Area Fill Generation With Inherent Data Volume Reduction *

Yu Chen[†], Andrew B. Kahng[‡], Gabriel Robins[§] Alexander Zelikovsky[¶] and Yuhong Zheng[£]

[†]Computer Science Department, UCLA, Los Angeles, CA 90095-1596

[‡]CSE and ECE Departments, UCSD, La Jolla, CA 92093-0114

[§]Department of Computer Science, University of Virginia, Charlottesville, VA 22903-2442

[¶]Department of Computer Science, Georgia State University, Atlanta, GA 30303

[£]Computer Science and Engineering Department, UCSD, La Jolla, CA 92093-0114

yuchen@cs.ucla.edu, {abk, yzheng}@cs.ucsd.edu, robins@cs.virginia.edu, alexz@cs.gsu.edu

Abstract

Control of variability and performance in the back end of the VLSI manufacturing line has become extremely difficult with the introduction of new materials such as copper and low-k dielectrics. Uniformity of chemical-mechanical planarization (CMP) requires the insertion of area fill features into the layout, in order to smoothen the variation of feature densities across the die and thus improve manufacturability. Because the size of area fill features is very small compared with the large empty layout areas that must be filled, the filling process can increase the size of a GDSII file by an order of magnitude or more. Data compression is therefore a significant issue in successful fill synthesis. In this paper, we introduce compressed fill strategies which exploit the GDSII array reference record (AREF) construct. We apply greedy and linear programming based optimization techniques, and obtain practical compressed filling solutions.

1 Introduction

Chemical-mechanical planarization (CMP) and other manufacturing steps in nanometer-scale VLSI processes have varying effects on device and interconnect features, depending on the local characteristics of the layout. To improve manufacturability and performance predictability, foundry rules require that a layout be made uniform with respect to prescribed density criteria, through the insertion of *area fill* geometries. Currently, area fill is added by physical verification tools (such as Mentor Graphics Calibre) in the form of a flat "target layer" [9] that is eventually merged with the actual layout geometries at the mask data preparation step of the manufacturing handoff. Interconnect layers above M1 have no natural hierarchy that can be exploited, which results in a flat filling. According to the 2001 International Technology Roadmap for Semiconductors, the fractured (MEBES format) layout data volume for a single critical layer will reach hundreds of gigabytes during the transition between 130nm and 90nm technologies [2]. To alleviate file transfer times, and to accommodate future regimes of maskless lithography (e.g., direct-write requires transfer of terabytes of layout data per second), layout data must be compressed as much as possible (required compression factors have been estimated at 20× or more [6]).

*This research was supported by a grant from Cadence Design Systems, Inc., by the MARCO/DARPA Gigascale Silicon Research Center, by a Packard Foundation Fellowship, by a National Science Foundation Young Investigator Award (MIP-9457412) by NSF grant CCR-9988331 and by the State of Georgia's Yamacraw Initiative.

In today's standard practice, the basic area fill feature is the same across the entire layout (the most common fill shape is square or rectangular). Moreover, filling patterns exhibit a high degree of spatial regularity across the layout [8]. Area fill feature dimensions scale with the underlying technology, since microloading and other mechanisms of process variability are exacerbated by large variations in feature dimensions. Thus, the number of fill geometries per layer is expected to scale with the number of non-feature geometries, i.e., at approximately 2× per technology node (ignoring the impact of reticle enhancement techniques such as OPC). To achieve the density requirements, the filling process tends to increase the size of a GDSII file by an order of magnitude, due to the small size of the area fill features relative to the large empty layout areas that must be filled. Higher data volume leads to increased read/write times and prevents leverage of hierarchical data processing, among other concerns. Thus, fill data compression becomes an important issue in successful fill synthesis.

Previous efforts in fill data compression used off-the-shelf data compression techniques including the Joint Bi-level Image Processing Group (JBIG), Ziv-Lempel(LZ77) and ZIP. A data processing system architecture and three compression algorithms (JBIG, LZ77, 2D-LZ) were studied in [6] for a direct-write maskless lithography system, and an interesting alternative compression is suggested in [7]. However, it is not possible to represent such compression directly inside the standard GDSII data stream format, and such techniques are therefore of limited use for manufacturers. Ueki et al. in [12] proposed a data compaction algorithm for mask data processing in vector scan electron beam writing systems, where 'array' and 'cell' constructs are used to represent the data.

In this paper we explore compression possibilities inherent in the GDSII format itself. There are two types of GDSII constructs which allow compression: Structure REFerence record (SREF) and Array REFerence record (AREF). The SREF construct enables the hierarchical representation of data, e.g., using macros. A more efficient way to represent spatially regular configurations is the AREF construct, which is defined mainly by the following six parameters: x- and y-coordinates of the left-bottom corner of an array, the horizontal and vertical periods (or steps) at which a given feature (e.g., a filling geometry) is replicated in the array, and the number of columns and rows in the array. It is also possible to specify additionald optional parameters such as transformation records STRANS, MAG, and ANGLE, which specify the orientation of the instances. Clearly, the AREF construct can compress a flat area fill representation by a factor proportional to the number of the area fill features. Therefore, using larger AREFs in area fill representation achieves better compression.

Proceedings of the Design,Automation and Test in Europe Conference and Exhibition (DATE'03)
1530-1591/03 $17.00 © 2003 **IEEE**

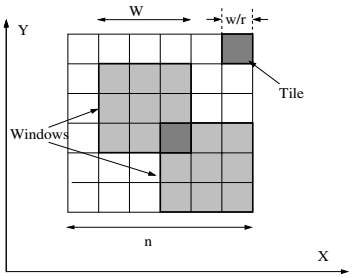

Figure 1: The fixed r-dissection regime, with $r = 3$; the n-by-n layout is partitioned by r^2 distinct overlapping dissections with window size $w \times w$.

This paper makes the following contributions:

- formulation of new problems for fill generation and compression using the GDSII AREF construct;
- new linear programming formulations for the above objectives with several practical variations;
- description of a greedy method to determine all maximal AREFs which satisfy the fill data requirement, as well as modifications that speed up the greedy algorithm; and
- experimental comparisons of different methods for compressed fill generation, confirming the advantages of our compressible fill strategies.

In the next section we briefly describe existing fill generation practices. In Section 3 we propose several problem formulations for compressed fill generation. Sections 4 and 5 outline linear programming based approaches and greedy algorithms, respectively, for these formulations. Our computational experience with the proposed methods is described in Section 6.

2 Fill Generation in Fixed-Dissection Regimes

All existing methods for the synthesis of area fill are based on discretization: the layout is partitioned into *tiles*, and filling constraints or objectives (e.g., minimizing the maximum density variation) are enforced for square *windows* each consisting of $r \times r$ tiles. Thus, to practically control layout density in *arbitrary* windows, density bounds are enforced only in a *finite* set of windows. More precisely, both foundry rules and EDA physical verification and layout tools attempt to enforce density bounds within r^2 overlapping *fixed dissections*, where r determines the "phase shift" w/r by which the dissections are offset from each other. The resulting *fixed r-dissection* (see Figure 1) partitions the $n \times n$ layout into tiles T_{ij}, then covers the layout by $w \times w$-windows W_{ij}, $i, j = 1, \ldots, \frac{nr}{w} - 1$, such that each window W_{ij} consists of r^2 tiles T_{kl}, $k = i, \ldots, i+r-1$, $l = j, \ldots, j+r-1$.

The fill generation problem in the fixed-dissection regime seeks a number of area fill features to be inserted into each tile. Two main filling objectives have been addressed in the recent literature:

- the *Min-Var Objective*, where the *variation* in window density (i.e., maximum window density minus minimum window density) is minimized while the window density does not exceed the given upper bound U; and
- the *Min-Fill Objective*, where the number of inserted area fill features is minimized while the density of any window remains in the given range (L, U).

Methods for area fill synthesis in the fixed-dissection context, include:

- Linear Programming (*LP*) methods based on rounding the solution to a relaxation of the corresponding integer linear program formulations [8, 10, 11];
- Greedy and Monte-Carlo (MC) methods which iteratively find a best or random tile for the next filling geometry to be added into the layout [5, 4, 11];
- Iterated Greedy (*IGreedy*) and Iterated Monte-Carlo (*IMC*) methods that improve the solution quality by atternating between phases of area fill insertion and deletion while optimizing the density variation [4].

3 The Compressed Fill Generation Problem

We represent each tile as an array of sites, which are possible positions for inserting area fill features. Some sites are forbidden since they are occupied by existing features. In the discussion below we use the term *AREF* to denote GDSII array reference records that covers only free sites (i.e., sites that are not occupied by any original layout features).

The Compressed Fill Generation Problem (CFGP): *Given a design rule-correct layout consisting of $m \times n$ tiles, and the site arrays for each tile, create the minimum number of AREFs such that the window density variation is within the given bounds (L, U).*

We address the CFGP formulation by first solving the fill generation problem and then compressing the area fill. Let F_{ij} be the number of area fill features for tile (i, j), computed using one of the methods described above.

The Fill Compression Problem: *Given a design rule-correct layout consisting of $m \times n$ tiles T_{ij}, the site arrays for each tile, and fill requirement F_{ij} for each tile, create the minimum number of AREFs such that each tile T_{ij} contains exactly F_{ij} area fill features.*

While solving the Min-Var and Min-Fill fill generation problems, we can obtain a *feasible range* for the number of area fill features in each tile, rather a single requirement F_{ij}. Below we give the corresponding optimization formulation.

The Ranged Fill Compression Problem: *Given a design rule-correct layout consisting of $m \times n$ tiles, and the site arrays for each tile, create the minimum number of AREFs such that each tile T_{ij} contains a number of area fill features in the range (LB_{ij}, UB_{ij}).*

4 Linear Programming Based Methods

Integer linear programming (ILP) approaches for the Fill Compression problem seek to minimize the number of AREFs for the given number of area fill features, while obeying the constraints which prescribe the exact number of area fill features to be inserted into each tile. We use the following definitions:

- $S_{ij,pq} \equiv$ site in position (p, q) in a tile, where the tile itself is in position (i, j) in the overall layout. Every empty site is a possible position for a fill feature.
- $A_\alpha \equiv$ feasible AREF in the layout, where α consists of the following eight parameters: starting site coordinate (ij, pq), width w, height h, horizontal step x, and vertical step y (see Figure 2).

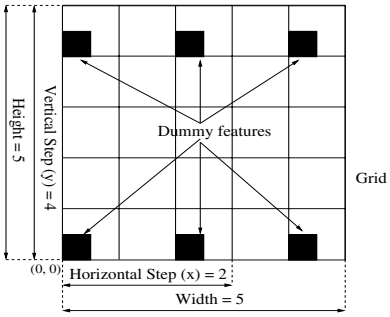

Figure 2: Illustration of AREF $a_{i,j;0,0;5,5;2,4}$. in tile (i, j)

4.1 Single-Tile Integer Linear Program

To insert the exact number of area fill features into each tile, a straightforward method for fill compression is to consider the problem independently in each tile. We call this *single-tile compression*.

For tile t_{ij}, we define the variables

$$s_{pq} \equiv \begin{cases} 1 & S_{ij,pq} \text{ is covered by some AREF;} \\ 0 & \text{otherwise;} \end{cases} \quad (1)$$

$$a_\alpha \equiv \begin{cases} 1 & \text{AREF } A_\alpha \text{ is chosen;} \\ 0 & \text{otherwise;} \end{cases} \quad (2)$$

We then seek the minimum number of AREFs in the slack sites of the tile t_{ij}. The total size of these AREFs must be equal to the given number of area fill features. The corresponding ILP is as follows.

Minimize:

$$\sum_{\text{all feasible AREFs}} a_\alpha \quad (3)$$

Subject to:

$$F_{ij} = \sum_{p=0}^{k-1} \sum_{q=0}^{l-1} s_{pq} \quad (4)$$

$$s_{pq} \geq a_\alpha \qquad \text{if } s_{pq} \text{ is covered by } a_\alpha \quad (5)$$

$$s_{pq} \leq \sum_{\text{all AREFs covering } s_{pq}} a_\alpha \quad (6)$$

$$s_{pq} = 0 \qquad \text{if } S_{ij,pq} \text{ is occupied by original features} \quad (7)$$

- Constraints (4) imply that the total number of covered slack sites is equal to the number of area fill features.
- Constraints (5) imply that once an AREF is chosen, all sites covered by it will be filled.
- Constraints (6) imply that if no AREF which covers site $S_{ij,pq}$ is chosen, the site $S_{ij,pq}$ can not be filled.
- Constraints (7) imply that we cannot fill any site $S_{ij,pq}$ which is already covered by an original layout feature.

Constraints (5) construct an inequality for each site covered by each AREF. However, we can replace these with the following to significantly decrease the number of constraints:

$$n_{pq} * s_{pq} \geq \sum_{\text{all AREFs covering } s_{pq}} a_\alpha \quad (8)$$

where n_{pq} is the number of AREFs which cover site $S_{ij,pq}$ in tile t_{ij}, and $\sum a_\alpha$ is the sum of all AREF variables which cover the site $S_{ij,pq}$. To decrease the ILP problem size, we determine all *valid* AREFs and then construct the ILP formulations based only on these valid AREFs. We call an AREF valid if all sites in it are empty, and its indices have reasonable physical meaning (e.g., $a_{i,j;0,0;2,1;0,0}$ in tile (i, j) is invalid since there is no such AREF with width = 2 and horizontal step 0).

4.2 Multiple-Tile Integer Linear Program

Ideally, we should seek AREFs for the fill features with respect to the entire layout, rather than for each tile independently. However, the large number of tiles and possible AREFs across the entire layout make such a global strategy intractable. Instead, we propose a multiple-tile compression approach, which offers a tradeoff between solution quality and runtime, as follows.

- Partition the layout into groups consisting of $A \times B$ tiles.
- Solve each group separately.

Thus, instead of finding non-overlapping AREFs in one tile, we seek AREFs for the empty spaces of neighboring $A \times B$ tiles. The difference between the Multiple-Tile ILP formulation and the Single-Tile ILP formulation is that the prescribed number of fill features for each tile must be simultaneously achieved. I.e., we replace the constraints (4) with:

$$F_{ij} = \sum_{p'=i*k}^{(i+1)*k-1} \sum_{q'=j*B}^{(j+1)*l-1} s_{p'q'} \quad (9)$$
$$i = 0, \ldots, A-1; j = 1, \ldots, B-1;$$

Here, constraints (9) imply that the total number of covered slack sites in each tile is equal to its given number of fill features.

4.3 Ranged Fill Compression

As noted in [4], excess fill features can be deleted without affecting the density variation to meet the Min-Fill objective. In other words, we can exploit the allowed range of fill features for each tile to relax the LP constraints in order to decrease the LP solver's runtime. The constraints (4) and (9) can thus be respectively rewritten as:

$$LB_{ij} \leq \sum_{p=0}^{k-1} \sum_{q=0}^{l-1} s_{pq} \leq UB_{ij} \quad (10)$$

$$LB_{ij} \leq \sum_{p'=i*k}^{(i+1)*k-1} \sum_{q'=j*B}^{(j+1)*l-1} s_{p'q'} \leq UB_{ij} \quad (11)$$
$$i = 0, \ldots, A-1; j = 1, \ldots, B-1;$$

Here, UB_{ij} is the upper bound for the number of fill features for tile (i, j) which can, e.g., be taken from the normal Monte-Carlo fill result in [3], and LB_{ij} is the lower bound for the number of fill features for tile (i, j) which can, e.g., be taken from the result after the deletion phase in [4].

4.4 Rounded Relaxation Integer Linear Programming

If we round the fractional relaxation of the above ILP's, then a solution may not be feasible since the number of fill features may be unequal to the F_{ij}'s. Therefore, after rounding we use a greedy algorithm (see next section) to add or remove AREFs in order to exactly add F_{ij} fill features into each tile T_{ij}.

5 Greedy Compression Approaches

We propose greedy heuristics to determine a minimum number of AREFs for the required number of area fill features in each tile. A strict greedy algorithm with impractical run time, and followed by two faster practical variants, are described for the multiple-tile ranged fill generation using either overlapping AREFs or non-overlapping AREFs. Single-tile and fixed fill are special cases of multiple-tile and ranged fill, respectively.

A Strict Greedy Method

The *strict greedy heuristic* repeatedly adds an AREF that fills the maximum number of unfilled free sites in multiple tiles, yet does not overfill any tile, and it iterates until the filling requirements in all of the tiles are satisfied. In the ranged filling context, the fill requirements in each tile are changed from a fixed F_{ij} to a certain range (L_{ij}, U_{ij}), where U_{ij} is used to control overfill, and L_{ij} is used to satisfy the minimum fill requirements. The status of all the valid AREFs is checked and updated at each iteration, resulting in a time complexity of $O(n^4)$. Our implementation of this algorithm provides good solutions, but is impractical due to its high time complexity. Letting $L_{ij} = U_{ij} = F_{ij}$ in Figure 3 will solve the fixed fill problem, and using a single tile as a multiple-tile will solve the single tile filling problem.

Strict Greedy Algorithm:

1. Get site set G of $M \times N$ multiple-tile consisting of tiles T_{ij}, $i = 1 \sim M, j = 1 \sim N$;
2. **For** each valid $AREF$ A_α in the multiple-tile **Do**
   Initialize S_α (number of unfilled free sites in A_α), and
   $S_{\alpha,ij}$ (number of unfilled free sites in A_α in each tile (i, j));
3. Pick the $AREF$ A^*_α, where $S^*_\alpha = max\{S_\alpha \mid S_{\alpha,ij} \le U_{ij}\}$;
4. Update the fill requirements:
   $U_{ij} = U_{ij} - S^*_{\alpha,ij}$, $L_{ij} = L_{ij} - S^*_{\alpha,ij}$;
5. Update G, S_α and $S_{\alpha,ij}$ of each $AREF$;
6. **If** $L_{ij} \le 0$ in each single tile (i, j) **Then** exit **Else** go to Step 3;

Figure 3: Strict Greedy Algorithm for multiple-tile ranged fill generation.

Greedy Speedup Approach 1

We propose speedups of the basic greedy approach that offer tradeoffs between compression performance and run time. Our first greedy speedup heuristic finds the largest AREFs originating from each free site, and picks an AREF that fills the maximum number of unfilled free sites but does not overfill the tiles (if such an AREF exists). Otherwise, it selects the maximum AREF from all the largest AREFs, and finds one of its sub-AREFs which does not overfill the tiles. This process is iterated until all the tile filling requirements are satisfied. Note that the solution is generated from all the largest AREFs (or their sub-AREFs) originating from each free site, instead of all the valid AREFs as in the Strict Greedy algorithm. For the Greedy Speedup approach 1, the sets of the largest AREFs originating from the free sites are different for the single tile option and the multiple tile option, and due to runtime considerations our heuristics (Step 3) will not necessarily choose the best sub-AREF. Thus, we cannot guarantee better behavior with the multiple tile option than with the single tile option. For example, for test case $T2$ and $s = 500$ or 250 in Table 2, the results of Greedy Speedup approach 1 with the multiple tile option are worse than those with the single tile option in terms of both $\#AREF$ count as well as run time. The time complexity of the algorithm is thus reduced to $O(n^3)$.

Greedy Speedup Approach 1:

1. Get site set G of $M \times N$ multiple-tile consisting of tiles T_{ij}, $i = 1 \ldots M, j = 1 \ldots N$;
2. **For** the largest $AREF$ A_α originated from each free site **Do**
   Initialize S_α (number of unfilled free sites in A_α), and
   $S_{\alpha,ij}$ (number of unfilled free sites in A_α in each tile (i, j));
3. Pick the $AREF$ that fills the maximum number of unfilled free sites A^*_α, where $S^*_\alpha = max\{S_\alpha \mid S_{\alpha,ij} \le U_{ij}\}$ **If** one exists;
   Else select the $AREF$ A^*_α where $S^*_\alpha = max\{S_\alpha\}$, and pick its sub-$AREF$ A^*_α by doubling x or y until $S^*_{\alpha,ij} \le U_{ij}$;
4. Update the fill requirements:
   $U_{ij} = U_{ij} - S^*_{\alpha,ij}$, $L_{ij} = L_{ij} - S^*_{\alpha,ij}$;
5. Update G, S_α and $S_{\alpha,ij}$ of each largest $AREF$;
6. **If** $L_{ij} \le 0$ in every single tile (i, j) **Then** exit **Else** go to Step 3;

Figure 4: Greedy Speedup Approach 1.

Greedy Speedup Approach 2

An even more efficient approach can be realized by picking *acceptable AREFs* originating from each free site, instead of maximum AREFs. An acceptable AREF is an AREF that fills the maximum number of unfilled free sites but does not overfill the tiles among all the AREFs, originating from the same free site, whose sizes are smaller than $K \times L$. Our second greedy speedup heuristic repeatedly adds an acceptable AREF originating from each free site and iterates until the tile filling requirements are satisfied. The time complexity of the algorithm is thus reduced to $O(KLn^2)$. Moreover, the algorithm is very efficient on actual benchmarks, where $K \cdot L << n$ and using an $AREF$ of size $K \times L$ yields adequate compression.

Greedy Speedup Approach 2:

1. Get site set G of $M \times N$ multiple-tile consisting of tiles T_{ij}, $i = 1 \ldots M, j = 1 \ldots N$;
2. $G' = G$;
3. **For** each free site in G' in the scan order **Do**
4. Calculate S_α (number of unfilled free sites in A_α) of the $AREF$s originating from each free site in G, where $S_{\alpha,ij} \le U_{ij}$, $w \le Kx$, and $h \le Ly$ ($S_{\alpha,ij}$: number of unfilled free sites in A_α in each tile (i, j))
5. Pick the $AREF$ A^*_α, where $S^*_\alpha = \max\{S_\alpha\}$;
6. Update the fill requirements:
   $U_{ij} = U_{ij} - S^*_{\alpha,ij}$, $L_{ij} = L_{ij} - S^*_{\alpha,ij}$;
7. Update G;
8. If $L_{ij} \le 0$ in every single tile (i, j) then Stop;

Figure 5: Greedy Speedup Approach 2.

6 Computational Experience

In the above formulations, the resulting AREFs may overlap with each other. This means that some area fill features will be described multiple times in a GDSII file. The need for a non-overlapping version of this formulation arises from practical concerns and our experimental data. To determine a minimum number of *non-overlapping* AREFs for the slack sites of the tile t_{ij}, the constraints (6) may be modified as follows:

$$s_{pq} \leq \sum_{\text{all AREFs covering } s_{pq}} a_{st,wh,xy} \leq 1 \qquad (12)$$

All of our experiments were performed using part of a metal layer extracted from an industry standard-cell layout (Table 1). Our experimental testbed integrates GDSII Stream input and internally-developed geometric processing engines, coded in C++ under Solaris 2.8. We use CPLEX version 7.0 as the linear programming solver. All runtimes are in CPU seconds on a 300 MHz Sun Ultra-10 with 1 GB of RAM.

Table 2 compares the data volumes from uncompressed fill results against compressed fill results (due to the infeasibility of running the LP solver on large test cases, only compression results from the Greedy Speedup-1 and Greedy Speedup-2 approaches are reported here). As shown in the table, the data volume increase due to insertion of area fill features becomes significant when the fill feature sizes are small (e.g., more than 100 MB for the single layer $T2$ when $s = 250$). The Greedy approach can achieve very large compression ratios for the resulting GDSII files, especially when the fill features are small. For example, in testcase $T3$, the fill data volumes are reduced by about 30 times from 73.8 MB to 2.5 MB when the fill feature size is decreased by 6 times from 1500 to 250. Furthermore, the run times of the Greedy approaches make them feasible in practice. The tradeoff between run time and solution quality is also apparent from Table 2 by comparing the Single-tile results with the Multiple-tile results. We can achieve a smaller number of AREFs by using the Multiple-tile approach, although it requires longer run times. As expected, the Greedy Speedup-2 approach is much faster than the Greedy Speedup-1 approach, with only a small degradation in solution quality.

Table 3 compares several fill compression methods proposed in this paper: the ILP method, the Greedy Speedup-1 method, and the Greedy Speedup-2 method. Since the run times of ILP methods make them infeasible for multiple-tile fill compression, we only report results for single-tile fill compression. Also, since the LP+Greedy method affects the fill quality, we do not detail its results here. We observe improvements in terms of the number of AREFs as well as run times for the ranged fill-compression approaches. The experiments also indicate that the results of the Greedy method are very close to those of the optimal ILP method, yet offer significant improvements in the run times (i.e., a decrease from several hours to less than one second).

7 Conclusions and Future Research

We introduced new compressed fill strategies which exploit the GDSII array reference record (AREF) construct in order to significantly reduce GDSII data volumes in filled layouts. We applied greedy and linear programming based optimization techniques, and obtained practical compressed filling solutions on industry layouts. Future work entails improving the compression ratios and scalability (capacity and runtime) of our algorithms. We also seek to exploit SREF constructs in GDSII (as well as potential extensions to the Stream format that is currently being considered by industry consortia), and the new Standard Layout Format (SLF) recently proposed for replacing the GDSII format. Finally, the problem of compressible fill generation when there is an underlying layout hierarchy (yet with context-dependent filling), or for specific mask writing and inspection tool sets, is of considerable interest.

Testcase	T1	T2	T3	T4	T5	T6
layout size	819,200	819,200	819,200	125,000	112,000	112,000
# rectangles	32,258	142,585	78,293	49,506	76,423	133,201

Table 1: Parameters of three industry test cases.

References

[1] D. Boning, B. Lee, T. Tugbawa, and T. Park, "Models for Pattern Dependencies: Capturing Effects in Oxide, STI, and Copper CMP", *Semicon/West Tech. Symp.: CMP Tech. for ULSI Manuf.*, July 2001.

[2] P. Buck (Dupont Photomasks), personal communication, *International Sematech Mask-EDA Workshop*, July 2001.

[3] Y. Chen, A. B. Kahng, G. Robins and A. Zelikovsky, "New Monte-Carlo Algorithms for Layout Density Control", *Proc. ASP-DAC*, 2000, pp. 523-528.

[4] Y. Chen, A. B. Kahng, G. Robins and A. Zelikovsky, "Practical Iterated Fill Synthesis for CMP Uniformity", *Proc. ACM/IEEE Design Automation Conf.*, Los Angeles, June 2000, pp. 671-674.

[5] Y. Chen, A. B. Kahng, G. Robins and A. Zelikovsky, "Hierarchical Dummy Fill for Process Uniformity", *Proc. ASP-DAC*, Jan. 2001, pp.139-144.

[6] V. Dai and A. Zakhor, "Lossless Layout Compression for Maskless Lithography Systems", *Proc. Emerging Lithographic Technologies IV*, Santa Clara, February 2000, SPIE Volume 3997, pp. 467-477.

[7] R. Ellis, A. B. Kahng, Y. Zheng, "JBIG Compression Algorithms for Dummy Fill VLSI Layout Data", *technical report* #CS2002-0709, UCSD CSE Department, June 2002.

[8] A. B. Kahng, G. Robins, A. Singh, H. Wang and A. Zelikovsky, "Filling Algorithms and Analyses for Layout Density Control", *IEEE Trans. Computer-Aided Design* 18(4) (1999), pp. 445-462.

[9] F.M. Schellenberg, L. Capodieci and B. Socha, "Adoption of OPC and the Impact on Design and Layout", *Proc. ACM/IEEE Design Automation Conf.*, Las Vegas, June 2001, pp. 89-92.

[10] R. Tian, D. Wong, and R. Boone, "Model-Based Dummy Feature Placement for Oxide Chemical Mechanical Polishing Manufacturability", *Proc. ACM/IEEE 2Design Automation Conf.*, June 2000, pp. 667-670.

[11] R. Tian, X. Tang and D. F. Wong, "Dummy feature placement for chemical-mechanical polishing uniformity in a shallow trench isolation process ", *Proc. ACM/IEEE International Symposium on Physical Design*, April 2001, pp. 118-123.

[12] S. Ueki, I. Ashida, and H. Kawahira, "Effective data compaction algorithm for vector scan EB writing system", *20th Annual BACUS Symposium on Photomask Technology*, Monterey, CA, Sept. 2000, pp. 589-600.

Testcase		Uncomp	Ranged Single-Tile				Ranged Multiple-Tile			
			GS-2		GS-1		GS-2		GS-1	
T/W/r/	s	Data(K)	Data	CPU	Data	CPU	Data	CPU	Data	CPU
T1/80K/4	1500	1538	1036 (1.48 ×)	0.23	1016 (1.51 ×)	3.07	1004 (1.53 ×)	0.26	1001 (1.54 ×)	8.87
T1/80K/4	1000	2303	1043 (2.20 ×)	0.60	1013 (2.27 ×)	17.10	1010 (2.28 ×)	0.73	993 (2.32 ×)	57.14
T1/80K/4	500	6156	1062 (5.80 ×)	3.27	1016 (6.06 ×)	357.36	1035 (5.95 ×)	4.47	1008 (6.10 ×)	1329.84
T1/80K/4	250	22256	1099 (20.25 ×)	20.75	1016 (21.90 ×)	11976.27	1061 (20.98 ×)	31.64	1027 (21.67 ×)	29081.66
T2/80K/4	1500	1672	1242 (1.35 ×)	0.20	1188 (1.41 ×)	1.56	1135 (1.47 ×)	0.22	1161 (1.44 ×)	4.63
T2/80K/4	1000	4846	2105 (2.30 ×)	0.75	1777 (2.73 ×)	8.52	1949 (2.49 ×)	1.06	1776 (2.73 ×)	30.64
T2/80K/4	500	27405	5015 (5.46 ×)	7.59	3671 (7.49 ×)	178.50	4731 (5.79 ×)	14.67	3925 (6.98 ×)	693.64
T2/80K/4	250	106664	7527 (14.17 ×)	64.17	3071 (34.73 ×)	5525.80	7460 (14.30 ×)	133.02	4251 (25.09 ×)	14166.58
T3/80K/4	1500	1448	993 (1.46 ×)	0.16	978 (1.48 ×)	1.07	960 (1.51 ×)	0.10	962 (1.51 ×)	2.99
T3/80K/4	1000	2867	1142 (2.51 ×)	0.48	1073 (2.67 ×)	7.06	1031 (2.78 ×)	0.50	1068 (2.68 ×)	21.83
T3/80K/4	500	18842	2719 (6.93 ×)	5.44	2583 (7.29 ×)	156.53	1946 (9.68 ×)	7.11	2957 (6.37 ×)	523.87
T3/80K/4	250	73825	3477 (21.23 ×)	39.68	1968 (37.51 ×)	4917.56	2512 (29.39 ×)	50.04	2888 (25.56 ×)	11066.06

Table 2: Data compression. **Notation:** *Ranged Single-Tile:* ranged single-tile fill compression approaches; *Ranged Multiple-Tile:* ranged multiple-tile fill compression approaches; *Uncomp:* fill solution without compression; *GS-1* Greedy Speedup-1 approach; *GS-2* Greedy Speedup-2 approach; *T/W/r:* Layout / window size / r-dissection; *s:* site size; *Data:* file size increase in kilobytes due to fill features (reduction factor relative to *Uncomp*); *CPU:* runtime (in CPU seconds).

Testcase		Fixed Fill-Compression						Ranged Fill-Compression					
		ILP		GS-1		GS-2		ILP		GS-1		GS-2	
T/W/r	s	#AREF	CPU	#AREF	CPU	#AREF	CPU	#AREF	CPU	#AREF	CPU	#AREF	CPU
T1/80K/4	1500	1187	11918	1307	2.04	1548	0.22	1151	7931	1159	2.01	1323	0.21
T2/80K/4	1500	1210	6932	1269	0.99	1429	0.10	1167	1154	1193	0.95	1371	0.09
T3/80K/4	1500	464	4502	456	0.62	552	0.09	381	759	384	0.58	483	0.10
T4/12K/4	200	850	30922	891	1.63	1317	0.15	787	4215	796	1.58	1272	0.16
T5/12K/4	200	3787	23002	4043	4.61	6712	0.50	3694	18126	3927	4.49	6403	0.47
T6/12K/4	200	1249	13813	1320	1.60	1727	0.15	1230	11466	1304	1.58	1666	0.17

Table 3: Performance of the fill compression methods. **Notation:** *Fixed Fill-Compression:* fixed fill compression approaches (where the number of fill features for each tile is fixed); *Ranged Fill-Compression:* ranged fill compression approaches (where the number of fill features for each tile is ranged); *ILP:* ILP based approach; *GS-1:* Greedy Speedup-1 Approach; *GS-2:* Greedy Speedup-2 Approach. *#AREF:* number of AREFs in the fill solution; *CPU:* runtime (in seconds).

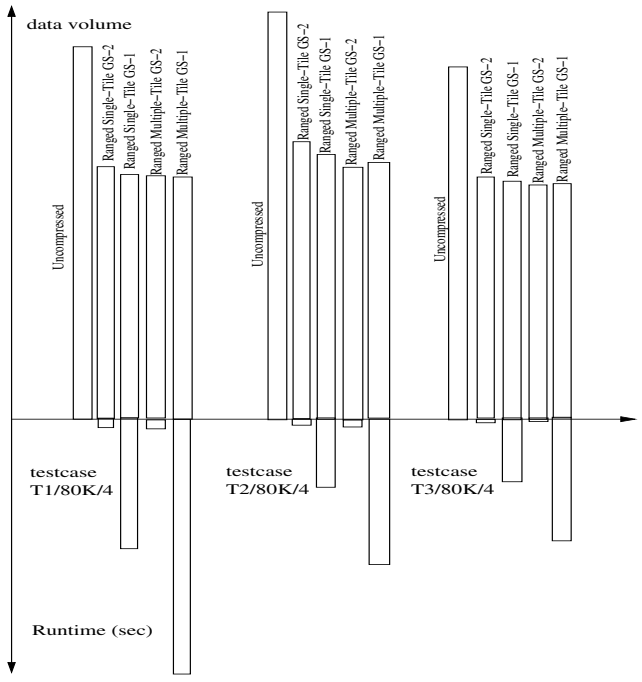

Figure 6: Data compression results (for more details, see Table 2).

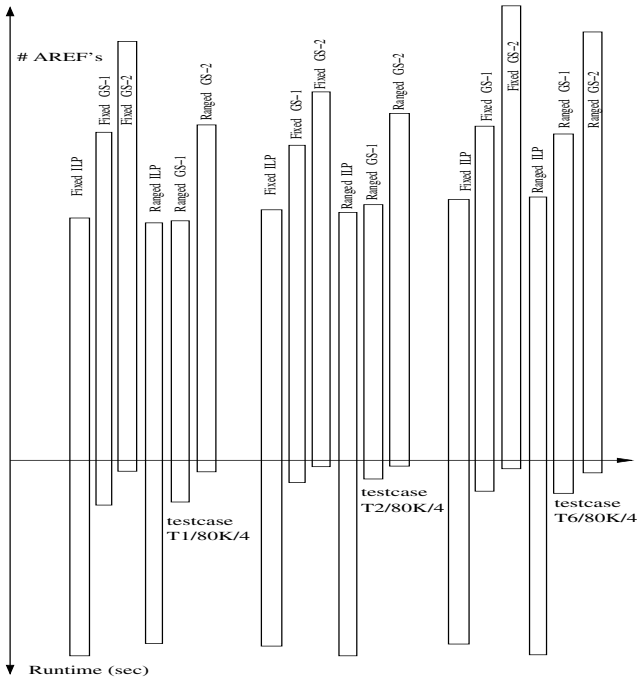

Figure 7: Performance results (for more details, see Table 3).