Analog Placement Constraint Extraction and Exploration with the Application to Layout Retargeting

Biying Xu
ECE Department, University of Texas at Austin
biying@utexas.edu

Bulent Basaran
Synopsys, Inc.
Bulent.Basaran@synopsys.com

Ming Su
Synopsys, Inc.
Ming.Su@synopsys.com

David Z. Pan
ECE Department, University of Texas at Austin
dpan@ece.utexas.edu

ABSTRACT

In analog/mixed-signal (AMS) integrated circuits (ICs), most of the layout design efforts are still handled manually, which is time-consuming and error-prone. Given the previous high-quality manual layouts containing valuable design expertise of experienced designers, exploring layout design constraints from the existing layouts is desirable. In this paper, we extract and explore analog placement constraints from previous quality-approved layouts, including regularity and symmetry (symmetry-island) constraints. For the first time, an efficient sweep line-based algorithm is developed to comprehensively extract the regularity constraints in a given analog placement, which can not only improve routability but also minimize the layout parasitics-induced circuit performance degradation. Furthermore, we propose a novel layout technology migration and performance retargeting framework, where we apply the constraint extraction algorithms to improve the placement quality while preserving previous design expertise. Experimental results show that the proposed techniques can preserve the symmetry and regularity constraints, and also reduce the placement area by 7.6% on average.

ACM Reference Format:
Biying Xu, Bulent Basaran, Ming Su, and David Z. Pan. 2018. Analog Placement Constraint Extraction and Exploration with the Application to Layout Retargeting . In *ISPD '18: 2018 International Symposium on Physical Design, March 25–28, 2018, Monterey, CA, USA.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3177540.3178245

1 INTRODUCTION

Analog/mixed-signal (AMS) integrated circuits (ICs) are used widely and heavily in many emerging applications, including consumer electronics, automotive, and Internet of Things (IoT). The increasing demand of these applications calls for a shorter design cycle and time-to-market of AMS ICs. However, most of the design efforts in AMS ICs are still handled manually, which is time-consuming and error-prone, especially as the design rules are becoming more and more complicated in nanometer-scale IC era, and as the circuit performance requirements are becoming more and more stringent. Despite the progress in the AMS IC layout design automation field [1–6], the automatic layout tools have not been widely used among AMS IC layout designers. The AMS IC design automation level is far from meeting the need for fast layout-circuit performance iterations for the rapid growth of the market.

Given the vast amount of previous high-quality analog layouts, it is desirable to explore the layout design constraints from them. The extracted layout constraints can be applied in layout technology migration [7, 8], retargeting to updated design specifications (performance retargeting) [9, 10], and knowledge-based layout synthesis [11], etc., to preserve experts' knowledge across designs, shorten design cycle and reduce design efforts. Layout technology migration and performance retargeting are called *layout retargeting*.

In [7], a layout technology migration tool Migration Assistant Shape Handler (MASH) was proposed, which addressed the geometry scaling between technologies and corrected design rule violations with minimum layout perturbations. Nevertheless, it did not consider the layout design constraints in AMS ICs, including symmetry (symmetry-island [3]), regularity, matching, etc. Then, in [8], N. Jangkrajarng *et al.* presented an intellectual property reuse-based analog IC layout automation tool, IPRAIL, that could automatically retarget existing analog layouts for new process technologies and new design specifications. [9] and [10] further improved the analog layout retargeting algorithm by considering layout parasitics and circuit performance degradation. The above-mentioned prior works [8–10] extracted the topological template from the existing layout so that the target layout had the same layout topology, and they automatically detected symmetry in the existing layouts and carried the symmetry constraints to the target layout, as well. More recently, P.-H. Wu *et al.* proposed a knowledge-based analog physical synthesis methodology to generate new layouts by integrating existing design expertise [11]. It extracted common subcircuits between previous designs and the target design and reused quality-approved layout topologies.

However, the previous works on analog layout retargeting had the following limitations. Firstly, although symmetry constraints were extracted from the existing layouts and preserved in the target layout, regularity constraints were not considered, including topological rows, columns, arrays, and repetitive structures, which are common in AMS ICs. Regular structures in AMS IC layouts not only improve routability, but also minimize the number of vias and bends of wires that are critical, and reduce the layout parasitics-induced

ISPD '18, March 25–28, 2018, Monterey, CA, USA

ACM ISBN 978-1-4503-5626-8/18/03…$15.00
https://doi.org/10.1145/3177540.3178245

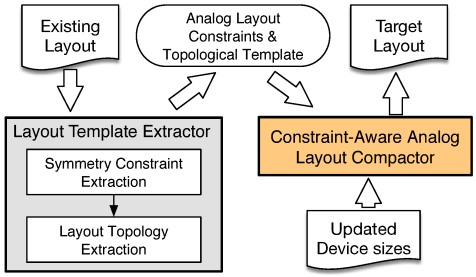

Figure 1: The conventional analog layout retargeting framework.

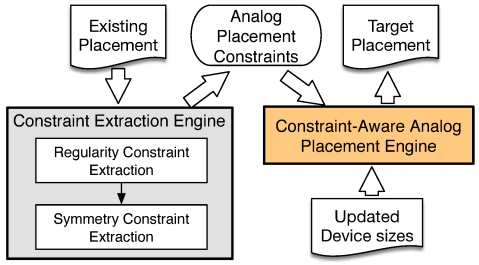

Figure 2: The overall flow of the proposed analog layout retargeting framework.

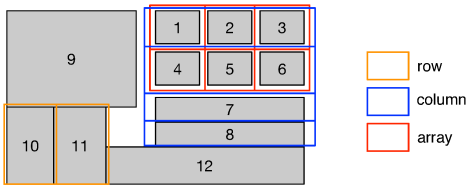

Figure 3: An example of AMS circuit placement. Orange: row constraint; Blue: column constraint; Red: array.

circuit performance degradation [12, 13]. Thus, it is beneficial to take regularity constraints into account. Secondly, previous works extracted a topological template from the existing layout to preserve the entire layout topology. Nevertheless, it may introduce unnecessary extra topological constraints and limit the solution space. In fact, due to new process technologies or updated design specifications, the device sizes may not scale proportionally during layout retargeting, and it may be more desirable to adopt a different layout topology. To address these limitations, in this paper, we extract and explore the regularity and symmetry (symmetry-island) constraints from previous quality-approved layouts. The extracted constraints are applied to a novel analog layout retargeting framework to preserve the design expertise. An efficient sweep line-based algorithm is developed to extract the regularity constraints from a given analog placement. Our main contributions include:

- We extract and explore symmetry and regularity constraints in previous high-quality layouts.
- For the first time, an efficient sweep line-based algorithm is developed to extract all the regularity constraints in an analog placement.
- We propose a novel analog layout retargeting framework based on the extracted constraints, which reduces the placement area compared with the conventional approach while preserving the design expertise.
- Experimental results show the effectiveness and efficiency of the proposed techniques.

The rest of this paper is organized as follows. Section 2 shows the overall flow of the proposed analog layout retargeting framework. Section 3 describes the algorithms to extract analog placement constraints. Section 4 describes the regularity and symmetry constraint-aware analog placement algorithm used in the proposed layout retargeting framework. Section 5 shows the experimental results. Finally, Section 6 concludes the paper.

2 OVERALL FLOW

Conventionally, most of the prior works on analog layout retargeting extract symmetry (symmetry-island) constraints and a topological template of the existing layout, followed by a constraint-aware layout compactor to preserve the extracted constraints and the entire layout topology. The conventional analog layout retargeting framework is shown in Fig. 1, where only symmetry constraints were considered. The layout compactor only generates the layout with the same topology as the existing one.

The proposed analog layout retargeting flow based on placement constraint extraction is shown in Fig. 2. First, the constraint extraction engine explores various analog placement constraints in the existing placement, including regularity and symmetry constraints. Different from the conventional framework that only detects symmetry constraints, we are the first to propose an efficient sweep line-based algorithm to extract the regularity constraints, which will be described in Section 3. Then, given the extracted constraints and the updated device sizes due to new process technologies or new design specifications, a constraint-aware analog placement is performed such that the design expertise is preserved. Adopting constraint-aware analog placement engine instead of analog layout compactor with fixed layout topology provides the potential to explore various different placement topologies, which may result in the placement quality improvement. The constraint-aware analog placement algorithm will be described in Section 4.

3 LAYOUT CONSTRAINT EXTRACTION

3.1 Regularity Constraint Extraction

3.1.1 Problem Definition. AMS IC placements often contain *regular structures* which are also called *regularity constraints* [12]. A regular structure is composed of a group of devices forming a slicing structure whose rectangular bounding box in the placement does not cut any device bounding box. Fig. 3 shows an example of an analog placement with regularity constraints. Each rectangle represents the bounding box of a placement device (e.g. a transistor). The orange bounding boxes indicate row constraints. For example, devices {10, 11} form a row. The blue bounding boxes indicate column constraints. For example, devices {1, 2, 3}, {4, 5, 6} and devices 7, 8 are four rows in a column. The array constraints are indicated by red bounding boxes. For instance, devices {1, 2, 3, 4, 5, 6} form a 2-row-by-3-column array. If all the devices in row/column/array A are inside another bigger row/column/array B, and the set of slicing lines of A is a subset of that of B, we say that row/column/array A is *dominated* by row/column B. As an example, the row consisting

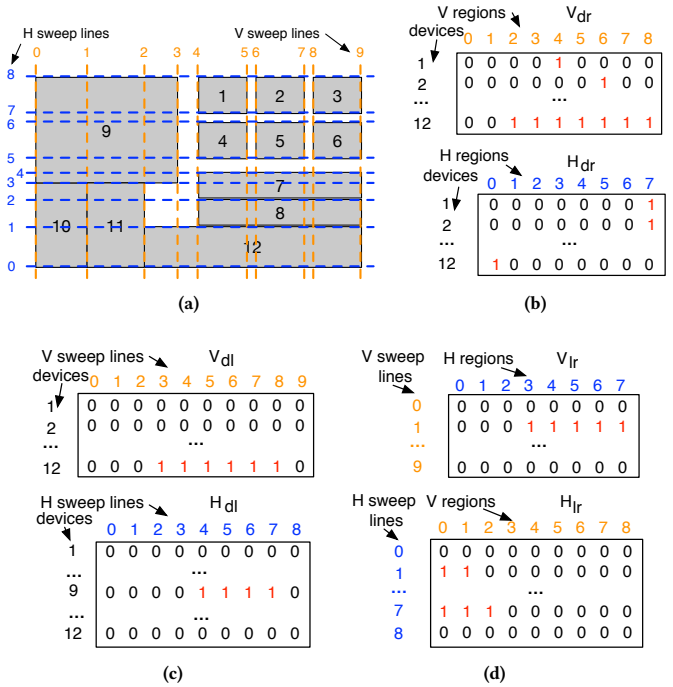

Figure 4: (a) The example analog placement in Fig. 3 with the grids. (b) LUT V_{dr} and H_{dr}. (c) V_{dl} and H_{dl}. (d) V_{lr} and H_{lr}.

of devices {1, 2} is dominated by the row consisting of devices {1, 2, 3}, and the row consisting of devices {1, 2, 3} is dominated by the array consisting of devices {1, 2, 3, 4, 5, 6}. The regularity constraint extraction problem is to find all the regular structures which are non-dominated by any other regular structure in an analog placement.

3.1.2 Lookup Table Construction. We first turn the placement into a grid-based representation. A device bounding box can be represented by the x (horizontal) coordinates of its left and right edges, and the y (vertical) coordinates of its bottom and top edges. The entire placement can be divided by the Hanan grid of all device bounding boxes, which is built based on the horizontal and vertical sweep lines. We construct the sets of horizontal and vertical sweep lines as follows. For each device bounding box in the placement, a *vertical sweep line* is added for the x coordinate of its left edge, and another vertical sweep line is added for the x coordinate of its right edge. For the vertical sweep lines with the same x coordinate, only one is added to the set. Therefore, the total number of vertical sweep lines is less than two times of the number of placement devices. Similarly, two *horizontal sweep lines* are added for the y coordinates of the bottom and top edges of a device, respectively. Fig. 4a shows the placement of Fig. 3 with the Hanan grid. In Fig. 4, "H/V" stands for "horizontal/vertical", respectively.

Then, several Boolean lookup tables (LUTs) are constructed which will be used by the sweep line-based algorithm in Section 3.1.3. For the subscripts of the LUTs, "d" stands for "device", "l" is for "sweep line", and "r" is for "region". For example, the LUT V_{dr} tells us the relationship between a device and a vertical region. The LUTs V_{dr} and H_{dr} shown in Fig. 4b indicate which *region* each device bounding box in Fig. 4a lies in. A region is defined as the range between adjacent sweep lines. For example, a *vertical region* i is between vertical sweep lines i and $i+1$. The number of vertical regions is one less than the number of vertical sweep lines. Each row in V_{dr} indicates which vertical region a device bounding box occupies. As an example, device 12 lies between vertical sweep lines 2 and 9, i.e. it occupies vertical regions $\{2, 3, \cdots, 8\}$. Hence, there are 1's at indices $\{2, 3, \cdots, 8\}$ and 0's elsewhere on the row corresponding to device 12 in V_{dr}. Similarly, the *horizontal region* is defined and H_{dr} is constructed.

The LUTs V_{dl} and H_{dl} shown in Fig. 4c indicate the vertical or horizontal sweep lines a device bounding box strictly intersects. We say that an *HR*/*VR*/ *HSL*/*VSL* (see Table 1) strictly intersect with a device when there is at least some portion of the *HR*/*VR*/*HSL*/*VSL* that is strictly inside the device bounding box, not including the borders of the bounding box. For instance, device 1 does not strictly intersect any vertical sweep line nor horizontal sweep line, so the rows corresponding to device 1 have all 0's in V_{dl} and H_{dl}. Device 9 strictly intersects horizontal sweep lines $\{4, 5, 6, 7\}$, so there are 1's at indices $\{4, 5, 6, 7\}$ and 0's elsewhere on the row corresponding to device 9 in H_{dl}. The first and the last columns in V_{dl} and H_{dl} consist of all 0's, since the first and the last sweep lines will not strictly intersect with any device. Once we have constructed V_{dr} and H_{dr}, V_{dl} and H_{dl} can be computed by bitwise operations efficiently:

$$V_{dl}[d] = V_{dr}[d] \& (V_{dr}[d] \ll 1)$$

$$H_{dl}[d] = H_{dr}[d] \& (H_{dr}[d] \ll 1)$$

where "d" is the device index, "&" stands for the bitwise *and* operation, and "≪" is the bitwise left shift operation.

The LUTs V_{lr} and H_{lr} shown in Fig. 4d indicate whether a sweep line strictly intersects any device bounding box in a region. Each row in the LUT corresponds to a sweep line, and each column in the LUT corresponds to a region. For example, the vertical sweep line 1 strictly intersects device 9 in horizontal regions 3 to 7. Therefore, in LUT V_{lr}, row 1 has 1's in columns $\{3, 4, \cdots, 7\}$. Note that the first and the last rows in V_{lr} and H_{lr} are all 0's, since the first and the last sweep lines will not strictly intersect with any device in any region. LUTs V_{lr} and H_{lr} can be computed from V_{dr}, H_{dr}, V_{dl} and H_{dl} as follows:

$$V_{lr}[i][j] = \bigvee_{d} (V_{dl}[d][i] \wedge H_{dr}[d][j])$$

$$H_{lr}[i][j] = \bigvee_{d} (H_{dl}[d][i] \wedge V_{dr}[d][j])$$

where "d" is the device index, "∧" is the logical conjunction operator, and "⋁" is the logical disjunction operator. Overall, the time complexity for LUT construction is $O(n)$, where n is the number of placement devices.

3.1.3 Sweep Line-Based Algorithm. A sweep line-based algorithm is proposed to extract all the regularity constraints in an analog placement. The notations used are listed in Table 1. The overall algorithm is described in Alg. 1. The algorithm is based on the observation that if there is a slicing line segment between point A and point B for the placement (i.e., the line segment AB does not strictly intersect any device bounding box), then for every point

Table 1: Notations

HSL	Horizontal sweep line.
VSL	Vertical sweep line.
HR	Horizontal region.
VR	Vertical region.
VR_j	The j-th *VR*.
R_{fin}	All regular structure results.
R_{par}	Intermediate/partial regular structures.
r_h, r_v	Consecutive *HR*s and *VR*s in the intermediate/final regular structure.
l_h, l_v	Slicing *HSL*s and *VSL*s in the intermediate/final regular structure.
(r_h, l_h, r_v, l_v)	Represents an intermediate or final regular structure.
f_v	Indicates if the consecutive *HR*s are slicing at a *VSL*.
f_h	Indicates if a set of *HSL*s are slicing at a *VR*.
Q	A set of consecutive *HR*s that are slicing at a *VSL*.

C on AB, AC and CB must also be slicing for the placement. The terminology is defined as follows:

Definition 1 (Slicing Region)**.** If an *HR*/*VR* does not strictly intersect with any device bounding box at a *VSL*/*HSL*, it is a slicing *HR*/*VR* at that *VSL*/*HSL*.

Definition 2 (Sweep Line within a Region)**.** An *HSL*/*VSL* is within a *HR*/*VR* (or a region covers a sweep line) when the coordinate of the *HSL*/*VSL* is within the region or at the region endpoints.

Fig. 5a, Fig. 5b, Fig. 5c and Fig. 5d show the intermediate steps after Alg. 1 proceeds to *VSL*s 1, 2, 3 and 4, respectively, when the algorithm is applied to the analog placement example in Fig. 3.

Lines 1-2 in Alg. 1 initialize the set of final regular structures R_{fin} and the set of intermediate (partial) regular structures R_{par}. For each *VSL* j, in line 4, we obtain the set of consecutive slicing *HR*s, Q, at the j-th *VSL*. We also initialize all consecutive *HR*s in Q to be irredundant, and will use this indicator in subsequent operations. Q can be obtained from the consecutive 0's on the j-th row of LUT V_{lr}, which is explained in Lemma 1. Lemma 1 is correct by construction of the LUT V_{lr}.

Lemma 1. *The consecutive HRs are slicing at a VSL j iff the corresponding indices of the consecutive HRs have consecutive 0's on the row $V_{lr}[j]$ in V_{lr}.*

Each intermediate or final regular structure can be represented by its (r_h, l_h, r_v, l_v) (see Table 1). For each intermediate regular structure in R_{par}, r_v is updated to contain the j-th *VR* (line 7 of Alg. 1). We define two indicator variables f_v and f_h, as shown in lines 11 and 12 and in Table 1.

f_v can be obtained in a way similar to the way of obtaining Q. f_v indicates whether all the consecutive *HR*s in r_h are slicing at *VSL* j. If it is true, one and only one q in Q will overlap with r_h, and Alg. 2 will be executed. In Alg. 2, we first add j to the set of *VSL*s in the intermediate result. Let q be the consecutive slicing *HR*s at j that overlap with r_h, and let l_h^q be the set of *HSL*s within q at which the j-th *VR* is slicing. If l_h^q is the same as l_h in the intermediate result under consideration, then q will be marked as redundant, and no new intermediate regular structure needs to be created for q, since it is contained in the existing intermediate result. This can be illustrated by Fig. 5b. Before processing *VSL* 2, $R_{par}[1]$ (in green) has r_h = {0, 1, 2} and l_v = {0, 1}. The set of consecutive slicing *HR*s Q at *VSL* 2 only has one element q = {0, 1, 2}. Therefore, f_v is true for $R_{par}[1]$ at *VSL* 2, and *VSL* 2 is added to l_v which makes l_v {0, 1, 2}. Since l_h^q = {0, 1, 2, 3} is not the same as l_h = {0, 3}, q is not marked as redundant when processing $R_{par}[1]$ at *VSL* 2.

Otherwise, if f_v is false, there may be zero or more *HR*s in Q overlapping with r_h, and Alg. 3 will be executed. In Alg. 3, for each consecutive slicing *HR*s q overlapping with r_h, l_h^q is obtained as in the case where f_v is true. Then l'_h, which is the intersection between l_h^q and l_h, is calculated and compared with l_h^q. If they are the same, then no new intermediate result for q needs to be added to R_{par}, and q is marked as redundant. After that, we find the minimal *HR*s r'_h covering l'_h and add that intermediate result with *VSL* j to the set R_{par}. This process can be illustrated by Fig. 5a. Before processing *VSL* 1, $R_{par}[0]$ (in red) has r_h = {1, 2, ⋯, 7}, but Q only consists of one element which is q = {0, 1, 2}. Hence, f_v is false for $R_{par}[0]$. l_h^q = {0, 3}, and l'_h is the same as l_h^q. Therefore, q is marked as redundant

Algorithm 1 Regularity Constraint Extraction Algorithm

Input: H_{lr}, V_{lr}
Output: All regular structures R_{fin}
1: $R_{fin} \leftarrow \emptyset$;
2: $R_{par} \leftarrow \emptyset$;
3: **for** each *VSL* j except the last one **do**
4: $Q \leftarrow$ The set of consecutive slicing *HR* at j;
5: Mark each q in Q as irredundant;
6: **for** each (r_h, l_h, r_v, l_v) in R_{par} **do**
7: $r_v \leftarrow r_v \cup VR_j$;
8: $f_v \leftarrow$ Whether r_h are all slicing at *VSL* j;
9: **if** f_v is true **then**
10: UpdateC1((r_h, l_h, r_v, l_v), R_{par}, R_{fin}, Q);
11: **else**
12: UpdateC2((r_h, l_h, r_v, l_v), R_{par}, R_{fin}, Q);
13: **end if**
14: $f_h \leftarrow$ Whether VR_j is slicing at every *HSL* in l_h;
15: **if** f_h is false **then**
16: $l_h^{r_h} \leftarrow$ The set of *HSL*s within r_h at which VR_j is slicing;
17: $l'_h \leftarrow l_h^{r_h} \cap l_h$;
18: $r'_h \leftarrow$ The minimal consecutive *HR*s covering l'_h;
19: Check (r_h, l_h, r_v, l_v) before adding to R_{fin};
20: $r_h \leftarrow r'_h$;
21: $l_h \leftarrow l'_h$;
22: **end if**
23: **end for**
24: **for** each irredundant *HR*s q in Q **do**
25: $l_h^{new} \leftarrow$ The set of *HSL*s within q where VR_j is slicing;
26: $r_v^{new} \leftarrow$ the j-th *VR*;
27: $l_v^{new} \leftarrow$ *VSL* j;
28: Add $(q, l_h^{new}, r_v^{new}, l_v^{new})$ to R_{par};
29: **end for**
30: **end for**
31: **for** each (r_h, l_h, r_v, l_v) in R_{par} **do**
32: $l_v \leftarrow l_v \cup$ the last *VSL*;
33: Check (r_h, l_h, r_v, l_v) before adding to R_{fin};
34: **end for**
35: **return** R_{fin};

Algorithm 2 UpdateC1((r_h, l_h, r_v, l_v), R_{par}, R_{fin}, Q)

1: $l_v \leftarrow l_v \cup j$;
2: $q \leftarrow$ The consecutive slicing *HR*s in Q overlapping r_h;
3: $l_h^q \leftarrow$ The set of *HSL*s within q at which VR_j is slicing;
4: **if** $l_h == l_h^q$ **then**
5: Mark q as redundant;
6: **end if**

Algorithm 3 UpdateC2((r_h, l_h, r_v, l_v), R_{par}, R_{fin}, Q)

1: **for** each consecutive slicing *HR*s q in Q overlapping r_h **do**
2: $l_h^q \leftarrow$ The set of *HSL*s within q where VR_j is slicing;
3: $l'_h \leftarrow l_h^q \cap l_h$;
4: **if** $l'_h == l_h^q$ **then**
5: Mark q as redundant;
6: **end if**
7: $r'_h \leftarrow$ The minimal *HR*s covering l'_h;
8: Add $(r'_h, l'_h, r_v, l_v \cup j)$ to R_{par};
9: **end for**

and no new intermediate result needs to be created for it. We find the minimal *HR*s $r'_h = \{0, 1, 2\}$, and create a new intermediate result with $(r'_h, l'_h, r_v, l_v \cup j)$ (see $R_{par}[1]$ in green in Fig. 5a). Lines 9 to 13 in Alg. 1 call the two functions "UpdateC1" (Alg. 2) and "UpdateC2" (Alg. 3) depending on f_v.

f_h can be obtained from the j-th column of LUT H_{lr} (see Lemma 2, and line 14 of Alg. 1). Similar to Lemma 1, Lemma 2 also holds by construction of H_{lr}.

Lemma 2. *VR j is slicing at an HSL i iff $H_{lr}[i][j]$ is 0.*

If f_h is true, it means the slicing *HSL*s in l_h continues to be slicing at the j-th *VR*. For instance, In Fig. 5a, before processing *VSL* 1, the r_v of $R_{par}[0]$ (in red) is {0} (only *VR* 0 is in r_v). Since all the three *HSL*s in $l_h = \{0, 3, 8\}$ are slicing at *VR* 1, f_h is true and *VR* 1 is added to r_v which becomes {0, 1} after processing *VSL* 1 (see Fig. 5a).

Otherwise, if f_h is false, it means at least one of the *HSL*s in l_h is no longer slicing at the j-th *VR*, and we will add the previous intermediate result to the final result after trimming and checking whether the resulting regular structure is valid (line 19 in Alg. 1). That is, r_h and r_v will be trimmed such that they are the minimal regions covering l_h and l_v (see Definition 2). We also check to ensure that each of the rectangular regions formed by l_h and l_v contains at least one device. This can be done by bitwise operations on V_{dr} and H_{dr}. We further prune the final result such that the regular structure contains more than 2 *HSL*s or more than 2 *VSL*s to form a valid regular structure. Finally, the resulting regular structure will be compared against the existing ones in R_{fin} to ensure that each regular structure is non-dominated by any other regular structure. After that, the existing intermediate result is updated to contain only the continuing slicing *VR* (lines 20 to 21 in Alg. 1). Fig. 5d shows the case where f_h is false. Before processing *VSL* 4, $R_{par}[1]$ (in green) have two *HSL*s $l_h = \{0, 3\}$ (as in Fig. 5c). However, the *VR* 4 is not slicing at *HSL* 3, resulting in f_h being false. We trim the r_v from {0, 1, 2, 3} to {0, 1} (since $l_v = \{0, 1, 2\}$), and make sure that each of the 2 columns in the row contains at least one device

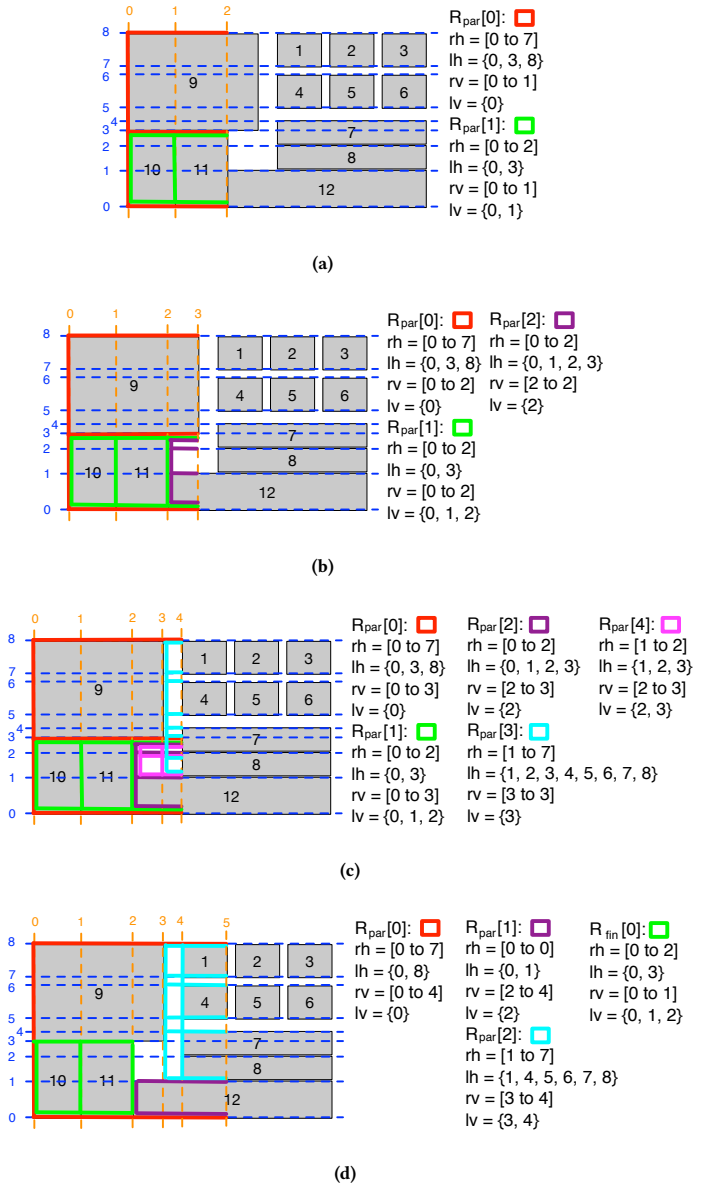

Figure 5: Applying Alg. 1 to the placement in Fig. 3 after processing (a) *VSL*s 0 and 1. (b) *VSL* 2. (c) *VSL* 3. (d) *VSL* 4.

(devices 10 and 11, respectively). Therefore, $R_{fin}[0]$ is added the final result (see Fig. 5d). Since there is only one *HSL* remaining in the original intermediate result which will no longer form a regular structure, it will be removed. Similarly, for $R_{par}[3]$ in Fig. 5c (in cyan), its l_h changes from {1, 2, 3, 4, 5, 6, 7, 8} to {1, 4, 5, 6, 7, 8}. It remains in R_{par} but becomes $R_{par}[2]$ in Fig. 5d since the original intermediate result $R_{par}[1]$ in Fig. 5c (in green) is removed. The resulting candidate to add to R_{fin} with $l_h = \{1, 2, 3, 4, 5, 6, 7, 8\}$ and $l_v = \{3, 4\}$ is not valid because it contains no device. Thus it is not added to R_{fin}.

After iterating through all the intermediate regular structures in R_{par}, for each of the consecutive slicing *HR*s q in Q that remains

irredundant (i.e. that is not marked as redundant by the algorithm), a new intermediate regular structure is added to R_{par}, as can be seen in lines 24-29 in Alg. 1. The new intermediate result has q, the j-th *VR*, and the *VSL* j. Its *HSL*s can be obtained from H_{lr} similarly to the way of getting f_h. Fig. 5b shows an example of adding a new intermediate regular structure for the irredundant consecutive slicing *HR*s in Q. The set of consecutive slicing *HR*s Q at *VSL* 2 only has one element q = {0, 1, 2}. It turns out that q remains irredundant after processing both $R_{par}[0]$ and $R_{par}[1]$. Therefore, a new intermediate result $R_{par}[2]$ (in purple) with r_h = {0, 1, 2}, l_h = {0, 1, 2, 3}, r_v = {2} and l_v = {2} is added to the set R_{par}.

The above procedure is performed for all the *VSL*s except the last one. Finally, in lines 31-34 in Alg. 1, the last *VSL* is added to the l_v for all the intermediate results in R_{par}, and the updated regular structures (r_h, l_h, r_v, l_v) are checked and trimmed before they are added to the R_{fin}, as already described above.

3.1.4 Algorithm Analysis. The proposed sweep line-based algorithm is, in essence, a smart enumeration algorithm with pruning, and is guaranteed to find all the regularity constraints.

Lemma 3. *Alg. 1 finds all regular structures that are non-dominated by any other regular structure in an analog placement.*

Proof. We begin with the proof that the results found by Alg. 1 are slicing structures and are non-dominated by any other regular structure. The proof is by induction. For the first *VSL*, the entire *VSL* is slicing, and it must be irredundant. Therefore, the first intermediate result in R_{par} is obviously slicing (lines 24-29 in Alg. 1). Now assume that the intermediate results in R_{par} are slicing just before the algorithm proceeds to the j-th *VSL*. There are two cases depending on f_v (lines 9-13 in Alg. 1). In Alg. 2, *VSL* j is slicing and is added to l_v, thus the intermediate regular structure is still slicing. In Alg. 3, $(r'_h, l'_h, r_v, l_v \cup j)$ is a slicing structure, and after it is added to R_{par}, the slicing property of the intermediate regular structure still holds. When l'_h is assigned to l_h (line 21 in Alg. 1), the slicing property is maintained. When $(q, l_h^{new}, r_v^{new}, l_v^{new})$ is added to R_{par} (line 28 in Alg. 1), the slicing property is still maintained. As a result, we know that after processing the j-th *VSL*, the intermediate results in R_{par} are all slicing. For the last *VSL*, the entire *VSL* is slicing, after adding it to the l_v of every intermediate regular structure, the results are still slicing. Note that before adding to R_{fin}, the regular structure is compared against other regular structures to ensure the non-dominance condition. Therefore, the results found by Alg. 1 are slicing structures and are non-dominated by any other regular structure.

Next, we show that any regular structure that is non-dominated by other regular structures will be found by Alg. 1. Let $(r_h^*, l_h^*, r_v^*, l_v^*)$ be any of these regular structures. Let j^* be the first *VSL* in l_v^*. *VSL* j^* must be slicing at some irredundant *HR*s q and $r_h^* \subseteq q$, since otherwise the regular structure must be in a larger regular structure, which contradicts with the non-dominance condition. Lines 24-29 in Alg. 1 adds the intermediate result $(q, l_h^{new}, r_v^{new}, l_v^{new})$ to R_{par}, where $r_h^* \subseteq q$, $l_h^* \subseteq l_h^{new}$, $r_v^{new} = VR\,j^*$, $l_v^{new} = VSL\,j^*$. Then, before processing the last *VSL* in l_v^*, for every *HSL* in l_h^{new} but not in l_h^*, there must be some *VR* where it is not slicing, therefore, f_h will be false and the *HSL* will be removed from l_h of the intermediate result. In contrast, all *HSL* in l_h^* will remain in l_h. As for the *VSL*s, from j^* until the last *VSL* in l_v^*, if the *VSL* $\notin l_v^*$, f_v must be false, it is not added to the l_v of the intermediate result. On the contrary, if the *VSL* $\in l_v^*$, there are two cases depending on f_v. If f_v is true, in Alg. 2, the *VSL* is added to l_v of the intermediate result. Otherwise, if f_v is false, in Alg. 3, there must be some $q \in Q$ overlapping r_h^*. r'_h must contain r_h^*, and l'_h must contain l_h^*, hence the newly added intermediate result by line 8 in Alg. 3 will continue to produce $(r_h^*, l_h^*, r_v^*, l_v^*)$. Before processing the last *VSL* in l_v^*, the intermediate result contains exactly l_h^* and all the *VSL*s in l_v^* except the last one. For the last *VSL* in l_v^*, f_v must be true and f_h must be false. The last *VSL* in l_v^* is added to the l_v of the intermediate regular structure which makes it the same as l_v^*, thus the regular structure $(r_h^*, l_h^*, r_v^*, l_v^*)$ is added to R_{fin}. Therefore, any regular structure that is non-dominated by other regular structures will be found by Alg. 1. □

Most of the operations in the proposed algorithm are bitwise operations and efficient. Let n be the number of placement devices. The outer loop of Alg. 1 is executed $O(n)$ times. In each iteration of the outer loop, the number of intermediate regular structures is $O(n^3)$. Inside the inner loop, with bitwise operations and parallelization, the run-time is $O(1)$. Therefore, the time complexity of the proposed algorithm is $O(n^4)$.

3.2 Symmetry Constraint Extraction

The symmetry and symmetry-island constraint extraction algorithm we used is similar to that in [8, 14]. The main steps are sorting the devices by the edges, widths, and heights. Due to the page limit, we will skip the details of the algorithm.

4 CONSTRAINT-AWARE PLACEMENT

After the regularity constraints and symmetry (symmetry-island) constraints are extracted from the existing layout, the next step is to perform constraint-aware analog placement considering the extracted constraints and updated device sizes. The extracted constraints are general and can be applied to any constraint-aware analog placement engine that can handle regularity constraints and symmetry (symmetry-island) constraints, e.g., [2, 6, 12, 15]. The analog placement engine used in our layout retargeting flow is based on the Mixed-Integer Linear Programming (MILP) formulation, and the non-overlapping constraints and symmetry-island constraints are formulated similarly to [6].

As for the regularity constraints, we have the following conditions:

1) Conditions for devices inside the constraint: The devices inside the regularity constraint are separated by slicing lines. The devices in the i-th row and j-th column of the regular structure must lie inside the box formed by the i-th and $(i+1)$-th horizontal slicing lines and the j-th and $(j+1)$-th vertical slicing lines.

Let $D_{i,j,k}$ be the set of devices in the i-th ($i < i^k$) row and j-th ($j < j^k$) column in the regularity constraint k, where i^k is the number of rows and j^k is the number of columns in the regular structure, respectively. Let x_d be the x coordinate and y_d be the y coordinate of device d. For a regularity constraint with i^k rows and j^k columns, there should be $i^k + 1$ *HSL*s and $j^k + 1$ *VSL*s separating the rows and columns. We introduce $i^k + 1$ auxiliary

Table 2: Benchmark AMS IC placements

Placement	#Devices	#Row Const.	#Column Const.	#Array Const.	#Sym. Const.
1	45	3	9	3	14
2	50	5	14	0	18
3	200	20	56	1	72

variables, i.e., $y_r^{k,0}, y_r^{k,1}, \ldots, y_r^{k,i^k}$, to represent the y coordinates of the *HSL*s. Similarly, we introduce $j^k + 1$ auxiliary variables, i.e., $x_r^{k,0}, x_r^{k,1}, \ldots, x_r^{k,j^k}$, to represent the x coordinates of the *VSL*s. Assuming that the rows and *HSL*s are ordered from bottom to top, and that the columns and *VSL*s are ordered from left to right, the devices in $D_{i,j,k}$ must lie inside the box formed by $x_r^{k,j}$, $x_r^{k,j+1}$, $y_r^{k,i}$, and $y_r^{k,i+1}$:

$$x_d \geq x_r^{k,j}, x_d \leq x_r^{k,j+1}, y_d \geq y_r^{k,i}, y_d \leq y_r^{k,i+1}, \forall d \in D_{i,j,k}$$

2) Conditions for devices outside of the constraint: The devices outside of the regularity constraint must not overlap the bounding box formed by the devices inside the regularity constraint.

For each $d' \notin D_{i,j,k}$, we introduce a pair of auxiliary binary variables $s_{d'}^k$ and $t_{d'}^k$ to ensure the non-overlapping property. The following inequalities must be satisfied:

$$\begin{cases} x_{d'} + M_W(s_{d'}^k + t_{d'}^k) \geq x_r^{k,j^k} \\ x_{d'} + w_{d'} - M_W(1 + s_{d'}^k - t_{d'}^k) \leq x_r^{k,0} \\ y_{d'} + M_H(1 - s_{d'}^k + t_{d'}^k) \geq y_r^{k,i^k} \\ y_{d'} + h_{d'} - M_H(2 - s_{d'}^k - t_{d'}^k) \leq y_r^{k,0} \end{cases} \quad \forall d' \notin D_{i,j,k}$$

where M_W and M_H are sufficiently large constants (big-M method [16]), such that no matter what binary values $s_{d'}^k$ and $t_{d'}^k$ are, only 1 out of the 4 inequalities takes effect, while other inequalities are left ineffective.

In the implementation, the orientations of the devices inside a regularity constraint are preserved during retargeting. Also, our placement engine takes the layout compaction result of the conventional layout retargeting approach as a starting point so that our placement quality will not be worse than the conventional approach.

5 EXPERIMENTAL RESULTS

All algorithms are implemented in C++ and all experiments are performed on a Linux machine with 3.4GHz CPU and 32GB memory. Table 2 lists the benchmark information used in our experiments, including small to medium scale analog circuits. In this section, "const." stands for constraints, "sym." stands for symmetry (symmetry-island), and the placement area unit is μm^2. The numbers of rows, columns and arrays can be obtained by a naïve exhaustive enumeration algorithm (with exponential complexity) which searches over all the possible combinations of *HSL*s and *VSL*s and guarantees to find all the regularity constraints.

5.1 Layout Constraint Extraction Results

For all benchmarks, our algorithm can correctly find all the regular structures, which are the same as those found by the naïve exhaustive enumeration algorithm. As an example, Fig. 6 shows the benchmark placement #1, with the regularity constraints found by the proposed algorithm indicated in blue boxes. For instance, devices {2, 7, 9, 15} form a column, and {30, 31, 32, 33} form a 2-by-2 array. Symmetry (symmetry-island) constraints are also extracted as in [8–10]. For all benchmarks, the run-time of the constraint extraction engine is very fast (less than 0.01 seconds), which demonstrates the efficiency of the proposed algorithms.

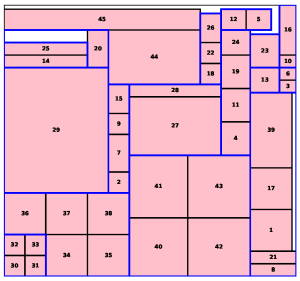

Figure 6: Regularity constraints in benchmark #1.

Table 3: Placement result comparison between the approach in [10] and our layout retargeting framework

Benchmark placement	[10]		Our work		Area reduction
	Area	Run-time	Area	Run-time	
1	23068	1.4s	20586	200s	10.8%
2	36248	2.0s	32752	200s	9.6%
3	134400	5.0s	131040	1200s	2.5%

5.2 Layout Retargeting Results

We implemented the layout topology extraction and layout compaction algorithms in [10] for comparison. For our constraint-aware analog placement engine, we implement it to take the layout compaction result of the approach in [10] as initial starting point so that we will only generate equal or better results. The change of device sizes due to different process technologies or updated design specifications is simulated by deviating the widths and heights from the original values by random percentages. According to the industrial designs, the percentages of the deviation are generated uniformly randomly in the range from -30% to +30% in our experiments. The size deviation percentage of the devices in a symmetry pair should be the same such that they have the same sizes.

The layout retargeting results are shown in Table 3, where for benchmark #1 and #2, we set the run-time limit to be 200s, and for benchmark #3, it is set to 1200s (see Fig. 8 for the selection of the run-time limit). In practice, users can set the run-time limit for our algorithm to achieve run-time and result quality tradeoff. Compared with the approach of [10], our framework explores more layout topologies and consistently achieves placement area reduction for all benchmarks. The layout retargeting results of benchmark #1 are shown in Fig. 7a and Fig. 7b, benchmark #2 in Fig. 7c and Fig. 7d, and benchmark #3 in Fig. 7e and Fig. 7f, respectively. We can see that the regularity and symmetry constraints are preserved for both approaches. However, our approach can achieve more compact placement than that of [10].

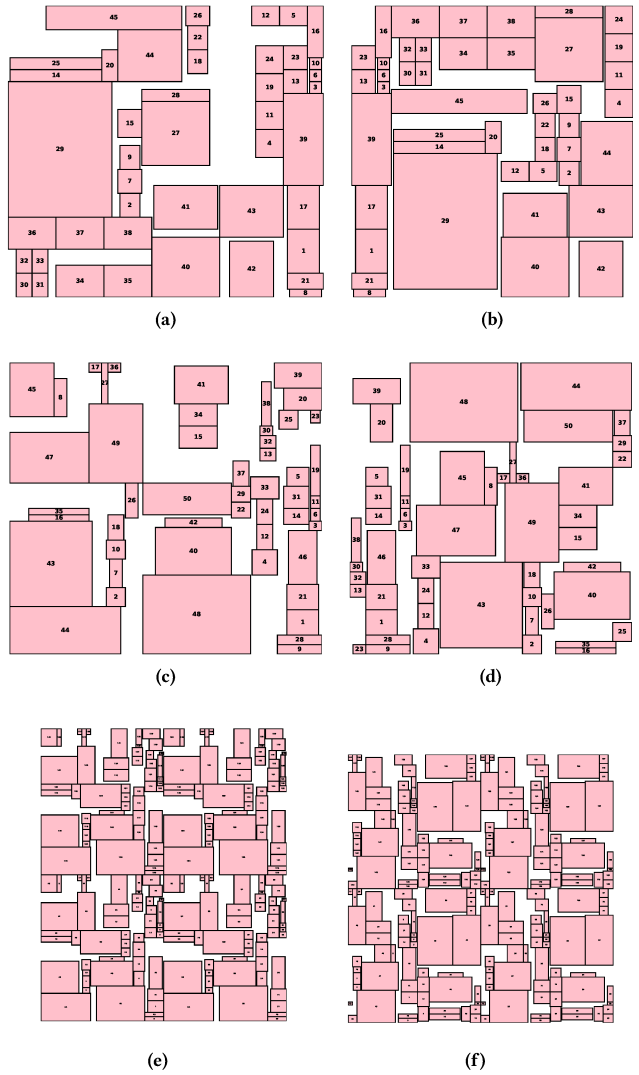

Figure 7: Layout retargeting results: benchmark #1 by (a) [10], and (b) our appoach; benchmark #2 by (c) [10], and (b) our appoach; benchmark #3 by (e) [10], and (f) our appoach.

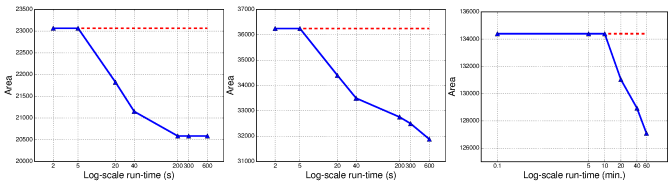

Figure 8: Layout retargeting result area and run-time trade-off of benchmark (a) #1, (b) #2, and (c) #3.

We further plot the convergence curves of running our layout retargeting framework on all benchmarks in Fig. 8. In Fig. 8, the run-time is plotted in logarithmic-scale. In Fig. 8a and 8b, the run-time unit is in "s", while in Fig. 8c the run-time unit is in "min.". The blue lines are the convergence curves of our framework, and the red lines indicate the result area of the approach in [10], which is also the initial starting point used by our placement engine. Although the approach in [10] finishes in a shorter period of time, it only returns one result with the same layout topology as the existing layout, which limits the solution space and may lead to inferior results. Currently, only symmetry and regularity constraints are captured in our method. Other features that are important to AMS circuits (e.g., signal integrity) should also be considered as the future work.

6 CONCLUSION

In this paper, we extract and explore the layout constraints and preserve the design expertise in the previous high-quality AMS IC layouts. Besides considering symmetry constraints as in the prior works, we further develop an efficient sweep line-based algorithm to extract the regularity constraints in an AMS circuit placement. We also propose a novel analog layout retargeting framework based on the extracted constraints, which can provide more flexibility and achieve better placement quality. Experimental results show the effectiveness of the proposed techniques.

ACKNOWLEDGMENT

This work is supported by the National Science Foundation under Grant No. 1527320.

REFERENCES

[1] K. Lampaert, G. Gielen, and W. M. Sansen, "A performance-driven placement tool for analog integrated circuits," *IEEE Journal Solid-State Circuits*, vol. 30, no. 7, pp. 773–780, 1995.
[2] M. Strasser, M. Eick, H. Gräb, U. Schlichtmann, and F. M. Johannes, "Deterministic analog circuit placement using hierarchically bounded enumeration and enhanced shape functions," in *Proc. ICCAD*, 2008, pp. 306–313.
[3] P.-H. Lin, Y.-W. Chang, and S.-C. Lin, "Analog placement based on symmetry-island formulation," *IEEE TCAD*, vol. 28, no. 6, pp. 791–804, 2009.
[4] Q. Ma, L. Xiao, Y.-C. Tam, and E. F. Young, "Simultaneous handling of symmetry, common centroid, and general placement constraints," *IEEE TCAD*, vol. 30, no. 1, pp. 85–95, 2011.
[5] H.-C. Ou, K.-H. Tseng, J.-Y. Liu, I.-P. Wu, and Y.-W. Chang, "Layout-dependent effects-aware analytical analog placement," *IEEE TCAD*, vol. 35, no. 8, pp. 1243–1254, 2016.
[6] B. Xu, S. Li, X. Xu, N. Sun, and D. Z. Pan, "Hierarchical and analytical placement techniques for high-performance analog circuits," in *Proc. ISPD*, 2017, pp. 55–62.
[7] F.-L. Heng, Z. Chen, and G. E. Tellez, "A vlsi artwork legalization technique based on a new criterion of minimum layout perturbation," in *Proc. ISPD*, 1997, pp. 116–121.
[8] N. Jangkrajarng, S. Bhattacharya, R. Hartono, and C.-J. R. Shi, "IPRAIL–intellectual property reuse-based analog ic layout automation," vol. 36, no. 4, pp. 237–262, 2003.
[9] L. Zhang, N. Jangkrajarng, S. Bhattacharya, and C.-J. R. Shi, "Parasitic-aware optimization and retargeting of analog layouts: A symbolic-template approach," *IEEE TCAD*, vol. 27, no. 5, pp. 791–802, 2008.
[10] Z. Liu and L. Zhang, "A performance-constrained template-based layout retargeting algorithm for analog integrated circuits," in *Proc. ASPDAC*, 2010, pp. 293–298.
[11] P.-H. Wu, M. P.-H. Lin, T.-C. Chen, C.-F. Yeh, X. Li, and T.-Y. Ho, "A novel analog physical synthesis methodology integrating existent design expertise," *IEEE TCAD*, vol. 34, no. 2, pp. 199–212, 2015.
[12] S. Nakatake, M. Kawakita, T. Ito, M. Kojima, M. Kojima, K. Izumi, and T. Habasaki, "Regularity-oriented analog placement with diffusion sharing and well island generation," in *Proc. ASPDAC*, 2010, pp. 305–311.
[13] S. Nakatake, "Structured placement with topological regularity evaluation," in *Proc. ASPDAC*, 2007, pp. 215–220.
[14] Y. Bourai and C.-J. Shi, "Symmetry detection for automatic analog-layout recycling," in *Proc. ASPDAC*, 1999, pp. 5–8.
[15] P.-H. Lin and S.-C. Lin, "Analog placement based on hierarchical module clustering," in *Proc. DAC*, 2008, pp. 50–55.
[16] S. Sutanthavibul, E. Shragowitz, and J. Rosen, "An analytical approach to floorplan design and optimization," *IEEE TCAD*, vol. 10, no. 6, pp. 761–769, 1991.