Double Patterning Lithography-Aware Analog Placement

Hsing-Chih Chang Chien\textsuperscript{1}, Hung-Chih Ou\textsuperscript{1}, Tung-Chieh Chen\textsuperscript{2}, Ta-Yu Kuan\textsuperscript{3}, and Yao-Wen Chang\textsuperscript{1,2}

\textsuperscript{1}Graduate Institute of Electronics Engineering, National Taiwan University, Taipei 106, Taiwan
\textsuperscript{2}Department of Electrical Engineering, National Taiwan University, Taipei 106, Taiwan
\textsuperscript{3}Synopsys, Hsinchu, Taiwan

\{lichee626, howard\}@eda.ee.ntu.edu.tw; \{donnie.chen, tayukuan\}@synopsys.com; ywchang@cc.ee.ntu.edu.tw

ABSTRACT
Double patterning lithography (DPL) is one of the most promising solutions for the 28nm technology node and beyond. The main idea of DPL is to decompose the layout into two sub-patterns and manufacture the layout by two masks. In addition to traditional analog design constraints, the pre-coloring constraint should also be considered, in which patterns of critical or sensitive modules have predefined masks before layout decomposition to reduce mismatches. In this paper, we present the first work that considers DPL during analog placement and simultaneously minimizes area, wirelength, and DPL conflicts. We first propose an extended conflict graph (ECG) to represent the relation between patterns of analog modules and apply an integer linear programming (ILP) formulation to determine the orientation of each module and the color of each pattern for conflict minimization. ILP reduction schemes are proposed to further reduce the runtime. Finally, we present a three-stage flow and DPL-aware perturbations to obtain desired solutions. Experimental results show that the proposed flow can effectively and efficiently reduce area, wirelength, and DPL conflicts.

Categories and Subject Descriptors
B.7.2 [Integrated Circuits]: Design Aids [Placement and Routing]

General Terms
Algorithms, Performance

Keywords
Analog ICs, Physical Design, Double Patterning Lithography, Placement

1. INTRODUCTION
Double patterning lithography (DPL) is one of the most promising solutions for the 28nm technology node and beyond [3] [13]. The main idea of DPL is to decompose patterns of a layer into two sub-patterns and then use two masks to manufacture the two sub-patterns, as illustrated in Figure 1. DPL is commonly used for poly, contact, metal, and even via layers. A double patterning decomposer assigns two patterns of a layer to two different masks if their spacing is less than the specific spacing $S_{\text{min}}$ provided by foundry. The double patterning layout decomposition problem is regarded as a two-coloring problem according to a conflict graph (CG), in which a vertex represents a pattern and an edge between two vertices represents that the spacing between the two corresponding patterns is less than $S_{\text{min}}$. Figure 2(a) illustrates the corresponding conflict graph for the layout in Figure 2(b).

![Figure 1: DPL is to decompose a dense pattern into two sub-patterns and then use mask1 and mask2 to manufacture the two sub-patterns.](image)

A DPL conflict occurs when two patterns separated by the distance less than $S_{\text{min}}$ belong to the same mask. To resolve DPL conflicts, a layout engineer could report the detected conflicts to designers and ask them to modify the layout so that the conflicting patterns can be relocated or removed. Nevertheless, layout modification is typically time-consuming and often very costly. Another way is to split one pattern into two sub-patterns and manufacture them by different masks to resolve the DPL conflict, which is regarded as a stitch insertion. Figure 2(c) shows an example of stitch insertion. Nevertheless, stitches might cause significant printability degradation and yield loss due to their overlay errors. Therefore, several works [16] [17] proposed simultaneous conflict and stitch minimization methods for layout decomposition. However, even with the stitches, there could be some irresolvable conflicts, called native conflicts, which cannot be resolved by stitch insertion.

![Figure 2: (a) The conflict graph corresponds to the layout on the right. (b) A DPL conflict is resulted from the two blue patterns separated by less than the specific spacing $S_{\text{min}}$. (c) Stitch insertion is to split one pattern into two sub-patterns, and the two sub-patterns are manufactured by different masks.](image)

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

DAC '13, May 29 - June 07 2013, Austin, TX, USA
Copyright 2013 ACM 978-1-4503-2071-9/13/05 ...$15.00.
considered DPL during detailed routing to generate highly decomposable layouts. Cho et al. [6] proposed the first DPL-friendly grid-based detailed routing algorithm. Lin et al. [9] proposed an innovative conflict graph (ICG) to assist DPL-aware gridless detailed router. However, no literature is known for DPL-aware analog placement.

To handle the analog placement problem, most previous works apply topological floorplan representations such as sequence-pair [4], TCGS [7], hierarchical B*-tree (HB*-tree) [8], O-tree [12], and corner stitching compliant B*-tree (CB-tree) [14]. Although various analog design constraints, such as symmetry, proximity, variant, fixed-boundary, minimum-separation, and boundary constraints, have been studied in previous works, DPL is not considered in these previous works. As a result, their resulting layouts may not be DPL-decomposable. Further, since the unavoidable misalignment of different masks would cause variations, analog designers usually reduce mismatches by predefining masks (colors) for the poly and contact layers of critical or sensitive modules before double patterning decomposition [2]. This pre-coloring constraint makes analog placement with DPL consideration much harder during design time.

The pre-coloring constraint significantly complicates DPL decomposition. Without pre-colors, DPL conflict detection is equivalent to odd cycle detection in the induced conflict graph; once the graph is two-colorable, there is no odd cycle. However, no odd cycle cannot guarantee that the layout is decomposable if a pre-coloring pattern causes a “conflict path” in the graph. Figure 3(a) shows an example of placement with pre-coloring constraints, and Figure 3(b) shows a poly layer before DPL decomposition with pre-defined color1 for critical modules. No matter what combination of colors is chosen, a conflict path always exists, as shown in Figure 3(c).

Figure 3: (a) The resulting analog placement with pre-coloring constraints. (b) A placed poly layer before DPL decomposition. (c) A conflict path exists after decomposition.

In this paper, we propose the first algorithm that solves the DPL-aware analog placement. We summarize our main contributions as follows:

- We propose an extended conflict graph (ECG) to model the global relationship among non-flipped and flipped patterns; DPL conflicts can be reduced by flipping modules to prevent area increase.
- We develop an integer linear programming (ILP) algorithm considering symmetry and pre-coloring constraints to minimize conflicts. Two speedup schemes, connected-component computation and two-edge-connected-component-computation, are proposed to reduce the runtime of ILP.
- We propose DPL-aware and module-merging perturbations to assist simulated annealing for achieving better solutions and better performance.
- A three-stage flow for DPL-aware analog placement is proposed. The three-stage flow minimizes area, wirelength, and DPL conflicts simultaneously.

The rest of this paper is organized as follows: Section 2 reviews the CB-tree representation and analog design constraints and formulates the DPL-aware analog placement. Section 3 introduces our ECG and ILP formulation. Section 4 proposes general and DPL-aware perturbations and presents our three-stage analog placement flow. Section 5 reports the experimental results. Finally, Section 6 concludes this paper.

2. PRELIMINARIES

In this section, we first give a brief review of the CB-tree [14] and then formulate the DPL-aware analog placement problem.

2.1 Review of CB-trees

For constraint-driven analog placement, CB-tree is the most effective and efficient topological representation with module adjacency information in the literature. Since we need the adjacency information to check whether there is any conflict in the placement result, we adopt the CB-tree representation in this work. A CB-tree [14] is a B*-tree integrated with modified corner stitching to augment a B*-tree for module adjacency handling. The B*-tree is an ordered binary tree representing a compacted placement [5], and corner stitching is a data structure for modeling rectangular objects with a tile plane to provide efficient geometric operations [11].

2.2 Review of Analog Constraints

For easier presentation (and also the space limit), we focus on symmetry and proximity constraints which are the two most important, intensively studied analog constraints addressed in the literature, and demonstrate how to integrate the DPL constraints with them for advanced analog placement. Though not presented in this paper, other less popular analog constraints can readily be implemented with a similar flow as ours to be presented later.

- **Symmetry constraints** are used to place some pairs of modules symmetrically along a vertical or a horizontal symmetry axis. It can reduce mismatches of sensitive modules.
- **Proximity constraints** are used to place modules at closest proximity to reduce process variation.

To handle the two most important constraints, Lin et al. proposed an HB*-tree with hierarchy and contour nodes [8]. The hierarchy nodes handle symmetry and non-symmetry modules simultaneously and guarantee feasible placement results. Since the symmetry pairs may not be rectangular, contour nodes representing top horizontal contour segments are introduced to handle rectilinear shapes of hierarchy nodes during packing. In a CB-tree, tile updating is based on the contour nodes so that it can also handle rectilinear regions.

2.3 Problem Formulation

The DPL-aware analog placement problem can be formally defined as follows:

- **The Double Patterning-Aware Analog Placement Problem:** Given a set of rectangular modules \( M = \{ m_k \mid 1 \leq k \leq |M| \} \), a set of nets \( N = \{ n_k \mid 1 \leq k \leq |N| \} \), a set of placement constraints \( S = \{ s_k \mid 1 \leq k \leq |S| \} \), and a set of pre-coloring constraints \( R = \{ r_k \mid 1 \leq k \leq |R| \} \), place all the modules in \( M \) to minimize the total area, wirelength, and DPL conflicts such that no modules overlap with each other and all the placement constraints are satisfied.

3. DPL CONFLICT HANDLING

In this section, we first discuss the techniques for resolving an analog placement with DPL conflicts. Considering pre-coloring constraints, we then present an extended conflict graph to model the global relationship among patterns/modules and derive an ILP formulation to determine the orientation of each module and the color of each pattern simultaneously to minimize DPL conflicts.

3.1 Module Flipping

To resolve an analog placement with DPL conflicts, we could simply increase the spacing between two conflicting patterns/modules. After increasing the spacing, however, we should pack the CB-tree again, and the layout area could become larger accordingly, as illustrated in Figs. 4(a) and (b). To minimize the area and the packing time, we could flip some modules to resolve the conflicts. As illustrated in Figure 4(c), we could flip the top module vertically not only to solve the conflict in
the contact layer but also to keep the same outline of layout. Moreover, since flipping would not change the coordinate of a module, we do not need to re-pack modules after module flipping.

![Distance < S_{\text{min}}](image1)

Figure 4: (a) The originally packed layout with DPL conflicts on the contact layer. (b) Conflict removal by enlarging the spacing between modules. (c) Conflict removal by flipping the top module.

### 3.2 Extended Conflict Graph (ECG)

In the traditional conflict graph construction, a vertex represents a pattern, and an edge is introduced if the spacing between two corresponding patterns is less than $S_{\text{min}}$. The traditional conflict graph considers the information from a fixed layout and cannot capture the location updates for polys and contacts with their orientation changes. Since a poly can only be placed vertically in advanced nodes, it might not be allowed to rotate a device arbitrarily. As a result, there are only two choices for module orientations: non-flipped and flipped. We use non-flipped and flipped vertices to present poly and contact patterns of non-flipped and flipped modules, respectively. For edges, since the orientations of patterns in the same module should be the same, we only construct edges from non-flipped vertices to non-flipped vertices and from flipped vertices to flipped vertices for patterns in the same module. Figure 5 illustrates a module and the corresponding vertices of poly and contact patterns.

![Non-flipped m1, Flipped m1](image2)

Figure 5: (a) A non-flipped module $m_1$. (b) Non-flipped vertices of poly and contact patterns in $m_1$. (c) A flipped module $m_1$. (d) Flipped vertices of poly and contact patterns in $m_1$.

To record the relation of patterns belonging to different modules, we should check the distance of all the patterns with different orientations. There are four combinations of orientations for every two modules: (1) non-flipped to non-flipped (see Figure 6(a)), (2) non-flipped to flipped (see Figure 6(b)), (3) flipped to non-flipped (see Figure 6(c)), and (4) flipped to flipped (see Figure 6(d)). We check the distances of all patterns for the four combinations and introduce an edge if the distance between two corresponding patterns is less than $S_{\text{min}}$. Figure 6(e) shows the ECG of module $m_1$ and module $m_2$.

![Non-flipped m1 and non-flipped m2, Flipped m1 and non-flipped m2, Flipped m1 and flipped m2, ECG of m1 and m2](image3)

Figure 6: (a) Non-flipped $m_1$ and non-flipped $m_2$. (b) Non-flipped $m_1$ and flipped $m_2$. (c) Flipped $m_1$ and non-flipped $m_2$. (d) Flipped $m_1$ and flipped $m_2$. (e) ECG of $m_1$ and $m_2$.

### 3.3 Basic Integer Linear Programming (ILP) Formulation

After the ECG construction, we apply an ILP formulation to determine the orientation of each module and the color of each pattern in the contact and poly layers so that the number of conflicts is minimized. To reduce the mismatches of critical modules, our ILP formulation also considers symmetry and pre-coloring constraints.

The notations used in the ILP formulation are as follows:

- $M$: set of modules.
- $M^S$: set of symmetry pairs.
- $V$: set of vertices.
- $V^c$: set of non-flipped vertices.
- $V':$ set of flipped vertices.
- $V^c$: set of vertices with pre-defined color1.
- $V^f$: set of vertices with pre-defined color2.
- $E$: set of edges.
- $m_i$: a module.
- $m^S_k$: a symmetry pair.
- $p_i$: a constant denoting the number of poly and contact patterns in module $m_i$.
- $v_{ij}$: vertex $j$ of module $m_i$.
- $e_{ij,i'j'}$: an edge connecting two vertices $v_{ij}$ and $v_{i'j'}$.
- $r^1(v_{ij})$ and $r^1(v_{ij})$: 0-1 integer variable that denotes the color of vertex $v_{ij}$. If the color of vertex $j$ of the module $m_i$ is 1, $r^1(v_{ij}) = 1$; otherwise, $r^1(v_{ij}) = 0$.
- $r^2(v_{ij})$: 0-1 integer variable that denotes the color of vertex $v_{ij}$. If the color of vertex $j$ of the module $m_i$ is 2, $r^2(v_{ij}) = 1$; otherwise, $r^2(v_{ij}) = 0$.
- $c(e_{ij,i'j'})$: 0-1 integer variable that denotes the existence of a conflict between two vertices $v_{ij}$ and $v_{i'j'}$. If the distance between the vertex $j$ of the module $m_i$ and the vertex $j'$ of the module $m_{i'}$ is less than the specified spacing $S_{\text{min}}$ and the color of them are the same, $c(e_{ij,i'j'})=1$; otherwise, $c(e_{ij,i'j'})=0$.

The conflict minimization problem can be formulated as follows:

$$
\text{minimize} \quad \sum_{e_{ij,i'j'} \in E} c(e_{ij,i'j'}) \\
\text{s.t.} \quad r^1(v_{ij}) + r^2(v_{ij}) + r^1(v_{i,j+p_i}) + r^2(v_{i,j+p_i}) = 1, \quad \forall v_{ij} \in V^c, \forall v_{i,j+p_i} \in V', \quad (1) \\
\quad r^1(v_{ij}) + r^2(v_{ij}) = r^1(v_{i,j+1}) + r^2(v_{i,j+1}), \quad \forall v_{i,j}, v_{i,j+1} \in V^c, \quad (2) \\
\quad r^1(v_{i1}) + r^2(v_{i1}) = r^1(v_{i',1}) + r^2(v_{i',1}), \quad \forall m_i, m'_i \in M^S_k, \forall m^S_k \in M^S, \quad (3) \\
\quad r^1(v_{ij}) + r^2(v_{ij}) \leq 1 + c(e_{ij,i'j'}), \forall e_{ij,i'j'} \in E, \quad (4) \\
\quad r^2(v_{ij}) + r^2(v_{i'j'}) \leq 1 + c(e_{ij,i'j'}), \forall e_{ij,i'j'} \in E, \quad (5) \\
\quad r^2(v_{ij}) = 0, \forall v_{ij} \in V^c, \quad (6) \\
\quad r^1(v_{ij}) = 0, \forall v_{ij} \in V'. \quad (7)
$$

The objective function is to minimize the total number of conflicts. Constraint (1) ensures that only one orientation and only one color can be chosen for each pattern. Figure 7 illustrates the corresponding patterns of the four variables in Constraint (1). If $r^1(v_{ij})$ equals one, the corresponding contact is non-flipped and colored color1. If $r^2(v_{ij})$ equals one, the corresponding contact is flipped and colored color2. Since there are four patterns in the module $m_1$, $p_1$ equals four and the corresponding flipped vertex of $v_{i1}$ is $v_{i1}$.5. If $r^1(v_{1,5})$ equals one, the corresponding contact is flipped and colored color1. If $r^2(v_{1,5})$ equals one, the corresponding contact is flipped and colored color2. Constraint (2) is used to guarantee that the poly and contact patterns in the same module $m_i$ are all non-flipped or...
all flipped. Constraint (3) guarantees that the modules belonging to a symmetry pair have the same orientation. Constraints (4) and (5) denote that if the distance between two vertices \( v_{i,j} \) and \( v_{i',j'} \) is less than \( 5m_{ii} \) and the two vertices have the same color, a conflict occurs. Constraints (6) and (7) model the pre-coloring constraint; if a vertex has a pre-assigned color, the variable of the opposite color should be equal to zero.

**Theorem 1.** Given an ECG with \( |V| \) vertices and \( |E| \) edges, the number of ILP variables and the number of ILP constraints are both \( O(|V|+|E|) \).

<table>
<thead>
<tr>
<th>(a)</th>
<th>(b)</th>
<th>(c)</th>
<th>(d)</th>
</tr>
</thead>
<tbody>
<tr>
<td>( r^1(v_{1,1}) = 1 )</td>
<td>( r^2(v_{1,1}) = 1 )</td>
<td>( r^1(v_{1,2}) = 1 )</td>
<td>( r^2(v_{1,2}) = 1 )</td>
</tr>
</tbody>
</table>

Figure 7: (a) The variable \( r^1(v_{1,1}) = 1 \) if the color of vertex 1 of the module \( m_1 \) is 1. (b) \( r^2(v_{1,1}) = 1 \) if the color of vertex 1 of the module \( m_1 \) is 2. (c) \( r^1(v_{1,5}) = 1 \) if the color of vertex 5 of the module \( m_1 \) is 1. (d) \( r^2(v_{1,5}) = 1 \) if the color of vertex 5 of the module \( m_1 \) is 2.

### 3.4 ILP Problem-Size Reduction

In this subsection, we propose two linear-time speedup schemes for the ILP formulation. The two schemes are based on the divide-and-conquer algorithm to divide a whole problem into subproblems.

#### 3.4.1 Connected Component Computation

A connected component of a graph is a subgraph in which any two vertices are connected to each other with paths. Since the poly and contact patterns of the same module should be considered simultaneously, these vertices belonging to the same module are clustered before we compute the connected component. Moreover, since the modules in the same symmetry group should be with the same orientation to reduce mismatches, the vertices of modules in a symmetry pair are also clustered. Different from the traditional graph division methods, we compute the connected components for a clustered graph. Since the components are independent, the ILP formulation can be applied for each component, and the solution of the overall problem can be obtained by taking the union of all sub-solutions and maintain the optimality.

#### 3.4.2 Two-Edge-Connected Component Computation

Two-edge-connected-component computation is to identify an edge whose removal would decompose the given graph into two components. We apply the ILP formulation to handle the two components independently. Since there is only one connection between the two components, the two sub-solutions can be merged with a simple color remapping process. For instance, we can find two components by removing an edge in Figure 8(a). After solving the two subgraphs, as shown in Figure 8(b), we recolor one of the components and merge them so that no conflict occurs. However, the conflict path may be disconnected if the pre-coloring vertices belong to different components. Figure 8(d) illustrates a graph and two pre-coloring vertices. If we remove an edge and perform the basic ILP formulation to each component, as shown in Figure 8(e), a conflict occurs after merging the components because there is a pre-coloring vertex in each component and the two components cannot be recolored. Figure 8(e) illustrates the condition of the conflict path. Therefore, we also cluster the pre-coloring vertices to handle the conflict path. After applying the ILP formulation on each two-edge-connected component, the solution of the overall problem can be obtained by merging sub-solutions.

After applying our proposed reduction schemes, the number of ILP variables and the number of ILP constraints are still \( O(|V|+|E|) \). However, the schemes significantly reduce the runtime of our ILP formulation for practical applications, as will be reported in Section 5.

#### 4. THE ALGORITHM FLOW

We propose a three-stage flow to handle the simultaneous area, wirelength, and DPL conflict minimization problem. Figure 9 shows the underlying ideas of each stage. Since our DPL-aware analog placement flow is based on simulated annealing, DPL-aware perturbations are proposed to find a desired solution with fewer conflicts.

![Figure 8:](image)

Figure 8: (a) Two-edge-connected-component computation. (b) Components are solved independently. (c) One of the components is recolored and the two components are merged. (d) Two-edge-connected-component computation with the existence of pre-coloring vertices. (e) The components are handled independently. (f) A conflict occurs after merging components.

![Figure 9:](image)

Figure 9: The overall flow of our DPL-aware analog placement.

#### 4.1 Perturbation

We apply the following operations to perturb a CB-tree in Stage 1 and Stage 3:

- Op1: Delete a node and insert it into another place.
- Op2: Swap two nodes.
- Op3: Merge two nodes according to merging guidelines.
- Op4: Demerge two nodes.

Perturbations for a classical CB-tree include deletion/insertion, rotation, and swapping. Because polys can be placed in only one direction (say, vertically) for advanced process and a module is not allowed to be rotated to an arbitrary orientation, we only adopt the deletion/insertion and swapping operations, as in Op1 and Op2. To obtain higher performance and lower parasitic, we apply the module merging operation, which is a common technique used in analog designs. Merging modules can shorten interconnects and reduce area to achieve better solution quality. We merge modules according to two guidelines: (1) merged modules are of the same diffusion type (PMOS/NMOS), and (2) merged modules belong to the same net [10] [15]. To handle a rectilinear module during packing, we use a hierarchy node to represent a corresponding merged module and a contour node to represent a top horizontal contour segment [8]. To prevent a solution from getting stuck at a local minimum, we also propose a demerging operation to separate merged modules during the simulated annealing process.

To consider DPL effects, we further apply the following DPL-aware operations to perturb a CB-tree in Stage 2:

- Op1: Extract the node with the maximum number of conflicts (the maximum-conflict node) and insert it into another place.
- Op2: Swap the maximum-conflict node with another node.
Figure 10: The detailed three-stage DPL-aware analog placement flow.

Figure 11: (a) The original placement with a conflict. (b) The placement after merging modules without conflict.

4.2 The Three-Stage Placement Flow

Figure 10 details our three-stage flow. In Stage 1, given an initial placement represented by a CB-tree, we randomly choose one perturbation and update the tile plane to handle analog constraints. The cost function $\phi_1$ of the placement is defined in Equation (8), where $\alpha$ and $\beta$ are user-specified parameters, $A$ is the chip area and $W$ is the half-perimeter wirelength (HPWL).

$$\phi_1 = \alpha A + \beta W.$$  \hspace{1cm} (8)

We use the cost function $\phi_1$ to evaluate the placement in Stage 1. With simulated annealing, we keep perturbing the CB-tree and record the best solution until a predefined termination condition is met. Since this stage is focused on obtaining a placement with acceptable area and wirelength and does not consider the conflict minimization, it does not apply the ILP formulation. In Stage 2, we further consider conflict minimization, while maintaining the area and wirelength quality. We perform DPL-aware perturbations to find a placement with minimum conflicts. During packing, once a node is packed, we update the tile plane and use area enumeration (in corner stitching) to check the distance between patterns of the corresponding node and patterns of adjacent nodes for ECG updating. In particular, the time complexity of the area enumeration is only $O(1)$ because the CB-tree selects an initial searching tile (with corner stitching) near the given node. As illustrated in Figure 12, the top module is being packed and the searching area is extended by the distance $S_{\text{min}}$ from the module boundary. After checking the distance between patterns, the ECG is updated, and then we formulate an ILP according to the updated ECG to determine the orientation of each module and the color of each pattern for conflict minimization. The cost function $\phi_2$ of the placement is defined in Equation (9), where $\alpha$, $\beta$, and $\gamma$ are user-specified parameters and $C$ is the number of conflicts of the placement.

$$\phi_2 = \alpha A + \beta W + \gamma C.$$  \hspace{1cm} (9)

We use the cost function $\phi_2$ to evaluate the placement in Stage 2. If the number of conflicts $C$ is zero, we go to Stage 3. Otherwise, we keep performing DPL-aware perturbations to find solutions with fewer conflicts until a predefined termination condition is met. In Stage 3, to reduce the runtime, we first evaluate the area and wirelength after every perturbation. The ILP programs for conflict minimization can be skipped if the corresponding area and wirelength are not acceptable. Moreover, because it is desired to obtain a decomposable layout for modern design, a placement with non-zero conflict is not acceptable in this stage. Therefore, we can maintain the placement with zero conflict while further improving the area and wirelength.

Table 1: Design Statistics and Constraint Number of case1 and case2.

<table>
<thead>
<tr>
<th></th>
<th>PMOS</th>
<th>NMOS</th>
<th>Net</th>
<th>Poly</th>
<th>Contact</th>
<th>Sym.</th>
<th>Pre-color</th>
</tr>
</thead>
<tbody>
<tr>
<td>Case1</td>
<td>8</td>
<td>8</td>
<td>11</td>
<td>32</td>
<td>230</td>
<td>2</td>
<td>8</td>
</tr>
<tr>
<td>Case2</td>
<td>14</td>
<td>16</td>
<td>34</td>
<td>92</td>
<td>344</td>
<td>2+2</td>
<td>8</td>
</tr>
</tbody>
</table>
Table 2: Comparison of Area, Wirelength, Conflict number and runtime between CB-tree [14] and our three-stage flow.

<table>
<thead>
<tr>
<th>Circuits</th>
<th>Area (nm²)</th>
<th>HPWL (nm)</th>
<th>Conflict number</th>
<th>Runtime (s)</th>
<th>Area (nm²)</th>
<th>HPWL (nm)</th>
<th>Conflict number</th>
<th>Runtime (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>W/o ILP</td>
<td>W/ILP</td>
<td>W/o ILP</td>
<td>W/ILP</td>
<td>W/o ILP</td>
<td>W/ILP</td>
<td>W/o ILP</td>
<td>W/ILP</td>
</tr>
<tr>
<td>Case1</td>
<td>2032050</td>
<td>9930</td>
<td>10</td>
<td>12</td>
<td>1960350</td>
<td>7373</td>
<td>0</td>
<td>41</td>
</tr>
<tr>
<td>Case2</td>
<td>3207437</td>
<td>1972</td>
<td>13</td>
<td>32</td>
<td>3014900</td>
<td>13155</td>
<td>0</td>
<td>197</td>
</tr>
<tr>
<td>Comp.</td>
<td>1.07</td>
<td>1.43</td>
<td>0.14</td>
<td>0.18</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
</tr>
</tbody>
</table>

5. EXPERIMENTAL RESULTS

We implemented our DPL-aware analog placement flow in the C++ programming language, and all the experiments were performed on an Intel Xeon X5647 2.93GHz Linux workstation with 48GB memory. We used the CPLEX12.3 [1] library to solve the ILP problems. To evaluate our flow, we used two industrial analog circuits Case1 and Case2. The circuits were scaled down to 28 nm for the experiments. The statistics of the two scaled circuits are shown in Table 1, where “PMOS”, “NMOS”, and “Net” list the number of PMOSs, NMOSs, and nets, respectively. Since the number of vertices in our proposed ECG is equal to the number of poly and contact patterns, we also report the number of patterns in Table 1, where “Poly” gives the number of total poly patterns, and “Contact” gives the number of total contact patterns. Finally, “Sym.” gives the number of symmetric MOSs and “Pre-color” gives the number of pre-colored poly and contact patterns.

Because there is no previous work on the DPL-aware analog placement, we compared our DPL-aware analog placement flow with the following two methods: (1) CB-tree-based placement without considering DPL and (2) CB-tree-based placement with our ILP-based conflict minimization. Table 2 compares the results, including area, half-perimeter wirelength length (HPWL), the number of DPL conflicts, and placement runtime. The column “W/o ILP” shows the results of CB-tree, which does not consider DPL; the column “W/ ILP” reports the results with our ILP formulation for conflict minimization after the CB-tree result is generated. The experimental results show that applying our ILP formulation after placement can resolve a large portion of conflicts. In particular, our flow considering DPL during placement can obtain zero conflict under reasonable runtime. With module merging (6 modules in Case1 and 8 modules in Case2), our method can achieve an average area reduction of 7% and an average wirelength reduction of 43%. Figs. 13(a) and (b) show the respective poly and contact layers of Case1 generated by our three-stage flow.

To show the efficiency of our three-stage flow, we implemented a one-stage flow, which applies DPL-aware perturbations and ILP formulation for conflict minimization in each iteration of simulated annealing. Table 3 shows the runtime comparisons between the one-stage flow and our three-stage flow, with and without applying our ILP reduction schemes. Here, “W/o Red.” denotes the runtime without the reduction and “W/ Red.” denotes the runtime with the reduction. The experimental results show that the runtime of the one-stage flow is more than two hours. In contrast, our three-stage flow can efficiently solve the problem because the unnecessary ILP computations can be avoided. Although our three-stage flow can further be speeded up with the ILP reductions.

Table 3: Runtime comparison between the one-stage and our three-stage flows.

<table>
<thead>
<tr>
<th>Circuits</th>
<th>One-stage Flow</th>
<th>Three-stage Flow</th>
</tr>
</thead>
<tbody>
<tr>
<td>Case1</td>
<td>&gt; 2 hr</td>
<td>&gt; 2 hr</td>
</tr>
<tr>
<td>Case2</td>
<td>&gt; 2 hr</td>
<td>&gt; 2 hr</td>
</tr>
<tr>
<td>Comp.</td>
<td>1.33</td>
<td>1.00</td>
</tr>
</tbody>
</table>

6. CONCLUSIONS

In this paper, we have presented a DPL-aware analog placement flow to simultaneously minimize area, wirelength, and DPL conflicts. We have proposed an extended conflict graph and ILP formulations for conflict minimization. In addition, the DPL-aware perturbations and the three-stage flow have been proposed to further improve the solution quality and efficiency. Experimental results based on two industrial analog designs have shown that our flow is effective and efficient.

7. REFERENCES