GREMA: Graph Reduction Based Efficient Mask Assignment for Double Patterning Technology

Yue Xu
Electrical and Computer Engineering
Iowa State University
Ames, IA 50011
yuexu@iastate.edu

Chris Chu
Electrical and Computer Engineering
Iowa State University
Ames, IA 50011
cnchu@iastate.edu

ABSTRACT

Double patterning technology (DPT) has emerged as the most hopeful candidate for the next technology node of the ITRS roadmap [1]. The goal of a DPT decomposer is to decompose the entire layout on each layer onto two masks. It assigns two features to different masks if their spacing is less than a predefined threshold. Besides, some features must be sliced and put onto two masks so that there would be a feasible solution for mask assignment. Such slicing will cause stitches that affect yield. So decomposer needs to minimize their number.

In this paper, we formulate the DPT decomposition problem as a maximum cut problem. We propose an extremely efficient two-stage decomposition algorithm called GREMA. The first stage of GREMA generates a set of candidate stitches to ensure that feasible solutions exist for DPT decomposition. The second stage uses maximum cut to find the minimal set of stitches. Our decomposer is able to solve much larger realistic design problems. Experiments demonstrated that GREMA achieved great performance on resolving conflicts with greatly reduced runtime.

1. INTRODUCTION

As the feature size of modern VLSI design continues to shrink and the image of photoresist patterns starts to blur at $45nm$ technology, previous lithography technology becomes obsolete for sub-$32nm$ technology. Due to the delay of the high index fluids immersion technology and Extreme Ultra Violet (EUV) technology, double patterning technology (DPT) [2] [3] [4] [5] has become one of viable lithography techniques for the $32nm$ node. Double patterning process uses two independent rounds of exposure and etching to form patterns on silicon wafer for each layer. Patterns on each layer are assigned to two masks. DPT can relax the minimum pitch of a circuit layout for individual decomposed exposure. This relaxation requires DPT decomposer to put a pair of patterns onto different masks if the minimum distance between them is less than a predefined DPT threshold, which is commonly set as twice the minimum spacing. While some works uses Euclidean distance spacing rules, some others, [6] and [7], use Manhattan distance spacing rules. In this paper, we use the latter type of rules.

We define DPT conflict between a pair of nearby patterns as the necessity for putting the patterns onto different masks. The entire set of conflicts needs to be resolved by assigning conflicting patterns onto different masks. However, due to the complex geometric relationship among patterns, successful decomposition usually could not be achieved. We define it as infeasibility because the initial layout is infeasible for DPT decomposition. Usually, infeasibility can be resolved by slicing single patterns and assigning sliced parts on to different masks. We call each slicing a candidate stitch. Resolving infeasibility by slicing patterns does not come free, it may generate stitches. Stitches tend to induce pinching or bridging that can affect yield so decomposition algorithms should generate as few stitches as possible to keep yield at a reasonable level. Thus, an optimal solution for double patterning decomposer is a conflict-free decomposition with minimal stitches.

DPT has already been successfully applied to DRAM manufacturing [2]. However, for random logics, one critical stage in the integration of DPT into current design flow is the automation of double patterning decomposition. Chiou et al. [8] categorize the DPT decomposition methodology into two types: model-based approach and rule-based approach. The more accurate model-based decomposition approach is based on optical simulation. It suffers from the extreme long runtime that makes it impractical for full-chip implementation. Meanwhile, the rule-based approach slices and assigns patterns based on geometric relationship between patterns. Thus it can achieve better efficiency.

There are limited works focusing on the decomposition algorithms. Recent publications, [8] [7] [9] [10] and [11], all use rule-based methods. [8], [7] and [9] use heuristics to greedily slice and assign patterns to resolve local conflicts and infeasibility. We find that the previous heuristic methods tend to narrowly focus on local pattern conflicts and overlook sequence of conflicts that exists on a larger scale. As a result, they either suffer from the inability of resolving all conflicts or lead to an unnecessarily large number of stitches.

On the other hand, Yuan et al. [10] formulate DPT decomposition problem as a 2-coloring problem and pre-slice patterns into squares with a side length of half pitch. The slicing virtually turns the boundary of each touching square

[1]This work was partially supported by IBM Faculty Award and NSF under grant CCF-0540998.

ICCAD'09, November 2–5, 2009, San Jose, California, USA.

pair into a candidate stitch. They employ ILP to resolve conflicts and minimize the number of stitches simultaneously. The problem size is prohibitive so that [10] uses partition heuristic to bring down the problem size. Although ILP-based methods can achieve the optimal solution theoretically, the partition heuristic ruins the optimality. The main contribution of [11] is a node splitting technique that focuses on how to slice patterns to resolve infeasibility. Besides, the work formulates an ILP to minimize design rule violations, the number of stitches and to maximize the overlap length of stitches. With a larger pre-sliced scale, it improves the scalability of ILP-based coloring problem.

In this work, we develop a very efficient and optimal rule-based two-stage double patterning layout decomposition tool called GREMA. It can be integrated into current EDA tools to generate double patterning solutions and notify designer if infeasibility persists and requires design modification. In the first stage, GREMA constructs a data structure called *conflict graph (CG)* that captures all the DPT conflicts in the layout. It then identifies all the infeasibility and generates candidate stitches to resolve infeasibility. By an analogy between assigning patterns onto masks and assigning nodes into two partitions, the second stage of GREMA models the conflict resolving and stitch minimizing DPT decomposition problem as a maximum cut problem in a data structure we called *decomposition graph (DG)*. Our key contributions are:

- We propose a two-stage scheme for DPT decomposition that separates the stage generating candidate stitches and the stage minimizing the number of stitches.
- GREMA does not use any pre-slicing technique so it generates a smaller set of candidate stitches to resolve the infeasibility.
- We abstract decomposition graph into a much simplified graph, called *flipping graph (FG)* that stores all the information of candidate stitches.
- We propose ways to simplify FG, which greatly reduce the problem size of our max-cut formulation.

The rest of this paper is organized as follows. Sec. 2 introduces the general concepts in DPT and gives an overview of GREMA. Sec. 3 describes the techniques to find and resolve infeasibility. Sec. 4 shows the formulation of the mask assignment problem, the simplification methods and solution for the problem. The experimental results are provided in Sec. 5 and we conclude in Sec. 6.

2. OVERVIEW OF GREMA

As stated in Sec. 1, the primary goal of DPT decomposition is mask assignment. Due to the infeasibility in layouts, decomposers have to introduce stitches to resolve the infeasibility. The decomposition problem can be divided into infeasibility resolving and mask assignment. So a natural flow of decomposer is to identify all the infeasibility, generate candidate stitches and do mask assignment for patterns. Following this nature flow, GREMA adopts a systematic two-stage approach with careful selection of candidate stitches and maximum cut based mask assignment.

The first stage of GREMA identifies and resolves infeasibility. First, GREMA constructs conflict graph to capture all the DPT conflicts in the layout. Then GREMA identifies odd cycles in CG, which represents the infeasibility. Since stitch is not necessary unless DPT decomposition infeasibility exists in the layout, GREMA does not introduce any pre-slicing that dramatically increases the problem size of mask assignment. GREMA only generates candidate stitch on patterns involved in infeasibility.

With candidate stitches and conflicts at hand, we formulate the mask assignment problem as a maximum-cut problem on decomposition graph. In DG, every node represents a pattern or a part of sliced pattern in the layout. There are two types of edges in DG: positively weighted *conflict edge* that represent DPT conflict and negatively weighted *stitch edge* that represent candidate stitch. As nodes are put into two partitions, we want to find a cut that crosses all conflict edges and as few stitch edges as possible. The cut we look for is actually the maximum cut on DG. Cutting all conflict edges is guaranteed by max-cut formulation because every conflict edge is set to have a weight too large to ignore. With the negative weight, the stitch edges would be avoided by the max-cut. Hence, the number of stitches will be minimized. Generally, the max-cut problem for a graph with both positively and negatively weighted edges is NP-hard.

In the second stage, GREMA optimizes the conflict resolving, stitch minimization mask assignment problem. Because all necessary candidate stitches are generated, nodes connected with conflict edges can be sequentially assigned to masks without two conflicting nodes residing on the same mask. We denote each connected component after ignoring all the stitch edges in DG as a *cluster*. GREMA abstracts each cluster into a node and converts DG into flipping graph. In FG, there only exists stitch edge. We call the graph flipping graph because there is still freedom to flip the mask assignment for each cluster. The cluster abstraction accomplishes conflict resolving and the only task left for GREMA is stitch minimization, for which GREMA formulates a maximum cut problem on FG. GREMA simplifies the FG and uses ILP to optimize the maximum cut problem.

It is worth noticing that in some situations, named as intrinsic infeasibility, the infeasibility persists no matter how candidate stitches are generated. This indicates that under design rules, the original layout cannot be DPT decomposed. GREMA will report the patterns involved in intrinsic infeasibility to designers.

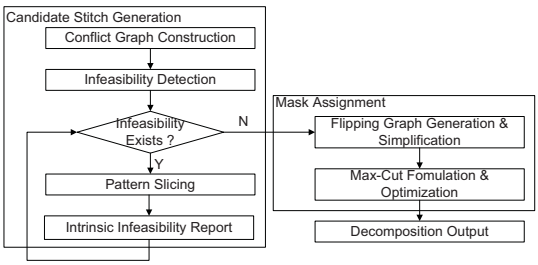


Figure 1: GREMA flow.

The flow of GREMA is illustrated in Fig. 1. First, GREMA constructs the conflict graph based on the conflicts in the layout. Then GREMA runs infeasibility detection algorithm on the conflict graph, slices all suitable patterns involved in the infeasibility and generates a set of candidate stitches. If any intrinsic infeasibility exists, GREMA reports the patterns that induce the intrinsic infeasibility. Once infeasibility is no longer an issue, GREMA generates and simplifies the flipping graph and uses an ILP-based max-cut algorithm

to minimize the number of stitches in the final mask assignment. Finally, GREMA outputs the decomposition results.

3. CANDIDATE STITCH GENERATION

We use conflict graph to capture all the DPT conflicts in the original layout. In CG, every node represents a pattern in the layout. Conflict edges are created between every pair of DPT conflicting nodes. We can use CG to identify all the infeasibility.

Looking into the conflict graph, we find that only one type of structure represents the infeasibility. The structure is odd cycle. We put one node in the odd cycle into one partition, put its neighbors along the cycle into the other partition and sequentially do the partition assignment for the rest of the nodes in the cycle. To the end of this operation, the last node will have its two cycle-neighbors in different partitions. So no matter how we place the last node, there would be a conflict edge with both of its endpoints in one partition. In mask assignment, we would have two conflicting patterns on the same mask, which is a violation of DPT rules. So GREMA needs to break all the odd cycles in order to generate a valid DPT decomposition

We can break an odd cycle and resolve the infeasibility by slicing one node in the cycle. Slicing generates two new nodes to replace the sliced node and creates a stitch edge between the newly created nodes. It converts a CG into DG. Such operation represents the creation of a candidate stitch so it is possible to assign two parts of a single pattern onto different masks.

Ideally, we can slice just one node to break an odd cycle. However, due to the lack of the entire view, randomly choosing and slicing one pattern for each odd cycle cannot guarantee the minimum number of stitches. For instance, we can slice a shared node between two touching odd cycles and the slicing will break the two odd cycles while creating an even one. Meanwhile, the randomly choosing and slicing method will most likely choose one node on each cycle and slice two nodes. So the cycle-detection based heuristic described above will generate a number of stitches up to two times the optimal value. This is why GREMA does not use the fast heuristic.

Because we cannot determine optimal stitches based on the information derived from an odd cycle, we will slice all the nodes along the odd cycle depending on their suitability of slicing, modeled by the stitch shield.

A stitch shield will prohibit candidate stitch appearing at certain locations because some slicing will not resolve infeasibility at all. One example is pattern B in Fig. 2(a) and Fig. 2(b). No matter how we slice pattern B, the sliced parts will have conflict with the same region of pattern A. The two sliced parts have to stay in the same mask, which makes the slicing useless. In order to generate useful candidate stitch, we take consideration of impacts from neighboring patterns to choose which pattern and where in the pattern to make the stitch and break the odd cycle. So part of our stitch shield incorporates the idea of node projection idea introduced in [11]. Furthermore, GREMA creates stitch shield at places where slicing would introduce pattern parts that violate minimum feature size design rule. In GREMA, candidate stitches should not cross any stitch shield.

GREMA uses breadth first search (BFS) to find odd cycles and choose all suitable candidate stitches. Unlike previous ILP based works, GREMA does not use any pre-slicing technique so every candidate stitch is generated for the infeasibility in the original layout. Although pre-slicing may break some odd cycles even before odd cycle detection, most of the candidate stitches generated from pre-slicing are useless and they dramatically increase the problem size of the mask assignment problem, which is NP-hard. Despite of the fact that GREMA has to carry out more BFS without pre-slicing, the total runtime would still be reduced. BFS has a complexity proportional to the number of edges during the search so the runtime for a few more rounds of BFS is negligible. More importantly, GREMA generates a much smaller candidate stitch set. In such a way, GREMA effectively controls runtime by reducing the size of the computational intensive mask assignment problem.

In Fig. 2(a), we have an example of intrinsic infeasibility. The conflicting parts of odd cycle $A \sim B \sim L \sim A$ are all stitch shielded so no candidate stitch could be introduced to break the odd cycle. GREMA will report all three patterns to designers. Moreover, the least connected pattern A would be bypassed during the later mask assignment stage, in an effort to resolve as many conflicts as possible. GREMA will continue and check whether the layout has too many stitches that bring down yield to an unacceptable level. If there are too many stitches, designers might be forced to redesign the entire layout.

An example of decomposition graph after conflict resolving is shown in Fig. 2(c). The decision on which stitches to make in the final solution is left to mask assignment algorithm, the second stage of GREMA.

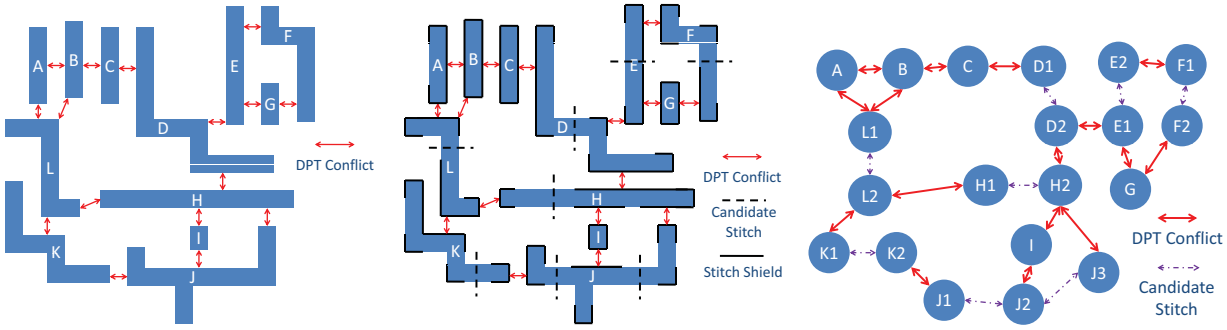
4. MASK ASSIGNMENT

Once every infeasibility has been taken care of, as stated in Sec. 3, GREMA needs to assign patterns to two masks to form a DPT decomposition solution.

4.1 Initial Mask Assignment for Clusters

Since infeasibility has been resolved in the last stage, GREMA can put one node in a partition, put all its conflict edge connected neighbors in the other partition and sequentially propagate the partition assignment for conflict edge connected components with a property that no conflict edge would have both of its endpoints assigned to the same partition. We call each conflict edge connected component a cluster and each cluster only have two choices during mask assignment.

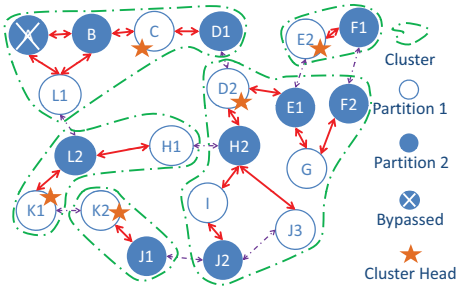
Instead of letting ILP decide mask assignment for each node in clusters, GREMA carries out the partition propagation discussed above so that ILP only needs to choose between the two choices for each cluster. In such a manner, we abstract linear complexity sub-problems from NP-hard mask assignment. GREMA arbitrarily picks up a node in each cluster and denotes the node as cluster head. Initially, every cluster head is assigned to partition 1. After the partition propagation, GREMA generates a bipartite cut for each cluster, with the endpoints of conflict edges on the cut assigned to different partitions, as shown in Fig. 3(a). Besides, it actually generates the first mask assignment solution, though it probably would not be optimal in terms of stitch minimization. Between clusters, there only exist stitch edges and cutting across them will generate stitches. We want to have a cut crossing as few stitch edges as possible. For some stitch edges in the initial mask assignment solution, their two endpoints might reside on different masks,



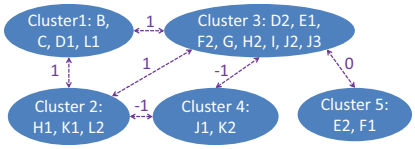
(a) Layout Example (b) Stitch Shield and Candidate Stitch (c) The DG Corresponding to (b)

Figure 2: Layout, Candidate Stitch Generation and Decomposition Graph

indicating that the candidate stitch is chosen and the specific stitch is made for the sliced pattern. For some stitch edges with both of their endpoints in the same partition, the candidate stitches are not chosen. To reduce the number of stitches, GREMA can generate subsequent solutions by flipping the partitions for clusters. In Fig. 3(a), node A is bypassed because it is the least connected component in some intrinsic infeasibility.



(a) Cluster Initialization



(b) Flipping Graph

Figure 3: Flipping Graph Generation

4.2 Generation of Flipping Graph

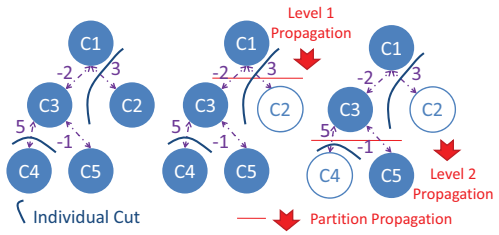
We abstract each cluster in DG into a single node and create a data structure called flipping graph, as shown in Fig. 3(b). The edges in FG have a weight of flipping gain, which we will describe later.

For two clusters C_i and C_j, we may have n_d stitch edges with the endpoints of each edge in different partitions and n_s stitch edges with their endpoints in the same partition. If we flip the partition of either cluster, there would be n_d stitches before the flipping and n_s stitches after the flipping. We define flipping gain, i.e. $fg_{i,j}$, as the difference $n_d - n_s$ for C_i and C_j. The flipping gain is the number of stitches reduced by putting the cluster heads of C_i and C_j into different partitions, comparing to the initial solution that they reside in the same partition.

GREMA formulates the stitch minimization mask assignment problem as a maximum cut problem on FG. Each edge being cut will have the cluster heads of its two endpoints assigned to different partitions. GREMA can change the cut solutions by flipping the mask assignment of each cluster. The cut solution will realize the flipping gain of the edges on the cut. The larger the sum of realized flipping gain, the more stitches GREMA will save, comparing to the initial solution in which all the cluster heads are assigned to cluster 1.

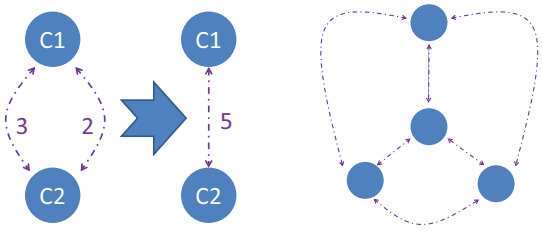
4.3 Simplification of Flipping Graph

Even though the FG presented in 3(b) has greatly reduced the problem size of mask assignment, we can ignore or greatly simplify certain structures in FG to further reduce runtime.



(a) Tree Structure Partition Propagation

(b) Serial Edge Simplification



(c) Parallel Edge Simplification (d) An Irreducible FG

Figure 4: FG Simplification Examples

For subgraphs of FG in tree structure, if the flipping gain of one edge is positive, GREMA will cut it, as shown in Fig. 4(a). Since cutting in tree structure can be made without

affecting cutting on other edges, GREMA can temporarily ignore these subgraphs and propagate the partitions after the partition for the rest of FG has been optimized, as shown in Fig. 4(a).

For nodes in a cycle, the two edges connected to a degree-two node can be merged into a single edge. As shown in Fig. 4(b), if GREMA decides to cut the new edge, it has to cut either one of the two original edges. It is obvious that it should cut the higher weighted edge, realizing a flipping gain of $max(fg_{e1}, fg_{e2})$. There are two cases if GREMA decides not to cut the new edge. It can cut both of the original edges, realizing a flipping gain of $fg_{e1} + fg_{e2}$. Or it can choose to cut neither of the original edges and realize a flipping gain of 0. GREMA chooses the better result from the two cases and realizes a flipping gain of $max(fg_{e1} + fg_{e2}, 0)$. The flipping gain of the new edge turns out to be $max(fg_{e1}, fg_{e2}) - max(fg_{e1} + fg_{e2}, 0)$.

Though parallel edges shown in Fig. 4(c) do not exist in the initial FG, they may emerge after the serial edge simplification. For parallel edges between two nodes, GREMA either cuts or does not cut both of the edges, so the flipping gain of the merged parallel edge is $fg_{e1} + fg_{e2}$.

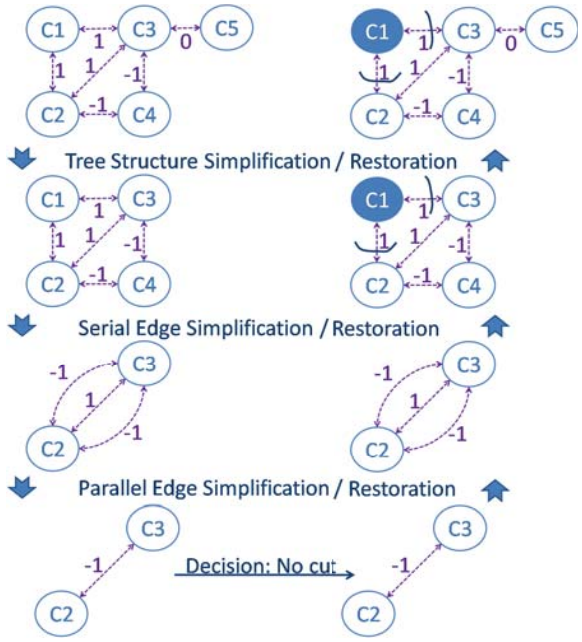


Figure 5: Simplification Procedure for the FG in Fig. 3(b)

GREMA conducts simplification according to the following procedure. At the beginning, GREMA detects all degree-one node, removes them temporarily from the FG and continues this degree-one node removal until all the nodes in FG is at least degree-two. This will remove all subgraphs in tree structures. GREMA carries out serial edge simplification and then parallel edge simplification. After the three types of simplification, GREMA starts a new round of simplification from tree structure removal because structures that could be simplified may emerge after previous simplifications. The simplification process will stop for two cases. In the first case, the simplified FG has only two nodes and one edge. Based on the weight of the edge, GREMA decides whether to cut the edge and flip the partition for one of the node. In the second case, every node in the simplified FG is at least degree three and GREMA formulates an ILP to generate the optimized max cut. The simplification process for the flipping graph in Fig. 3(b) is given in Fig. 5, the FG is so greatly simplified that GREMA does not even need to formulate an ILP.

After the max cut for the final simplified FG is optimized, GREMA restores previously simplified structures and extends the cut decision onto these structures. For a new edge generated from parallel edge simplification, the cut decision on the new edge will be replicated for the original edges. For a new edge generated from serial edge simplification, if GREMA decides to cut it, GREMA will cut the larger weighted edge of the original two edges. If GREMA decides not to cut the new edge, it will cut both of the two original edges if the sum of their weights is positive and will not cut if the sum is negative. For tree structured subgraphs temporarily removed, GREMA will recover them and carry out the tree structure partition propagation shown in Fig. 4(a). Fig. 5 also shows the structure restoration and decision extension process.

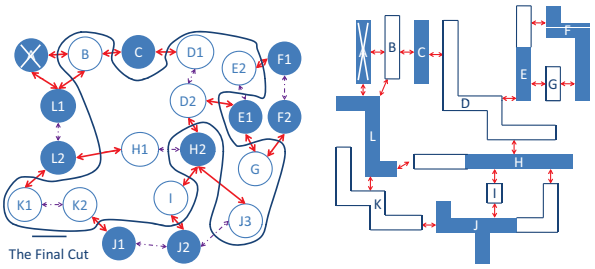
4.4 ILP Formulation for Max Cut Problem

Despite of the fact that GREMA simplifies FG in every possible way, the stitch minimizing flipping problem is still intrinsically NP-Hard and an irreducible example is given in Fig. 4(d). To achieve the optimality of stitch minimization, we formulate the flipping-gain maximization problem as an ILP, as shown below:

$$\begin{aligned} OBJ \quad &: \quad max \sum_{i,j} a_{i,j} \\ s.t. \quad &: \quad a_{i,j} \le fg_{i,j}(4 - (x_i + x_j)) \quad \forall i,j \\ &\quad\quad a_{i,j} \le fg_{i,j}((x_i + x_j) - 2) \quad \forall i,j \\ &\quad\quad x_i \in \{1,2\} \quad \forall i \end{aligned} \tag{1}$$

In the ILP formulation, x_i represents which partition the cluster head of cluster C_i belongs to. $fg_{i,j}$ is the flipping gain for the edge between clusters C_i and C_j. $a_{i,j}$ is the realized flipping gain. We can see that when x_i and x_j are the same, $a_{i,j} \le 0$. When x_i and x_j are different, $a_{i,j} \le fg_{i,j}$.

From the optimized solution, GREMA infers the flipping of cluster partitions and assigns patterns in each cluster to masks according to their partition assignment. Fig. 6(a) and Fig. 6(b) show the maximum cut on DG and mask assignment for the layout given in Fig. 2(a) respectively.



(a) Max Cut on DG (b) Final Mask Assignment

Figure 6: Max Cut and Mask Assignment Solutions

5. EXPERIMENTAL RESULTS

We implement GREMA in C and use linear programming solver QSopt [12]. All the experiments are performed on a

Linux machine with $2.8GHz$ Intel Xeon and $4GB$ RAM. We run GREMA on the benchmarks in [11]. The statistics of the benchmarks are shown in Table 1. These benchmarks use cells from *Artisan* $90nm$ libraries with 140nm minimum spacing and $100nm$ minimum line width. They are scaled by $0.4\times$ down to $56nm$ and $40nm$ respectively during the experiment. In the five benchmarks, AES is a real design. The artificial TOP-A, TOP-B, TOP-C and TOP-D instantiate more types of cells to generate more DPT conflicts and intrinsic infeasibility to test the full potential of GREMA. The benchmarks are placed with row utilizations of 90%. The DPT threshold is set to be 60nm.

Table 1: Experimental Results for GREMA

		GREMA		[11]	
Design	# Patterns	# Stitches	cpu(s)	# Stitches	cpu(s)
AES	90394	30	5.9	33	17.8
TOP-A	275650	6191	15.4	7903	155.0
TOP-B	545000	10265	32.2	15755	500.2
TOP-C	2725000	64385	310.7	78709	4655
TOP-D	1090000	24767	85.0		

To demonstrate the effectiveness of various graph reduction techniques used in GREMA, Table 2 shows the average number of nodes in the connected components in different types of graphs during GREMA flow. We exclude the connected components in CG with less than three nodes in Table 2. They do not induce infeasibility and stay unchanged throughout the flow of GREMA. When GREMA introduces stitches to resolve infeasibility, the average number grows 2.94 times. The transformation of DG into FG reduces the average number by 2.85 times and the simplification techniques further bring down it by 2.06 times. Most of the connected components are fully simplified by GREMA as the final average number is approaching 2, where GREMA does not need ILP to figure out a solution. Actually, less than 10% of connected components need to go through ILP to achieve a solution.

Table 2: Transition for the Number of Nodes in Connected Component

Design	CG → DG	DG → FG	FG Simplification
AES	3.23 → 5.61	5.61 → 2.13	2.13 → 2.00
TOP-A	4.28 → 13.89	13.89 → 4.81	4.81 → 2.08
TOP-B	4.26 → 13.73	13.73 → 4.66	4.66 → 2.05
TOP-C	4.26 → 14.02	14.02 → 4.85	4.85 → 2.08
TOP-D	4.27 → 13.77	13.77 → 4.79	4.79 → 2.06

We compare the performance of GREMA with [11] in Table 1. # Stitches is number of stitches for the mask assignment solution and cpu is the total runtime. [11] did not report its results for TOP-D. Due to the reason that [11] considers the maximization of overlap length of stitches, which may significantly increase the number of stitches, the number of stitches from GREMA and [11] is not directly comparable. However, we hold the believe that generating less stitches is a much more desirable property for DPT decomposer.

GREMA runs roughly one order of magnitude faster than [11]. More importantly, GREMA shows bigger speed up for larger benchmarks. As shown in Fig. 7, the runtime of GREMA is almost linearly proportional to the number of patterns in the benchmarks.

We also tried to solve the DPT decomposition problem using the formulation presented in [10] to investigate the effectiveness of the techniques used in GREMA. Unfortunately, all benchmarks except for AES consume too much memory to run. For the only benchmark we can run, we cannot achieve the solution after 24 hours of computation.

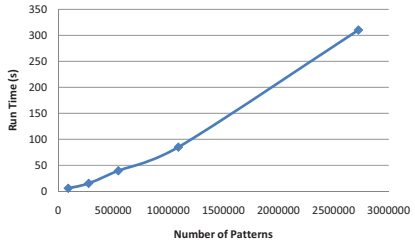


Figure 7: GREMA Runtime vs. Number of Patterns

6. CONCLUSION

In this paper, we presented a new DPT decomposition tool called GREMA. GREMA uses a two-stage flow that first resolves infeasibility and later assigns patterns onto masks to minimize the number of stitches. Based on the avoidance of pre-slicing, abstraction of decomposition graph into flipping graph and the simplification of FG, GREMA achieves greatly reduced run-time without loss of optimality, comparing to previously published works.

Our future work will be devoted to the refinement of this work, an investigation into the probability to replace ILP with other more efficient formulation and the automation of DPT guided layout modification.

7. REFERENCES

[1] http://www.itrs.net/reports.html

[2] Chang-Moon Lim et al., "Positive and negative tone double patterning lithography for 50nm flash memory," Proc. SPIE 6154, 2006

[3] Jungchul Park et al., "Application challenges with double patterning technology (DPT) beyond 45 nm," Proc. SPIE 6349, 2006

[4] Mircea Dusa, et al., "Pitch doubling through dual-patterning lithography challenges in integration and litho budgets," Proc. SPIE 6520, 2007

[5] Martin Drapeau et al., "Double patterning design split implementation and validation for the 32nm node," Proc. SPIE 6521, 2007

[6] Yuichi Inazuki et al., "Decomposition difficulty analysis for double patterning and the impact on photomask manufacturability," Proc. SPIE 6925, 2008

[7] George E. Bailey et al., "Double pattern EDA solutions for 32nm HP and beyond," Proc. SPIE 6521, 2007

[8] Tsann-Bim Chiou et al., "Development of layout split algorithms and printability evaluation for double patterning technology," Proc. SPIE 6924, 2008

[9] Anton van Oosten et al., "Pattern split rules! A feasibility study of rule based pitch decomposition for double patterning," Proc. SPIE 6730, 2007

[10] Kun Yuan et al., "Double Patterning Layout Decomposition for Simultaneous Conflict and Stitch Minimization," International Symposium on Physical Design (ISPD), San Diego, March 2009

[11] A.B. Kahng et al., "Layout decomposition for double patterning lithography," ICCAD 2008, Nov. 2008

[12] http://www2.isye.gatech.edu/ wcook/qsopt/