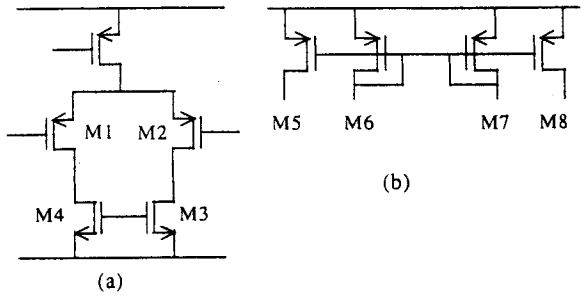



Figure 1: Symmetrical pairs are: (a). (M1,M2), (M3,M4) (b) (M5,M8), (M6,M7).

To guarantee the proper analog circuit performance, those signal nets that connect the symmetric devices are usually symmetric with respect to the symmetry axis. Therefore our method can be introduced as follows:

- Isolate the devices of a circuit by identifying signal nets or net factorization.
- Compare the isolated devices to find which of them are symmetric.
- Detect net symmetries and synthesize the layout symmetry.

The rest of this paper is organized as follows. Sections 3 and 4 describe, respectively, signal net identification and device isolation. Section 5 presents an algorithm to detect symmetric devices. Section 6 addresses the layout symmetry synthesis. Section 7 analyzes the complexity of the proposed method. Section 8 concludes the paper.

## 3. NET FACTORIZATION

In our method the layout mask is internally represented by the Corner Stitching (CS) data structure [7]. Consequently, the factorization procedure is similar to that of Magic's net extractor [9], which is based on the flooding approach. The fundamental operation is to mark all the tiles that belong to a single electrical node (net). Starting at a given tile $T$, for example the lower left tile of the connecting layer, mark recursively all its neighbors. Once the marking procedure is finished all the tiles in the node are marked with the same node number $n$. By repeating the

procedure for each unmarked tile all the nets are marked (and factorized out). The marking procedure can be sketched as follows:

1. See if $T$ has been marked. If so return.
2. Mark the tile $T$ as belonging to the node $n$.
3. Visit all the neighbors of the tile that connect to $T$. Recursively process each of the neighbors that are electrically connected to $T$.
4. Contact tiles are duplicated on all planes they connect. If $T$ is a contact, the corresponding contact tiles on other layers are found using the point search algorithm. Each such tile is then processed recursively.

The explicit representation of connectivity by corner-stitches, coupled with the simplicity of the flooding approach, makes net factorization extremely fast. Figure 2 shows an example of net factorization.

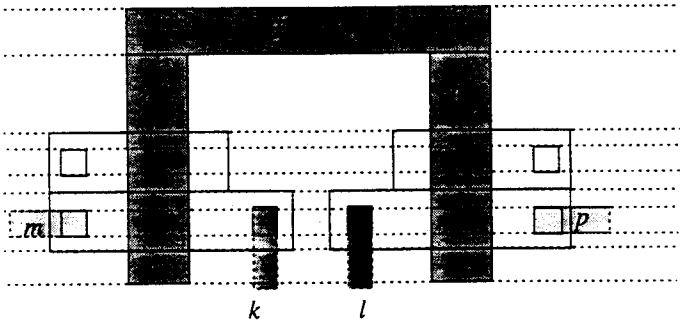



Figure 2: Net factorization, where five nets are marked by *k, l, m, n,* and *p*.

## 4. DEVICE LAYOUT ISOLATION

At the end of net factorization, we can differentiate between the tiles that form the nets from those that form the devices. Thus by scanning the layout, the tiles that form the net are ignored. By doing so, the layout becomes a set of isolated entities.

Each entity is formed by a set of intersected rectangles. We define an intersecting graph of rectangles $G = (V,E)$, where each $v \in V$ is a rectangle, and there is an edge $e \in E$ between $v_1 \in V$ and $v_2 \in V$ if the two corresponding rectangles intersect. To find these entities, we use the algorithm in [3], which finds the *Connected Components* (CC) of an intersecting graph of rectangles in the plane, as illustrated by Figure 3. Thus, each connected component represents a device in the layout. The intersected rectangles are found by the algorithm described in [1], which is based on the Priority Search Tree (PST) technique [5]. We have chosen the PST technique to report the intersected rectangles because of its powerfulness and because it is easy to implement. At the end of the scanning process, we obtain a set of connected components each being a *device* placed in a plane.

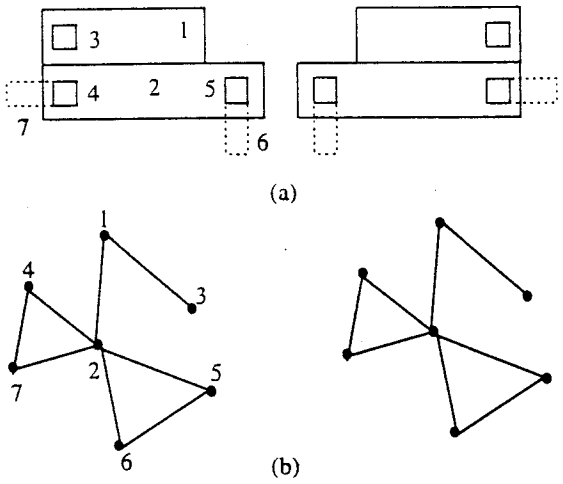


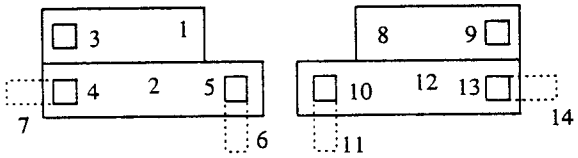

Figure 3: (a) Isolated devices,(b) Connected Components of the intersecting graph of rectangles.

## 5. SYMMETRIC DEVICE DETECTION

Consider those devices that are horizontally aligned. We first sort them according to their lowest bottom edge and stored them in a "device" queue to be swept.

We sweep the device queue. The devices that share the present scan-line are pair-wise compared. If two devices are identical and connected by a net, they are detected to be symmetric. Therefore, we pop them up and generate a temporary symmetry axis.

The detection algorithm is described as follows. The first device is collected in a heap $H_1$, where the rectangles are sorted in increasing manner according to their left edge. The second one is then mirrored i.e. it is collected in another heap $H_2$ where its rectangles are sorted in a decreasing manner according to the right edge as depicted by Figure 4. By scanning the two heaps, the rectangles that share the present vertical scan-line are gathered in two different lists. Each list is sorted three times: according to the bottom edge of the rectangles, according to their height, and finally according to their width.




$H_1 = \{7,1,2,3,4,5,6\}$ $H_2 = \{14,8,12,9,13,10,11\}$

Figure 4: Devices and their heaps.

The triple sorting is necessary to avoid the conflict depicted by Figure 6. Thus the two devices are symmetric if at each vertical scan- line, the compared two lists are exactly identical as shown by the algorithm in Figure 5. Once the scanning is over, and using the net information obtained from the net factorization phase, we eliminate the redundant axes.

## 6. LAYOUT SYMMETRY SYNTHESIS

Once the symmetric devices are detected, all the nets that have the same node as the terminal of these symmetric devices are also symmetric. Thus, since the tiles of each net have been already marked with the node number of the net, these tiles can be gathered in the same set. For example in Figure 2, the tiles of the net $m$ and net $p$ are symmetric with respect to the symmetry axis generated by the devices to which net $m$ and net $p$ are connected. By visiting the nets, a symmetric set $S$ of tiles can be constructed for each symmetric axis. With this, the symmetry constrains for compaction can be generated in a straightforward manner.

## 7. COMPLEXITY ANALYSIS

Our algorithm uses the corner-stitching data structure. Because this data structure induces a plethora of small tiles that are rectangles, each procedure of the algorithm must be fast enough to yield a good performance.

During the net extraction phase, two major procedures are used: the enumeration procedure to find the starting tile, and the neighboring search procedure. It has been proved in [7] that the expected running time of the directed enumeration algorithm is linear in the number of tiles intersecting the search area. The expected running time of neighboring search is linear in the number of neighbors.

In the device extraction phase, using the balanced Priority Search Tree to find the intersecting rectangles has the expected running-time complexity of $O(nlogn+k)$ and the space complexity $O(n+k)$, where $n$ is the number of rectangles, $k$ is the number of pair-wise intersections, and $k$ is much smaller than $n$ as shown in [1]. The UNION FIND algorithm [11] is used to cluster the intersecting rectangles in connected components, which has the expected running-time complexity $O(nlogn+k)$. Thus the overall running-time of our algorithm is $O(nlogn+k)$, where $k$ is smaller than $n$.

The algorithm is coded in the C++ programming language. A test example is illustrated in Figure 7. By scanning the layout and applying the algorithm of Figure 5, three local symmetry axes have been initially found (see Figure 7 (a)). After eliminating the redundant ones we have kept a unique axis which is common to all the symmetric objects as shown by Figure 7 (b).

## 8. CONCLUSIONS

A fast yet simple algorithm to detect analog-circuit symmetry at the mask layout level has been presented. By combined with layout compaction with symmetry constraints, automatic analog-layout recycling can be achieved.

## REFERENCES


[1] T. Asano, M. Sato and T. Ohtsuki, " Computational geometry algorithms", in *Layout Design and Verification* (T. Ohtsuki ed.), Elsvier Science Pub., pp. 295-347, 1986.

[2] E. Felt, E. Malavasi, E. Charbon, R. Totaro and A. Sangiovanni-Vincentelli, "Performance-driven compaction problem with symmetry constraints," *Proceedings of IEEE Custom Integrated Circuits Conference*, pp. 1731-1735, May 1992.

[3] H. Imai and T. Asano, "Finding the connected components and a maximum clique of an intersecting graph of rectangles in the plane," *Journal of Algorithms*, vol. 4, pp. 310-323, 1983.

[4] B. Liu and A. R. Newton, "Kahlua: A hierarchical circuit disassembler," *Proceedings of IEEE/ACM Design Automation Conference*, pp. 311-316, 1987.

[5] M. Mc Creight, "Priority search trees," *SIAM Journal on Computing*, vol. 14, no.2, pp. 257-276, May 1985.

[6] R. Okuda, T. Sato, H. Onedera and K. Tamaru, "An efficient algorithm for layout compaction problem with symmetry constraints," *Proc. IEEE/ACM International Conference on Computer Aided Design*, pp. 148-151, Nov. 1989.

[7] J. K. Ousterhout, "Corner stitching: A data-structuring technique for VLSI layout tools," *IEEE Transactions on Computer Aided-Design of integrated Circuits and Systems*, vol. CAD-3, no.1, pp. 87-100, January 1984.

[8] T. Sato, N. Ohba, H. Watanabe and S. Saito, "Stick diagram extraction program: SKELETON," *Proceedings of IEEE/ACM International Conference on Computer Aided Design*, pp. 318-321,1988.

[9] W. S. Scott and J. K. Ousterhout, "MAGIC's circuit extractor," *Proceedings of IEEE/ACM Design Automation Conference*, pp. 286-292, 1985.

[10] Y. Shigshiro, T. Nagata, I. Shirakawa, I. Arungsrisangohai and H. Takahashi, "Automatic layout recycling based on layout description and linear programming," *IEEE Transactions on Computer Aided-Design of Integrated Circuits and Systems*, vol. 15, no. 8, pp. 959-967, August 1996.

[11] R. E. Tarjan "Efficient of a good but not linear set union algorithm," *Journal of ACM*, 22, pp. 215-225, 1975.

***Input***: *Two CCs (Connected Components) of intersecting graph of rectangles $CC_1$ and $CC_2$ with the same number of rectangles;*
***Output***: *is true if the two CCs are symmetrical;*

```
{
/* Rectangle is defined as a structure R(l,b,r,t) where l,b,r,t are
the left, bottom right and top edges respectively */
Rectangle R_i , R_j, R_s, R'_s ;
Boolean result;
Heap H_1, H_2;
List L^0_1, L^1_1, L^2_1, L^3_1, L^0_2, L^1_2, L^2_2, L^3_2 ; /* list of rectangles */
/*H_1 sorts increasingly the rectangles of CC_1 according to their
left edge*/
For each R_i ∈ CC_1
    insert(H_1, R_i(l ));

/*H_2 sorts decreasingly the rectangles of CC_2 according to their
right edge*/
For each R_j ∈ CC_2
   insert(H_2, R_j( r));

/* get the left most element of each heap */
R_s  = left_most_of(H_1);
R'_s = leftt_most_of(H_2);

While (R_s ≠ nil)
   {
    initialize(L^0_1); initialize(L^0_2);
    for each R_i ∈ H_1 and R_i(l) = R_s(l)
            insert(L^0_1, R_i);
    for each R_j ∈ H_2 and R_j(r) = R'_s(r)
            insert(L^0_2, R_j)
    /* sort L^0_1 and L^0_2 according to the bottom of the rectangles*/
    L^1_1=sort_b(L^0_1);L^1_2=sort_b(L^0_2);
    /*sort L^1_1 and L^1_2 according to the height of the rectangles*/
    L^2_1=sort_h(L^1_1);L^2_2= sort_h(L^1_2); /
    / * sort L^2_1 and L^2_2 according to the width of the rectangles*/
    L^3_1=sort_w(L^2_1);L^3_1= sort_w(L^2_2);
    result = is_same(L^3_1,L^3_2);
    if (result = false)
      return false;
   R_s  = get_next_element(H_1, R_s);
   R'_s = get_next_element(H_2, R'_s);
   } /* end of while */
 return true;
```

Figure 5: The device-symmetry detection algorithm.

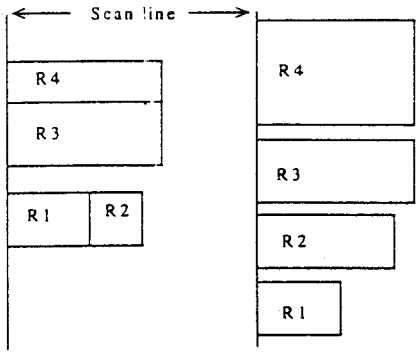
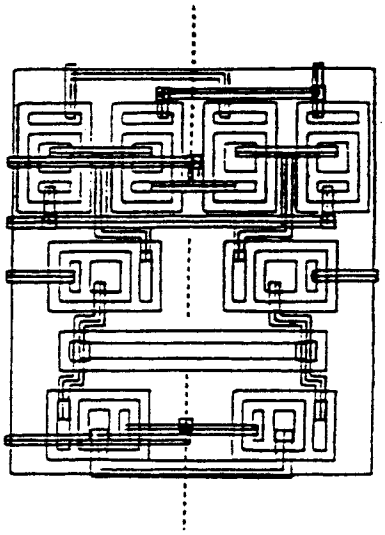
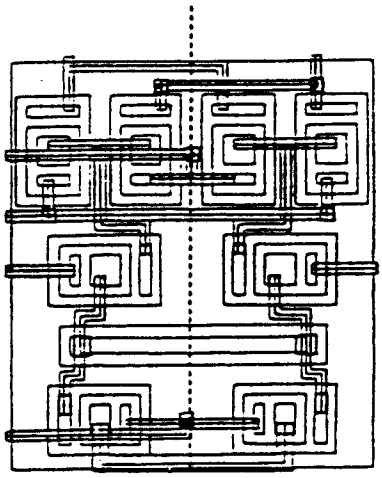



Figure 6: Differentiating between rectangles by triple sorting.



(a) Three local symmetry axes



(b) The three local axes are merged into one axis

Figure 7: A layout example.