Layout Decomposition for Double Patterning Lithography

Andrew B. Kahng†, Chul-Hong Park†, Xu Xu†, and Hailong Yao†
†CSE and ‡ECE Departments, UC San Diego, La Jolla, CA
§Blaze DFM, Inc., Sunnyvale, CA
abk@cs.ucsd.edu, chpark@vlsicad.ucsd.edu, xuxu@blaze-dfm.com, hailong@cs.ucsd.edu

ABSTRACT

In double patterning lithography (DPL) layout decomposition for 45nm and below process nodes, two features must be assigned opposite colors (corresponding to different exposures) if their spacing is less than the minimum coloring spacing [11, 9, 5]. However, there exist pattern configurations for which pattern features separated by less than the minimum color spacing cannot be assigned different colors. In such cases, DPL requires that a layout feature be split into two parts. We address this problem using a layout decomposition algorithm that includes graph construction, conflict cycle detection, and node splitting processes. We evaluate our technique on both real-world and artificially generated testcases in 45nm technology. Experimental results show that our proposed layout decomposition method effectively decomposes given layouts to satisfy the key goals of minimized line-ends and maximized overlap margin. There are no design rule violations in the final decomposed layout.

1. INTRODUCTION

As Moore’s law continues to drive performance and integration with smaller circuit features, lithography is pushed to new extremes. For 32nm node patterning, prospects for new lithography techniques such as extreme ultraviolet (EUV) and immersion ArF (IArF) are unclear. An EUV imaging system is composed of mirrors coated with multilayer structures designed to have high reflectivity at 13.5nm wavelength. There are significant technical hurdles to implementation of EUV lithography in terms of mask-blank fabrication, high output power source, resist material, etc. Challenges to production use of IArF include very high-refractive index fluids (to enable NA = 1.55 ~ 1.6), and accompanying advances in high-index resist and optical materials. Double Patterning Lithography (DPL) involves the partitioning of dense circuit patterns into two separate exposures, whereby decreased pattern density in each exposure improves resolution and depth of focus (DOF). DPL is likely to play an even more important role than previously anticipated, since the EUV adoption timeline has been delayed [6, 11]. However, DPL increases manufacturing cost in two fundamental ways: (1) complex process flows due to double exposure patterning, and (2) tight overlay control between the two patterning exposures.

There are distinct approaches to DPL, notably the LELE (litho-etch-litho-etch) and self-aligned approaches. In the LELE approach [11], the first etch step is necessary to transfer the pattern of the first resist layer into an underlying hardmask [12, 16] which is not removed during the second exposure. Photoresist is re-coated on the surface of the first layer, is exposed and then the flow finishes up with the hardmask and resist of second exposure. In self-aligned DPL [14, 17], the patterns for the first layer are transferred into the hardmask and then nitride spacers are formed on the sidewall of the patterns. A spacer is formed by deposition or reaction of the film on the pattern, followed by etching to remove all the film material but leave only the material on the sidewalls. Then, film materials between spacers produce the patterns for the second layer [15, 18]. The major concern of DPL is overlay control, which leads to requirements for more accurate overlay metrology, more representative sampling, reduction in model residuals, and improved overlay correction [10]. According to the ITRS [4], DPL requires overlay control of between 9nm and 6nm, which is a major hurdle for production deployment.

A key issue in DPL from the design point of view is the decomposition of the layout for multiple exposure steps [9]. This recalls strong Alt-PSM (Alternating-Phase Shift Mask) coloring issues and automatic phase conflict detection and resolution methods [7]. DPL layout decomposition must satisfy the following requirement: two features must be assigned opposite colors (corresponding to mask exposures) if their spacing is less than the minimum coloring spacing. However, there exist pattern configurations for which features within this minimum coloring spacing cannot all be assigned different colors [5, 19]. In such cases, at least one feature must be split into two or more parts. The pattern splitting increases manufacturing cost and complexity due to (1) generation of excessive line-ends, which causes yield loss due to overlay error in double-exposure, as well as line-end shortening under defocus; and (2) resulting requirements for tight overlay control, possibly beyond currently envisioned capabilities. Other risks include line edge (CD) errors due to overlay error, and interference mismatch between different masks. Therefore, a key optimization goal is to reduce the total cost of layout decomposition, considering the above-mentioned aspects.

Unlike previous works on Alt-PSM, which adopt graph bipartization algorithms, we formulate the optimization of DPL layout decomposition using integer linear programming (ILP). A pre-processing step fractures layout features into
small pieces according to vertex coordinates of neighboring features. From the fractured polygon pieces, we optimize polygon splitting with a process-aware cost function that avoids small jogging line-ends, maximizes overlap at dividing points of polygons, and preferentially makes splits at landing pads, junctions and long runs [9]. A layout partitioning heuristic helps achieve scalability for large layouts. We achieve an overall layout decomposition method for DPL which includes graph construction, conflict cycle detection, and node splitting processes. Our contributions are as follows.

- Our conflict cycle detection algorithm efficiently finds patterns with unresolved color assignment during the pattern splitting process. Because such patterns cannot be manufactured by DPL even with availability of layout decomposition and two mask exposures, this detection step enables fast feedback to the designer to modify the layout pattern.
- Our node splitting method determines mask features with maximum overlap length after decomposition, so that the pre-specified overlay margin can be observed.
- Our integer linear programming (ILP) based color assignment algorithm minimizes the number of line-ends and avoids design rule violations.
- Our layout partitioning technique improves scalability of the ILP-based coloring solution, whose runtime would otherwise increase unmanageably with layout size.

The remainder of this paper is organized as follows. Section II gives the overall flow of our layout decomposition system. Section III formally states the DPL color assignment problem and gives details of the node splitting and color assignment techniques. Section IV describes testcases, experimental setup and experimental results. Section V concludes with ongoing research directions.

2. DPL LAYOUT DECOMPOSITION FLOW

Figure 1: Overall DPL layout decomposition flow.

Figure 1 shows the overall flow for DPL layout decomposition. Given a layout, the polygonal layout features are first fractured into a set of non-overlapping rectangles using the minimum-silver fracturing algorithm of [13]. The minimum-silver fracturing minimizes the number of small rectangles and helps simplify downstream operations. Next, a conflict graph is constructed over the rectangular features according to the given minimum spacing constraint, $t$. Each node in the graph represents a rectangular feature; an edge exists between two nodes if the corresponding features are not touching each other, and the distance between the features is less than $t$.

We cast DPL layout decomposition as a problem of modifying the conflict graph by decomposing selected layout feature nodes (i.e., thus adding new nodes and inducing new edges) so that the graph can be properly 2-colored. To this end, the key is the removal of conflict cycles (CCs), which are the odd-length - and hence not 2-colorable - cycles in the conflict graph. We use a breadth first search (BFS) based conflict cycle detection algorithm to find the conflict cycles in the graph.\(^1\) When a conflict cycle is found, node splitting is applied to find the best layout feature to split, considering the maximization of overlap lengths. If the maximum possible overlap length is less than a given required overlap margin, then the layout has an unresolved conflict cycle (UCC) which must be flagged to the designer for layout modification. Otherwise, the layout feature will be split into smaller features to remove the conflict cycle. (Note that the node splitting process is required to remove a conflict cycle while generating line-ends having overlap length greater than the required overlap margin; there is no other option to remove a conflict cycle other than costly layout modification.) The graph is then updated, and the conflict cycle detection and node splitting processes are iterated until no conflict cycle remains in the graph.

After the iterative conflict cycle detection and node splitting process, we perform ILP-based coloring on the final conflict cycle-free graph to find an optimal coloring solution, considering minimization of the number of cuts (or line-ends) and design rule violations. Post-processing functions include analysis of minimum overlap lengths for all pairs of touching features (= adjacent split parts of an original layout feature, which have been assigned different mask colors), and design rule checking in the final mask solution.

Figure 2 illustrates DPL-based coloring according to our layout decomposition flow. Polygonal layout features in (a) are fractured into rectangles, over which the conflict graph is constructed and conflict cycles detected as shown in (b). To remove the conflict cycles in the conflict graph, the node splitting process is carried out along the dividing points denoted by the dashed lines in (b). After the node splitting process for conflict cycle removal, the conflict graph is updated with newly generated nodes and updated edges as in (c). Finally, ILP-based coloring is carried out to obtain the final coloring solution on the 2-colorable conflict graph (d).

3. DPL COLOR ASSIGNMENT PROBLEM

3.1 Problem Formulation

Fracturing and DPL Color Assignment Problem

Given: Layout $L$, and maximum distance between two features (i.e., polygons), $t$, at which the color assignment is constrained.

Find: A fracturing of $L$ and a color assignment of fractured features to minimize total cost.

Subject to: (i) Two non-touching fractured features corresponding to nodes $n_i$ and $n_j$ with $0 < d_{i,j} < t$ must be assigned different colors. (ii) Two touching features with $d_{i,j} = 0$, if assigned different colors, incur a cost $c_{i,j}$.

Figure 3 illustrates the color assignment problem. Feature $n_3$ (respectively, $n_4$) is assigned a different color from $n_1$.

\(^1\)Depth first search (DFS) based cycle detection may also be used. The BFS-based conflict cycle detection is more efficient for conflict cycles with fewer edges ($\leq 10$), whereas DFS-based detection is more efficient for conflict cycles with more edges ($> 10$). Since most conflict cycles in the layouts we have studied have fewer than 7 edges, we adopt BFS-based conflict cycle detection.
Figure 2: An example of the graph and layout coloring according to the DPL flow: (a) input layout, (b) fractured layout and conflict graph, (c) conflict cycle removal, and (d) ILP-based DPL coloring.

Figure 3: Example of color assignment problem: feature \( n_3 \) is assigned a different color from \( n_2 \) and \( n_1 \) because \( d_{2,4} < t \) and \( d_{3,5} < t \).

Figure 4: Two examples of dividing points. (a) Extended features at the dividing point cause coloring violation. (b) Extended features at the dividing point still satisfy the minimum coloring spacing rule.

3.2 Fracturing and Conflict Graph Construction

Given a layout \( L \), a rectangular layout \( L_R \) is obtained by fracturing layout polygons into rectangles. We fracture into rectangles \([13]\) so that distance computation and other feature operations (e.g., feature splitting) become easier. Our layout decomposition process begins with construction of a conflict graph based on the fractured layout. As illustrated in Figure 5, given a (post-fracturing) rectangular layout \( L_R \), the layout graph \( G = (V, E) \) is constructed by: (1) representing each feature (i.e., rectangle) by a node \( n \); (2) for any two non-touching features within distance \( t \), connecting the two corresponding nodes with an edge \( e \).

If two non-touching features are adjacent in the graph, either they belong to different original polygonal layout features, or there do not exist any other features between them (i.e., no features entirely block the two non-touching features). In Figure 5, we see edge set \( \{E_{1,3}, E_{3,5}, E_{5,6}\} \). There is no edge between \( n_2 \) and \( n_4 \) since node \( n_3 \) blocks these two nodes.

Figure 5: Example of conflict graph construction: every (rectangle) feature is represented by a node, and no feature entirely blocks two non-touching features that are adjacent in the graph.

3.3 Conflict Cycle Detection

To detect pattern configurations in which non-touching features cannot be assigned different colors, we find odd-length cycles in the conflict graph.

Definition 1: A conflict cycle is a cycle in the conflict graph which contains an odd number of edges.

Given a conflict graph as in Figure 2(b), we apply a breadth-first search (BFS) technique, given in Algorithm 1 and 2, to detect conflict cycles. Time complexity of the conflict cycle detection algorithm is \( O(V + E) \), where \( V \) and \( E \) are respectively the number of nodes and edges in the conflict graph \( G \).

Conflict cycle detection and conflict cycle removal (cf. the node splitting process in Section 3.4) processes are carried out in an iterative manner. Each time a conflict cycle is detected, the conflict cycle removal process is invoked to remove the conflict cycle. Further rounds of detection and removal are performed until the graph \( G \) is conflict cycle-free. As described in the experimental results below, total runtime for the whole process, including the conflict cycle detection,
conflict cycle removal and min-cost color assignment, is reasonable: less than 12 minutes for layouts of more than 545K features (2.1M rectangles after fracturing).\footnote{This runtime includes all stages: layout partitioning, all rounds of conflict cycle detection and removal, ILP-based color assignment, etc. The total runtime of BFS-based conflict cycle detection (Algorithm 1 and 2) across all conflict cycle detection rounds is less than 0.42 seconds.}

**Algorithm 1** Conflict cycle detection algorithm.

**Input:** Conflict graph \( G \).

**Output:** Report the nodes in one conflict cycle if there are any conflict cycles.

1. Set distance \( d \leftarrow 0 \) for each node \( n_i \in G \).
2. Make a queue \( Q \) and enqueue node \( n_0 \in G \) into \( Q \).
3. Set distance \( d_0 \leftarrow 0 \) for \( n_0 \).
4. while \( Q \) is not empty do
5. Dequeue the first node \( n_i \) in \( Q \).
6. for all nodes \( n_k \) adjacent to \( n_i \) do
7. if \( d_k = d_i \) then
8. Report a conflict cycle as given in Algorithm 2;
9. end if
10. end for
11. end if
12. end for
13. Set \( d_s \leftarrow d_i + 1 \).
14. Enqueue \( n_s \) into \( Q \).
15. end if
16. end while

**Algorithm 2** Conflict cycle reporting algorithm.

**Input:** Conflict graph \( G \) with marked distances and nodes \( n_j \) and \( n_k \) in the detected conflict cycle in Algorithm 1.

**Output:** Report the nodes in the detected conflict cycle in a double-linked list, where edges exist between adjacent nodes in the list.

1. Make a map \( F \) to store the father node for each node.
2. Set \( F(n_j) \leftarrow 0 \), \( F(n_k) \leftarrow 0 \);
3. Make a queue \( Q \) and enqueue nodes \( n_j \) and \( n_k \) into \( Q \);
4. while \( Q \) is not empty do
5. Dequeue the first node \( n_i \) in \( Q \);
6. for all nodes \( n_s \) adjacent to \( n_i \) do
7. if \( n_s \) is visited then
8. Make a double-linked list \( L \);
9. Push \( n_i \) into \( L \);
10. Push \( n_s \) to the front of \( L \);
11. Set \( n_j \leftarrow F(n_i) \);
12. while \( n_j \neq \text{null} \) do
13. Push \( n_j \) to the front of \( L \);
14. Set \( n_j \leftarrow F(n_j) \);
15. end while
16. Set \( n_j \leftarrow F(n_i) \);
17. while \( n_j \neq \text{null} \) do
18. Push \( n_j \) to the front of \( L \);
19. Set \( n_j \leftarrow F(n_j) \);
20. end while
21. return \( L \);
22. end if
23. Set \( F(n_i) \leftarrow n_e \);
24. Mark \( n_e \) as visited;
25. Enqueue \( n_s \) into \( Q \);
26. end if
27. end for
28. end while

**3.4 Node Splitting**

Node splitting is applied to nodes in conflict cycles so that we may eventually obtain a graph without any conflict cycles. Each time a conflict cycle is detected, we compute the overlap lengths over all possible node splits that remove the conflict cycle (recall Figure 4). For each node with achievable overlap length greater than the required overlap margin, the node splitting process is carried out to split the node into two nodes and eliminate the conflict cycle. We apply an ILP-based coloring algorithm (Section 3.6 below) to decide which nodes to split in the final decomposition result having a minimized number of cuttings and maximized overlap lengths. The conflict graph is then updated with newly generated nodes and updated edges, and another iteration with BFS-based conflict cycle detection begins. The time complexity of node splitting is \( O(C) \), where \( C \) is the number of nodes in the conflict cycle.

**Definition 2:** The node projection \( P_{1,i} \) from node \( n_i \) to node \( n_j \) is a set of points on \( n_i \) that have distance to node \( n_j \) less than \( t \).

**Fact 1:** In the conflict graph, node projections between each pair of nodes that are adjacent in the conflict graph are non-empty.

![Node projection examples](image)

**Figure 6:** Node projection examples.

Figure 6 shows two examples of node projections. We now give a more precise description of node splitting.

**Definition 3:** A horizontal (vertical) merged projection \( m_h(P) (m_v(P)) \) for a given projection \( P \) on node \( n \) is the union of \( P \) and all the projections on \( n \) that horizontally (vertically) overlap with \( P \).

**Definition 4:** Node projections are separable if they are disjoint.

**Definition 5:** The overlap length of a newly generated node for the corresponding dividing point is the length that the node can be extended across the dividing point without introducing new edges in the conflict graph.

**Rule-based node splitting:** Given a conflict cycle and a node \( n_i \) in the cycle, if (i) the horizontal (vertical) merged projections \( m_h(P_{1,i}) \) \( m_v(P_{1,i}) \) correspond to adjacent nodes \( n_j \) and \( n_k \) are separable, (ii) the resulting overlap lengths of the horizontal (vertical) splitting are not less than the given overlap margin, and (iii) there are no design rule violations after splitting, then node \( n_i \) can be horizontally (vertically) split into two nodes to remove the conflict cycle. The dividing point may be chosen in between the merged projections such that no merged projections are cut, and no violations of overlap margin or design rules occur.

![Example of rule-based node splitting](image)

**Figure 7:** Example of rule-based node splitting: \( o_{1,j} \) and \( o_{1,k} \) are overlap lengths.

Figure 7 shows an example of rule-based node splitting. In the figure, assume there is a conflict cycle between nodes \( n_i \), \( n_j \) and \( n_k \), and the horizontal merged projections \( m_h(P_{1,i}) = P_{1,i} \cup P_{1,i} \) and \( m_h(P_{1,k}) = P_{1,k} \) on node \( n_i \) corresponding to nodes \( n_j \) and \( n_k \) are separable with overlap lengths not less
than the given overlap margin. Hence, node 4 can be split into two new nodes 3 and 4 at the dividing point, with corresponding overlap lengths of 3 and 4. The dividing point is in between the lower point of 4 and the upper point of 5. Generally, the position of the dividing point is chosen so as to maximize the smaller overlap length. A more detailed illustration of overlap length is given in Figure 8, where the two touching features 4 and 5 are assigned different colors, and thus the overlap between 4 and 5 is required to be larger than the given overlap margin to guarantee successful manufacturing. The overlap lengths of the touching features 4 and 5 are denoted as 3 and 4, respectively. When computing overlap length, two features of the same color cannot be extended such that the distance between them is less than the minimum coloring spacing (e.g., in the figure, 4 cannot be extended to touch the projection of feature 7).

Figure 8: Example of overlap length calculation: 3 and 4 are the overlap lengths for 4 and 5, respectively.

Figure 9: Example of unresolvable conflict cycle (uCC): (a) uCC with zero overlap length and (b) uCC with non-zero overlap length (less than the overlap margin).

Of course, not all conflict cycles can be eliminated by the node splitting method. DPL layout decomposition fails when pattern features within the color spacing lower bound cannot be assigned different colors. Such a failure, which we call an unresolvable conflict cycle (uCC), consists of two cases: (a) there is no dividing point to remove the conflict cycle among all of rectangles which have nonzero overlap length, and (b) the overlap length is less than the overlap margin, even if there is a dividing point to remove the conflict cycle. Figure 9 illustrates these two types of uCC. If we divide the rectangle in the center as shown in Figure 9(a), the size of the rectangle violates the minimum design rule (CD). Removal of the conflict cycle may be achieved by layout perturbation which increases the spacing to neighboring patterns to be > t (as shown in orange color). In Figure 9(b), the overlap length at the dividing point is less than the required overlap margin. A fix by layout perturbation is similarly available. We observe that the space required to increase overlap length is less than that required to remove the conflict cycle, i.e., the pattern can be split after a smaller perturbation.

In summary, node splitting handles conflict cycles in two ways: (i) splitting a node in the conflict cycle, or (ii) reporting an unresolvable conflict cycle for layout optimization to eliminate. Whenever a conflict cycle is detected in the conflict graph, it is eliminated using one of these ways. By construction, the rule-based node splitting does not cause any violation of design rules or overlap margin for the newly generated nodes. On the other hand, if any feature split in the conflict cycle will result in a violation, then an unresolvable conflict cycle is reported. Hence, the iterative conflict cycle detection and node splitting process will terminate without any conflict cycle in the graph, and we have:

Fact 2: The iterative conflict cycle detection and node splitting method obtains a conflict graph which is 2-colorable.

3 After merging the projections 5 and horizontally, the lower point of the merged projection is the same as that of projection 5. Since there is no projections that horizontally overlap with projection 5, the horizontal merged projection of 5 is equal to 5.

35 Layout Partitioning
In most placements, the conflict graph between cells is sparse, i.e., due to the required poly-to-cell boundary and whitespace between cells, there are not many edges between the cells. As a result, many “islands” can be found in the conflict graph. At the same time, the runtime of ILP-based coloring algorithm increases dramatically when the number of nodes in the graph is large and the runtime for solving the coloring problem in the whole conflict graph is not endurable. Therefore, we merge the nodes into small clusters according to the connectivity information, with no edges or nodes of a given polygon occurring in multiple clusters. Each cluster has its separate conflict graph, and the ILP-based coloring algorithm is carried out on each cluster in sequence. Because there are no edges between clusters, and no polygon has nodes in more than one cluster, the solution is obtained as the union of solutions for all the small clusters. Our layout partitioning algorithm is given in Algorithm 3. The time complexity of Steps 2–4 is \(O(E)\), the complexity of Steps 5–8 is \(O(V' + E')\), and the complexity of Steps 9–11 is \(O(C' + V)\), where \(V'\) is the total number of edges between nodes, \(V'\) is the total number of polygons, \(E'\) is the total number of edges between polygons, and \(C'\) is the total number of clusters on polygonal layout features.

Algorithm 3 Layout partitioning algorithm.

**Input:** Conflict graph \(G\) and the mapping information from nodes to polygons.

**Output:** A set of clusters of nodes where no edges or nodes of the same polygon exist in between any pair of clusters.

1. Make a new graph \(G'\) on polygons with each node \(n'\) representing a polygon;
2. for all \(e = (u, v) \in G\) do
   3. Set edge \(e' \leftarrow (u', v') \in G'\), where \(u'\) and \(v'\) corresponds to the polygons in which rectangles of \(u\) and \(v\) are respectively located;
4. end for
5. while there is an unvisited node \(n' \in G'\) do
   6. Make a new cluster \(c_{i}'\) containing \(n'\);
   7. Perform breadth first search from \(n'\) and add all visited nodes into \(c_{i}'\);
8. end while
9. for all clusters \(c_{i}'\) do
   10. Make cluster \(c_{i}\) on the nodes of the polygons in \(c_{i}'\);
11. end for
3.6 Min-Cost Color Assignment Problem Formulation

Finally, we have:

**Min-Cost Color Assignment Problem**

**Given:** A list of rectangles $R$ which is color assignable, and maximum distance between two features, $t$, at which the color assignment is constrained.

**Find:** A color assignment of rectangles to minimize the total cost.

**Subject to:** For any two non-touching rectangles with $0 < d(i, j) \leq d_{ij}$, assign different colors.

For any two touching rectangles with $d(i, j) = 0$, if they are assigned different colors, there is a corresponding cost $c_{ij}$. $c_{ij}$ is the cost for assigning nodes $n_i$ and $n_j$ different colors. We cast this as an integer linear program (ILP):

Minimize: $\sum c_{ij} \times y_{ij}$

Subject to:

$$x_i + x_j = 1$$ (1)

$$x_i - x_j \leq y_{ij}$$ (2)

$$x_i - x_j \leq y_{ij}$$ (3)

where $x_i$ and $x_j$ are binary variables (0/1) for the colors of rectangles $r_i$ and $r_j$, and $y_{ij}$ is a binary variable for any pair of touching rectangles $r_i$ and $r_j$. Constraint (1) specifies that non-touching rectangles $r_i$ and $r_j$ within distance $t$ should be assigned different colors. Constraints (2) and (3) are used for evaluating the cost when touching rectangles $r_i$ and $r_j$ are assigned different colors. The cost for touching rectangles is defined as follows:

$$c_{ij} = \alpha \cdot f(w_{ij})/(f(l_i) \cdot f(l_j)) + \beta + \gamma/\min(\alpha_{ij}, \alpha_{ji})$$ (4)

where $w_{ij}$ is width of the rectangle edge between rectangle $r_i$ and $r_j$, $l_i$ and $l_j$ are lengths of the rectangle edges of $r_i$ and $r_j$, which are opposite to the touching edge, $\alpha_{ij}$ and $\alpha_{ji}$ are the minimum possible overlap lengths of nodes $n_i$ and $n_j$, respectively, and $\alpha$, $\beta$, and $\gamma$ are user-defined parameters for scaling the three items for different optimization objectives. Function $f$ is defined as follows.

$$f(x) = \begin{cases} FS_{\text{min}} & \forall x \geq FS_{\text{min}} \\ x & \forall x < FS_{\text{min}} \end{cases}$$ (5)

where $FS_{\text{min}}$ is minimum feature size, i.e., the threshold on the features, below which a design rule violation will occur. Our ILP problem formulation seeks to minimize design rule violations, the number of cuts on the layout polygons and maximize the overlap lengths between touching features of different colors.

**Minimizing design rule violations:** During the layout fracturing process, small rectangles may be generated due to specific polygonal layout features (e.g., $n_5$ and $n_6$ in Figure 10). According to Equation (4), it is easy to understand that higher costs will be assigned to pairs of touching rectangles of smaller sizes. By minimizing the total cost, the ILP-based problem formulation aims to minimize the design rule violations in the final layout.

**Minimizing the number of cuts:** The cuts on the layout features will introduce more line-ends, which is undesirable due to line-end shortening effects. Therefore, the number of cuts should be minimized to improve the quality of the decomposed layouts. In Equation (4), by setting the second term ($\beta$) to be greater than the first term, cut minimization can be given higher priority relative to consideration of design rules.

**Maximizing the overlap length:** Maximum overlap length satisfying the required overlap margin is also desirable. In Equation (4), a larger $\gamma$ value corresponds to more emphasis on the overlap length optimization.

Figure 10: Example of cost function: $l_5 < l_6 < FS_{\text{min}}$, $c_{i,j} = \alpha/l_5 + \beta + \gamma/\min(\alpha_{ij}, \alpha_{ji})$, $c_{i,j} = \alpha \cdot FS_{\text{min}}/(l_5 \cdot l_6) + \beta + \gamma/\min(\alpha_{ij}, \alpha_{ji})$; (4) $c_{i,j} = \alpha \cdot FS_{\text{min}}/(l_5 \cdot l_6) + \beta + \gamma/\min(\alpha_{ij}, \alpha_{ji})$; (5) $c_{i,j} > c_{i,j}$. From this computation, we get $c_{i,j} > c_{i,j}$.

Given such costs on the touching rectangles, the ILP solver can output the color assignment results as illustrated in Figure 10. Since the length of rectangle $n_5$, $l_5$, is less than the minimum feature size $FS_{\text{min}}$, the design rule will be violated if $n_5$ is assigned a different color from that of $n_6$. In this way, the cost function supports the minimization of design rule violations.

4. EXPERIMENTAL RESULTS

We empirically test our approach on one real-world design and three artificial designs. We evaluate our DPL solutions with respect to (1) solution quality, (2) scalability, and (3) correctness.

4.1 Experimental Setup

Table 1: Testcase parameters. Minimum spacing (140nm) and minimum line width (100nm) are scaled by 0.4x to 56nm and 40nm, respectively.

<table>
<thead>
<tr>
<th>Design</th>
<th>#Cells</th>
<th>#Polygons</th>
<th>#Rects</th>
</tr>
</thead>
<tbody>
<tr>
<td>AES</td>
<td>17384</td>
<td>90194</td>
<td>362380</td>
</tr>
<tr>
<td>TOP-A</td>
<td>304900</td>
<td>275650</td>
<td>1043950</td>
</tr>
<tr>
<td>TOP-B</td>
<td>60800</td>
<td>545000</td>
<td>2066800</td>
</tr>
<tr>
<td>TOP-C</td>
<td>305000</td>
<td>272500</td>
<td>10334000</td>
</tr>
</tbody>
</table>

Our layout decomposition system is implemented in C++. We use one real-world design (AES) implemented using Artisan 90nm libraries using Synopsys Design Compiler v2003.06-SP1 [2]. Because real-world synthesized netlists do not use all of the available standard-cell masters, we also run experiments with three artificial designs (TOP-A, TOP-B and TOP-C).
that instantiate more than 600 different types of cell masters from the same library. The testcases are placed with row utilizations of 70% and 90% using Cadence First Encounter v3.3 [3]. Table 1 shows key parameters of the testcases. In the original 90nm layouts, minimum spacing between features is 140nm, and minimum feature size is 100nm. To reflect future nodes with smaller feature sizes, we scale GDS layout of all testcases by a factor of $0.4\times$, which results in 56nm minimum spacing and 40nm minimum feature size.

4.2 Experimental Results

Solution quality. We sweep the color spacing lower bound as well as placement utilization, and evaluate solution quality according to various metrics, including number of conflict cycles, number of unresolvable conflict cycles, and minimum, average and standard deviation of overlap lengths. Table 2 shows the experimental results of our layout decomposition system. In Table 2, $t$ is the minimum coloring spacing, $CCs$ is the number of detected conflict cycles, $uCCs$ is the number of unresolvable conflict cycles, and $Cuts$ is the number of touching rectangle pairs with different colors. (Note that we do not consider the cuts on unresolvable conflict cycles. Thus, as the number of unresolvable conflict cycles increases, the reported total number of cuts may decrease.) The minimum, average and the standard deviation of the overlap length values for all the cuts in the final decomposed layout are also reported. We furthermore verify that there are no design rule violations in the final decomposed layout.
Our ongoing research is in the following directions.

- The two mask exposures in DPL can result in distinct CD populations with different statistical distributions, which may increase guardbanding compared to the guardband of a single-exposure process. We are investigating optimal timing/power model guardbanding under the bimodal CD distribution in DPL.

- We are introducing variability-awareness in the DPL layout decomposition cost function. Examples are (i) minimizing the difference between the pitch distributions of two masks, and (ii) minimizing the number of distinct DPL layout solutions across all instances of a given master cell (to reduce variability between the instances).

- We are also investigating the possibility of integrating all key optimization objectives, including the number of unresolvable conflict cycles, the number of line-ends, the overlap lengths between touching rectangles, etc., into a unified ILP problem formulation.

- Besides the ILP-based approach, we are also investigating combinatorial optimization frameworks, including node- and edge-deletion based graph bipartiteizations, that offer potentially attractive runtime-quality tradeoffs.

6. REFERENCES


Table 2: Experimental results of layout decomposition system (four testcases, 70% and 90% utilization).

<table>
<thead>
<tr>
<th>Design</th>
<th>t</th>
<th>CCs</th>
<th>uCCs</th>
<th>Cuts</th>
<th>min</th>
<th>mean</th>
<th>σ</th>
<th>CPU</th>
</tr>
</thead>
<tbody>
<tr>
<td>AES</td>
<td>58</td>
<td>0</td>
<td>0</td>
<td>29</td>
<td>44</td>
<td>237.7</td>
<td>149.2</td>
<td>17.6</td>
</tr>
<tr>
<td>59</td>
<td>0</td>
<td>0</td>
<td>33</td>
<td>56</td>
<td>111.5</td>
<td>58.6</td>
<td>19.1</td>
<td></td>
</tr>
<tr>
<td>60</td>
<td>0</td>
<td>0</td>
<td>33</td>
<td>56</td>
<td>111.5</td>
<td>58.6</td>
<td>19.0</td>
<td></td>
</tr>
<tr>
<td>(70%)</td>
<td>61</td>
<td>55</td>
<td>1</td>
<td>132</td>
<td>18</td>
<td>55.6</td>
<td>48.7</td>
<td>19.9</td>
</tr>
<tr>
<td>TOP-A</td>
<td>58</td>
<td>326</td>
<td>0</td>
<td>6261</td>
<td>11</td>
<td>145.2</td>
<td>81.1</td>
<td>136.7</td>
</tr>
<tr>
<td>(70%)</td>
<td>59</td>
<td>503</td>
<td>150</td>
<td>8013</td>
<td>9</td>
<td>140.4</td>
<td>77.8</td>
<td>145.8</td>
</tr>
<tr>
<td>TOP-B</td>
<td>60</td>
<td>503</td>
<td>400</td>
<td>7884</td>
<td>22</td>
<td>143.1</td>
<td>75.9</td>
<td>149.7</td>
</tr>
<tr>
<td>(70%)</td>
<td>58</td>
<td>654</td>
<td>0</td>
<td>12502</td>
<td>11</td>
<td>145.0</td>
<td>81.0</td>
<td>148.5</td>
</tr>
<tr>
<td>TOP-C</td>
<td>59</td>
<td>1004</td>
<td>300</td>
<td>15997</td>
<td>9</td>
<td>134.0</td>
<td>77.7</td>
<td>144.4</td>
</tr>
<tr>
<td>(70%)</td>
<td>60</td>
<td>1004</td>
<td>800</td>
<td>15752</td>
<td>22</td>
<td>142.9</td>
<td>75.7</td>
<td>145.1</td>
</tr>
<tr>
<td>(90%)</td>
<td>58</td>
<td>1517</td>
<td>3565</td>
<td>51400</td>
<td>11</td>
<td>96.3</td>
<td>76.0</td>
<td>141.9</td>
</tr>
<tr>
<td>AES</td>
<td>58</td>
<td>0</td>
<td>0</td>
<td>29</td>
<td>44</td>
<td>184.3</td>
<td>146.7</td>
<td>17.8</td>
</tr>
<tr>
<td>(90%)</td>
<td>58</td>
<td>0</td>
<td>0</td>
<td>33</td>
<td>56</td>
<td>130.3</td>
<td>53.9</td>
<td>17.7</td>
</tr>
<tr>
<td>TOP-A</td>
<td>59</td>
<td>503</td>
<td>150</td>
<td>8025</td>
<td>9</td>
<td>140.9</td>
<td>77.7</td>
<td>143.7</td>
</tr>
<tr>
<td>(90%)</td>
<td>60</td>
<td>503</td>
<td>400</td>
<td>7903</td>
<td>22</td>
<td>143.5</td>
<td>75.5</td>
<td>150.0</td>
</tr>
<tr>
<td>TOP-B</td>
<td>58</td>
<td>325</td>
<td>0</td>
<td>6291</td>
<td>11</td>
<td>145.1</td>
<td>80.9</td>
<td>135.1</td>
</tr>
<tr>
<td>(90%)</td>
<td>60</td>
<td>503</td>
<td>150</td>
<td>8025</td>
<td>9</td>
<td>140.9</td>
<td>77.7</td>
<td>143.7</td>
</tr>
<tr>
<td>TOP-C</td>
<td>59</td>
<td>1004</td>
<td>300</td>
<td>15990</td>
<td>9</td>
<td>134.0</td>
<td>77.8</td>
<td>144.4</td>
</tr>
<tr>
<td>(90%)</td>
<td>60</td>
<td>1004</td>
<td>800</td>
<td>15752</td>
<td>22</td>
<td>142.9</td>
<td>75.7</td>
<td>145.1</td>
</tr>
<tr>
<td>AES</td>
<td>58</td>
<td>0</td>
<td>0</td>
<td>29</td>
<td>44</td>
<td>184.3</td>
<td>146.7</td>
<td>17.8</td>
</tr>
<tr>
<td>(90%)</td>
<td>58</td>
<td>0</td>
<td>0</td>
<td>33</td>
<td>56</td>
<td>130.3</td>
<td>53.9</td>
<td>17.7</td>
</tr>
<tr>
<td>TOP-A</td>
<td>59</td>
<td>503</td>
<td>150</td>
<td>8025</td>
<td>9</td>
<td>140.9</td>
<td>77.7</td>
<td>143.7</td>
</tr>
<tr>
<td>(90%)</td>
<td>60</td>
<td>503</td>
<td>400</td>
<td>7903</td>
<td>22</td>
<td>143.5</td>
<td>75.5</td>
<td>150.0</td>
</tr>
<tr>
<td>TOP-B</td>
<td>58</td>
<td>325</td>
<td>0</td>
<td>6291</td>
<td>11</td>
<td>145.1</td>
<td>80.9</td>
<td>135.1</td>
</tr>
<tr>
<td>(90%)</td>
<td>60</td>
<td>503</td>
<td>150</td>
<td>8025</td>
<td>9</td>
<td>140.9</td>
<td>77.7</td>
<td>143.7</td>
</tr>
<tr>
<td>TOP-C</td>
<td>59</td>
<td>1004</td>
<td>300</td>
<td>15990</td>
<td>9</td>
<td>134.0</td>
<td>77.8</td>
<td>144.4</td>
</tr>
<tr>
<td>(90%)</td>
<td>60</td>
<td>1004</td>
<td>800</td>
<td>15752</td>
<td>22</td>
<td>142.9</td>
<td>75.7</td>
<td>145.1</td>
</tr>
</tbody>
</table>