On Refining Standard Cell Placement for Self-aligned Double Patterning*

Ye-Hong Chen, Sheng-He Wang and Ting-Chi Wang
Department of Computer Science, National Tsing Hua University, Hsinchu City 30013, Taiwan

***Abstract*—In this paper, we study the problem of refining a standard cell placement for self-aligned double patterning (SADP), which asks to simultaneously refine a detailed placement and find a valid SADP layout decomposition such that both overlay violation and wirelength are as small as possible. We first present an algorithm that adopts the technique of white space insertion for an SADP-aware single-row cell placement problem. Based on the single-row algorithm, we then describe an approach to the addressed placement refinement problem. Finally, we report encouraging experimental results to support the efficacy of our approach.**

I. Introduction

Double patterning with 193nm immersion lithography has been adopted to mass production in 20 nm technology node and beyond [1]. Double patterning lithography (DPL) can be classified into two types: litho-etch-litho-etch (LELE) [2, 3, 4] and self-aligned double patterning (SADP) [5, 6, 7]. In general, SADP has better overlay controllability than LELE [5, 6, 7, 8].

Spacer-Is-Dielectric (SID) with the cut process is one widely adopted SADP. Fig. 1 shows an example where a layout is decomposed into two masks. The first one is *core mask* which defines core patterns. Spacers are deposited around core patterns and viewed as protection. The second mask is *cut mask* which defines cut patterns. The regions not covered by spacers or cut patterns form the final patterns. In this example, to produce the three target patterns A, B, and C in Fig. 1(a), pattern A is chosen as a core pattern, while patterns B and C are defined by the spacers and cut patterns as shown in Fig. 1(b). It can be seen that each of pattern B and pattern C has a side without the protection of spacers. Overlay errors may happen and each of such situations is referred to as an *overlay violation* [9]. Note that overlay effect on pattern ends (i.e., pattern tips) is negligible [5, 9].

The cut process is also flexible for two-dimensional layouts in such a way that merge-and-cut techniques [9] could be applied to break odd cycles of conflicts. As illustrated in Fig. 2, there is an odd cycle among three patterns in Fig. 2(a), where the double-arrow sign between two patterns indicates a conflict and means that the distance between the two patterns is less than the same-mask minimum spacing. Suppose the distance between patterns A and B is not smaller than the minimum width of a cut pattern. As shown in Fig. 2(b), we can add a padding pattern to merge patterns A and B into a core pattern, and then use a cut pattern to remove the padding part and get the final patterns.

Although SADP has better overlay tolerance than LELE, overlay violations may still occur on pattern sides without the protection of spacers. Especially in two-dimensional layouts, much more complex layout patterns and design rules could make spacers harder to protect all pattern sides, and then induce lots of overlay violations. Therefore, it is important to minimize total overlay violation for SADP layout decomposition.

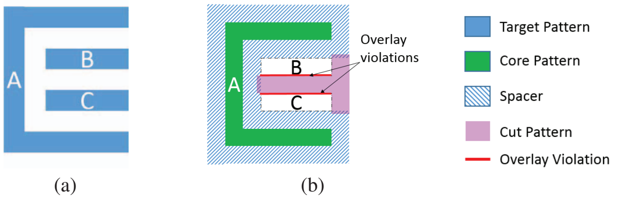


Fig. 1: Example of SID-type SADP with cut process. (a) Target layout. (b) Layout decomposition with overlay violations.

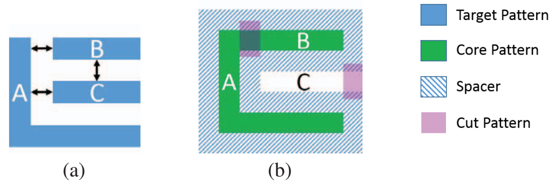


Fig. 2: Illustration of merge-and-cut. (a) Target layout. (b) Layout decomposition with merge-and-cut.

In the past several years, research efforts in SADP have been made along several directions. Design compliance was discussed in [5, 10] to ease the difficulty of layout decomposition. In the placement stage, [11] proposed a greedy algorithm to legalize conflicting standard cells for layout decomposability, but it did not adopt any merge-and-cut technique to help resolve conflicts. SADP-aware detailed routing algorithms were proposed in [8, 12, 13, 14]. The works in [9, 15, 16, 17] proposed effective layout decomposition solutions.

Existing works in physical design for SADP focus more on layout decomposition and routing, while very few attempts are on placement. Because modifying SADP unfriendly patterns in a final layout requires high efforts, how to generate an SADP friendly layout, especially in an early physical design stage such as placement, is important. Though SADP layout decomposition is NP-hard in general, [9] pointed out that it can be solved in polynomial time for the M1 layer in a cell-based row-structure layout. It is observed that patterns on the M1 layer are most hard-to-decompose for standard cells, and hence in this paper we adopt the SID-type SADP with the cut process and study how to simultaneously refine a standard cell placement and generate a valid SADP layout decomposition for the M1 layer. We first present an algorithm that adopts the technique of white space insertion for an SADP-aware single-row cell placement problem. Based on the single-row algorithm, we then develop an approach for the placement refinement problem. Experimental results are reported to support the efficacy of our approach.

The rest of this paper is organized as follows. The problem formulation is given in Section II. The details of our single-row cell placement algorithm and placement refinement approach are respectively described in Sections III and IV. Section V reports our experimental results, and the conclusion is drawn in Section VI.

*This work was supported in part by the Ministry of Science and Technology of Taiwan under Grant No. 105-2221-E-007-119-MY2.

978-3-9815370-8-6/17/$31.00 ©2017 IEEE

II. Problem Formulation

We assume that each standard cell is always SADP-decomposable; that is, there exists at least one SADP layout decomposition for each cell. Let s denote the same-mask minimum distance between two core (or cut) patterns, d the minimum width of a core (or cut) pattern, and w the width of the spacer. In practice, we may set $w = d$, $d < s$, and $s < d + 2w$ [9]. The SADP-aware placement refinement problem is formally defined as follows.

Problem 1 (SADP-Aware Placement Refinement). *Given a set of mask rules (i.e., s, d, w) and a standard cell placement with m rows, the problems asks to find a legal placement by re-distributing white space between cells in each row and generate a valid SADP decomposition for the M1 layer. The objective is to minimize the total length of overlay violations and the total half-perimeter wirelength (HPWL) of nets.*

To maintain the original design quality (e.g., wirelength) of the given standard cell placement as much as possible, our problem does not consider changing the cell order in each row, but allows to adjust white space inserted between cells (i.e., allows cell shifting) for a better decomposition quality. The total length of overlay violations is computed by summing up the corresponding length of each overlay violation. For example, in Fig. 1(b), there are two overlay violations respectively on the bottom side of pattern B and on the top side of pattern C, and hence the total length of overlay violations is the sum of the lengths of red segments on the two sides.

III. Algorithm for SADP-aware Single-row Placement

In this section, we formulate an SADP-aware single-row placement problem and present an algorithm for it.

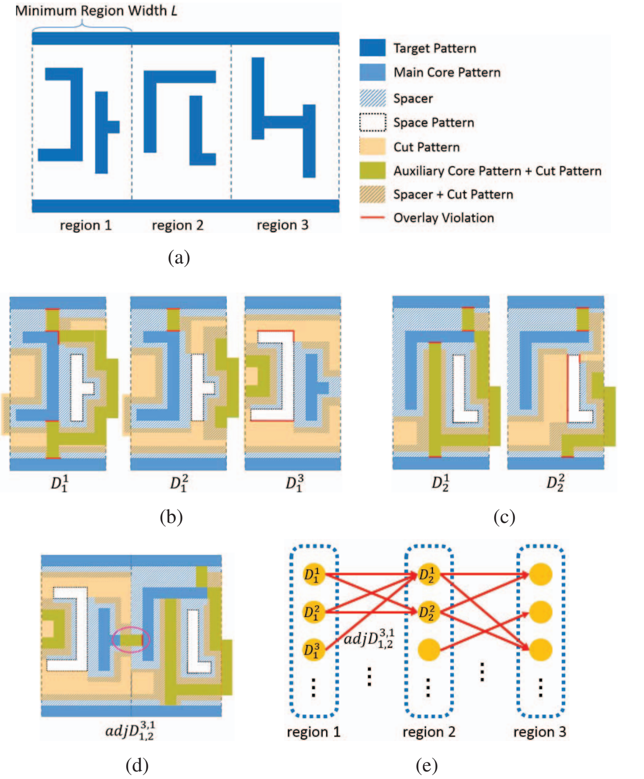
Problem 2 (SADP-Aware Single-Row Placement). *Given a set of mask rules, a set of standard cells to be placed on a fixed-width row, and the cell order from left to right, this problem asks to find a legal placement for the row as well as a valid SADP layout decomposition for the M1 layer such that the given cell order is preserved and the total length of overlay violations is as small as possible.*

Since our focus is on considering SADP decomposition during placement, it is natural to collect all valid SADP decompositions of each standard cell first. These valid SADP decompositions are generated by an existing algorithm introduced in [9]. Then we apply the same SADP decomposition algorithm to find all valid decompositions around the boundaries of any two cells. Finally, we find a legal placement as well as a valid decomposition from the decomposition information collected previously. The details of our SADP-aware single-row placement algorithm are described in the rest of this section.

A. Construction of Cell Solution Graph

The layout decomposition algorithm proposed in [9] divides *a cell row* into *regions*, and ensure that if each region has the minimum region width L or is wider, the decompositions of non-adjacent regions would not affect each other. Furthermore for a region, the decompositions around its left-side boundary would not affect the decompositions around its right-side boundary. Note that L has to be not smaller than $3s + 2d + 2w$.

To apply the algorithm of [9] to generate all valid SADP decompositions for a standard cell, we first divide the cell into regions such that all regions but the last one have their widths equal to the minimum region width L while the width of the last region is equal to or larger than L. If the width of a cell is smaller than L, we will extend the right boundary of the cell to a minimum extent such that the new cell width is larger than or equal to L. The regions of each cell are numbered by 1, 2, ..., from left to right, as illustrated in Fig. 3(a). Then the decomposition algorithm proposed in [9] is applied to find all valid decompositions for each region. Let D_i^j denote the jth valid decomposition of region i. Fig. 3(b) shows three valid decompositions of region 1 (D_1^1, D_1^2 and D_1^3), and Fig. 3(c) shows two valid decompositions of region 2 (D_2^1 and D_2^2). Note that we only show some valid decompositions in this example.



(a)

(b)

(c)

(d)

(e)

Fig. 3: Construction of cell solution graph. (a) A cell is divided into three regions. (b) Three decompositions of region 1. (c) Two decompositions of region 2. (d) An adjacent region decomposition between regions 1 and 2. (e) Cell solution graph, where the length of each overlay violation stored on a vertex or an edge is not shown.

There are two ways to generate a target pattern in the decomposition process. The first way is to assign a target pattern as a *core pattern* such that the target pattern is generated directly through the core pattern on the core mask. The second way is to assign the target pattern as a *space pattern*, which places an *auxiliary core pattern* surrounding the target pattern so that the generated spacer from the auxiliary core pattern can define the target pattern. Note that since auxiliary core patterns are not target patterns, they should be removed later on by cut patterns. Due to the different usage, a core pattern introduced in the first way is called *main core pattern*. A *core/space assignment* of a region corresponds to assigning each target pattern in the region as a core pattern or space pattern, but no both. If there are u target patterns in a region, the total amount of core/space assignments for the region is 2^u. Each core/space assignment may induce 0, 1, or more valid decompositions. For example, the two decompositions shown in Fig. 3(c), D_2^1 and D_2^2, have the same core/space assignment for the two target patterns such that the left one is assigned as core pattern and the right one as space pattern, but obviously their decompositions are different.

To store all valid decompositions of a cell, a *cell solution graph* is built such that each decomposition of a region and the corresponding total length of overlay violations are stored into a vertex in the graph (see Fig. 3(e)). Each column of vertices in the graph represents all valid decompositions of the corresponding region. If the amount of vertices in a column is too large, we only keep the first r decompositions with smaller total length of overlay violations, where r is a user-defined parameter and set to 10 in our experiments.

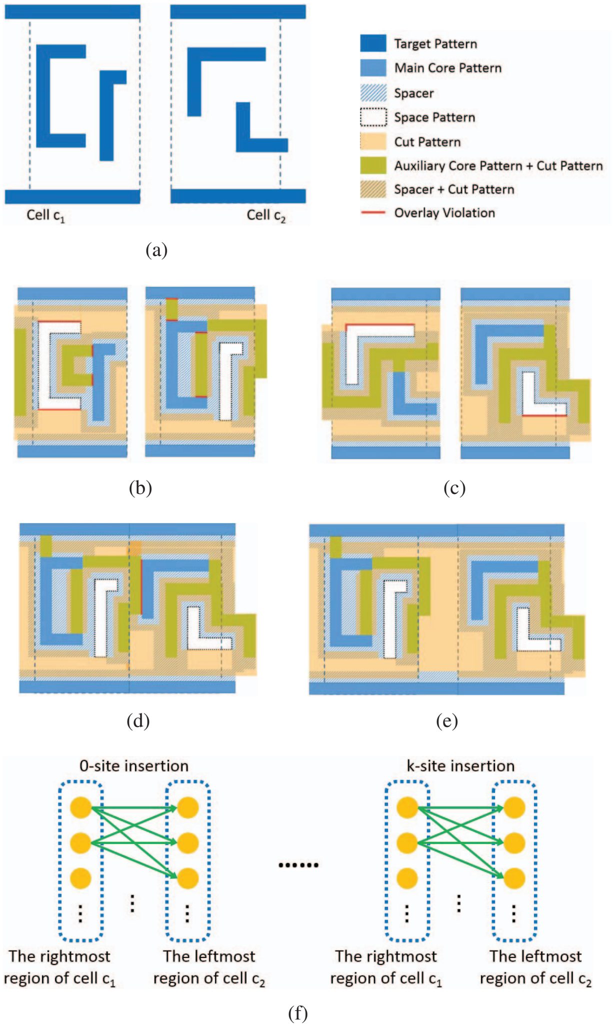
A directed edge connecting two vertices in adjacent columns is added to the cell solution graph if the two corresponding region decompositions are *compatible*. Two decompositions from adjacent regions are said to be compatible if they can be combined to form a valid decomposition according to their region order. The compatibility checking is done by a method similar to the one for finding a decomposition of a region [9], and may add/remove auxiliary core patterns to/from the union of the two corresponding region decompositions. The corresponding cut patterns are also modified accordingly. The added or removed patterns and the total length of new overlay violations (if there is any) are stored in each edge of the cell solution graph. Let $adjD_{i,i+1}^{j,j'}$ denote the region boundary decomposition from the jth decomposition of region i and the j'th decomposition of region $i+1$. Fig. 3(d) shows a region boundary decomposition, $adjD_{1,2}^{3,1}$, where the two region decompositions are from the third one of region 1 (D_1^3) and the first one of region 2 (D_2^1). It can be seen that there is a new auxiliary core pattern and a corresponding cut pattern added into the region boundary, and a new overlay violation occurs.

B. *Construction of Cell Boundary Solution Graph*

We then consider how to generate all valid decompositions around the boundary between two cells. We only need to focus on *cell boundary regions* in the decomposition process. The cell boundary regions are the rightmost region of the left cell and the leftmost region of the right cell. Let k denote the smallest integer such that two cells would never affect each other in decomposition after inserting $k+1$ sites of white space between them. Therefore there are k+1 scenarios for inserting a different amount of white space between the two cells. For each different amount of white space inserted between two cells, a *cell boundary solution graph* is built to store the decomposition information for the boundary between the two cells.

To build a cell boundary solution graph, the first step is to copy the last-column vertices (the first-column vertices, respectively) from the cell solution graph of the left cell (the right cell, respectively) into the cell boundary solution graph. Then a directed edge connecting two vertices in different columns is added to the cell boundary solution graph if the two corresponding region decompositions together with the inserted amount of white space can be combined to form a valid decomposition according to their cell order. This combining step is done in a way similar to the compatibility checking step of building an edge in a cell solution graph. The added/removed patterns and the total length of new overlay violations are stored in each edge as well.

Fig. 4 gives an example to illustrate how to build cell boundary solution graphs. Fig. 4(a) shows two cell boundary regions, i.e., the rightmost region of the left cell c_1 and the leftmost region of the right cell c_2. Fig. 4(b) shows two valid decompositions of the rightmost region of cell c_1, and Fig. 4(c) shows two valid decompositions of the leftmost region of cell c_2. All valid region decompositions are stored into vertices of each cell boundary solution graph, and there are $k+1$ cell boundary solution graphs due to the different amount of white space insertion as shown in Fig. 4(f). Fig. 4(d) and Fig. 4(e) show that cell boundary decompositions would be different if the amounts of inserted white space are different. When considering the combination of the second decomposition from the left cell and the second decomposition from the right cell, Fig. 4(d) is the decomposition result without inserting white space, and a new overlay violation (marked by the red segment) is induced. But in Fig. 4(e), the new overlay violation does not exist after inserting one site of white space.


Cell c_1
Cell c_2
Target Pattern
Main Core Pattern
Spacer
Space Pattern
Cut Pattern
Auxiliary Core Pattern + Cut Pattern
Spacer + Cut Pattern
Overlay Violation
0-site insertion
k-site insertion
The rightmost region of cell c_1
The leftmost region of cell c_2
The rightmost region of cell c_1
The leftmost region of cell c_2

(a)

(b)

(c)

(d)

(e)

(f)

Fig. 4: Construction of cell boundary solution graph. (a) Cell boundary regions of two cells c_1 and c_2. (b) Two decompositions of the rightmost region of cell c_1. (c) Two decompositions of the leftmost region of cell c_2. (d) Decomposition without inserting white space. (e) Decomposition with inserting 1 site of white space. (f) Cell boundary solution graphs for white space insertion from 0 to k sites.

C. *Single-Row Algorithm*

For the speed-up purpose, we off-line construct the cell solution graph for each cell in the cell library as well as the cell boundary solution graphs for each pair of cells. These graphs are stored in a look-up table.

We are now ready to describe our algorithm for the SADP-aware single-row placement problem. Suppose there are n cells to be placed on a row with the fixed cell order c_1, c_2, ..., c_n (from left to right) as shown in Fig. 5(a). Our algorithm first constructs a graph, called *row solution graph*. The cell solution graph of each cell c_i is copied into the row solution graph. We also take the total length of SADP overlay violations of each decomposition as the cost of the corresponding vertex or edge. For example, Fig. 5(b) shows the cell solution graphs of two cells c_1 and c_2, and they both are copied into the row solution graph as shown in Fig. 5(d). Next we merge the cell boundary solution graphs of each neighboring cell pair (c_i, c_{i+1}) into the row solution graph. There could be more than one cell boundary solution graph for two neighboring cells, and for these cell boundary solution graphs, edges connecting the same vertex pair should be preserved in the row solution graph unless they are *dominated edges*. An edge is dominated by another edge if it has the same or larger cost on the length of overlay violations and inserts more white space. For example in Fig. 5(c), the pairs of edge cost (i.e., length of overlay violations) and the amount of inserted white space (i.e., the value of q) for three edges connecting from vertex y to vertex z are respectively $(5, 0)$, $(2, 1)$ and $(4, 2)$. It is obvious that the edge with respect to $(4, 2)$ is dominated by the edge with respect to $(2, 1)$. Because we could preserve more than one edge to be merged into the row solution graph between the same vertex pair, to prevent multiple edges connecting the same vertex pair in our row solution graph, each non-dominated edge is replaced by a new vertex which also stores the corresponding information about the cost and white space. Then, we add two directed edges with zero cost to connect the new vertex and the two end vertices of the replaced edge. As illustrated in Fig. 5(c), there are two non-dominated edges connecting vertices y and z, and now in Fig. 5(d), they are replaced by two new vertices which store their respective information for cost and white space; there are also four edges added with zero cost to connect the new vertices from y and to z. After merging cell boundary solution graphs, we add a source vertex s and a target vertex t into the row solution graph. The source vertex s connects to all vertices at the first column of the first cell with zero cost edges, and the target vertex t connects from all vertices at the last column of the last cell with zero cost edges (see Fig. 5(d)).

Since the width of the given row is fixed, the total amount of white space that can be inserted between cells is also fixed. Therefore, we apply a resource-constrained least-cost path algorithm [18] on the row solution graph to find our solution. The algorithm takes the white space as the resource, and finds out a path from the source vertex s to the target vertex t such that without exceeding the white space limit, the total cost (i.e., the total length of overlay violations) of vertices and edges on the path is smallest. The resultant placement and SADP decomposition is derived from the path accordingly.

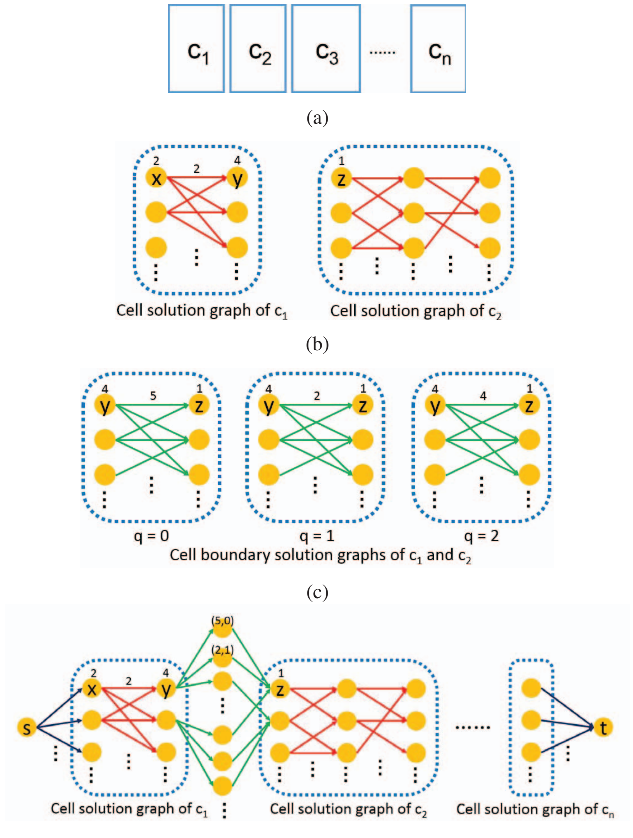
IV. Approach for SADP-Aware Placement Refinement

Our approach for the SADP-aware placement refinement problem consists of two stages. The first stage is to refine the given placement for minimizing the total length of overlay violations, while the second stage is to further improve the placement for wirelength minimization but without increasing the total length of overlay violations. The details of the two stages are described in the following two subsections.

A. Stage 1: Refinement for SADP Decomposition

Due to the mask rules, there is no interaction between the decompositions of patterns (except power/ground tracks) in adjacent rows. Therefore, the first stage of our approach refines the cell placement for each row independently by applying our single-row algorithm. Furthermore, each cell row is refined concurrently.

For each cell row, there are four possible core/space assignments for its power and ground tracks, denoted by core-core (cc), core-space (cs), space-core (sc) and space-space (ss). In order to generate consistent decompositions of power/ground tracks in adjacent cell rows, we further divide the row solution graph of each cell row into four sub-graphs such that each sub-graph corresponds to a different core/space assignment for the power and ground tracks in the row. As a result, there are four placement and SADP decomposition results produced by our single-row algorithm after respectively finding a resource-constrained least-cost path on each sub-graph.


c1
c2
c3
......
cn
2
2
4
x
y
1
z
Cell solution graph of c1
Cell solution graph of c2
4
5
1
y
z
4
2
1
y
z
4
4
1
y
z
q = 0
q = 1
q = 2
Cell boundary solution graphs of c1 and c2
(5,0)
(2,1)
s
t
......
Cell solution graph of c1
Cell solution graph of c2
Cell solution graph of cn

(a)

(b)

(c)

(d)

Fig. 5: Construction of row solution graph. (a) An ordered set of cells to be placed on a row. (b) Illustration of cell solution graphs of cells c_1 and c_2. (c) Three cell boundary solution graphs with different amounts of white space inserted. (d) The row solution graph.

Next we construct a whole-layout solution graph as shown in Fig. 6, where each row contains four vertices representing four different placement and SADP decomposition results. Each vertex is also associated with a cost denoting the total length of overlay violations of the corresponding decomposition. The label inside each vertex is the core/space assignment of the power/ground tracks. Then we add the directed edges with zero cost between vertices in adjacent rows. Because adjacent rows share the same power/ground track, the core/space assignments of two vertices connected by an edge must be consistent. Finally, we add two vertices, s and t, as the source and the target, and connect them to the graph with zero edge cost. By finding a least-cost path from s to t in the whole-layout solution graph, we get the whole-layout placement and SADP decomposition result with minimal overlay violation length.

B. Stage 2: Refinement for Wirelength Minimization

Since the first stage of our approach only aims at layout decomposition, the wirelength quality of the resultant placement may degrade. In addition, it is likely that for some cell

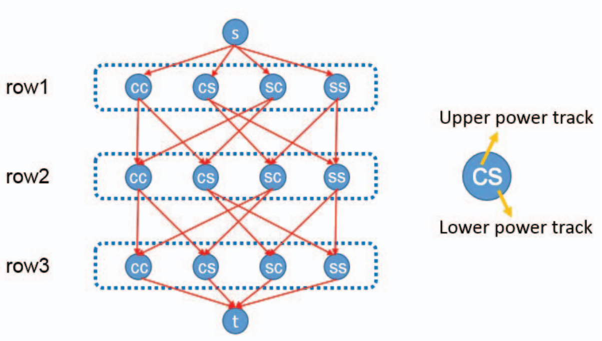
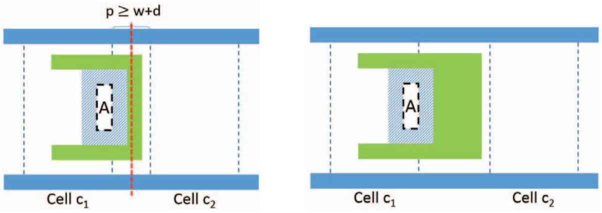


Fig. 6: Illustration of the whole-layout solution graph for a 3-row example.



(a) (b)

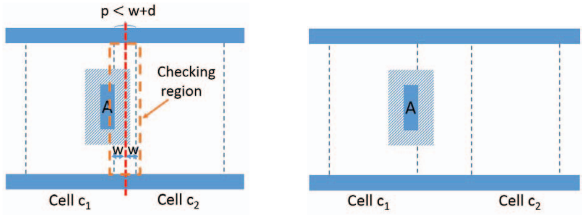
Fig. 7: Illustration of $p \geq w + d$ with the central line intersecting an auxiliary core pattern. (a) Before increasing white space. (b) After increasing white space.

rows, their amounts of inserted white space may not reach their white space limits yet. Therefore, the second stage of our approach keeps the decomposition of each cell intact and applies wirelength-driven linear placement [19] row by row to properly distribute the remaining white space for minimizing the HPWL. In order not to increase the total length of overlay violations, the amount of white space between each pair of adjacent cells will not be decreased in the second stage. On the other hand, whether the white space between two cells can be increased needs careful checking.

For each pair of adjacent cells, we draw the central line of the white space between them. Let p denote the distance between two adjacent cells; when p is 0, it means the two cells abut without any white space between them. We have two cases to consider: $p \geq w+d$ and $p < w+d$ (recall that d is the minimum width of a core or cut pattern, and w is the width of the spacer). To make the case analysis easier, we assume $d = w$.

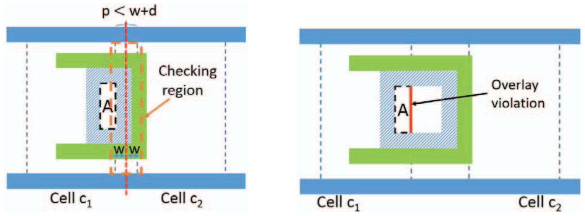
When $p \geq w + d$, the white space between the two cells can be freely increased by linear placement because it will not induce any new overlay violation. For example, Fig. 7(a) shows two cells whose distance p is equal or greater than $w+d$. There is a space pattern A aligned with the left cell's boundary and the central line intersects the auxiliary core pattern of A. Fig. 7(b) shows the new auxiliary core pattern after enlarging the white space. For this example, we also need to update the cut mask for removing the new auxiliary core pattern. It is clear that no new overlay violation induced in this example.

When $p < w + d$, if the central line does not intersect any spacer, we can freely insert additional white space, similar to the case $p \geq w + d$. Otherwise, we need to check the region formed by horizontally extending a distance w from the central line. If there is no space pattern intersecting this region, we can increase white space freely, as illustrated in Fig. 8 which shows a main core pattern but no space pattering intersecting the red dash-line region. If there is a space pattern intersecting this region, no additional white space should be inserted to prevent the generation of any new overlay violation. For example, Fig. 9(a) has a space pattern A intersecting the red dash-line region. After inserting additional white space, the auxiliary core pattern



(a) (b)

Fig. 8: Illustration of $p < w + d$ with a main core pattern intersecting the checking region. (a) Before increasing white space. (b) After increasing white space.



(a) (b)

Fig. 9: Illustration of $p < w + d$ with a space pattern intersecting the checking region. (a) Before increasing white space. (b) After increasing white space, which induces a new overlay violation.

and spacer need to be modified, which causes the right side of pattern A not to be protected by the spacer and induces a new overlay violation, as show in Fig. 9(b). Therefore, to avoid situations like Fig. 9(b), we merge the two cells together with the original white space between them into a new cell before linear placement. Each of the two cells is called *forbidden cell* and the new cell is called *super cell*. This cell merging operation could also merge more than two consecutive cells together with their associated white space into a super cell when the white space between any two adjacent cells is forbidden to increase.

After performing cell merging operations on each row, we get a new set of cells on each row. We then apply linear placement on each new cell row. During linear placement, we treat the original white space between two adjacent cells in a new cell row as a dummy cell whose width is set to the corresponding amount of white space. The cells and dummy cells naturally form a left-to-right order in each row. With dummy cells, the distance between two adjacent cells will be larger or equal to the original white space after linear placement. In addition, with super cells, the original white space between two forbidden cells are kept intact by linear placement. Therefore the linear placement result will not degrade the decomposition quality. Our linear placement method is based on the one proposed in [19] and is able to iterate over all cell rows until the HPWL improvement converges.

V. Experimental Results

The proposed SADP-aware placement refinement approach was implemented in C++ language. All the experiments were run on a Linux machine with 2.40 GHz Intel Xeon CPU and 96 GB memory. A set of cell types was selected from the 45 nm Nangate Open Cell Library [20] and used in the experiments. These cells were scaled down to fit a more advanced technology node, and the locations and shapes of patterns in each cell were also adjusted to ensure that at least one SADP layout decomposition exists. To test the performance and scalability of our approach, we first randomly generated 8 cell placement results with various amounts of rows, cells, and patterns as respectively shown in the "#Row", "#Cell", and "#Pattern" columns of Table I. We then applied our linear placement

TABLE I: Experimental results.

Test Case				Initial Placement			Refined Placement		
Name	#Row	#Cell	#Pattern	HPWL (x$10^6 um$)	TOL (x$10^4 um$)	T(s)	HPWL (x$10^6 um$)	TOL (x$10^4 um$)	T(s)
T1	200	51002	1530192	6.88	1.06	1.65	7.01	1.01	14.36
T2	500	177553	5326264	46.16	3.68	4.85	46.74	3.49	50.96
T3	1000	365006	10949848	155.6	7.57	9.83	156.85	7.18	103.81
T4	2000	1090098	32704076	872.97	22.63	32.91	878.52	21.45	337.54
T5	3000	795317	23859940	835.07	16.5	23.76	837.04	15.65	227.79
T6	3500	1312320	39367076	1632.16	27.22	36.31	1636.75	25.81	382.75
T7	4000	499985	14995412	668.76	10.37	15.05	669.35	9.84	132.18
T8	5000	1824964	54744660	3157.86	37.89	51.15	3164.08	35.91	536.92
Ratio	-	-	-	1	1	1	1.007	0.949	9.93

method to optimize the HPWL of each placement result. These HPWL-optimized placement results became the test cases for our approach to refine.

Since the first stage of our approach can be easily parallelized by assigning each cell row to a thread, we conducted a parallelism analysis by running our approach on each test case with different amounts of threads. Our experimental results reveal that our approach could respectively get 1.76X, 3.26X, and 6.07X average speedup when it was run with 2, 4, and 8 threads.

To see the impact of our approach on the reduction of overlay violations, we also implemented a layout decomposition algorithm based on [9] to produce the SADP layout decomposition result without changing any cell location for each test case. Table I shows the experimental results of each test case before and after running our approach. We report the wirelength results in the "HPWL" columns, the results of the total length of overlay violations in the "TOL" columns, and the runtime results measured in second in the "T(s)" columns. The runtime results of the layout decomposer and our refinement approach were all collected under the 8-thread setting. It can be seen from Table I that our approach on average helps reduce the total length of overlay violations by 5.1% with only 0.7% increase in the HPWL. Note that since the cells we used were not originally designed for SADP, the large amounts of overlay violations are expected. In addition, it is reasonable for our approach to run slower than the layout decomposer because our approach not just performs layout decomposition but also refines the placement. The runtime results of our approach are quite acceptable as it took less then 10 minutes for the largest test case which has more than 1.8 million cells.

VI. Conclusion

In this paper we present an approach to simultaneously refine a standard cell placement and find a valid SADP layout decomposition. The experimental results are also shown to support our approach.

References

[1] Y. Wei and R. L. Brainard, "Advanced processes for 193-nm immersion lithography," in *SPIE Press Book*, pp. 215–225, 2009.

[2] W. Arnold, M. Dusa, and J. Finders, "Manufacturing challenges in double patterning lithography," in *Proceedings of International Symposium on Semiconductor Manufacturing*, pp. 283–286, 2006.

[3] G. E. Bailey, A. Tritchkov, J.-W. Park, L. Hong, V. Wiaux, E. Hendrickx, S. Verhaegen, P. Xie, and J. Versluijs, "Double pattern eda solutions for 32nm hp and beyond," in *Proceedings of SPIE*, vol. 6521, pp. 65211K–1–65211K–12, 2007.

[4] A. B. Kahng, C.-H. Park, X. Xu, and H. Yao, "Layout decomposition for double patterning lithography," in *Proceedings of International Conference on Computer-Aided Design*, pp. 465–472, 2008.

[5] Y. Ma, J. Sweis, H. Yoshida, Y. Wang, J. Kye, and H. J. Levinson, "Self-aligned double patterning (sadp) compliant design flow," in *Proceedings of SPIE*, vol. 8327, pp. 832706–1–832706–13, 2012.

[6] K. Oyama, E. Nishimura, M. Kushibiki, K. Hasebe, S. Nakajima, H. Murakami, A. Hara, S. Yamauchi, S. Natori, K. Yabe, *et al.*, "The important challenge to extend spacer dp process towards 22nm and beyond," in *Proceedings of SPIE*, vol. 7639, pp. 763907–1–763907–6, 2010.

[7] H. Yaegashi, K. Oyama, K. Yabe, S. Yamauchi, A. Hara, and S. Natori, "Novel approaches to implement the self-aligned spacer double-patterning process toward 11-nm node and beyond," in *Proceedings of SPIE*, vol. 7972, pp. 79720B–1–79720B–7, 2011.

[8] 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 *Proceedings of Design Automation Conference*, pp. 1–6, 2014.

[9] Z. Xiao, Y. Du, H. Tian, and M. D. Wong, "Optimally minimizing overlay violation in self-aligned double patterning decomposition for row-based standard cell layout in polynomial time," in *Proceedings of International Conference on Computer-Aided Design*, pp. 32–39, 2013.

[10] G. Luk-Pat, A. Miloslavsky, B. Painter, L. Lin, P. De Bisschop, and K. Lucas, "Design compliance for spacer is dielectric (sid) patterning," in *Proceedings of SPIE*, vol. 8326, pp. 83260D–1–83260D–13, 2012.

[11] J.-R. Gao, B. Yu, R. Huang, and D. Z. Pan, "Self-aligned double patterning friendly configuration for standard cell library considering placement impact," in *Proceedings of SPIE*, vol. 8684, pp. 868406–1–868406–10, 2013.

[12] J.-R. Gao and D. Z. Pan, "Flexible self-aligned double patterning aware detailed routing with prescribed layout planning," in *Proceedings of International Symposium on Physical Design*, pp. 25–32, 2012.

[13] M. Mirsaeedi, J. A. Torres, and M. Anis, "Self-aligned double-patterning (sadp) friendly detailed routing," in *Proceedings of SPIE*, vol. 7974, pp. 79740O–1–79740O–9, 2011.

[14] C. Kodama, H. Ichikawa, K. Nakayama, T. Kotani, S. Nojima, S. Mimotogi, S. Miyamoto, and A. Takahashi, "Self-aligned double and quadruple patterning-aware grid routing with hotspots control," in *Proceedings of Asia and South Pacific Design Automation Conference*, pp. 267–272, 2013.

[15] Y. Ban, K. Lucas, and D. Pan, "Flexible 2d layout decomposition framework for spacer-type double pattering lithography," in *Proceedings of Design Automation Conference*, pp. 789–794, 2011.

[16] H. Zhang, Y. Du, M. D. Wong, and R. Topaloglu, "Self-aligned double patterning decomposition for overlay minimization and hot spot detection," in *Proceedings of Design Automation Conference*, pp. 71–76, 2011.

[17] Z. Xiao, Y. Du, H. Zhang, and M. D. Wong, "A polynomial time exact algorithm for self-aligned double patterning layout decomposition," in *Proceedings of International Symposium on Physical Design*, pp. 17–24, 2012.

[18] M. Ziegelmann, *Constrained shortest paths and related problems*. PhD thesis, Universitätsbibliothek, 2001.

[19] A. B. Kahng, P. Tucker, and A. Zelikovsky, "Optimization of linear placements for wirelength minimization with free sites," in *Proceedings of Asia and South Pacific Design Automation Conference*, pp. 241–244, 1999.

[20] "Nangate Open Cell Library." https://projects.si2.org/openeda.si2.org/projects/nangatelib.