Efficient Solution Space Exploration Based on Segment Trees in Analog Placement with Symmetry Constraints*

Florin Balasa Sarat C. Maruvada Karthik Krishnamoorthy

Dept. of Computer Science, University of Illinois at Chicago

Abstract – The traditional way of approaching device-level placement problems for analog layout is to explore a huge search space of absolute placement representations, where cells are allowed to illegally overlap during their moves [2, 7, 8]. This paper presents a novel analog placement technique operating on the set of binary tree representations of the layout [3], where the typical presence of an arbitrary number of symmetry groups of devices is directly taken into account during the exploration of the solution space. The efficiency of the novel approach is due to a data structure called *segment tree*, mainly used in computational geometry.

1 Introduction

In high-performance analog circuits it is often required that groups of devices are placed symmetrically with respect to one or several axes. The main reason of symmetric placement and routing is to match the layout-induced parasitics in the two halves of a group of devices. Failure to match these parasitics in, for instance, differential analog circuits can lead to higher offset voltages and degraded power-supply rejection ratio [2]. Placement symmetry can also be used to reduce the circuit sensitivity to thermal gradients. Failure to adequately balance thermal couplings in a differential circuit can even introduce unwanted oscillations. In order to combat potentially-induced mismatches, the thermally-sensitive device couples should be placed symmetrically relative to the thermally-radiating devices.

The problems related to analog placement are numerous: for instance, eliminating systematically-induced mismatches due to dissimilar geometrical choices for matching devices identically specified; arranging devices such that critical structures are shared in common in order to reduce both layout density and induced parasitics; handling thermal constraints, dealing with noise coupling, or clustering based on oxide thickness. Some of the analog-specific constraints can be handled during a combinatorial optimization solving the placement problem by embedding accurate models in the cost function. Different from these, symmetry constraints can be directly taken into account during the exploration of the solution space, diminishing significantly its size when placement configurations are represented topologically.

2 Exploration of the solution space in device-level analog placement

The traditional way of approaching device-level placement problems for analog layout is to explore a huge search space using, for instance, simulated annealing as the optimization engine, where the cells are represented by means of absolute coordinates allowing illegal overlaps during their moves (translations, changes of orientation) in the chip plane, since no restriction is made referring to the relative position of a cell with respect to another cell. A penalty cost term is associated with infeasible overlaps, and this penalty must be driven to zero in the optimization process. This exploration based on the *absolute* representation is used by placement tools in several software systems for analog layout [2, 7, 8].

An alternative approach is to use a *topological* representation encoding the positioning (topological) relations between any pair of cells. While placement techniques based on the absolute representation trade-off a larger number of moves (in a combinatorial optimization framework) for easier- and quicker-to-build layout configurations – not always physically achievable, the techniques adopting topological representations trade-off a smaller number of moves for more complex-to-build, feasible layouts. Note that the use of topological representations in analog design do not raise any scalability problems, since analog circuits and the analog portions of mixed-signal systems-on-a-chip are significantly smaller than digital circuits – usually up to 100 devices in an analog block (and less than 20,000 devices in a complete subsystem) [2].

2.1 Brief overview of nonslicing topological representations

The first remarkable encoding system not restricted to slicing floorplan topologies was proposed by Murata *et al.*, who suggested to encode the "left-right" and "above-below" positioning relations between blocks using two sequences of block permutations, named *sequence-pair* [9]. A $O(n^2)$ algorithm based on constructing a pair of horizontal and vertical constraint graphs was used for sequence-pair evaluation. More recently, a different approach of $O(n \log\log n)$ complexity – based on computing the longest common subsequence in a pair of weighted sequences – was proposed by Tang *et al.* [13]. Nakatake *et al.* devised a meta-grid structure (called *bounded-sliceline grid* or *BSG*) without physical dimensions to define orthogonal relations between blocks, where the construction of the placement configuration is of quadratic complexity [10]. Guo *et al.* proposed the *ordered tree* (*O-tree*) data structure to reduce the drawback effect of redundancies in the two previous representations [4]. Chang *et al.* [3] suggested a representation based on binary trees. This encoding is based on the so-called *natural correspondence* between forests of rooted trees and binary trees [6], and therefore it could be regarded as a representation equivalent to the ordered trees [4]. The corner block list, proposed more recently [5], is a representation that can be used

*This research was partly sponsored by the Design Automation Conference Graduate Scholarship Program.

0-7803-7607-2/02/$17.00 ©2002 IEEE

to encode floorplans with zero dead-space ("mosaic" floorplans). Different from the tree- based representations [4, 3], the sequence-pair, the bounded-sliceline grid, and the corner block list define the topological relations between blocks independent of their dimensions.

2.2 Selecting a topological representation for analog placement

At a first glance, the main selection criteria should be a reduced redundancy of the topological representation of choice and the existence of an efficient code evaluation algorithm (of complexity $O(n)$) which builds the placement configuration from the current code. Without denying the importance of the above criteria, such a selection analysis is superficial since it ignores that many analog designs contain – along an asymmetric component – an arbitrary number of symmetry groups of devices (that is, groups having distinct symmetry axes), each group containing an arbitrary number of pairs of symmetric devices, as well as self-symmetric devices – presenting a geometrical symmetry and sharing the same axis with its group. Due to this characteristic, *most of the codes of any topological representation would be infeasible in symmetry point of view.* Only a tiny fraction of the solution space of a topological representation contains symmetric-feasible codes (relative to a given set of constraints), as exemplified in [1] for the sequence-pair representation.

While the absolute representation is well-suited to handle symmetry constraints since they can be algebraically modeled in a straightforward way, within topological representations symmetry is difficult to model, since separating the codes leading to placements which are feasible in symmetry point of view from the infeasible representations is far less than obvious. Moreover, maintaining the "symmetric-feasibility" of the codes during the annealer's moves is, in general, a nontrivial task, very specific to the topological representation employed.

Extensive experimental work [1] led to the conclusion that a topological representation is a good candidate for the exploration of the solution space in analog placement only if one can find a property characterizing those codes able to generate placements such that symmetry constraints are satisfied. Such a property would allow to efficiently restrict the exploration only to the subspace of these "symmetric-feasible" encodings. In the absence of such a property, the fact that a certain representation has an evaluation algorithm of linear complexity (able to deal only with the typical positioning constraints) is close to irrelevance since most of the codes would be infeasible. This is the reason why the only efficient topological exploration techniques for analog placement existent so far are using sequence-pairs [1] and trees (e.g., [11]), while the corner block list [5] seems quite unfit for this type of problems.

This paper presents a novel analog placement approach operating on a subset of the binary tree representations [3], where the typical presence of an arbitrary number of symmetry groups of devices is directly taken into account during the exploration of the solution space. The efficiency of the novel method is due to a data structure – mainly used in computational geometry – called *segment tree* [12].

Section 3 will present a novel $O(n \log n)$ evaluation algorithm which complexity is better than of any other existent topological algorithm supporting symmetry. Section 4 will give an overview of the experimental results, showing that the novel approach is superior to both the traditional exploration technique and all the newer methods using different topological encodings. Finally, Section 5 will present the basic conclusions of this research.

3 Placement with symmetry using a segment tree data structure

The evaluation algorithm described below – executed in each inner-loop iteration of the simulated annealing – assumes that the given placement encoding is a binary tree; but due to the *natural correspondence* [6], it can operate with minor modifications if the encoding is an ordered tree. Moreover, this binary tree is assumed to possess property (1), which represents a sufficient condition for building a placement satisfying the symmetry constraints[1] : for any distinct cells A, B in a symmetry group,

$$A \overset{inorder}{\prec} B \iff sym(B) \overset{preorder}{\prec} sym(A) \qquad (1)$$

that is, node A *precedes* B in the inorder traversal of the binary tree [6] iff node $sym(A)$ – corresponding to cell A's symmetric pair – *succeeds* node $sym(B)$ in the preorder traversal (i.e., visit the root, traverse the left subtree, traverse the right subtree). An example of such a binary tree will be given in Section 3.1.

A binary tree which nodes represent rectangular blocks imposes the following vertical (y-) and horizontal (x-) positioning constraints: (a) each block in the left subtree is above its parent block; (b) if the y- projections of two blocks are overlapping, the block visited the first in a preorder traversal is to the left of the block visited the second.

The analog devices to be placed on the chip area are represented by rectangular blocks $B_1, \ldots, B_n$, each block B_i having the width w_i and the height h_i, and having (x_i, y_i) as coordinates of its left-bottom corner.

3.1 Computation of the block ordinates

The computation of the y- coordinates is performed with at most two preorder traversals of the binary tree encoding of the device placement. In the absence of symmetry constraints, one traversal of the tree is sufficient; however, a second traversal may be necessary in order to satisfy all the y- symmetry constraints of the type: $y_i = y_j$ for two symmetric devices B_i and B_j. The fact that the binary tree encoding satisfies the property (1) is very important: it ensures that after at most two traversals the y- coordinates of the blocks are correctly computed.

`initialize` $y_i = 0$; $repeat\_traversal = False$;
for `each node` B_i (visited in a preorder traversal)
  if `node` B_k `is the closest ancestor of` B_i (if any)
    `such that` B_i `is in the left subtree of` B_k
  then $y_i = \max\{y_i, y_k + h_k\}$; **else** $y_i = \max\{y_i, 0\}$;
  if `blocks` (B_j, B_i) `are symmetric`
    if B_j `has already been visited and` $y_j < y_i$

[1] The annealer moves can be designed to preserve property (1).

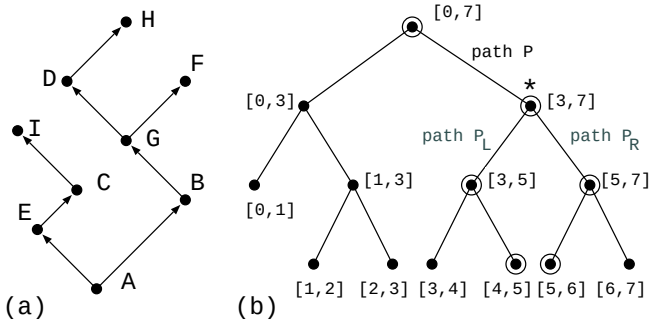

Figure 1: (a) Binary tree symmetric-feasible; (b) the complete segment tree for the root interval [0,7], and the nodes visited during the insertion of the interval [4,6]

then $repeat\_traversal = True$;
$y_j = y_i$;
end_for
if $repeat\_traversal$ **then** repeat the tree traversal;

Example Consider the binary tree representation in Fig. 1(a) for nine rectangular blocks having the widths and heights indicated: A(5x2), B(5x2), C(1x1), D(1x1), E(2x3), F(2x3), G(5x1), H(3x3), and I(2x3). Let us assume that the pairs of cells (A,B), (C,D), and (E,F) are respectively symmetric and they must be placed to form a symmetry group, sharing the same symmetry axis. Note that the binary tree in Fig. 1(a) satisfies the property (1) relative to these symmetry constraints. For instance, node C is visited *before* A in the inorder traversal of the tree (EICADHGFB), but in the preorder visit (AECIBGDHF) node D appears *after* B (the symmetric pairs of C and A).

The computation of the y- coordinates yields successively (preorder visit) $y_A = y_B = 0$; $y_E = y_F = 2$; $y_C = y_D = 2$; $y_I = 3$; $y_B = y_A = 0$; $y_G = 2$; $y_D = y_C = 3$; $y_H = 3$; $y_F = y_E = 2$. For instance, $y_D = \max\{y_D, y_G + h_G\} = \max\{2, 2+1\} = 3$ since G is the closest ancestor of D, having D in its left subtree. Note that after y_D is computed, the ordinate of its symmetric y_C is equally modified from 2 to 3. But the computation of y_I used the old value of y_C. In the second traversal y_I is recomputed and becomes equal to 4.

The computation of y_i's is done in $O(n)$ time basically since the traversal of the tree encoding is performed in linear time. The correctness of the algorithm results from the following theorem (stated without proof due to lack of space):

Theorem 1: Given a binary tree satisfying property (1), the algorithm computes the ordinates of the blocks such that the y- positioning and symmetry constraints are verified.

3.2 Computation of the block abscissae

The subtle part of the algorithm computing the block abscissae is the use of a *segment tree*, a data structure mainly employed in computational geometry, originally introduced by Bentley [12], designed to handle operations with intervals whose extremes belong to a fixed set of coordinates[2].

The (complete) segment tree is, basically, a rooted binary tree, where each node v has attached an interval $v.I = [c, d]$ with integer bounds. If $d - c > 1$, then node v has a left and a right descendant – denoted below as $v.left$ and $v.right$ – having associated the intervals $[c, \lfloor \frac{c+d}{2} \rfloor]$ and, respectively, $[\lfloor \frac{c+d}{2} \rfloor, d]$. The intervals attached to the nodes are called *standard*, while those pertaining to the leaves and having the length equal to 1 are named *elementary*. The complete segment tree is balanced, its depth being $\lceil \log_2(d-c) \rceil$ [12]. A complete segment tree is shown in Fig. 1(b).

The computation of the block abscissae can be performed during only two traversals of the binary tree code if the tree satisfies property (1) (see *Theorem 2*). During the first traversal a segment tree data structure will be gradually built. In our application, the nodes v of the segment tree have also attached – in addition to the interval $v.I$ – a value $v.x$ used for the computation of the cell abscissae x_i. The segment tree will represent after each iteration the contour of the *right* border of the (partial) placement configuration.

The second traversal of the binary tree encoding will only update the values $v.x$ of the nodes in the existent segment tree – which will describe this time the contour of the *left* border of the placement. All the x- symmetry constraints of the form $(x_i + w_i) + x_j = 2 \cdot x_{symAxis}$ for pairs of symmetric devices (B_i, B_j) will be eventually satisfied.

For the sake of readability, we shall present a simplified algorithm, assuming, for instance, that all the devices subject to symmetry constraints belong to a single symmetry group. However, the implementation is more refined, able to deal with an arbitrary number of symmetry groups. Self-symmetric devices are handled as well. Notice that most of the circuits in Table 1 contain several symmetry groups.

First, the y- coordinates of the blocks are normalized by replacing each of them by its rank in their bottom-up order. Note that the algorithm operates with intervals $[a_i, b_i]$ rather than $[y_i, y_i + h_i]$, where a_i, b_i are the indices of y_i and, respectively, $y_i + h_i$ in S – the sorted set of the interval endpoints. In this way, the size of the segment tree will be kept minimal. Without loss of generality, the y- coordinates can hence be considered integers in the range $[0, n]$. The first traversal of the binary tree encoding is shown below:

```
initialize  x_i = 0  and  x_symAxis = 0;
sort increasingly the set S = {y_i} ∪ {y_i + h_i} ,
   eliminating the duplicate values;
let m be the number of elements of set S;
SegmTreeNode v_0 = CreateNode ([0, m-1], 0);
for each block B_i (visited in a preorder traversal)
  let a_i be the index of y_i, and
  let b_i be the index of y_i + h_i in set S;
  UpdateSegTree (v_0, [a_i, b_i]);
  if blocks (B_j, B_i) are symmetric
    if node B_j has already been visited
    then x_i = max{x_i , 2x_symAxis - (x_j + w_j)};
    else x_symAxis = max{x_symAxis , x_i + w_i};
  UpdateRightContour (v_0, [a_i, b_i]);
end_for
```

After the ordinate normalization, the segment tree is recursively built by the procedure *UpdateSegTree*. The *CreateNode* procedure is the constructor of a new node v in the segment tree – the two parameters being the interval $v.I$ and the value $v.x$. The procedure *UpdateSegTree* is inserting the normalized interval of $[y_i, y_i + h_i]$ into the segment tree, decomposing it into standard intervals. At the same time, block B_i's abscissa x_i is computed by taking the maximum

[2]Since y_i are already known in this phase, the extremes of the intervals defining the left and right border contours – the elements of the set $S = \{y_i\} \cup \{y_i + h_i\}$ – are currently fixed.

over all the values $v.x$ of the nodes with standard intervals.

procedure *UpdateSegTree* $(v, [a,b])$
  if $v.I \subseteq [a,b]$ **then** $x_i = \max\{x_i, v.x\}$;
  else let $v.I = [c,d]$ and $mid = \lfloor \frac{c+d}{2} \rfloor$;
    if $v.left == Null$ **then**
      $v.left = CreateNode\ ([c, mid], 0)$;
      $v.right = CreateNode\ ([mid, d], 0)$;
    if $v.x > \max\{v.left.x, v.right.x\}$
      then $v.left.x = v.right.x = v.x$;
    if $a < mid$ **then** $UpdateSegTree(v.left, [a,b])$;
    if $mid < b$ **then** $UpdateSegTree(v.right, [a,b])$;
end_procedure

UpdateSegTree visits the nodes in the segment tree along a tour having the following general structure (see Fig. 1(b)): a (possibly empty) initial path P from the root to a node called the *fork* – marked with a star in the figure, from which two (possibly empty) paths P_L and P_R issue. Either the interval being inserted is allocated entirely to the fork (in which case P_L and P_R are both empty), or all the right sons of nodes of P_L , as well as all the left sons of nodes of P_R identify the fragmentation of $[a,b]$ into standard intervals. Subsequently, the procedure *UpdateRightContour* sets the values of all the nodes corresponding to these standard intervals to $x_i + w_i$ – the abscissa of B_i's right border.

procedure *UpdateRightContour* $(v, [a,b])$
  if $v.I \subseteq [a,b]$ **then** $v.x = x_i + w_i$;
  else let $v.I = [c,d]$ and $mid = \lfloor \frac{c+d}{2} \rfloor$;
    if $a < mid$ **then** $UpdateRightContour(v.left, [a,b])$;
    if $mid < b$ **then** $UpdateRightContour(v.right, [a,b])$;
    $v.x = \max\{v.left.x, v.right.x\}$;
end_procedure

Example (cont'd) All the cell abscissae are initially zero. After the sorting and elimination of duplicates, set $S = \{y_i\} \cup \{y_i + h_i\} = \{0,2,3,4,5,6,7\}$ has $m = 7$ elements. Due to the normalization, the root node v_0 of the segment tree has associated the interval [0,6]. The interval $[y_A, y_A + h_A] = [0,2]$ of the first node visited in the preorder traversal is normalized to $[a,b] = [0,1]$ (since the indices of the elements 0 and 2 in set S are 0 and, respectively, 1). As $v_0.I = [0,6]$ and $mid = 3$, two new nodes v_1 and v_2 are created having attached the intervals [0,3] and [3,6]. $UpdateSegTree(v_1, [0,1])$ is then recursively called and two new nodes v_3 and v_4 – having the intervals [0,1] and [1,3] – are created (Fig. 2(a)). The execution of $UpdateSegTree(v_0, [0,1])$ yields $x_A = \max\{v_3.x, x_A\} = 0$ since $v_3.I = [0,1]$. Afterwards, *UpdateRightContour* will visit once again the same nodes in the segment tree in order to update the values of the nodes. In this case, the only node is v_3 (which is actually the "fork", the paths P_L and P_R being empty): therefore, $v_3.x = x_A + w_A = 5$.

Figures 2(a-i) display the segment tree after each iteration, the cells being successively placed in the preorder traversal after the insertion of their normalized y- spanning intervals $[a_i, b_i]$ in the segment tree. The nodes corresponding to the standard segments are represented as double circles, while the "fork" nodes are marked with a star. The final value of the root, $v_0.x = 11$, is the current width of the chip. The placement after the first traversal is shown in Fig. 2(i). Note that the x- symmetry constraint is still not satisfied for the pair of cells (E,F): indeed, $((x_E + w_E) + x_F = 11) > (2x_{symAxis} = 10)$.

In the first traversal, the position of the symmetry axis is determined and the rightmost blocks in the symmetry pairs are positioned attempting to meet the $x-$ symmetry constraints $(x_j + w_j) + x_i = 2x_{symAxis}$. However, due to the positioning constraints, some of these blocks may be pushed further to the right. A second traversal in the inverse order will reposition the left symmetric blocks (and other blocks to their left) to meet the $x-$ symmetry constraints. First, the values of the nodes in the segment tree are initialized to the right border abscissa. Afterwards, the blocks are inserted one by one in the reverse order of the precedent traversal, updating after each insertion the contour of the left border of the partial placement. The structure of the segment tree will not be modified anymore: only the abscissae x_i and the values $v.x$ will be differently updated (e.g., $x_i = \min\{x_i, v.x - w_i\}$).

for all the nodes v in the segment tree
  set $v.x = v_0.x$; // all values equal to the root one
for each block B_i (visited in *inverse* preorder)
  let a_i, b_i be the indices of y_i, $y_i + h_i$ in set S;
  $UpdateSegTree\ (v_0, [a_i, b_i])$;
  if blocks (B_i, B_j) are symmetric
    if node B_j has already been visited
    then $x_i = 2x_{symAxis} - (x_j + w_j)$;
  $UpdateLeftContour\ (v_0, [a_i, b_i])$;
end_for

Example (cont'd) All the node values are set to 11, the abscissa of the right border of the chip. Inserting the normalized y- spanning intervals in the reverse order as in Fig. 2(a-i), the final segment tree and placement – shown in Fig. 2(j) – are obtained. Note that, since x_E is now equal to -1, all the symmetry constraints are satisfied. The final value of $v_0.x$ (i.e., -1) is the abscissa of the left border of the chip.

The decomposition of the root segment into standard intervals is done in $O(\log n)$ time since the height of the segment tree is at most $\lceil \log_2(m-1) \rceil$, hence upper-bounded by $\lceil \log_2 n \rceil$. This entails the same complexity for the procedures *UpdateSegTree*, *UpdateRightContour*, *UpdateLeftContour*; the overall complexity is thus $O(n \log n)$. The correctness of the algorithm results from the following theorem:

Theorem 2: Given a binary tree satisfying property (1), the algorithm computes the block abscissae such that the x-positioning and symmetry constraints are verified.

4 Experimental results

A placement tool for analog layout using alternative exploration algorithms has been implemented. The tool can operate both with different topological representations (sequence-pairs and trees) and different code evaluation algorithms. In addition, a complementary placement algorithm based on the traditional absolute representation has been implemented as well. The tool uses the simulated annealing algorithm as the combinatorial optimization engine. In order to ensure a correct comparative evaluation, the simulated annealing schedule is identical for all the placement algorithms. The evaluation of a placement configuration is based on a complex cost function defined as a weighted sum of the chip area and the estimated total wire length, as

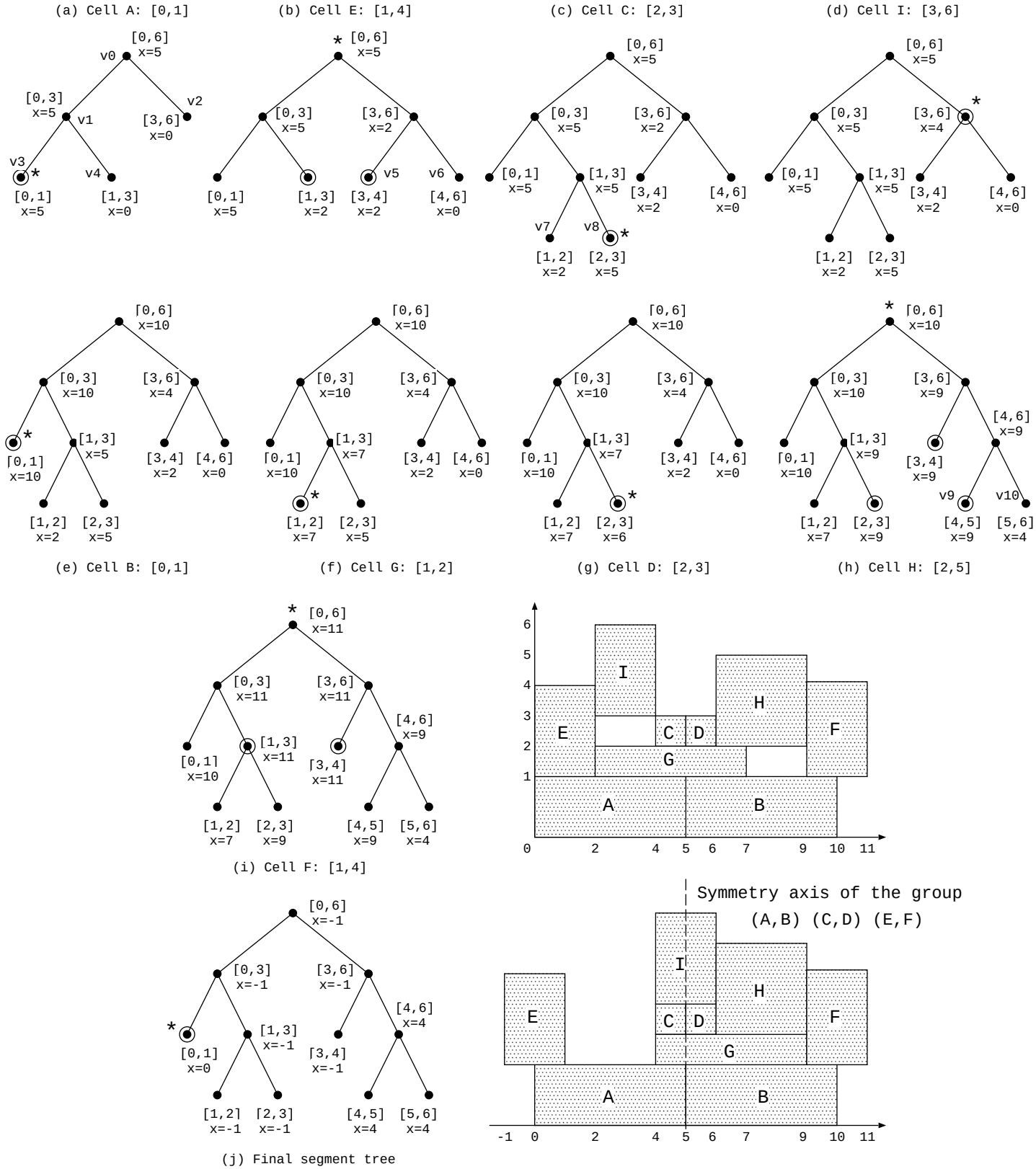

Figure 2: The segment tree evolution and the placements after each traversal (the y- coordinates are normalized)

Design	Nr. cells groups: cells	Seq-pairs [1]		O-trees [11]		SegmentTree	
		Time	Area	Time	Area	Time	Area
tx_current	47 (4: 27)	7.5	50.4	9.6	51.5	3.6	48.4
lpf2_b25b	52 (1: 23)	5.2	38.0	4.4	36.2	2.7	35.8
biasynth_2p4g	65 (3: 25)	12.6	5.4	14.8	5.6	7.2	5.4
modbias_2p4g	87 (5: 46)	28.8	56.9	32.7	59.5	15.1	57.9
lnamixbias_2p4g	110 (6: 40)	47.1	51.0	55.1	54.1	21.0	50.3

Table 1: Placement of analog blocks with symmetry constraints (Time [min], Area [$10^3 \times \mu m^2$]). Column 2 shows also the number of symmetry groups and their total size.

well as weighted penalty terms, for instance, modeling device separation constraints – which allow the designer to specify a maximum separation distance between pairs of critically matched devices. Besides symmetry constraints, the tool handles systematically-induced device mismatches, alignment constraints, and performs shape optimizations for parametric cells and for "soft" cells (like capacitors), with the aspect ratio varying continuously between given limits.

Table 1 displays the results of our experiments carried out on a SUN Blade 100 workstation. The benchmarks are several analog blocks containing symmetry groups of devices, components of a spread spectrum transceiver used in wireless modems. One of these blocks is displayed in Fig. 3.

Our experiments show that using topological representations is significantly better both in terms of computational effort and placement quality than using the more traditional absolute representation. The running times obtained when using our novel approach are regularly better than the running times obtained when using sequence-pairs [1], or ordered trees [11] (although in terms of placement quality all these algorithms proved to be quite effective). The advantage of the novel approach is twofold: the exploration space has a smaller size than the set of trees and, at the same time, the evaluation algorithm using a segment tree is more efficient. Note that the placement algorithm [1] explores only a *subset* of the sequence-pair representations, which can be smaller than the number of trees. As a result, the algorithm [1] usually runs faster than [11] when the symmetric component of the design is complex.

The running times are higher than those obtained by other block placement tools operating with topological representations on examples without symmetry (e.g., [13]). But this is perfectly normal since in the presence of symmetry constraints the moves within the simulated annealing optimizer are, typically, a few times more computationally expensive. Restricting the moves within the subset of symmetric-feasible codes is costly for sure, but this strategy is significantly better than the exploration of the entire solution space of the topological representation employed.

5 Conclusions

This paper has presented a novel analog placement technique operating on a subset of binary tree representations of the layout, where symmetry constraints – very typical in analog placement – are directly taken into account during the exploration of the solution space. The efficiency of the new approach is due to a data structure called *segment tree*, which is mainly used in computational geometry.

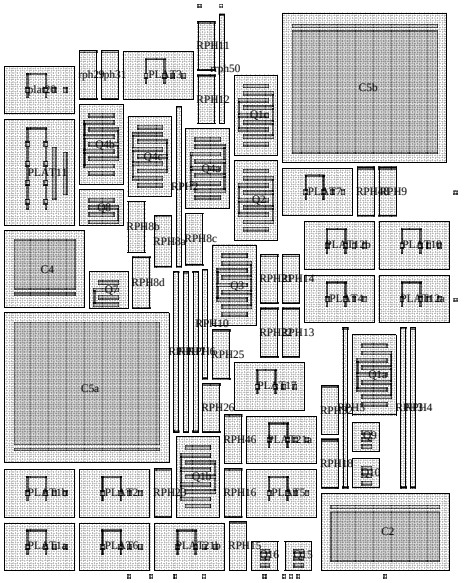

Figure 3: Placement for the analog block *biasynth_2p4g*

References

[1] F. Balasa, K. Lampaert, "Symmetry within the sequence-pair representation in the context of placement for analog design," *IEEE Trans. CAD*, Vol. 19, pp. 721-731, July 2000.

[2] J. Cohn, D. Garrod, R. Rutenbar, L. Carley, *Analog Device-Level Automation*, Kluwer Acad. Publ., 1994.

[3] Y.-C. Chang, Y.-W. Chang, G.-M. Wu, S.-W. Wu, "B*-Trees: a new representation for non-slicing floorplans," *Proc. 37th ACM/IEEE Des. Aut. Conf.*, pp. 458-463, 2000.

[4] P.-N. Guo, C.-K. Cheng, T. Yoshimura, "An O-tree representation of non-slicing floorplan and its applications," *Proc. 36th ACM/IEEE Design Aut. Conf.*, pp. 268-273, 1999.

[5] X. Hong *et al.*, "Corner block list: an effective and efficient topological representation of non-slicing floorplan," *Proc. IEEE Int. Conf. on Comp.-Aided Design*, pp. 8-12, 2000.

[6] D.E. Knuth, *The Art of Computer Programming*, Vol. 1 (3rd edition), Addison-Wesley Publ., 1997.

[7] K. Lampaert, G. Gielen, W. Sansen, "A performance-driven placement tool for analog integrated circuits," *IEEE J. of Solid-State Circ.*, Vol. SC-30, No. 7, pp. 773-780, July 1995.

[8] E. Malavasi, E. Charbon, E. Felt, A. Sangiovanni-Vincentelli, "Automation of IC layout with analog constraints," *IEEE Trans. CAD*, Vol. 15, No. 8, pp. 923-942, Aug. 1996.

[9] H. Murata *et al.*, "VLSI module placement based on rectangle-packing by the sequence-pair," *IEEE Trans. CAD*, Vol. 15, No. 12, pp. 1518-1524, Dec. 1996.

[10] S. Nakatake, K. Fujiyoshi, H. Murata, Y. Kajitani, "Module packing based on the BSG-structure and IC layout applications," *IEEE Trans. CAD*, Vol. 17, pp. 519-530, June 1998.

[11] Y.-X. Pang *et al.*, "Block placement with symmetry constraints based on the O-tree non-slicing representation," *Proc. 37th ACM/IEEE Des. Aut. Conf.*, pp. 464-467, 2000.

[12] F.P. Preparata, M.I. Shamos, *Computational Geometry*, Springer-Verlag, 1985.

[13] X. Tang, D.F. Wong, "FAST-SP: A fast algorithm for block placement based on sequence pair," *Proc. Asia-S. Pacific Design Aut. Conf.*, pp. 521-526, Yokohama, Japan, 2001.