Using Red-Black Interval Trees in Device-Level Analog Placement with Symmetry Constraints*

Florin Balasa Sarat C. Maruvada Karthik Krishnamoorthy

Dept. of Computer Science, University of Illinois, Chicago, IL 60607

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 [3, 10]. This paper presents a novel exploration technique for analog placement, operating on the set of tree representations of the layout [6, 2], where the typical presence of an arbitrary number of symmetry groups of devices is directly taken into account during the search of the solution space. The efficiency of the novel approach is due to the use of *red-black interval trees* [4], data structures employed to support operations on dynamic sets of intervals.

1 Introduction

Topological representations of block placements are encoding systems of feasible placement configurations, encrypting the positioning (topologic) relations between blocks. They gained attention as an alternative approach to the absolute representation (see, e.g., [3]) since they ensure the exploration of a solution space restricted to only feasible placements. While placement techniques based on 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 realizable, the techniques adopting topological representations trade-off a smaller number of moves for more complex-to-build, feasible layouts.

The first remarkable nonslicing encoding system named *sequence-pair* was introduced by Murata *et al.* [9]. An $O(n^2)$ algorithm building the placement was used for the evaluation of the codes consisting of pairs of sequences of block permutations. More recently, a different approach – based on computing the longest common subsequence in a pair of weighted sequences – was proposed by Tang *et al.* [13]. This algorithm achieves $O(n \log\log n)$ running time using an efficient implementation of priority queues. Nakatake *et al.* devised a meta-grid structure called *bounded-sliceline grid* to define "left-right" and "above-below" positioning relations between blocks [11], but the redundancy of this representation is very high. The corner block list, proposed more recently [7], is a representation that can be used to encode floorplans with zero dead-space ("mosaic" floorplans).

Different from the previous encoding systems, the tree -based representations define the topologic relations between blocks dependent on their dimensions. Guo *et al.* proposed the *ordered tree* (*O-tree*) representation – a tree with $n+1$ nodes, encoded by $(\mathcal{T},\pi)$, where $\mathcal{T}$ is a $2n$-bit string identifying the branching structure of the tree relative to a traversal order and π is a permutation of the blocks [6]. Chang *et al.* [2] suggested a representation based on binary trees, this encoding being based on the one-to-one mapping between forests of rooted trees and binary trees [4].

2 Analog placement and topological representations

The traditional way of approaching device-level placement problems for analog layout is to explore a huge search space, where the cells are represented by means of absolute coordinates, using typically the simulated annealing combinatorial optimization algorithm. This exploration, which allows cell overlaps during their moves (translations, changes of orientation) in the chip plane, is used by placement tools in several software systems for analog layout [3, 10]. Since this strategy often exhibits a slow convergence, the alternative approach is to use nonslicing topological representations in order to explore only feasible placement solutions. However, selecting an appropriate encoding is not straightforward.

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. The main reason of symmetric placement (and routing) is to match the layout-induced parasitics in the two halves of a group of devices. 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 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.

Due to symmetry constraints, most of the codes of any topological representation would be infeasible in symmetry point of view. Only a tiny fraction (typically under 1%) of the solution space of a topological representation contains symmetric-feasible codes relative to a given set of constraints [1]. A topological representation is a good candidate for the exploration of the solution space in analog placement only if it possesses a well-defined 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 those "symmetric-feasible" codes. Appropriate properties could be formulated for sequence-pairs [1] and tree representations [12, 1] (see Section 3), but whether the corner block list [7] possesses one is still a research topic.

This paper presents a novel exploration technique for the solution space of analog placement problems operating on a subset of the binary tree representations [2], where the typical pres-

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

ence of an arbitrary number of symmetry groups of devices is directly taken into account. Note that the paper focuses on the efficiency of the exploration of the solution space; other important aspects specific to analog placement like, for instance, eliminating systematically-induced mismatches for matching devices identically specified, arranging devices such that critical structures are shared in common, handling thermal constraints, dealing with noise coupling, are beyond the scope of the paper, although the implementation (Section 4) takes into account part of these aspects. The efficiency of the novel approach is due to the use of *red-black interval trees* [4], data structures mainly employed to support operations on dynamic sets of intervals.

This paper is organized as follows. Section 3 will present the novel $O(n \log n)$ evaluation algorithm which complexity is better than the one of any other existent topological algorithm *supporting symmetry constraints*.[1] An illustrative example will show the algorithm flow. Finally, Section 4 will present an overview of the experimental results and Section 5 will summarize the conclusions of this research.

3 Analog placement with symmetry using a red-black interval tree

The algorithm described below is executed in each inner-loop iteration of the simulated annealing, evaluating the layout code after each move. It assumes that the placement encoding is a binary tree,[2] like the one in Fig. 3(a), whose nodes correspond to the devices. The binary tree representation imposes the following vertical and horizontal positioning constraints: (a) each device in the left subtree is above its parent device; (b) if the *y*- projections of two devices are overlapping, the device of the node visited first in a preorder traversal of the tree (visit any node before its left and right subtrees) is to the left of the device whose node is visited the second. In addition, since not all the binary trees can lead to a layout satisfying a given set of symmetry constraints, we shall consider only the trees having the following "symmetric-feasibility" property: for any devices A and B in a symmetry group, node A *precedes* B in the inorder traversal (visit any node in between its left and right subtrees) of the binary tree if and only if node $sym(A)$ – corresponding to A's symmetric pair – *succeeds* node $sym(B)$ in the preorder traversal. The tree in Fig. 3(a) satisfies this property for the group of symmetric pairs (F,G) and (C,J). It was proven [1] that this property is sufficient to ensure the correctness of the layout in symmetry point of view. Moreover, it ensures the correctness of the algorithm presented in this paper. The absence of the proof here does not impede understanding the algorithm flow.

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.

Algorithm: Computation of the y- coordinates of the devices

for each node B_i (visited in a preorder traversal)
{ **if** node B_k is the closest ancestor of B_i
  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 devices (B_j, B_i) are symmetric
    if node B_j has been visited and $y_j < y_i$
    then repeat this preorder traversal once again;
    $y_j = y_i$;
}

The algorithm computes the coordinates y_i since in the preorder traversal the nodes corresponding to the devices *below* are visited before the nodes of the devices *above* due to the vertical positioning constraints (a) (see the example in Section 3.2). In the absence of symmetry constraints one traversal of the tree is sufficient; however, a second traversal may be necessary in order to satisfy the *y*- symmetry constraints of the type: $y_i = y_j$ for two symmetric devices (B_i, B_j). As the traversal of the binary tree is performed in linear time, the computation of y_i's is done in $O(n)$ time.

3.1 Using an interval tree for the evaluation of the binary tree representation

After the computation of the *y*- coordinates, the abscissae of the devices are determined inserting them in the layout while traversing the binary tree representation and updating the contour of the left or right border of the partial placement configuration. The subtle part of the algorithm is the use of a *red-black interval tree*, data structure efficiently supporting operations on dynamic sets of intervals.

The interval tree is a binary search tree, each node having associated a closed interval whose interior is disjoint from the intervals of the other nodes, but whose union is a closed interval as well. In our case, the union of the node intervals will always be $[0, H]$, where H is the chip height. In addition, the intervals of the nodes in any left subtree are to the left (on the real line) of the node interval, while the intervals in the right subtree are to the right of the node interval. Moreover, the interval tree is organized as a *red-black tree* [4] – a binary search tree with an extra bit of storage per node: its *color*, which can be either *red* or *black*.[3] In addition, if a node is red its children must be black (the *Nil* pointers are also considered black leafs), and every path from a node to a descendant leaf contains the same number of black nodes. An example of a red-black interval tree as described above is displayed in Fig. 1. By constraining the way nodes can be colored on any path from the root to a leaf, red-black trees ensure that no such path is more than twice as long as any other [4], so the tree is *approximately* balanced.

The general scheme of the algorithm computing the device abscissae is given below. First, the root of the red-black interval tree is created, the node having attached the interval $[0, H]$. Afterwards, the binary tree topological representation is walked in preorder, such that the blocks to the *left* are visited before the ones to the *right* (see the horizontal positioning constraints). The red-black interval tree is iteratively updated as a result of the new *y*- spanning interval $[y_i, y_i + h_i]$ of device B_i, which is inserted in the tree.

In the absence of symmetry constraints, one preorder traversal of the binary tree encoding would suffice. In most of the cases, one single traversal cannot yield a feasible placement in symmetry point of view. The fact that the binary tree code possesses the

[1] All the algorithms in the literature using topological representations and supporting symmetry [12, 1] have quadratic complexity.

[2] Due to the correspondence between forests of trees and binary trees [4], it can operate with minor modifications on O-trees [6], too.

[3] This color convention was introduced by Guibas and Sedgewick [5] who studied the properties of red-black trees at length.

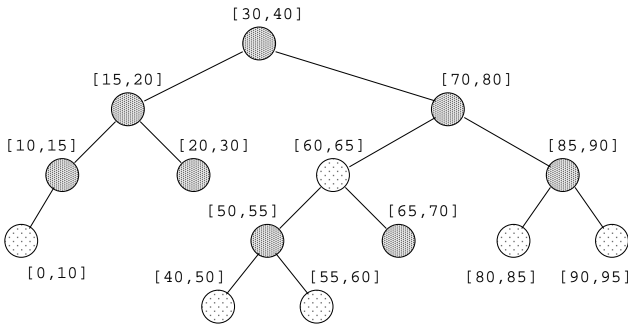

Figure 1: Example of red-black interval tree
(the *red* nodes are shaded, the *black* nodes are darkened)

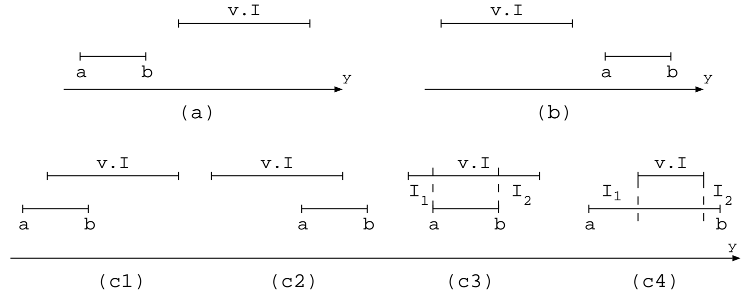

Figure 2: The interval trichotomy for the two closed intervals $v.I$ and $[a,b]$

property of "symmetric-feasibility" [1] is crucial: as proven in [1], it ensures that one additional traversal of the tree encoding in *inverse* preorder is sufficient to fix all the x- symmetry constraints of the form $(x_i + w_i) + x_j = 2 \cdot x_{symAxis}$, where (B_i, B_j) is a pair of symmetric devices. For the sake of readability, the algorithm was simplified assuming, for instance, that all the devices subject to symmetry constraints belong to a single symmetry group. The implementation is more refined though, able to deal with an arbitrary number of symmetry groups; self-symmetric devices are handled as well. Note that most of the circuits in Table 1 contain several symmetry groups.

Algorithm: Computation of the x- coordinates of the devices

let H be the total height of the chip;
RedBlackTreeNode $v_0 = InsertNode([0,H], 0, black)$;
initialize $x_i = 0$ and $x_{symAxis} = 0$;
preorder = True;
for each node/device B_i (visited in a preorder traversal)
  $UpdateRedBlackTree\ (v_0, [y_i, y_i + h_i])$;
compute $W = \max\{v.x\}$; delete all the nodes v;
RedBlackTreeNode $v_0 = InsertNode([0,H], W, black)$;
preorder = False;
for each node/device B_i (visited in *inverse* preorder)
  $UpdateRedBlackTree\ (v_0, [y_i, y_i + h_i])$;

The procedures *InsertNode* and *DeleteNode* insert/ delete a node from the red-black interval tree, preserving the properties of this tree, which were stated at the beginning of this section. The insertion and deletion techniques take $O(\log n)$ time each and are fully discussed in [4]. In addition, the *InsertNode* procedure contains the constructor of a "red-black" node v having attached an interval denoted $v.I$ (its low and high extremes being denoted $\min\{v.I\}$ and $\max\{v.I\}$), an abscissa $v.x$ for the computation of the x_i values, and the node color (*red* or *black*). In the preorder traversal, $v.x$ represent abscissae of vertical segments on the right border of the chip; in the inverse traversal, they are abscissae of the left border. Hence, $v.x$ and x_i are computed using (C-style) conditional assignments.

The procedure *UpdateRedBlackTree* follows the cases of the interval trichotomy of the two intervals $v.I$ and $[a,b]$, that is, the three cases: (a) $\max[a,b] \le \min\{v.I\}$;
(b) $\max\{v.I\} \le \min[a,b]$; (c) $v.I$ and $[a,b]$ overlap;
in this last case, there are four situations (see Fig. 2).

procedure $UpdateRedBlackTree\ (v, [a,b])$
 if $b \le \min\{v.I\}$ // case (a) (Fig. 2)
  then $UpdateRedBlackTree\ (v.left, [a,b])$;
 ef $\max\{v.I\} \le a$ // case (b) (**ef** means **else if**)
  then $UpdateRedBlackTree\ (v.right, [a,b])$;
 else // case (c): 4 situations
  $x_i =$ preorder ? $\max\{x_i, v.x\}$: $\min\{x_i, v.x - w_i\}$;
  if $a < \min\{v.I\}$ && $b < \max\{v.I\}$
   then $v.I = [b, \max\{v.I\}]$; // case (c1)
     $UpdateRedBlackTree\ (v.left, [a,b])$;
   ef $\min\{v.I\} < a$ && $\max\{v.I\} < b$
    then $v.I = [\min\{v.I\}, a]$; // case (c2)
     $UpdateRedBlackTree\ (v.right, [a,b])$;
   ef $\min\{v.I\} \le a$ && $b \le \max\{v.I\}$
    then // case (c3): $[a,b] \subseteq v.I$
     if $(I_1 = [\min\{v.I\}, a]) \neq \emptyset$
      then $InsertNode(I_1, v.x, red)$;
     if $(I_2 = [b, \max\{v.I\}]) \neq \emptyset$
      then $InsertNode(I_2, v.x, red)$;
    else // case (c4): $v.I \subset [a,b]$
     if $(I_1 = [a, \min\{v.I\}]) \neq \emptyset$
      then $DeleteInterval(v.left, I_1)$;
     if $(I_2 = [\max\{v.I\}, b]) \neq \emptyset$
      then $DeleteInterval(v.right, I_2)$;
   if devices (B_i, B_j) are symmetric
    if node B_j has already been visited
     then $t = 2x_{symAxis} - (x_j + w_j)$;
       $x_i =$ preorder ? $\max\{x_i\ ,\ t\}$: t;
     else if (*preorder*) **then**
       $x_{symAxis} = \max\{x_{symAxis}\ ,\ x_i + w_i\}$;
   $v.I = [a,b]$; $v.x =$ preorder ? $x_i + w_i$: x_i;
   $MergeAdjacentIntervalsWithSameAbscissae(v)$;
end_procedure

In the cases (a) and (b), the procedure *UpdateRedBlackTree* is recursively called for the left and, respectively, right subtree. The cases (c1) and (c2) are similarly handled; the only difference is that the interval $v.I$ is shortened by eliminating the overlap with $[a,b]$ since the intervals in the tree must be disjoint. The number of nodes in the interval tree can increase only in the situation (c3) due to the fragmentation of the interval $v.I$ in at most three segments. On the other hand, the number of nodes in the interval tree can decrease only in the case (c4), when all the nodes (but one) having intervals completely overlapped by $[a,b]$ will be recursively deleted by the procedure *DeleteInterval*.

The procedure *DeleteInterval* eliminates (using *DeleteNode*) the nodes with intervals entirely overlapped by $[a,b]$. Its pseudocode had to be skipped due to lack of space, but its behavior should be clear from the illustrative example. The procedure

MergeAdjacentIntervalsWithSameAbscissae identifies the nodes v_1, v_2 having the intervals adjacent to $v.I$. For instance, if v is the root in Fig. 1 ($v.I = [30,40]$), v_1 is the node whose interval is [20,30] and v_2 is the node whose interval is [40,50] . If, e.g., $v_1.x = v.x$, then the root would get its interval modified to [20,40], while v_1 would be removed since it is not necessary any longer, the two segments of the contour being collinear. Finding the successor and predecessor nodes in a binary search tree is easy [4].

3.2 Illustrative example

Consider the binary tree in Fig. 3(a) representing a layout with ten rectangular blocks having the widths and heights indicated: A(14x3), B(3x1), C(4x2), D(5x2), E(4x3), F(2x6), G(2x6), H(2x3), I(5x6), J(4x2). In the preorder traversal the nodes of this binary tree are visited in the alphabetical order A,B,...,J.

The computation of the y- coordinates yields successively $y_A = 0$, $y_B = y_A + h_A = 3$, $y_C = y_J = y_B + h_B = 4$, $y_D = y_C + h_C = 6$, $y_E = y_D + h_D = 8$, $y_F = y_G = y_H = y_A + h_A = 3$, $y_I = y_H + h_H = 6$. Note that the computation of y's could be done in this case while walking only once the binary tree: the devices G and J – from the symmetric pairs (F,G) and (C,J) – were not pushed upper than F and, respectively, C, which had been visited first.

The first root node v_0 of the red-black interval tree (see Fig. 4, the first tree) has associated the interval $[0,H] = [0,12]$ since the height of the chip is $H = \max\{y_i + h_i\} = 12$. The y- spanning interval $[y_A, y_A + h_A] = [0,3]$ of the first node visited in the preorder traversal of the binary tree in Fig. 3(a) is the argument of *UpdateRedBlackTree* in the first iteration. Since in the interval trichotomy the case is as in Fig. 2(c3), and $I_2 = [3,12] \neq \emptyset$, the root will get a new right child having attached the interval I_2. The abscissa of block A is $x_A = \max\{x_A, v_0.x\} = 0$. The root interval is modified to $v_0.I = [0,3]$ and the abscissa of the root becomes $v_0.x = x_A + w_A = 14$ (see Fig. 4, the second tree).

The processing of block B will insert a new node as a red right child in the tree interval (case (b), then case (c3) – Fig. 2 – in the recursive call). Since the red node $[3,4]$ has a red child $[4,12]$ (red-black property violation!), a left rotation as well as a modification of the node coloring are performed in order to restore the red-black property (see Fig. 4, the third tree).

The successive modifications of the red-black interval tree are displayed in Fig. 4. After each iteration, the inorder walk of the red-black interval tree describes exactly the contour of the *right* border of the chip. Note that in the "block F"-iteration the case is as in Fig. 2(c4), since $[y_F, y_F + h_F] = [3,9]$ covers the intervals $[3,4]$, $[4,6]$, and $[6,8]$ in the red-black tree. The interval of the first node is modified and the other two corresponding nodes are removed by the procedure *DeleteInterval*. The last tree in Fig. 4 corresponds to the placement in Fig. 3(b). The current width of the layout is $W = \max\{v.x\} = 15$.

Note that the x- symmetry constraint for the pair of cells (C,J) is still not satisfied: indeed, $((x_C + w_C) + x_J = 15) > (2x_{symAxis} = 14)$. In the first traversal, the position of the symmetry axis is determined and the rightmost devices in the symmetry pairs are positioned attempting to meet the $x-$ symmetry constraints $(x_i + w_i) + x_j = 2x_{symAxis}$. However, due to the topologic constraints, some of these devices may be pushed further to the right. A second traversal of the binary tree encoding (Fig. 3(a)) in the inverse order (that is, J,I,...,B,A) will yield the final placement in Fig. 3(c). The red-black interval tree restarts with only one node v_0 having $v_0.I = [0,12]$ and the abscissa $v_0.x = W = 15$. Afterwards, the red-black tree is evolving such that after each iteration it describes the *left* border of the chip.

3.3 Complexity analysis

The red-black tree can have at most n nodes since there are at most n segments on the border contours determined by the y-spanning intervals $[y_i, y_i + h_i]$, hence the red-black tree has a maximum height of $\lceil 2\log_2(n) \rceil$ [4]. Since the node insertions and deletions take $O(\log n)$, we may be tempted to consider $O(\log n)$ the worst-case time bound per iteration. However, this is not true due to the case (c4): when the tree decreases in size, up to $O(n)$ nodes can be deleted.[4]

However, using the *aggregate method* of *amortized analysis* [4], it can be shown that the amortized time bound is $O(\log n)$ per iteration. In an amortized analysis, the time required to perform a sequence of operations is averaged over all the operations performed. Intuitively, the reason is that each node can be deleted at most once for each time it is created. Since there are n iterations per binary tree traversal, the overall time complexity is $O(n \log n)$.

Note that a balanced search (AVL) tree [8] could be also used to achieve the same time complexity. However, balance is maintained in AVL trees by as many as $\Theta(\log n)$ rotations after a node deletion,[5] whereas at most two rotations are necessary to maintain the red-black tree after an insertion, and at most three rotations after a deletion [4]. Such an *approximately* balanced tree structure is sufficient to achieve a complexity of $O(n \log n)$, at the same time being more efficient in terms of practical computation effort.

4 Experimental results

A placement tool for analog layout using selectable exploration algorithms has been implemented in C++ on a SUN Blade 100 workstation. The tool can operate both with different topological representations and different code evaluation algorithms. In addition, a complementary placement algorithm based on the traditional absolute representation has been embedded in the tool as well. The tool uses a simulated annealing optimizer with a complex cost function comprising, along with the chip area and estimated wire length, different penalty terms like, e.g., modeling device separation constraints. Besides symmetry constraints, the tool handles systematically-induced device mismatches, alignment constraints, and performs shape optimizations.

Table 1 displays (part of) the results of our experiments. The performance of the algorithm described in this paper has been evaluated in comparison with the algorithm based on O-trees from [12], and a simpler algorithm of quadratic complexity using binary trees [1]. Additional tests with a traditional algorithm

[4] Such a situation could occur if the red-black tree had $O(n)$ nodes and in the next iteration the block had the height of the whole chip; the red-black tree would be reduced to a single node with $v.I = [0,H]$.

[5] The deletion of the rightmost node in a Fibonacci tree [8] is such an example. Rebalancing after insertion never needs more than a single or a double rotation though.

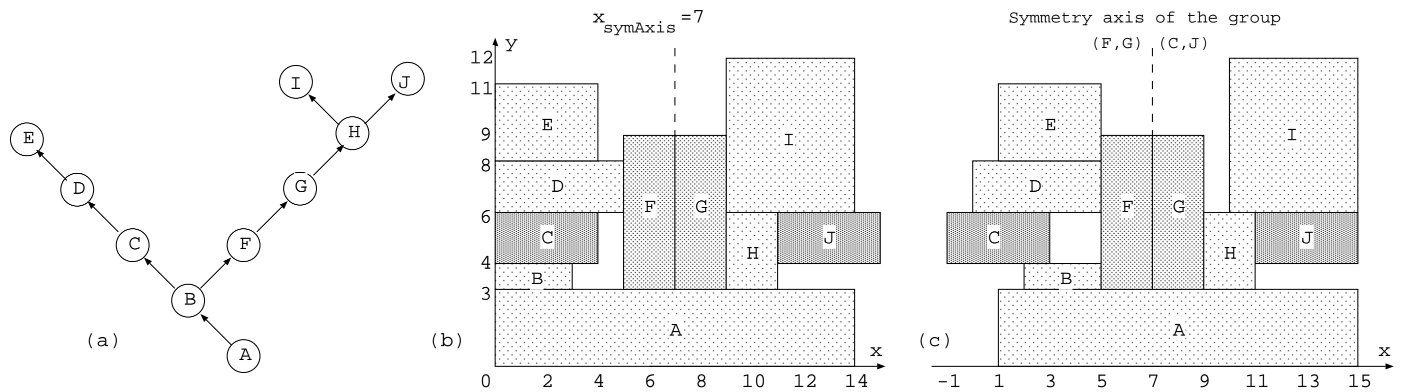

Figure 3: Illustrative example: (a) binary tree representation of a layout with a group of two pairs of symmetric devices (F,G) and (C,J); (b) device placement after the preorder traversal, and (c) the final device placement

[0,12] x=0

[0,3] x=14 [3,12] x=0
A: $[y_A, y_A+h_A]$=[0,3]

[3,4] x=3 [0,3] x=14 [4,12] x=0
B: [3,4]

[3,4] x=3 [0,3] x=14 [4,6] x=4 [6,12] x=0
C: [4,6]

[3,4] x=3 [0,3] x=14 [6,8] x=5 [4,6] x=4 [8,12] x=0
D: [6,8]

[3,4] x=3 [0,3] x=14 [6,8] x=5 [4,6] x=4 [8,11] x=4 [11,12] x=0
E: [8,11]

[3,9] x=7 [0,3] x=14 [9,11] x=4 [11,12] x=0
F: [3,9]

[3,9] x=9 [0,3] x=14 [9,11] x=4 [11,12] x=0
G: [3,9]

[3,6] x=11 [0,3] x=14 [9,11] x=4 [6,9] x=9 [11,12] x=0
H: [3,6]

[3,9] x=11 [0,3] x=14 [9,12] x=14
I: [6,12]

[4,6] x=15 [3,4] x=11 [6,12] x=14 [0,3] x=14
J: [4,6]

Figure 4: The red-black interval tree after each iteration in the first traversal. Each node *v* has attached an interval *v.I* and an abscissa *v.x*. The last red-black tree corresponds to the layout in Fig. 3(b).

Design	Nr. cell	Symmmetry groups	O-trees [12] Time/Area	B*-trees [1] Time/Area	Crt. alg. Time/Area
dffrsdch	37	6/4	2.6 / 6.3	1.5 / 6.2	1.6 / 6.3
lpf2_b25b	52	23	4.4 / 36.2	3.2 / 36.9	2.4 / 36.4
dcservo_cmfb	66	5/4	11.8 / 60.5	8.6 / 60.5	6.0 / 60.4
modbias_2p4g	87	16/12/6/6/6	32.7 / 59.0	17.8 / 56.9	10.0 / 55.1
div_by_2or4	116	6/4/8/12/15	47.0 / 58.5	29.1 / 54.6	14.7 / 54.0

Table 1: Placement with symmetry constraints (Time [min], Area [$10^3 \times \mu m^2$]). Column 3: devices in each sym. group.

based on the absolute representation, as well as an algorithm using sequence-pairs [1] have been performed as well.[6] Actually, all the known exploration techniques supporting symmetry constraints have been evaluated. The test benchmarks are analog blocks containing symmetry groups of devices, components of a spread spectrum transceiver used in wireless modems. Figure 5 displays the placement solution for one of these examples.

The experiments show that our current technique is better in terms of computational effort than the algorithm [12]. The advantage of the novel approach is twofold: the search space has a smaller size (since it explores only a *subset* of binary tree representations), and the evaluation algorithm using a red-black interval tree is more efficient. The algorithm described in this paper usually outperforms the simpler algorithm [1] for more complex designs, since when the input size gets larger, it begins to exploit its $O(n \log n)$ running time characteristic. However, for smaller examples (up to 30-40 blocks), the current algorithm can be outperformed by the simpler algorithm [1] due to the overhead of maintaining the red-black interval trees. In addition, the results when using the absolute representation are much worse, both in terms of time and quality.

The symmetry constraints affect significantly the running times of all the algorithms based on topological representations. Besides more complex evaluation algorithms, the *moves* within the simulated annealing optimizer are more expensive since they often involve whole symmetry groups.

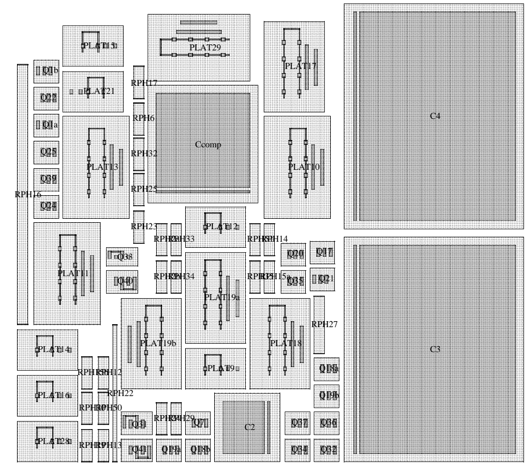

Figure 5: Placement of the analog block *dcservo_cmfb*

5 Conclusions

This paper has presented a novel analog placement technique operating on a subset of binary tree topological 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 novel evaluation approach, executed after each modification of the binary tree representation, employs a contour modification algorithm. Its efficiency is due to the use of a red-black interval tree, data structure supporting operations on dynamic sets of intervals.

References

[1] F. Balasa, C.S. Maruvada, "Using non-slicing topological representations for analog placement," *IEICE Trans. on Fundamentals of Electr., Comm. & Comp. Sc.*, Vol. E84-A, No. 11, pp. 2785-2792, Nov. 2001.

[2] 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 Design Automation Conf.*, pp. 458-463, June 2000.

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

[4] T.H. Cormen, C.E. Leiserson, R.L. Rivest, *Introduction to Algorithms*, McGraw-Hill, 1990.

[5] L.J. Guibas, R. Sedgewick, "A diochromatic framework for balanced trees," *Proc. 19th Annual Symposium on Foundations of Computer Science*, pp. 8-21, 1978.

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

[7] 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, Nov. 2000.

[8] D.E. Knuth, *The Art of Computer Programming*, Vol. 3, Addison-Wesley, 1973.

[9] H. Murata, K. Fujiyoshi, S. Nakatake, Y. Kajitani, "VLSI module placement based on rectangle-packing by the sequence-pair," *IEEE Trans. CAD of IC's and Systems*, Vol. 15, No. 12, pp. 1518-1524, Dec. 1996.

[10] E. Malavasi, E. Charbon, E. Felt, A. Sangiovanni-Vincentelli, "Automation of IC layout with analog constraints," *IEEE Trans. on Comp.-Aided Design of IC's and Systems*, Vol. 15, No. 8, pp. 923-942, Aug. 1996.

[11] S. Nakatake *et al.*, "Module packing based on the BSG-structure and IC layout applications," *IEEE Trans. on CAD of IC's and Syst.*, Vol. 17, pp. 519-530, June 1998.

[12] Y.-X. Pang, F. Balasa, K. Lampaert, C.-K. Cheng, "Block placement with symmetry constraints based on the O-tree non-slicing representation," *Proc. 37th ACM/IEEE Des. Aut. Conf.*, pp. 464-467, June 2000.

[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, Jan. 2001.

[6] Due to lack of space, these latter results could not be displayed.