Evaluation of the New OASIS Format for Layout Fill Compression*

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

UbiTech, Inc. San Jose, CA 95110

† 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

¶ UCSD CSE Departments, La Jolla, CA 92093-0114

yuchen@ubitechnology.com, abk@cs.ucsd.edu, robins@cs.virginia.edu, alexz@cs.gsu.edu, yzheng@cs.ucsd.edu

Abstract

A new Open Artwork System Interchange Standard (OASIS) has been recently proposed for replacing the GDSII format. A primary objective of the new OASIS format is to enhance the compressibility of layout data. We compare the data compression capability of the *Full OASIS* set of operators with those also present in GDSII, which we refer to as the *Restricted OASIS* format. We measure the compression quality of the OASIS and GDSII operators in two contexts: (1) *compressible fill generation*, where the fill amounts are specified and compressible fill is then generated, and (2) *post-fill data compression*, when fill has already been generated and is then compressed. Our experimental results confirm the advantages of the OASIS compression operators: compressed file sizes using the *Full OASIS* format are on average about twice as small as those obtained using the *Restricted OASIS* format. We propose new OASIS-based compression algorithms which outperform industry physical verification tools. We also evaluate the respective merits of the individual repetition operators in OASIS and suggest possible improvements to the OASIS repetition operators.

1 Introduction

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 features*. Currently, area fill is added by physical verification tools (such as Mentor Graphics' Calibre) in the form of a flat "target layer" [19], which is eventually merged with the actual layout features at the mask data preparation step of the manufacturing handoff. Interconnect layers above M1 have little natural hierarchy that can be exploited, and contexts for instantiations of IP blocks may be different; this typically leads to a flat filling solution. According to the 2002 International Technology Roadmap for Semiconductors[14], 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 [3]. 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 [11]).

Off-the-shelf data compression techniques such as the Joint Bi-Level Image Processing Group (JBIG), Ziv-Lempel (LZ77) and ZIP cannot directly be used inside the standard GDSII Stream data format, and such techniques are therefore of limited use in today's design-to-manufacturing flows. However, for a direct-write maskless lithography system, a data processing system architecture and three compression algorithms are compared in [11] and an interesting alternative compression is suggested in [13]. Ueki et al. in [22] propose 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.

Traditionally, the standard format for layout data interchange is GDSII Stream, which uses basic shapes (polygon, path, box) and two compression operators: structure reference (SREF) and array reference (AREF). SREF allows hierarchy and AREF allows compact representation of regular two-dimensional arrays of structures. However, due to the rapidly growing volume of layout and fill data, GDSII file sizes have recently become unwieldy, in some cases growing to many tens of gigabytes [1]. Meeting a clear need for a more compressible format, a new standard layout data representation format called the *Open Artwork System Interchange Standard* (OASIS)[1] has been recently proposed.

This paper is motivated by two basic questions:

- How should new fill generation methods exploit the compression operators available in the OASIS layout representation format?
- Can the proposed OASIS standard be improved to further reduce compressed-fill data volume?

In the next section we measure the compression efficacy of the corresponding operators in the OASIS and GDSII formats. We then propose OASIS-based algorithms for generating compressible fill from scratch (Section 3), as well as for the compression of existing fill (Section 4). Section 5 describes the experimental results and proposes a new compression operator.

2 Compression Operators in OASIS

The OASIS format supports primitive geometric shapes (rectangle, polygon, path, trapezoid, circle and X-geometries), recursively defined *cells* and compression operators, called *repetitions*. A repetition is an array of either cells, geometries or texts. There are eight *repetition types* (see Table 1), as illustrated (for a rectangular fill feature) in Figure 1.

*This research was supported by a Packard Foundation Fellowship, by the MARCO Gigascale Silicon Research Center, by NSF Young Investigator Award MIP-9457412, and by NSF grant CCR-9988331.

[1]This new standard is expected to be simple enough to be widely supported by all EDA suppliers and CAD groups with prior knowledge of GDSII. OASIS is capable of representing the commonly used GDSII data types, which will enable a smooth migration from GDSII to the new and more efficient OASIS format.

TYPE	FORMAT	DESCRIPTION		# INT	R_c	Asymptotic R_c
1	x-dim y-dim [x-space] [y-space]	matrix or vector with uniform orthogonal spacing	$M \times N$ $(N > 0, M > 0)$	4	$7MN/11$	MN
2	x-dim x-space		$1 \times N$ $(N > 1)$	2	$7N/9$	N
3	y-dim y-space		$M \times 1$ $(M > 1)$	2	$7M/9$	M
4	x-dim x-space$_1$...x-space$_{N-1}$	vector with non-uniform orthogonal spacing	$1 \times N$ $(N > 1)$	N	$7N/(7+N)$	7
5	y-dim y-space$_1$...y-space$_{M-1}$		$M \times 1$ $(M > 1)$	M	$7M/(7+M)$	7
6	n-dim m-dim [n-disp] [m-disp]	repetition with uniform and non-orthogonal displacements	$N \times M$ $(N > 0, M > 0)$	4, 5, or 6	$7MN/(11, 12, or 13)$	MN
7	dim disp		P-element $(P > 1)$	2 or 3	$7P/9 \sim 7P/10$	P
8	dim disp$_1$ ... disp$_{P-1}$	repetition with non-uniform and non-orthogonal displacements	P-element $(P > 1)$	$2 \cdot P - 1$	$7P/(6+2P)$	3.5

Table 1: Syntax of the 8 OASIS repetition types & compression ratios R_c for the different types of rectangle repetitions. The parameters x-dim, y-dim, x-space, y-space, dim, n-dim and m-dim are all integers, while disp, n-disp and m-disp are all two-dimensional displacements called g-deltas. M and N are the numbers of rows and columns in the repetition. # INT is the number of integers needed to represent a repetition.

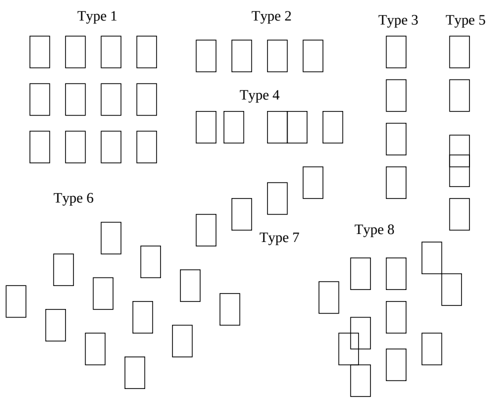

Figure 1: Examples of the 8 types of OASIS repetitions.

OASIS repetition types are based on *unsigned integers*[2] and the *g-delta* constructs representing a general (x, y) displacement using a pair of unsigned integers, or an orthogonal or 45-degree displacement using a single unsigned-integer. Table 1 summarizes the different types of repetitions and provides estimates of the number of unsigned integers required to represent each type. We also define the *starting point* of the repetition to be the left bottom point of the left bottom element of the repetition. Here, Type 6 repetitions may have different sizes, depending on the displacement direction: if one or both of the directions are multiples of orthogonal or 45-degree displacements, then their size may be either 4, 5, or 6 integers. Similarly, the size of a Type 7 repetition may also vary between 2 and 3 integers, depending on the displacement direction being either orthogonal or 45-degrees.

To compare the compression capabilities of different operators, we define the following measure of compression efficacy.

Definition 1 *The compression ratio R_c is the ratio of the size of a flat OASIS file (S_{orig}) to the size of its compressed version ($S_{compressed}$).*

Consider an $M \times N$ array of *Rectangles*, and let A be the number of integers required to represent a single record without any repetitions. Let R be the number of integers required to store the additional information when using repetitions. Then, the total number of integers required to store $M \times N$ independent rectangles would be $M \cdot N \cdot A$, while the total number of integers required to store these rectangles as an array with repetition would be $A + R$. The estimated *compression ratio* is hence given by:

$$R_c = \frac{(M \cdot N \cdot A)}{(A + R)} \tag{1}$$

Since R is always smaller than $(M \cdot N - 1) \cdot A$, there will be a reduction in data volume when using repetitions.

Table 1 gives the estimated compression ratio of each repetition type when area fill features are rectangles. Repetition Types 1 and 6 are the most powerful compression operators, followed by Types 2, 3 and 7, while repetition Types 4 and 5 are the least powerful compression operators. Type 8 is subsumed by the other repetition operators and thus has no significant effect on single-level data compression; however, it may still be useful in hierarchical cases, similarly to the SREF operator in GDSII.

We further define the *Full OASIS format* as allowing the use of repetition Types 1 through 7. The GDSII AREF construct can represent only the first three OASIS repetition types, We therefore define the *restricted OASIS format* as one that may use only repetition Types 1 through 3. To compare the compression capabilities of different operators, we define the following measure of compression efficacy.

In the next two sections we apply the compression operators of the Restricted and Full OASIS formats in two different contexts: (i) while generating compressible fill for a given fill distribution, and (ii) while compressing an existing fill pattern.

3 Compressible Fill Generation

A recent method of generating compressible fill using the GDSII constructs appears in [8]. This method applies in the regime where the fill generation is explicitly aware of compressibility and data volume issues. In this subsection we adapt the method of [8] to compare *Restricted OASIS* (repetition Types 1 through 3) with *Full OASIS* (repetition Types 1 through 7). All existing methods for area fill synthesis 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.

The Compressible Fill Generation Problem (CFGP): *Given a design rule-correct layout, generate a minimum number of OASIS operators to represent area fill features that keep window density variation within the given bounds (L, U).*

In the fixed-dissection regime, when the layout is partitioned into tiles, let F_{ij} be the required number of area fill

[2] An OASIS *unsigned integer* is an N-byte integer value.

features to be inserted into tile T_{ij}, computed using any existing method [3]. The following two formulations have been addressed in [8].

The CFGP in Fixed-Dissection Regimes: *Given a design rule-correct layout consisting of $m \times n$ tiles T_{ij}, and fill requirements F_{ij} for each tile, generate a minimum number of OASIS operators to represent area fill, such that each tile T_{ij} contains exactly F_{ij} area fill features.*

The Ranged CFGP in Fixed-Dissection Regimes: *Given a design rule-correct layout consisting of $m \times n$ tiles, generate a minimum number of OASIS operators to represent area fill, such that each tile T_{ij} contains a number of area fill features within the given range (L_{ij}, U_{ij}).*

The proposed algorithm in Figure 2 first greedily finds repetitions of Types 1, 2, 3, 6 and 7 with sufficiently large R_c, i.e., larger than the given lower bound L_{R_c}, and then completes the generation using repetition Types 4 and 5. The time complexity is $O(K \cdot L \cdot n^3)$, where $K \cdot L$ upper bounds the number of elements in any repetition. In Figure 2, $S_{\alpha,ij}$ is the number of unfilled free sites covered by a fill repetition A_α in each tile (i, j); letting $L_{ij} = U_{ij} = F_{ij}$ will solve the fixed fill problem.

Compressible Fill Generation with OASIS operators

1. Input site set G of an $M \times N$ multiple-tile (i, j),
   $i = 1 \ldots M$, $j = 1 \ldots N$;
2. **For** each free site in G in scan order **Do**
3. Find a fill repetition A_α among Type {1, 2, 3, 6, 7} with maximum R_c that does not overfill tiles;
4. **If** $R_c > L_{R_c}$ (given lower bound of compression ratio)
5. Update the fill requirements:
   $L_{ij} = L_{ij} - S_{\alpha,ij}$; $U_{ij} = U_{ij} - S_{\alpha,ij}$;
6. Update G;
7. **End For**
8. **While** ($L_{ij} > 0$) **Do**
9. Find a fill repetition A_α among Type {4, 5} with maximum compression ratio R_c that does not overfill tiles;
10. Update the fill requirements:
    $L_{ij} = L_{ij} - S_{\alpha,ij}$; $U_{ij} = U_{ij} - S_{\alpha,ij}$;
11. Update G;
12. **End While**

Figure 2: The greedy algorithm for the compressilble fill generation problem with OASIS operators.

4 Fill Compression

An alternative stage in the design-to-manufacturing flow where the fill data volume can be reduced is after the fill for the entire layout has been generated. Then the GDSII AREF construct or OASIS repetition operators are applied without changing the filled design.

The OASIS Fill Data Compression Problem (FDCP): *Given a layout containing area fill features, represent these area fill features using the OASIS repetition operators in a way that minimizes the resulting data volume.*

To detect compressible fill patterns, we represent the fixed-dissection layout region as a binary $m \times n$ matrix M $(=[m_{ij}]$, $1 \le i \le m, 1 \le j \le n$), where 1's correspond to area fill features, and 0's denote empty areas or original features. The intersection of a row (i) and column (j) in the matrix M is denoted by m_{ij}. Each nonzero element of the input matrix corresponds to a basic fill feature. A 0-1 matrix representation of fill layout is possible for the outputs of all major commercial fill insertion tools (e.g., Mentor's Calibre, Synopsys' Hercules and Cadence's Assura), even when operating in modes that output "tilted fill" or tiled "fill cells".

All of the algorithms that we study utilize the following priority scheme to search for repetitions:

1. Find a repetition of Type 1, 2, 3, 6, or 7 with maximum compression ratio R_c (this priority in identifying compression operators is based on the compression ratio analysis from Table 1).
2. If $R_c > L_{R_c}$, output the repetition and update the fill data. The constant L_{R_c} is an experimentally derived lower bound on the compression ratio for acceptance of repetitions of Type {1, 2, 3, 6, 7}. Our algorithms all use $L_{R_c} = 5.0$.
3. Repeat 1 & 2 until no repetition exists with $R_c > L_{R_c}$.
4. Find a minimum number of Type 4 or 5 repetitions to cover the remaining fill geometries using bipartite matching.

Step 4 of the algorithm above can be implemented optimally, since each repetition of Type 4 (respectively Type 5) covers an entire row (respectively column). The problem of covering all remaining points with the minimum number of repetitions of Types 4 and 5 is therefore equivalent to the well-known problem of finding minimum vertex cover in a bipartite graph $H = (V = R \cup C, E)$, where vertices correspond to the set R of rows and the set C of columns, and edges connect columns and rows iff there is a 1 at their intersection in the 0-1 matrix. Applying Konig's Theorem, the minimum vertex cover in a bipartite graph H can be derived from a maximum matching which can be found within time $O(N\sqrt{m+n})$, where N is the number of 1's [10].

4.1 Exhaustive Search Based Greedy Algorithm (ESBG)

A straightforward but inefficient way to represent the fill features using the OASIS repetition operators is to perform exhaustive search for each repetition type. In our *ESBG* method, each 1-element of the input 0-1 matrix is treated as the bottom-left corner of a potential maximal parallelogram or rectangle (the latter being a special case of the former) with all the corners of the parallelogram containing 1's (Types 1, 2, 3, 6 and 7). If a repetition type originating from a 1-element with maximum compression ratio satisfies the given minimum compression ratio requirements, it is then saved and all of the sites covered by this repetition are marked as *visited*. This process is repeated, each time using the remaining unvisited 1-elements in the matrix as potential starting points, until all of the 1-elements have been either included in repetitions or tried as starting points. Finally, any remaining 1-elements that have not been marked as visited are covered by repetition Types 4 and 5 using a bipartite perfect matching based minimum vertex cover method as outlined above. Figure 3 gives a formal description of our algorithm.

4.2 Regularity Search Based Greedy Algorithm (RSBG)

The exhaustive search based greedy method described above may be computationally inefficient for large layouts. We can

[3]Examples include LP methods [16] [20], greedy and Monte-Carlo methods [6] [7], and iterated greedy / Monte-Carlo methods [7].

ESBG Algorithm for Fill Data Compression
1. Input fill matrix M;
2. **Do**
3. Find next unmarked and unvisited site in B in
4. Search exhaustively for maximum repetitions of Types $\{1, 2, 3, 6, 7\}$ originating from the site;
5. Pick the repetition type having the maximum R_c assuming that $R_c > L_{R_c}$ (given lower bound on R_c);
6. Mark all sites covered by the repetition;
7. **Until** all fill sites are marked or used as starting points;
8. Use bipartite matching to find a minimum number of repetition Types $\{4, 5\}$ covering remaining unvisited sites.

Figure 3: Exhaustive search based greedy algorithm for the fill data compression problem.

improve on this approach using a regularity detection technique for planar pointsets [15]. The work of [15] addresses two problems for a given set of N points (Figure 4):

- Finding subsets of equally spaced collinear points.
- Finding subsets of regularly spaced parallelogram cells.

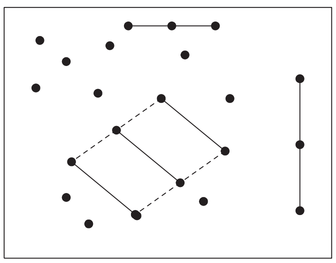

Figure 4: Finding regularities in a pointset.

Our proposed *greedy* fill compression algorithm uses 1-elements of the input matrix to represent points in the plane, and 0-elements of the input matrix to denote intermediate spaces. Three levels of granularity among the elements are considered during the search to find maximal repetition types: points, line segments formed by pairs of points, and parallelograms formed by sets of four distinct points.

Our heuristic finds a group of congruent adjacent cells[4] and selects repetitions which contain maximal numbers of unvisited points located among cells of this group. The proposed greedy method (Figure 5) starts with the set of points determined by the input 0-1 matrix M, and forms all possible segments over point pairs. Next, these segments are sorted in non-decreasing order of their length, as projected onto the axes. Parallelograms are then formed from pairs of segments having the same length and slope; this is accomplished by scanning the sorted segment list in a left-right and bottom-up order. Next, a graph is constructed where each graph vertex corresponds to a cell, and edges correspond to geometric adjacency between neighboring cells (i.e., congruent cells that share a common edge). For each connected component of this graph, we search for maximal repetitions of Types 1, 2, 3, 6 and 7, as detailed in Steps (14) through (20) of the algorithm in Figure 5. This method for identifying maximal repetitions extends the method of [15].

The largest of all maximal repetitions is thus identified, and the corresponding points are marked as visited. The

[4] A cell is defined by a pair of distinct line segments having the same length and slope. The offset between two segments S_i and S_j is defined by a horizontal width $w(S_i, S_j)$ and a vertical height $h(S_i, S_j)$.

RSBG Algorithm for Fill Data Compression
Input: binary matrix of fill pattern;
Output: Compressed OASIS file
1. Add segments between all pairs of points into the segment array R by scanning the input binary matrix;
2. Sort all the segments in R by their y-distance, then x-distance using bucket sort; Build an adjacency list G (i.e., list of congruent adjacent cells)
3. **For** each subset Q in R where each segment has the same x and y **Do**
4. $G = (V, E) = (\emptyset, \emptyset)$;
5. **For** $k_1 = 1 : s(Q) - 2$ **Do**
6. $k_2 = k_1 + 1$; $k_3 = k_1 + 2$;
7. **While** $k_3 \leq s(Q)$ **Do**
8. $(A, B, C) = (k_1, k_2, k_3)$;
9. **If** $w(A, B) = w(B, C)$ and $h(A, B) = h(B, C)$
10. $V = V \bigcup \{(A, B), (B, C)\}$; $E = E \bigcup \{((A, B), (B, C))\}$;
11. **Else If** $h(A, B) > h(B, C)$ or $w(A, B) > w(B, C)$, $k_3 = k_3 + 1$;
12. **Else** $k_2 = k_2 + 1$;
13. Sort all vertices in the adjacency list G by its w and h using bucket sort;
/* Find repetitions in G */
14. **For** each subset P of G where cells have same w, h **Do**
15. **For** each vertex $v \in P$ **Do**
16. Search the repetition Types 1, 2, 3, 6 and 7 originating from v;
17. Save into the set S the repetition with the maximum compression ratio bigger than L_{R_c} (based on the number of unvisited points);
18. Select from S a repetition with the maximum compression ratio, and mark the points contained in the selected repetition as visited;
19. Update compression ratios of the repetitions in S, remove the selected repetition and the repetitions whose compression ratios are smaller than L_{R_c};
20. Go to Step 18 until $S = \emptyset$;
21. Use a bipartite matching algorithm to find the minimum number of instances of repetition Types 4 and 5 that cover the remaining unvisited points;
22. Output all saved repetitions.

Figure 5: Regularity detection based greedy approach for fill data compression.

same process is repeated for the remaining set of points, until no feasible new repetitions nor unvisited points remain. Note that the resulting repetitions may overlap with each other, and thus some points may be covered by more than one repetition (i.e., some fill elements may be represented redundantly). Finally, Step (21) of Figure 5 *optimally* covers the remaining points with repetitions of Types 4 and 5.

The overall time complexity of this algorithm is $O(N^3)$, which may still be prohibitive if there are millions of fill elements. We may therefore partition the layout into k blocks to speed up the runtime, with the time complexity of this modified variant being $O(N^3/k^2)$. Note that since the minimum vertex cover can be found much faster, it is possible to employ a coarser partition into blocks (or forgo altogether the partition into blocks) in Step (19). Alternatively, we can initially run an optimal algorithm for finding a minimum covering with repetitions of Types 4 and 5. Then, each candidate repetition may be selected only if it reduces the size of the minimum covering.

5 Computational Experience

All of our experiments were performed on metal layers extracted from industry standard-cell layouts (Table 2). Our experimental testbed integrates GDSII Stream input and internally-developed geometric processing engines, coded in C++ under Solaris 2.8. All runtimes are reported in CPU seconds on a 300 MHz Sun Ultra-10 with 1 GB of RAM.

Testcase	T1	T2	T3	T4	T5	T6
layout size	819,200	522,060	819,200	819,200	125,000	112,000
# rectangles	142.585	133,873	32,258	78,293	49,506	76,423

Table 2: The industry test cases.

5.1 Compressible Fill Generation Results

We assess the OASIS-based compressible fill generation approaches by inserting low density fill as well as high density fill into the layout.[5] Table 4 reports the compressible fill generation results. Columns 2 and 3 of Table 4 indicate that the compression ratios using the Full OASIS format are on average twice those using the Restricted OASIS Format. This confirms an advantage of the OASIS compression operators over those of GDSII.[6] Table 4 also shows the performance of different types of repetition combinations. Since the sizes of repetition Types 1, 2, 3, 6, and 7 are independent of the number of 1's covered, and the sizes of repetition Types 4 and 5 are dependent on the number of 1's, using repetition Types 4 and 5 achieves better compression results if the fill requirements are small. Otherwise, if the fill requirements are large, using Type 1, 2, 3, 6, and 7 repetitions will achieve better compression results. We also find that Full OASIS compression is slightly worse than using only Types 1, 2, 3, 4 and 5. The reason is that using Type 6 and 7 repetitions may break large instances of Type 1, 2 or 3, yet may fail to decrease the number of Type 1, 2 and 3 repetitions since these are all independent of the number of 1's.

Finally, Table 3 compares the greedy compressible fill generation algorithm with OASIS operators (GCF) with the off-the-shelf data compression tool GZIP. GZIP has been applied to the GDSII file containing only the generated fill. The average compression ratio is 5.5 for GZIP, 8.4 for GCF, and 50.6 for GCF followed by GZIP. The average runtime is 13.3 seconds for GZIP, 2.5 for GCF, and 3.8 seconds for GCF followed by GZIP. These results indicate that GCF is faster and better than GZIP and these two compressions can be applied simultaneously without worsening compression ratios of each other.

5.2 Fill Compression Results

Experimental comparison and verification of the two fill data compression algorithms – the greedy algorithms based on regularity detection (RSBG, Figure 5), and the exhaustive search (ESBG, Figure 3) – have been performed over the same test cases filled by the following four methods: Iterated Monte-Carlo (IMC) as described in [7], the Mentor Graphics Calibre tool, the Cadence Assura tool, and our compressible fill generation method using either the Restricted OASIS or Full OASIS formats. Table 5 presents fill compression results for low and high density fill data using the ESBG and RSBG methods. The compression using the Full OASIS format is better than using the Restricted OASIS format by factor of 1.4 for ESBG and by 2.0 for RSBG. ESBG has advantage in compression and runtime over RSBG for the restricted OASIS and RSBG has advantage over ESBG for the full OASIS format.

We can expect that compressible fill should enable better compressibility than IMC-generated fill. Since high density fill covers almost all free sites in the layout, compressible fill generation in a high-density scenario does not offer much advantage, as expected. However, low density fill produced by the compressible fill generation method always yields better compression than IMC.

[5] The high density fill was produced by fill synthesis with the Min-Var objective, while the low density fill was produced by fill synthesis with the Min-Fill objective.

[6] Full OASIS compression will work well on specific styles of fill, such as "tilted fill" using repetition Types 6 and 7.

Testcase	Fill	GZIP		GCF		GCF+GZIP	
T/W/r/s	CPU	C_Ratio	CPU	C_Ratio	CPU	C_Ratio	CPU
T1/80k/4/15	34.07	5.24	8.63	12.90	5.04	71.59	5.64
T2/80k/4/15	40.44	5.44	12.36	4.04	5.62	25.94	7.57
T3/80k/4/15	36.11	5.89	10.73	10.48	0.39	61.07	1.16
T4/12k/4/2	13.61	5.51	10.48	7.99	1.29	49.75	2.34
T5/12k/4/2	31.80	5.44	31.92	6.49	0.54	42.91	3.53
T6/12k/4/2	18.13	5.55	5.78	8.44	1.86	52.18	2.49

Table 3: Comparison of fill synthesis methods: GZIP, GCF and GCF followed by GZIP. **Notation:** *T/w/r/s:* testcase / window size / r-dissection / site size; *C_Ratio:* compression ratio; *CPU:* runtime in second.

testcase	Low Density Fill				High Density Fill			
T/w/r/s	I	II	III	IV	I	II	III	IV
T1/32k/2/15	6.9	11.0	11.0	6.4	3.4	8.5	8.6	7.4
T1/32k/2/10	7.2	11.8	11.9	6.7	4.0	8.4	8.6	6.8
T1/32k/2/5	13.0	16.7	16.5	6.9	7.6	10.8	11.4	6.9
T1/32k/4/15	2.6	6.1	6.1	6.0	2.7	7.3	7.3	6.5
T1/32k/4/10	3.4	7.4	7.5	6.7	3.5	7.8	8.0	6.8
T1/32k/4/5	8.0	11.3	11.5	7.0	6.7	10.3	10.5	6.9
T2/32k/2/15	7.5	9.8	10.3	6.5	3.4	6.7	6.8	5.6
T2/32k/2/10	6.2	10.2	10.5	6.5	3.7	7.6	7.8	6.3
T2/32k/2/5	11.0	14.4	15.1	6.9	6.3	9.2	9.7	6.8
T2/32k/4/15	3.2	5.7	5.7	5.9	2.6	5.8	5.8	5.5
T2/32k/4/10	3.7	7.2	7.3	6.2	3.2	7.3	7.3	6.3
T2/32k/4/5	6.2	9.6	9.8	6.9	5.6	9.0	9.3	6.8

Table 4: Compression ratios of low/high density compressible fill. **Notation:** *T/w/r/s:* testcase / window size / r-dissection / site size; *I:* Restricted OASIS; *II:* Full OASIS; *III:* Using Type 1, 2, 3, 4, and 5 repetitions; *IV:* Using Type 4 and 5 repetitions;

Table 6 reports fill compression results for fill data generated by the Mentor Graphics Calibre tool and the Cadence Assura tool. Our method yields better compression than Calibre does on its own output data. When analyzing the number of different repetition types used in fill compression, we have observed that repetitions Types 6 and 7 are not ideal choices, unless there are many Type 6 and 7 patterns in the layout. We further observe that repetitions Types 4 and 5 clearly provide Full OASIS with additional compression capability as compared to Restricted OASIS.[7]

[7] After extracting repetition Types 1, 2, 3, 6, and 7, we apply both the perfect matching based minimum vertex cover method, as well as the greedy algorithm for finding repetition instances of Type 4 and 5. The minimum vertex cover method wins 3.8% over greedy in ESBG, and 0.7% over greedy in RSBG.

Low Density Fill data generated by IMC								
testcase	ESBG		RSBG		ESBG		RSBG	
T/w/r/s	Rest	CPU	Rest	CPU	Full	CPU	Full	CPU
T1/32k/2/15	3.6	1	3.3	16	5.5	342	6.0	3
T1/32k/2/10	3.6	4	3.3	106	6.2	3134	6.9	30
T1/32k/2/5	4.4	35	3.5	1078	6.5	23945	7.8	664
T2/32k/2/15	4.6	1	4.1	3	6.0	116	7.2	6
T2/32k/2/10	4.1	2	3.8	56	6.7	810	7.7	171
T2/32k/2/5	4.7	42	3.6	816	7.1	9694	8.2	2383
Low Density Compressible fill generation data								
testcase	ESBG		RSBG		ESBG		RSBG	
T/w/r/s	Rest	CPU	Rest	CPU	Full	CPU	Full	CPU
T1/32k/2/15	5.4	0.3	5.1	11	7.6	459	7.8	6
T1/32k/2/10	5.9	2	4.8	96	8.0	1335	8.2	223
T1/32k/2/5	9.2	110	4.6	934	9.6	6106	8.6	2048
T2/32k/2/15	6.5	0.3	6.5	4	7.9	149	9.5	13
T2/32k/2/10	6.0	3	4.5	57	8.6	842	8.3	247
T2/32k/2/5	9.2	45	4.6	730	9.7	4377	8.6	2968
High Density Fill data generated by IMC								
testcase	ESBG		RSBG		ESBG		RSBG	
T/w/r/s	Rest	CPU	Rest	CPU	Full	CPU	Full	CPU
T1/32k/2/15	4.0	6	3.3	107	7.1	5137	8.0	1246
T1/32k/2/10	4.9	54	3.4	344	6.8	23621	7.8	5719
T1/32k/2/5	7.8	540	4.6	2118	7.7	146729	9.1	22586
T2/32k/2/15	3.8	1	3.7	19	6.5	1479	7.0	99
T2/32k/2/10	4.4	4	3.7	83	6.6	3918	7.6	366
T2/32k/2/5	6.7	51	4.4	583	7.1	38416	8.4	3099
High Density Compressible fill generation data								
testcase	ESBG		RSBG		ESBG		RSBG	
T/w/r/s	Rest	CPU	Rest	CPU	Full	CPU	Full	CPU
T1/32k/2/15	4.0	2	3.6	139	7.5	4184	8.5	626
T1/32k/2/10	4.9	47	3.3	576	6.8	7607	7.8	5464
T1/32k/2/5	7.9	572	4.5	973	7.8	117108	9.0	36297
T2/32k/2/15	3.8	1	3.8	19	6.4	759	7.0	255
T2/32k/2/10	4.4	5	3.7	82	6.7	2948	7.5	285
T2/32k/2/5	6.8	72	4.4	577	7.1	18074	8.4	13167

Table 5: Fill compression ratios for low/high density fill data. **Notation:** *Rest:* Restricted OASIS; *Full:* Full OASIS; *CPU:* runtime (in seconds.)

Test	Commercial	ESBG		RSBG	
Tl/T/F	Tools	Rest	Full	Rest	Full
MGC/T1/1	1.64	6.58	7.28	3.69	8.03
MGC/T2/1	2.30	8.07	8.60	4.45	8.60
MGC/T2/2	1.86	6.09	8.05	4.18	8.15
MGC/T2/3	2.02	5.52	7.83	4.89	7.83
MGC/T2/4	2.19	4.86	6.02	3.41	8.00
Assura/T2/1	N/A	3.63	5.51	2.46	5.81

Table 6: Fill compression ratios for fill data generated by Calibre and Assura. **Notation:** *Rest:* Restricted OASIS; *Full:* Full OASIS; *Tl/T/F:* Tool/Testcase/File, Tool is Mentor Graphic Calibre (MGC) or Cadence Assura (Assura);

6 Conclusions

In this paper, we compared fill data volume reduction using the old GDSII format against the newly proposed OASIS format. We also compared the merit of using OASIS, versus using a restricted version of OASIS where only a subset of the operators are utilized. Our experimental results illustrate the superiority of the OASIS compression operators over the corresponding GDSII operators.

The following two interesting potential modifications to the OASIS *repetition* operator may enhance compression effectiveness. We propose a new *irregular array* OASIS construct, equivalent to the combination of Type 4 and 5 repetitions, i.e., a new repetition type with possibly non-uniform spacing between elements along the x- and y-directions. This repetition type will (i) have a potentially unbounded compression ratio O(MNA/(M+N+A)), and (ii) be less uniform, yet highly compressible fill. By varying the x and y spacings, it is possible to achieve fill distributions closer to that generated by efficient Monte-Carlo methods [5].

We also propose the inclusion of a pseudo-random number generator in the OASIS format, which will substantially simplify the application of Monte-Carlo methods for CMP layout density control. When generating compressible fill (see [8]), large amounts of feasible sites can be described using a small number of repetitions, and a built-in pseudo-random number generator can thus be used to reproducibly and completely specify the filling of a prescribed number of sites.

References

[1] New Standards Specification for Open Artwork System Interchange Standard, Dec. 11, 2002. http://www.semi.org/web/wcontent.nsf/url/stds_blueballot,

[2] 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 VLSI Manuf.*, July 2001.

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

[4] Computers and Intractability: NP-Completeness, Garey and Johnson, 1978.

[5] Y. Chen, A. B. Kahng, G. Robins and A. Zelikovsky, "Smoothness and Uniformity of Filled Layout for VDSM Manufacturability", *Proc. ACM/IEEE Intl. Symp. on Physical Design*, April 2002, 137-142.

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

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

[8] Y. Chen, A. B. Kahng, G. Robins, A. Zelikovsky, and Y.H.Zheng, "Data Volume Reduction in Dummy Fill generation", Proc. Design,Automation and Testing in Europe, Munich, March 2003, pp. 868-873.

[9] W. J. Cook and A. Rohe, "Blossom IV - a minimum weighted perfect matching solver", [online], Available: http://www.or.uni-bonn.de/home/rohe/matching.html.

[10] T. H. Cormen, C. E. Leiserson, R. L. Rivest and C. Stein, "Introduction to Algorithms," MIT Press, 2001.

[11] 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.

[12] 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.

[13] 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.

[14] International Technology Roadmap for Semiconductors, 2002 http://public.itrs.net

[15] Andrew B. Kahng and G. Robins, "Optimal algorithms for extracting spatial regularity in images", *Pattern Recognition Letters*, 12(1991), 757-764.

[16] 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.

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

[18] D. Ouma, D. Boning, J. Chung, G. Shinn, L. Olsen, and J. Clark, "An Integrated Characterization and Modeling Methodology for CMP Dielectric Planarization", *Intl. Interconnect Technology Conference*, San Francisco, CA, June 1998.

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

[20] 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.

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

[22] 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.