Fast and Efficient Phase Conflict Detection and Correction in Standard-Cell Layouts

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 [3, 4, 5, 6, 8, 9].

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 in standard-cell blocks. 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 [8]. 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. The proposed layout modification scheme can handle all phase conflicts in standard-cell blocks with small increases in area. Our experiments show that the percentage area increase for making standard-cell blocks phase-assignable ranges from 1.7-9.1%.

1 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. 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^o and 180^o); and
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.

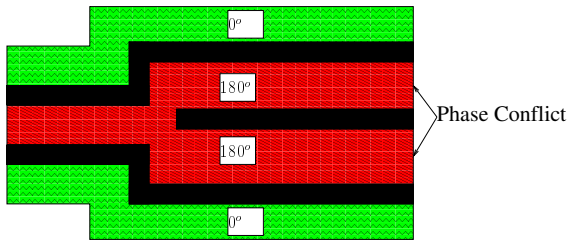


Figure 1. Example of incorrect phase assignment.

Our contributions are summarized as follows:

- 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 5.9x, while maintaining the same quality of results as the best results available today [8].
- 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. Compared to the approach in [8], 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. The proofs of all the theorems introduced in the paper is given in the Appendix.

2 Previous Work

The phase conflict detection problem is addressed in [1, 2, 5, 6, 8]. 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 graphs[1] and an optimal solution is provided for that case. The most recent work in this area is presented in [8]. 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 [6] 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 out-performs other existing work in the area.

Previous work in phase conflict correction falls into two major categories. Layout modification based approaches remove the conflicts by increasing spacing between features or widening critical features [1, 2, 4, 3, 5, 6, 8]. Most of these works focus on dark-field AAPSM[2]. The first layout modification scheme for correcting bright-field phase conflicts is presented in [8]. 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 [12]. 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.

3 Proposed Phase Conflict Detection Scheme

In this section, the proposed phase conflict detection scheme is presented. The proposed conflict detection flow in shown in Figure 2. 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*** [8] (an average reduction of 31% in the number of edges is achieved using the simpler graph).

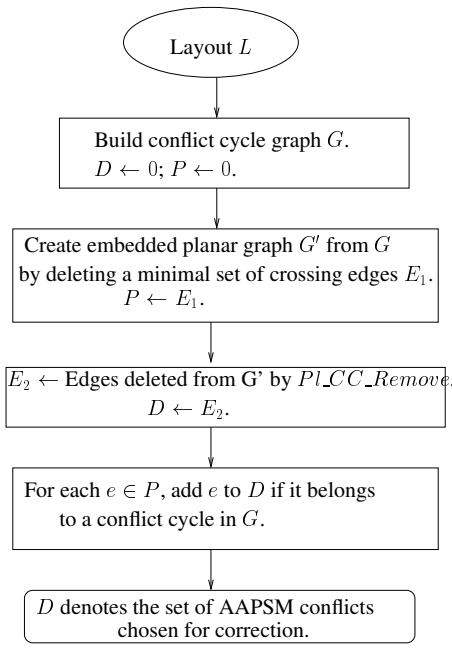


Figure 2. Phase Conflict Detection Flow.

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, which will be described in Section 3.2, 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 [8], 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 [8].

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 lay-

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

[2] 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.

out.

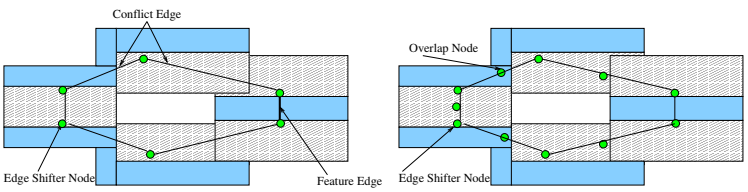
- Reduction of the phase conflict detection problem to a minimum-weight conflict cycle problem (to be defined later) on the conflict cycle graph and an optimal polynomial-time algorithm for the same, when the graph is an embedded planar graph.
- Improvements to the intermediate reductions such that edges 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.

3.1 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[3] 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.



(a) Conflict Cycle Graph (b) Phase Conflict Graph

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

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 bipartition. 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 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.

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: See Appendix for proof.

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. 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.

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 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.

3.2 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**.

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.*

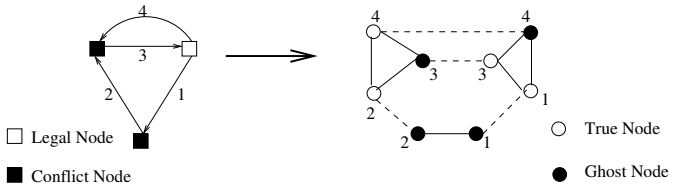
Proof: See Appendix for proof.

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:
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

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

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 [7] 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 [8] in which T is the set of all nodes with odd degrees, in our formulation, T may include nodes with odd or even degrees. In this paper, we will refer to our problem as the **extended T-join** problem.



(a) Original Graph (b) Gadget graph

Figure 4. Gadget graph construction from dual graph.

Next, we describe how the **extended 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:

1. Each node v in the dual graph G^D corresponds to a set of nodes in $\mathcal{G}$, which is denoted as G_v. Each edge e connecting v and v' in dual graph is arbitrarily assigned to either v or v'. If the edge is assigned to v, it will appear as a *true node* t_e^v in G_v. If the edge is not assigned to v, it will appear as *ghost node* g_e^v in G_v. The assignment is done such that the following conditions are satisfied:
   (a) For each conflict node u, the parity of the number of ghost nodes in G_u is different from the parity of its node degree.
   (b) For each legal node v, the parity of the number of ghost nodes in G_v is equal to the parity of its node degree.

   It is easy to ensure that the conditions are always satisfied since it is always possible to turn a ghost node g_e^v into a true node t_e^v and adding an edge of weight $w(e)$ between t_e^v and $t_e^{v'}$ to meet the parity requirement at the cost of increasing the node number by one.

2. The nodes in G_v are connected to each other by weighted edges as follows:
   (a) An edge of 0 weight is added between t_e^v and $t_{e'}^v$.
   (b) An edge of weight $w(e')$ is added between t_e^v and $g_{e'}^v$.
   (c) An edge of weight $w(e)$ is added between g_e^v and $t_{e'}^v$.
   (d) An edge of weight $(w(e) + w(e'))$ is added between g_e^v and $t_{e'}^v$.

Figure 4 illustrates the construction of the gadget graph from the dual graph. Note that the assignment of the edges (indicated by edges pointing to the node) follows the rules specified in Step 1 of the construction.

Theorem 3 *The* **extended 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 $G' = (V', E', w')$.*

Proof: See Appendix for proof.

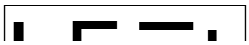
The perfect matching problem can be optimally solved in polynomial time. In our implementation, we integrate the code of Cook and Rohe [11].

It should be noted that using the proposed conflict cycle graph and the extended 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, 8], which in turn resulted in the increased complexity of their constructed gadget graphs.

4 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 [8] 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. This method when applied directly to standard-cell blocks can cause large increases in area (experimental results are shown in Section 5).

The algorithm presented in this section 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 5. 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