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

Iou-Jen Liu, Shao-Yun Fang, *Member, IEEE*, and Yao-Wen Chang, *Fellow, IEEE*

***Abstract*—Self-aligned double patterning (SADP) is one of the most promising techniques for sub-20 nm 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, this paper 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.**

***Index Terms*—Algorithms, design, manufacturability, performance, routing, self-aligned double pattering (SADP) lithography.**

I. Introduction

BEFORE next generation lithography technologies such as electron beam lithography and extreme ultraviolet lithography are ready, double patterning with 193 nm 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-20 nm technology nodes.

Manuscript received February 21, 2015; revised July 9, 2015 and November 11, 2015; accepted December 23, 2015. Date of publication December 30, 2015; date of current version August 18, 2016. This work was supported in part by Genesys Logic, in part by IBM, in part by MediaTek, in part by TSMC, in part by Academia Sinica, in part by the MOST of Taiwan under Grant NSC 102-2221-E-002-235-MY3, Grant NSC 102-2923-E-002-006-MY3, Grant MOST 103-2221-E-002-259-MY3, Grant MOST 103-2812-8-002-003, Grant MOST 104-2221-E-002-132-MY3, and in part by the National Taiwan University (NTU) under Grant NTU-ERP-104R8951 and Grant NTU-ERP-105R8951. A preliminary version of this paper was presented at the Proceedings of ACM/IEEE Design Automation Conference in June 2014 [1]. This paper was recommended by Associate Editor C. C.-N. Chu.

I.-J. Liu is with the Department of Electrical and Computer Engineering, University of Illinois at Urbana–Champaign, Urbana, IL 61801 USA (e-mail: yrliu@eda.ee.ntu.edu.tw).

S.-Y. Fang is with the Department of Electrical Engineering, National Taiwan University of Science and Technology, Taipei 106, Taiwan (e-mail: syfang@mail.ntust.edu.tw).

Y.-W. Chang is with the Department of Electrical Engineering and Graduate Institute of Electronics Engineering, National Taiwan University, Taipei 106, Taiwan (e-mail: ywchang@ntu.edu.tw).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TCAD.2015.2513670

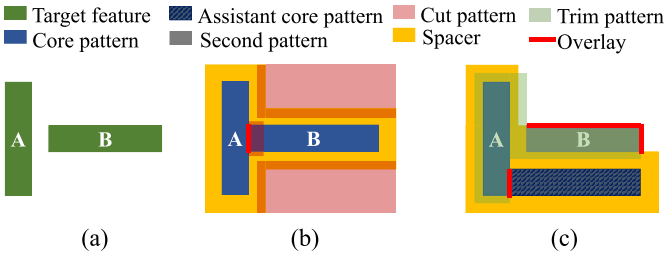

Fig. 1. (a) Target layout. SADP using the (b) cut process and (c) trim process.

Compared to litho-etch-litho-etch (LELE) double patterning, SADP has better critical dimension control due to its intrinsic self-aligned property [2].

In an SADP process, a layout is decomposed into two masks, a core mask and a trim/cut mask. Different from 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: 1) the SADP cut process using a cut mask [3] and 2) the SADP trim process using a trim mask [2]. 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. Fig. 1(b) shows the layout decomposition result of a given layout shown in Fig. 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 Fig. 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 Fig. 1(b), a section of the right boundary of pattern A induces an overlay. Also, in Fig. 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 [4]. Note that assist core patterns provide a protecting spacer for second patterns and effectively reduce overlays. Consequently, many existing layout decomposition algorithms use assist core patterns for overlay minimization [5]–[9].

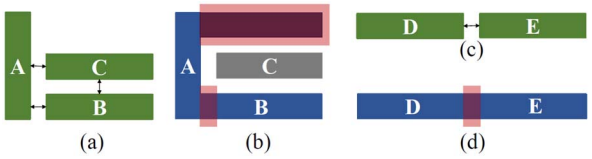

Fig. 2. SADP using the cut process provides higher design flexibility. (a) and (b) Odd cycle can be decomposed by merging patterns A and B, and then applying a cut pattern to them. (c) and (d) Patterns placed in a tip-to-tip manner can be assigned the same color.

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 illustrated in Fig. 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 two-colorable, the layout is not decomposable with the trim process [2], [10], [11]. Odd cycles, however, can be decomposed in the SADP cut process. As shown in Fig. 2(b), the odd cycle is decomposed by merging patterns A and B and assigning the remaining two patterns to different masks (different colors). Then, a cut pattern is used to separate patterns A and B [3], [5], [12], [13]. 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 Fig. 2(c) and (d).

Since the SADP decomposability and the overlay controllability of an arbitrary layout are quite restricted, an SADP-aware detailed router is desirable. Some SADP-aware detailed routing algorithms have been proposed for the trim process [10], [11], [14], [15] and for the cut process [16]. Du *et al.* [10] proposed a graph model to capture the decomposition violations, and Gao and Pan [11] performed routing and layout decomposition simultaneously. 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 [16] targets on SADP using the cut process. Nevertheless, the work [16] fails to apply the cut technique to solve odd cycles, and the merger of core patterns and assistant core patterns results in severe overlays. Consequently, this paper is the first work that applies the cut technique for solving odd cycles during the routing stage.

In this paper, 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 major contributions of this paper are summarized as follows.

1) This paper proposes the first work that applies the merge technique to decompose odd cycles of layout patterns during the routing stage.
2) The identified overlay scenarios are complete, which is the foundation for systematical overlay minimization.
3) A novel overlay constraint graph is proposed to capture all overlays in a routing result, which provides precise overlay information for our router.
4) We propose a linear-time color flipping algorithm that gives a router more flexibility to utilize routing resources. The proposed color flipping algorithm finds the optimal color assignment of a given layout in linear time under the condition that no cycle exists in the overlay constraint graph.
5) The routing results are guaranteed to be conflict-free and thus are decomposable.
6) 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.

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

II. Preliminaries

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

A. Side Overlay

The major advantage of the SADP processes over the LELE double patterning process is the better overlay control achieved by spacer protection [8]. 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 Fig. 3, two patterns are, respectively, fabricated by the LELE double patterning process [see Fig. 3(a)] and by the SADP cut process [see Fig. 3(b)]. In both cases, patterns A and B are assigned different colors, and the second/cut mask shifts w_o toward left. In Fig. 3(a), the shift of the second mask results in a shorter distance between patterns A and B. In Fig. 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 Fig. 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 noncritical overlays because tip

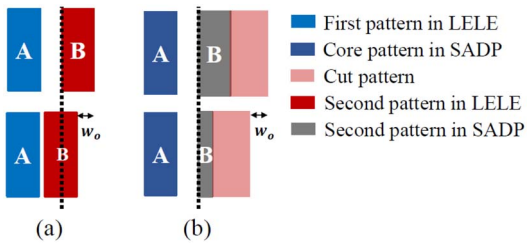

Fig. 3. Misalignment of two masks causes different effects in LELE double patterning and in SADP. (a) Two patterns get closer in LELE double patterning. (b) Second pattern is seriously distorted in SADP.

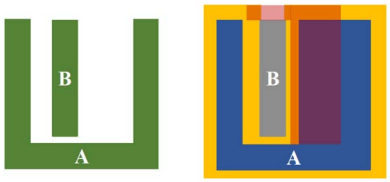

Fig. 4. Side boundary of a feature not protected by a spacer results in a side overlay.

overlays induce little critical dimension change [3], [7], but side overlays should be minimized to reduce yield loss.

B. Design Rules and Cut-Mask Conflict

In semiconductor manufacturing, design rules are set to ensure good manufacturability. The design rules considered in this paper (and previous papers) are listed below.

1) w_{line}: The minimum width of a metal line.
2) w_{spacer}: The width of a spacer, which is equal to the minimum spacing between two metal lines in a grid-based design.
3) w_{cut}: The minimum width of a cut pattern.
4) w_{core}: The minimum width of a core pattern.
5) d_{cut}: The minimum distance between two cut patterns.
6) d_{core}: The minimum distance between two core patterns.
7) $d_{overlap}$: The length that a cut pattern overlaps with a spacer.

In practice, we set the constraint as follows [7]:

$$w_{\text{line}} = w_{\text{spacer}} \tag{1}$$

$$w_{\text{cut}} = w_{\text{core}} < d_{\text{cut}} = d_{\text{core}} \tag{2}$$

$$d_{\text{core}} < w_{\text{line}} + 2w_{\text{spacer}} - 2d_{\text{overlap}}. \tag{3}$$

Core patterns with distance within d_{core} can be merged by applying the merging technique [12], as shown in Fig. 4, where pattern A is merged with the assistant core pattern. Ma *et al.* [12] pointed out that, for SADP, mask rule check (MRC) challenges mostly come from the cut mask [13], [17], and MRC violations occur over a spacer can be ignored because they do not affect the final layout [12]. As shown in Fig. 5(a), the distance between cut patterns is smaller than d_{cut}. Therefore, the printed cut patterns may contain irregular redundant patterns [Fig. 5(b)]. The irregular redundant cut patterns, however, would not affect the final target pattern, because they occur on spacers instead of target patterns. As illustrated in Fig. 5(b), patterns A and B remain intact

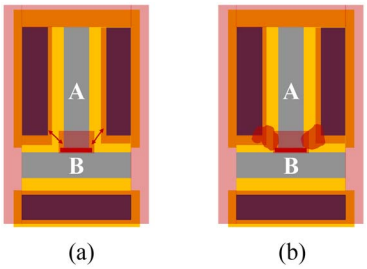

Fig. 5. (a) Distance between cut patterns is smaller than d_{cut}. (b) Patterns A and B remain intact while MRC violations occur on spacers.

while MRC violations occur on spacers. Therefore, 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}) over a target pattern. To produce a decomposable routing result, cut conflicts should strictly be forbidden.

C. Problem Formulation

Ma *et al.* [3], [12] pointed out that tip-to-side overlays (i.e., side overlay whose length is w_{line}) are considered SADP-friendly. By allowing tip-to-side overlays, many complex patterns, such as odd cycles, can be decomposed. Therefore, we define a side overlay whose length is longer than w_{line} as a hard overlay; otherwise, it is a nonhard overlay. To simultaneously consider design flexibility and manufacturability, we allow nonhard 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 nonhard overlays such that no cut conflict, hard overlay, and design rule violation occur.

III. Overlay-Constraint Graph Guided SADP-Aware Detailed Routing

In this section, the identified potential overlay scenarios, the overlay constraint graph, the linear-time color flipping algorithm, the cut conflict removal methods, and the overall routing scheme are presented in the following sections.

A. 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.

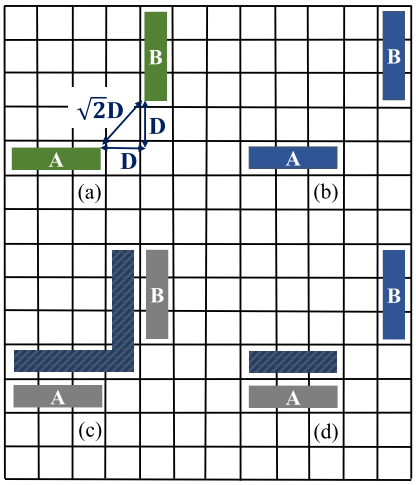

Fig. 6. $D = w_{\text{line}} + 2w_{\text{spacer}}$. (a) Two patterns with distance $\sqrt{2} \cdot (w_{\text{line}} + 2 \cdot w_{\text{spacer}})$. Patterns A and B can be assigned to (b) core patterns simultaneously, because $\sqrt{2} \cdot (w_{\text{line}} + 2 \cdot w_{\text{spacer}}) > d_{\text{core}}$ and (c) second patterns simultaneously, because there are enough space for the necessary assistant core patterns. (d) Pattern A is second pattern and pattern B is core pattern. The distance between pattern B (core) and the assistant core pattern for pattern A is $\sqrt{(w_{\text{line}} + 2w_{\text{spacer}})^2 + w_{\text{spacer}}^2}$, which must not be less than d_{core}. Hence, no overlay is induced.

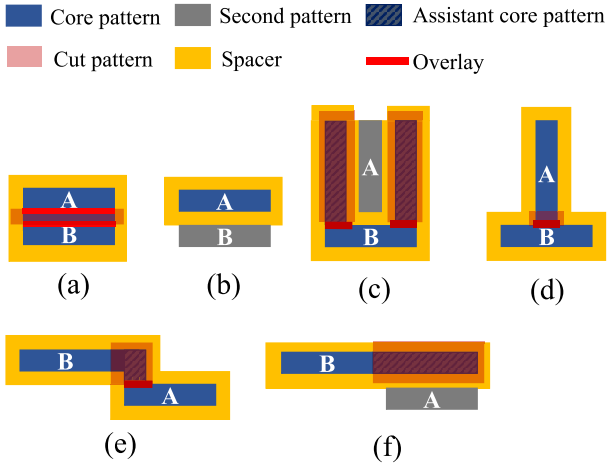

Fig. 7. Assigning patterns A and B (a) and (b) different colors would not induce any overlay, (c) and (d) same color would induce one side overlay, and (e) and (f) different colors would not induce any side overlay.

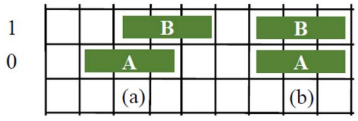

Fig. 8. (a) and (b) have the same optimal color assignments even though their geometry scenarios are different.

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 $d_{\text{indep}} = \sqrt{2} \cdot (w_{\text{line}} + 2 \cdot w_{\text{spacer}})$.

Proof: Suppose there are two patterns with distance $\sqrt{2} \cdot (w_{\text{line}} + 2 \cdot w_{\text{spacer}})$ [Fig. 6(a)]. Obviously, the two patterns can be simultaneously assigned to the core mask, because $\sqrt{2} \cdot (w_{\text{line}} + 2 \cdot w_{\text{spacer}}) > d_{\text{core}}$ [Fig. 6(b)]. If the two patterns are both second patterns, there is enough space between the two patterns to form the necessary assist core patterns without applying the merging technique with core patterns. Therefore, no overlay is induced [Fig. 6(c)]. In addition, if one pattern is a core pattern and the other is a second pattern, the distance between the core pattern and the assistant core pattern for the second pattern is no less than $\sqrt{(w_{\text{line}} + 2w_{\text{spacer}})^2 + w_{\text{spacer}}^2}$, which must not be less than d_{core}. Hence, the core pattern and the assist core pattern can simultaneously exist without merging, and thus no overlay is induced [Fig. 6(d)]. Moreover, for a pair of parallel patterns, if the distance between them is larger than d_{indep}, the distance between them is at least twice the width of a routing track. Obviously, the distance between them would allow any color combination without inducing any side overlay. ■

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 Fig. 7(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.

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) $\text{Dir}(A, B)$: The directions of two rectangles A and B. $\perp$ means A and B are orthogonal, and $\parallel$ means A and B are parallel.

Then, the geometry relationship of a pair of dependent rectangles can be described by the three tuple $(X_{\min}(A, B),\ Y_{\min}(A, B),\ \text{Dir}(A, B))$. For example, the geometry relationship of rectangles A and B in Fig. 8(a) is denoted as $(0, 1, \parallel)$. Note that the geometry relationship of the two rectangles in Fig. 8(b) is the same as that in Fig. 8(a) despite the different geometry scenarios. Because the minimum boundary distance and the directions of the two rectangles are the same in Fig. 8(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 Fig. 9. Then, we have the following theorem.

Theorem 2: The identified potential overlay scenarios are complete for two dependent rectangle patterns.

Proof: By Theorem 1, proving Theorem 2 is equivalent to proving the identified potential overlay scenarios

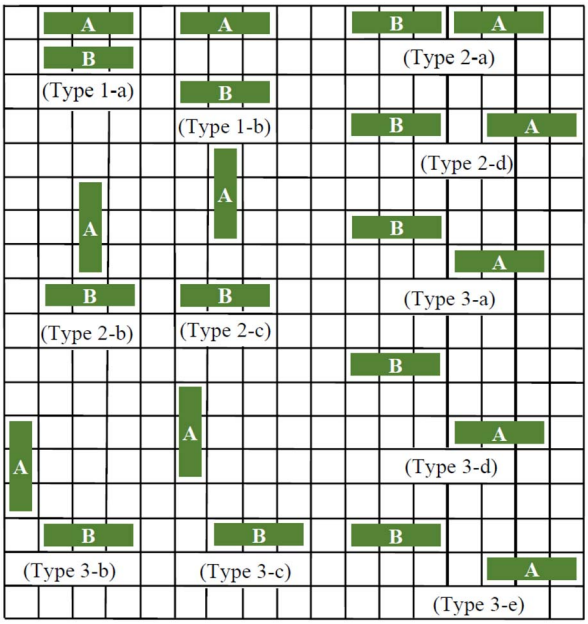

Fig. 9. Identified potential overlay scenarios.

TABLE I
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

include all the geometry relationships of a pair of dependent rectangle patterns. The geometry relationships of two rectangles, rectangles A and B, can be divided into two categories.

1) $X_{\min}(A, B) = 0$ *or* $Y_{min}(A, B) = 0$*:* In this case, the distance between A and B is larger than d_{indep} if $Y_{\min}(A, B) \geq 3$ or $X_{\min}(A, B) \geq 3$. Therefore, there are totally eight geometry relationships of a pair of dependent rectangles: $(0, 1, \text{Dir}(A, B))$, $(0, 2, \text{Dir}(A, B))$, $(1, 0, \text{Dir}(A, B))$, and $(2, 0, \text{Dir}(A, B))$, where $\text{Dir}(A, B)$ is either $\perp$ or $\parallel$.
2) $X_{min}(A, B) > 0$ *and* $Y_{min}(A, B) > 0$*:* Under this circumstance, the distance between A and B is larger than d_{indep} if $Y_{\min}(A, B) \geq 2$ and $X_{\min}(A, B) \geq 2$. Thus, there are totally six geometry relationships of a pair of dependent rectangles: $(1, 1, \text{Dir}(A, B))$, $(1, 2, \text{Dir}(A, B))$, and $(2, 1, \text{Dir}(A, B))$, where $\text{Dir}(A, B)$ is either $\perp$ or $\parallel$.

Note that $(x, y, \perp)$ is equivalent to $(y, x, \perp)$. For example, $(1, 2, \perp)$ is equivalent to $(2, 1, \perp)$. After excluding the equivalent scenarios, there are six dependent geometry relationships in 1), which are corresponding to types 1 and 2 in Fig. 9, and five dependent geometry relationships in 2), which are corresponding to type 3 in Fig. 9. ■

According to Theorem 2, there are 11 potential overlay scenarios for two rectangle patterns, and every potential overlay scenario has at most 2^2 color assignment. We enumerate all the color assignments for every potential overlay scenario, as shown in Figs. 23–34. Table II summarizes the enumeration. "color rule" is the color assignments that minimize side overlays in every potential overlay scenario, and the notation of

TABLE II
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 SO	0	0	0	1	0	0	0	0
max SO	$2 \cdot L$	L	2	2	1	1	1	1

L represents the length of overlaps between two parallel rectangle patterns.
SO represents the length of side overlays.

color assignments is shown in Table I; "min SO" represents the total length of side overlays the scenario induces if the optimal color rule is followed; "max SO" 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 induce any side overlays, the three scenarios are not considered. Below we use two examples to explain why the patterns in Fig. 9 are potential overlay scenarios. Type 2-b in Fig. 9 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 Fig. 7(c). An alternative assignment is to assign both patterns A and B to the core mask, which results in a unit of side overlay on pattern B, as shown in Fig. 7(d). To minimize the total length of side overlays, the latter assignment is preferred. In addition, type 3-a in Fig. 9 is a potential overlay scenario because if both patterns A and B are core pattern, it induces a unit of side overlay on pattern A, as shown in Fig. 7(e). On the other hand, if patterns A and B are assigned different colors, no side overlay will be induced between them, as shown in Fig. 7(f).

The above analysis is for two rectangle patterns. For a pair of polygonal features, we have the following theorem.

Theorem 3: The identified potential overlay scenarios are complete and can be applied to any pair of dependent rectilinear polygons.

Proof: Every rectilinear polygon is first fragmented into rectangle patterns. After that, if two dependent rectangles belong to the same rectilinear polygon, no overlay will be induced between the pair of rectangles since they are made of the same core pattern or the same second pattern. In contrast, if two dependent rectangles belong to different rectilinear polygons, the identified potential overlay scenarios are complete for the two rectangles by Theorem 2. Therefore, we can conclude that the identified potential overlay scenarios are complete and can be applied to any pair of dependent rectilinear polygons. ■

Fig. 10(a) shows an example, where features A and B are fragmented into A1, A2, 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 overlay scenario. In box 3, C and B2 conform to the type 1-b potential overlay scenario. Therefore, 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 1-b color rules are hard constraints, because the violations of the type 1-a and 1-b color rules result in hard overlays, which are strictly forbidden.

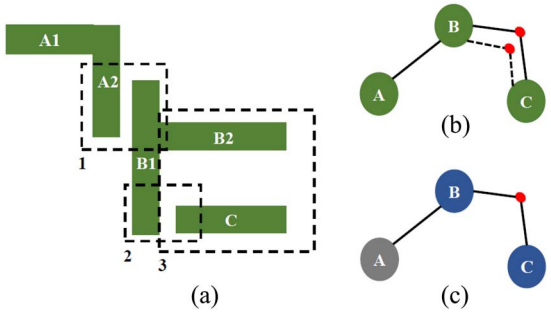

Fig. 10. (a) Features A and B are fragmented into rectangles, and three potential overlay scenarios are identified. (b) Overlay constraint graph of (a). (c) Optimally colored overlay constraint graph.

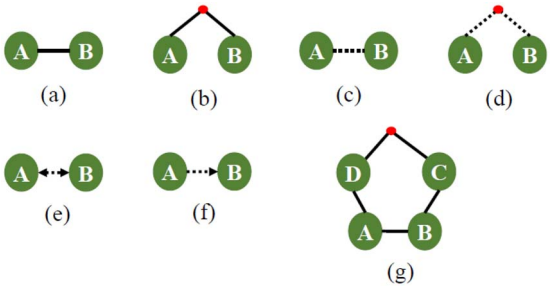

Fig. 11. A and B have to be in (a) different colors (hard constraint) and (b) same color (hard constraint). A and B should be in (c) different colors (nonhard constraint) and (d) same color (nonhard constraint). (e) Both A and B should be the second pattern (nonhard constraint). (f) Only the color assignment CS is not allowed (nonhard constraint). (g) Hard overlay odd cycle.

B. 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 Fig. 11(a)–(f). The bold lines represent hard constraints and the dashed lines represent nonhard constraints. The straight lines in Fig. 11(a) and (c) represent that two vertices should be assigned different colors. The lines with dummy vertices (the red vertices) in Fig. 11(b) and (d) indicate that vertices A and B should be assigned the same colors. The line with double arrows in Fig. 11(e) represents type 3-b, where both two vertices should be the second pattern. The line with one arrow in Fig. 11(f) represents type 3-c, where only the color assignment CS is not allowed.

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, Fig. 10(b) shows the overlay constraint graph of Fig. 10(a), where a type 1-b edge and a type 2-b edge are between vertices B and C, and a type 1-a edge is between vertices B and 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 colors. Fig. 10(c) shows an optimal coloring result. Note that in Fig. 10(b), the nonhard dashed edge between vertices B and C is redundant because there exists a hard bold edge between the two vertices. Therefore, the nonhard edge is removed, as shown in Fig. 10(c).

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 are strictly forbidden. In Fig. 11(g), vertices A, B, C, and D and a dummy vertex form an odd cycle composed of five hard constraint edges, and thus there is no legal color assignment without hard constraint violations. We extend the constant-time odd cycle detection method for LELE double patterning conflict cycle detection from [18] on our overlay constraint graph to efficiently detect hard constraint violations.

In addition, because the coloring of even cycles composed of the same-type hard constraint edges are trivial, we reduce the even cycles into two super vertices for further graph reduction. For example, the vertices that will be in the same color in the even cycle shown in Fig. 12 are merged into a super vertex. The overlay constraint graph keeps updating during the sequential routing process, which gives our router precise and complete overlay information.

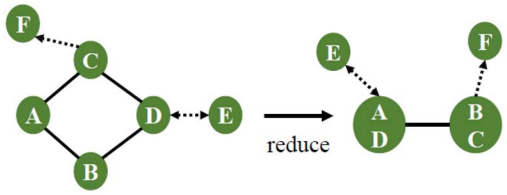

Fig. 12. Even cycle reduction. Since C/A and B/D must be assigned the same color, C/A and B/D are merged into a super node.

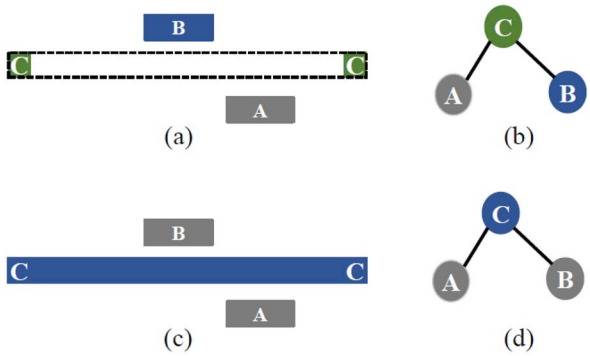

Fig. 13. (a) and (b) With nets A and B routed and color assigned, net C cannot route with the shortest path. (c) and (d) By flipping the color of net B to a second pattern, net C can route with the shortest path.

C. Linear-Time Color Flipping Algorithm

In this section, we propose a linear-time color flipping algorithm that finds the 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 [10], [11], [16], 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 Fig. 13(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 route with the shortest path because it causes conflicts with either nets A or B, as shown in Fig. 13(b). To complete the routing, net C has to detour, which increases the wirelength and may induce 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 Fig. 13(c) and (d). Thus, color flipping provides the router more 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 we 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 nonhard constraint edge is set to be the total length of side overlays the potential overlay scenarios may induce, and the cost of a hard constraint edge is set to be a constant larger than any cost of nonhard constraint edges. Thus, a tree with most "significant" edges will be selected by running the maximum spanning tree algorithm. Fig. 14 gives an example. Fig. 14(b) shows the overlay constraint graph of the layout shown in Fig. 14(a). In Fig. 14(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 $v_i \in V$ 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. Fig. 14(c) illustrates the flipping graph of the extracted overlay constraint tree from Fig. 14(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 the leave vertices to the root vertices on G'. The optimal substructure of the dynamic programming-based algorithm is as follows:

$$\text{Cost}_v\left(v_i^q\right) = \begin{cases} \sum_{v_j \in \text{children}(v_i)} \min\limits_{p \in \{C,S\}} \left\{ \text{Cost}_v\left(v_j^p\right) + \text{Cost}_e\left(v_j^p, v_i^q\right) \right\} & \text{if } v_i^q \text{ is not a leaf vertex} \\ 0, & \text{otherwise} \end{cases} \tag{4}$$

where $p, q \in \{C, S\}$, and children(v_i) is the set of children vertices of v_i. Fig. 14(c) shows the calculated costs on each vertex. The optimal color assignment solution can be found by backtracing the flipping graph from the root vertex with the least cost. The color assignment result of Fig. 14(b) is shown in Fig. 14(d), whose layout decomposition result is shown in Fig. 14(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. The proposed dynamic programming can find an optimal solution, no matter which vertex in the overlay constraint graph is chosen as the root.

Theorem 4: The proposed color flipping algorithm finds the optimal color assignments of an overlay constraint tree in linear time.

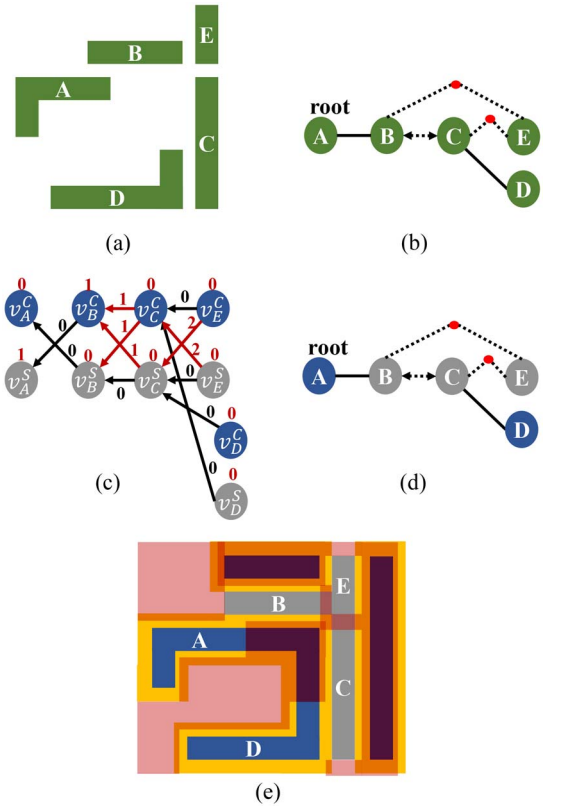

Fig. 14. (a) Example layout of five routed nets. (b) Overlay constraint graph that represents the layout in (a). (c) Flipping graph of the overlay constraint tree of (b), where the edge between C and E is ignored. (d) Color assignment after applying the color flipping algorithm. (e) Layout decomposition result of (a).

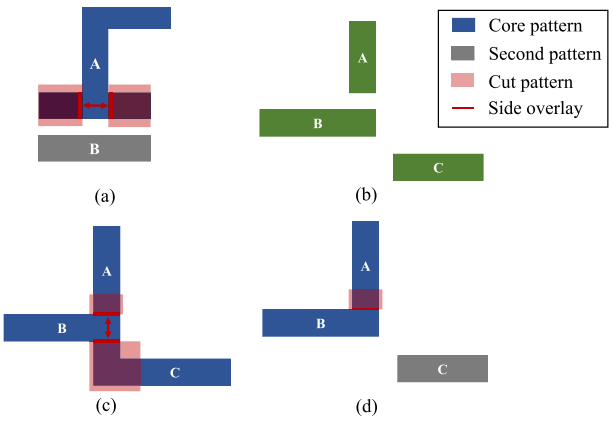

Fig. 15. (a) Example of type A cut conflicts. (b) Given layout with three target patterns. (c) Example of type B cut conflicts. (d) Cut conflict is removed during the overlay minimization process.

Proof: First, we prove that the cost of a vertex v_i^q computed by (4) is the cost of a subtree rooted at v_i^q, $q \in \{C, S\}$. If v_i^q is a leaf vertex in the flipping graph G', the cost of a subtree rooted at v_i^q is zero because no edge is in the subtree. Suppose v_i^q is not a leaf vertex in G', $v_1, v_2, \ldots, v_n \in \text{children}(v_i)$, and the costs of $v_1^p, v_2^p, \ldots, v_n^p$ are the respective costs of $2n$ subtrees rooted at $v_1^p, v_2^p, \ldots, v_n^p$, $p \in \{C, S\}$. According to (4), the cost of v_i^q is the sum of n subtree costs and n edge costs. Because the n edges are connecting edges between the n subtrees and v_i^q, the cost of v_i^q is the cost of a subtree rooted at v_i^q.

Then, we prove that a tree rooted at v_i^q and found by using (4) has the minimum cost. If v_i^q is a leaf on G', $\text{Cost}_v(v_i^q)$ is zero, which is minimized. Suppose v_i^q is not a leaf vertex on G', $v_1, v_2, \ldots, v_n \in \text{children}(v_i)$, and $\text{Cost}_v(v_1^p)$, $\text{Cost}_v(v_2^p), \ldots, \text{Cost}_v(v_n^p)$ are minimized. Assume $\text{Cost}_v(v_i^q)$ computed according to (4) is not minimum, there is at least one $v_j \in \text{children}(v_i)$ such that $\min_{p\in\{C,S\}}\{\text{Cost}_v(v_j^p) + \text{Cost}_e(v_j^p, v_i^q)\}$ is not minimum. Because $\text{Cost}_e(v_j^p, v_i^q)$ is fixed for any $p \in \{C, S\}$, either $\text{Cost}_v(v_j^C)$ or $\text{Cost}_v(v_j^S)$ is not minimum, which causes a contradiction. Thus, a tree rooted at v_i^q and found by using (4) has the minimum cost.

The complexity of the algorithm is dominated by the cost computing process. By (4), the minimum cost of every vertex $v \in V'$ can be found by checking all children of v. By applying the dynamic programming-based algorithm from leafs to the root, every vertex and edge is checked only once, and thus the time complexity is $O(E + V)$. ■

D. Cut Conflict Removal

In this section, we present our cut conflict removal methods. There are two types of cut conflicts: 1) a cut pattern violating the minimum width rule (w_{cut}) and 2) a pair of cut patterns violating the minimum distance rule (d_{cut}) over a target pattern. Because the width of all cut patterns in our algorithms are set to be larger than or equal to w_{cut}, no minimum width cut conflict would occur.

For minimum distance cut conflicts, we categorize them into the following two scenarios.

1) *Type A Conflicts:* Cut conflicts induced by a pair of patterns.

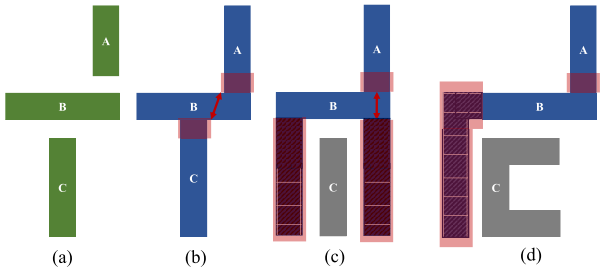

Fig. 16. (a) Given layout. (b) and (c) Cut conflict occurs over pattern B no matter what colors are assigned. (d) Net C is ripped up and rerouted to remove the cut conflict.

2) *Type B Conflicts:* Cut conflicts induced by more than two patterns.

A pair of patterns that induce type A conflicts would fall into one of the identified potential overlay scenarios. Therefore, we can forbid type A conflicts by forbidding the color combinations that would induce cut conflicts directly on the overlay constraint graphs. As shown in Fig. 15(a), patterns A and B form a type 1-b potential overlay scenario. From the enumerated color assignments of potential overlay scenarios (Fig. 27), we can find that if patterns A and B are assigned to core pattern and second pattern, respectively (CS), one cut conflict would occur. Therefore, the color assignment CS should be forbidden. Hence, cut conflicts would not occur.

Type B conflicts are more complicated than type A conflicts. Therefore, we do not handle them directly on overlay constraint graphs. Type B conflicts occur in the scenarios where two parallel boundary sections of a target pattern are defined directly by two cut patterns. Hence, by minimizing the number of target pattern boundaries that are defined directly by cut patterns (i.e., side overlays), type B conflicts can be controlled effectively without any additional computational efforts. As shown in Fig. 15(b) and (c), if all the three patterns are assigned to core patterns, a type B cut conflict would occur over pattern B. During our overlay minimization process, pattern C would be reassigned to second pattern to reduce overlays. As a result, because one of the cut patterns is removed during the overlay minimization process, the cut conflict is solved [Fig. 15(d)].

In addition, to further remove type B conflicts, a cut conflict check scheme would be performed after each net is routed. With potential overlay scenario identification, we can easily locate the cut patterns caused by routed nets. We refer to cut patterns that directly define edges of target patterns as critical cut patterns. Note that only critical cut patterns may induce cut conflicts. After a net is routed, therefore, only the critical cut patterns generated by the newly routed net have to be checked. If a newly routed net would induce cut conflicts with original nets whatever color it is assigned to, the new net would be ripped up and rerouted to avoid cut conflicts. As shown in Fig. 16(b) and (c), a cut conflict would occur over pattern B no matter what colors are assigned. To solve the cut conflict, net C is ripped up and rerouted. As a result, the cut conflict is removed at the expense of longer wirelength [Fig. 16(d)]. (In the experiments, the maximum iterations of rip-up and reroute of a net is set to three.)

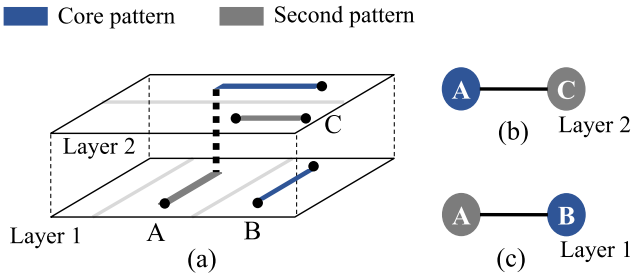

Fig. 17. (a) Net A is assigned to second pattern and core pattern in layers 1 and 2, respectively. (b) Layer 2 overlay constraint graph. (c) Layer 1 overlay constraint graph.

E. Overall Routing Scheme

In this section, 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 $C_{\text{grid}}(j)$, can be computed as follows:

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

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 $T_{2b}(j)$ 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 II, all potential overlay scenarios except type 2-b do not induce overlay if colored properly. A type 2-b potential overlay scenario induces 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, and γ are user-defined parameters to adjust weights among wirelength, the via cost, and the type 2-b cost, respectively.

In multilayer layout decomposition, a net can be assigned to different colors in different routing layers. Therefore, in our routing scheme, each routing layer has its corresponding overlay constraint graph. Overlay constraint graphs that correspond to different routing layers are independent. Fig. 17(a) shows an example, where the segment of net A in layer one is assigned to second pattern, and the segment of net A in layer two is assigned to core pattern. Because two routing layers are used, we maintain two independent overlay constraint graphs that correspond to the two routing layers, as shown in Fig. 17(b) and (c).

The overall routing flow is shown in Fig. 18 and summarized in Fig. 19. After routing a net, the overlay constraint graphs are updated accordingly (lines 4 and 5). Next, the updated overlay constraint graphs are checked whether odd cycles composed of hard constraint edges exist. In addition, the cut conflict check scheme is performed. If there are such odd cycles or cut conflicts, the grids that cause the odd cycles and cut conflicts would be set to a high cost for the net, and the net is ripped up and rerouted (lines 6–9). Then, the color of the net would be pseudo-colored with least hard overlay violations and induced overlay, which is executed in line 11. Color flipping would be performed if the total length of side overlays induced by the newly routed net is longer than a user-defined

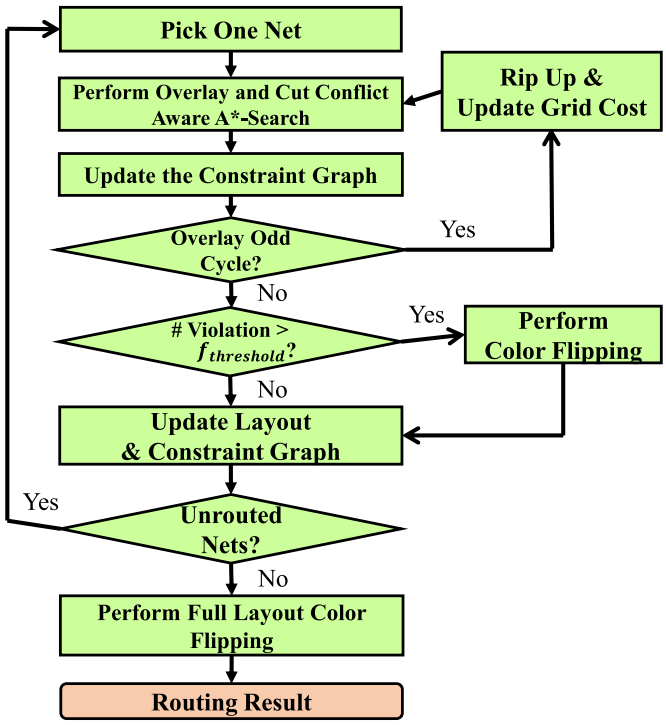

Fig. 18. Flow chart of the overlay-aware detailed routing algorithm.

Algorithm: Overlay-aware Detailed Routing Algorithm
Input: M, /*routing map*/
N, /*net list*/
R, /*a set of design rules*/
Output: M, /*routing map with routed and colored nets*/

```
1  Initialize overlay constraint graph G
2  for ∀ net n_i ∈ N
3      repeat
4          OverlayAwareA*Search(M, n_i, R)
5          UpdateConstraintGraph(G)
6          if IsOverlayOddCycle(G)
               or IsCutConflict(n_i, M)
7              RipUp(n_i, M, G)
8              IncreaseCost(n_i, M)
9          endif
10     until (!IsOverlayOddCycle(G)
               and !IsCutConflict(n_i, M))
               or #iteration > B
11     Pseudocoloring(n_i)
12     if SideOverlay(n_i) > f_threshold
13         ColorFlipping(G, n_i, M)
14     endif
15 endfor
16 ColorFlipping(G, M)
```

Fig. 19. Overlay-aware detailed routing algorithm.

TABLE III
EXPERIMENTAL RESULTS ON SINGLE PIN CANDIDATE LOCATION BENCHMARKS

Circuit	# Net	Size (μm^2)	[11]				[16]				Ours			
			Rout. (%)	Overlay Length	CPU (s)	#C	Rout. (%)	Overlay Length	CPU (s)	#C	Rout. (%)	Overlay Length	CPU (s)	#C
Test1	1500	6.8 x 6.8	94.0	3593	8.5	329	75.4	1519	3.0	76	97.0	187	0.9	0
Test2	2700	9.6 x 9.6	94.0	5769	15.7	531	76.3	2627	11.2	150	96.3	247	1.1	0
Test3	5500	16.0 x 16.0	97.1	10509	37.6	802	79.4	4986	79.4	268	98.0	379	2.4	0
Test4	12000	24.0 x 24.0	97.1	22463	87.1	1654	79.6	10442	430.0	554	98.0	768	5.4	0
Test5	28000	36.0 x 36.0	97.6	43956	146.1	2968	80.8	21282	2397.1	1033	98.2	1458	25.0	0
Comp.			0.9848	25.94	12.018		0.8037	12.0215	44.3160		1.000	1.000	1.000	

TABLE IV
EXPERIMENTAL RESULTS ON MULTIPLE PIN CANDIDATE LOCATIONS BENCHMARKS

Circuit	# Net	Size (μm^2)	[10]				Ours			
			Rout. (%)	Overlay Length	CPU (s)	#C	Rout. (%)	Overlay Length	CPU (s)	#C
Test6	1500	6.8 x 6.8	90.13	2300	738	0	96.5	193	0.7	0
Test7	2700	9.6 x 9.6	93.25	4097	2919	0	97.6	245	2.7	0
Test8	5500	16.0 x 16.0	93.07	7521	19019	0	97.8	339	3.6	0
Test9	12000	24.0 x 24.0	NA	NA	$>$ 100000	NA	98.1	745	5.3	0
Test10	28000	36.0 x 36.0	NA	NA	$>$ 100000	NA	98.4	1289	50.8	0
Comp.			0.950	19.940	2520		1.000	1.000	1.000	

threshold $f_{\text{threshold}}$ (lines 12–14). After all nets are routed, the color flipping would be performed on the full layout to further perform overlay minimization (line 16). The time complexity analysis of the overlay-aware detailed routing algorithm is as follows: if there are m grids in the routing map M, and n nets in the netlist N, the time to perform overlay-aware A^*-search in line 4 is $O(m \log m)$. Updating the overlay constraint graph and checking the overlay odd cycle in lines 5 and 6 take $O(n)$ time. Cut conflict check in line 6 is actually performed during the backtrace step of overlay-aware A^*-search. Pseudo-coloring in line 11 takes $O(1)$ time and, by Theorem 4, color-flipping in line 13 takes $O(n)$ time. Therefore, inside the for loop, lines 4–13 take $O(n + m \log m)$ time, and the for loop of lines 2–15 would be performed by $O(n)$ times. These take a total of $O(n(n + m \log m))$ time. Moreover, since $n = O(m)$, the total running time of the proposed overlay-aware detailed routing algorithm is $O(nm \log m)$.

IV. 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 [10], [11], and [16]. Because the binary codes of [10] and [16] are currently unavailable, we implemented [10] and [16] for comparative studies. The work [10] adopts multiple pin candidate locations, where the source pin and the target pin of every two-pin net have multiple candidate locations, while the works [11], [16] do not allow multiple pin candidate locations. As a result, we had two sets of benchmarks: 1) benchmarks in which the source and target pins of every two-pin net have multiple candidate locations and 2) benchmarks in which the locations of the source and target pins of each two-pin net are fixed. Every benchmark has three routing layers. Our algorithm can apply to both sets of benchmarks. The user-defined parameters in (5) are set as follows: $\alpha = \beta = 1$, $\gamma = 1.5$, and the flipping threshold $f_{\text{threshold}} = 10$. The benchmarks are scaled down to 10 nm node, where the design rules of SADP using the cut process are set as follows: $w_{\text{line}} = w_{\text{spacer}} = w_{\text{cut}} = w_{\text{core}} = 20$ nm, and $d_{\text{cut}} = d_{\text{core}} = 30$ nm.

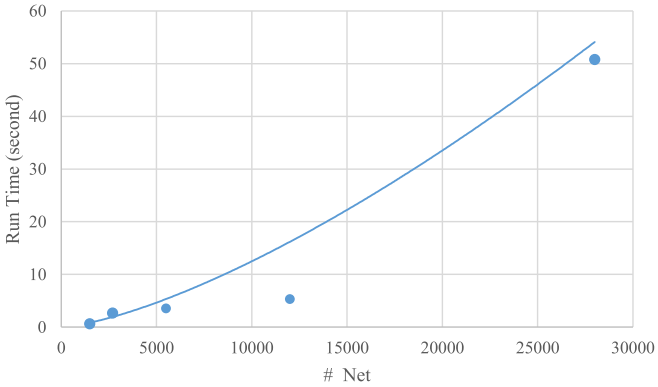

Fig. 20. Running time for the overlay-aware detailed routing.

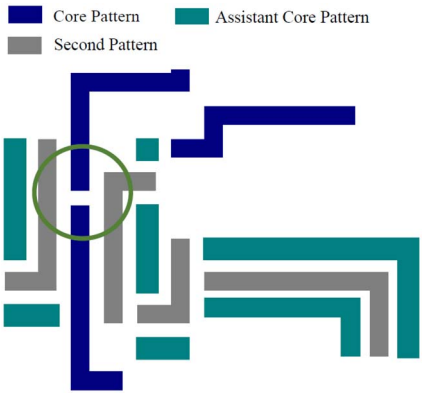

Fig. 21. Partial routing result of the proposed routing algorithm. The odd cycle is decomposable by applying the merge and cut technique.

The experimental results are listed in Tables III and IV, where "*R*out." 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 cut conflict for the cut

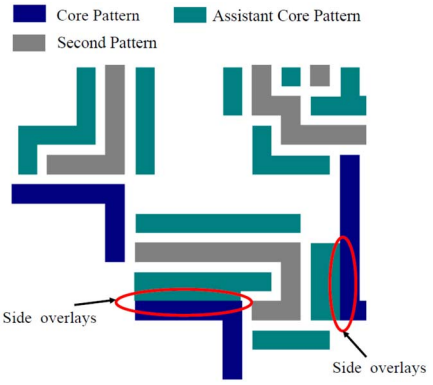

Fig. 22. Partial routing result of [16]. The merging of the core patterns and the assistant core patterns induces severe side overlays.

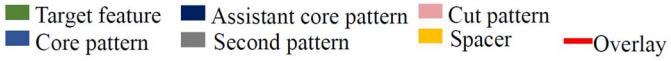

Fig. 23. Colors that represent different patterns.

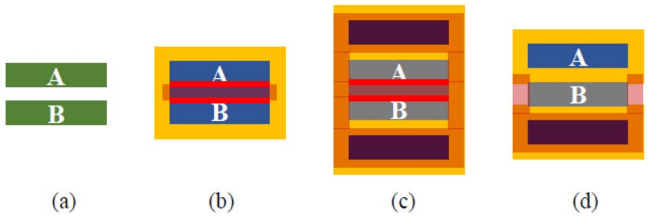

Fig. 24. (a) Type 1-a potential overlay scenario. Color assignments (b) and (c) CC and SS induce side overlays longer than w_{line}, which are strictly forbidden and (d) CS and SC do not induce side overlay.

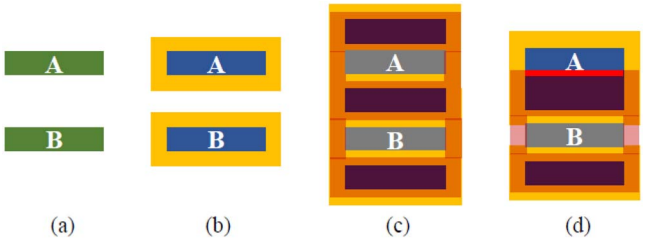

Fig. 25. (a) Type 1-b potential overlay scenario. Color assignments (b) and (c) CC and SS do not induce side overlay and (d) CS and SC induce side overlays longer than w_{line}, which are strictly forbidden.

process, and the trim conflicts for the trim process, which are induced by parallel line ends [2], [10].

We first conducted an experiment on the first set of benchmarks. Compared with [11] and [16], our algorithm efficiently reduces the total length of side overlays by more than 90%, with zero cut conflicts (implying decomposable layouts). Compared with [10] on the second set of benchmarks, our algorithm reduces the total length of side overlays by more than 90% with a 2520× speedup and 5% higher routability. In Fig. 20, the running time of our algorithm is plotted as a function of the number of nets. Based on the least-squares analysis, the empirical time complexity of our algorithm is around $n^{1.42}$ to the number of nets n.

Compared with the three state-of-the-art studies, the experimental results show that our algorithm can achieve the best

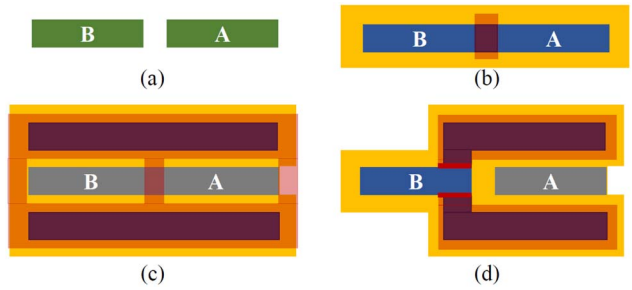

Fig. 26. (a) Type 2-a potential overlay scenario. Color assignments (b) and (c) CC and SS do not induce side overlay and (d) CS and SC induce side overlays. Further, they may also induce the cut pattern conflicts.

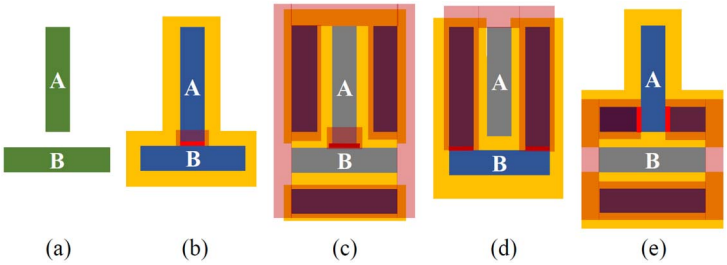

Fig. 27. (a) Type 2-b potential overlay scenario. Color assignments (b) and (c) CC and SS induce one unit of side overlays and (d) and (e) CS and SC induce two units of side overlays. Further, CS may also induce the cut conflicts.

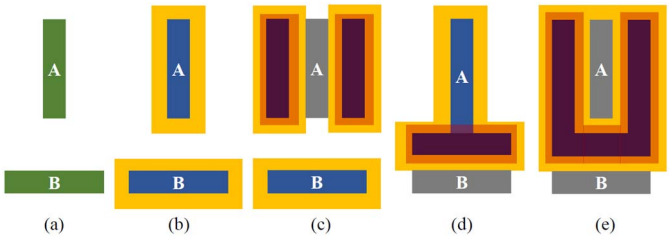

Fig. 28. (a) Type 2-c potential overlay scenario. (b)–(e) Type 2-c potential overlay scenario does not induce side overlay regardless of the color assignments.

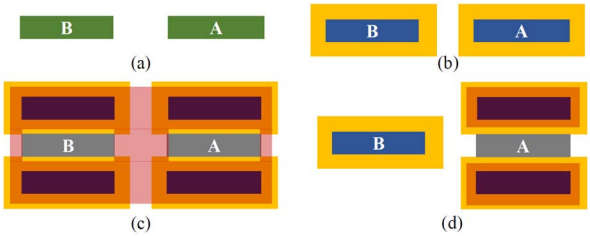

Fig. 29. (a) Type 2-d potential overlay scenario. (b)–(d) Type 2-d potential overlay scenario does not induce side overlay regardless of the color assignments.

quality and efficiency, with zero cut conflicts, smallest overlay length, and highest routability. A partial routing result of our algorithm is shown in Fig. 21, where the odd cycle is decomposed. The side overlays are no longer than one unit. A partial routing result of [16] is shown in Fig. 22, where the merging of the core patterns and the assistant core patterns induces severe side overlays.

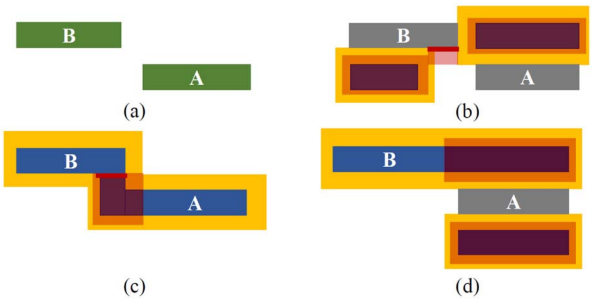

Fig. 30. (a) Type 3-a potential overlay scenario. (b) Color assignment SS induces side overlays. The assistant core pattern for B is shorter than B, or the distance between the two assistant core patterns is smaller than d_{core}. (c) Color assignment CC induces side overlays. (d) Color assignments CS and SC do not induce side overlay.

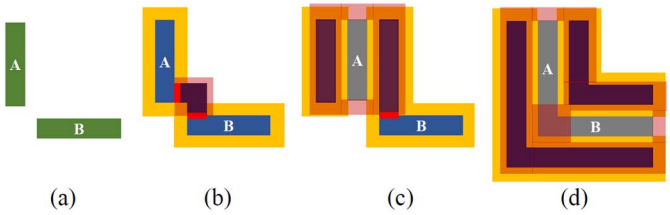

Fig. 31. (a) Type 3-b potential overlay scenario. Color assignment (b) CC induces side overlays, if the right side of A and top side of B are merged, (c) SC induces side overlays, and (d) SS does not induce side overlay.

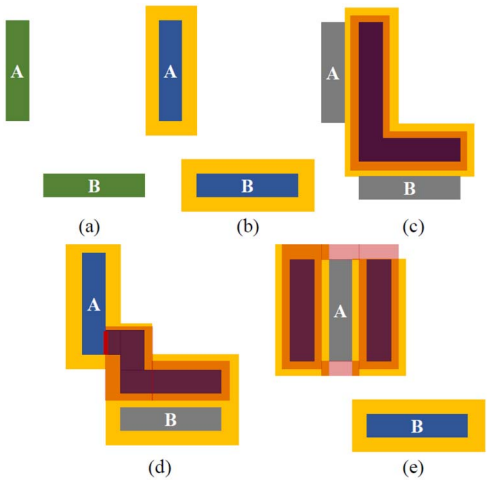

Fig. 32. (a) Type 3-c potential overlay scenario. (b)–(e) Color assignments CC, SS, and SC do not induce side overlay. (d) Color assignment CS induces side overlays.

V. Conclusion

We have presented the first work that handles all overlay scenarios and applies the cut technique for decomposing odd cycles during 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 conflicts and the best quality and efficiency among all published works.

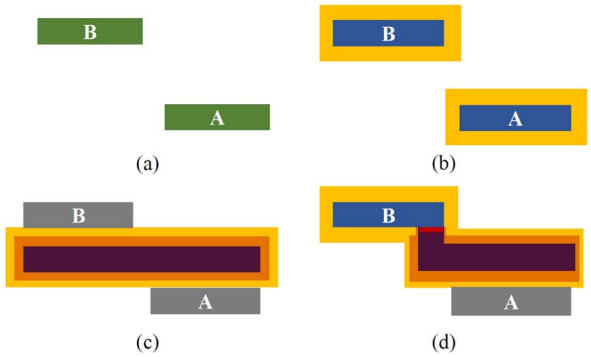

Fig. 33. (a) Type 3-d potential overlay scenario. Color assignments (b) and (c) CC and SS do not induce side overlay and (d) CS and SC induce side overlays.

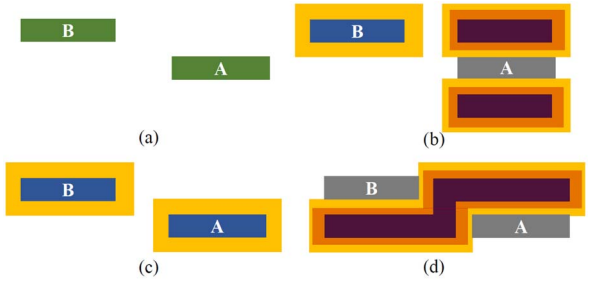

Fig. 34. (a) Type 3-e potential overlay scenario. (b)–(d) Type 2-e potential overlay scenario does not induce side overlay regardless of the color assignments.

Appendix
Color Assignments for the Identified Potential Overlay Scenarios

In the Appendix, we enumerate all the color assignments for every potential overlay scenario, as shown in Figs. 24–34. By the enumeration, we identify the color assignments that induce minimum side overlays for every potential overlay scenario. The identified color assignments constitute the color rules, which gives our router valuable information during the coloring stage. Note that some assistant core patterns that do not merge with any pattern are not shown in the figures. The symbols used in Figs. 24–34 are listed in Fig. 23.

References

[1] I.-J. Liu, S.-Y. Fang, and Y.-W. Chang, "Overlay-aware detailed routing for self-aligned double patterning lithography using the cut process," in *Proc. ACM/IEEE Design Autom. Conf.*, San Francisco, CA, USA, 2014, pp. 1–6.

[2] G. Luk-Pat *et al.*, "Design compliance for spacer is dielectric (SID) patterning," in *Proc. SPIE*, vol. 8326. San Diego, CA, USA, 2012, Art. ID 83260D.

[3] Y. Ma *et al.*, "Self-aligned double patterning (SADP) compliant design flow," in *Proc. SPIE*, vol. 8327. San Diego, CA, USA, 2012, Art. ID 832706.

[4] L. Liebmann, "The escalating design impact of resolution-challenged lithography," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design*, San Jose, CA, USA, 2013.

[5] Y. Ban, K. Lucas, and D. Pan, "Flexible 2D layout decomposition framework for spacer-type double pattering lithography," in *Proc. ACM/IEEE Design Autom. Conf.*, New York, NY, USA, 2011, pp. 789–794.

[6] Y.-W. Chang, S.-Y. Fang, and Y.-S. Tai, "Layout decomposition for spacer-is-metal (SIM) self-aligned double patterning," in *Proc. IEEE/ACM Asia South Pac. Design Autom. Conf.*, Chiba, Japan, 2015, pp. 671–676.

[7] 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," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design*, San Jose, CA, USA, 2013, pp. 32–39.
[8] 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 Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 32, no. 8, pp. 1228–1239, Aug. 2013.
[9] H. Zhang, Y. Du, M. D. F. Wong, and R. Topaloglu, "Self-aligned double patterning decomposition for overlay minimization and hot spot detection," in *Proc. ACM/IEEE Design Autom. Conf.*, New York, NY, USA, 2011, pp. 71–76.
[10] Y. Du *et al.*, "Spacer-is-dielectric-compliant detailed routing for self-aligned double patterning lithography," in *Proc. ACM/IEEE Design Autom. Conf.*, Austin, TX, USA, 2013, pp. 1–6.
[11] J.-R. Gao and D. Z. Pan, "Flexible self-aligned double patterning aware detailed routing with prescribed layout planning," in *Proc. ACM Int. Symp. Phys. Design*, Santa Rosa, CA, USA, 2012, pp. 25–32.
[12] Y. Ma *et al.*, "Double patterning compliant logic design," in *Proc. SPIE*, vol. 7949. 2011, Art. ID 79490D.
[13] S.-Y. Fang, "Cut mask optimization with wire planning in self-aligned multiple patterning full-chip routing," in *Proc. IEEE/ACM Asia South Pac. Design Autom. Conf.*, Chiba, Japan, 2015, pp. 396–401.
[14] C. Kodama *et al.*, "Self-aligned double and quadruple patterning-aware grid routing with hotspots control," in *Proc. IEEE/ACM Asia South Pac. Design Autom. Conf.*, Yokohama, Japan, 2013, pp. 267–272.
[15] M. Mirsaeedi, J. A. Torres, and M. Anis, "Self-aligned double-patterning (SADP) friendly detailed routing," in *Proc. SPIE*, vol. 7974. 2011, Art. ID 79740O.
[16] F. Nakajima *et al.*, "Detailed routing with advanced flexibility and in compliance with self-aligned double patterning constraints," in *Proc. SPIE*, vol. 8684. 2013, Art. ID 86840A.
[17] W.-K. Mak, Y. Ding, and C. Chu, "Throughput optimization for SADP and e-beam based manufacturing of 1D layout," in *Proc. ACM/IEEE Design Autom. Conf.*, San Francisco, CA, USA, 2010, pp. 1–6.
[18] Y.-H. Lin and Y.-L. Li, "Double patterning lithography aware gridless detailed routing with innovative conflict graph," in *Proc. ACM/IEEE Design Autom. Conf.*, Anaheim, CA, USA, 2010, pp. 398–403.

Iou-Jen Liu received the B.S. degree in electrical engineering from National Taiwan University (NTU), Taipei, Taiwan, in 2012, and the M.S. degree from the Graduate Institute of Electronics Engineering, NTU, in 2014. He is currently pursuing the Ph.D. degree with the Department of Electrical and Computer Engineering, University of Illinois at Urbana–Champaign, Urbana, IL, USA.

His current research interests include physical design and design for manufacturability.

Shao-Yun Fang (S'11–M'13) received the B.S. degree in electrical engineering from National Taiwan University (NTU), Taipei, Taiwan, in 2008, and the Ph.D. degree from the Graduate Institute of Electronics Engineering, NTU, in 2013.

She is currently an Assistant Professor with the Department of Electrical Engineering, National Taiwan University of Science and Technology, Taipei. Her current research interests include physical design and design for manufacturability for integrated circuits.

Dr. Fang was a recipient of the First Place of the 2012 ACM/SIGDA Student Research Competition (Graduate Student Category), the Silver Award of the 2012 TSMC Outstanding Student Research Award (Category I: Circuit Design Technologies), the Best Paper Award from the 2010 Very Large-Scale Integration/Design CAD Symposium (EDA Category), and two best paper nominations from the 2012 and 2013 International Symposium on Physical Design.

Yao-Wen Chang (S'94–A'96–M'96–SM'12–F'13) received the B.S. degree from National Taiwan University (NTU), Taipei, Taiwan, in 1988, and the M.S. and Ph.D. degrees from the University of Texas at Austin, Austin, TX, USA, in 1993 and 1996, respectively, all in computer science.

He is currently the Associate Dean of the College of Electrical Engineering and Computer Science, and a Distinguished Professor with the Department of Electrical Engineering, NTU. He has served as the Independent Board Director of Genesys Logic, Taipei, and a Technical Consultant of Faraday, Hsinchu, Taiwan, MediaTek, Hsinchu, and RealTek, Hsinchu. His current research interests include physical design and manufacturability for integrated circuits. He has co-edited a textbook on electronic design automation and co-authored a book on routing and over 240 ACM/IEEE conference and journal papers in the above areas.

Dr. Chang was a recipient of four awards at the 50th DAC in 2013 for the First Most Papers (34 DAC papers) in the Fifth Decade, the Most Prolific Author in a Single Year (seven papers in 2012 and 2013 each), the DAC Prolific Author Award (40 Club), the Top-5 Longest Publication Streaks (recent 15 years), the First Place of five recent contests, such as the 2013 Placement Contest at ICCAD, the 2012 DAC Placement Contest, the 2012 ISPD Discrete Gate Sizing Contest, the 2011 PATMOS Timing Analysis Contest, and the 2009 ISPD Clock Tree Synthesis Contest, the DAC/ISPD/ICCAD contests on placement, global routing, clock network synthesis, discrete gate sizing, process defect detection, and mask optimization (15 times), six Best Paper Awards and 23 best paper award nominations from DAC (five times), ICCAD (four times), and ISPD (six times), the Distinguished Research Award (Highest Honor) from the Ministry of Science and Technology of Taiwan (three times), the IBM Faculty Award (three times), the CIEE Distinguished EE Professorship, the MXIC Young Chair Professorship, and the Distinguished (highest honor)/Excellent Teaching Award from NTU (eight times). He has served on the Editorial Board of the IEEE TRANSACTIONS ON COMPUTER-AIDED DESIGN OF INTEGRATED CIRCUITS AND SYSTEMS and the IEEE DESIGN AND TEST OF COMPUTERS. He has served as the General Chair of ICCAD and ISPD, the Program Chair of ASP-DAC, FPT, ICCAD, and ISPD, and the Steering Committee Chair of ISPD. He is currently the IEEE CEDA Vice President of Technical Activities. He has served on the Technical Program Committee of all major EDA conferences. He has also served as the Chair of the EDA Consortium of the Ministry of Education, Taiwan.