Overlay-Aware Detailed Routing for Self-Aligned Double Patterning Lithography Using the Cut Process*

Iou-Jen Liu[1], Shao-Yun Fang[2], and Yao-Wen Chang[1,3]
[1]Graduate Institute of Electronics Engineering, National Taiwan University, Taipei 106, Taiwan
[2]Department of Electrical Engineering, National Taiwan University of Science and Technology, Taipei 106, Taiwan.
[3]Department of Electrical Engineering, National Taiwan University, Taipei 106, Taiwan
yrliu@eda.ee.ntu.edu.tw; syfang@mail.ntust.edu.tw; ywchang@ntu.edu.tw

ABSTRACT

Self-aligned double patterning (SADP) is one of the most promising techniques for sub-20nm technology. Spacer-is-dielectric SADP using a cut process is getting popular because of its higher design flexibility; for example, it can decompose odd cycles without the need of inserting any stitch. This paper presents the first work that applies the cut process for decomposing odd cycles during routing. For SADP, further, overlay control is a critical issue for yield improvement; while published routers can handle only partial overlay scenarios, our work identifies all the scenarios that induce overlays and proposes a novel constraint graph to model all overlays. With the developed techniques, our router can achieve high-quality routing results with significantly fewer overlays (and thus better yields). Compared with three state-of-the-art studies, our algorithm can achieve the best quality and efficiency, with zero cut conflicts, smallest overlay length, highest routability, and fastest running time.

Categories and Subject Descriptors

B.7.2 [**Integrated Circuits**]: Design Aids

General Terms

Algorithms, Design, Performance

Keywords

Self-Aligned Double Patterning, Overlay, Routing, Manufacturability

1. INTRODUCTION

Before next generation lithography technologies such as *electron beam lithography* (EBL) and *extreme ultraviolet lithography* (EUVL) are ready, *double patterning* with 193nm immersion lithography is still the major technique for technology scaling, in which *self-aligned double patterning* (SADP) is one of the most promising candidates for sub-20nm technology nodes. Compared to litho-etch-litho-etch (LELE) double patterning, SADP has better critical dimension control due to its intrinsic self-aligned property [7].

In an SADP process, a layout is decomposed into two masks, a *core mask* and a *trim/cut mask*. Different from litho-etch-litho-etch (LELE) double patterning, a feature in the SADP process may not be directly defined by one of the two masks. There are two types of SADP processes: the SADP cut process using a cut mask [9] and the SADP trim process using a trim mask [7]. In both processes, every core pattern is surrounded by a spacer. The difference between the cut process and the trim process is the role of the second mask: in the cut process, the regions not covered by a spacer and the cut mask form the final layout. Figure 1(b) shows the layout decomposition result of a given layout shown in Figure 1(a) by using the cut process. In contrast, the regions not covered by the spacer but covered by the trim mask produce the final layout in the trim process, as the layout decomposition result shown in Figure 1(c). We refer to the patterns directly defined by the core mask as *main core patterns*, the patterns defined by a spacer and a trim/cut mask as *second patterns*, and the core patterns not existing in the final layout as *assistant core patterns*. In addition, an *overlay* can be produced in either process, which is defined as a section of a feature boundary that is not surrounded by a spacer. For example, in Figure 1(b), a section of the right boundary of pattern A induces an overlay. Also, in Figure 1(c), the top and right sides of pattern B are not surrounded by the spacer, and thus overlays occur. Since overlays can cause pattern distortion due to the misalignment of two masks, overlay minimization is one of the most critical issues for yield improvement in SADP [5]. Note that assistant core patterns provide protecting spacers for second patterns and effectively reduce overlays. Consequently, many existing layout decomposition algorithms use assistant core patterns for overlay minimization [1, 12, 13, 14].

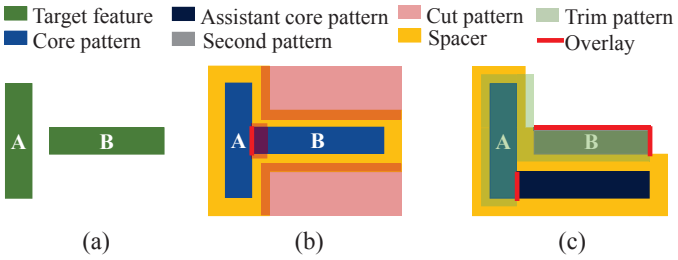

Figure 1: (a) A target layout. (b) SADP using the cut process. (c) SADP using the trim process.

The major advantage of SADP using the cut process over SADP using the trim process is that the cut process has higher design flexibility. In the trim process, a pattern is generated either by a core pattern or by a trim pattern, and they should be assigned to different masks if they are too close to be generated by the same mask. Since assigning each pattern to one of the two masks is analogous to assigning each pattern one of two colors, we define the mask spacing rule as the *minimum coloring distance*. As shown in Figure 2(a), the distance between each pair of the patterns is shorter than the minimum coloring distance, forming an odd cycle. Since an odd cycle is not 2-colorable, the layout is not decomposable with the trim process. Odd cycles, however, can be decomposed in the SADP cut process. As shown in Figure 2(b), the odd cycle is decomposed by merging pattern A and pattern B and assigning the remaining two patterns to different masks (different colors). Then, a cut pattern is used to separate patterns A and B [1, 9, 8]. In addition, two tip-to-tip placed patterns can also be merged first and be separated by using a cut pattern, increasing the design flexibility, as illustrated in Figures 2(c) and (d).

Since the SADP decomposability and the overlay controllability of an arbitrary layout are quite restricted, an SADP-aware detailed

*This work was partially supported by Genesys Logic, IBM, MediaTek, TSMC, and NSC of Taiwan under Grant No's. NSC 103-2918-I-002-012, NSC 102-2221-E-002-235-MY3, NSC 102-2923-E-002-006-MY3, NSC 101-2221-E-002-191-MY3, and NSC 100-2221-E-002-088-MY3.

DAC'14, June 01 - June 05, 2013, San Francisco, CA, USA.

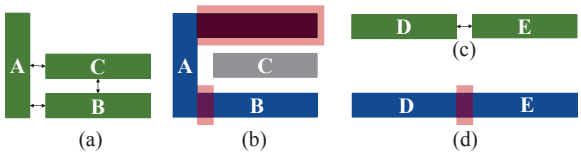

Figure 2: SADP using the cut process provides higher design flexibility. (a)(b) Odd cycle can be decomposed by merging patterns A and B and then applying a cut pattern on them. (c)(d) Patterns placed in a tip-to-tip manner can be assigned the same color.

router is desirable. Some SADP-aware detailed routing algorithms have been proposed for the trim process [2, 3, 4, 10] and for the cut process [11]. Du et al. proposed a graph model to capture the decomposition violations [2], and Gao et al. performed routing and layout decomposition simultaneously [3]. However, both studies do not consider assistant core patterns during routing, resulting in significant overlays. To the best of our knowledge, only the existing work [11] targets on SADP using the cut process. Nevertheless, the work [11] fails to apply the cut technique to solve odd cycles, and the merging of core patterns and assistant core patterns results in severe overlays.

This paper presents the first work that applies the cut technique for solving odd cycles during routing. Specifically, we propose the first SADP-aware detailed router using the merge technique in the cut process to decompose odd cycles of layout patterns. We identify all the scenarios that induce overlays and propose a novel constraint graph to precisely capture all overlays in a routing result. In addition, a color flipping algorithm is proposed to further reduce overlays and preserve routing resource.

The rest of this paper is organized as follows: Section 2 introduces side overlays, design rules for the SADP cut process, and the problem formulation. In Section 3, a detailed routing algorithm guided by the proposed overlay constraint graph is presented. Section 4 reports our experimental results. Finally, we conclude our work in Section 5.

2. PRELIMINARIES

This section gives some preliminaries for the addressed problem: the importance of overlay control for SADP, process rules adopted by this work, the definition of hard overlay, and the problem formulation.

2.1 Side Overlay

The major advantage of the SADP processes over the LELE double patterning process is the better overlay control achieved by spacer protection [13]. Pattern distortions due to overlays, however, still occur because of the shift of the second mask, which results in the misalignment between core patterns and trim/cut patterns. We denote the shift amount of the second mask as w_o. In addition, feature boundaries not protected by a spacer and directly defined by the cut mask may cause more severe pattern distortions. As illustrated in Figure 3, two patterns are respectively fabricated by the LELE double patterning process (see Figure 3(a)) and by the SADP cut process (see Figure 3(b)). In both cases, pattern A and pattern B are assigned different colors, and the second/cut mask shifts w_o toward left. In Figure 3(a), the shift of the second mask results in a shorter distance between pattern A and pattern B. In Figure 3(b), however, the shift of the cut shrinks the width of pattern B by w_o, which may result in electrical violations or yield loss. Therefore, to utilize the intrinsic self-aligned property of the SADP process, overlays have to be controlled very carefully.

We define a *side overlay* as a section of a side boundary of a feature not protected by a spacer and a *tip overlay* as a feature tip not protected by a spacer. As shown in Figure 4, the side boundaries of pattern A not protected by a spacer induces a side overlay, and a tip of pattern B has a tip overlay. Tip overlays are considered as non-critical overlays because tip overlays induce little critical dimension change [9, 12], but side overlays should be minimized to reduce yield loss.

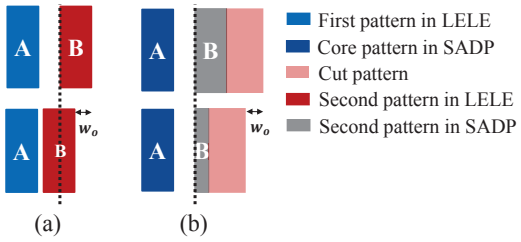

Figure 3: The misalignment of two masks causes different effects in LELE double patterning and in SADP. (a) Two patterns get closer in LELE double patterning. (b) The second pattern is seriously distorted in SADP.

Figure 4: A side boundary of a feature not protected by a spacer results in a side overlay.

2.2 Design Rules and Cut-Mask Conflict

In semiconductor manufacturing, design rules are set to ensure good manufacturability. The design rules considered in this work (and previous papers) are listed below: (1)w_{line} is the minimum width of a metal line. (2)w_{spacer} is the width of a spacer, which is equal to the minimum spacing between two metal lines in a grid-based design. (3)w_{cut} and w_{core} are the minimum widths of a cut pattern and a core pattern. (4)d_{cut} and d_{core} are the minimum distances between two cut patterns and two core patterns

Core patterns with distance within d_{core} can be merged by applying the merging technique [8], as shown in Figure 4 where pattern A is merged with the assistant core pattern. Ma et al. [8] pointed out that, for SADP, Mask Rule Check (MRC) challenges mostly come from the cut mask. We define a *cut conflict* as a cut pattern violating the minimum width rule (w_{cut}) or a pair of cut patterns violating the minimum distance rule (d_{cut}). Note that MRC violations occur over a spacer can be ignored because they do not affect the final layout [8]. To produce a decomposable routing result, cut conflicts should be strictly forbidden.

2.3 Problem Formulation

The work [8] has shown that a side overlay with length not longer than w_{line} is SADP-friendly. Therefore, we define a side overlay whose length is longer than w_{line} as a *hard overlay*; otherwise, it is a *non-hard overlay*. To simultaneously consider design flexibility and manufacturability, we allow non-hard overlays and strictly forbid hard overlays in our routing algorithm. Our overlay-aware detailed routing problem is then formulated as follows:

PROBLEM 1. *Given a netlist, a set of blockages, a grid-based routing plane, and a set of design rules, simultaneously perform detailed routing and layout decomposition to minimize the number of non-hard overlays such that no cut conflict, hard overlay, or design rule violation occur.*

3. OVERLAY-CONSTRAINT GRAPH GUIDED SADP-AWARE DETAILED ROUTING

To develop an overlay-aware SADP router, in this section we first explore potential overlay identification scenarios, then model an overlay constraint graph, and propose a liner-time color flipping algorithm and finally the overall scheme for SADP-aware detailed routing.

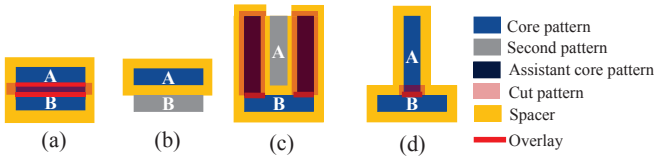

Figure 5: (a)(b) Assigning patterns A and B different colors would not induce any overlay. (c)(d) Assigning patterns A and B the same color would induce one side overlay.

3.1 Potential Overlay Scenarios

To control overlays efficiently and systematically, the first step is to have the information of which geometry relationships and color assignments of patterns induce overlays. There are three sources of overlays: (1) the merging of two core patterns, (2) the merging of a core pattern and an assistant core pattern, and (3) a second pattern not protected by a spacer. In the above three cases, some feature boundaries must be defined by cut patterns, which results in overlays. We first define two neighboring patterns to be *independent* if and only if the two patterns would not induce any overlay regardless of their color assignments. We have the following theorem:

THEOREM 1. Two neighboring patterns are independent if and only if the distance between the two patterns is larger than or equal to d_{indep}, where $d_{indep} = \sqrt{2} \cdot (w_{line} + 2 \cdot w_{spacer})$.

Then, we define *potential overlay scenarios* to be the geometry relationships of a pair of neighboring patterns that induce overlays when the patterns are assigned specific color combinations. Obviously, patterns in potential overlay scenarios are dependent. An example of a potential overlay scenario is shown in Figures 5(a) and (b). If both patterns A and B are core patterns, side overlays are induced on the bottom side of pattern A and on the top side of pattern B due to the cut pattern. In contrast, no overlay will be induced between the two patterns if pattern A is a core pattern and pattern B is a second pattern.

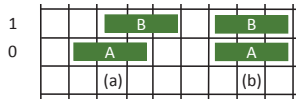

Figure 6: (a) and (b) have the same optimal color assignments even though their geometry scenarios are different.

To identify all potential overlay scenarios, we first examine the *geometry relationship* of a pair of dependent rectangles in a grid-based design, which can be characterized by using the following terms: (1)$X_{min}(A, B)$: the minimum track difference in the x-direction of two rectangles A and B. (2)$Y_{min}(A, B)$: the minimum track difference in the y-direction of two rectangles A and B. (3)$Dir(A, B)$: the directions of two rectangles A and B. $\perp$ means that A and B are orthogonal, and $\parallel$ means that A and B are parallel. (4)($X_{min}(A, B)$, $Y_{min}(A, B)$, $Dir(A, B)$): the geometry relation of rectangles A and B.

Then, the *geometry relationship* of a pair of dependent rectangles can be described by the three tuple ($X_{min}(A, B)$, $Y_{min}(A, B)$, $Dir(A, B)$). For example, the geometry relationship of rectangles A and B in Figure 6(a) is denoted as $(0, 1, \parallel)$. Note that the geometry relationship of the two rectangles in Figure 6(b) is the same as that in Figure 6(a) despite the different geometry scenarios. Because the minimum boundary distance and the directions of the two rectangles are the same in Figures 6(a) and (b), the optimal color assignments are also the same. Thus, we classify the two geometry scenarios into the same category of geometry relationship.

Since the number of dependent rectangles of a target rectangle is limited (implied by Theorem 1), all the potential overlay scenarios of a pair of dependent rectangles can be enumerated according to all types of geometry relationships, as shown in Figure 7. Then, we have the following theorem:

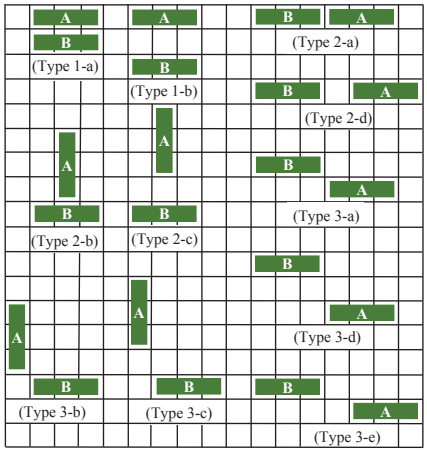

Figure 7: The identified potential overlay scenarios.

Table 1: Notation of color assignment.

Notation	CC	CS	SC	SS
color	A core, B core	A core, B second	A second, B core	A second, B second

Table 2: The color rules and the number of induced side overlays of the identified potential overlay scenarios.

Type	1-a	1-b	2-a	2-b	3-a	3-b	3-c	3-d
Color Rule	CS,SC	CC,SS	CC,SS	CC,SS	CS,SC	SS	CC,SS,SC	CC,SS
min SOL	0	0	0	1	0	0	0	0
max SOL	$2 \cdot L$	L	2	2	1	1	1	1

THEOREM 2. The potential overlay identification scenarios are complete for two dependent rectangle patterns.

According to Theorem 2, there are eleven potential overlay scenarios for two rectangle patterns, and every potential overlay scenario has at most 2^2 color assignments. We enumerate all the color assignments for every potential overlay scenario. Table 2 summarizes the enumeration. "Color rule" is the color assignments that minimize side overlays in every potential overlay scenario, and the notation of color assignments is shown in Table 1; "min SOL" represents the total length of side overlays the scenario induces if the optimal color rule is followed; "max SOL" represents the maximum total length of side overlays the scenario induces if the proposed color rule is not followed. Because Types 2-c, 2-d, and 3-e do not incur any side overlays, the three scenarios are not considered. Below we use an example to explain why the patterns in Figure 7 are potential overlay scenarios. Type 2-b in Figure 7 is a potential overlay scenario because if pattern A is a second pattern and pattern B is a core pattern, the assistant core patterns for pattern A must be merged with pattern B, which results in side overlays of total length equal to 2 on pattern B, as shown in Figure 5(c). An alternative assignment is to assign both pattern A and pattern B to the core mask, which results in a unit of side overlay on pattern B, as shown in Figure 5(d). To minimize the total length of side overlays, the latter assignment is preferred.

The above analysis is for two rectangle patterns. We can generalize the analysis to a pair of polygonal features as follows:

THEOREM 3. The potential overlay identification scenarios are complete for any pair of dependent rectilinear polygons.

Figure 8(a) shows an example, where feature A (respectively, B) is fragmented into A1 and A2 (respectively, B1 and B2). In dashed box 1, A2 and B1 conform to the Type 1-a potential overlay scenario. In box 2, B1 and C conform to the Type 2-b potential

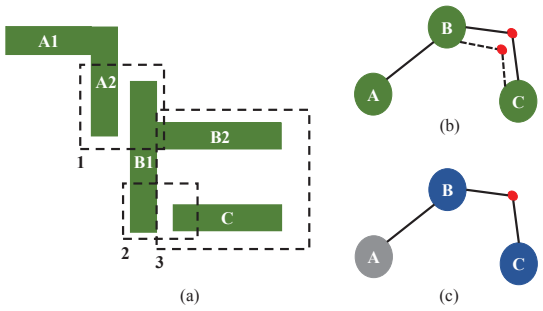

Figure 8: (a) Features A and B are fragmented into rectangles, and three potential overlay scenarios are identified. (b) The overlay constraint graph of (a). (c) The optimally colored overlay constraint graph.

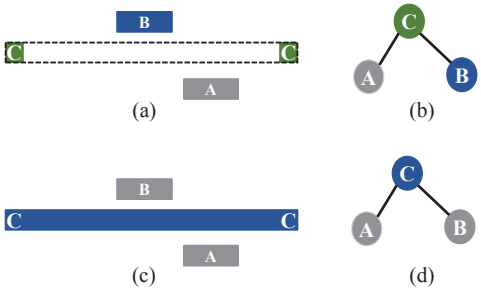

Figure 10: (a)(b) With nets A and B being routed and color assigned, net C cannot be routed with the shortest path. (c)(d) By flipping the color of net B to a second pattern, net C can be routed with the shortest path.

overlay scenario. In box 3, C and B2 conform to the Type 1-b potential overlay scenario. As a result, features A, B, and C induce three potential overlay scenarios and should be colored according to the color rules to minimize side overlays. Note that the Type 1-a and Type 1-b color rules are *hard constraints*, because the violations of the rules result in hard overlays, which are strictly forbidden.

3.2 Overlay Constraint Graph

To handle all the overlay scenarios during the routing stage, we propose a novel overlay constraint graph $G(V, E)$ that captures all the scenarios of overlays. In the constraint graph, every vertex represents a routed net and every edge represents a color rule for a specific potential overlay scenario. There are six different edges, as shown in Figures 9(a)–(f). The bold lines represent hard constraints and the dashed lines represent non-hard constraints. The straight lines in Figures 9(a) and (c) imply that the two vertices should be assigned different colors. The lines with dummy vertices (the red vertices) in Figures 9(b) and (d) indicate that vertices A and B should be assigned the same color. The line with double arrows in Figure 9(e) represents the Type 3-b scenario, where both vertices should be the second pattern. The line with one arrow in Figure 9(f) represents the Type 3-c scenario, where only the color assignment CS is not allowed. In an overlay constraint graph, dummy vertices are used to identify odd cycles composed of hard constraint edges, which causes hard constraint violations. To produce routing results without any hard constraint violation, odd cycles composed of hard constraint edges(see Figure 9(g)) are strictly forbidden.

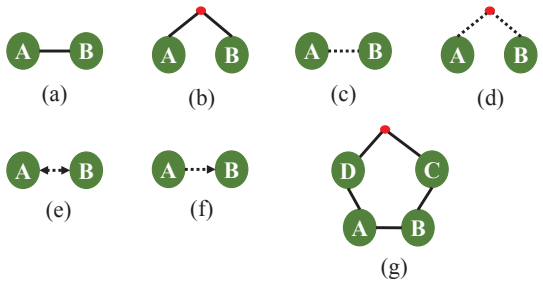

Figure 9: (a) A and B have to be in different colors (hard constraint). (b) A and B have to be in the same color (hard constraint). (c) A and B should be in different colors (non-hard constraint). (d) A and B should be in the same color (non-hard constraint). (e) Both A and B should be the second pattern (non-hard constraint). (f) Only the color assignment CS is not allowed (non-hard constraint). (g) A hard overlay odd cycle.

Two vertices connected by an edge means that the two vertices form a potential overlay scenario. It is possible that two vertices are connected by two or more edges, which represents that the two patterns induce more than one potential overlay scenario. For example, Figure 8(b) shows the overlay constraint graph of Figure 8(a), where a Type 1-b edge and a Type 2-b edge are between vertex B and vertex C, and a Type 1-a edge is between vertex B and vertex A. According to the overlay constraint graph, vertices A and B should be assigned different colors, and vertices B and C should be assigned the same color. Figure 8(c) shows an optimal coloring result. Note that in Figure 8(b), the non-hard dashed edge between vertices B and C is redundant because there exists a hard bold edge between the two vertices. Therefore, the non-hard edge is removed, as shown in Figure 8(c).

3.3 Linear-Time Color Flipping Algorithm

In this subsection, we propose a linear-time color flipping algorithm that finds an optimal color assignment on a given overlay constraint graph under the condition that no cycle exists in the overlay constraint graph. In most previous studies on SADP-aware detailed routing [2, 3, 11], colors of nets are determined when nets are routed. However, fixing the colors of routed nets reduces the flexibility of routing, which may increase the wirelength, the total length of side overlays, and the number of conflicts. In Figure 10(a), for example, nets A and B are routed and assigned to be a second pattern and a core pattern respectively. With fixed colors of nets A and B, net C cannot be routed with the shortest path because it causes conflicts with either net A or net B, as shown in Figure 10(b). To complete the routing, net C has to detour, which increases the wirelength and may incur more overlays and conflicts. In the worst case, net C would fail to find a conflict-free route. However, net C can be routed in the shortest path without inducing any conflict and overlay by flipping the color of net B, as shown in Figures 10(c) and (d). Thus, color flipping provides the router with higher flexibility and results in better solution quality. However, locally flipping the color of a net may cause more overlays and conflicts in a component of an overlay constraint graph. Thus, the main challenge of color flipping is to optimize overlay globally. To achieve the goal, we first extract a tree from every component of an overlay constraint graph, and then propose a linear-time dynamic programming-based algorithm to find an optimal solution of color flipping on the tree.

We first extract a tree from an overlay constraint graph by applying a maximum spanning tree algorithm. In the overlay constraint graph, the cost of a non-hard constraint edge is set to be the total length of side overlays the potential overlay scenarios may incur, and the cost of a hard constraint edge is set to be a constant larger than any cost of non-hard constraint edges. Thus, a tree with most "significant" edges will be selected by running the maximum spanning tree algorithm. Figure 11 gives an example. Figure 11(b) shows the overlay constraint graph of the layout shown in Figure 11(a). In Figure 11(b), vertices B, C, and E form a cycle. By running the maximum spanning tree algorithm, the edge between vertices B and E is removed.

Having an overlay constraint tree $G(V, E)$, a corresponding flipping graph $G'(V', E')$ is constructed. In $G'(V', E')$, every vertex

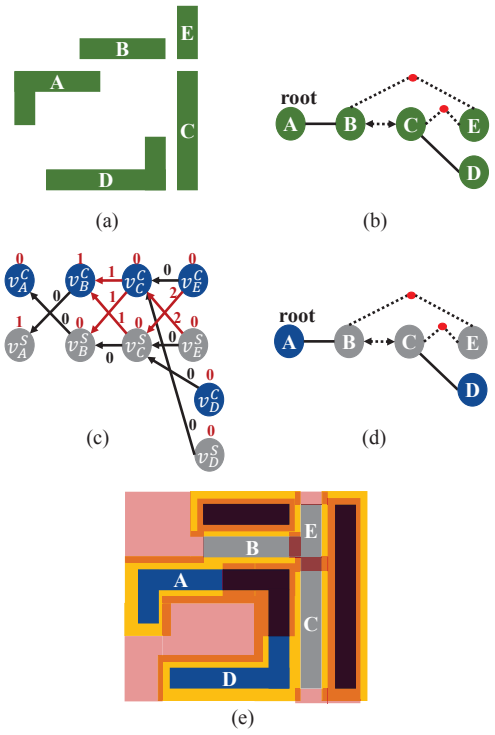

Figure 11: **(a) An example layout of five routed nets. (b) The overlay constraint graph that represents the layout in (a). (c) The flipping graph of the overlay constraint tree of (b), where the edge between C and E is ignored. (d) The color assignment after applying the color flipping algorithm. (e) The layout decomposition result of (a).**

$v_i \in V$ is split into two vertices, v_i^C and v_i^S, where v_i^C represents that net i is assigned to be a core pattern, and v_i^S represents that net i is assigned to be a second pattern. In addition, every edge $(v_i, v_j) \in E$, where v_i is the parent vertex of v_j, has four corresponding directed edges $(v_j^C, v_i^S), (v_j^C, v_i^C), (v_j^S, v_i^S), (v_j^S, v_i^C) \in E'$, which represent four color assignment combinations of v_i and v_j. The cost of each directed edge is set to be the total length of side overlays induced by the color assignment. Figure 11(c) illustrates the flipping graph of the extracted overlay constraint tree from Figure 11(b), where the edge between vertices B and E is ignored.

After the flipping graph construction, finding the optimal color assignments of an overlay constraint tree $G(V, E)$ is equivalent to finding a minimum cost tree $T^*(V_T, E_T)$ on the flipping graph $G'(V', E')$. We find T^* such that for each vertex $v_i \in V$, either $v_i^C \in V_T$ or $v_i^S \in V_T$, and the total edge cost of T^* is minimized. To find such a minimum cost tree, a dynamic programming-based algorithm is applied from leaf vertices to the root vertex on G'. The optimal substructure of the dynamic programming-based algorithm is as follows:

$$Cost_v(v_i^q) = \begin{cases} \sum_{v_j \in children(v_i)} \min_{p \in \{C,S\}} \{Cost_v(v_j^p) + Cost_e(v_j^p, v_i^q)\}, \\ \qquad \text{if } v_i^q \text{ is not a leaf vertex,} \\ 0, \qquad \text{otherwise.} \end{cases} \quad (1)$$

where $p, q \in \{C, S\}$, and $children(v_i)$ is the set of children vertices of v_i. Figure 11(c) shows the calculated costs on each vertex. The optimal color assignment can be found by backtracing the flipping graph from the root vertex with the least cost. The color assignment result of Figure 11(b) is shown in Figure 11(d), whose layout decomposition result is shown in Figure 11(e). Note that there is only a unit of side overlay on the left side of pattern E, which is inevitable. In addition, the odd cycle formed by patterns B, C, and E is solved by applying the merging and cut technique.

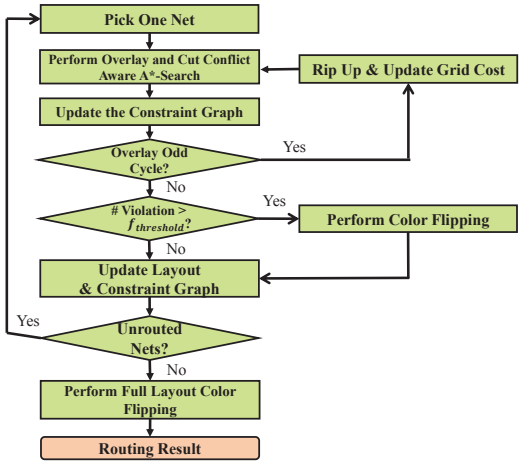

Figure 12: **The flow chart of the overlay-aware detailed routing algorithm.**

The proposed dynamic programming can find an optimal solution, no matter which vertex is chosen as the root.

THEOREM 4. The proposed color flipping algorithm finds an optimal color assignment of an overlay constraint tree in linear time.

3.4 Overall Routing Scheme

In this subsection, we present the overall flow of our overlay-aware detailed routing algorithm. Our router is based on the A*-search algorithm and is guided by the proposed constraint graph. When performing A*-search for a net n, and a path from a grid i to an adjacent gird j is considered, the routing cost of grid j, which is denoted as $C_{\text{grid}}(\text{j})$, can be computed as follows:

$$C_{\text{grid}}(j) = C_{\text{grid}}(i) + \alpha \cdot C_{\text{wl}}(i,j) + \beta \cdot C_{\text{via}}(i,j) + \gamma \cdot T_{2\text{b}}(j),$$

where $C_{\text{wl}}(i,j)$ and $C_{\text{via}}(i,j)$ are the extra wirelength and the number of vias caused by the routing path from grid i to grid j respectively, and $T_{2\text{b}}(j)$ is set to be one if grid j induces a Type 2-b potential overlay scenario with other routed nets. Note that according to Table 2, all potential overlay scenarios except Type 2-b do not induce overlay if colored properly. A Type 2-b potential overlay scenario incurs at least a unit of side overlay regardless of color assignment. Hence, a routing path that induces a Type 2-b potential overlay scenario should be discouraged. α, β, γ are user-defined parameters to adjust weights among wirelength, the via cost, and the Type 2-b cost, respectively.

The overall routing flow is shown in Figure 12. After routing a net, the color of the net would be pseudo-colored with least hard overlay violations and induced overlays, and the overlay constraint graph is updated accordingly. Next, the updated overlay constraint graph is checked whether odd cycles composed of hard constraint edges exist. If there are such odd cycles, the grids that cause the odd cycles would be forbidden for the net, and the net is ripped up and rerouted. If there is no hard overlay odd cycle, color flipping would be performed if the total length of side overlays induced by the newly routed net is longer than a user-defined threshold $f_{threshold}$. After all nets are routed, the color flipping would be performed on the full layout to further minimize overlay.

4. EXPERIMENTAL RESULTS

Our algorithm was implemented in the C++ programming language on a 2.93 GHz Linux work station with 48 GB memory. We compared our results with the works [2, 3, 11]. Because the binary codes of the wroks [2, 11] are currently unavailable, we implemented the algorithms in [2, 11] for comparative studies. The work [2] adopts multiple pin candidate locations, where the source pin and the target pin of every two-pin net have multiple candidate

Table 3: Experimental results on single pin candidate location benchmarks.

Circuit	# Net	Size (μm^2)	[3]				[11]				Ours			
			Rout. (%)	Overlay Length	CPU (s)	#C	Rout. (%)	Overlay Length	CPU (s)	#C	Rout. (%)	Overlay Length	CPU (s)	#C
Test1	1000	6.8 x 6.8	94.1	621	1.3	33	91.4	196	0.6	9	95.3	104	0.4	0
Test2	1800	9.6 x 9.6	95.0	1391	2.2	75	87.5	762	3.5	63	96.7	134	1.6	0
Test3	4000	16.0 x 16.0	96.9	1879	4.1	106	94.7	560	12.0	21	97.2	268	6.0	0
Test4	8000	24.0 x 24.0	97.3	3206	7.8	220	95.6	1048	49.2	28	97.4	503	23.1	0
Test5	12000	36.0 x 36.0	98.8	3211	8.8	200	96.8	972	135.4	25	98.3	424	56.2	0
Comp.			0.9944	7.462	1.157		0.9611	2.807	2.071		1.000	1.000	1.000	

Table 4: Experimental results on multiple pin candidate locations benchmarks.

Circuit	# Net	Size (μm^2)	[2]				Ours			
			Rout. (%)	Overlay Length	CPU (s)	#C	Rout. (%)	Overlay Length	CPU (s)	#C
Test6	1000	6.8 x 6.8	87.3	568	381	0	96.1	209	0.3	0
Test7	1800	9.6 x 9.6	87.8	1064	1435	0	96.6	260	1.4	0
Test8	4000	16.0 x 16.0	87.6	1485	11000	0	96.4	642	5.4	0
Test9	8000	24.0 x 24.0	88.1	2522	47711	0	96.7	1040	24.2	0
Test10	12000	36.0 x 36.0	NA	NA	> 100000	NA	97.8	1044	55.3	0
Comp.			0.7279	2.310	1217		1.000	1.000	1.000	

locations, while the works [3, 11] do not allow multiple pin candidate locations. As a result, we had two sets of benchmarks: (1) benchmarks in which the locations of the source and target pins of each two-pin net are fixed, and (2) benchmarks in which the source and target pins of every two-pin net have multiple candidate locations. Our algorithm can be applied to both sets of benchmarks. The user-defined parameters in Equation (2) are set as follows: $\alpha = \beta = 1$, $\gamma = 1.5$, and the flipping threshold $f_{threshold} = 10$. The design rules of SADP using the cut process are set as follows: $w_{line} = w_{spacer} = w_{cut} = w_{core} = 20$ nm, and $d_{cut} = d_{core} = 30$ nm.

The experimental results are listed in Tables 3 and 4, where "Rout." gives the routability, "Overlay Length" reports the total length of side overlays, "#C" reports the number of cut/trim conflicts, and "CPU" gives the running time in second. Note that "Conflict" represents the number of cut conflicts for the cut process or the number of trim conflicts for the trim process, which are induced by parallel line ends [2, 7].

We first conducted an experiment on the first set of benchmarks. Compared with [3, 11], our algorithm efficiently reduces the total length of side overlays by more than 65%, *with zero cut conflicts* (implying decomposable layouts). Compared with [2] on the second set of benchmarks where the source and target pins of every two-pin net have two candidate locations, our algorithm reduces the total length of side overlays by more than 43% with a $1217X$ speedup and 10% higher routability. The improvement in routability is mainly due to the proposed color flipping algorithm which gives our router more flexibility to access the routing resources. In contrast, the work [2] assigned core and second tracks alternatively, which suffers from restricted routing resources. For example, for a net whose source candidate locations are all in core tracks and target candidate locations all in second tracks, the algorithm in [2] cannot find a legal routing path because the source and target are in tracks of different color.

Compared with the three state-of-the-art studies, the experimental results show that our algorithm can achieve the best quality and efficiency with zero cut conflicts, smallest overlay length, and highest routability.

5. CONCLUSIONS

We have presented the first work that handles all overlay scenarios and applies the cut technique for decomposing odd cycles during SADP-aware routing. We have proposed a novel overlay constraint graph that captures all overlays in a given layout, and a linear-time color flipping algorithm that enhances the routing flexibility. Experimental results have shown that our algorithm can achieve zero cut conflict and the best quality and efficiency among all published works.

6. REFERENCES

[1] Y. Ban, K. Lucas, and D. Pan "Flexible 2D layout decomposition framework for spacer-type double pattering lithography," *Proc. DAC*, pp. 789 - 794, 2011.

[2] Y. Du, Q. Maz, H. Songz, J. Shielyz, G. Luk-Patz, A. Miloslavskyz and M. D. F. Wong, "Spacer-is-dielectric-compliant detailed routing for self-aligned double patterning lithography," *Proc. DAC*, pp. 1–6, 2013.

[3] J.-R. Gao and D. Z. Pan, "Flexible self-aligned double patterning aware detailed routing with prescribed layout planning," *Proc. ISPD*, pp. 25–32, 2012.

[4] C. Kodama, H. Ichikawa, K. Nakayama, T. Kotanib, S. Nojimac, S. Mimotogi, S. Miyamoto, and A. Takahashi, "Self-aligned double and quadruple patterning-aware grid routing with hotspots control," *Proc. ASP-DAC*, pp. 267–272, 2013.

[5] L. Liebmann, "Keynote speech: The escalating design impact of resolution-challenged lithography," *ICCAD*, 2013.

[6] Y.-H. Lin, and Y.-L. Li, "Double patterning lithography aware gridless detailed routing with innovative conflict graph," *Proc. DAC*, pp. 398–403, 2010.

[7] G. Luk-Pat, A. Miloslavskya, and B. Painter, "Design compliance for spacer is dielectric (SID) patterning," *Proc. SPIE*, vol. 8326, pp. 83260D, 2012.

[8] Y. Ma, J. Sweis, C. Bencher, Y. Deng, H. Dai, H. Yoshida, B. Gisuthan, J. Kye, and H. J. Levinson, "Double patterning compliant logic design," *Proc. SPIE*, vol. 7974, pp. 79740D, 2011.

[9] Y. Ma, J. Sweis, H. Yoshida, Y. Wang, J. Kye, and H. J. Levinson, "Self-aligned double patterning (SADP) compliant design flow," *Proc. SPIE*, vol. 8327, pp. 832706, 2012.

[10] M. Mirsaeedi, J. A. Torres, and M. Anis, "Self-aligned double-patterning (SADP) friendly detailed routing," *Proc. SPIE*, vol. 7974, pp. 79740O, 2011.

[11] F. Nakajima, C. Kodama, H. Ichikawab, K. Nakayamaa, S. Nojimaa, T. Kotania, S. Mimotogia and S. Miyamotoa, "Detailed routing with advanced flexibility and in compliance with self-aligned double patterning constraints," *Proc. SPIE*, vol. 8684, pp. 86840A, 2013.

[12] Z. Xiao, Y. Du, H. Tian, and M. D. F. Wong, "Optimally minimizing overlay violation in self-aligned double patterning decomposition for row-based standard cell layout in polynomial time," *Proc. ICCAD*, pp. 32 - 39, 2013.

[13] Z. Xiao, Y. Du, H. Zhang, and M. D. F. Wong, "A polynomial time exact algorithm for overlay-resistant self-aligned double patterning (SADP) layout decomposition," *IEEE TCAD*, vol. 32, no. 8, pp. 1228–1239, 2013.

[14] H. Zhang ,Y. Du, M. D. F. Wong, and R. Topaloglu "Self-aligned double patterning decomposition for overlay minimization and hot spot detection," *Proc. DAC*, pp. 71 - 76, 2011.