

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.

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 [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. 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 [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°); 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.

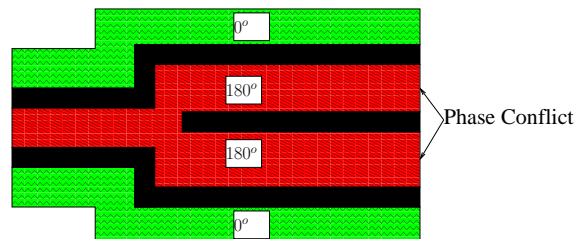


Figure 1. Example of incorrect phase assignment.

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 [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. 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. 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 [3, 4, 5, 6, 2]. 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 graphs¹ 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 [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 [3, 4, 8, 7, 5, 6, 2]. Most of these works focus on dark-field AAPSM². The first layout modification scheme for correcting bright-field phase conflicts is presented in [2]. 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 [13]. 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 is shown in

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

²In 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.

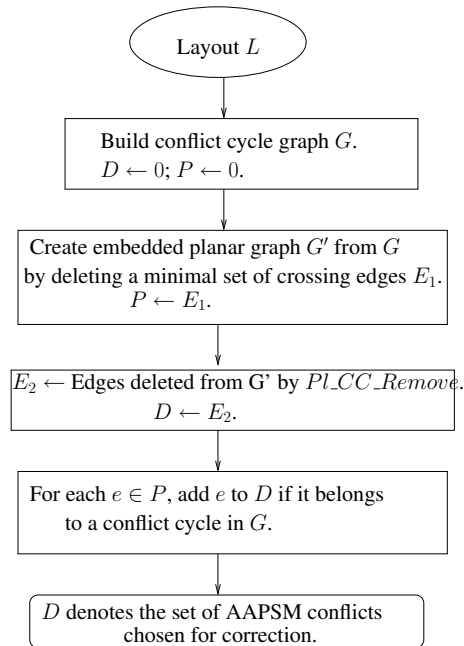


Figure 2. Phase Conflict Detection Flow.

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* [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, 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 [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 lay-

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 shifters³ 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.

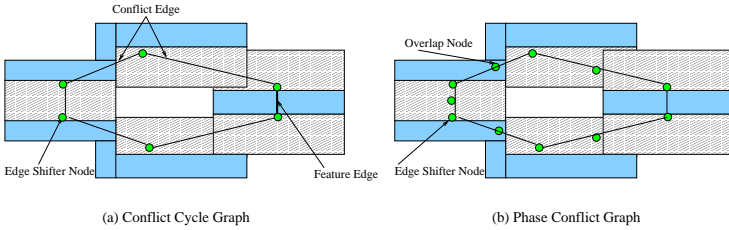


Figure 3. (a) Conflict Cycle Graph. (b) Phase Conflict Graph.

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 edges. 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

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

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 embedded planar graph. This optimal algorithm is referred to as `PLCC_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

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 [9] 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. In this paper, we will refer to our problem as the **extended T-join** problem.

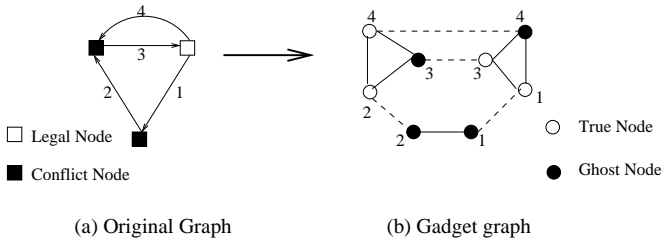


Figure 4. Gadget graph construction from dual graph.

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 [12].

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, 2], 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 [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. 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

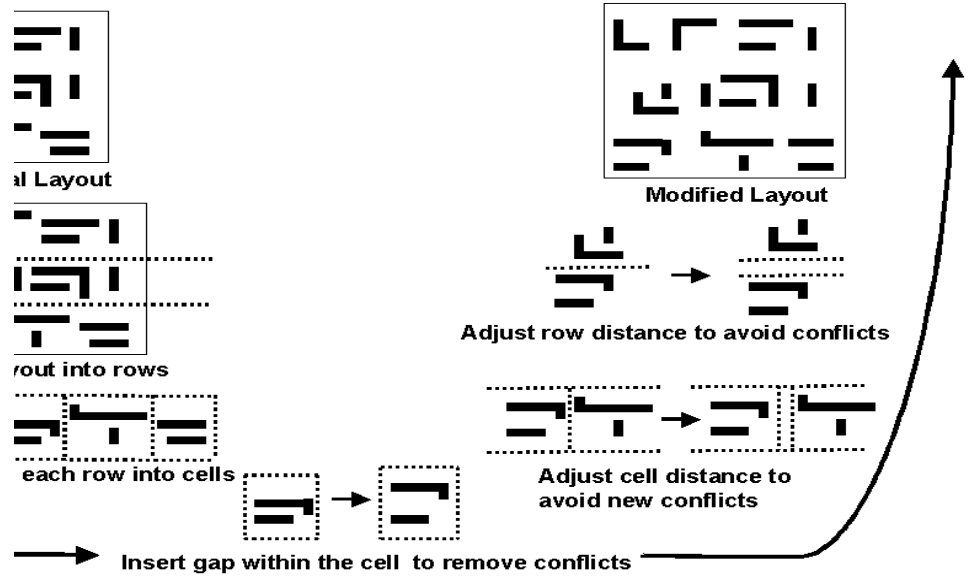


Figure 5. Details of Layout Modification Algorithm.

spaces in the cell. A minimum-weighted set-covering problem (similar to the one proposed in [2]) is used to determine the locations and the widths of the end-to-end spaces. 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.

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].

Figure 6 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

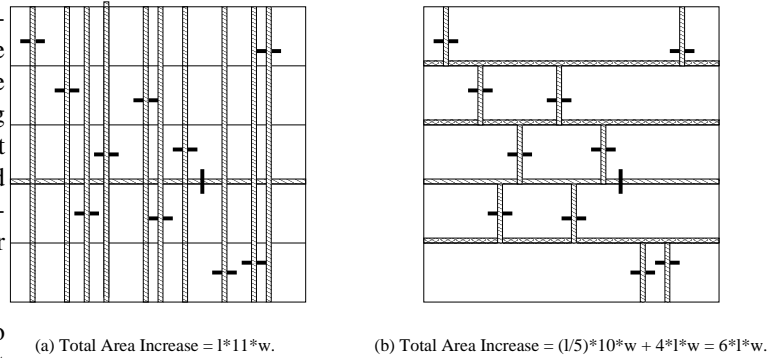


Figure 6. Comparing the area increases produced by the layout modification scheme in [2] with the proposed scheme.

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 $11lw$, whereas the area increase with our scheme is only $6lw$.

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 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 hierarchical layouts. As shown in Figure 7, a hierarchical layout can be represented using the *partition tree*, where

Design	# Plgns/#Shft Ovlp	Flow in [2]				Proposed Flow			
		#edges	#nodes/#edges GG	Runtime	# Conflicts	#edges	#nodes/#edges GG	Runtime	# Conflicts
Design1	10274/24580	98347	75086/242093	2.53	938	66875	25198/46346	0.38	910
Design2	13630/32257	112599	62480/203971	1.90	963	79672	20910/36403	0.22	946
Design3	21868/53749	182809	103912/338810	3.33	1558	128836	34806/61227	0.55	1534
Design4	20425/50059	173319	98834/320499	3.18	1678	121546	33266/57705	0.48	1664
Design5	25784/63760	216691	122324/397999	3.87	1854	152655	40614/70853	0.60	1839
Design6	48787/157668	484999	355760/1147057	12.17	6330	325769	8077/45208	2.78	5989
Design7	44121/142707	436297	339538/1084677	12.30	5010	295001	112152/207784	2.47	4865
Design8	72101/237557	729980	567532/1817272	20.05	10275	489922	189986/351007	4.23	9631
Design9	105882/376707	1133279	949064/299758	41.35	18148	757581	303408/565217	7.95	17463
Design10	159070/552767	1667581	1415620/4437522	66.40	27308	1115928	444082/829898	11.58	26349

Table 1. Phase conflict detection results. Experiments were run on a 4X400 Mhz Ultra-Sparc II with 4.0 GB of RAM.

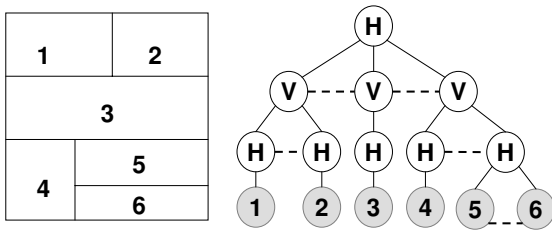


Figure 7. Hierarchical layout and its partition tree.

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 upper level, new phase conflicts are avoided by inserting gaps along the cut lines. The algorithm shown in Figure 6 is the special case when the input layout can be represented as a two-level tree.

5 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 [10]. 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 [11].

5.1 Phase Conflict Detection Results

Table 1 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 runtimes, 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 [8]”. The results obtained using our phase conflict detection method are grouped under “Proposed Flow”. Columns 5 and 9 compare the runtimes of the flow in [2] and our proposed flow, respectively⁴. 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 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 labeled “# Conflicts” under the subgroup “Flow in [2]” for the results obtained using the method in [2]) and Column labeled “# Conflicts” under sub-group “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.

5.2 Phase Conflict Correction Results

Table 2 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 1 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

⁴It 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]
Design1	25173.96	937	61	1.7	18.1
Design2	16397.82	995	197	5.4	23.1
Design3	31416.21	1589	284	5.8	26.8
Design4	25715.23	1724	238	6.1	28.8
Design5	40409.68	1720	322	4.7	32.12
Design6	61705.52	6257	770	5.8	57.47
Design7	58414.06	5100	586	6.1	59.1
Design8	94178.09	10141	512	7.3	80.2
Design9	148231.77	18657	2672	9.1	>100.0
Design10	249210.41	28121	4224	9.0	>100.0

Table 2. Layout modification Results for Standard-Cell Blocks.

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%. 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.

6 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 Bipartizations for Double-Exposure, Bright Field Alternating Phase-Shift Mask Layout. In *ASP-DAC*, pages 133–138, 2001.

[7] 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.

[8] 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.

[9] W.J. Cook, W.H. Cunningham, W.R. Pulleyblank and A. Shrijver. Combinatorial Optimization. *Wiley Inter-Science*, New York, 1998.

[10] C. Pierrat, F.A. Driessen and G. Vandenberghe. Full phase-shifting methodology for 65 nm node lithography. In *SPIE (5040)*, pages 282–293, 2003.

[11] P. Ghosh, C. Kang, M. Sanie and J. Huckabay. PsmLint: Bringing Altpsm benefits to the IC design stage. In *SPIE (5042)*, pages 314–325, 2003.

[12] W. Cook and A. Rohe. Computing Minimum-Weight Perfect Matchings. August, 1998. <http://www.or.unibonn.de/home/rohe/matching.html>.

[13] K. Cao, J. Hu and M. Cheng. Layout modification for library cell Alt-PSM composability. In *SPIE*, 2004.

APPENDIX

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

Proof: (→) Assume L is phase assignable. Let all the edge shifter nodes be colored with the same phases as the shifters in L . It is true that the node colorings of G satisfy the following two conditions: nodes connected by a feature edge have different colors and nodes connected by a conflict edge have the same color. Let us assume further that there exists a conflict

cycle C and let $\{n_1, n_2, \dots, n_k, n_1\}$ 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. WLOG, assume G is connected (if G is not connected, the proof can be done for each of connected components of G). Assume further that L is not phase-assignable. Then, there exists at least two shifters s_1 and s_2 that do not satisfy the conditions for phase assignment. Let n_1 and n_2 be the edge shifter nodes corresponding to s_1 and s_2 , respectively. Let the edge connecting them be denoted as e . Let C be a cycle that contains e and let n_0 be an intermediate node in C (any node other than n_1 and n_2).

There are two possible cases:

1. Shifters s_1 and s_2 are on opposite sides of critical feature and have same color: Since the phases are only changed across feature edges, if n_1 has the same phase 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 phase as n_1 , and hence as n_0 . Thus, there must be an even number of feature edges from n_0 to n_2 . Thus, C must contain an odd number of feature edges and hence C is a conflict cycle.
2. Overlapping shifters s_1 and s_2 have different colors: If n_1 and n_0 have the same phase, then there must be an even number of feature edges on the path from n_1 to n_0 . By assumption, n_2 has an opposite phase from n_1 and hence an opposite phase 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 hence 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

Definition 4 Two faces are **neighboring faces** if they share at least one common edge.

Definition 5 Two neighboring faces can form a **merged face** by deleting at least one common edge.

Lemma 1 The parity of the number of feature edges of the merged face of two neighboring faces 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 f 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. It is obvious that if the original graph has no conflict cycles, then every face is legal (by definition). \square

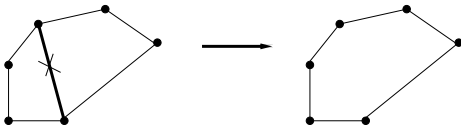


Figure 8. Deleting all common edges (in this case, only one) results in a merged face.

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: Assume that an odd number of edges are removed from all conflict faces and an even number of edges are removed from every legal face of G . As shown in Figure 8, the deletion of common edges results in the creation of a merged face. Let G_1 be the graph after the edge deletion. Any face in G_1 must be the result of merging a set S of faces in G . Let $S = S_1 \cup S_2$, where S_1 is the set of all conflict faces in S and S_2 is the set of all legal faces in S . The sum of the numbers of deleted edges of all the faces in S must be even, since any deleted edge belongs to two faces of S and is counted twice. The sum of the numbers of deleted edges of all the faces in S_2 is even since an even number of edges are removed from any legal face. Hence, the sum of the numbers of deleted edges of all the faces in S_1 should also be even. Since an odd number of edges are removed from any conflict face, the number of faces in S_1 must be even. Thus, the sum of the numbers of feature edges of all the faces in S_1 is even. It is obvious that the sum of the numbers of feature edges of all the faces in S_2 is even since every legal face has an even number of feature edges. Therefore, the sum of the numbers of feature edges of all the faces in S is even. From Lemma 1, the number of feature edges of the merged face is also even. So the merged face is legal. Thus, G_1 has no conflict cycles (Lemma 2). \square

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: (\rightarrow) Mapping perfect matching solution of the gadget graph G' to a valid solution of the **extended T-join** problem on the dual graph: For any node v whose gadget is G_v ,

1. If t_e^v is matched within G_v , $e \in S$;
2. If t_e^v is not matched within G_v , then $e \notin S$;
3. If g_e^v is matched within G_v , then $e \notin S$;
4. If g_e^v is not matched within G_v , then $e \in S$;

The set S thus constructed is a valid solution to the **extended T-join** problem. Assume a ghost nodes are matched within G_v , b ghost nodes are not matched with G_v , c true nodes are matched within G_v and d true nodes are not matched within G_v . In any perfect matching solution, the number of nodes matched within G_v , $(a + c)$, is even. The parity of $(a + b)$ is the same as the parity of $(a + b + a + c)$, whose parity is the same as the parity of $(b + c)$. Here, $(b + c)$ is the number of edges $\notin S$ and $(a + b)$ is the number of ghost nodes in G_v . For conflict node, we require that the parity of the number of ghost nodes in G_v , $(a + b)$, to be different from the parity of the node degree. Therefore, the parity of the number of edges $\notin S$, $(b + c)$, is also different from the parity of the node degree. Hence, the number of edges $\in S$ is odd for a conflict node. Similarly, it can be shown that for legal node, the number of edges $\in S$ is even. Therefore, the solution S is a valid solution of the **extended T-join** problem.

(\leftarrow) Mapping a solution S of the **extended T-join** problem to a solution of the perfect matching problem of the gadget graph can be done as follows: For any node v in the dual graph G^D , divide the edges connecting node v into four sets:

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

Let the cardinality of S_1 , S_2 , S_3 and S_4 be a , b , c and d , respectively. The a ghost nodes and the c true nodes are matched within G_v and the remaining nodes are matched outside G_v . Since S is a valid solution of the **extended T-join** problem, the parity of $(b + c)$ is the same as the parity of the number of ghost nodes $(a + b)$ (by construction). Thus, $((a + b) + (b + c)) = (a + c) + 2b$ is even. Hence, $(a + c)$ is even, which is the number of nodes to be matched within G_v . Obviously, it is always possible since there is an edge between any two nodes in G_v . \square