Hierarchical Dummy Fill for Process Uniformity *

Yu Chen, Andrew B. Kahng†, Gabriel Robins‡ and Alexander Zelikovsky¶

Computer Science Department, UCLA, Los Angeles, CA 90095-1596
†UCSD CSE and ECE Departments, 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
yuchen@cs.ucla.edu, abk@cs.ucsd.edu, robins@cs.virginia.edu, alexz@cs.gsu.edu

Abstract— To improve manufacturability and performance predictability, we seek to make a layout uniform with respect to prescribed density criteria, by inserting "fill" geometries into the layout. Previous approaches for flat layout density control are not scalable due to the necessity of solving very large linear programs, the large data volume of the solution, and the impact of hierarchy-breaking on verification. In this paper, we give the first methods for *hierarchical* layout density control for process uniformity. Our approach trades off naturally between runtime, solution quality, and output data volume. We also allow generation of compressed GDSII of fill geometries. Our experiments show that this hybrid hierarchical filling approach saves data volume and is scalable, while yielding solution quality that is competitive with existing Monte-Carlo and linear programming based approaches.

I. Introduction

To improve manufacturability and performance predictability, modern design methodologies must make layouts uniform with respect to feature density criteria, by inserting "dummy fill" geometries into layouts. According to [1], the so-called Filling Problem may be defined as follows:

The Filling Problem: Given a design rule-correct layout in an $n \times n$ *layout region*, along with a *window size* $w < n$, and upper (U) and lower (L) bounds on the feature density in any window, add *dummy fill* geometries to create a *filled layout* such that either:

- *(Min-Var Objective)* 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; or
- *(Min-Fill Objective)* the number of inserted fill geometries is minimized while the density of any window remains in the given range (L, U).[1]

Literature on dummy fill has focused on chemical-mechanical polishing (CMP) of spin-on glass (SOG) interlayer dielectrics (ILD) [6] [8] [13]. Post-polish ILD thickness variation is kept within acceptable limits by controlling local feature density, relative to a process-specific "window size" (on the order of 1-3mm), that depends on CMP pad material, slurry composition, and other factors [3]. We observe that the 1999 International Technology Roadmap for Semiconductors [9] added copper interconnect dishing to the fundamental roadmap parameters for interconnect. (The 2000 ITRS will add copper interconnect thinning in CMP to the fundamental parameters.) So, density-mediated process variation has become a first-order concern for interconnects.

Applications of dummy fill on device layers (diffusion, poly, thin-ox) are equally (or more) critical. Isolated transistors are susceptible to contact overetch in reactive ion etch (RIE) process steps, which results in leakage. Chemical vapor deposition steps are also subject to iso-dense variations. CVD and etch process variation are particularly troubling with respect to today's lightly-doped drain (LDD) device properties. The complex effects of these process variations are well-known, e.g., Garofalo et al. [4] document 10% error in interline capacitance resulting from a 5% variation in linewidth, and 12% error in ring oscillator frequency solely from proximity effects. At the same time, it is also well-known that the uniformity of feature density obtained via dummy fill can mitigate macroscopic process proximity effects such as contact etch variation in reactive ion etch, and nonuniformity of chemical vapor deposition.

Dummy fill creates a number of critical flow issues, including:

* 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 a GSU Research Initiation Grant.

[1] The Min-Var objective was introduced in [5], and captures the "manufacturing side" of the Filling Problem by seeking the most uniform density distribution possible. The Min-Fill objective was introduced in [12], and captures the "design side" by seeking to minimize total coupling capacitance and uncertainty caused by dummy fill. Minimizing dummy fill has the side benefit of reducing the complexity of the output GDSII.

0-7803-6633-6/01/$10.00 ©2001 IEEE.

- physical design and verification must understand the dummy fill in order to estimate RC parasitics, gate/interconnect delays, and even device reliability;
- master cell and macro characterizations (performance models) must be a priori compatible with later insertion of dummy fill; and
- dummy fill must be consistent with design hierarchy so as not to break verification or data capacity.

The first issue applies to dummy fill on interconnect layers, which have non-hierarchical layouts (with exception of memories, and logic M1 with certain combinations of cell library and router styles). The second issue can be avoided by judicious "buffer distance" rules for dummy fill insertion (i.e., dummy fill is restricted to locations where it does not change electrical performance). In this paper, we focus on the third issue: dummy fill generation that is consistent with *hierarchy-related* requirements.

Hierarchy arises in both custom and semi-custom design flows. In custom design, hierarchy is mostly for management of the decomposition of the design problem. In semi-custom design, hierarchy is associated more with reuse of standard cells, whose layouts include device layers and local interconnect, or IP blocks. The key observation is that hierarchical designs become difficult to verify when flattened. Hence, hierarchical dummy filling can enable simpler and faster verification of the filled layout, since verification can still follow the original hierarchy. Hierarchical filling can also decrease data volume for standard-cell designs. (In general, data volume is a big issue for dummy fill since a filling solution can consist of many millions of tiny geometries.) Thus, hierarchical fill generation is an emerging requirement for future commercial EDA tools [10].

Our present work investigates approaches and tradeoffs inherent in filling master cells rather than individual instances. We consider hierarchical filling as a post-processing step performed (on device layers) after placement. When router access to local interconnect (salicide) and M1 layers is strongly restricted[2] then hierarchical filling may be performed after routing as well. Hierarchical filling faces obvious difficulties:

- when dummy fill is inserted into a master cell, it must satisfy density constraints in all contexts for instantiations of the master;
- there are many interactions or interferences at master cell boundaries and at distinct levels of the hierarchy;
- solution quality in terms of either the Min-Var or Min-Fill objective will be worse for hierarchical solutions than flat solutions, simply because the former are more constrained; and
- hierarchical filling explodes the number of constraints in linear programming formulations, and thus cannot use the LP techniques which have been successful for flat filling [5] [12].

[2]E.g., Cadence and Avant! gridded routers are often restricted to well-defined pin availabilities at points of the routing grid.

The main contribution of this paper is a new proposed hierarchical filling algorithm which mitigates these drawbacks. Our approach is based on hybridizing hierarchical filling techniques with a flat filling postprocessor, in a way that smoothly trades off (in a user-controlled manner) the efficiency of the former with the accuracy of the latter.

The remainder of this paper is organized as follows. Section II reviews the various models for density calculation for CMP and previous approaches for solving the flat Filling Problem, including linear programming formulations and the Monte-Carlo approach. We give the formulation of the Hierarchical Filling Problem and our proposed solution to it in Section III. Finally, computational results of our proposed hierarchical fill approach, as compared with results for flattened hierarchical designs, are reported in Section IV.

II. Previous Work on Dummy Fill Synthesis

A computationally efficient model for CMP of oxide planarization proposed in [8] is based on the determination of the effective initial pattern density, and is easy to calibrate [11]. An approach that unifies the two pattern density definitions studied in [5] and [12], enables the application of the same layout density control methods to both scenarios [1] (the pattern density is a local property and therefore depends at each point on the neighboring spatial pattern density).

A standard practice in discretizing the filling problem is to consider only windows (i.e., floating rectangle region of given size) from a *fixed dissection.* However, bounding the effective density in $w \times w$ windows of a fixed dissection can incur error, since other windows that are not part of the dissection could still violate the effective density bounds. Therefore, a common industry practice is to enforce density bounds in r^2 overlapping dissections, where r determines the "phase shift" w/r by which the dissections are offset from each other. Thus, density bounds are enforced only for windows of the *fixed r-dissection* (see Figure 1), in the hope that this would also control the density bounds of arbitrary windows.[3]

The work of [3] considers the deformation of the polishing pad during the CMP process, while [12] uses an elliptical weighting function with experimentally determined constants. A discretized effective local pattern density ρ for a window W_{ij} in the fixed-dissection regime (henceforth referred to as *effective window density*) is defined in [12] as:

[3]The $n \times n$-layout is partitioned into tiles T_{ij}, then covered 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}, $k = i, \ldots, k + r - 1$, $l = j, \ldots, j + r - 1$ (see Figure 1). Windows are "wrapped around" the layout, e.g., windows overlapping the upper (left) edge of the layout also containing tiles at the bottom (right) of the layout, reflecting the fact that density at the edge of one die may affect CMP of the die's neighbor on the wafer.

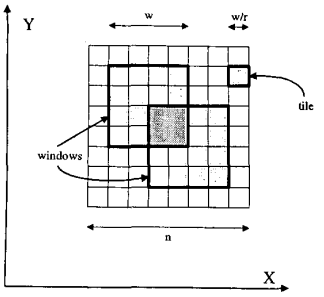

Figure 1: The fixed dissection approach: a layout is divided into r^2 ($r = 4$) distinct dissections (each with window size $w \times w$), into $\frac{nr}{w} \times \frac{nr}{w}$ *tiles*. Each $w \times w$ window (dark) consists of r^2 tiles, and pairs of windows from different dissections may overlap.

$$\rho(W_{ij}) = \sum_{k=i}^{i+r-1} \sum_{l=j}^{j+r-1} area(T_{kl}) \cdot f(k-(i+r/2), l-(j+r/2)) \quad (1)$$

where the arguments of the elliptical weighting function f are the x- and y-distances of the tile T_{kl} from the center of the window W_{ij}. More recently, the authors of [12] suggested a more accurate model that takes into account the influence of density variation in lower layers of a layout on the density variation in the upper layers.

Previous algorithms for generating flat dummy fill can be classified into two categories: **linear-programming** (LP) based approaches [5] [12], and **Monte-Carlo** based methods [1] [2].

The first linear programming formulation for the Min-Var objective was suggested in [5], where for each tile the computed fill amounts are constrained to not exceed the actual area available for filling (*slack*), which is computed during density analysis. The Min-Fill objective for the Filling Problem corresponding to the *Ranged Variation* LP formulation was proposed in [12]. Although an LP solution is optimal, it has several drawbacks: (1) solving a large LP is too time consuming (e.g., the runtime is $O(r^6)$ since the number of variables and the number of constraints in the LP are both $O((\frac{nr}{w})^2)$); (2) the optimal solution for a given number of dissections is not necessarily the optimal solution for other dissections, and in general may result in a high *floating* window density variation; and (3) when the tile size is sufficiently small the problem becomes an instance of integer programming and rounding errors become crucial.

A Monte-Carlo method for the Min-Var objective was introduced in [2]. The Min-Var Monte-Carlo algorithm chooses a tile and increments its density by a prescribed fill amount, and this is repeated until the density of all tiles exceeds the lower bound threshold. This process is efficient, but has the drawback that it may insert an excessive amount of fill. This problem can be mitigated by a *Min-Fill* Monte-Carlo approach, which attempts to maintain the lower bound L on window density by iteratively deleting filling geometry from tiles [2]. The *iterated* Monte-Carlo method alternates the Min-Var and Min-Fill objectives, which tends to monotonically narrow the gap between the minimum and maximum window densities. Such an iterated approach is reasonably fast as well as accurate, thus retaining the advantages of its non-iterated counterparts. However, this method is still beset by the large data volume problem associated with flat fill approaches.

While LP based algorithms are highly accurate, they tend to be slow due to the large number of variables. For flat layouts, Monte-Carlo methods are faster than LP approaches, although typically less accurate [1] [2].

III. The Hierarchical Filling Problem

The filling problem for hierarchical (standard-cell) layouts is similar to its counterpart for flat layouts, except that the hierarchical structure of master cells must be preserved, i.e., the same filling geometry is simultaneously added to *all* instances of the same master cell. Here, we assume that we can fill the slack area of each master cell independently and uniformly, as is the case when the size of fill geometries is sufficiently small.

The Hierarchical Filling Problem: Solve the Filling Problem for a given standard-cell layout so that:

- Filling geometries are added only to master cells;
- Each cell of the filled layout is a filled version of the corresponding original master cell; and
- The increase in (hierarchical) layout data volume does not exceed a given threshold.

The above constraints make the LP approach for hierarchical filling problem infeasible. Instead of using $O((\frac{nr}{w})^2)$ variables and constraints corresponding to each tile and window in the LP formulation for flat fill, we must define the variables and constraints for each window, each master cell instance, and all feasible fill positions for each master cell and window combination. This will greatly increase the number of variables and constraints (since the number of grids is much larger than the number of tiles). The LP formulation will furthermore be complicated by the transformations of master cell instances and the overlaps between the instances. Based on these considerations, Monte-Carlo method becomes the only feasible approach for the hierarchical filling problem.

Our proposed hierarchical filling algorithm (see Figure 2) starts by computing the slack for all master cells (cell overlaps are possible and must be addressed carefully, as detailed below). We then create buffer zones around master cells to avoid overfilling the regions near master cell

boundaries. Master cells are then filled in a Monte-Carlo fashion, according to a priority scheme where master cells that are more severely underfilled receive higher priority for filling at each iteration. This process continues until all master cells are filled past the lower bound density threshold or the slack in all underfilled master cells is exhausted.

Monte-Carlo Hierarchical Filling Algorithm

Input: Hierarchical layout, fixed r-dissection, buffer distance, $w \times w$ window, upper bound U on window density

Output: Hierarchical layout with filled master cells

1. **For** each Master Cell M_i in the layout **Do**
2. Partition the Master Cell M_i according to the given grid size
3. **For** all grids in the Master Cell **Do**
4. Mark the status of grid "occupied" if it is covered by the original features or the sub Master Cell
5. **For** all instances I_j of the Master Cell M_i **Do**
6. **If** the instance I_j is overlapped with features or instances of other Master Cells **Then**
7. Update the status of grids which are covered
8. Calculate the priority of the Master Cells
9. **While** the sum of priority > 0 **Do**
10. Use the Monte-Carlo method to select one Master Cell M_i
11. Randomly select a slack grid position in the master cell
12. **For** each corresponding position of the grid in all instances of the Master Cell M_i **Do**
13. **If** the insertion causes any window density to exceed the upper bound U **Then**
14. Discard the insertion
15. Lock slack grid position
16. Go over all other grids in master cell covered by the exceeded window and lock them
17. **Else**
18. Increase the fill area of the Master Cell
19. Add the fill geometry into the Master Cell
20. Update the relevant windows' densities

Figure 2: Monte-Carlo Hierarchical Filling Algorithm.

A. Slack Computation for Hierarchical Layouts

For each master cell, dummy fill may be inserted only into the slack (i.e., free) area of a master cell, not into its subcells. Computing the slack of a master cell proceeds by first determining the number of grid positions inside the bounding box of the master cell, while excluding all positions that overlap with either a "bloated" feature (i.e., a forbidden buffer zone around each feature) or a "bloated" subcell. However, slack area computation is complicated by the fact that instances of master cells may overlap. Such overlaps can occur between the master cell instance and the features, or between two or more master cell instances (see Figure 3). In general, overlaps may have a very complicated structure. We distinguish the following cases:

(1) The overlap between a master cell instance and a feature.

(2) The overlap between two instances of different master cells.

(3) The overlap between more than two instances of different master cells.

(4) The overlap between two or more than two instances of the same master cell.

For each region of master cell overlap we must determine which master cell "owns" that intersection region. In other words, it is necessary to assign the space available for filling to the slack of a *single* master cell. We resolve the "ownership" problem by fixing an order of all master cells starting from the global master cell (containing entire layout) down to individual features. The hierarchy can be represented as the acyclic directed graph H with the set of nodes consisting of all master cells and individual features. The graph H has an arc from the master cell A to the master cell or the feature B if B participates in the definition of A. The topological order of the graph H is an order of its nodes in which the beginning of any arc is later than its end in respect to this order. The topological order of the graph H is obtained by breadth-first-search traversing of H starting from the global master cell. The *containment-based topological ordering of the hierarchy* corresponds to the topological order of the graph H. Then no master cell later in the order may use in its definition master cells appeared earlier in the order. Every time when we have an intersection of master cell instances, we check which of the master cells appears later in the topological order and assign the intersection area to this master cell. This way we correctly resolve the overlap cases (1-3). Unfortunately, the case (4) cannot be resolved in this manner because hierarchy cannot distinguish different instances of the same master cell. Thus, we exclude the overlapping regions from the slack of the master cell thus leaving the them unavailable for fill.

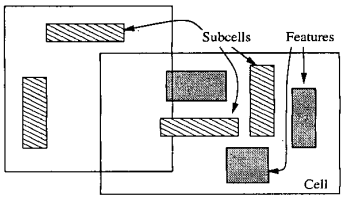

Figure 3: Computing master cell intersections: the dark features and patterned subcells may either completely or partially overlap with a given master cell.

B. A Hybrid Hierarchical / Flat Filling Approach

Pure hierarchical filling may tend to result in some sparse or unfilled regions (e.g., due to overlaps between different instances of master cells and features or due to the interactions among the "bloat" regions around master

cells), which could yield an unacceptably high layout density variation. A natural and simple solution is to apply a post-processing "cleanup" phase, i.e., apply a standard flat fill algorithm to the output of the hierarchical phase. However, a purely flat fill approach, even when applied as a secondary post-processing phase, may greatly increase the resulting data volume and runtime, negating the benefits of using a hierarchical approach in the first place.

We propose a new algorithm for mitigating this drawback, by combining hierarchical filling techniques with a flat filling approach, in a way that smoothly trades off the respective efficiency and accuracy of these two approaches. In our proposed method, varying a user-controlled parameter yields a smooth tradeoff among solution quality, data volume, and runtime, as confirmed by our computational experience. Our three-phase hybrid hierarchical-flat filling approach is summarized as follows:

1. A purely hierarchical fill phase; followed by
2. A split-hierarchical phase, where certain master cells that were deemed to be underfilled in phase 1, would be replicated so that distinct copies of the same master cell may be filled differently than other copies of the same master cell; and finally,
3. A flat fill "cleanup" phase (say, Monte-Carlo based), which will fill any remaining sparse or unfilled regions that were not processed satisfactorily during the first two phases.

The overall goal with this strategy is to quickly fill as much of the layout as possible in phases 1 and 2 while keeping the fill output data volume relatively low, and then further improve and tune the resulting filled layout using a flat filling approach in phase 3 on the (presumably small number of) remaining sparse or unfilled areas.

In particular, phase 2 consists of repeatedly *splitting* master cells located in regions which were determined to be underfilled during phase 1, as follows. Given a top-down containment-based topological ordering of the n master cells, i.e. $C_1, C_2, C_3, \ldots, C_{n-2}, C_{n-1}, C_n$, where a master cell C_i can only contain master cell C_j iff $i < j$, a master cell C_i may be split into two master cells $C_{i,1}$ and $C_{i,2}$ and any C_j containing master cell C_i is then modified to point to either the copy $C_{i,1}$ or $C_{i,2}$ (say, randomly chosen). More generally, rather than performing only two-way splits, we can perform k-way splits (see Figure 4).

Varying the parameter k (which controls the split factor) from 1 (pure hierarchical) to infinity (pure flat), yields a smooth tradeoff between solution quality, data volume, and runtime. As k is increased, the solution quality asymptotically approaches that of flat fill. If the result of hierarchical filling does not satisfy the technological constraints, then we recommend foregoing the original hierarchy in favor of a more uniform filling. This can be implemented by storing in the original cell library different *filled versions* of each master. Such a scheme will not necessarily slow down verification, since having fixed permanent structure, they can be "pre-verified", and thus dramatically improve the uniformity of hierarchical filling without a large runtime increase.

k-Way Master Cell Splitting Algorithm
Input: Hierarchical layout, and a splitting parameter k
Output: New hierarchical layout with new copies of master cells
For $i = 1$ to n **Do**
  Create k new copies of C_i, namely $C_{i,1}, C_{i,2}, \ldots C_{i,k}$
  For any master cell C' containing in the master cell C_i **Do**
    For all $1 \leq j \leq k$ **Do**
    put an arc from the master cell $C_{i,j}$ to C'
For any master cell C which contains master cell C_i **Do**
  Replace C_i inside C with copy $C_{i,j}$ for random j, $1 \leq j \leq k$
  In hierarchy H, replace the arc (C, C_i) with $(C, C_{i,j})$
Output resulting new hierarchical layout

Figure 4: Improving the hierarchical filling approach by splitting master cells k-ways: each master cell is replaced with k distinct masters, each of which may be filled independently and differently.

Following the approach of [1], our implementation has the following capabilities: grid slack computation; doughnut area computation; wraparound window density analysis and synthesis; and compressed fill insertion.

IV. Computational Experience

Our experimental testbed integrates GDSII Stream input and internally-developed geometric processing engines, coded in C++ under Solaris. Our experiments were performed using part of a metal layer extracted from hierarchical GDSII from an industry custom-block layout. Table 1 lists the attributes of our three test cases, i.e., layout size and number of rectangles.[4]

Test Cases			
Test case	Case 1	Case 2	Case 3
layout size	260000	288000	504000
# rectangles	216	432	540

Table 1: Parameters of test cases.

Table 2 compares the minimum window density, data volume (i.e., the number of fill geometry references in the resulting GDSII output file), and the number of dummy fill features (i.e., the number of fill geometries on the resulting layout after flattening) for five heuristics: (i) hierarchical, (ii) flat, (iii) 2-way splitting, (iv) hybrid of hierarchical and flat, and (v) hybrid of the hierarchical, splitting and flat approaches. For each test case, we ran

[4]In the given coordinate system, 40 units is equivalent to 1 micron.

all the five filling heuristics on both the spatial density model and the effective density model, with the window density upper bound equal to the original maximum window density.

DenModel	Spatial Den			Effective Den		
	data	#fill	MinDen	data	#fill	MinDen
	Testcase 1					
OrgLayout			0.070			0.291
Hier	645	5136	0.11	1054	2608	0.369
H+F	1562	6053	0.335	2758	4312	0.655
H+S	2321	7601	0.17	1552	4166	0.525
H+S+F	2834	8114	0.339	2908	5522	0.676
Flat	5219	5219	0.403	5732	5732	0.735
	Testcase 2					
OrgLayout			0.167			0.145
Hier	2081	16060	0.272	2142	16972	0.248
H+F	2451	16430	0.393	5630	17460	0.320
H+S	4368	17494	0.410	4531	18126	0.365
H+S+F	4374	17500	0.421	7234	20829	0.383
Flat	13974	13974	0.527	23415	23415	0.443
	Testcase 3					
OrgLayout			0.000			0.091
Hier	4995	22566	0.071	4449	20320	0.157
H+F	7472	25043	0.532	9461	25332	0.371
H+S	9690	23622	0.102	8575	22990	0.159
H+S+F	12212	26144	0.540	13285	25700	0.394
Flat	17695	17695	0.547	31204	31204	0.483

Table 2: The Hierarchical, Flat and Hybrid Filling Approaches. **Notation:** *OrgLayout*: original layout; *Spatial Den*: spatial density model; *Effective Den*: effective density model; *data*: data volume, i.e., the number of fill geometry references in resulting GDSII output file; *# fill*: number of real dummy fill features on the resulting layout; *MinDen*: minimum window density of the layout; *Hier*: hierarchical filling approach; *H+F*: hierarchical + flat filling approach; *H+S*: hierarchical + 2-way master cell splitting filling approach; *H+S+F*: hierarchical + 2-way master cell splitting + flat filling approach; *Flat*: flat filling approach.

Table 2 indicates that the Flat Monte-Carlo approach obtains the best-quality result (i.e., highest minimum density) but results in the largest data volume. On the other hand, the Hierarchical Monte-Carlo approach saves on data volume but yields low-quality results. The hybrids of hierarchical and flat fill approaches produce substantially improved results, with only a modest increase in data volume. Finally, we observe that the k-way Master Cell Splitting approach smoothly trades off between performance and data volume, i.e., it provides better results than the pure Hierarchical Fill approach and less data volume than the pure Flat Filling approach.

V. Conclusions and Future Directions

In conclusion, we have addressed the hierarchical filling problem in layout density control for CMP uniformity. We presented a practical approach to *hierarchical* fill synthesis, which trades off runtime, solution quality, and output data volume. Distinct copies of a master cell are allowed to be filled differently, which improves the solution quality in a user-controlled manner. Our system also generates filling geometries in compressed GDSII format, which reduces the resulting fill data volume. Experiments indicate that this new hybrid hierarchical filling approach is scalable, efficient, and highly competitive with previous Monte-Carlo and LP-based methods.

Ongoing research includes developing alternate pure-hierarchical filling heuristics, and developing more robust hierarchy manipulators for in-memory layout representations, in order to enable smoother tradeoffs between solution quality and data volume. We also seek to make our fill solutions reusable, so that fill solutions can be stored in a library along with the master cells, and would not have to be recomputed from scratch in cases where a cell is used in a context that has different density constraints. However, the reusability methodology can be only applied to master cells which are neither overlapped with other master cells, nor routed over. One way of achieving such "unrollable" solutions is to produce and store a fill solution in a "monotone" manner, so that successively longer prefixes of a fill solution would still constitute valid fill solutions in lower density contexts.

References

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

[2] 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.

[3] R. R. Divecha, B. E. Stine, D. O. Ouma, J. U. Yoon, D. S. Boning, et al., "Effect of Fine-line Density and Pitch on Interconnect ILD Thickness Variation in Oxide CMP Process", *Proc. CMP-MIC*, 1998.

[4] J. G. Garofalo, J. Q. Zhao, J. Blatchford and E. Nease, "Applications of enhanced optical proximity correction models", *Proc. SPIE Optical Microlithography XI*, SPIE Vol. 3334, Feb. 1998.

[5] 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.

[6] H. Landis, P. Burke, W. Cote, W. Hill, C. Hoffman, et al., "Integration of Chemical-Mechanical Polishing into CMOS Integrated Circuit Manufacturing", *Thin Solid Films* 220(20) (1992), pp. 1-7.

[7] W. Maly, "Moore's Law and Physical Design of ICs", (special address), *Proc. ISPD*, 1998.

[8] G. Nanz and L. E. Camilletti, "Modeling of Chemical Mechanical Polishing: A Review", *IEEE Trans. on Semiconductor Manufacturing* 8(4) (1995), pp. 382-389.

[9] *International Technology Roadmap for Semiconductors*, Dec 1999, `www.itrs.net/1999_SIA_Roadmap/Home.htm`

[10] J. Rey, *personal communication*, 2000.

[11] B. Stine, "A Closed-Form Analytical Model for ILD Thickness Variation in CMP Processes", *Proc. CMP-MIC*, 1997.

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

[13] M. Tomozawa, "Oxide CMP Mechanisms", *Solid State Technology* 40(7) (1997), pp. 169-175.