Fast and Efficient Bright-Field AAPSM Conflict Detection and Correction

Charles Chiang, Andrew B. Kahng, *Member, IEEE*, Subarnarekha Sinha, Xu Xu, *Student Member, IEEE*, and Alexander Z. Zelikovsky

***Abstract*—Alternating-aperture phase shift masking (AAPSM), a form of strong resolution enhancement technology, will be used to image critical features on the polysilicon layer at smaller technology nodes. This technology imposes additional constraints on the layouts beyond traditional design rules. Of particular note is the requirement that all critical features be flanked by opposite-phase shifters while the shifters obey minimum width and spacing requirements. A layout is called phase assignable if it satisfies this requirement. Phase conflicts have to be removed to enable the use of AAPSM for layouts that are not phase assignable. Previous work has sought to detect a suitable set of phase conflicts to be removed as well as correct them. This paper has two key contributions: 1) a new computationally efficient approach to detect a minimal set of phase conflicts, which when corrected will produce a phase-assignable layout, and 2) a novel layout modification scheme for correcting these phase conflicts with small layout area increase. Unlike previous formulations of this problem, the proposed solution for the conflict detection problem does not frame it as a graph bipartization problem. Instead, a simpler and more computationally efficient reduction is proposed. This simplification greatly improves the runtime while maintaining the same improvements in the quality of results obtained in Chiang *et al.* (*Proc. DATE*, 2005, p. 908). An average runtime speedup of 5.9× is achieved using the new flow. A new layout modification scheme suited for correcting phase conflicts in large standard-cell blocks is also proposed. The experiments show that the percentage area increase for making standard-cell blocks phase assignable ranges from 1.7% to 9.1%.**

***Index Terms*—Automatic layout, phase conflict, phase-shift mask, resolution enhancement.**

I. Introduction

As ADVANCED technologies in wafer manufacturing push the patterning processes toward a lower k_1 subwavelength printing, reticle based resolution enhancement techniques (RET) have played a critical enabling role. Alternating-aperture phase shift masking (AAPSM) is a form of strong RET that uses phase modulation at the mask level to enhance the resolution limit of current lithography equipment. At smaller technology nodes, it will be widely used to image features of the polysilicon layer. Among several variants of AAPSM, bright-field AAPSM is the most viable technology for the polysilicon layer [1]. In a simple model of bright-field AAPSM, each critical feature, which is a shape in the design whose width is below a certain threshold value, must be flanked by two phase shifters of opposing phases in order to create destructive interference between them. There are additional constraints of size and spacing that the shifters must obey in order to ensure a manufacturable mask.

Manuscript received November 1, 2005; revised February 27, 2006. This paper was presented in part at the 2005 ICCAD. This work was supported in part by the National Science Foundation under Award 0429735. This paper was recommended by Associate Editor S. Sapatnekar.

C. Chiang and S. Sinha are with Synopsys, Inc., Mountain View, CA 94043-4024 USA (e-mail: Charles.Chiang@synopsys.com; Subarna.Sinha@synopsys.com).

A. B. Kahng is with the Department of Computer Science and Engineering, and the Department of Electrical and Computer Engineering, University of California at San Diego, La Jolla, CA 92093-0114 USA (e-mail: abk@ucsd.edu).

X. Xu is with Blaze DFM, Sunnyvale, CA 94089 USA (e-mail: xuxu@blaze-dfm.com).

A. Z. Zelikovsky is with the Computer Science Department, Georgia State University, Atlanta, GA 30303 USA (e-mail: alexz@cs.gsu.edu).

Digital Object Identifier 10.1109/TCAD.2006.882642

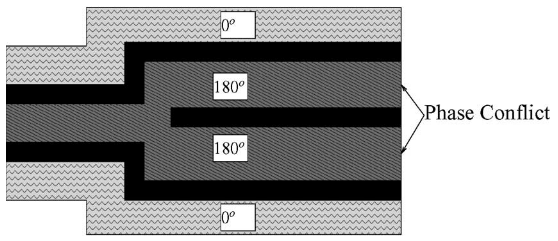


Fig. 1. Example of incorrect phase assignment.

Given a layout with shifters inserted around each critical feature, it is phase assignable if and only if there is a phase-assignment solution which meets the following requirements.

1) Shifters on opposite sides of every critical feature are assigned opposite phases (0° and 180°).
2) Shifters that are separated by less than the minimum shifter spacing should be merged and assigned the same phase. Two shifters separated by less than the minimum shifter spacing will be referred to as overlapping shifters.

Two shifters are in phase conflict if they violate the previous conditions in a phase-assignment solution in which each shifter is assigned a phase. Fig. 1 illustrates an example where the previous conditions are violated due to a cyclic sequence of shifters that cannot be properly mapped. The phase conflict detection problem seeks to find a minimum set of phase conflicts, which when corrected will result in a phase-assignable layout. The phase conflict correction problem corrects a given set of phase conflicts by layout/mask modification with minimum increases in area or mask complexity.

Our contributions are summarized.

1) A new and computationally efficient algorithm for detecting phase conflicts, which when corrected, will render the given layout phase assignable. Unlike the bipartization formulation, which is the basis of all previous work, we formulate the conflict detection problem as a

conflict cycle removal problem. This leads to a substantial reduction in the number of nodes/edges in the constructed graphs and thereby produces average runtime speedups of 5.9× while maintaining the same quality of results as the best results available today [2].

2) A novel layout modification algorithm for correcting a selected set of phase conflicts that achieves small area increase and good scalability for large standard-cell blocks.[1] Compared to the approach in [2], which presents the only available results on layout modification for correcting phase conflicts for bright-field AAPSM, our new approach can reduce the maximum area increase from $> 100.0\%$ to 9.1% for large designs.

This paper is organized as follows: Section II briefly reviews the previous work in phase conflict detection and correction. In Section III, we provide a detailed discussion of the new theory of the proposed phase conflict detection flow. Section IV discusses our new conflict correction algorithm. Experimental results are presented in Section V. We end with conclusions and directions for future work in Section VI.

II. Previous Work

The phase conflict detection problem is addressed in [2], [4]–[7]. The basic underlying principle in these works is to translate the phase conflict detection problem to a graph bipartization problem of a suitably constructed graph. Thus, a conflict detection would involve identifying a set of edges such that the modified graph obtained after deleting the edges is bipartite. The work in [6] and [7] formulates the phase conflict detection problem as a minimum-weight graph bipartization problem to minimize the amount of layout modification necessary to render the layout phase assignable. It is assumed that the constructed graphs will always be embedded planar graphs,[2] and an optimal solution is provided for that case. The most recent work in this area is presented in [2]. This algorithm works on general layouts and is a generalization of the scheme presented in [7]. The layout is represented as a graph called the phase conflict graph. A new bipartization algorithm that does not require the input graph to be an embedded planar graph is proposed. The algorithm creates a planar subgraph of the given graph, applies a computationally efficient version of the optimal bipartization algorithm [6] on the planar subgraph to get an optimal solution, and then combines this solution with a greedy solution for the edges deleted during planarization. The quality of results (in terms of number of conflicts selected for correction and runtime numbers) was significantly better than previous work in this area. This phase conflict detection algorithm will be used as our reference for comparison because it outperforms other existing work in the area.

Previous work in phase conflict correction falls into two major categories. Mask-level correction based approaches [8] split shifter regions whenever two shifters of opposite phases overlap to avoid the layout modification. However, the mask complexity is increased and it is not always possible to split the shifter regions without negatively affecting process latitude. Layout modification based approaches remove the conflicts by increasing the spacing between features or widening critical features [2], [4]–[6], [9], [10]. Most of these works focus on dark-field AAPSM.[3] The first layout modification scheme for correcting the bright-field phase conflicts is presented in [2].[4] The key idea in this paper is to add a minimal number of end-to-end spaces through the layouts to separate all shifter pairs corresponding to the phase conflicts by the desired spacing. While this technique is suitable for standard cells and some macro blocks with a relatively small number of conflicts, experimental results show that it is highly unsuitable for standard-cell blocks with a large number of phase conflicts for correction.

There are also some cell-based solutions that propose to add blank space around each cell to avoid introducing phase conflicts between the neighboring cells or introduce an additional requirement that all the boundary elements should have the same phase [11]. No results were presented in the context of standard-cell blocks. However, we believe both these methods are very conservative and could lead to unnecessary increases in area since only a small fraction of the phase conflicts involve features of different cells.

III. Proposed Phase Conflict Detection Scheme

In this section, the proposed phase conflict detection scheme is presented. As shown in Fig. 2, the proposed conflict detection flow is presented as follows.

1) Conflict cycle graph generation. A conflict cycle graph G is constructed from a given layout L.
2) Planar graph embedding. The phase conflict graph G is not necessarily an embedded planar graph, which is required by the optimal algorithm. Hence, G is converted to an embedded planar graph G' by greedily removing minimum weight conflict edges that cross other edges. These conflict edges are added to a potential set of AAPSM conflicts P.
3) Optimal conflict removal for planar graph. An optimal minimum-weight conflict cycle removal algorithm "Bipartize," described in Section III-B, is applied to G' for choosing the minimum set of AAPSM conflicts that when corrected will produce a phase-assignable layout. The list of edges deleted by the algorithm is added to D, which denotes a minimal set of AAPSM conflicts which when removed will ensure that G' is phase assignable.
4) Computation of final set of AAPSM conflicts. It is necessary to check if any of the edges deleted during planar embedding, i.e., the conflict edges in P, lead to phase

[1]T-shaped phase conflicts and other local phase conflicts can be easily detected with simple DRC and corrected with phase splitting [8], feature widening [14], or cell redesign. Like the approach in [2], we assume that T-shaped conflicts are already corrected by other methods. We only consider conflict correction with spacing, i.e., increasing space between features, due to its small impact on timing and mask complexity.

[2]An embedded planar graph is one that has no line crossings when embedded in a plane.

[3]In dark-field AAPSM, phases are assigned to the critical features themselves. This form of phase is not likely to be used on the polysilicon layer.

[4]Although the work in [7] proposes the conflict detection methods for bright-field phase conflicts based on feature widening, it neglects the layout modification problem.

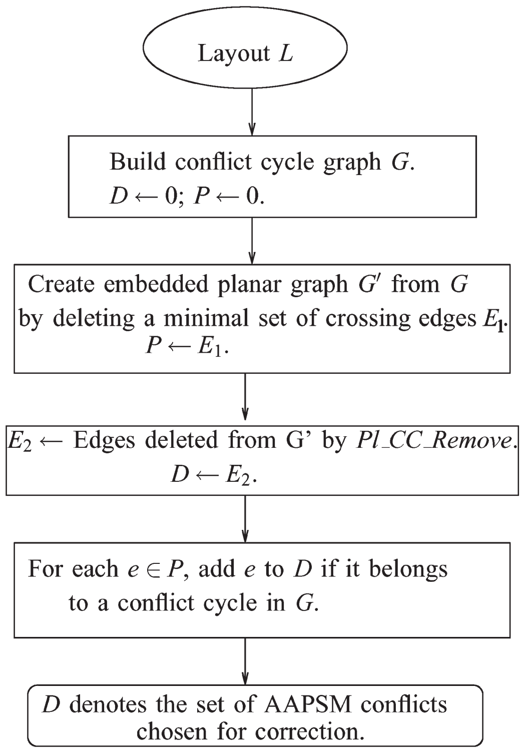

Layout L
Build conflict cycle graph G. $D \leftarrow 0$; $P \leftarrow 0$.
Create embedded planar graph G' from G by deleting a minimal set of crossing edges E_1. $P \leftarrow E_1$.
$E_2 \leftarrow$ Edges deleted from G' by *Pl_CC_Remove*. $D \leftarrow E_2$.
For each $e \in P$, add e to D if it belongs to a conflict cycle in G.
D denotes the set of AAPSM conflicts chosen for correction.

Fig. 2. Phase conflict detection flow.

conflict. This is accomplished by two-coloring G' after deleting the edges in D. If the two shifters of $e \in P$ are in phase conflict, e is added to the set D. At this point, D has a minimal set of edges/AAPSM conflicts which when removed will make G phase assignable.

Unlike previous formulations of the problem, the proposed solution does not reduce the problem to a bipartization problem. Instead, the phase conflict detection problem is reduced to a new problem called the minimum-weight conflict cycle removal problem (the problem will be introduced formally in the next section). This new reduction enables the construction of a much simpler graph from the layout. This graph, called the conflict cycle graph, removes all superfluous edges that were introduced in the phase conflict graph construction to make them bipartite for phase-assignable layouts. This simplification enables a significant reduction in the number of edges compared to the phase conflict graph [2] (an average reduction of 31% in the number of edges is achieved using the simpler graph).

A further advantage of this new formulation is that an optimal polynomial-time algorithm exists for the minimum-weight conflict cycle removal problem when the input graph is an embedded planar graph. The optimal algorithm is used as a subroutine in the proposed phase conflict detection algorithm. The use of the optimal algorithm ensures that the quality of results returned by our phase conflict detection algorithm is comparable to the best results returned by previous work [2], since large subgraphs of the input graph are solved using an optimal algorithm. In addition, the new theory enables the removal of certain edges that are marked undeletable and cannot be selected by the phase conflict detection algorithm. This also results in significant speedups of the phase conflict detection algorithm. Experimental results on representative examples show average speedups of $5.9\times$ using the proposed approach while maintaining the same quality of results as the method in [2].

The key novel and distinguishing features of the proposed phase conflict detection scheme from previous methods can be summarized.

1) Representation of the layout as a conflict cycle graph and development of its relationship to phase assignability of the layout.
2) Reduction of the phase conflict detection problem to a minimum-weight conflict cycle problem (to be defined later) on the conflict cycle graph and an optimal polynomial-time algorithm for the same, when the graph is an embedded planar graph.
3) Improvements to the intermediate reductions such that edges that cannot be selected by the conflict detection algorithm do not need to be explicitly represented.

The following sections include a detailed discussion of these points.

A. *Conflict Cycle Graph*

The first step of the conflict detection algorithm is to build a conflict cycle graph. Given a layout L, the conflict cycle graph $G = (N, E \cup F)$ consists of shifter nodes N, conflict edges E, and feature edges F.

1) For every shifter, create an edge shifter node $n \in N$.
2) For two overlapping shifters[5] s_1 and s_2, create a conflict edge $e \in E$ connecting n_1 and n_2. Here, n_1 and n_2 are the edge shifter nodes for s_1 and s_2, respectively.
3) Create a feature edge $f \in F$ between the two shifters that are on opposite sides of a critical feature.

Fig. 3(a) shows an example of a conflict cycle graph for the layout shown earlier in Fig. 1. The conflict cycle graph has six nodes and six edges. By comparison, the phase conflict graph [shown in Fig. 3(b)] of the same layout has 11 nodes and 11 nodes. Experimental results in a later section show a substantial reduction in the node/edge count, which results in significant runtime improvements. However, unlike the phase conflict graph, the conflict cycle graph does not equate the phase assignability of its corresponding layout to bipartition. Therefore, a conflict cycle graph may be bipartite even if its corresponding layout is not phase assignable. For instance, the layout in Fig. 3(a) is not phase assignable, even though its corresponding conflict cycle graph is bipartite. Thus, a new criterion is needed for detecting the phase conflicts using the conflict cycle graph.

It should be further clarified that in the conflict cycle graph, the feature edges and conflict edges play different roles. Nodes connected by feature edges should be assigned different phases, and nodes connected by conflict edges should be assigned the same phase. Later in this section, we discuss how in certain applications the feature edges are only used to appropriately classify the cycles they belong to and can be dropped during

[5]Two shifters that are separated by less than the minimum shifter spacing are called overlapping shifters.

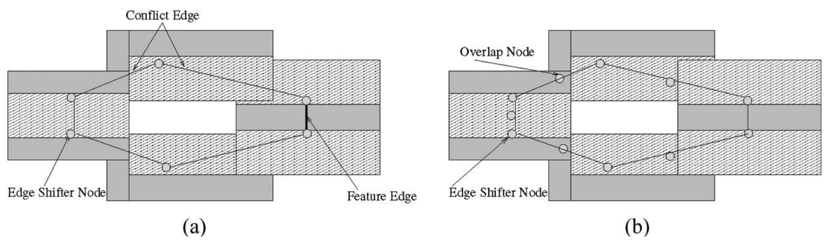


Fig. 3. (a) Conflict cycle graph. (b) Phase conflict graph.

```
Input: Conflict cycle graph $G$
Output: Phase assignment of $G$
1. failed$\leftarrow$False; all nodes are uncolored
2. While (failed==False AND $\exists$ uncolored nodes)
3.    Pick a random uncolored node $n_0$ as the
        root and assign it color 0, $S_0 \leftarrow \{n_0\}$
4.    Put all the nodes connected with at
        least one node in $S_0$ in a set $S_1$.
5.      For (all colored nodes $n_1 \in S_1$)
6.        Check nodes $n_2 \in S_0$ connected to
            $n_1$ with edge $e$ for the two rules:
            (1) if $e$ is a conflict edge, the color of
              $n_1$ should be the same as $n_2$;
            (2) if $e$ is a feature edge, the color of
              $n_1$ should be different from $n_2$.
            If the rules are violated, failed$\leftarrow$True
7.      If (all nodes in $S_1$ are colored)
          return to Step 2
        Else
8.        For (any uncolored nodes $n_1 \in S_1$)
9.          Arbitrarily choose one node $n_2 \in S_0$
              connected to $n_1$ with edge $e$:
              (1) if $e$ is a conflict edge, the color
                of $n_1$ is the same as $n_2$;
              (2) if $e$ is a feature edge, the color
                of $n_1$ is different from $n_2$.
10.       $S_0 \leftarrow S_1$, return to Step 4
11. If (failed==True) $G$ is not phase assignable
```

Fig. 4. Phase-assignment algorithm.

some intermediate graph constructions, thereby producing more speedups.

The general phase-assignment algorithm is shown in Fig. 4. A layout is phase assignable if and only if the Boolean variable "failed" remains false at the end of the assignment process.

In the rest of this section, we present the theory for phase conflict detection using the conflict cycle graph.

Definition 1: A conflict cycle is a cycle which contains an odd number of feature edges.

Theorem 1: A layout is phase assignable if and only if the corresponding conflict cycle graph has no conflict cycles.

Proof: ($\rightarrow$) Assume L is phase assignable. Let all the edge shifter nodes be colored with the same phases as the shifters in L. It is true that the node colorings of G satisfy the following two conditions. Nodes connected by a feature edge have different colors, and nodes connected by a conflict edge have the same color. Let us assume further that there exists a conflict cycle C and let $\{n_1, n_2, \ldots, n_k, n_1\}$ be a closed walk along C. By the definition of a conflict cycle, there are an odd number of feature edges in C. Hence, starting from n_1, the node phases will flip an odd number of times in C. Therefore, the node n_1 will be assigned two different phases, which is impossible. Hence, our assumption that G, whose corresponding layout L is phase assignable, has a conflict cycle is wrong.

($\leftarrow$) Assume G does not contain any conflict cycles and L is not phase assignable. Then, the phase-assignment process specified in Fig. 4 must violate the rules for two nodes n_1 and n_2 connected with the edge e. There must be two paths from the root to n_1 and n_2. Let n_0 to be the last common node on the two paths. Then, there is a cycle $C = \{n_0, \ldots, n_1, n_2, \ldots, n_0\}$. The following are the two possible cases.

1) n_1 and n_2 have the same color and e is a feature edge: Since the colors are only changed across the feature edges, if n_1 has the same color as n_0, then there must be an even number of feature edges from n_0 to n_1. By assumption, n_2 has the same color as n_1, and hence as n_0. Thus, there must be an even number of feature edges from n_0 to n_2. Then, C must contain an odd number of feature edges, and hence, C is a conflict cycle.
2) n_1 and n_2 have different colors and e is a conflict edge: If n_1 and n_0 have the same color, then there must be an even number of feature edges on the path from n_1 to n_0. By assumption, n_2 has a different color from n_1 and hence a different color from n_0. Thus, the path from n_0 to n_2 must have an odd number of feature edges. Hence, C must contain an odd number of feature edges and is a conflict cycle.

This contradicts our initial assumption that G has no conflict cycles. Hence, our assumption that L is not phase assignable is wrong. ■

In order to make the layout phase assignable, it is necessary to remove all conflict cycles from the conflict cycle graph by deleting the edges. The deleted edges directly correspond to the phase conflicts that have to be corrected. Each edge has a given weight which reflects the negative effects of correcting the phase conflict.[6] A large number of phase conflicts selected for correction would imply large changes to the layout and/or mask, which is highly undesirable. Hence, it is essential to minimize the sum of weights of the edges to be deleted during the conflict cycle removal.

The minimum-weight conflict cycle removal problem is defined as follows: Given a conflict cycle graph $G = (V, E)$, remove a minimum-weight set of edges E' such that the modified graph $G' = (V, E \setminus E')$ does not have any conflict cycles.

It can be easily proved that this problem is NP hard for general graphs by doing a simple reduction to the minimum-weight bipartization problem. However, an optimal polynomial-time

[6]The weighting scheme depends on the layout modification methods, which will be discussed in Section IV.

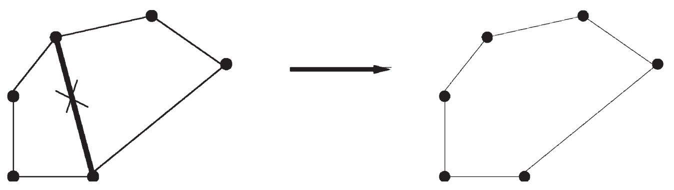


Fig. 5. Deleting all common edges (in this case, only one) results in a merged face.

algorithm exists when the graph is an embedded planar graph. This optimal algorithm is referred to as `Pl_CC_Remove` in Fig. 2 and will be discussed in detail in the next section.

B. Optimal Minimum-Weight Conflict Cycle Removal Algorithm for Embedded Planar Graphs

The theory of the optimal polynomial-time algorithm for minimum-weight conflict cycle removal for embedded planar graphs is presented in this section. Let G denote an embedded planar graph for which we seek the optimal solution of the minimum-weight conflict cycle removal problem.

Definition 2: A conflict face of G is a face corresponding to a conflict cycle in G. A face of G that is not a conflict face is a legal face.

Definition 3: The dual graph G^{D} of the conflict graph G is constructed by representing every face f of G with a node n. An edge e which belongs to faces g_1 and g_2 in G is represented with an edge $\mathrm{e}' = \{n_1, n_2\}$ in G^{D}. A node $n \in G^{\mathrm{D}}$ corresponding to a conflict face $f \in G$ is called a conflict node. A node that is not a conflict node is a legal node.

Definition 4: Two faces are neighboring faces if they share at least one common edge. The merged face of two neighboring faces is formed by deleting all common edges.

Lemma 1: The parity of the number of feature edges of the merged face is equal to the parity of the sum of the numbers of feature edges of two faces.

Proof: Let the two faces have m_1 and m_2 feature edges and they share m_3 feature edges. Then, the merged face has $m_1 + m_2 - 2m_3$ feature edges, which has the same parity as $m_1 + m_2$. ■

Lemma 2: A planar embedded graph G has no conflict cycles if and only if all faces are legal.

Proof: For a planar embedded graph, any cycle is the result of merging n faces. If all faces are legal, we know that the number of feature edges in the merged face is even from Lemma 1. Therefore, by definition, the graph has no conflict cycles. If the original graph has no conflict cycles, then every face is legal by definition. ■

Theorem 2: Removing an odd number of edges from every conflict face and an even number of edges from every legal face will generate a graph with no conflict cycles.

Proof: Let G' be the graph obtained after the edge deletion. As shown in Fig. 5, the deletion of one or more common edges results in the creation of a merged face. Any face in G' must be the result of merging a set S_1 of conflict faces and a set S_2 of legal faces in G. Let $S = S_1 \bigcup S_2$.

We first want to prove that the cardinality of S_1, $|S1|$, is even. Let $r(f)$ denote the number of removed edges for each face $f \in S$. Since all the removed edges belong to two faces in S and are counted twice, $\sum_{f \in S} r(f)$ is even. $\sum_{f \in S} r(f) = \sum_{f \in S_1} r(f) + \sum_{f \in S_2} r(f)$. From the assumption, an even number of edges are removed from every legal face, i.e., $r(f)$ is even for $f \in S_2$. Therefore, $\sum_{f \in S_2} r(f)$ is even and hence $\sum_{f \in S_1} r(f)$ is even. Since $r(f)$ is odd for every $f \in S_1$, $|S1|$ must be even.

Then, we want to prove that the sum of the feature-edge numbers of all faces in S is even. Since every face in S_1 has odd number of feature edges and there are an even number of faces in S_1, the sum of the numbers of feature edges of all faces in S_1 is even. Also, the sum of the numbers of feature edges of all faces in S_2 is even, since every face in S_2 has even number of feature edges according to the definition of legal faces. Therefore, the sum of the feature-edge numbers of all faces in S is even.

According to Lemma 1, the feature-edge number of the merged face is even since the sum of the feature-edge numbers of all faces in S is even. Hence, any merged face in G' is legal. Thus, G' has no conflict cycles according to Lemma 2. ■

The problem of deleting a minimum-weight set of edges, such that an odd number of edges are deleted from every conflict face and an even number of edges are deleted from every legal face of G, translates to the following problem on its dual graph G^{D} [7]:

Find the minimum-weight set of edges S to be deleted in $G^{\mathrm{D}} = (V, E)$ such that: 1) **an odd number of edges in S are incident on every conflict node $u \in V$ and** 2) **an even number of edges in S are incident on every legal node $\nu \in V$.**

This is similar in spirit to the T-join problem [12] on a graph G which can be optimally solved. The T-join problem of a graph seeks a minimum-weight edge set S such that a node u is incident to an odd number of edges of S if and only if u belongs to the node subset T of the given graph. Our problem reduces to the T-join problem if and only if the set of all conflict nodes is denoted as the set T. Unlike the problem formulation in [2] in which T is the set of all nodes with odd degrees, in our formulation, T may include nodes with odd or even degrees.

Next, we describe how the T-join problem can be reduced to a perfect matching problem on a suitably constructed gadget graph $\mathcal{G}$. The gadget graph construction consists of steps.

1) Dual edge assignment. Each edge e connecting ν and ν' in dual graph is assigned to ν, ν' or both. The assignment is done such that the following conditions are satisfied.[8]
   a) For each conflict node ν, the number of true nodes in G_ν is odd.
   b) For each legal node ν, the number of true nodes in G_ν is even.

[7]Given a planar graph G, its geometric dual G_D is constructed by placing a vertex in each face of G (including the exterior face) and, if two faces have an edge in common, joining the corresponding vertices by a dual edge.

[8]To ensure that the conditions are satisfied, we use the edge assignment method in [6] to assign the edges such that for any gadget of n nodes, the number of ghost nodes is at least $\lfloor \frac{n}{2} \rfloor$. Then, for any gadget G_ν, which violates the parity requirement, it is always possible to turn a ghost node g_e^ν into a true node t_e^ν (i.e., assign the edge e to both ν and ν') to meet the parity requirement at the cost of increasing the node number of the gadget graph by one.

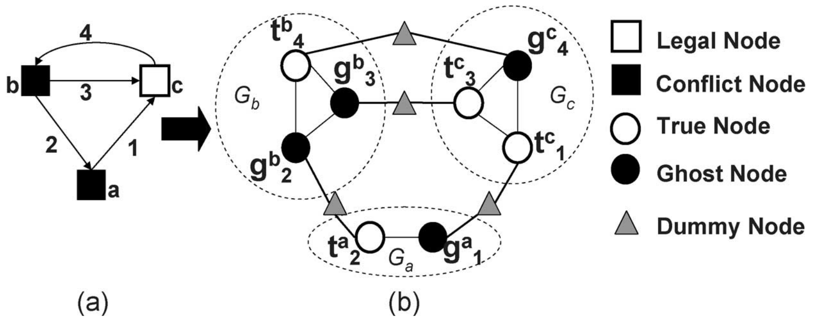


Fig. 6. Gadget graph construction from dual graph. The directions on the edges in (a) are used to signify the edge assignment. (a) Dual graph. (b) Gadget graph.

In Fig. 6(a), directed edges are used to represent the assignment.

2) Gadget node construction. If a dual edge e connecting ν and ν' is assigned to ν and not assigned to ν', it will appear as a true node t_e^ν in G_ν and a ghost node $g_e^{\nu'}$ in $G_{\nu'}$.[9] As a result, each node ν of degree k in the dual graph G^{D} becomes a gadget of k nodes in $\mathcal{G}$, which is denoted as G_ν. The weight of $g_e^{\nu'}$ is $w(e)$ and the weight of t_e^ν is zero. In other words, the weight of any dual edge is always assigned to its corresponding ghost node. Both nodes are connected to a dummy node with zero weight edges. In any perfect matching solution for the gadget graph, exactly one node in each pair of t_e^ν and $g_e^{\nu'}$ will be matched within gadgets since the other node will be matched with the dummy node.

3) Complete gadget construction. The nodes in G_ν are connected to each other by weighted edges to form a complete graph. The weight of any edge in G_ν is the total weight of its two nodes. Fig. 6(b) shows the gadget graph constructed from the dual graph of Fig. 6(a).

In summary, $\mathcal{G} = (V', E')$, where V' includes the true nodes, ghost nodes, and dummy nodes, and E' is the set of edges between nodes in V'.

Theorem 3: The T-join problem for a graph $G^{\mathrm{D}} = (V, E, w, T)$, where T denotes the conflict nodes and $V \setminus T$ denotes the legal nodes in V, can be reduced to a minimum-weighted perfect matching on the gadget graph $\mathcal{G} = (V', E', w')$.

Proof: $(\rightarrow)$ Mapping a perfect matching solution of the gadget graph $\mathcal{G}$ to a valid solution of the T-join problem on G^{D}. For any node ν in the dual graph G^{D}, divide the edges of node ν into four sets.

1) $S_1 = \{e | g_e^\nu \text{ matched within } G_\nu\}$.
2) $S_2 = \{e | g_e^\nu \text{ not matched within } G_\nu\} = \{e | t_e^{\nu'} \text{ matched within } G_{\nu'}\}$.
3) $S_3 = \{e | t_e^\nu \text{ matched within } G_\nu\}$.
4) $S_4 = \{e | t_e^\nu \text{ not matched within } G_\nu\} = \{e | g_e^{\nu'} \text{ matched within } G_{\nu'}\}$.

We need to prove that the set $S = S_1 \bigcup S_4$ thus constructed is a valid solution to the T-join problem. Let the cardinality of S_1, S_2, S_3, and S_4 be a, b, c, and d, respectively. In any perfect matching solution, the number of nodes matched within G_ν, $(a + c)$, is even. If ν is a conflict node, the number of true nodes in G_ν, $c + d$, is odd by construction and $(a + c) + (c + d) = (a + d) + 2c$ is odd. Therefore, the number of edges in S, $a + d$, is odd. Similarly, if ν is a legal node, the number of edges in S is even. Therefore, the solution S is a valid solution of the T-join problem.

Since the weight of any edge e in the dual graph is always assigned to its corresponding ghost node g_e^ν, the total weight of the edges in the T-join solution, $S = S_1 \bigcup S_4$, is equal to the total weight of all ghost nodes matched within gadgets.

On the other hand, the total weight of the matching solution, i.e., the total weight of the matched edges, $=$ the total weight of the matched edges within gadgets (since the weights of edges incident to dummy nodes are all zero), $=$ the total weight of all nodes matched within gadgets (since the edge weight is the total weight of its two nodes), and hence $=$ the total weight of all ghost nodes matched within gadgets (since the weights of all true nodes are zero). Therefore, the total weight remains the same during the mapping.

$(\leftarrow)$ Mapping a solution S of the T-join problem to a solution of the perfect matching problem of $\mathcal{G}$ can be done as follows. For any node ν in the dual graph G^{D}, divide the true nodes and ghost nodes in G_ν into four sets.

1) $S_1 = \{g_e^\nu | e \in S\}$.
2) $S_2 = \{g_e^\nu | e \notin S\}$.
3) $S_3 = \{t_e^\nu | e \notin S\}$.
4) $S_4 = \{t_e^\nu | e \in S\}$.

Let the cardinality of S_1, S_2, S_3, and S_4 be a, b, c, and d, respectively. We need to prove that there is a perfect matching solution in which the a ghost nodes in S_1 and the c true nodes in S_3 are matched within G_ν, and the remaining nodes are matched outside G_ν. If ν is a conflict node, since S is a valid solution of the T-join problem, the number of edges $\in S$, $a + d$, is odd. The number of true nodes $(c + d)$ is odd by construction. Thus, $((a + d) + (c + d)) = (a + c) + 2d$ is even. Hence, $(a + c)$ is even. Similarly, we can prove that $(a + c)$ is even when ν is a legal node in G^{D}. It is always possible to match an even number of nodes in a complete graph.

Since the edge weight in the dual graph is always assigned to its corresponding ghost node, we only need to consider the nodes in S_1 and S_2 (true nodes in S_3 and S_4 have corresponding ghost nodes in other gadgets). Among them, only the nodes

[9]For example, edge 3 from node b to c means that edge 3 is assigned to node c, and it appears as t_3^c in G_c and g_3^b in G_b.

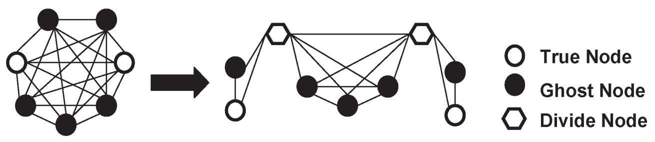


Fig. 7. Decompose a complete gadget with divide nodes.

in S_1 are matched within gadgets, and their weights are included in the matching solution. In other words, the ghost node weight is included in the matching solution if and only if its corresponding dual edge is in the T-join solution. Therefore, the matching solution has the same weight as the T-join solution. ■

The perfect matching problem can be optimally solved in polynomial time. In our implementation, we integrate the code of Cook and Rohe [13].

It should be noted that using the proposed conflict cycle graph and the T-join formulation implies the classification of a face which does not rely on its edge number. Therefore, further simplifications can be done on the dual graph when it is converted to the gadget graph that is input to the perfect matching problem.

For instance, feature edges are only needed to classify the faces as conflict faces or legal faces and can be dropped during the dual graph construction if they cannot be picked by the phase conflict detection algorithm (in the next section, we discuss why this might be the case). This simplification results in a further reduction of the number of nodes and edges in the gadget graph without affecting the correctness of the above reductions. However, this simplification could not be done with previous bipartite formulation in [6], [7], and [2], which in turn resulted in the increased complexity of their constructed gadget graphs.

C. Gadget Decomposition With Divide Nodes

For a large gadget of n nodes, the number of edges is $O(n^2)$ since a gadget is a complete graph. Therefore, we propose a method to decompose a large gadget into a set of small complete gadgets with **divide nodes** to reduce the edge number.

The gadget graph construction with divide node consists of the following steps.

1) Dual edge assignment and gadget node construction. These two steps are the same as Steps 1 and 2 of the construction without divide nodes.
2) Gadget construction with divide nodes. The nodes in each gadget G_ν are divided into $2i+1 (i \geq 0)$ subsets $G_{\nu,j}$ $(j = 1,\ldots,2i+1)$, which are linked with $2i$ divide nodes with zero weight. All nodes in $G_{\nu,j}$ and $G_{\nu,j+1}$ are connected to divide nodes $d_{\nu,j}$, $(j = 1,\ldots,2i)$. Each pair of the neighboring divide nodes $\{d_{\nu,j}, d_{\nu,j+1}\}$ $(j = 1,\ldots,2i-1)$ is connected. The weight of any edge in G_ν is the total weight of its two nodes. Fig. 7 shows one example of dividing a big gadget into three small subsets with divide nodes.

The edge numbers can be greatly reduced with divide nodes for large gadgets. For example, a complete gadget of n nodes has $n(n-1)/2$ edges. If the nodes are divided into subsets with divide nodes such that each subset has at most three nodes and at most one subset has less than three nodes, the edge number is at most $10(n+2)/3$. Therefore, the edge number is reduced from $O(n^2)$ to $O(n)$ while the node number is still $O(n)$, which leads to a reduction in perfect matching runtime for large gadgets.

The constructed gadget with divide nodes has the following important property.

Lemma 3: For any subset $S_1 \subseteq \{G_{\nu,1} \bigcup G_{\nu,2},\ldots, \bigcup G_{\nu,2i}\}$ $(i \geq 1)$, there is a perfect matching solution to match all the nodes in S_1 and the divide nodes $d_{\nu,1},\ldots,d_{\nu,2i-1}$.

Proof: We prove this lemma inductively. For $i = 1$, there are three possible cases.

1) Both $G_{\nu,1}$ and $G_{\nu,2}$ have even nodes $\in S_1$. We can match those nodes within $G_{\nu,1}$ and $G_{\nu,2}$ since we can always match even number of nodes within a complete graph. Then, $d_{\nu,1}$ can match $d_{\nu,2}$.
2) Both $G_{\nu,1}$ and $G_{\nu,2}$ have odd nodes $\in S_1$. We can match one node in $G_{\nu,1} \cup S_1$ with $d_{\nu,1}$ and one node in $G_{\nu,2} \cup S_1$ with $d_{\nu,2}$. Other nodes can be matched within $G_{\nu,1}$ and $G_{\nu,2}$.
3) Either $G_{\nu,1}$ or $G_{\nu,2}$ has odd nodes $\in S_1$. $d_{\nu,1}$ can match one node in S_1, which is in the subset with odd nodes $\in S_1$. Then, the other nodes $\in S_1$ are matched within the subsets.

Therefore, the lemma is true for $i = 1$. Suppose the lemma is true for $i = k$. For $i = k+1$: If $d_{\nu,2k}$ is matched, then we only need to match the nodes to be matched in $G_{\nu,2k+1}$ and $G_{\nu,2k+2}$ and $d_{\nu,2k+1}$, which is the same case as $i = 1$ if we view $G_{\nu,2k+1}$ as $G_{\nu,1}$, $G_{\nu,2k+2}$ as $G_{\nu,2}$ and $d_{\nu,2k+1}$ as $d_{\nu,1}$.

If $d_{\nu,2k}$ is not matched, there are four cases.

1) Both $G_{\nu,2k+1}$ and $G_{\nu,2k+2}$ have even nodes $\in S_1$. We can matched those nodes within $G_{\nu,2k}$ and $G_{\nu,2k+1}$, and match $d_{\nu,2k}$ with $d_{\nu,2k+1}$.
2) Both $G_{\nu,2k+1}$ and $G_{\nu,2k+2}$ have odd nodes $\in S_1$. We can match one node of $G_{\nu,2k+1} \cup S_1$ with $d_{\nu,2k}$ and one node of $G_{\nu,2k+2} \cup S_1$ with $d_{\nu,2k+1}$. Other nodes $\in S_1$ can be matched within $G_{\nu,2k+1}$ and $G_{\nu,2k+2}$.
3) $G_{\nu,2k+1}$ has odd nodes $\in S_1$ and $G_{\nu,2k+2}$ has even nodes $\in S_1$. We can match one node of $G_{\nu,2k+1} \cup S_1$ with $d_{\nu,2k}$ and match $d_{\nu,2k+1}$ with $d_{\nu,2k+2}$. The other nodes are matched within the subsets.
4) $G_{\nu,2k+1}$ has even nodes $\in S_1$ and $G_{\nu,2k+2}$ has odd nodes $\in S_1$. We can match one node of $G_{\nu,2k+2} \cup S_1$ with $d_{\nu,2k+2}$ and match $d_{\nu,2k}$ with $d_{\nu,2k+1}$. The other nodes are matched within the subsets.

Therefore, the lemma is true for $i = k+1$. ■

Theorem 4: The T-join problem for a graph $G^{\mathrm{D}} = (V, E, w, T)$, where T denotes the conflict nodes and $V \setminus T$ denotes the legal nodes in V, can be reduced to a minimum-weighted perfect matching on the gadget graph with divide nodes $\mathcal{G} = (V', E', w')$.

Proof: $(\rightarrow)$ Mapping the perfect matching solution of the gadget graph with divide nodes $\mathcal{G}$ to a valid solution of the T-join problem on G^{D}: For any node $\nu \in G^{\mathrm{D}}$, let its gadget G_ν to be $2i+1$ subsets $G_{\nu,j}$ $j = 1,\ldots,2i+1$ connected with divide nodes in the gadget graph. The edges of node ν can be grouped into four sets.

1) $S_1 = \{e | g_e^\nu \text{ matched within } G_\nu\}$.
2) $S_2 = \{e | g_e^\nu \text{ not matched within } G_\nu\} = \{e | t_e^{\nu'} \text{ matched within } G_{\nu'}\}$.
3) $S_3 = \{e | t_e^\nu \text{ matched within } G_\nu\}$.
4) $S_4 = \{e | t_e^\nu \text{ not matched within } G_\nu\} = \{e | g_e^{\nu'} \text{ matched within } G_{\nu'}\}$.

We need to prove that the set $S = S_1 \bigcup S_4$ thus constructed is a valid solution to the T-join problem. Let the cardinality of S_1, S_2, S_3, and S_4 be a, b, c, and d, respectively. In any perfect matching solution, the number of nodes matched within G_ν (including $2i$ divide nodes), $(a + c + 2i)$, is even. If ν is a conflict node, the number of true nodes in G_ν, $c + d$, is odd by construction and $(a + c + 2i) + (c + d) = (a + d) + 2c + 2i$ is odd. Therefore, the number of edges in S, $a + d$, is odd. Similarly, if ν is a legal node, the number of edges in S is even. Therefore, the solution S is a valid solution of the T-join problem.

Since the weight of any edge e in the dual graph is always assigned to its corresponding ghost node g_e^ν, the total weight of the edges in the T-join solution, $S = S_1 \bigcup S_4$, is equal to the total weight of all ghost nodes matched within gadgets.

On the other hand, the total weight of the matching solution, i.e., = the total weight of the matched edges the total weight of the matched edges within gadgets (since the weights of edges incident to dummy nodes are all zero), = the total weight of all nodes matched within gadgets (since the edge weight is the total weight of its two nodes), and hence = the total weight of all ghost nodes matched within gadgets (since the weights of all true nodes are zero). Therefore, the total weight remains the same during the mapping.

$(\longleftarrow)$ Mapping a solution S of the T-join problem to a solution of the perfect matching problem of $\mathcal{G}$ can be done as follows. For any node $\nu \in G^{\mathrm{D}}$ whose gadget is G_ν, which includes $2i+1$ subsets $G_{\nu,j}$ $j = 1, \ldots, 2i + 1$ linked with divide nodes, the true nodes and ghost nodes can be grouped into four sets.

1) $S_1 = \{g_e^\nu | e \in S\}$.
2) $S_2 = \{g_e^\nu | e \notin S\}$.
3) $S_3 = \{t_e^\nu | e \notin S\}$.
4) $S_4 = \{t_e^\nu | e \in S\}$.

Let the cardinality of S_1, S_2, S_3, and S_4 be a, b, c, and d, respectively. We need to prove that there is a perfect matching solution in which all the divide nodes, the a ghost nodes in S_1 and the c true nodes in S_3 are matched within G_ν and the remaining nodes are matched outside G_ν.

If ν is a conflict node, since S is a valid solution of the T-join problem, the number of edges $\in S$, $a + d$, is odd. The number of true nodes $(c + d)$ is odd by construction. Thus, $((a + d) + (c + d)) = (a + c) + 2d$ is even. Hence, the number of true nodes and ghost nodes to be matched within G_ν, $(a + c)$, is even. Since all the $2i$ divide nodes should be matched within G_ν, the total number of nodes to be matched, $(a + c) + 2i$, is even. According to Lemma 3, all nodes to be matched in $G_{\nu,1}, \ldots, G_{\nu,2i}$ and the divide nodes $d_{\nu,1}, \ldots, d_{\nu,2i-1}$ can be matched in a matching solution. The remaining even number of nodes is located in a complete graph $G_{\nu,2i} \bigcup \{d_{\nu,2i}\}$. It is always possible to match an even number of nodes in a complete graph. Similarly, we can prove that there is a perfect solution when ν is a legal node in G^{D}.

Since the edge weight in the dual graph is always assigned to its corresponding ghost node, we only need to consider the nodes in S_1 and S_2 (true nodes in S_3 and S_4 have corresponding ghost nodes in other gadgets). Among them, only the nodes in S_1 are matched within gadget and their weights are included in the matching solution. In other words, the ghost node weight is included in the matching solution if and only if its corresponding dual edge is in the T-join solution. Therefore, the matching solution has the same weight as the T-join solution. ■

IV. Layout Modification

The primary task of AAPSM-related layout modification is to correct the phase conflicts that the conflict detection algorithm selected in the previous step. Phase conflicts can be corrected either by adding a space between shifters corresponding to a conflict (equivalent to correcting a conflict edge) or by widening critical features (equivalent to correcting a feature edge). However, widening critical features may introduce significant timing problems. Hence, in this paper, we only focus on phase conflicts that can be solved by increasing the spacing between features such that the corresponding shifters are separated by the required shifter spacing. However, merely increasing the spacing between the shifters corresponding to the phase conflict may cause design-rule-checking (DRC) violations as well as introduce new phase conflicts as the relative locations of the neighboring features may change. The work presented in [2] solved this problem by adding end-to-end spaces throughout the layout. The spaces are inserted such that only the length of the polyinterconnect is increased. This technique could only be applied to standard cells and macro blocks with a low density of phase conflicts. Experiments indicate that this method when applied directly to standard-cell blocks can cause large increases in area.

Our layout modification algorithm exploits the fact that standard-cell blocks can be naturally partitioned into rows and that the phase conflicts in each row can be solved independently without introducing any DRC errors. The overall flow of the algorithm is presented in Fig. 8. The algorithm consists of the following steps.

1) First the standard-cell block is partitioned into rows and its constituent cells. The rows are identified by locations of the power grid lines.
2) Next, the phase conflicts that are strictly between features of a cell are corrected by adding a minimal number of end-to-end spaces in the cell as illustrated in Fig. 9. In this scheme, horizontal and vertical spaces of variable width are added along a cut line throughout the cell to correct the chosen AAPSM conflicts. As shown in Fig. 9(a), the space insertion is equal to moving all features on the right side of the cut line to the right by a distance B. For any feature across the cut line, if it is not connected with the features on the right side of the cut line [Fig. 9(a)], it will not be moved; otherwise, if it is not connected with the features on the left side of the cut line [Fig. 9(b)], it will be

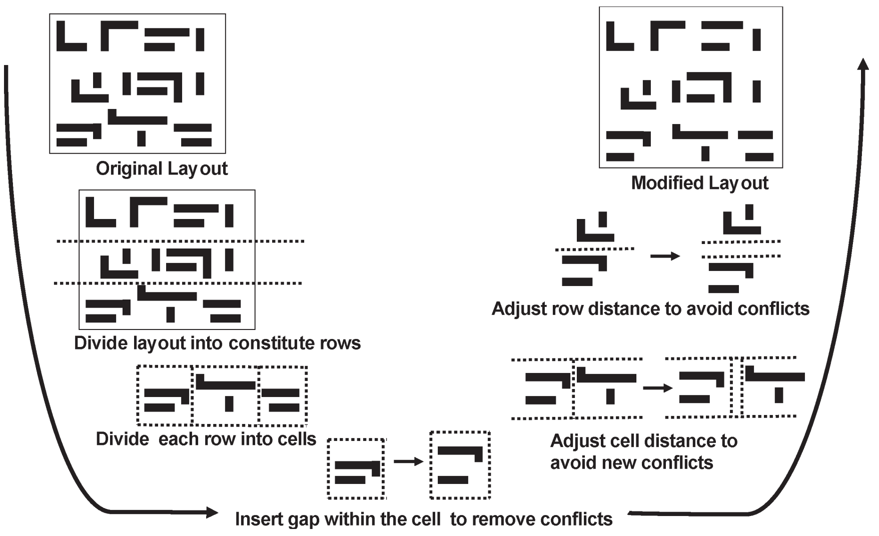

Original Layout
Divide layout into constitute rows
Divide each row into cells
Insert gap within the cell to remove conflicts
Adjust cell distance to avoid new conflicts
Adjust row distance to avoid conflicts
Modified Layout

Fig. 8. Details of layout modification algorithm.

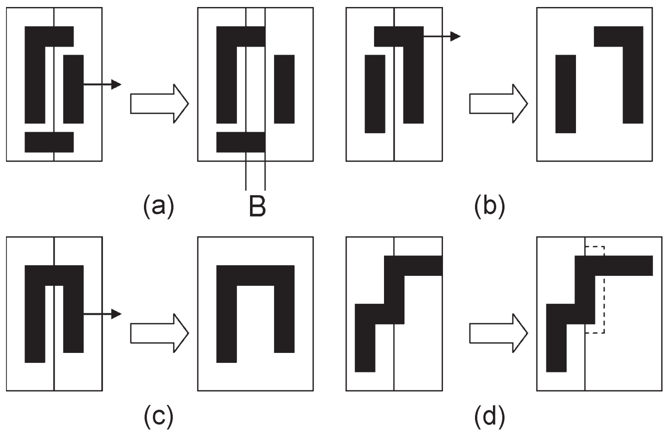


Fig. 9. Layout modification with vertical space insertion.

moved to the right by a distance B; if it is connected with the features on both sides [Fig. 9(c) and (d)], the spaces are added such that only the gate widths are increased but the gate lengths remain the same. Therefore, the straight cut line will be replaced by the dashed line as shown in Fig. 9(d). This prevents any major timing problems after a layout modification.[10]

3) The modified cells in a row are now assembled such that no phase conflicts exist between any two features of adjacent neighboring cells. The height of each row is equal to the height of the tallest cell, and the width is equal to the sum of the widths of the standard cell plus the widths of the inserted spaces between the standard cells. The spaces that are occupied by filler cells are made available at this step to avoid any unnecessary area increase.

[10]In practice, the timing impact due to layout modification is negligible since: 1) the cell is small; 2) a minimum-weighted set-covering problem (similar to the one proposed in [2]) is used to determine cut lines to avoid most cases like Fig. 9(c) and (d); and 3) if the cases like Fig. 9(c) and (d) cannot be avoided to correct one conflict, its corresponding edge in the conflict cycle graph will be assigned a large weight to prevent the edge from being selected for correction.

4) The final step consists of assembling the modified rows. Here, again, horizontal space is added only as needed. Space is added only if there is an existing conflict between features of cells on adjacent rows, or if relative locations of the features in adjacent rows are changed from the original configuration (this can only happen if vertical space is added at different locations on adjacent rows).

Hence, in this algorithm, end-to-end spaces are only added within a cell. The spaces between the cells and between the rows are smartly managed such that no phase conflicts remain or are introduced after the changes to the individual cells. This results in much smaller area increases for correcting phase conflicts when compared to the method in [2]. According to our layout modification algorithm, the weighting scheme of the conflict cycle graph is as follows. The weights of feature edges are assigned as infinity since we do not permit feature widening. The vertical conflicts (i.e., conflicts that can be solved by adding vertical end-to-end spaces) are assigned a much lower weight than horizontal conflicts (i.e., conflicts that can be solved by adding horizontal end-to-end spaces), since it is less disruptive to increase the width of the standard cells than their height. In our implementation, the weights of conflict edges of vertical conflicts are assigned as the width of the spacing to be added to solve the conflict, i.e., B in Fig. 9, to reflect the area increase due to the layout modification; the weights of conflict edges of vertical conflicts are assigned as $10\times$ spacing width. The weights of conflicts which may result in increased gate width are assigned as $50\times$ spacing width.

Fig. 10 compares our layout modification algorithm with the one presented in [2] on a hypothetical example. The layout is a square and is composed of five rows of standard cells. Let l denote the length of each side of the layout. The shaded rectangles denote the spaces added in the layout and the bold dark lines are used to represent the locations of the phase conflicts being corrected. Let w denote the width of horizontal and vertical spaces added (assumed to be the same for simplicity). The area increase with the scheme in [2] is $11lw$, whereas the area increase with our scheme is only $6lw$.

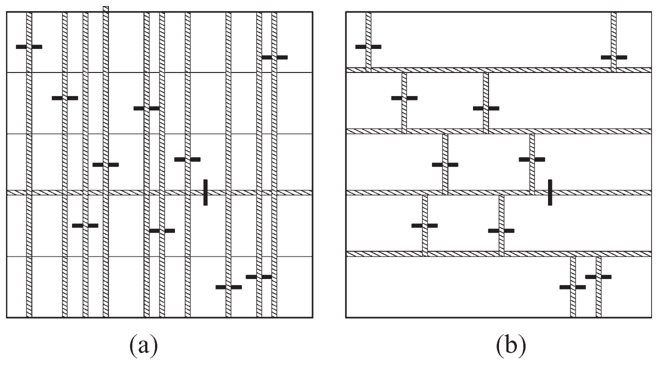


Fig. 10. Comparing the area increases produced by the layout modification scheme in [2] with the proposed scheme. (a) Total Area Increase $= l * 10 * w$. (b) Total Area Increase $= l/5 * 10 * w + 4 * l * w = 6 * l * w$.

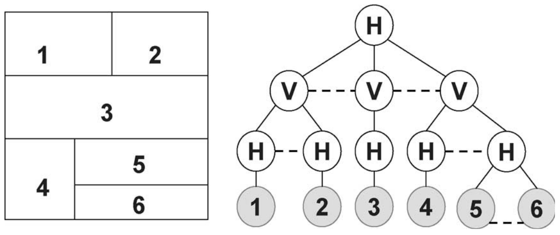


Fig. 11. Hierarchical layout and its partition tree.

The presented algorithm can be applied as an additional processing step during postplacement optimizations. The inserted spaces are integer multiples of the M1 routing pitch, and, hence, the modifications introduced by the proposed flow do not introduce any additional complications for the router. This is also a difference from the method in [2] that does not match the inserted spaces to the routing pitch. This solution needs to be applied even if the placement is done with AAPSM-compliant standard cells, i.e., cells that have no phase conflicts. This is because the phase conflicts can exist between features of neighboring cells. The only difference is that Step 2 of the proposed layout modification algorithm may be omitted.

Our proposed algorithm can also be easily extended to solve the phase conflicts in slicing hierarchical layouts, i.e., the layouts whose floorplan can be represented by slicing trees. As shown in Fig. 11, a slicing hierarchical layout can be represented using the partition tree, where each leaf node represents a layout region. The H (or V) node represents a region which is partitioned into several child regions using horizontal (or vertical) cut lines; there is a dashed line between any two neighboring child nodes which represents the cut line. The layout modification can be solved using a bottom-up algorithm. First, the phase conflicts are corrected within each leaf node by inserting end-to-end spaces. Then, for each upper level, new phase conflicts are avoided by inserting gaps along the cut lines. The algorithm shown in Fig. 8 is the special case when the input layout can be represented as a two-level tree.

V. Experimental Results

This section presents the experiments we conducted to test the benefits of the proposed ideas. All our examples are 90-nm designs, and assume typical values of threshold width for critical features, shifter dimensions, and shifter spacing. This paper focuses mainly on the phase conflicts that can be solved by increasing the spacing between features in the layout. Thus, phase conflicts caused by T-shapes are not handled. These can be corrected by feature widening or mask splitting [8]. Phase conflicts caused by line-end conflicts between the neighboring features can be detected and corrected which are not considered as they can be efficiently detected and corrected using additional DRC checks during layout generation [15].

A. Phase Conflict Detection Results

Table I compares the runtime and the quality of results (number of edges deleted, or in other words, number of phase conflicts selected for correction) of the proposed flow with other state-of-the-art approaches. The flow presented in [2] is our main comparison point since their results are best in terms of the number of phase conflicts chosen for comparison and runtime, when compared to other state-of-the-art approaches [6], [7]. Columns 1 and 2 give the design names and design statistics (like number of polygons and number of shifter overlaps, respectively). The results obtained after applying the flow in [2] are grouped under the columns "Flow in [2]." The results obtained using our phase conflict detection method are grouped under "Proposed Flow." Columns 5 and 9 compare the runtime of the flow in [2] and our proposed flow, respectively.[11] As can be seen, our runtimes are significantly better than those obtained using the flow in [2] with an average improvement of 5.9×. This can be primarily attributed to the significant reduction in the number of edges in the conflict cycle graph compared to the phase conflict graph used in [2] and the removal of undeletable edges (in our case, feature edges) during intermediate graph constructions. This is reflected in the number of nodes and edges of the gadget graph constructed during perfect matching. The gadget graphs constructed in our flow are significantly smaller than the ones constructed in [2]. While the examples presented are not very large, we believe the same trend of speedups should also be present in much larger examples. The limitations of the current code prevented us from testing our idea on larger examples.

The table also shows that the quality of our results (in terms on number of phase conflicts chosen for correction) is also better than the results obtained using the method in [2] (please look at Column labeled "# Conflicts" under the subgroup "Flow in [2]" for the results obtained using the method in [2] and Column labeled "# Conflicts" under subgroup "Proposed Flow" for the results obtained with our method). This improvement is primarily due to the fact that the number of edges deleted during planarization of the conflict cycle graph (second step in Fig. 2) is smaller than the edges deleted during the corresponding planarization step of the phase conflict graph [2]. Hence, the optimal algorithm can be applied to a larger subgraph of the original graph.

[11]It should be noted that only the time spent in solving the perfect matching problem is reported in both cases as this is the most compute-intensive portion of the algorithm. The gadget graph construction was also sped-up by 2× using the new graph, but the perfect matching times has a greater contribution to the total run time.

TABLE I
PHASE CONFLICT DETECTION RESULTS. EXPERIMENTS WERE RUN ON A 4 × 400-MHz ULTRA-SPARC II WITH 4.0 GB OF RAM

		Flow in [2]				Proposed Flow			
Design	# Plgns/#Ovlps	#edges	#nodes/#edges GG	CPU(s)	# Conflicts	#edges	#nodes/#edges GG	CPU(s)	# Conflicts
1	10274/24580	98347	75086/242093	2.53	938	66875	25198/46346	0.38	910
2	13630/32257	112599	62480/203971	1.90	963	79672	20910/36403	0.22	946
3	21868/53749	182809	103912/338810	3.33	1558	128836	34806/61227	0.55	1534
4	20425/50059	173319	98834/320499	3.18	1678	121546	33266/57705	0.48	1664
5	25784/63760	216691	122324/397999	3.87	1854	152655	40614/70853	0.60	1839
6	48787/157668	484999	355760/1147057	12.17	6330	325769	8077/45208	2.78	5989
7	44121/142707	436297	339538/1084677	12.30	5010	295001	112152/207784	2.47	4865
8	72101/237557	729980	567532/1817272	20.05	10275	489922	189986/351007	4.23	9631
9	105882/376707	1133279	949064/299758	41.35	18148	757581	303408/565217	7.95	17463
10	159070/552767	1667581	1415620/4437522	66.40	27308	1115928	444082/829898	11.58	26349

TABLE II
LAYOUT MODIFICATION RESULTS FOR STANDARD-CELL BLOCKS

Design	Area	Conflict	Outside	% Area Inc.	% Area Inc. [2]
1	25173.96	937	61	1.7	18.1
2	16397.82	995	197	5.4	23.1
3	31416.21	1589	284	5.8	26.8
4	25715.23	1724	238	6.1	28.8
5	40409.68	1720	322	4.7	32.12
6	61705.52	6257	770	5.8	57.47
7	58414.06	5100	586	6.1	59.1
8	94178.09	10141	512	7.3	80.2
9	148231.77	18657	2672	9.1	> 100.0
10	249210.41	28121	4224	9.0	> 100.0

B. *Phase Conflict Correction Results*

Table II reports the results of using the proposed layout modification scheme for correcting the phase conflicts chosen by the detection step on the same layouts. Column "Area" reports the area of the designs in square micrometers. Column "Conflict" specifies the number of phase conflicts selected by the detection algorithm for each design (the numbers are slightly different from the ones in Table I due to the use of different weighting schemes). Column "Outside" reflects the number of phase conflicts that is selected for correction and occurs between features of neighboring cells. As can be seen, it is a very small fraction of the total number of phase conflicts in any design. This strengthens our view that it is too conservative to leave a blank space around all the cells or force the boundary features of each cell to have the same phase and could cause large area increases. The fifth column reports the percentage area increase for these layouts as a result of the added spaces. The area increase for these layouts ranges from 1.7% to 9.1%, with an average increase of 6.1%. The area increase goes up slightly with the size of the test cases. For comparison, the layout increase caused by the method in [2] is also reported in the last column. As can be seen, the area increases caused by the method in [2] are very large.

VI. CONCLUSION

A new theory for bright-field phase conflict detection was presented in this paper. The proposed method greatly simplified the graph constructed from the layout which resulted in a substantial reduction in its edge count. Unlike previous constructions, the proposed graph does not equate the phase assignability of its corresponding layout to its bipartition. Hence, a new property of the graph called conflict cycles was introduced, and an optimal algorithm for removing conflict cycles in embedded planar graphs was presented. The algorithm was also generalized so that a minimal solution could be obtained for nonplanar graphs. Supporting experimental results were also presented that illustrated huge improvements in the runtime in the process while maintaining the same quality of results (in terms of number of phase conflicts chosen for correction) as the best available previous work in this area.

A novel layout modification algorithm for standard-cell blocks was also presented. Experimental results confirm that the new method produces much smaller increases in area than the previous work in this area. The small area increases make it suitable for use in a true industrial flow as a postplacement optimization step. The current algorithm does not assume that the standard cells used in the placement are phase assignable. However, the proposed method can also be applied to a placement done with AAPSM-complaint cells and will produce much smaller area increases, when compared to other methods being considered for building phase-assignable placements. The proposed method has to be extended to allow a feature widening for certain phase conflicts that cannot be solved by increasing the spacing between features. It will also be desirable to integrate the layout modification method with a timing engine since the layout modifications produced by the method can cause some timing violations.

REFERENCES

[1] L. W. Liebmann, T. H. Newman, R. A. Ferguson, R. M. Martino, A. F. Molless, M. O. Neisser, and J. T. Weed, "A comprehensive evaluation of major phase shift mask technologies for isolated gate structures in logic designs," *Proc. SPIE*, vol. 2197, pp. 612–623, 1994.

[2] C. Chiang, A. Kahng, S. Sinha, X. Xu, and A. Zelikovsky, "Bright-field AAPSM conflict detection and correction," in *Proc. DATE*, 2005, pp. 908–913.

[3] C. Chiang, A. Kahng, S. Sinha, and X. Xu, "Fast and efficient phase conflict detection and correction in standard-cell layouts," in *Proc. ICCAD*, 2005, pp. 149–156.

[4] A. Moniwa, T. Terasawa, N. Hasegawa, and S. Okazaki, "Algorithms for phase-shift mask design with priority on shifter placement," *Jpn. J. Appl. Phys.*, vol. 34, no. 12B, pt. 1, pp. 6584–6589, 1993.

[5] K. Ooi, S. Hara, and K. Koyama, "Computer-aided design software for designing phase-shift masks," *Jpn. J. Appl. Phys.*, vol. 32, no. 12B, pp. 5887–5891, Dec. 1993.

[6] P. Berman, A. B. Kahng, S. Mantik, I. L. Markov, and A. Zelikovsky, "Optimal phase conflict removal for layout of dark field alternating phase shifting masks," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 19, no. 2, pp. 1265–1278, 2000.

[7] A. B. Kahng, S. Vaya, and A. Zelikovsky, "New graph bipartizations for double-exposure, bright field alternating phase-shift mask layout," in *Proc. ASP-DAC*, 2001, pp. 133–138.
[8] C. Pierrat, F. A. Driessen, and G. Vandenberghe, "Full phase-shifting methodology for 65 nm node lithography," *Proc. SPIE*, vol. 5040, pp. 282–293, 2003.
[9] K. Ooi, K. Koyama, and M. Kiryu, "Method of designing phase-shifting masks utilizing a compactor," *Jpn. J. Appl. Phys.*, vol. 33, no. 12B, pp. 6774–6778, Dec. 1994.
[10] A. Moniwa, T. Terasawa, K. Nakajo, J. Sakemi, and S. Okazaki, "Heuristic method for phase conflict minimization in automatic phase-shift mask design," *Jpn. J. Appl. Phys.*, vol. 34, no. 12B, pp. 6584–6589, Dec. 1995.
[11] K. Cao, J. Hu, and M. Cheng, "Layout modification for library cell Alt-PSM composability," *Proc. SPIE*, vol. 5379, pp. 253–259, May 2004.
[12] W. J. Cook, W. H. Cunningham, W. R. Pulleyblank, and A. Shrijver, *Combinatorial Optimization*. New York: Willey Inter-Science, 1998.
[13] W. Cook and A. Rohe, *Computing Minimum-Weight Perfect Matchings*, Aug. 1998. [Online]. Available: http://www.or.unibonn.de/home/rohe/matching.html
[14] L. Liebmann, J. Lund, F. L. Heng, and I. Graur, "Enabling alternating phase shifted mask designs for a full logic gate level: Design rules and design rule checking," in *Proc. DAC*, 2001, pp. 78–84.
[15] P. Ghosh, C. Kang, M. Sanie, and J. Huckabay, "PsmLint: Bringing Altpsm benefits to the IC design stage," *Proc. SPIE*, vol. 5042, pp. 314–325, 2003.

Charles Chiang received the B.S. degree from the Department of Computer Science, New Mexico State University, Las Cruces, in 1986, and the M.S. and Ph.D. degrees from the Department of Electrical Engineering and Computer Science, Northwestern University, Evanston, IL, in 1988, and 1991, respectively.

After working with IBM and other EDA companies for 10 years, he joined the Advanced Technology Group at Synopsys, Inc., Mountain View, CA, in 2001. His research interests include routing, placement, floorplan, and signal integrity. His main research focus is now on the design for manufacturability (DFM). He has published approximately 40 papers and has filed nine U.S. patents.

Dr. Chiang received the Superior Design Recognition Award and the ADAL Award from IBM Rochester in 1993 and 1994, respectively. He has served on the technical committee of ICCAD from 2004 to 2006, on that of Field Programming Logic (FPL) from 2002 to 2003, as well as on the committee of ASP-DAC in 2005.

Andrew B. Kahng (A'89–M'03) received the A.B. in applied mathematics from Harvard College, Cambridge, Massachusetts, and the M.S. and Ph.D. degrees in computer science from the University of California (UC) San Diego, La Jolla, CA.

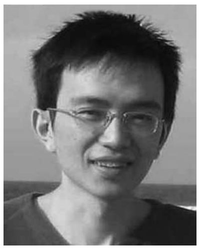
From 1989 to 2000, he was a member of the UCLA computer science faculty. He is a Professor of CSE and ECE with UC San Diego, La Jolla. He has published over 300 papers in the VLSI CAD literature. His research is mainly in physical design and performance analysis of VLSI as well as the VLSI design manufacturing interface. Other research interests include combinatorial and graph algorithms, and large-scale heuristic global optimization.

Dr. Kahng received four Best Paper Awards and an NSF Young Investigator Award. Since 1997, he has defined the physical design roadmap for the International Technology Roadmap for Semiconductors (ITRS), and from 2000 through 2003 chaired the U.S. and international working groups for Design technology for the ITRS. He has been active in the MARCO Gigascale Silicon Research Center since its inception. He was also the founding General Chair of the ACM/IEEE International Symposium on Physical Design, and cofounded the ACM Workshop on System-Level Interconnect Planning.

Subarnarekha Sinha received the B.S. degree from the Department of Electronics and Electrical Communication Engineering, Indian Institute of Technology, Kharagpur, in 1992 and the M.S. and Ph.D. degrees from the Department of Electrical Engineering and Computer Sciences, University of California, Berkeley, in 1994, and 1996, respectively.

She is currently with the Advanced Technology Group at Synopsys, Inc., Mountain View, CA. Her research interests include logic synthesis, physical synthesis, and design for manufacturability (DFM).

Dr. Sinha was the recipient of the Institute Silver Medal for outstanding academic performance in 1992. She is currently serving on the technical program committee of IWLS and FPL.



Xu Xu (S'01) received the B.S. degree from Special Class for Gifted Young, University of Science and Technology of China, Hefei, China, in 1998 and the Ph.D. degree from the University of California, San Diego, CA in 2006.

He was with Ammocore Technology, Inc., Santa Clara, CA, in 2002. He was with Synopsys, Inc., Mountain View, CA, in 2004. In 2006, he joined Blaze DFM, Sunnyvale, CA. He has authored more than 20 refereed publications on VLSI physical design, DNA array design, and IC manufacturing cost minimization.

Alexander Z. Zelikovsky received the Ph.D. degree in computer science from the Institute of Mathematics, Belorussian Academy of Sciences, Minsk, Belarus, in 1989.

From 1989 to 1995, he worked with the Institute of Mathematics in Kishinev, Moldova. Between 1992 and 1995, he visited Bonn University and the Institut fur Informatik in Saarbrueken, Germany. He was a Research Scientist with University of Virginia from 1995 to 1997, and a Postdoctoral Scholar with UCLA from 1997 to 1998. In 1999, he joined the Computer Science Department of Georgia State University, University Plaza, Atlanta, where he is an Associate Professor. He has authored of more than 120 refereed publications. His research interests include VLSI physical layout design, *ad hoc* wireless networks, discrete and approximation algorithms, combinatorial optimization, and computational biology.