

Fast and Efficient Bright-Field AAPSM Conflict Detection and Correction

C. Chiang A. B. Kahng S. Sinha X. Xu A. Z. Zelikovsky

Abstract—*Alternating-Aperture Phase Shift Masking (AAPSM), a form of strong Resolution Enhancement Technology (RET), 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; (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 [2]. An average runtime speedup of 5.9x is achieved using the new flow. A new layout modification scheme suited for correcting phase conflicts in large standard-cell blocks is also proposed. Our experiments show that the percentage area increase for making standard-cell blocks phase-assignable ranges

from 1.7-9.1%. Alternating-Aperture Phase Shift Masking (AAPSM), a form of strong Resolution Enhancement Technology (RET), 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; (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 [2]. An average runtime speedup of 5.9x is achieved using the new flow. A new layout modification scheme suited for correcting phase conflicts in large standard-cell blocks is also proposed. Our experiments show that the percentage area increase for making standard-cell blocks phase-assignable ranges from 1.7-9.1%.

C. Chiang and S. Sinha are with Synopsys. E-mail: {Charles.Chiang, Subarna.Sinha}@synopsys.com.

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

X. Xu is with the Department of Computer Science and Engineering, University of California at San Diego, La Jolla, CA 92093-0114. E-mail: xuxu@cs.ucsd.edu.

A. Z. Zelikovsky is with the Computer Science Department, Georgia State University, University Plaza, Atlanta, Georgia 30303. E-mail: alexz@cs.gsu.edu.

I. INTRODUCTION

As advanced technologies in wafer manufacturing push the patterning processes toward lower k_1 sub-wavelength printing, reticle based resolution enhancement techniques (RET) have played a critical enabling role. Alternating-Aperture Phase Shift Masking 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.

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 above conditions in a phase assignment solution in which each shifter is assigned a phase. Figure 1 illustrates an example where the above 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 as follows:

- A new and computationally efficient algorithm for detecting phase conflicts,

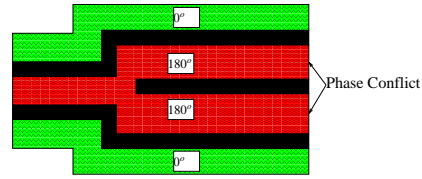


Fig. 1. Example of incorrect phase assignment.

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 5.9x, while maintaining the same quality of results as the best results available today [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.

The paper is organized as follows: Section 2 briefly reviews the previous work in phase conflict detection and correction. In Section 3, we provide a detailed discussion of the new theory of the proposed phase conflict detection flow. Section 4 discusses our new conflict correction algorithm. Experimental results are presented in Section 5. We end with conclusions and directions for future work in Section 6.

¹*T-shaped phase conflicts* and other local phase conflicts can be easily detected with simple design rule checking and corrected with phase splitting [7], feature widening [13] or cell re-design. 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.

II. PREVIOUS WORK

The phase conflict detection problem is addressed in [2] [3] [4] [5] [6]. 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, conflict detection would involve identifying a set of edges such that the modified graph obtained after deleting the edges is bipartite. The work in [5][6] 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 graphs² 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 [6]. 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 sub-graph of the given graph, applies a computationally efficient version of the optimal bipartization algorithm [5] on the planar sub-graph 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 [7] split shifter regions whenever two shifters of opposite phases overlap to avoid 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 spacing between features or widening critical features [3] [4] [9] [8] [5] [2]. Most of these works focus on dark-field AAPSM³. The first layout modification scheme for correcting bright-field phase conflicts is presented in [2].⁴ The key idea in this work 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 neighboring cells or introduce an additional requirement that all the boundary elements should have the same phase [10]. 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 Figure 2, the proposed conflict detection flow is presented below:

- 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

³In 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.

⁴Although the work in [6] proposes the conflict detection methods for bright-field phase conflicts based on feature widening, it neglects the layout modification problem.

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

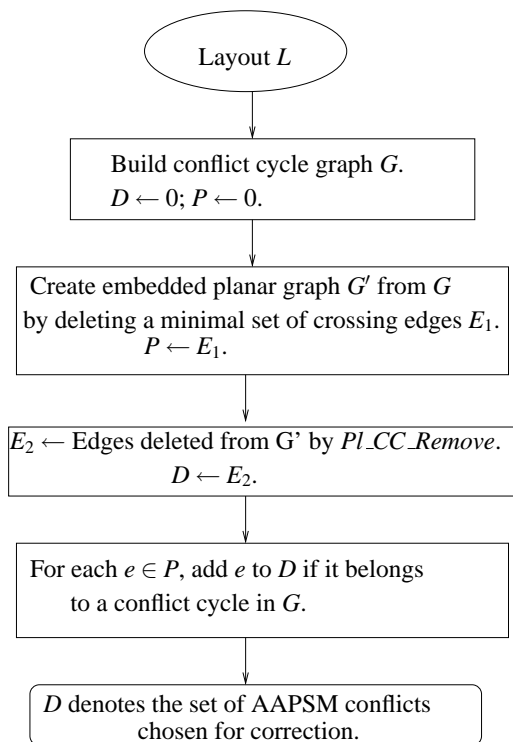


Fig. 2. Phase Conflict Detection Flow.

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 Remove 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 conflict. This is accomplished by

2-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 sub-graphs 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 speed-ups of the phase conflict detection algorithm. Experimental results on representative examples show average speedups of 5.9x 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 as follows:

- Representation of the layout as a conflict cycle graph and development of its relationship to phase-assignability of the layout.
- 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.
- 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 shifters⁵ 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.

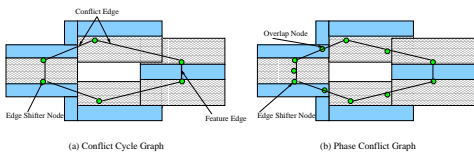


Fig. 3. (a) Conflict Cycle Graph. (b) Phase Conflict Graph.

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

Figure 3(a) shows an example of a conflict cycle graph for the layout shown earlier in Figure 1. The conflict cycle graph has 6 nodes and 6 edges. By comparison, the phase conflict graph (shown in Figure 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 phase-assignability of its corresponding layout to bipartiteness. So, a conflict cycle graph may be bipartite even if its corresponding layout is not phase-assignable. For instance, the layout in Figure 3(a) is not phase-assignable, even though its corresponding conflict cycle graph is bipartite. Thus, a new criterion is needed for detecting 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 some intermediate graph constructions, thereby producing more speed-ups.

The general phase assignment algorithm is shown in Figure 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

| |
|---|
| Input: Conflict cycle graph G |
| Output: Phase assignment of G |
| <ol style="list-style-type: none"> 1. failed ← 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: <ol style="list-style-type: none"> (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. <p style="margin-left: 20px;">If the rules are violated, failed ← True</p> 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: <ol style="list-style-type: none"> (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.

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 $\{n_1, n_2, \dots, n_k, n_1\}$ 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 Figure 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 $C = \{n_0, \dots, n_1, n_2, \dots, n_0\}$. There are 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 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. \square

In order to make the layout phase assignable, it is necessary to remove all conflict cycles from the conflict cycle graph by deleting edges. The deleted edges directly correspond to 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 conflict cycle removal.

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

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 algorithm exists when the graph is an embedded planar graph. This optimal algorithm is referred to as `PL_CC_Remove` in Figure 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 $e' = \{n_1, n_2\}$ in G^D . A node $n \in G^D$ 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 $m_1 + m_2$. \square

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. \square

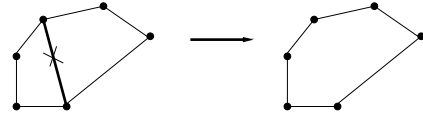


Fig. 5. Deleting all common edges (in this case, only one) results in a merged face.

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 Figure 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 $S = S_1 \cup S_2$.

We first want to prove that the cardinality of S_1 , $|S_1|$, is even. Let $r(f)$ denote the number of removed edges for each face $f \in S$. Since all removed edge 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 $f \in S_1$, $|S_1|$ 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. \square

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 $G^D = (V, E)$ such that: (a) an odd number of edges in S are incident on every conflict node $u \in V$; (b) an even number of edges in S are incident on every legal node $v \in V$.

This is similar in spirit to the **T-join** problem [11] 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 the following steps:

⁷Given 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.

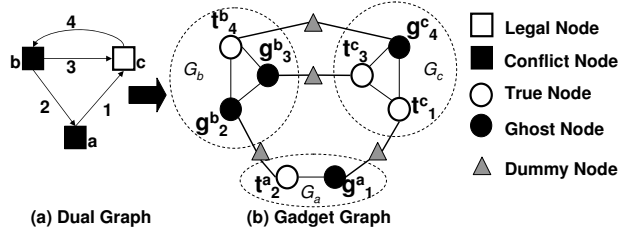


Fig. 6. Gadget graph construction from dual graph. The directions on the edges in (a) are used to signify the edge assignment.

1) **Dual Edge Assignment** Each edge e connecting v and v' in dual graph is assigned to v , v' or both. The assignment is done such that the following conditions are satisfied:⁸

- a) For each conflict node v , the number of true nodes in G_v is odd.
- b) For each legal node v , the number of true nodes in G_v is even.

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

2) **Gadget Node Construction** If a dual edge e connecting v and v' is assigned to v and not assigned to v' , it will appear as a *true node* t_e^v in G_v and a *ghost node* $g_e^{v'}$ in $G_{v'}$.⁹ As a result, each node v of degree k in the dual graph G^D becomes a gadget of k nodes in \mathcal{G} , which is denoted as G_v . The weight of $g_e^{v'}$ is $w(e)$ and the weight of t_e^v is 0. 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 0 weight edges. In any perfect matching solution for the gadget graph, *exactly one node in each pair of t_e^v and $g_e^{v'}$ will be matched within gadgets* since the other node will be

⁸To ensure that the conditions are satisfied, we use the edge assignment method in [5] 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_v which violates the parity requirement, it is always possible to turn a ghost node $g_e^{v'}$ into a true node t_e^v (i.e., assign the edge e to both v and v') to meet the parity requirement at the cost of increasing the node number of the gadget graph by one.

⁹For 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 .

matched with the dummy node.

- 3) **Complete Gadget Construction** The nodes in G_v are connected to each other by weighted edges to form a complete graph. The weight of any edge in G_v is the total weight of its two nodes. Figure 6 (b) shows the gadget graph constructed from the dual graph of Figure 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 $G^D = (V, E, w, T)$, 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 perfect matching solution of the gadget graph \mathcal{G} to a valid solution of the T-join problem on G^D : For any node v in the dual graph G^D , divide the edges of node v into four sets:

- $S_1 = \{e | g_e^v \text{ matched within } G_v\}$.
- $S_2 = \{e | g_e^v \text{ not matched within } G_v\} = \{e | t_e^{v'} \text{ matched within } G_{v'}\}$.
- $S_3 = \{e | t_e^{v'} \text{ matched within } G_{v'}\}$.
- $S_4 = \{e | t_e^{v'} \text{ not matched within } G_{v'}\} = \{e | g_e^{v'} \text{ matched within } G_{v'}\}$.

We need to prove that the set $S = S_1 \cup S_4$ 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_v , $(a+c)$, is even. If v is a conflict node, the number of true nodes in G_v , $c+d$, 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 v 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^v , the total weight of the edges in the T-join solution, $S = S_1 \cup S_4$, 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 0), = 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 0). 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 v in the dual graph G^D , divide the true nodes and ghost nodes in G_v into four sets:

- $S_1 = \{g_e^v | e \in S\}$.
- $S_2 = \{g_e^v | e \notin S\}$.
- $S_3 = \{t_e^{v'} | e \notin S\}$.
- $S_4 = \{t_e^{v'} | e \in S\}$.

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_v and the remaining nodes are matched outside G_v . If v 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 v 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 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 corre-

sponding dual edge is in the T-join solution. Therefore, the matching solution has the same weight as the T-join solution. \square

The perfect matching problem can be optimally solved in polynomial time. In our implementation, we integrate the code of Cook and Rohe [12].

It should be noted that using the proposed conflict cycle graph and the T-join formulation implies the classification of a face 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 [5] [6] [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 Step 1 and 2 of the construction without divide nodes.
- 2) **Gadget Construction with Divide Nodes** The nodes in each gadget G_v are divided into $2i + 1 (i \geq 0)$ subsets $G_{v,j} (j = 1 \dots 2i + 1)$, which are linked

with $2i$ divide nodes with 0 weight. All nodes in $G_{v,j}$ and $G_{v,j+1}$ are connected to divide nodes $d_{v,j}, (j = 1 \dots 2i)$. Each pair of neighboring divide nodes $\{d_{v,j}, d_{v,j+1}\} (j = 1 \dots 2i - 1)$, are connected. The weight of any edge in G_v is the total weight of its two nodes. Figure 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 $\frac{n(n-1)}{2}$ 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 $\frac{10(n+2)}{3}$. Therefore, the edge number is reduced from $O(n^2)$ to $O(n)$ while the node number is still $O(n)$, which leads to reduction in perfect matching runtime for large gadgets.

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

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

Proof: We prove this lemma in a recursive-way. For $i = 1$, there are three possible cases:

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

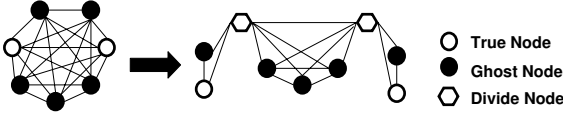


Fig. 7. Decompose a complete gadget with divide nodes.

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

If $d_{v,2k}$ is not matched, there are four cases:

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

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

Theorem 4: The T-join problem for a graph $G^D = (V, E, w, T)$, 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 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 $v \in G^D$, let its gadget G_v to be $2i + 1$ subsets $G_{v,j}$ $j = 1 \dots 2i + 1$ connected with divide nodes in the gadget graph. The edges of node v can be grouped into four sets:

- $S_1 = \{e | g_e^v \text{ matched within } G_v\}$.
- $S_2 = \{e | g_e^v \text{ not matched within } G_v\} = \{e | t_e^{v'} \text{ matched within } G_{v'}\}$.
- $S_3 = \{e | t_e^v \text{ matched within } G_v\}$.
- $S_4 = \{e | t_e^v \text{ not matched within } G_v\} = \{e | g_e^{v'} \text{ matched within } G_{v'}\}$.

We need to prove that the set $S = S_1 \cup S_4$ 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_v (including $2i$ divide nodes), $(a + c + 2i)$, is even. If v is a conflict node, the number of true nodes in G_v , $c + d$, 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 v 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^v , the total weight of the edges in the T-join solution, $S = S_1 \cup S_4$, 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 0), = 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 0). 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 $v \in G^D$ whose gadget is G_v , which includes $2i + 1$ subsets $G_{v,j}$ $j = 1 \dots 2i + 1$ linked with divide nodes, the true nodes and ghost nodes can be grouped into four sets:

- $S_1 = \{g_e^v | e \in S\}$.
- $S_2 = \{g_e^v | e \notin S\}$.
- $S_3 = \{t_e^v | e \notin S\}$.

- $S_4 = \{t_e^v | e \in S\}$.

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 divide nodes, the a ghost nodes in S_1 and the c true nodes in S_3 are matched within G_v , and the remaining nodes are matched outside G_v .*

If v 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_v , $(a + c)$, is even. Since all the $2i$ divide nodes should be matched within G_v , the total number of nodes to be matched, $(a + c) + 2i$, is even. According to Lemma 3, all nodes to be matched in $G_{v,1}, \dots, G_{v,2i}$ and the divide nodes $d_{v,1}, \dots, d_{v,2i-1}$ can be matched in a matching solution. The remaining even number of nodes are located in a complete graph $G_{v,2i} \cup \{d_{v,2i}\}$. 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 v 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. \square

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 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 the present work, 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 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 poly interconnect 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 phase conflicts in each row can be solved independently without introducing any DRC errors. The overall flow of the algorithm is presented in Figure 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 Figure 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 Figure 9 (a), the space insertion is equal to moving all features on the right side of the cut line

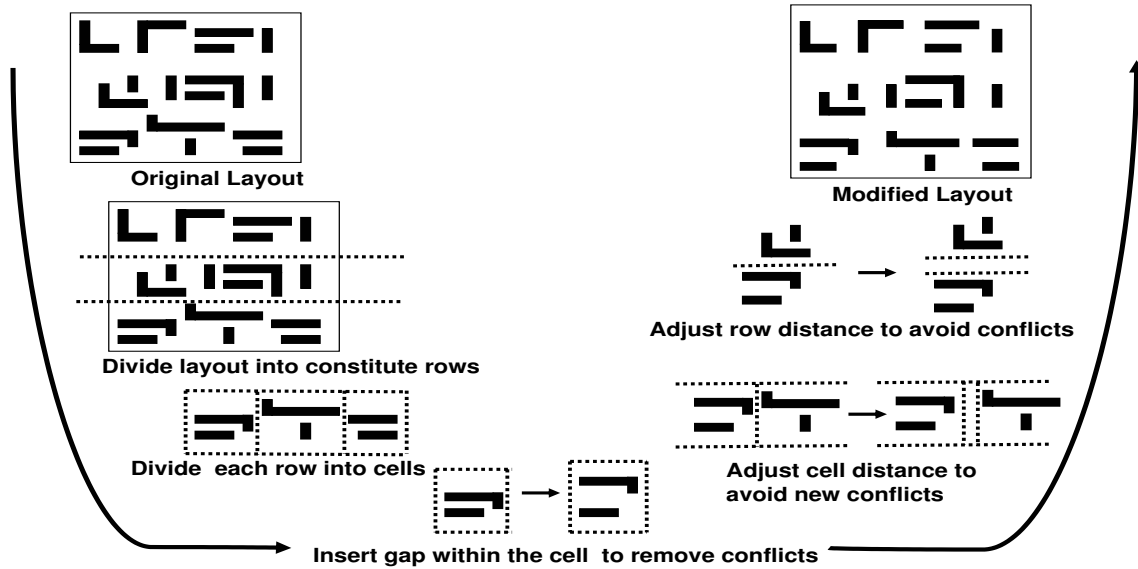


Fig. 8. Details of Layout Modification Algorithm.

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 (Figure 9 (a)), it will not be moved; otherwise, if it is not connected with the features on the left side of the cut line (Figure 9 (b)), it will be moved to the right by a distance B ; if it is connected with the features on both sides (Figure 9 (c) (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 Figure 9 (d). This prevents any major timing problems after 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.

- 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 is 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

¹⁰In 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 Figure 9 (c) and (d); and (3) if the cases like Figure 9 (c) and (d) can not 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.

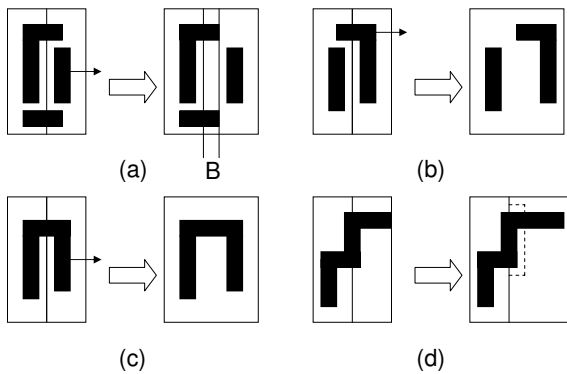


Fig. 9. Layout modification with vertical space insertion.

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 Figure 9, to reflect the area increase due to 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.

Figure 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 5 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 denotes the width of horizontal and vertical spaces added (assumed to be the same for simplicity). The area increase with the scheme in [2] is $11l/w$, whereas the area increase with our scheme is only $6l/w$.

The presented algorithm can be applied as an additional processing step during post-placement optimizations. The inserted spaces are integer multiples of the M1 routing pitch

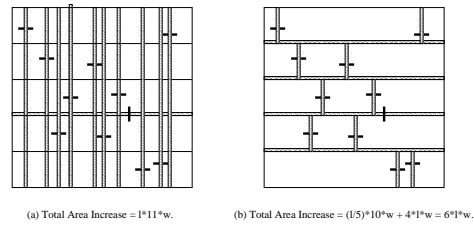


Fig. 10. Comparing the area increases produced by the layout modification scheme in [2] with the proposed scheme.

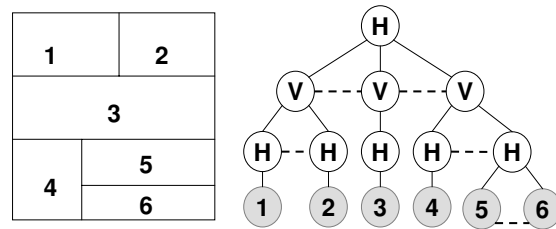


Fig. 11. Hierarchical layout and its partition tree.

and hence the modifications introduced by the proposed flow does 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 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 phase conflicts in slicing hierarchical layouts, i.e., the layouts whose floorplan can be represented by slicing trees. As shown in Figure 11, a slicing hierarchical layout can be represented using the *partition tree*, where each leaf node represent 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

| Design | # Plgns/#Ovlp | Flow in [2] | | | | Proposed Flow | | | |
|--------|---------------|-------------|------------------|--------|-------------|---------------|------------------|--------|-------------|
| | | #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 I
PHASE CONFLICT DETECTION RESULTS. EXPERIMENTS WERE RUN ON A 4X400 MHZ ULTRA-SPARC II
WITH 4.0 GB OF RAM.

upper level, new phase conflicts are avoided by inserting gaps along the cut lines. The algorithm shown in Figure 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 work focuses mainly on 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 [7]. Phase conflicts caused by line-end conflicts between neighboring features can be detected and corrected are not considered as they can be efficiently detected and corrected using additional DRC checks during layout generation [14].

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 [5] [6]. 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, respectively¹¹. As can be seen, our runtimes are significantly better than those obtained using the flow in [2] with an average improvement of 5.9x. 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

¹¹It 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 2x using the new graph, but the perfect matching times has a greater contribution to the total run-time.

| 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 |

TABLE II
LAYOUT MODIFICATION RESULTS FOR STANDARD-CELL BLOCKS.

of speed-ups 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 labelled “# 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 Figure 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 sub-graph of the original graph.

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 microns. 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 are selected for correction and occur 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 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 1.7-9.1%, with an average increase of 6.1%. The area increase goes up slightly with the size of the testcases. 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. CONCLUSIONS

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 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 non-planar 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 previous work in this area. The small area increases make it suitable for use in a true industrial flow as a post-placement 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 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. In *SPIE (2197)*, pages 612–623, 1994.
- [2] C. Chiang, A. Kahng, S. Sinha, X. Xu and A. Zelikovsky. Bright-Field AAPSM Conflict Detection and Correction. In *DATE*, pages 908–913, 2005.
- [3] A. Moniwa, T. Terasawa, N. Hasegawa and S. Okazaki. Algorithms for Phase-Shift Mask Design with Priority on Shifter Placement. In *Jpn J. of App. Phys. 34*, pages 6584–6589, 1993.
- [4] K. Ooi, S. Hara and K. Koyama. Computer-Aided Design Software for Designing Phase-Shift Masks. In *Jpn J. of App. Phys. 32*, pages 5887–5891, 1993.
- [5] 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.. In *IEEE TCAD (9)*, pages 1265–1278, 1999.
- [6] A.B. Kahng, S. Vaya and A. Zelikovsky. New Graph Bipartitions for Double-Exposure, Bright Field Alternating Phase-Shift Mask Layout. In *ASP-DAC*, pages 133–138, 2001.
- [7] C. Pierrat, F.A. Driessen and G. Vandenbergh. Full phase-shifting methodology for 65 nm node lithography. In *SPIE (5040)*, pages 282–293, 2003.
- [8] K. Ooi, K. Koyama and M. Kiryu. Method of Designing Phase-Shifting Masks Utilizing a Compactor. In *Jpn J. of App. Phys. 32*, pages 6774–6778, 1994.
- [9] A. Moniwa, T. Terasawa, K. Nakajo, J. Sakemi and S. Okazaki. Heuristic Method for Phase-Conflict Minimization in Automatic Phase- Shift Mask Design. In *Jpn J. of App. Phys. 34*, pages 6584–6589, 1995.
- [10] K. Cao, J. Hu and M. Cheng. Layout modification for library cell Alt-PSM composability. In *SPIE*, 2004.
- [11] W.J. Cook, W.H. Cunningham, W.R. Pulleyblank and A. Schrijver. Combinatorial Optimization. *Wiley Inter-Science*, New York, 1998.
- [12] W. Cook and A. Rohe. Computing Minimum-Weight Perfect Matchings. August , 1998. <http://www.or.unibonn.de/home/rohe/matching.html>.
- [13] L. Liebmann, J. Lund, F.L. Heng, I. Graur. Enabling alternating phase shifted mask designs for a full logic gate level: design rules and design rule checking. In *DAC*, pages 78–84, 2001.
- [14] P. Ghosh, C. Kang, M. Sanie and J. Huckabay. Psm-Lint: Bringing Altpsm benefits to the IC design stage. In *SPIE (5042)*, pages 314–325, 2003.

Dear Professor Macii:

We have thoroughly revised the manuscript taking all reviewer comments in consideration. The main changes are:

- We added the explanation for Figure 2, which explains our phase conflict detection flow. Non-planar graphs is greedily converted to planar graphs in a planarization step.
- We add the phase assignment algorithm in Figure 4 and modified the proof of Theorem 1.
- We modified the proof of Theorem 2 to make it clearer.
- We added a footnote in Page 2 to clarify that we only consider spacing-based conflict correction and T-shaped poly connections are assumed to be corrected by other methods.
- The weighting scheme is added in Section IV.
- We modified Figure 6 and introduced dummy nodes to help explain the gadget graph construction.
- The proof of equal-weight during mapping is added for Theorem 3 and 4.

We thank the diligent referees for their very careful reviews. Their comments have helped us to improve the paper. We have addressed the specific comments as follows:

COMMENTS OF THE ASSOCIATE EDITOR

COMMENTS OF REVIEWER #1.

This is a solid work in general. There are plenty of experimental results to show the effectiveness of the proposed algorithms and flow. However, there are still a few points that need to be improved as a journal article.

Comment 1.1: *It is strange that the authors claim ref[2] as the first work on bright field PSM, as ref[6] is also on bright field PSM and appeared earlier. Or there is subtle difference between AA (Alternating Aperture) and “double exposure”. Please clarify this.*

Response: The authors of ref[6] only consider layout conflict detection. They did not present any layout conflict correction methods. Therefore, we add a footnote in Page 3 as follows: Although the work in [6] proposes the conflict detection methods for bright-field phase conflicts based on feature widening, it neglects the layout modification problem.

Comment 1.2: *Ref[7] suggested to cut phase conflict regions instead of increasing spacing. Please discuss the pro and con of the proposed method versus [7].*

Response: We add the following in the introduction section (Page 4): Mask-level correction based approaches [7] split shifter regions whenever two shifters

of opposite phases overlap to avoid layout modification. However, the mask complexity is increased and it is not always possible to split the shifter regions without negatively affecting process latitude.

Comment 1.3: *Since the two nodes of a conflict edge are always in the same color, how about shrink such edges into single nodes? Then, the problem is still an odd cycle problem.*

Response: The two nodes of a conflict edge are in the same color *only if the layout is phase-assignable*, i.e., there are no phase conflicts. The goal of the conflict detection process is to select some conflict edges to be deleted. Therefore, we can shrink the conflict edges into single nodes *only if* we are sure that these edges will not be deleted.

Comment 1.4: *Page 8, line 6, typo, “the graph is an an embedded planar graph.”*

Response: Corrected.

Comment 1.5: *According the proof of Theorem 2, the condition of Theorem is satisfied only when $|S1|$ is odd, in other words, only when the number of conflict faces is odd. What will happen and is your approach still valid if $|S1|$ is even in a layout?*

Response: Actually, Theorem 1 is satisfied when $|S1|$ is even. We prove that $|S1|$ must be even if we remove an odd number of edges from every conflict face and an even number of edges from every legal face. Also removing an odd number of edges from every conflict face and an even number of edges from every legal face is equal to the T-join problem. So the main purpose of Theorem 2 is to convert the layout phase conflict detection problem to the T-join problem. We rewrite the proof to make this clear.

Comment 1.6: *The basic algorithm here is based on planar graph. Please discuss the chance of non-planar in practice.*

Response: As shown in Figure 2, our proposed flow can handle the non-planar cases with a planarization step. We first remove some conflict edges to make the graph a planar graph and put these edges in the set P . After we delete some edges to make the graph phase-assignable and assign the phases, we check the conflict edges in P whether they should be deleted or not. We add a description of the flow shown in Figure 2 to make the flow clear.

Comment 1.7: *Currently, there are commercial tools providing PSM clean cell layout. It is not clear why you have to work on the cell level PSM conflicts. Looks only focusing on inter-cell conflicts should be sufficient.*

Response: Our proposed method is a general method which works for both PSM clean cells and cells having PSM conflicts. For PSM clean cells, the whole layout conflict detection and correction are also needed. Although we can choose a fixed phase assignment for each cell, the area increase is large since most conflicts can be avoided with correct phase assignment. There are some other cell-based approaches with additional constraints on cells. We discuss these methods in the last paragraph of Section II.

Comment 1.8: *This work does not consider T-shaped poly connections for the reason of maintaining timing performance. However, the poly may be field poly rather than gate poly, widening field poly will not affect timing.*

Response: It is easy to incorporate feature widening into our solution. Conflict detection would remain unchanged except that feature edges can also be deleted. Feature widening would correspond to inserting spaces in feature regions. However, widening some features could negatively impact timing. In such cases, it may be preferable to resolve the corresponding conflicts using mask splitting.

We added a footnote for T-shaped poly connections in Page 2: *T-shaped phase conflicts* and other local phase conflicts can be easily detected with simple design rule checking and corrected with phase splitting [7], feature widening [13] or cell re-design. 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.

We believe that there are two kinds of conflict: one is “local” conflicts (such as T-shaped conflicts) which can be easily detected but can not be solved by spacing; the other is “global” conflicts (such as odd-even conflicts) which can not be easily detected with simple design rule checking and can be corrected by spacing. The focus of this paper is mainly on the “global” conflicts.

COMMENTS OF REVIEWER #2.

This paper introduces a novel data structure and an efficient algorithm for identifying phase conflicts as well as edges to be deleted in AAPSM, together with a scheme to fix the phase conflicts with small area penalty. The algorithm runs fast, and can produce comparable results with [2]. However, there are several issues need attention before the paper can be published, as detailed below.

Comment 2.1: *In the proof of theorem 1, authors try to prove “ G has no conflict cycle \rightarrow the layout is phase assignable.” This point can be proved in a*

stricter way. Current proof is actually based on the following assumption: If any L is not phase-assignable, there must exist an assignment in which n_1 and n_2 (which are neighboring) are the ONLY edge shifter nodes that can not be assigned properly, and there is no other violation in L . But authors did not provide proof for this. Maybe a constructive proof, which actually assigns the phase, can be more illustrative.

Response: We agree with the reviewer’s opinion. The main reason for this confusion is that the phase assignment algorithm is missing. We add the phase assignment algorithm in Figure 4. We also modify the proof to incorporate the reviewer’s comments.

Comment 2.2: *In conflict cycle resolution, the algorithm to determine the edges to be deleted (T-join) is based on the minimum weight deletion of edges. However, authors did not provide a specific weighting scheme for the conflict cycle graph in the paper. How will various schemes of weighting factors affect the final experimental results?*

Response: We add the following in Section III.A “Each edge has a given weight which reflects the negative effects of correcting the phase conflict.” as well as a footnote: “The weighting scheme depends on the layout modification methods, which will be discussed in Section IV.”

The following weighting scheme is added in section IV: 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 Figure 9, to reflect the area increase due to 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.

Comment 2.3: *In Figure 5 which shows the how to convert a dual graph to a gadget graph, the figures and the illustrations below are somewhat confusing: (1) The dual graph is essentially an undirected graph, but the edges are directed in Figure 5 (a). Is there any special meaning for the directed edges?*

(2) Figure 5(b) is illustrated as “gadget graph”, but there is no edge between gadget groups, and this is different from the definition of gadget graph.

(3) Figure 5(c) shows a final gadget graph with all unnecessary edges deleted, but it is not used in later perfect matching. It is actually not used in later part of the paper. The proof for theorem 3 is essentially based on Figure 5(b) (with all gadgets properly connected), not Figure 5(c). Thus drawing Figure 5(c) is somewhat confusing here.

(4) Adding a specific definition of a gadget graph $G = V, E$, and defining what V and E consists of, can show the idea more clearly. Currently in the paper, gadget graph is defined in a constructive way, instead of a specific way. In the step 2 of building gadget graph, in item (d), t_e^v should be g_e^v .

Response: (1) We add the following: “Each edge e connecting v and v' in dual graph is assigned to v , v' or both. If the edge is assigned to v , it will appear as a true node t_e^v in G_v ; otherwise, it will appear as a ghost node g_e^v in G_v . In Figure 6 (a), directed edges are used to represent the assignment.” And a footnote to show an example: “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 .”

(2)(3) We introduce dummy nodes in Figure 6 (b) to help the explanation. We delete Figure 6 (c).

(4) We add: $\mathcal{G} = (V', E')$, where V' includes ghost nodes, true nodes and dummy nodes and E' is a set of edges between nodes in V' . Then we add “As a result, each node v of degree k in the dual graph becomes a gadget of k nodes in \mathcal{G} , which is denoted as G_v .” to explain the construction of the complete graphs (or gadgets).

Comment 2.4: The proof of Theorem 3 shows the equivalence of the perfect matching solution on a gadget graph is equivalent to the solution of the corresponding T-join problem. However, only the validity is shown here, but not the optimization part, i.e., authors did not show the minimum perfect matching solution does correspond to the minimum solution to the T-join problem. This can be analyzed by using the weighting scheme introduced in item (b) on page 10. The similar part is also missing from the proof of Theorem 4.

Response: We modify the weighting scheme description as follows:

- 1) For any edge e of weight $w(e)$ between two nodes v and v' in the dual graph which appears as g_e^v and $t_e^{v'}$ in the gadget graph, the weight of ghost node g_e^v is $w(e)$ and the weight of $t_e^{v'}$ is 0. In other words, the weight of the edge e , $w(e)$, is always assigned to the ghost node instead of the true node.
- 2) The weight of the edge in the gadget graph is the sum of the weights of its two nodes.

We also modify the proof of Theorem 3 and 4 to prove that the total weight remains the same during the mapping.

Comment 2.5: In part IV, layout modification, authors also discuss the application of layout modification to hierarchical layout. It seems this approach is good for slicing style hierarchical layout, but not non-slicing layout. It is more safe to say the scheme can be applied to slicing hierarchical layout.

Response: We agree with the reviewer and modified the draft.

Comment 2.6: The experimental results show the area increase from reference [2] is much larger than this approach. This can be due to the different layout modification scheme used in [2] and here. However, to compare the qualify of the conflict spots identified in [2] and in this paper, authors can also find the area increase for fixing conflicts spot found with [2], but using the modification scheme proposed in this paper.

Response: This experiment was done and we found that using the conflict detection scheme used in [2] with the new layout modification scheme produced much smaller area increase than the layout modification used in [2]. The results were omitted for the sake of brevity.

Comment 2.7: (1) Page 7, the third line of case 1, replace the comma with period before “Thus”. (2) Page 11, in the proof of theorem 3, “T-join” problem can be stated more clearly as “extended T-join” problem. (3) Page 12, item (d) in the middle of page, t_e^v should be g_e^v . (4) Page 18, at the beginning of part A, “Table I” instead of “Table IV”. (5) Page 17, Table I, there is no unit for the runtime.

Response: Modified.

COMMENTS OF REVIEWER #3.

A fast algorithm for detecting the phase conflicts in the layout to be used with AAPSM is proposed. Several contributions are claimed including the introduction of a new graph, the “conflict cycle graph” to represent phase-assignability of the layout, a new formulation for the problem as a minimum-weight conflict cycle problem, rather than a bipartition problem, and other speed ups. Overall, it’s a solid paper, and it is very well written. Here are several points that should be taken into account in preparing the manuscript for publication:

Comment 3.1: Figure 2 on page 4 should be accompanied by more explanation - it is not intelligible at that point of the discussion. Perhaps, it’s worth moving it to later in the text.

Response: We add a description of the flow shown in Figure 2 to make the flow clear. The main purpose of Figure 2 is to propose the conflict detection flow in which the edge deletion for planar graph is the third

step and is the most important step. Then the rest part of Section III will focus on this step.

Comment 3.2: *It is not clear from the text how the authors propose to weight the edges of the conflict graph. Obviously, this is an essential issue since the goodness of solution depends on the proper choice of edge weights.*

Response: The weighting scheme depends on the layout modification methods. The following weighting scheme is added in section IV: 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 Figure 9, to reflect the area increase due to 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.

Comment 3.3: *The authors do not say much about the impact of the proposed layout changes for fixing phase conflicts on timing. Making certain gates wider and thus faster may, in fact, lead to "major timing problems". It would be desirable if the authors tried to address this issue more directly.*

Response: We added the description of our layout modification method in Figure 9. Also a footnote is added to explain why our method will not cause major timing problems: 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 Figure 9 (c) and (d); and (3) if the cases like Figure 9 (c) and (d) can not 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.

We agree that as a general method, we need to consider the timing impact even if it is negligible for our current testcases. The impact on timing is currently being investigated. We are currently looking at schemes to translate slack information after timing into flexibility for changing gate widths.

Comment 3.4: *On p. 15, in the last paragraph, should it say "only the widths of features are increased but the lengths of the features remain the same. This prevents any major timing problems after layout modification."*

Response: We agree with the reviewer and modified the draft.

COMMENTS OF REVIEWER #4.

The main contribution of the paper is the formulation of PSM phase assignment problem as a conflict cycle graph, and algorithms in solving the conflict detection problem. These are significant contributions. This reviewer will like to see the following revision:

Comment 4.1: *The paper will be more complete if some practical phase conflict issues are addressed. In practice there are a number of tricky phase conflicts legalization problems, these are not addressed by the authors. For example, two merged shifters may not be a legal phase shape. Please refer to "Enabling AltPSM for a Full Logic Gate Level: Design Rules and Design Rule Checking", L. Liebmann, J. Lund, F.L. Heng, I. Graur, Design Automation Conference, June 2001, for other phase legality issues.*

Response: We added a footnote for T-shaped poly connections and other local conflicts in page 2: *T-shaped phase conflicts* and other local phase conflicts can be easily detected with simple design rule checking and corrected with phase splitting [7], feature widening [13] or cell re-design. 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.

We believe that there are two kinds of conflict: one is "local" conflicts (such as T-shaped conflicts) which can be easily detected but can not be solved by spacing; the other is "global" conflicts (such as odd-even conflicts) which can not be easily detected with simple design rule checking and can be corrected by spacing. The focus of this paper is mainly on the "global" conflicts.

Comment 4.2: *Please clarify if reference [2] handled the T-shapes problem. If so, the comparison is not a fair one, since the conflict cycle graph does not handle T-shapes. The 5.9x runtime improvement will need to be discounted. Please also clarify how T-shapes are detected in the present algorithm, since the conflict edge due to shifters of the a T-shape might be deleted if not identified.*

Response: Reference [2] did not handle the T-shapes problem since T-shapes can not be corrected with spacing. However, they can be easily detected and solved by previous methods like [7] [13]. Basically, T-shapes are not the focus of this paper.

Comment 4.3: *Some typos in section V: Table IV does not exist, should be Table I. "Flow in [8]" should have been "Flow in [2]". There is another wrong referece to Table IV.*

Response: Corrected.
Sincerely,

Xu Xu
Computer Science & Engineering Department
University of California, San Diego
9500 Gilman Dr.
La Jolla, CA 92093-0114, USA