Deterministic Analog Circuit Placement using Hierarchically Bounded Enumeration and Enhanced Shape Functions

Martin Strasser, Michael Eick, Helmut Gräb, Ulf Schlichtmann, and Frank M. Johannes
Institute for Electronic Design Automation, Technische Universität München, Munich, Germany

***Abstract*— The analog placement algorithm *Plantage*, presented in this paper, generates placements for analog circuits with comprehensive placement constraints. Plantage is based on a hierarchically bounded enumeration of basic building blocks, using B*-trees. The practically relevant solution space is thereby enumerated quasi-complete. The sets of possible placements of the basic building blocks are represented and combined in a new efficient way, using enhanced shape functions. The result of Plantage is the Pareto front of placements with respect to different aspect ratios. The whole approach is deterministic, in contrast to existing analog placement algorithms.**

I. Introduction

In modern integrated circuits, analog parts gain more and more importance. The function and performance of these analog parts are heavily influenced by the placement of the modules of the circuit. For digital circuits, many layout approaches are successfully used in the semiconductor industry. For analog circuits, different approaches are currently in research to avoid time-consuming and error-prone manual design. The automation of analog layouts presents a different problem, because such layouts must fulfill many constraints. These constraints are necessary to meet the performance specifications. For example, unbalanced parasitics due to an asymmetrical layout may be detrimental to the power supply rejection ratio or the offset voltage of an analog circuit.

A. Analog Circuit Placement Requirements

A common layout approach is to separate placement and routing. This paper addresses the automation of the placement process of analog circuits. Typical placement constraints of analog layouts are considered. These placement constraints, listed below, are usually formulated to reduce the impact of parasitics, process variations, and different operating conditions on circuit performance.

- *Device-proximity constraints* are used for different reasons. Due to local variations during the fabrication process, devices exhibit unwanted deviations from each other ("mismatch"), which can result in performance degradation. Variations in the operating conditions, such as temperature or supply voltage, have the same effect. The placement of matched devices in close proximity reduces the impact of these variations [1].
- *Symmetry constraints* are used for geometric and electrical issues. For example, symmetric placement reduces the sensitivity to on-die thermal gradients. It is also used to balance parasitic resistors and capacitors on both halves of a differential circuit. Every placeable element of a circuit, such as a transistor or a capacitor, is called module in this paper. Symmetry constraints are formulated as linear equations. For some module i, the point (x_i, y_i) denotes the lower left cooordinates of i in the layout. The tuple (w_i, h_i) represents the width and height of i. The module i' denotes a second module that is to be placed symmetrically to i. For modules, which are self symmetrical, i is equal to i'. An example of symmetry constraints is shown in Fig. 1(a), where all modules are arranged with respect to a vertical symmetry axis. For a vertical symmetry, linear equations are formulated as follows:

$$\forall_i \left(\frac{1}{2}(x_i + \frac{w_i}{2} + x_{i'} + \frac{w_{i'}}{2}) = x_{\text{sym}} \right) \tag{1}$$

$$\forall_i \left(y_i + \frac{h_i}{2} = y_{i'} + \frac{h_{i'}}{2} \right) \tag{2}$$

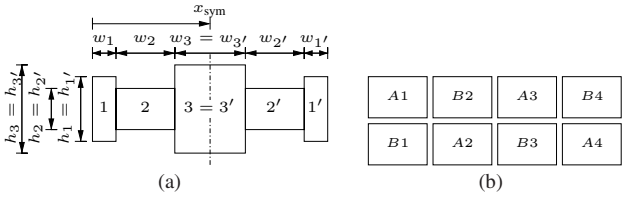

Fig. 1. Placement with symmetry (a) and common centroid (b) constraint

The equations for a horizontal symmetry axis are defined analogously.

- A *common centroid constraint* is formulated to arrange the centers of gravity for groups of modules. This improves the beneficial effects of the symmetry constraint [2]. For example, a differential pair can be formed by eight transistors and arranged as shown in Fig. 1(b). The transistors A1, A2, A3, and A4 are connected in parallel. The same applies for the transistors B1, B2, B3, and B4. All transistors have the same size and the two groups of transistors share the same center of gravity. For two groups of modules, $\mathcal{A}$ and $\mathcal{B}$, a common centroid constraint is defined as follows:

$$\sum_{i \in \mathcal{A}} \left(x_i + \frac{w_i}{2} \right) = \sum_{j \in \mathcal{B}} \left(x_j + \frac{w_j}{2} \right) \tag{3}$$

$$\sum_{i \in \mathcal{A}} \left(y_i + \frac{h_i}{2} \right) = \sum_{j \in \mathcal{B}} \left(y_j + \frac{h_j}{2} \right) \tag{4}$$

- *Minimum distance constraints* are formulated if modules cannot abut on each other directly, but need a minimum distance. For example, some transistors sharing the same well may abut directly, but a minimum distance is needed from transistors outside of the common well. In addition, routing area has to be considered during placement. Formally, this constraint can be defined as a linear inequality.
- *Variant constraints* restrict the combination of possible realizations (variants) of circuit modules. For example, realization variants are used for transistors, which may have a different number of fingers, or for capacitors, which may have different possible aspect ratios. Additionally, some modules can be rotated or mirrored. When a module has a set of possible variants, the placement algorithm has a higher degree of freedom. Better placements can be achieved. In practice, however, the combination of the different variants is not completely free. For example, the transistors of a differential pair must be realized using the same number of fingers to ensure matching.

A successful routing of the layout will also depend on the placement constraints being satisfied. For example, in a differential circuit, the wires of both signal nets must have symmetric parasitic resistances and capacitances [2][3]. This can be ensured if the connected modules are laid out symmetrically as well. An economic criterion is the compactness of the layout represented by the area usage of the placement. In conclusion, analog layouts have to be as compact as possible, while meeting all placement constraints. The minimization of the netlength is regarded secondary to compactness.

978-1-4244-2820-5/08/$25.00 ©2008 IEEE

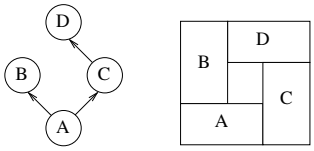

Fig. 2. B*-tree and its corresponding placement

B. Context of this work

For analog placement, existing approaches can be divided into two classes: the positions of the modules are stored using either *absolute* coordinates or *topological* representations. The approaches [4][5][6][7] use absolute coordinates. Using this representation, arbitrary constraints can be formulated using the coordinates directly and every possible placement can be described. A disadvantage is the high dimensionality of the solution space, which is $\mathbb{R}^{2n}$ for n modules. The high dimensionality results in high computation times for placement algorithms based on this representation.

The topological representations of placements, proposed during the last years, show a much smaller solution space, while being able to encode all *admissible* [8] placements. These encodings do not allow module overlaps in contrast to absolute representations. Topological representations include Sequence Pair [9], Bounded Sliceline Grid [10], O-Tree [8][11], Corner Block List [12][13], TCG-S [14], and B*-tree [15][16] representations. As pointed out in [15][16], the B*-tree has the lowest solution space redundancy. Therefore, the presented placer Plantage is based on B*-trees.

Fig. 2 shows an example of a B*-tree and the corresponding placement. Each node in a B*-tree represents a module. The algorithm proposed in [15] forms the placement of a given B*-tree according to the following rules: the module of a left child node is placed above its parent module, the module of a right child node is placed to the right of its parent module. If the y-projections of two modules overlap, the module that comes first in the preorder traversal of the tree is placed to the left of the other. The preorder traversal of the tree in the example above is ABCD. A and B are placed to the left of C, B is placed to the left of D. The resulting placement is compacted to the lower left corner.

Constraints can be used efficiently to restrict the solution space [17][11][16][18][19][20]. The authors of these papers propose Simulated Annealing algorithms that consider symmetry constraints with a restricted solution space using O-trees [11], B*-trees [16], Sequence Pairs [18][19], and Sequence Pairs with Johnson's Priority Queue [20]. In [21], a different approach is presented, based on two modifications of the B*-tree: the first is to handle symmetry constraints with so-called Symmetry Islands and the second to combine these Symmetry Islands with the rest of the modules. A placement algorithm with symmetry and other placement constraints is presented in [22]. The concept of dummy nodes in constraint graphs is introduced to fulfill symmetry constraints. The authors of [13] propose a Sequence Pair based approach for analog placement with the capability of handling common centroid constraints. Modules with common centroid constraints are clustered and placed with either a modified Corner Block List or a grid based algorithm. All previous works mentioned here use Simulated Annealing to optimize placement. In contrast, Plantage is deterministic.

C. Contributions of this Paper

In this section, an overview of the approach will be given.

All admissible placements of a given circuit can be generated based on B*-trees. A complete enumeration of all B*-trees would yield the optimal solution. However, this would result in a impractically large number of B*-trees to be evaluated. Hierarchy is used to solve this complexity problem. For small parts of the circuit, all placements are enumerated. These partial solutions are then combined to generate a placement for the whole circuit. Joining the partial solutions together using their bounding boxes would deteriorate the area usage. To avoid this, a new concept, the enhanced shape function, is introduced. The result of the proposed algorithm is a Pareto front of placements with different aspect ratios.

Analog circuits show a hierarchical structure [1][23]. An analog circuit can be decomposed into several building blocks [24][25], e.g., current mirrors or differential pairs. Thus, an analog circuit can be described as a hierarchy tree, where the leaf nodes are the modules and the root node represents the whole circuit.

The presented algorithm, Plantage, is a new and deterministic approach for analog circuit placement. Plantage uses a hierarchy tree and incorporates a bottom-up approach. The algorithm starts with **basic module sets**, consisting of only a few leaf nodes of the hierarchy tree, which share the same parent node. The algorithm enumerates the complete solution space for the basic module sets. This enumeration is done based on B*-trees. The enumeration is accelerated using feasibility checks, as proposed in [16][20]. A new algorithm is proposed in Section II to transform a given B*-tree into a placement considering all constraints for analog circuit placement. The Pareto optimal placements for the basic module sets are stored using *enhanced shape functions*, also introduced in this paper.

After calculating all possible placements for the basic module sets, the algorithm steps up to next level in the hierarchy: it then combines the results of the previously calculated placements for the basic module sets. The Pareto optimal results are stored in an enhanced shape function for the current hierarchy level. Suboptimal combinations are discarded in every hierarchy level to limit the computational effort in subsequent steps. This methodology is executed until the highest hierarchy level is reached, covering the whole circuit. The enhanced shape function of the whole circuit represents the Pareto front of optimal layouts, in contrast to other state-of-the-art approaches which produce a single layout. This enables the designer to select among different valid designs having different aspect ratios.

Plantage has the following features:

- Analog circuit placement constraints (see Section I-A) are fulfilled
- Computation of a set of possible placements with different aspect ratios instead of a single solution
- Based upon a non-slicing topological placement structure, the B*-tree
- Full enumeration of basic module sets, guided by the hierarchy of the circuit
- Deterministic algorithm, suitable for parallelization
- Variant selection is integrated seamlessly in the enumeration

This paper is organized as follows: Section II describes the new algorithm to generate a placement for a given B*-tree. In Section III, enhanced shape functions are introduced. Section IV describes the hierarchical placement based on enhanced shape functions. Section V shows experimental results computed by the presented approach. A conclusion is given in Section VI.

II. B*-tree Placement Considering Linear Constraints

Many different B*-trees have to be evaluated for parts of the circuit as well as for the whole circuit. The evaluation of a B*-tree is based on its corresponding placement. The proposed methodology is used to generate placements with respect to arbitrary linear constraints from a feasible B*-tree. Linear constraints are needed for symmetry and common centroid constraints, as well as for minimum distance constraints (see Section I-A).

To model the horizontal and vertical relationships between modules, two constraint graphs are used. A constraint graph (CG) is a directed graph consisting of a set of nodes $\mathcal{V}$ and a set of directed edges $\mathcal{A}$. Directed edges are ordered pairs of nodes. In this approach,

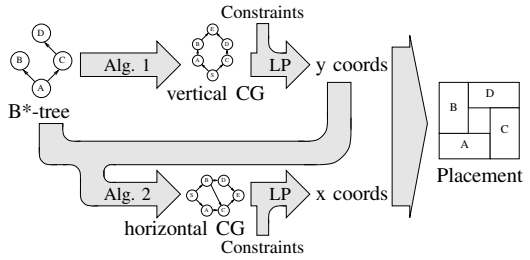

Fig. 3. Placement generation from a B*-tree considering constraints

each module has a corresponding node in the horizontal and the vertical constraint graph.

Fig. 3 shows an overview of the methodology. Algorithm 1 generates a vertical CG for the given B*-tree. For every edge (n_i, n_j) in the vertical CG, an inequality $y_i + h_i < y_j$ is formulated, which requires module j to be placed above i. Fig. 4 shows the vertical CG for the example in Fig. 2. For this vertical CG, inequalities are formulated as follows: $y_A \geq y_S, y_B \geq y_A + h_A, y_C \geq y_S, y_D \geq y_C + h_C, y_E \geq y_B + h_B, y_E \geq y_D + h_D$. Together with the linear constraints (symmetry, common centroid, minimum distance), a linear program (LP) is set up minimizing the height of the placement. The results of the LP are the y coordinates of the modules. Algorithm 2 generates a horizontal CG based on the y coordinates *and* the given B*-tree. A linear program is set up analogously to the vertical case, with the results being the x coordinates.

Algorithm 1: buildVerticalCG(CGNode thisNode, predecessor)

```
begin
    if B*-tree node of thisNode has a left child then
        leftNode ← new CG node for left child;
        add edge from thisNode to leftNode;
        buildVerticalCG(leftNode, thisNode);
    else
        add edge from thisNode to the end node;
    if B*-tree node of thisNode has a right child then
        rightNode ← new CG node for right child;
        add edge from predecessor to rightNode;
        buildVerticalCG(rightNode, predecessor);
end
```

The vertical constraint graph is built as described in Algorithm 1: if module i is a left child of module j in the B*-tree, then n_i is the direct successor of n_j in the CG. If module i is a right child of module j in the B*-tree, then n_i and n_j share the same predecessor. To start Algorithm 1, a start node is created. Also, a node corresponding to the root node of the B*-tree is added with the start node being its predecessor. Both nodes are passed to the algorithm.

The CG for the vertical axis is used to formulate a linear program:

$$\mathbf{y}_{opt} = \arg\min_{\mathbf{y}} y_E$$
$$\text{s.t.} \quad \underbrace{\mathbf{M}_v \cdot \mathbf{y} \geq \mathbf{d}_v}_{\text{Minimum distance constraints}} \quad , \quad \underbrace{\mathbf{C}_v \cdot \mathbf{y} = \mathbf{k}_v}_{\text{Symmetry \& common centroid constraints}} \tag{5}$$

The vector $\mathbf{y}$ is the vector of y coordinates for all modules and y_E is the y coordinate of the end node. The matrix $\mathbf{M}_v$, together with $\mathbf{d}_v$, defines the minimum vertical distances between the modules. The matrix $\mathbf{C}_v$, together with the vector $\mathbf{k}_v$, define the symmetry and common centroid constraints for the vertical axis. Minimizing y_E is equivalent to minimizing the total height of the placement. The vector $\mathbf{y}_{opt}$ contains the optimal y coordinates for the given B*-tree.

Based on the results of the vertical LP, the y coordinates of the modules are known, and a *list of y regions* is set up. The modules are then "registered" consecutively in the list of y regions and a CG for the horizontal axis is formulated. This is described in Algorithm 2.

Algorithm 2: buildHorizontalCG()

```
begin
    startNode ← new CG node as start;
    endNode ← new CG node as end;
    create list of y regions of all modules;
    initialize all y regions with startNode;
    forall modules in preorder do
        modNode ← new CG node for module;
        forall regions r in region list from module.y to module.y
        + module.height do
            shadowNode ← node of the module which is
            registered in r;
            add edge to modNode from shadowNode;
            register module in r; // Shadow that segment
        remove multiple edges;
    add edges from all nodes in region list to endNode;
end
```

Fig. 5 demonstrates this algorithm for a simple example. On the left side in the Figs. 5(a) to 5(f), the list of y regions is shown. To set up this list of y regions, y_N as well as $y_N + h_N$ is stored for each module in a list of borders. This list of borders is then sorted in ascending order and multiple entries at the same y coordinate are removed. The regions between two borders form the y regions list. A module can be registered in a y region if its y projection overlaps the region. The modules are registered in the order defined by the preorder traversal of the B*-tree. In Fig. 5(a), no module is registered yet, and all entries of the region list are initialized with S, denoting the startNode of the CG. Module A is registered as shown in Fig. 5(b). Module A is now an entry in the region list, and an edge from the start node to the module node is added. The same is done for module B in Fig. 5(c). In Fig. 5(d), module C is registered. The y projection of this module shadows module A and parts of module B. As a result, two edges are added from A to C and from B to C. The two shadowed entries of the region list are then replaced by C. Module D is then added, shadowing parts of module B. An edge is added from B to D and the upper entry of the region list is replaced by D. After all modules are registered, CG nodes which have corresponding entries in the region

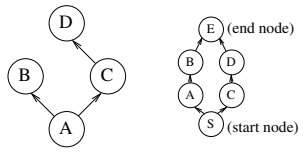

Fig. 4. Example: B*-tree and its corresponding vertical CG.

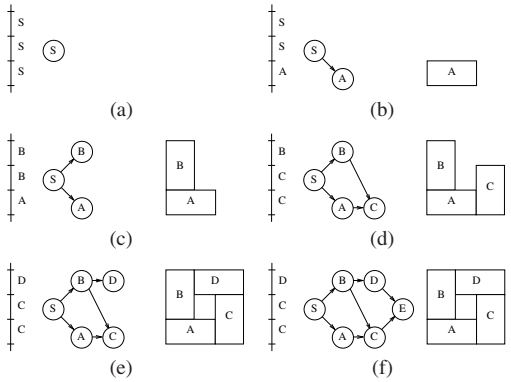

Fig. 5. Example demonstrating Algorithm 2

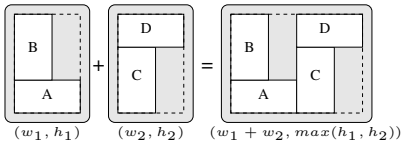

Fig. 6. Standard shape addition

list are connected to the end node.

A horizontal LP is then set up according to the horizontal CG. This LP is defined similarly to the LP for the vertical axis, as defined in (5). After solving the horizontal LP subject to the linear constraints of the horizontal axis, the x and y coordinates of all modules with nodes in the B*-tree are known. These coordinates represent the corresponding placement.

III. Enhanced Shape Functions

In this section, enhanced shape functions are introduced to handle the combination of different partial placements efficiently. A short overview of standard shape functions is given in the next section. The enhanced shape function and the enhanced combination of shapes is described in the subsequent sections.

A. Overview of Shape Functions

Shape functions [26] are conventionally used to calculate compact placements for a set of rectangular modules. A shape function is defined as an ordered set of shapes, representing placements with different aspect ratios. Each shape describes one possible placement of a module set by its bounding rectangle size, formulated as a tuple (w, h), where w and h denote the width and height, respectively.

To determine placements, the shape function of the module set is calculated recursively: The set of modules is partitioned into two subsets. A shape function is calculated for each subset. If the subset only contains one module, the shape function only has one shape, representing the size of this module. To combine the shape functions of the two subsets, all possible combinations of the shapes of both shape functions are evaluated. This can be done quickly. Since shapes represent the bounding rectangles of their corresponding placement, combining two shapes means finding a common bounding rectangle for the two corresponding placements. It is possible to combine two placements horizontally and vertically. This is called horizontal and vertical addition. The result of a horizontal addition of two shapes (w_1, h_1) and (w_2, h_2) is $(w_1 + w_2, max(h_1, h_2))$. An example of horizontal addition is shown in Fig. 6. The result of a vertical addition is $(max(w_1, w_2), h_1 + h_2)$. Fig. 7 shows an example of a shape function after all combinations have been evaluated. In this plot, there are suboptimal shapes which have a bigger height than other shapes having the same or even lower width, and are removed before further calculations are performed. Removing suboptimal shapes dramatically reduces the time needed in subsequent steps, while the quality of the solution remains unchanged. This is a key feature of shape functions. A continuous shape function can be drawn based on the remaining shapes, as shown in Fig. 7, which can be considered the Pareto front of possible placements.

After the recursive algorithm has terminated, the final result is a shape function for the circuit, representing different placements with different aspect ratios. This is a key feature of shape functions.

B. Definition of Enhanced Shape Functions

A corresponding placement of a shape can be described as a slicing tree, because it is built by horizontal and vertical additions of other placements [27]. A slicing tree cannot handle non-slicing placements. This limits the solution space and degrades the solution quality.

Enhanced shape functions, proposed here, have the key features of shape functions while at the same time being able to handle non-slicing placements. An enhanced shape is defined as (w, h, α).

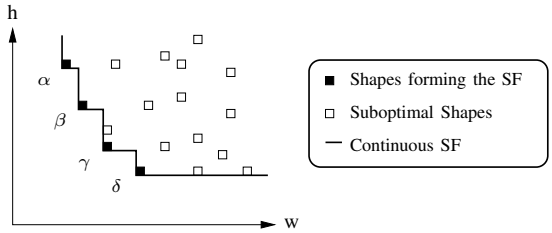

Fig. 7. All shapes of a resulting shape function (SF), denoting the corresponding B*-trees by Greek letters

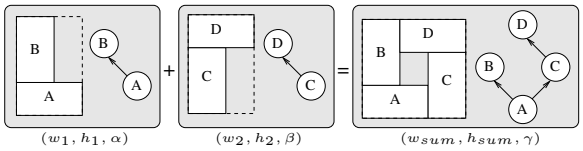

Fig. 8. Horizontal enhanced shape addition

In contrast to standard shapes, the corresponding B*-tree α of a placement is stored in addition to the placement's bounding box (w, h). The B*-tree allows efficient combination of the enhanced shape functions and their underlying modules, as described in the subsequent section. The widths and heights are used to calculate the Pareto front, and to select the suboptimal enhanced shapes to be removed. An enhanced shape is suboptimal in two cases:

- It has a bigger height than other enhanced shapes having the same or even lower width.
- It has a higher netlength than other enhanced shapes, having the same width and height.

In this paper, B*-trees are denoted by Greek lower case letters.

C. Enhanced Shape Function Combination

To combine two enhanced shape functions, their enhanced shapes are combined in pairs. In contrast to standard shapes, the combination of two enhanced shapes (w_i, h_i, α) and (w_j, h_j, β) is done using the B*-trees. The widths w_i, w_j and the heights h_i, h_j can be used to estimate the resulting enhanced shape. For a horizontal addition, an upper bound for the size of the resulting placement can be defined as $(w_i + w_j, max(h_i, h_j))$. Using the B*-trees α and β, the size of the resulting placement can be smaller than $(w_i + w_j, max(h_i, h_j))$.

Figs. 6 and 8 show the differences between the horizontal addition of conventional and enhanced shapes for a simple example. Comparing these figures, it is obvious that $w_1 + w_2$ is greater than w_{sum}. Generally speaking, better placements can be reached if the enhanced shape function combination is used.

Two methods are proposed to add the B*-trees of enhanced shapes horizontally and vertically. They are described in the following paragraphs. It is shown, that the outcome of adding two feasible B*-trees is also a feasible B*-tree for both methods.

Horizontal Addition: Given two B*-trees α and β, a horizontal addition is performed by attaching the root node of β to the *lowest, rightmost* node of α. The lowest, rightmost node of a B*-tree is defined as the node with no right child, while the node itself as well as all its predecessors are either right children or the root node.

The resulting placement is compact to the lower left corner, due to the characteristics of the placement algorithm for B*-trees. Horizontal

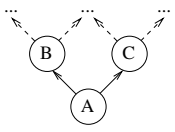

Fig. 9. An arbitrary B*-tree to define the in- and preorder traversal

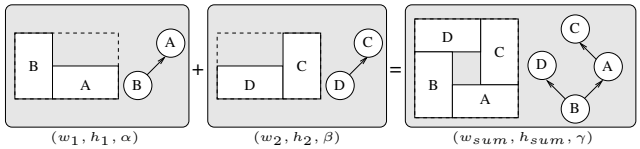

Fig. 10. Vertical enhanced shape addition

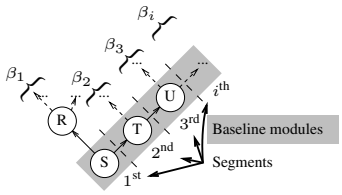

Fig. 11. Baseline modules and segments of a B*-tree β

addition has a notable property. Any constraint, which was satisfied by α and β before the addition is also satisfied by the resulting B*-tree[1]. This can be derived from the in- and preorder traversals of the B*-trees. The in- and preorder traversals of an arbitrary B*-tree, as shown in Fig. 9, are defined as ordered lists recursively by $\text{in(A)} := \text{in(B)}, \text{A}, \text{in(C)}$ and $\text{pre(A)} := \text{A}, \text{pre(B)}, \text{pre(C)}$, respectively. Two order relations are defined on the traversals: The order relation $A \overset{\text{IN}}{\prec} B$ means "A is a predecessor of B in the inorder traversal", while $A \overset{\text{PRE}}{\prec} B$ means "A is a predecessor of B in the preorder traversal."

The topologies of the B*-trees α and β are not changed by the horizontal addition. Only one edge is added to connect the two B*-trees. The feasibility of the constraints can be checked using the relative positions of the modules in the in- and preorder traversals [16]. Without loss of generality, the node C can be considered the root node of β, and A the lowest, rightmost node of α. Due to this fact, the relative positions of the modules in α and β in the in- and preorder traversals do not change. Thus, any constraint, which was satisfied before, is also satisfied after the addition of the two B*-trees. The example given by Fig. 8 illustrates a horizontal addition.

Vertical Addition: A vertical addition is intended to arrange two partial placements vertically, generating compact results. Fig. 10 shows a compact result of a vertical addition and its corresponding B*-tree γ. Unlike the horizontal addition, the resulting B*-tree γ cannot be generated by adding an edge from α to β. For this reason an algorithm is proposed which iteratively forms a new B*-tree to be the result of the addition of α and β. The topology is changed in order to achieve better placements.

In a B*-tree, all modules, which can be reached from the root node traversing right edges only are placed close to the baseline. These modules are denoted as *baseline modules* in this paper. The proposed approach segments the B*-tree β, with the root node of each segment being a baseline module. Fig. 11 shows the baseline modules and the segments of a B*-tree. Adequate nodes of α are then determined to serve as the new parents of the segment root nodes, using a contour based algorithm. This is done one segment after the other, from "left to right," in the order given in Fig. 11.

Fig. 12 illustrates how the B*-tree γ of Fig. 10 was built by the algorithm. In Fig. 12(a), no segment of β is added to α. A contour is drawn as a thick line above the placement for α. Then, the first segment of β, the node D, is added in Fig. 12(b). The x projection of D shadows B and parts of A. Node D is added as a left child of module B, which limits the y coordinate of D when shifting it downwards. According to the same rules, C is appended as a left child of A, shown in Fig. 12(c).

[1]This is true as long as no additional constraints apply for the superset of the modules of the two B*-trees.

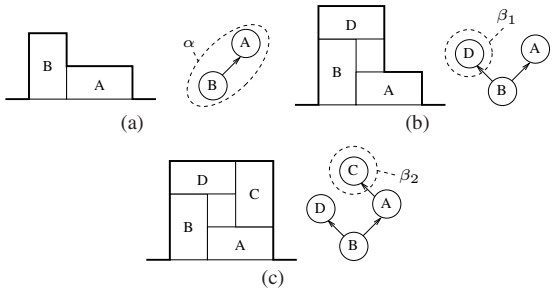

Fig. 12. Iteratively changing the B*-tree topology for a vertical addition

Also for the vertical addition, the constraints remain satisfied: The i^{th} and $(i+1)^{\text{th}}$ segments of β are denoted as β_i and β_{i+1}, and $\text{root}(\beta_i)$ denotes the root node of β_i. The segments are added in ascending order, as defined in Fig. 11. Before the addition, the in- and preorder positions of the segments fulfill the following conditions: $\text{in}(\text{root}(\beta_i)) \overset{\text{IN}}{\prec} \text{in}(\text{root}(\beta_{i+1}))$ and $\text{pre}(\text{root}(\beta_i)) \overset{\text{PRE}}{\prec} \text{pre}(\text{root}(\beta_{i+1}))$. The root node of β_i is then added to a node of α, denoted as $\text{parent}(\beta_i)$, as a left child. Since the segments are added "from left to right", the parents fulfill the condition $\text{parent}(\beta_i) \overset{\text{IN}}{\prec} \text{parent}(\beta_{i+1})$ and $\text{parent}(\beta_i) \overset{\text{PRE}}{\prec} \text{parent}(\beta_{i+1})$. As a consequence, the relative positions of the modules in the in- and preorder traversals of α, β_i and β_{i+1} do not change. Therefore, the feasibility of the constraints remains unchanged[2].

The key advantages of standard shape functions are retained by this approach. After the addition of enhanced shape functions, all suboptimal enhanced shapes are stripped. This reduces the computational effort in subsequent steps. Furthermore, a set of possible placements is stored, instead of a single solution.

IV. Hierarchically Guided Enumeration

The enumeration of the complete solution space yields the optimal result. However, this cannot be performed for most circuits because of long run times. This becomes clear when considering the number of different B*-Trees for n modules [16]. There are 336 different B*-Trees for 4 modules, while for 8 modules, there are $57,657,600$ different B*-Trees. Thus, a complete enumeration is impossible in practical cases. As a consequence, the presented enumeration approach is guided by hierarchy to limit the number of elements which are considered in an enumeration run. The hierarchy can be illustrated as a hierarchy tree, where the root represents the whole circuit. The leaf nodes represent the modules, their parents represent analog structures, such as differential pairs (DP) or current mirrors (CM). Fig. 13 shows a typical schematic of a Miller operational amplifier, together with its hierarchy tree. In this approach, the hierarchy tree is generated automatically from the netlist. This is done by an algorithm similar to the one proposed in [24][25] for the automatic generation of sizing rules for analog circuits.

The hierarchy tree is used to perform a bottom-up enumeration. First, all possible placements for the basic module sets are evaluated. These basic module sets are formed by the modules of leaf nodes having the same parent node in the hierarchy tree. In the given example of Fig. 13, these sets are {P1, P2}, {N3, N4}, {P5, P6, P7}, and {C, N8}.

The enumeration of all possible placements of a basic module set is done by evaluating all possible B*-Trees for the modules of this set. This procedure is called *basic enumeration*. For all feasible B*-Trees, basic enumeration evaluates all possible variants of the modules, e.g., different numbers of fingers for a transistor, or different aspect ratios

[2]Similar to the horizontal addition, this is true as long as no additional constraints apply for the superset of the modules of the two B*-trees.

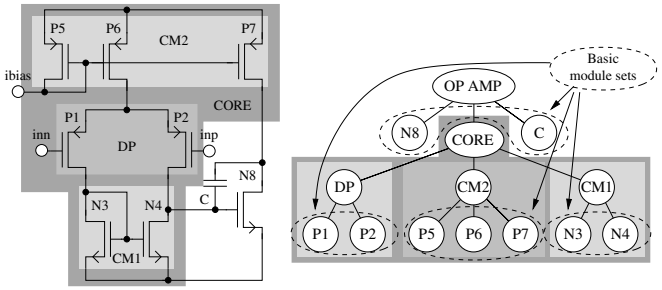

Fig. 13. Miller op amp schematic and hierarchy tree

Algorithm 3: enumerateOnHierarchyLevelOf(element)

```
begin
    resultESF ← empty enhanced shape function;
    if element's children are modules only then
        basicEnumeration(element's modules);
        store resulting enhanced shape function in resultESF;
    else
        ESFList ← empty list of enhanced shape functions;
        // Generate enhanced shape functions for the children
        forall children of element do
            childESF ← enumerateOnHierarchyLevelOf(child);
            store childESF in ESFList;
        // Try all combinations of the enhanced shape functions
        forall combination sequences of the enhanced shape
        functions in ESFList do
            forall enhanced shapes in two enhanced shape
            functions to be added do
                combine B*-trees of the enhanced shapes
                (Section III-C);
                generate placements for B*-trees for evaluation
                (Section II);
                append resulting enhanced shapes to resultESF;
        drop suboptimals from resultESF;
    return resultESF;
end
```

for a capacitor. Variant constraints are considered in this step. The result of the basic enumeration is an enhanced shape function storing only the placements which potentially contribute to a good result for the whole circuit. Since the basic enumerations are independent of each other, they can be parallelized easily.

After Plantage has determined the enhanced shape functions of all basic module sets, the algorithm steps up to the next hierarchy level. At this level, the enhanced shape functions of the basic module sets are combined in every possible sequence (see Section III-C). This is described in Algorithm 3, which is the key algorithm in this approach. In the example given by Fig. 13, the algorithm combines the enhanced shape functions of the differential pair DP and the two current mirrors CM1 and CM2 in every possible sequence[3]. These sequences are DP+CM1+CM2, DP+CM2+CM1, CM1+DP+CM2, CM2+DP+CM1, CM1+CM2+DP, CM2+CM1+DP. Then, the resulting enhanced shape function is combined in every possible sequence with the enhanced shape function of the basic module set $\{C, N8\}$.

The hierarchy is also used to fulfill proximity constraints. Because the addition is performed based on a sequence defined by the hierarchy, modules will be placed in close proximity, which are close to each other in hierarchy.

Finally, the result of the presented approach is a set of possible placements for the circuit, having different aspect ratios.

[3]Enhanced shape function additions are not commutative.

V. Experimental Results

The approach proposed in this paper was implemented in C++. All results were computed on a Pentium 4, running at 3.2GHz with 1024 MB RAM on Fedora Linux. To demonstrate the approach, placements for different circuits are shown and discussed in Section V-A. Publicly available benchmark circuits for analog placement do not yet exist. To compare Plantage with other placement methods, two circuits extracted from [18] are used. The comparison is discussed in Section V-B.

A. Discussion of the Presented Approach

To demonstrate the effectiveness of the proposed approach, four different circuits were placed. The results of the conducted experiments are summarized in Table I. The Examples 3 and 4 are discussed in more detail describing their constraints and where they can be seen in the placements. The sizings of the modules originate from a large semiconductor manufacturer and are taken from an up-to-date process library. Thus, they can be considered representative of current analog circuits.

Example 3 is a folded cascode op amp, consisting of 22 modules. The schematic of this circuit and its constraints are shown in Fig. 14, together with the hierarchy tree. There are three device proximity constraints, a symmetry constraint, nine variant constraints, and a minimum distance constraint between the nMOS and pMOS transistors. Plantage generates 12 different placements with different aspect ratios in 44 seconds. Building the hierarchy tree takes 0.5 seconds. Fig. 15 shows the Pareto front and three placements. The symmetry group is colored light gray. The placement is dominated by four big modules P7-P10, causing a corner in the Pareto front, marked with a "$\star$". An example of the close proximity constraints is that N1-N6 must always be placed close to each other. The area usages of the placements are always above 121%, because of empty areas caused by the minimum distance constraints between nMOS and pMOS transistors.

Example 4 is a CMOS buffer amplifier similar to [28], shown in Fig. 16. The complete circuit consists of 46 modules. The two differential pairs DP1 (N1, N2) and DP2 (P3, P4) are replaced by 2 common centroid structures (N1a, N1b, N2a, N2b) and (P3a, P3b, P4a, P4b), respectively. For this circuit, two common centroid constraints, twelve variant constraints, two device proximity constraints, as well as a minimum distance constraint between the nMOS and pMOS transistors are formulated. Three placements are shown in the Figs. 17(b), (c), and (d). As demonstrated by the figures, a minimum distance constraint is kept between nMOS and pMOS transistors. In the placements, DP1 and DP2 are colored light gray. Since DP1 and DP2 are close to each other in hierarchy, they are placed in

Example	**1**	**2**	**3**	**4**
Name	**Miller**	**Comparator**	**Folded cascode**	**Buffer**
# of modules	13	10	22	46
# of variants per module	3-6	2	2-4	2-7
Constraints				
# Device proximity	3	1	3	2
# Symmetry	2	1	1	0
# Common centroid	1	1	0	2
# Minimum distance	2	1	1	1
# Variant	3	5	9	12
Results				
# of placements	35	4	12	114
Best area usage	115%	110%	121%	111%
Runtimes in seconds				
Plantage	14	1	44	134
Building the hierarchy tree	0.3	0.3	0.5	1.2

TABLE I
SUMMARY OF EXAMPLE CIRCUITS AND RESULTS GENERATED BY THE PROPOSED APPROACH.

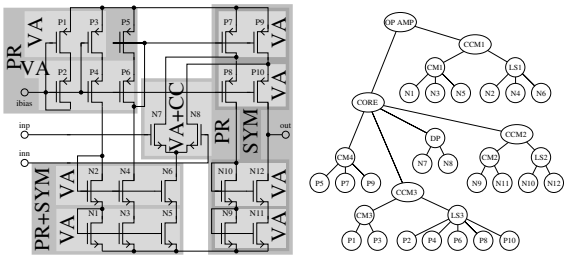

Fig. 14. Example 3: Schematic (left) and hierarchy tree (right) of the folded cascode op amp. There are variant (VA), symmetry (SYM), and device proximity (PR) constraints.

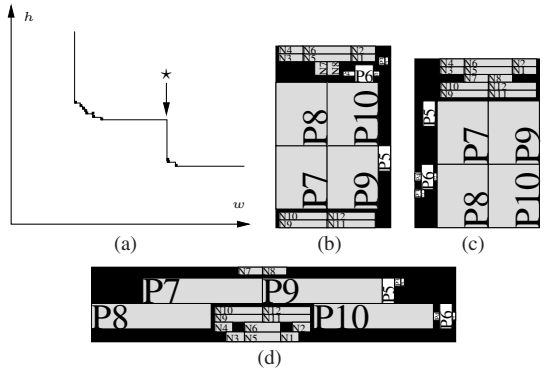

Fig. 15. Example 3: Placements and the corresponding Pareto front

close proximity. The same applies to the modules P22 and N24, as well as P25, N26, P27, N28, all of them are marked dark gray. For several transistors, different variants with different sizings are given representing different numbers of fingers of the transistor gate. For example, P39 has three different sizes in the shown placements. The Pareto front is depicted for this circuit in Fig. 17(a). It represents 114 different placements, having different aspect ratios. Plantage calculates these placements in 134 seconds. The hierarchy tree was built in 1.2 seconds.

B. Comparison with Other Approaches

In the following, the presented approach is compared with other approaches. For that reason, placements were generated using Plantage for the circuits "biasynth_2p4g" and "lnamixbias_2p4g" used in [18], [16], [29], [22], [21], and [20]. The module sizings were extracted from [16]. For "biasynth_2p4g", Plantage generates 10 placements in 337 seconds. For "lnamixbias_2p4g", 32 placements are generated in 387 seconds. Since no netlist information is given, balanced trees with 4-6 children per node are taken as the hierarchy trees. The placements with the lowest area usages are shown in Figs. 18 and 19, respectively. The symmetry groups within these circuits are colored in different shades of gray. The runtimes and area usages of other approaches are taken from [21] and [20]. The area usage is taken as a quality metric.

The results, summarized in Table II, show, that the area usages of the presented approach are approximately equal to the area usages of the best recently-published placers. The area usage of Plantage is 1% and 2% higher than the best for the two examples, respectively, but about 10% lower than area usage of other approaches.

It is hard to compare the runtimes of the presented approach with the runtimes of other approaches, because they were measured on different computers. Additionally, other approaches use Simulated Annealing. Their runtimes may vary from run to run for different seeds, but no mean values and standard deviations are known for the runtimes of other approaches. In contrast, Plantage is deterministic, and the runtimes are constant for each example. On all accounts, the runtimes of Plantage, even for the big circuit "lnamixbias_2p4g" (110 modules), allow industrial application.

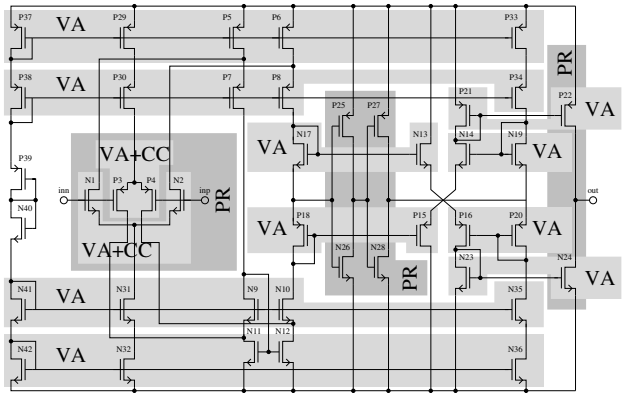

Fig. 16. Example 4: Schematic of the buffer amplifier similar to [28]. There are common centroid (CC), variant (VA), and device proximity (PR) constraints.

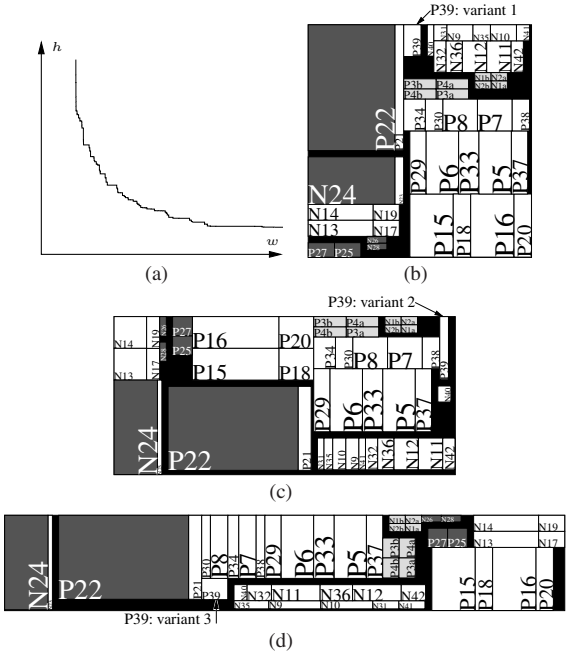

Fig. 17. Example 4: Placements and the corresponding Pareto front

VI. Conclusion

In this paper, a new deterministic approach for analog circuit placement, called Plantage, was introduced. The hierarchy of an analog circuit is used to guide a bottom-up enumeration efficiently. For small parts of the circuit all placements are enumerated. A new concept, the enhanced shape function, is used to combine these placements efficiently with a recursive algorithm based on the hierarchy. In contrast to other approaches, the final result of Plantage is a set of placements instead of a single solution. A new algorithm is presented, which generates a placement for a B*-tree considering linear constraints. Plantage considers device-proximity, symmetry, common centroid, minimum distance, and variant constraints. The presented approach is the first to handle all these constraints deterministically. The results of the presented approach show an area usage which is comparable to the best-published results of other placers. Plantage generates results in reasonable time allowing industrial application.

Circuit description				Approaches													
Name	# of Modules	# of Sym. Mods.	Mod. Area ($10^3 \mu m^2$)	SP [18] Area	SP [18] Time*	ST [16] Area	ST [16] Time*	SP+LP [29] Area	SP+LP [29] Time†	SPwD [22] Area	SPwD [22] Time†	SymIs [21] Area	SymIs [21] Time†	SP+JPQ [20] Area	SP+JPQ [20] Time*	**This work** Area	**This work** Time†
biasynth_2p4g	65	8 + 12 + 5	$4.70 \hat{=} 100\%$	114.89	780	114.89	246	106.38	403	118.51	134	104.68	22	N/A	N/A	**104.96**	**337**
lnamixbias_2p4g	110	16 + 6 + 6 + 12 + 4	$46.00 \hat{=} 100\%$	110.43	2824	109.35	726	108.59	3252	113.50	227	105.72	43	109	480	**107.68**	**387**

TABLE II

COMPARISON OF AREA USAGE AND RUNTIMES FOR DIFFERENT APPROACHES: THE SEQUENCE PAIR (SP), SEGMENT TREE (ST), SEQUENCE PAIR AND LINEAR PROGRAMMING (SP+LP), SEQUENCE PAIR WITH DUMMY NODES (SPWD), SYMMETRY ISLANDS (SYMIS), SEQUENCE PAIR WITH JOHNSON'S PRIORITY QUEUE(SP+JPQ)[4], AND THE PRESENTED APPROACH, BASED ON TWO INDUSTRIAL CIRCUITS. ALL TIMES ARE MEASURED IN SECONDS[5] AND ALL AREA USAGES IN % OF THE TOTAL MODULE AREA.

Fig. 18. Result of "biasynth_2p4g" obtained by the approach of this paper. (Time: 5.6min, Area usage: 104.96%)

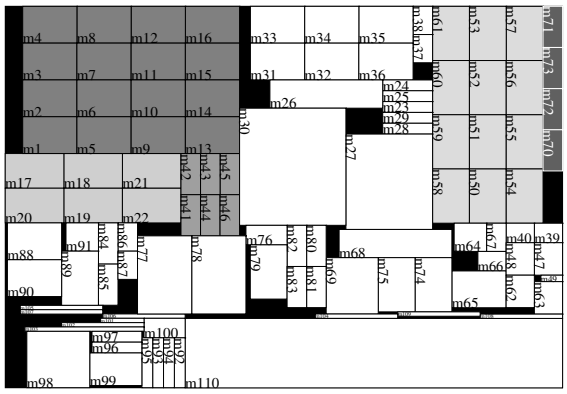

Fig. 19. Result of "lnamixbias_2p4g" obtained by the approach of this paper (rotated by 90° clockwise). (Time: 6.4min, Area usage: 107.68%)

REFERENCES

[1] Rob A. Rutenbar, L. Richard Carley, John M. Cohn, and David J. Garrod. *Analog Device-Level Layout Automation*. Kluwer Academic Publishers, 1994.

[2] Alan Hastings. *The Art of Analog Layout*. Prentice-Hall, 2001.

[3] Enrico Malavasi and Alberto Sangiovanni-Vincentelli. Area routing for analog layout. *IEEE Transactions on Computer-Aided Design of Circuits and Systems*, 12(8):1186–1197, August 1993.

[4] D.W. Jepsen and C.D. Gellat Jr. Macro placement by monte carlo annealing. In *IEEE International Conference on Computer Design (ICCD)*, pages 495–498, 1983.

[5] John M. Cohn, David J. Garrod, Rob A. Rutenbar, and L. Richard Carley. Koan/anagram ii: New tools for device-level analog placement and routing. *IEEE Journal of Solid-State Circuits SC*, 26(3):330–342, March 1991.

[6] Koen Lampert, Georges Gielen, and Willy M. Sansen. A performance-driven placement tool for analog integrated circuits. *IEEE Journal of Solid-State Circuits SC*, 30(7):773–780, July 1995.

[7] Enrico Malavasi, Edoardo Charbon, Eric Felt, and Alberto Sangiovanni-Vincentelli. Automation of ic layout with analog constraints. *IEEE Transactions on Computer-Aided Design of Circuits and Systems*, 15(8):923–942, August 1996.

[8] Pei-Ning Guo, Chung-Kuan Cheng, and Takeshi Yoshimura. An o-tree representation of non-slicing floorplan and its applications. In *ACM/IEEE Design Automation Conference (DAC)*, volume 36, pages 268–273, June 1999.

[9] H. Murata, K. Fujiyoshi, S. Nakatake, and Kajitani. VLSI module placement based on rectangle-packing by the sequence-pair. *IEEE Transactions on Computer-Aided Design of Circuits and Systems*, 15(12):1518–1524, 1996.

[10] S. Nakatake, K. Fujiyoshi, H. Murata, and Y. Kajitani. Module placement on BSG-structure and IC layout applications. In *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 484–493, 1996.

[11] Yingxin Pang, Florin Balasa, Koen Lampaert, and Chung-Kuan Cheng. Block placement with symmetry constraints based on the o-tree non-slicing representation. In *ACM/IEEE Design Automation Conference (DAC)*, pages 464–468, June 2000.

[12] X. Hong., G. Huang, Y. Cai, J. Gu, S. Dong, C.-K. Cheng, and J. Gu. Corner block list: An effective and efficient topological representation of non-slicing floorplan. In *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, November 2000.

[13] Qiang Ma, Evangeline F. Y. Yong, and K. P. Pun. Analog placement with common centroid constraints. In *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, November 2007.

[14] Jai-Ming Lin and Yao-Wen Chang. Tcg-s: Orthogonal coupling of p-admissible representations for general floorplans. *IEEE Transactions on Computer-Aided Design of Circuits and Systems*, 23(6):968–980, June 2004.

[15] Yun-Chih Chang, Yao-Wen Chang, Guang-Ming Wu, and Shu-Wei Wu. B*-trees: A new representation for non-slicing floorplans. In *ACM/IEEE Design Automation Conference (DAC)*, volume 37, pages 458–463, 2000.

[16] Florin Balasa, Sarat C. Maruvada, and Karthik Krishnamoorthy. On the exploration of the solution space in analog placement with symmetry constraints. *IEEE Transactions on Computer-Aided Design of Circuits and Systems*, 23(2):177–191, February 2004.

[17] Ammar Nassaj, Jens Lienig, and Göran Jerke. A constraint-driven methodology for placement of analog and mixed-signal integrated circuits. In *IEEE International Conference on Electronics, Circuits and Systems (ICECS)*, 2008.

[18] Florian Balasa and Koen Lampaert. Symmetry within the sequence-pair representation in the context of placement for analog design. *IEEE Transactions on Computer-Aided Design of Circuits and Systems*, 19(7):721–731, July 2000.

[19] Karthik Krishnamoorthy, Sarat C. Maruvada, and Florin Balasa. Fast evaluation of symmetric-feasible sequence-pairs for analog topological placement. In *5th IEEE Int. Conf. on ASIC (ASICON)*, pages 71–74, 2003.

[20] Karthik Krishnamoorthy, Sarat C. Maruvada, and Florin Balasa. Topological placement with multiple symmetry groups of devices for analog layout design. In *IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 2032–2035, May 2007.

[21] Lin Po-Hung and Lin Shyh-Chang. Analog placement based on novel symmetry-island formulation. In *ACM/IEEE Design Automation Conference (DAC)*, pages 465–470, June 2007.

[22] Yiu-Cheong Tam, Evangeline F. Y. Young, and Chris Chu. Analog placement with symmetry and other placement constraints. In *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, November 2006.

[23] David A. Johns and Ken Martin. *Analog Integrated Circuit Design*. John Wiley & Sons, 1997.

[24] H. Graeb, S. Zizala, J. Eckmueller, and K. Antreich. The sizing rules method for analog integrated circuit design. In *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 343–349, 2001.

[25] Tobias Massier, Helmut Graeb, and Ulf Schlichtmann. Sizing rules for bipolar analog circuit design. In *Design, Automation and Test in Europe (DATE)*, March 2008.

[26] R.H.J.M. Otten. Efficient floorplan optimization. In *IEEE International Conference on Computer Design (ICCD)*, pages 499–501, October 1983.

[27] Gerhard Zimmermann. A new area and shape function estimation technique for VLSI layouts. In *ACM/IEEE Design Automation Conference (DAC)*, volume 25, pages 60–65, 1988.

[28] J. Fisher and R. Koch. A highly linear CMOS buffer amplifier. *IEEE Journal of Solid-State Circuits SC*, 22:330–334, 1987.

[29] Shinichi Kouda, Chikaaki Kodama, and Kunihiro Fujiyoshi. Improved method of cell placement with symmetry constraints for analog ic layout design. In *ACM/SIGDA International Symposium on Physical Design (ISPD)*, April 2006.

[4] In [20], only the placement of "lnamixbias_2p4g" is shown. No area usage is given in that paper. For comparison, the area usage in Table II was calculated based on Fig. 3 of [20].

[5] Times which are marked with $\star$ were measured on a Sun Blade 100, 500MHz, and times which are marked with † were measured on a Pentium 4, 3.2 GHz.