Exact Basic Geometric Operations on Arbitrary Angle Polygons Using only Fixed Size Integer Coordinates

Alexey Lvov
IBM T.J.Watson Res. Center
1101 Kitchawan Rd. (rt. 134) 34-159
Yorktown Heights, NY, 10598
Email: lvov@us.ibm.com

Ulrich Finkler
IBM T.J.Watson Res. Center
1101 Kitchawan Rd. (rt. 134) 32-118
Yorktown Heights, NY, 10598
Email: ufinkler@us.ibm.com

***Abstract*— As the semiconductor technology is scaling down to nanometer regime, accurate layout analysis requires operations with simulated contours of VLSI layouts which are non-rectilinear.**

The basic operations on non-rectilinear shapes include "intersection", "difference", "union", "find connected components" and "find boundary". Whatever precision one would choose for the representation of vertices of shapes A and B the exact representation of, say, $A \cap B$ would require an even higher precision. Consequently rounding of intermediate results of a chain of basic operations seems to be inevitable. The presence of rounding errors is unacceptable because the implementation of algorithms for all operations which involve topology, such as "find connected components", " find boundary", etc, becomes impossible or extremely complex

We present a complete solution to the following problem: Let a VLSI design or a simulated through a lithography process image of a VLSI design be given by a number of two-dimensional point sets. Each point set consists of polygons with arbitrary (not necessarily 90^o) angles and with vertices on an integer grid. Perform any amount of sequential basic operations so that:

• The resulting point sets are mathematically exact, that is no rounding errors are allowed. In particular the connectivity of the point sets remains intact, the boundaries remain undistorted and the statements like "$A = (A \setminus B) \cup (A \cap B)$", "$(A \setminus B)$ is disjoint with B" and "$A \cap B \subset A$" always hold.

• The amount of time and memory per one basic geometric operation on elementary polygons (trapezoids) is constant. For example it would not be an acceptable solution to represent the vertices of the new shapes which appear as a result of the operations of intersection, difference, etc, by unlimited length rational numbers.

I. Introduction

One of the key problems of deep-subwavelength technologies (90nm and below) is the decreasing ability to control the manufacturing process [1][2]. Main sources of variability and distortion of manufactured contours are the lithographic patterning [3] and layer thickness [4].

To deal with these effects, the semiconductor industry is employing simulation technologies to analyze and compensate the effects of the manufacturing process, in particular in the area of lithography. For example post-tapeout manufacturability enhancements, like OPC (Optical Proximity Correction), aim to compensate for systematic distortions caused by the manufacturing process. These enhancements modify the design shapes such that the wafer contours resulting from manufacturing process are close to the shapes produced by the pre-tapeout design phase [5][6] [7][8] [9] [10] [11][12].

VLSI designs are analyzed in detail before production. Yield prediction, the extraction of electrical properties and other parameters are traditionally performed on the undistorted VLSI design, the rectilinear target contours. The increasing deviations of the manufactured contours from the target contours have lead to an increasing trend to perform *through-process* analysis. I.e., the analysis tasks are not performed on the target contours, but the contours obtained by a simulated manufacturing process, predominantly the simulation of lithographic effects.

While the VLSI target shapes consist predominantly of rectilinear contours, the contours resulting from through process simulation are highly non-rectilinear. Thus, any analysis of through-process contours has to perform geometric operations on large quantities of non-rectilinear shapes.

There is a rich body of literature on computational geometry for general angle contours. They can roughly be classified into several groups.

The first group uses infinite precision arithmetic and focuses on the efficient implementation of these arithmetics [14][13] [21][19] [15][16][17]. Infinite precision arithmetic is orders of magnitude slower than basic integer arithmetic. Thus, VLSI design applications based on this approach would run orders of magnitude slower than their counterparts for rectilinear shapes, which is prohibitive due to the data volumes encountered in VLSI design processing.

A second group of approaches for geometry kernels uses inexact arithmetic and minimizes the impact of rounding errors [13][15] [18][21][20]. Rather sophisticated techniques are necessary to ensure the proper completion of algorithms and the avoidance of topologically invalid results. For VLSI applications, small variations can create results that do not preserve the electrical connectivity (see Section II). The amount of complexity that is necessary to ensure correct results makes it difficult to obtain correct implementations and leads to runtimes that are inadequate for many through-process VLSI applications.

A third group of practice oriented geometry kernels takes advantage of the fact that the variety of non-rectilinear geometry is very limited in VLSI designs (often only angles of 45 degrees are legal) and that the number of critical geometric operations is limited. This approach is widely used in VLSI applications and has been shown to be sufficiently efficient and accurate when operating on designs of strongly limited non-recti-linearity. The contours in through-process analysis do not adhere to the restrictions that are required for this group of geometry kernels.

In "constructive solid-geometry" CSG representations [23][22] polyhedra are represented through planes in three dimensions [24] [25] [27]. [26] proposes a very interesting *approximate* representation of polyhedra through planes containing the boundary of the polyhedra. Though non-exact, this representation preserves the connectivity and boundary under all basic geometric operations.

We present a technique based on the *exact* representation of two-dimensional polygons as intersections of a finite number of half planes (dual representation). Each half plane can be described through at most eight integers of fixed size. A geometry kernel implemented on the basis of this technique provides a set of operations that

978-1-4244-2820-5/08/$25.00 ©2008 IEEE

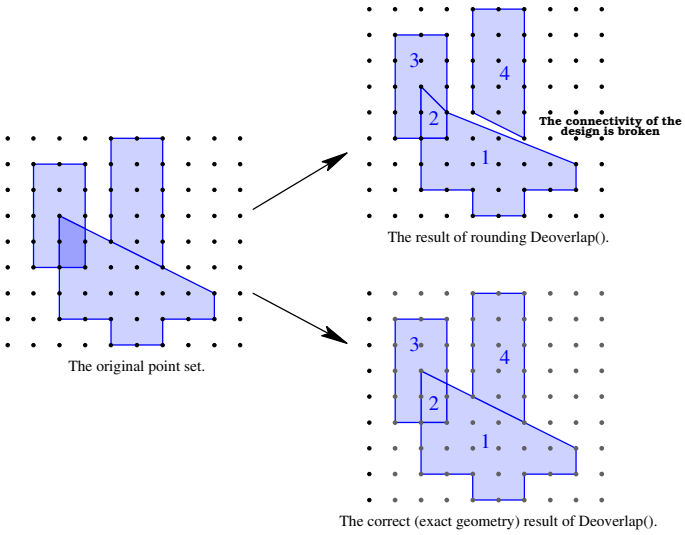

Fig. 1.

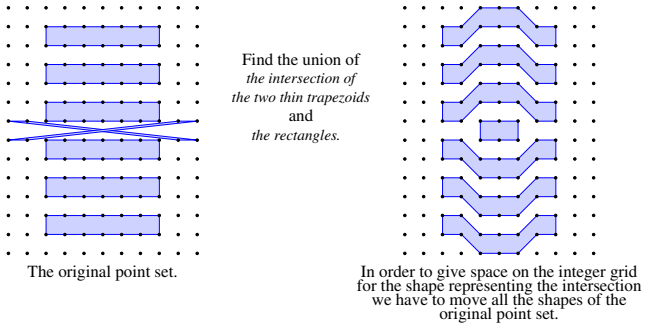

Fig. 2.

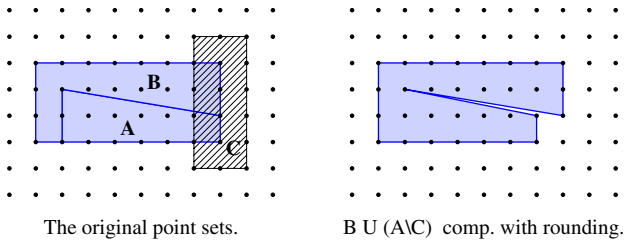

Fig. 3.

are critical for through-process VLSI design analysis and combines exactness with excellent performance. Its runtime on highly non-rectilinear contours has been shown to be one to two orders of magnitude faster than an industrial VLSI processing tool based on *smart rounding*.

II. Why Exact Geometry is Necessary

The rounding errors create a whole spectrum of problems for operations on VLSI designs. The accumulation of errors is the most harmless of them and can be overcome just by using floating point numbers with high enough precision. The following three problems are much more severe and make it practically impossible to create an effective rounding-based computer aided design tool for VLSI circuits with arbitrary angle shapes.

A. Breakage of the Electrical Connectivity of the Design

The electrical connectivity graph is one of the most important characteristics of the design. Rounding errors can easily create electrical shorts as well as connectivity breakage (Figure 1).

The situation above is by far not the only type of connectivity breakage that happens due to integer coordinates limitation. If it was alone, some special adjustments ("smart rounding") for this particular situation could be made. Unfortunately there exist worse examples, see Figure 2. Here having an electrical short is guaranteed unless we make adjustments to *all* shapes of the original design.

All the types of connectivity distortion due to rounding do not admit any reasonable classification. If they occur in combinations they can not be corrected simultaneously unless we spend time proportional to the size of the whole point set per each short or breakage.

B. Boundary Distortion

What is the length of the boundary of the set $B \cup (A \setminus C)$ on Figure 3? The correct answer is 20.1805... . However the computation with rounding to the integer grid would give us 30.1818... .

C. The Most Severe Problem: Breakage of the Main Equalities and Impossibility of Building More Complicated Algorithms on Top of Inexact Basic Geometric Operations

Just as "$a+b-a=b$" does not hold for floating point numbers, the basic statements such as

$$A = (A \setminus B) \cup (A \cap B),$$
$$(A \setminus B) \text{ and } B \text{ are disjoint},$$
$$A \cap B \subset A, \text{ etc.}$$

do not hold for point sets in the presence of rounding errors. For example testing if two point sets A and B are equal is an extremely easy task if the computations are exact:

$$A = B \text{ if and only if } A \setminus B = \emptyset \text{ and } B \setminus A = \emptyset.$$

However it becomes a very "unpleasant" operation if rounding errors can occur: One must set some threshold ε for the area, and then check if $area(A \setminus B) < \varepsilon$ and $area(B \setminus A) < \varepsilon$, where $A \setminus B$ and $B \setminus A$ are unions of possibly overlapping polygons.

As one can see the inexactness of computation in the example above creates some problems which can be overcome with certain additional efforts. However if one needs to design a more complicated algorithm, like "deoverlap" or "find boundary", one would face a bunch of practically unsolvable problems.

"Deoverlap": Given a set of overlapping shapes find an equivalent set of disjoint shapes. In the exact model the algorithm is trivial: Find a pair of overlapping shapes A and B and replace it with the pair $(A \setminus B,\ B)$. Repeat until there remains any overlapping shapes. In the presence of rounding errors this scheme no longer works because $(A \setminus B)$ and B are not necessarily disjoint. Adding an extra check if the intersection $A \cup B$ is small enough to be counted an empty set is conjugated with a lot of extra problems and practically leads to the increase of runtime by ≥ 10 times and to the tremendous increase of the complexity of the program.

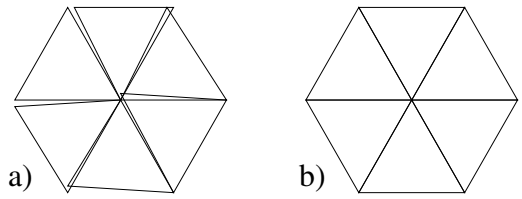

Fig. 4.

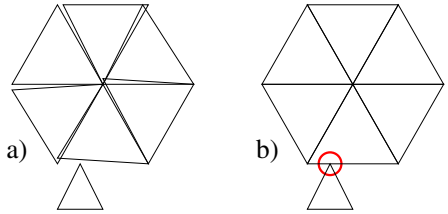

Fig. 5.

"Find boundary": How would you find a boundary of a point set shown on Figure 4a? One way would be: introduce some small constant $\varepsilon > 0$, merge all vertices at distance $\leq \varepsilon$, and then find the boundary of the point set 4b. But under a closer look this fails to be a valid method: what if we had another shape close enough to an edge (but not to a vertex) of one of the polygons (Figure 5a)? In that case our small movement of vertices would create an electrical short (Figure 5b). We can take some additional care of this type of unwanted effects too, but if we want our algorithm to work correctly in all cases we will have an endless chain of such corrections. This gives very strong intuitive grounds to assume that an algorithm for finding a boundary of a point set in the presence of rounding errors either does not exits or, if exists, requires a very long time for both programming and execution and must have an extremely complicated prove of correctness.

III. The Exact Geometry Algorithm

We assume that point sets are given as unions (may be not disjoint) of polygons. The algorithm consists of two parts:

1) Define a class of *elementary shapes* so that any polygon with integer vertices can be decomposed into a number of shapes from this class. For example the elementary shapes could be triangles with integer vertices, triangles with one side parallel to y-axis, trapezoids with bases parallel to y-axis or something else. In our algorithm the elementary shapes are a special kind of vertical trapezoids (with non-integer vertices) described later in this section.
2) Preform the basic geometric operations on elementary shapes so that all the resulting shapes remain within the class of elementary shapes.

A. The Elementary Shapes

We assume that all input point sets belong to some universal bounding box

$$[-CoordMax, CoordMax] \times [-CoordMax, CoordMax].$$

Definition 3.1: Call a non-vertical line passing through two different integer points inside of the universal bounding box an "I-line".

Definition 3.2: Call a trapezoid with vertical bases a "V-trapezoid" if

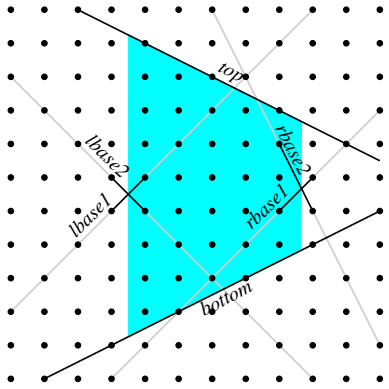

Fig. 6.

- The x-coordinate of its left base is equal to the x-coordinate of the point of intersection of a pair of I-lines,
- The x-coordinate of its right base is equal to the x-coordinate of the point of intersection of a pair of I-lines,
- Its lower and upper sides are segments of I-lines.

see Figure 6.

Note: Vertices of a V-trapezoid are not necessarily integer.

Note: A V-trapezoid is completely defined by the six-tuple of I-lines

$$\left\{ \quad (lbase1, lbase2), \quad (rbase1, rbase2), \quad bottom, \quad top \quad \right\}.$$

It requires 24 integers from the segment $[-CoordMax, CoordMax]$ to store a V-trapezoid.

Lemma 3.3: Any polygon with integer vertices can be represented as a disjoint union of V-trapezoids.

Proof: Draw a vertical cut through each vertex until it remains inside of the polygon. This breaks the polygon into disjoint y-parallel trapezoids. The lower and upper sides of each trapezoid are non-vertical segments of the boundary of the original polygon and thus are segments of I-lines. The x coordinates of the bases of the trapezoids are the x coordinates of the vertices of the polygon. Each vertex is an integer point and can be represented as the intersection of the I-lines going through it at $+45^o$ and -45^o angles. Thus the y-parallel trapezoids of the decomposition are V-trapezoids.

B. The Outline of the Algorithm

We have to show that the intersection, difference and union of two V-trapezoids can be decomposed into a number of disjoint V-trapezoids. The input to our algorithm are two V-trapezoids A and B:

Basic(*V-trapezoid* A, *V-trapezoid* B);

The output is a set of disjoint V-trapezoids with tags which can take three values: "belongs to A only", "belongs to B only" and "belongs to both A and B". This way the algorithm finds $A \cup B$, $A \cap B$, $A \setminus B$ and $B \setminus A$ simultaneously. It can be checked by looking at all possible configurations of the input V-trapezoids that the number of the output V-trapezoids can not exceed 15.

The input V-trapezoids are defined by six I-lines each:

$$A = \left\{ \quad (al1, al2), \quad (ar1, ar2), \quad ab, \quad at \quad \right\},$$
$$B = \left\{ \quad (bl1, bl2), \quad (br1, br2), \quad bb, \quad bt \quad \right\}.$$

Property of the algorithm # 1: All V-trapezoids of the output are defined by some six-tuples of I-lines which are subsets of the above set of 12 I-lines. No new I-lines are ever created during the basic geometric operations.

Property of the algorithm # 2: All operations with numeric coordinates are wrapped into just two boolean functions on I-lines described in the next subsection. With the exception of these two functions the algorithm uses only combinatorial operations.

C. The Two Boolean Operations on I-lines Required for the Algorithm

1. `bool` point_A_is_not_to_the_left_of_point_B(I-line $a1$, I-line $a2$, I-line $b1$, I-line $b2$);

Returns `true` if and only if I-lines $a1$ and $a2$ intersect at some point A, I-lines $b1$ and $b2$ intersect at some point B and (x coordinate of A) $\geq$ (x coordinate of B). Or in terms of the coordinates returns `true` if and only if

$$\det A \neq 0 \quad \text{and} \quad \det B \neq 0 \quad \text{and} \quad \det_x A \cdot \det B \geq \det_x B \cdot \det A,$$

where

$$\det A = (a1_{y1} - a2_{y1})(a2_{x2} - a1_{x2}) - (a1_{y2} - a2_{y2})(a2_{x1} - a1_{x1}),$$
$$\det B = (b1_{y1} - b2_{y1})(b2_{x2} - b1_{x2}) - (b1_{y2} - b2_{y2})(b2_{x1} - b1_{x1}),$$
$$\det_x A = (a1_{y1}a2_{x1} - a1_{x1}a2_{y1})(a2_{x2} - a1_{x2}) - (a1_{y2}a2_{x2} - a1_{x2}a2_{y2})(a2_{x1} - a1_{x1}),$$
$$\det_x B = (b1_{y1}b2_{x1} - b1_{x1}b2_{y1})(b2_{x2} - b1_{x2}) - (b1_{y2}b2_{x2} - b1_{x2}b2_{y2})(b2_{x1} - b1_{x1}).$$

The computation requires only addition and multiplication of integer numbers. $|\det A|$ and $|\det B|$ are bounded by $8(CoordMax)^2$, $|\det_x A|$ and $|\det_x B|$ are bounded by $8(CoordMax)^3$, and the maximum absolute value of the integers involved is $\leq 64(CoordMax)^5$. Thus the function uses only constant amount of time and memory.

2. `bool` line_a_is_not_below_line_b_at_minus_infinity(I-line a, I-line b);

Returns `true` if and only if $f_a(x) \geq f_b(x)$ everywhere on some interval $(-\infty, t)$, where $t \in \mathbb{R}$ and $f_a(x), f_b(x)$ are the functions which graphs are I-lines a, b correspondingly. Note that this function defines a correct linear order relation on I-lines, in particular

either line_a_is_not_below_line_b_at_minus_infinity(a, b),
or line_a_is_not_below_line_b_at_minus_infinity(b, a)

and if both are true then a and b coincide.

In terms of the coordinates: Assume that $a_{x2} \geq a_{x1}$. The function returns `true` if and only if

$$(\det > 0)$$
$$\text{or}$$
$$\Big(\det = 0 \text{ and } (a_{y2} - a_{y1})b_{x1} + a_{y1}a_{x2} - a_{y2}a_{x1} \geq b_{y1}(a_{x2} - a_{x1})\Big),$$

where

$$\det = (a_{x2} - a_{x1})(b_{y2} - b_{y1}) - (a_{y2} - a_{y1})(b_{x2} - b_{x1}).$$

The computation requires only addition and multiplication of integer numbers. The maximum absolute value of the integers involved is $\leq 8(CoordMax)^2$. The function uses only constant amount of time and memory.

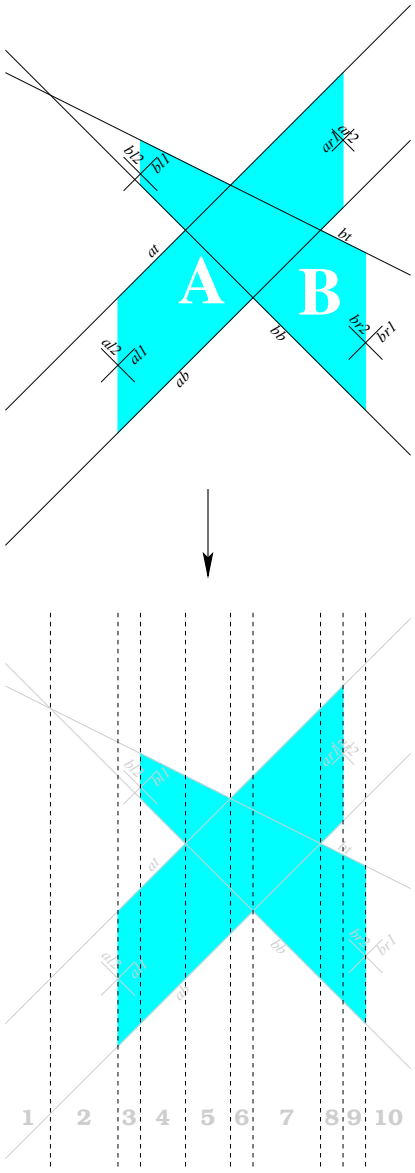

Fig. 7.

D. The Algorithm

Consider the following ≤ 10 points.
The 4 points which define the x coordinates of the bases of the input V-trapezoids:

$$(al1, al2),\ (ar1, ar2),\ (bl1, bl2),\ (br1, br2) \tag{1}$$

and the ≤ 6 points of the pairwise intersections of the sides of the input V-trapezoids:

$$(at, ab),\ (at, bb),\ (at, bt),\ (ab, bb),\ (ab, bt),\ (bb, bt). \tag{2}$$

For any two sides which are parallel or coincide skip the point.

By using function
"point_A_is_not_to_the_left_of_point_B(...)" order these points left to right. The vertical lines drawn through the points partition the plane into ≤ 11 vertical columns, see Figure 7. In the interior of each column the I-lines ab, at, bb and bt either do

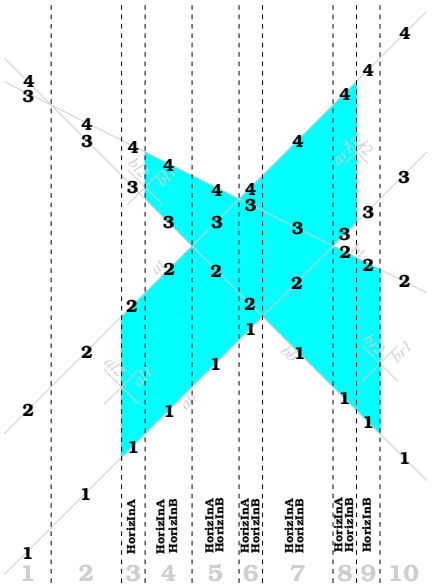

Fig. 8.

not intersect or coincide. So it is possible to order them vertically. The columns except for the first and the last ones (≤ 9) are split by ab, at, bb and bt into ≤ 27 trapezoids (and several unbounded pieces). These trapezoids are V-trapezoids. In fact: the x coordinates of their bases are defined by pairs of I-lines from sets (1) and (2) and their sides are segments of ab, at, bb or bt. Each of the newly created V-trapezoids either belongs to A or is disjoint with A and the same holds in respect to B. So it is possible to correctly assign tags "A", "B", "AB" to them. It remains to show how to do that.

By using function "line_a_is_not_below_line_b_at_minus_infinity(...)" order ab, at, bb and bt in the leftmost (unbounded from the left) column bottom to top. Call this ordered set of 4 elements S_1. Iterate through the columns left to right. Every time we pass a point from the set (2) make the corresponding transposition of two elements of S_i to form S_{i+1} in the $i+1$-th column (some extra care is needed for the degenerate cases when two or more points define the same vertical cut). Also keep track of whether or not we are between the bases of A and between the bases of B: For the leftmost column $\mathrm{HorizInA}_1 = \mathrm{HorizInB}_1 = false$ and these flags controllably change every time we pass a point from the set (1).

Finally, when the vertical order S_i of ab, at, bb and bt and the values of flags $\mathrm{HorizInA}_i$ and $\mathrm{HorizInB}_i$ in each column are known (Figure 8), assign the correct tags "A", "B" or "AB" to the newly created V-trapezoids by scanning each column bottom to top.

References

[1] P. Rickert. *Problems or Opportunities? Beyond the 90 nm Frontier*. Keynote, ICCAD 2004.

[2] P. Gelsinger. Keynote address to 41st DAC, 2004.

[3] L. Liebmann, A. Barish, Z. Baum, H. Bonges, S. Bukofsky, C. Fonseca, S. Halle, G. Northrop, S. Runyon, L. Sigal. *High performance circuit design for the RET-enabled 65-nm technology node*. Proc. SPIE Vol. 5379, 2004.

[4] T. Tugbawa, T. Park, D. Boning, T. Pan, P. Li, S. Hymes, T. Brown, L. Camiletti. *Modeling of Pattern Dependencies for Multi-Level Copper Chemical Mechanical Polishing Process*. CMP Symposium, MRS Spring Meeting, April 2001.

[5] M. Lavin, F.-L. Heng, G. Northrop. *Backend CAD Flows for Restrictive Design Rules*. Proc. ICCAD, 739-746, 2004.

[6] J. Sawicki. *DFM, Magic bullet or marketing hype?*. Proc. SPIE Vol. 5379, 2004.

[7] J. Ho, Y. Wang, Y. Hou, B.S. Lin, C. Yu, C.L. Ma, K. Wu. *DFM: a practical layout optimization procedure for the improved process window for an existing 90-nm product*. SPIE 2006.

[8] W.J. Poppe, A.R. Neureuther, L. Capodieci. *Platform for collaborative DFM*. SPIE 2006.

[9] K. Veelenturf. *The Road to better Reliability and Yield Embedded DFM Tools*. Proc. SIGDA DATE 2000, p. 67.

[10] L. Huang, M. Wong. *Optical Proximity Correction (OPC)-Friendly Maze Routing*. Proc. DAC 2004.

[11] Mark Lavin, Lars Liebmann *CAD computation for manufacturability: can we save VLSI technology from itself?* Proceedings of the 2002 IEEE/ACM international conference on Computer-aided design ICCAD '02, November 2002

[12] Puneet Gupta, Fook-Luen Heng *Design for manufacturing: Toward a systematic-variation aware timing methodology* Proceedings of the 41st annual conference on Design automation DAC, June 2004

[13] K. Mehlhorn, S.Naeher. *LEDA, A platform for combinatorial and geometric computing*. Cambridge University Press, 1999.

[14] *The GNU MP library*. http://gmplib.org/

[15] *Computational Geometry Algorithms Library*. http://www.cgal.org/

[16] C. Yap. *Towards exact geometric computation.* Comput. Geom. Theory Appl., 7(1):3-23, 1997.

[17] C. K. Yap and T. Dub. *The exact computation paradigm.* In D.-Z. Du and F. K. Hwang, editors, Computing in Euclidean Geometry, volume 4 of Lecture Notes Series on Computing, pages 452-492. World Scientific, Singapore, 2nd edition, 1995.

[18] Stefan Schirra. *Robustness and precision issues in geometric computation.* In rg-diger Sack and Jorge Urrutia, editors, Handbook of Computational Geometry, chapter 14, pages 597-632. Elsevier Science Publishers B.V. North-Holland, Amsterdam, 2000.

[19] C. Burnikel, R. Fleischer, K. Mehlhorn, S. Schirra *Efficient exact geometric computation made easy* Proceedings of the fifteenth annual symposium on Computational geometry SCG June 1999

[20] Herv Brnnimann, Ioannis Z. Emiris, Victor Y. Pan, Sylvain Pion *Computing exact geometric predicates using modular arithmetic with single precision.* Proceedings of the thirteenth annual symposium on Computational geometry SCG August 1997

[21] C. Burnikel, J. Knemann, K. Mehlhorn, S. Nher, S. Schirra, C. Uhrig *Exact geometric computation in LEDA* Proceedings of the eleventh annual symposium on Computational geometry SCG '95 September 1995

[22] C. M. Hoffmann. *The problems of accuracy and robustness in geometric computation.* IEEE Computer, 22(3):31-41, March 1989.

[23] C. Hoffmann. *Geometric and Solid Modeling.* Morgan-Kaufmann, San Mateo, CA, 1989.

[24] Kokichi Sugihara, *An Intersection Algorithm Based on Delaunay Triangulation*, IEEE Computer Graphics and Applications, v.12 n.2, p.59-67, March 1992

[25] Kokichi Sugihara, *Resolvable representation of polyhedra*, Proceedings on the second ACM symposium on Solid modeling and applications, p.127-135, May 19-21, 1993, Montreal, Quebec, Canada

[26] R. Banerjee, J.R. Rossignac. *Topologically exact evaluation of polyhedra defined in CSG with loose primitives.* Computer Graphics forum, Volume 15 (1996), number 4, pp. 205-217.

[27] David Dobkin, Leonidas Guibas, John Hershberger, Jack Snoeyink *An efficient algorithm for finding the CSG representation of a simple polygon.* ACM SIGGRAPH Computer Graphics , Proc. of the 15th annual conf. on Computer graphics and interactive techniques SIGGRAPH '88, Vol. 22 Issue 4 June 1988.