SUMMARY Layout design for analog circuits has historically been a time consuming, error-prone, manual task. Its complexity results not so much from the number of devices, as from the complex interactions among devices or with the operating environment, and also from continuous-valued performance specifications. This paper addresses the problem of device-level placement for analog layout in a non-traditional way. Different from the classic approaches—exploring a huge search space with a combinatorial optimization technique, where the cells are represented by means of absolute coordinates, being allowed to illegally overlap during their moves in the chip plane—this paper advocates the use of non-slicing topological representations, like (symmetric-feasible) sequence-pairs, ordered- and binary- trees. Extensive tests, processing industrial analog designs, have shown that using skillfully the symmetry constraints (very typical to analog circuits) to remodel the solution space of the encoding systems, the topological representation techniques can achieve a better computation speed than the traditional approaches, while obtaining a similar high quality of the designs.

key words: topological representations, analog placement, sequence-pairs, ordered trees, binary trees

1. Introduction

In recent years, complete systems that before occupied separate chips are being integrated on a single chip. Examples of such systems-on-a-chip (SoC’s) include telecommunications integrated circuits (IC’s), wireless designs—as components in RF receivers and transmitters, and networking interfaces. Although most functions in such integrated systems are implemented with digital circuitry, the analog circuits needed at the interface between the electronic system and the real world are now being integrated on the same die for reasons of cost and performance.

Layout design for analog circuits has historically been a time consuming, error-prone, manual task. If layout for digital IC’s is usually regarded as a difficult task mainly because of the scale of the problem (but also complex delay modeling, timing optimization), analog circuits and the analog portions of mixed-signal SoC’s are significantly smaller—usually up to 100 devices in a cell, and less than 20,000 devices in a complete subsystem [14]. The task complexity for analog layout results not so much from the sheer number of devices, as from the complex interactions among devices or with the operating environment, and also continuous-valued performance specifications. As most analog circuits are very distinct among themselves, today they are usually still designed and laid out by hand.

This paper addresses an essential problem of analog layout design—the device-level analog placement—in a non-traditional way, using recent results on topological representations for non-slicing floorplans. The traditional way of approaching the analog placement problem is to explore a huge search space with a combinatorial optimization method (for instance, simulated annealing) where the cells are represented by means of absolute coordinates relative to an arbitrary system of axes, and also they can illegally overlap during their moves in the chip plane. However, the topological representations—the main focus of the research plan—are based on a different idea: to define an encoding system as a solution space, each code representing a feasible placement configuration.

While the classic absolute representation approach trades off a larger number of (annealing) moves for easier and quicker-to-build layout configurations which may not be always physically realizable, the idea of using topological representations is to trade off more complex (but physically correct!) layout constructions for a smaller number of moves. Our research results show that the skillful use of symmetry constraints (very typical to analog circuits) can remodel the encoding systems, diminishing significantly the solution space size and, therefore, making its exploration more effective in comparison with the classic approaches.

The paper is organized as follows: Section 2 discusses the slicing and non-slicing topological representations of placement configurations; then, Sect. 3 presents the device-level analog placement problem and discusses several techniques to solve it based on different topological representations; finally, Sect. 4 gives an overview of the experimental results, and Sect. 5 presents the basic conclusions of this research.

2. Topological Representations

The topological representations of floorplans got recently a renewed attention in the context of block place-
ment. A brief historical overview will introduce many concepts used along this paper.

The block placement problem subject to nonoverlapping constraints, often called packing, was proven to be NP-hard even for rectangular blocks of fixed, rectangular shape [11]. The nature of the problem entailed the use of combinatorial optimization methods (as simulated annealing, genetic algorithms) to explore the solution space of placement configurations.

A combinatorial optimization method can equally operate with two classes of representations of block configurations. The most straightforward and widely used is the absolute (or flat) representation, introduced by Jepsen and Gellat [6], where the positions of the blocks are directly specified in terms of coordinates relative to an arbitrary system of axes in the chip plane [3]. However, the convergence of the exploration may be slow due to the huge size of the search space (which contains also infeasible placement configurations, since blocks are allowed to illegally overlap).

An alternative approach to the absolute representation is to define a set of codes—each code representing a placement configuration—as a solution space. An encoding system of feasible placement configurations is usually referred to as a topological representation, being an encryption of the positioning (topological) relations between any pair of modules. The first topological representation was derived from the slicing floorplan model where the blocks are organized in a set of slices which recursively bisect the layout horizontally and vertically. The direction and nesting of the slices is recorded in a (binary) slicing tree or, equivalently, in a normalized Polish expression (see, e.g., [3]).

The slicing representation limits the set of layout topologies. This can degrade layout density, especially when cells are very different in size, which is often the case in analog layout. Consequently, it was widely acknowledged that slicing placement is not a good choice for high-performance analog design. This is the reason why topological representations were in general disregarded, and the most effective analog placement tools existent so far employed simulated annealing—as optimization engine—operating with absolute placement representations [3], [9], [10].

More recently, several novel topological representations, not restricted to slicing floorplan topologies, have been proposed. Murata et al. suggested to encode the “left-right” and “up-down” positioning relations between blocks using two sequences of block permutations, named sequence-pair [11]. Nakatake et al. devised a meta-grid structure (called bounded-sliceline grid) without physical dimensions to define orthogonal relations between modules [12]. Guo et al. proposed the O-tree data structure to reduce the drawback effect of redundancies in the two previous representations [4]. The corner block list, proposed very recently [5], is a representation that can be used to encode floorplans with zero dead-space (“mosaic” floorplans). Different from the tree-based representations, the sequence-pair, the bounded-sliceline grid, and the corner block list define the topological relations between blocks independent of their dimensions.

The advent of non-slicing topological representations is likely to have an important impact in device-level placement for analog layout, as the superiority claim—undisputed until recently—of the absolute representation for analog layout becomes questionable.

Although Kahng expressed some concerns regarding topological representations in general—as scalability, or “the packing obsession” [7], these concerns are hardly justified in our context. Scalability to instances of hundreds of thousands of cells is a problem indeed, but only for digital circuits. Our results are very effective (see Sect. 4), even when processing real-life analog placement problems, where part of the blocks are “soft” capacitors or having a number of alternative realizations. Moreover, the optimization deals with a complex cost function with multiple objectives subject to “hard” and “soft” constraints.

The problem of analog placement is a perfect opportunity for non-slicing topological representations to prove their practical value.

3. Device-Level Analog Placement

In order to automatically produce analog device-level layouts matching in density and performance the high-quality manual layouts, an analog placement tool must be provided with the capability of dealing with analog-specific features:

1) The ability to deal with topological constraints for symmetry and device matching.

In high-performance analog circuits it is often required that groups of devices are placed symmetrically with respect to one or several axes. Analog circuits use very often differential architectures based on electrically symmetric networks, and therefore, layout symmetry matches the induced parasitics in the two halves of a group of devices. Symmetry is also used to prevent unwanted oscillations by balancing thermal couplings in differential structures, or to reduce the thermal sensitivity of the circuit. In order to reduce systematically-induced mismatches—produced by dissimilar geometrical choices, matching groups of devices should be constrained to the same orientation and variant.

2) The ability to arrange devices such that critical structures are shared in common (device merging or geometry sharing) in order to reduce both layout density and induced parasitics.

3) The existence of a library of device generators and the ability to exploit their reshaping capability.

The device-matching constraints and the geometry sharing situations can be handled relatively easy when using topological encodings 1) by imposing constraints
at the move-set level of the combinatorial optimization
algorithm employed to explore the set of topological
representations, 2) by introducing penalty terms in the
cost function, and 3) by modifying the procedure which
builds the placement configurations from the topologi-
cal encodings.

However, dealing efficiently with symmetry con-
straints in the framework of topological representations
has proven to be a problem of unexpected difficulty. In
our first empirical tests using the sequence-pair rep-
resentation [11], we were initially tempted to perform
minor changes to the search space exploration: if the
current encoding proved to be consistent with the sym-
metry constraints then the cost of the placement con-
figuration would be evaluated and the annealing algo-

This experiment led to the following conclusion: a
topological representation proves to be effective only if
the analog placement tool is able to explore only those
encodings which comply with the imposed set of sym-
metry constraints. This obviously better strategy en-
counters two hurdles though:

a) how to recognize the encodings complying with
the given symmetry constraints, without building the
answering layout?

b) how to efficiently restrict the exploration (per-
formed by a combinatorial optimization, like simulated
annealing) only to the subspace of these “symmetric-feasible” (S-F) encodings?

3.1 Using Symmetric-Feasible Sequence-Pairs

In the case of the sequence-pair representation [11], the
questions above have already received satisfactory an-
swers [1].

Let \((\alpha, \beta)\) be the sequence-pair of a placement
configuration containing a number of symmetry blocks
(each group composed of pairs of symmetric blocks,
and self-symmetric blocks relative to a common ver-
tical axis). Denoting by \(\alpha^{-1}_A\) the position of block \(A\) in
sequence \(\alpha\) (as \(\alpha\) can be viewed as a one-to-one mapp-
ing, \(\alpha^{-1}\) is well-defined) and defining similarly \(\beta^{-1}_A\),
and also denoting by \(\text{sym}(x)\) the block symmetric to \(x\),
the sequence-pair \((\alpha, \beta)\) is called symmetric-feasible
(S-F) if for any distinct blocks \(x, y\) in any of the symmetry
groups

\[
S: \quad \alpha^{-1}_x < \alpha^{-1}_y \iff \beta^{-1}_{\text{sym}(y)} < \beta^{-1}_{\text{sym}(x)}
\]

Choosing \(y = \text{sym}(x)\) and taking into account
that \(\text{sym}(\text{sym}(x)) = x\), condition (S) shows that any
symmetric pair of cells appears in the same order in
both sequences \(\alpha\) and \(\beta\). In addition, two cells \(x, y\)
belonging to distinct symmetric pairs and, respectively,
their symmetric cells appear in reversed order in the
two sequences of the encoding, as well as any two self-
symmetric cells in the symmetry group.

Figure 1 displays two placement configurations
xjorresponding to the sequence-pairs \((CDAFBGE, DCBGAFE)\) and \((EBAFCDG, EBCDFAG)\),
respectively. Assuming the existence of a symmetry
group composed of the pairs of symmetric cells \((C, D)\)
and \((B, G)\), and the self-symmetric cells \(A\) and \(F\), the
first encoding is infeasible in symmetry point of view:
for instance, the symmetric cells \(C\) and \(D\) do not satisfy
condition (S) as they do not appear in the same order
in the sequences \(\alpha\) and \(\beta\): \(\alpha^{-1}_C < \alpha^{-1}_D (1 < 2)\), but
\(\beta^{-1}_C > \beta^{-1}_D (2 > 1)\). On the other hand, the second
sequence-pair is symmetric-feasible in the sense of con-
dition (S), leading to a placement solution where the
symmetry group was laid out correctly (see Fig.1(b)).

An \(O(n^2)\) algorithm building a minimum area
placement, satisfying the given symmetry constraints,
from a symmetric-feasible sequence-pair was thor-
oughly described in [1]. Notice that algorithms of a bet-
ter complexity have been reported for building place-
ment configurations from sequence-pairs (for instance,
[15] having the complexity \(O(n \log n)\), but symmetry
constraints were not an issue.

The exploration of the sequence-pair encodings
using simulated annealing can be restricted to the
symmetric-feasible codes as follows: it is sufficient to start the search from an initial sequence-pair which can be easily built to verify condition (S) (for instance, corresponding to a placement where the symmetric pairs in any symmetry group top one another) and, afterwards, to constrain the move-set of the optimizer such that the property (S) of symmetric-feasibility is preserved: for instance, if two cells from symmetric pairs are interchanged in sequence \( \alpha \), then their symmetric counterparts must be also interchanged in sequence \( \beta \).

According to our tests (see Sect. 4), the strategy described above has proven highly beneficial in terms of both computation time and quality of solutions. The basic reason is that, instead of exploring a search space of both computation time and quality of solutions. The described above has proven highly beneficial in terms of both computation time and quality of solutions. The counterpart must be also interchanged in sequence \( \beta \).

\[ n^2 \]

The exploration of the O-trees is done with simulated annealing as well. After each new O-tree is generated, a verification routine—having the complexity \( O(n^2) \)—attempts to detect as early as possible whether the current encoding is symmetric-feasible or not. Two constraint graphs—vertical \( G_v \) and horizontal \( G_h \)—are built taking into account both the positioning constraints derived from the O-tree and from the given symmetry constraints. The O-tree is symmetric-feasible if \( G_h \) is acyclic and \( G_v \) does not contain positive cycles [13]. Only in this case the move is considered for acceptance, according to the probabilistic hill-climbing of the annealer. Otherwise, the move is immediately rejected as the placement built from the O-tree cannot satisfy the symmetry constraints.

3.3 Using S-F Binary Trees

Let \( T = (T_1, \ldots, T_k) \) be an O-tree, where \( T_1, \ldots, T_k \) are its subtrees relative to the root. The O-tree can be transformed recursively into a binary tree \( B(T) \) as follows:

a) if \( k = 0 \), \( B(T) \) is empty;

b) if \( k > 0 \), the root of \( B(T) \) is the root of \( T_1 \);

the left subtree of \( B(T) \) is \( B(T_{i_1}, T_{i_2}, \ldots) \), where \( T_{i_j} \) are the subtrees of \( T_1 \); the right subtree of \( B(T) \) is \( B(T_2, \ldots, T_k) \).

The binary tree derived from the O-tree in Fig. 2(a) is shown in Fig. 2(b) (the root of the O-tree becomes obsolete). As the inverse transformation is straightforward, it follows that there is a one-to-one correspondence between the sets of O-trees and of binary trees having one node less.

A binary tree which nodes represent rectangular blocks imposes the following vertical and horizontal positioning constraints:

(a) each block in the left binary subtree is above its root block;

(b) if two blocks are overlapping along their y-coordinate projection, the block visited first in a preorder traversal of the binary tree is to the left of the block visited the second.

Binary trees can lead to a better performance than both the sequence-pair and the O-tree representations: while offering in general the same solution space size as the O-trees (due to a one-to-one mapping property [8]), in the presence of symmetry constraints the exploration can be restricted to a proper subset of binary trees which were called symmetric-feasible (S-F) binary trees.

\[ \text{(i.e., } 35,280 \text{ when } p = s = 2) \]

Fig. 2 (a) O-tree representation, (b) binary tree representation of the block placement in Fig. 1(a).

†††A more general formula is proven in the Appendix of [1].

†††However, when the asymmetric part of the circuit is larger, the number of O-trees can become smaller than the number of symmetric-feasible sequence-pairs.
Let \( ((\text{leftSubtree}) \text{root} \ \text{rightSubtree}) \) and 
\((\text{root}(\text{leftSubtree} \ \text{rightSubtree}) \) be the recursive in-
order (traverse the left subtree, visit the root, traverse the 
right subtree) and, respectively, preorder representations 
of a binary tree. Let \((\gamma, \delta)\) be the pair of se-
nce derived from the inorder and preorder repre-
sentations by dropping the parentheses (used only to 
express the tree structure). Using the same notations 
as for sequence-pairs, the binary tree encoding is called 
*symmetric-feasible* (S-F) relative to a symmetry group 
\(G\) if for any distinct blocks \(x, y\) in \(G\):

\[
(S') \quad \gamma_x^{-1} < \gamma_y^{-1} \iff \delta_{\text{sym}(y)}^{-1} < \delta_{\text{sym}(x)}^{-1}
\]

Remark: Usually \((\gamma, \delta)\) is not equal to the sequence-
pair representation \((\alpha, \beta)\). There are sequences \(\alpha\) and 
\(\beta\) which do not correspond to the inorder and preorder 
traversals of any binary tree: such a sequence-pair is, 
for instance, \((CAB, ABC)\).

The S-F binary trees have a desirable property: 
their number is smaller than both the S-F sequences-
pairs and O-trees [2]. Consequently, the optimization 
time during analog placement is likely to be better.

The construction of a placement configuration 
from a symmetric-feasible binary tree encoding is 
briefly described below. For the sake of clarity, the 
pseudo-code corresponds to the case of (at most) one 
symmetry group. However, this is not a constraint of 
our method, as it will be subsequently explained.

Let \(B_1, \ldots, B_n\) be rectangular blocks to be placed 
on a chip area, each block \(B_i\) having width \(w_i\) and 
height \(h_i\), and having \((x_i, y_i)\) as coordinates of its left-
bottom corner. First, the \(y\) coordinates are computed 
with at most two traversals of the binary tree:

initialize \(y_i = 0\) and repeat traversal \(= False\);

for each block \(B_i\) (visited in a preorder traversal) 
if block \(B_k\) is the closest ancestor of \(B_i\) 
such that \(B_k\) is in the left subtree of \(B_k\) 
then \(y_i = \max\{y_i, y_k + h_k\}\);

if \(B_i\) has a symmetric \(\text{sym}(B_i)\) 
if \(\text{sym}(B_i)\) has already been visited 
and \(y_{\text{sym}(i)} < y_i\) 
then repeat traversal \(= True\);

\(y_{\text{sym}(i)} = y_i\);

if repeat traversal then repeat preorder traversal

Next, denoting \(Y_k\) the intervals \((y_k, y_k + h_k)\), the 
\(x\) coordinates of the blocks are computed as follows:

initialize \(x_i = 0\) and \(x_{\text{sym}Ax}\) = 0;

for each block \(B_i\) (visited in a preorder traversal) 
for each block \(B_k\) already visited 
if \(Y_k\) overlaps \(Y_i\) 
then \(x_i = \max\{x_i, x_k + w_k\}\);

if \(B_i\) has a symmetric \(\text{sym}(B_i)\) 
if \(\text{sym}(B_i)\) has not been visited yet 
then \(x_{\text{sym}Ax} = \max\{x_{\text{sym}Ax}, x_i + w_i\}\);

else \(\text{sym}(B_i) \rightarrow (x_{\text{sym}(i)}, y_{\text{sym}(i)})\)

\(x_i = \max\{x_i, 2x_{\text{sym}Ax} - (x_{\text{sym}(i)} + w_i)\}\); 
// only in the presence of symmetry constraints

for each block \(B_i\) (visited in inverse preorder) 
for each block \(B_k\) already visited 
if \(Y_k\) overlaps \(Y_i\) then \(x_i = \min\{x_i, x_k - w_k\}\);

if \(B_i\) has a symmetric \(\text{sym}(B_i)\) 
and \(\text{sym}(B_i)\) has already been visited 
then \(x_i = 2x_{\text{sym}Ax} - (x_{\text{sym}(i)} + w_i)\);

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 \((x_i + w_i) + x_{\text{sym}(i)} = 2x_{\text{sym}Ax}\).

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.

**Theorem:** Given a symmetric-feasible binary tree, the 
placement algorithm yields a layout satisfying the topo-
logical and symmetry constraints.

**Proof:** The blocks cannot overlap in this placement: 
if \(B_k\) precedes \(B_i\) in the preorder traversal of the tree, 
either \(y_i \geq y_k + h_k\) when \(B_i\) is in the left subtree 
of \(B_k\), or \(x_i \geq x_k + w_k\) when their \(y\)-projections overlap. 
Therefore, the positioning constraints are satisfied.

The placement along the \(x\) axis could fail only if 
there are two symmetric pairs \((A, \text{sym}(A))\) and 
\((B, \text{sym}(B))\) preventing each other from satisfying the 
\(x\)-symmetry constraints. This can happen only if the 
two pairs are interleaved and their \(y\)-projections are 
overlapping. But this situation is not possible when 
condition \((S')\) is satisfied. Indeed, assuming node \(A\) 
is visited first, there are three possibilities:

1. \(\delta_{A}^{-1} < \delta_{B}^{-1} < \delta_{\text{sym}(A)}^{-1} < \delta_{\text{sym}(B)}^{-1}\) when the two 

   pairs are interleaved. Due to the \((S')\) condition, 

   \(\gamma_{B}^{-1} < \gamma_{A}^{-1} < \gamma_{\text{sym}(B)}^{-1} < \gamma_{\text{sym}(A)}^{-1}\). 

   The inequalities imply that 

   node \(B\) is in the left subtree of \(A\), and \(\text{sym}(B)\) in the 

   left subtree of \(\text{sym}(A)\). But then, their \(y\)-projections 

   do not overlap.

2. \(\delta_{A}^{-1} < \delta_{B}^{-1} < \delta_{\text{sym}(B)}^{-1} < \delta_{\text{sym}(A)}^{-1}\). 

   Due to the \((S')\) condition, 

   \(\gamma_{B}^{-1} < \gamma_{A}^{-1} < \gamma_{\text{sym}(B)}^{-1} < \gamma_{\text{sym}(A)}^{-1}\). 

   In this situation, the \(y\)-projections of the two pairs may indeed 

   overlap. But then, the two pairs are embedded, they 

   are not interleaved.

3. \(\delta_{A}^{-1} < \delta_{\text{sym}(A)}^{-1} < \delta_{B}^{-1} < \delta_{\text{sym}(B)}^{-1}\). 

   Condition \((S')\) implies 

   \(\gamma_{B}^{-1} < \gamma_{\text{sym}(B)}^{-1} < \gamma_{\text{sym}(A)}^{-1}\). Then node \(B\) 

   is in the left subtree of both \(A\) and \(\text{sym}(A)\). It follows that 

   node \(A\) is in the left subtree of \(\text{sym}(A)\), or vice 

   versa. But this is not possible, since a block cannot be 

   above its symmetric. Therefore, the two traversals of 

   \(O(n^2)\) complexity are sufficient to perform the compu-

   tation of the \(x\) coordinates if condition \((S')\) is satisfied.

During the first traversal of the placement along 
the \(y\) axis, the blocks in the symmetry pairs are posi-
ioned attempting to meet the \(y\)-symmetry constraints
Table 1  Placement results. (Time [min], Area [10^3 \times \mu m^2])

<table>
<thead>
<tr>
<th>Design</th>
<th>Nr. cells</th>
<th>Sym. constr.</th>
<th>Absolute</th>
<th>Seq-pair</th>
<th>S-F Seq-pair</th>
<th>O-tree</th>
<th>S-F Bin-tree</th>
</tr>
</thead>
<tbody>
<tr>
<td>bias crit. generator</td>
<td>85</td>
<td>-</td>
<td>41</td>
<td>45.1</td>
<td>32</td>
<td>44.0</td>
<td>25</td>
</tr>
<tr>
<td>lqj2l_bb25b (1)</td>
<td>52</td>
<td>12</td>
<td>29</td>
<td>153.0</td>
<td>26</td>
<td>198.2</td>
<td>12</td>
</tr>
<tr>
<td>modbias_2p4g (5)</td>
<td>87</td>
<td>30</td>
<td>67</td>
<td>65.9</td>
<td>99</td>
<td>84.7</td>
<td>41</td>
</tr>
<tr>
<td>inambias_2p4g (6)</td>
<td>110</td>
<td>42</td>
<td>112</td>
<td>114.4</td>
<td>157</td>
<td>116.5</td>
<td>76</td>
</tr>
<tr>
<td>frequency divider (5)</td>
<td>116</td>
<td>46</td>
<td>92</td>
<td>59.6</td>
<td>131</td>
<td>72.4</td>
<td>68</td>
</tr>
</tbody>
</table>

Fig. 3 Placement of inambias_2p4g using S-F (a) sequence-pairs, and (b) binary trees.

The proper algorithm is more refined, able to deal with an arbitrary number of symmetry groups—that is, groups of cells having distinct symmetry axes. Then, the cells within each group must satisfy a distinct (S′) condition. Notice that the last three circuits in Table 1 contain several symmetry groups. For instance, our placement tool dealt with 6 distinct symmetry axes for the analog block in Fig. 3.

The annealing algorithm can be adapted to explore only the subspace of S-F binary trees rather than the whole space of binary trees: (1) the initial binary tree must be symmetric-feasible. If (a_i, b_i) are \( p \) pairs of symmetric blocks, such a binary tree is, e.g., \((a_1 \cdots a_p b_p \cdots b_1 \cdots)\), using the preorder traversal representation; (2) the moves during the exploration are chosen such that property (S′) continues to hold. The moves are mainly node interchanges and node cut-and-paste operations. When blocks outside the symmetry groups are involved, the property (S′) is not affected. However, if two nodes corresponding to blocks from distinct symmetric pairs are interchanged, then the nodes corresponding to their pairs must be interchanged too. Also, when a node is moved, its symmetric pair is moved too in one of the limited admissible locations, determined by analyzing the tree representations.

### 3.4 Using the Absolute Representation

A complementary placement tool using the absolute representation has been implemented as well, for the purpose of comparative evaluation. The cells are specified in terms of absolute coordinates on a gridless plane. The moves are simple coordinate shifts or changes in cell orientation. Cells are allowed to overlap in possibly illegal ways\(^\dagger\), as no restriction is made referring to

\(\dagger\)In analog layout, cells can overlap not only in legal but also beneficial ways (device merging or geometry sharing [3]).
the relative position of a cell with respect to another cell. A (weighted) penalty cost term is associated with infeasible overlaps, and this penalty is driven to zero in the optimization process. The tool employs virtual symmetry axes [10], having mobile positions, to model multiple symmetry groups.

4. Experimental Results

A placement tool for analog layout has been implemented and embedded in a retargetable object symbolic environment for layout design. The tool can use alternative optimization algorithms based both on the absolute representation (similar to other traditional approaches), and on different topological representations. In order to ensure a correct comparative evaluation, the simulated annealing schedule is identical for all the placement algorithms.

In addition to symmetry aspects, the placement tool can also eliminate systematically-induced mismatches due to dissimilar geometrical choices for matching devices. For instance, in order to reduce the degree of electrical mismatch due to area effects, matching groups of cells can be defined, the pairs of cells being constrained to the same (or mirrored) orientation and device shape. The matching groups are handled similarly to [9]—as constraints at the move-set level of the annealer. In addition, device proximity constraints are modeled with an imaginary proximity net of a relatively high weight, and a shape optimization of the capacitors is performed as well†.

Figure 3 displays two placement solutions—one obtained using S-F sequence-pairs, the other using S-F binary trees—of a 110-cell analog circuit having six symmetry groups. Figure 4 shows other two examples, the first obtained using the absolute representation, the second—based on O-trees. Note that in all these examples, the (usually large) capacitors are soft cells, which shapes are optimized and regenerated at the end of the placement.

Table 1 displays the results obtained for several analog blocks, components of a digital spread spectrum transceiver used in cordless telephone applications and wireless modems. The results have been obtained on an HP 9000/777 workstation. The conclusions derived from these experiments are the following:

1) the use symmetric-feasible sequence-pairs is highly beneficial (both in terms of CPU time and solution quality) in comparison with the “classic” sequence-pair representation where time is spent investigating codes which are infeasible in symmetry point of view;

2) the results for the “classic” sequence-pairs on tests exhibiting symmetry are much worse than those obtained using the traditional absolute representation; however, in general, the use of S-F sequence-pairs has yielded better results: therefore, topological representations should be used with care in the presence of symmetry constraints;

3) using S-F sequence-pairs is more effective in terms of speed than using O-trees in the examples exhibiting more symmetry, but less effective otherwise;

4) if the asymmetric part of circuit is predominant, the CPU times when using S-F binary- and O-trees are similar; otherwise, using S-F binary trees is the most effective in terms of speed, as the solution space is the smallest.

In general, all the algorithms produced high quality placement solutions. However, the optimization techniques operating with the absolute and tree representations have proven difficult to tune. On the other hand, the algorithm using S-F sequence-pairs exhibits

†However, other important analog issues like thermal constraints, noise coupling, parasitics, or clustering based on oxide thickness are currently not supported.
a clearly higher robustness.

5. Conclusions

This paper has addressed the problem of device-level analog placement in a non-traditional way, using very recent results on topological representations for non-slicing floorplans. Using skillfully the symmetry constraints to remodel and diminish the size of the solution space, the topological representation techniques have proven to be more effective in terms of computation speed than traditional approaches based on the absolute representation.

References


Florin Balasa received the M.S. and Ph.D. degrees in computer science from the Polytechnical University of Bucharest in 1981 and 1994, respectively. He received the M.S. degree in mathematics from the University of Bucharest, Romania, in 1990 and the Ph.D. degree in electrical engineering from the Katholieke Universiteit Leuven, Belgium, in 1995. Dr. Balasa is currently an Assistant Professor of Electrical Eng. and Computer Science at the University of Illinois at Chicago. He worked over seven years at the R&D Institute for Electronic Components, Bucharest, Romania. From 1990 to 1995, he worked at the Interuniversity Microelectronics Center (IMEC), Leuven, Belgium. From the fall of 1995 to 1999, he worked as a Senior Design Automation Engineer at Conexant Systems Inc., Newport Beach, CA. He coauthored the book Custom Memory Management Methodology: Exploration of Memory Organization for Embedded Multimedia System Design, Kluwer Acad. Publ. His research interests include high-level synthesis, physical design, and combinatorial optimization techniques.

Sarat C. Maruvada received his B.E. degree in computer science and engineering from Srim Engineering College, University of Madras in 2000. He is currently a research assistant in the Dept. of Electrical Engineering and Computer Science at the University of Illinois at Chicago, where he is pursuing a graduate program. Since 1998, he has been a member of Indian Institute of Computer Engineers (IIICE). His research interests are in the field of CAD VLSI, with emphasis on physical design automation.