A PATTERN MATCHING ALGORITHM FOR VERIFICATION AND ANALYSIS OF VERY LARGE IC LAYOUTS

Mariusz Niewczas[*], *Wojciech Maly and Andrzej Strojwas*
Dept. of Electrical and Computer Engineering, Carnegie Mellon University,
5000 Forbes Ave., Pittsburgh PA 15213, fax: (412) 268 3204, email: mn@ece.cmu.edu
[*]On leave from Dept. of Electronics, Warsaw University of Technology, Warsaw, Poland.

ABSTRACT

We propose a simple, isometry invariant pattern matching algorithm for an effective data reduction useful in layout-related data processing of very complex IC designs. The repeatable geometrical features and attributes are stored in a pattern database. Original pattern instance, or its geometrical attributes, may be quickly regenerated based both on the information stored within the pattern and position of the pattern instance. We also show preliminary results of analysis of the state-of-the-art ICs which suggest that the diversity of patterns does not significantly increase with the increase of chip size.

1. INTRODUCTION

Verification of sub-half-micron IC's presents many challenges that have not been fully addressed by the current CAD tools. Complex physical and chemical phenomena involved in DUV lithography, the use of phase shifting masks and optical proximity corrections for technologies below quarter-micron cause the 3-dimensional topography of the manufactured IC to be very different from the one created by the simplistic superposition of lithographic masks. Thus, most of the design verification and analysis tasks, especially accurate circuit extraction, critical area/volume extraction and optical proximity correction, in deep submicron domain must use accurate mask to topography mapping as a starting point. Moreover, the enormous size/complexity of modern ICs require efficient hierarchical approaches to be employed for the purpose of handling the layout data.

Given the complexity of today's ICs and the increasing necessity to process the 3-D topography instead of mask layout, a data reduction technique is needed which is effective in terms of memory requirements and still allows CAD algorithms to operate on the data without a big performance penalty. Several IC verification tools (e.g., DRCs, circuit extractors, critical area extractors for yield prediction) have been developed in the past to handle the repetitive nature of VLSIC's by employing the design hierarchy. One of the main difficulties in practice is that the real-life hierarchy is often not clean; it includes overlapping cells and over-the-cell-routing wires. Moreover, most applications require accounting for interaction of cell instances with their neighborhood which complicates the use of design hierarchy even further. There are three currently published approaches to this problem which do not completely exclude each other. The first one relies on local flattening of the design hierarchy [1], [2]. The second one applies decomposition of cells into parts based on interactions of their instances with the surrounding primitives and other instances which "cleans" the hierarchy at the expense of an increase in the number of cells [3],[4]. Third approach adds some complexity to hierarchy by applying the inverse layout trees [5]. Although such solutions may work for some tasks and some lower layout levels (e.g., up to local interconnect layer), they break down for the metal interconnect layers in which the cell-based hierarchy is not followed at all. Processing of such layers, given their complexity, is a challenge. Therefore, the data reduction technique must have an option of adapting to the statistical pattern of mask features instead of being always forced into the framework of cell hierarchy.

This paper presents a pattern matching algorithm which is, in our opinion, a basic step to address the above problem. The matching is based on a signature which stores a shape template in a way which is both compact and allows for a quick regeneration of any shape instance or its attributes (e.g. bounding box, convex hull, area). We focus on this algorithm since we have observed that fragments of shapes on masks of interconnect layer repeat often even if the entire mask shapes do not. Such repeatability of geometrical features is an advantage, since thanks to the pattern matching we may store details of these features only once even if they are rotated or mirrored. The repeatable geometrical patterns (and all their repeatable attributes such as 3-D topography profiles) are stored in a pattern database. The geometrical CAD database uses instances of these patterns. This is analogous to the way instances of library cells are used in a hierarchical layout representation, but allows to account for "finer granularity" of repetitive patterns.

This paper is organized as follows. Sec. 2 introduces our canonical representation of shapes, the concept of contour equivalence and the previous work in the area of shape matching. Sec. 3 presents our pattern matching algorithm in general terms. Sec. 4 describes our approach to building libraries of patterns based on this algorithm and presents considerations specific to IC mask geometry. Sec.5 contains experimental results. Sec. 6 presents conclusions.

2. BACKGROUND

We begin with the discussion of contours, i.e., oriented borders of regions defined by simple polygons on 2D plane. These regions may correspond either to mask shapes or to some other entities which are used in physical CAD algorithms (for example, the interaction area in DRC, the device geometry, etc.). We assume that a region may be either an interior or an exterior of the contour, i.e., a solid or a hollow.

Definition 1. Let **L** be a simple polygon described by a sequence of vertices (V_1,...,V_n). By a **contour K** (Fig. 1) we will understand the simple polygon **L** with the attribute of orientation of sequence of its vertices. The sequence of vertices is oriented counter-clockwise if **L** represents a border of a hollow or clockwise if it represents the border of a solid.

The matching algorithm which we propose in this paper is isometry invariant (i.e., it recognizes as equivalent the contours which are translated, rotated and reflected). Moreover, the contour orientation is an attribute which is preserved under reflection. To be more precise, our matching algorithm checks for the equivalence of contours defined as follows.

Definition 2. Two contours $\mathbf{K}_1$ and $\mathbf{K}_2$ belong to the same **contour equivalence class** if there exists an isometric transformation which maps the polygon of contour $\mathbf{K}_1$ into the polygon of contour $\mathbf{K}_2$ and both contours have the same orientation. Contours belonging to the same equivalence class are called **equivalent**.

Published in Proc. of International Symposium on Physical Design, April 1998, Monterey, CA, pp. 129-134.

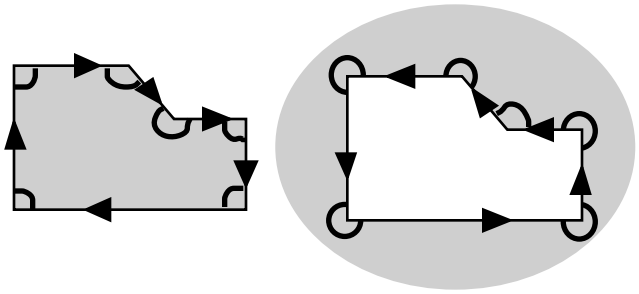

FIGURE 1. Illustration of Definition 1. Orientation defines that the interior is "to the right" of directed line segments of the respective contour. The arcs denote internal angles.

Many interesting pattern matching algorithms have been developed in the past [7],[8],[9],[10],[11]. As will be explained below, we base our pattern matching on vertex angles and edge segment lengths of contours. The angles and lengths are isometry invariants and have been used in various forms in more complicated algorithms performing inexact matching (e.g. [8],[11]).

3. PATTERN MATCHING ALGORITHM

We will now describe an algorithm for checking whether two given contours are equivalent in the sense of Definition 2. The algorithm is based on a concept of a contour signature, a sequence of numerical values which uniquely characterizes a contour equivalence class and which can be quickly constructed for a given contour.

3.1. The signature

Let us choose an arbitrary vertex **V** out of **n** vertices of a contour **K**. We construct our signature as a sequence of angles and edge lengths built by following the vertices along the contour. To describe the signature and to prove its uniqueness within equivalence classes, we will have to introduce two auxiliary concepts.

Definition 3. The **vertex sequence** of contour **K** starting in vertex **V** is a series of vertices ordered as they appear when the polygon of **K** is traversed in a given direction starting from **V**. We will call a vertex sequence a **direct** one if the order of vertices agrees with the orientation of contour **K** and we will call it an **indirect** (or **mirrored**) one in the opposite case. The sequence will be called **full** if it contains all **n** vertices of **K**.

Definition 4. For a given full vertex sequence $\mathbf{A}=\{\mathbf{V}_i\}_{i=0,..,n-1}$, by the **angle-segment sequence** of contour **K** we will understand the sequence:

$$\mathbf{S} = \{(\alpha_i, \mathbf{L_i})\}_{\ i=0,..,n-1}, \qquad (1)$$

where, α_i is a magnitude of internal angle of vertex $\mathbf{V}_i$ and $\mathbf{L}_i$ is the length of the line segment connecting $\mathbf{V}_i$ with the vertex $\mathbf{V}_{(i+1) \bmod n}$ (i.e., with the next vertex of **K**). **S** is **direct** when **A** is direct, and **S** is **indirect** (or **mirrored**) otherwise.

Observe that vertex sequence is a chain of points with specific locations on the plane whereas the angle-segment sequence is isometry invariant and thus is position-independent.

Fig. 2 illustrates the above concepts. The example sequences start at the same $\pi/4$ corner. Note that the inverted sequence of **K** is equal to the direct sequence of **K'**. This is because **K'** is a reflection of **K** and the sense of angles is changed, but contour orientation is preserved.

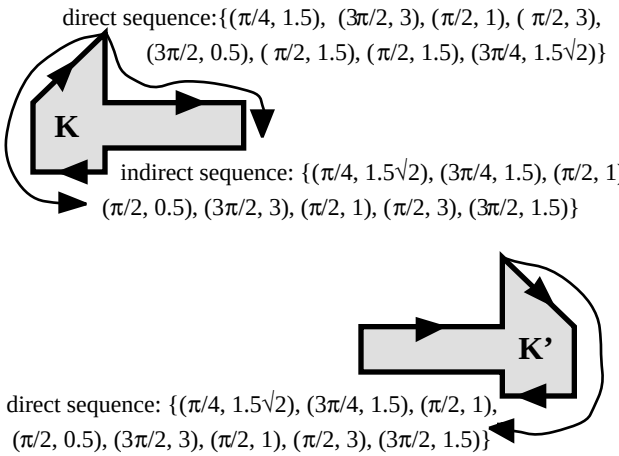

FIGURE 2. Example angle-segment sequences (1) for two equivalent contours: **K** and its reflection **K'**. Angles are in radians.

Observation. Let us consider a given contour **K** and let **S** be any of this contour's angle-segment sequences in a form given by (1). **S** uniquely identifies the contour equivalence class to which **K** belongs.

Proof. We have to show that: (a) only contours equivalent to **K** may have the given angle-segment sequence **S** and (b) the sequence **S** exists for all equivalent contours if it exists for any of them.

The proof of (b) is immediate since the angle-length sequence **S** is isometry invariant and each member of a contour equivalence class can be mapped by definition to another one by isometry.

To prove (a) let us consider any two contours **K** and **K'** for which the sequence **S** exists. Let $\mathbf{A} = \{\mathbf{V}_i\}_{i=0,..,n-1}$ and $\mathbf{A'}=\{\mathbf{V}_i'\}_{i=0,..,n-1}$ be the respective vertex sequences for the angle-length sequence **S** of **K** and **K'**. To prove that **K** and **K'** are equivalent, we have to show existence of isometry, which transforms **K** to **K'**. This can be done in a following way: let us consider the triangle $\mathbf{V}_0\mathbf{V}_1\mathbf{V}_2$ determined by the first three vertices of **A** and the triangle $\mathbf{V}_0'\mathbf{V}_1'\mathbf{V}_2'$ determined by the first three vertices of **A'**. Without any loss of generality we can assume that $\mathbf{V}_0$, $\mathbf{V}_1$ and $\mathbf{V}_2$ are not collinear - we merge such edge segments in all contours.

Since **A** and **A'** generate the same angle-segment sequence **S**, both triangles are equal (two sides and the angle between them are equal) which implies that there is an isometry **Z** which transforms triangle $\mathbf{V}_0\mathbf{V}_1\mathbf{V}_2$ to triangle $\mathbf{V}_0'\mathbf{V}_1'\mathbf{V}_2'$ so that:

$$\mathbf{V}_0' = \mathbf{Z}(\mathbf{V}_0),\ \mathbf{V}_1' = \mathbf{Z}(\mathbf{V}_1) \text{ and } \mathbf{V}_2' = \mathbf{Z}(\mathbf{V}_2). \qquad (2)$$

However, **Z** is also the isometry which makes **K** and **K'** equivalent since the fact that $\mathbf{V}_i' = \mathbf{Z}(\mathbf{V}_i)$ for i=3,..,n-1 can be explicitly deduced from the fact that **K** and **K'** have the same sequence **S** (we can build the next pairs of triangles isometric under **Z** traversing both contours concurrently). This is because both the edge lengths and the angle magnitudes are preserved under **Z**, which ends the proof.

Note that a contour **K** with **n** vertices may be represented by 2**n** full vertex sequences; a direct one and a mirrored one starting at each vertex. However, not all the angle-segment sequences based on these vertex sequences are necessarily different since the contour may have internal symmetry. For example, a rectangular contour has only two different angle-length sequences.

We base our contour signature on an angle-segment sequence. The idea is to construct the signature in such a way that we obtain the same distinct angle-segment sequence for any contour belong-

ing to a given equivalence class. The signature can be efficiently built if we choose the **lexicographically smallest sequence**. To find such a sequence for a given contour, we first compute any direct angle-segment sequence by choosing any random vertex as the starting point. Then, we may easily obtain any of other direct sequences for that contour by just changing the index of the starting point and "connecting" the end of the original sequence with its beginning. Thus, we can look up all sequences and efficiently find the smallest one. The binary predicate we have chosen for the lexicographical ordering treats angle magnitudes as more important. Only if the compared sequences have the same sub-sequence of angle magnitudes are the segment lengths compared. Theoretically, the lexicographical sort can be done in O(n) time [13]. In our case, the average number of vertices is small (usually in order of tens). Moreover, we can often take advantage of specific characteristics of the data set as explained later in Section 4.

Definition 5. The vertex at which a lexicographically smallest angle-segment sequence was starts will be called the **smallest vertex**.

To enable matching against mirrored contours we also have to find the lexicographically smallest indirect angle-segment sequence. Any indirect angle-segment sequence for a contour can be generated from a direct one by traversing it in the opposite direction. This allows us to a find minimal indirect sequence by applying the same lexicographical ordering procedure as outlined above.

Definition 6. The **contour signature** consists of the lexicographically smallest direct angle-segment sequence and lexicographically smallest indirect angle-segment sequence.

Observe that for the implementation purposes it is enough to store the direct sequence only, the smallest indirect sequence being identified by an index. Moreover, if a contour has an axial symmetry, then the minimal indirect angle-segment sequence is identical to the direct one. If the contour has an *k*-fold center of symmetry the signature consists of the same sequence repeating *k* times. For such contours the signature can be simplified. Also, squares, rectangles and trapezoids can obviously be tested for equivalence directly without any need for generation of the signature. We omit these issues in the following part of this section for the sake of clarity.

It should be also noted, that in some CAD applications (e.g. detailed circuit extraction) it is convenient to associate some properties with the edge segments or fragments of these edges. Such a case is often referred to as "edge coloring". Our signature and pattern matching algorithm can be easily made "color sensitive" by adding appropriate color sequences to the signature presented above.

3.2. Signature comparison

Let us assume that two contours **P** (pattern) and **C** (candidate) are given. **P** would be typically a pattern stored in a database and **C** would be a new contour. **P** and **C** are equivalent if and only if one of the following is true:

- minimal direct angle-segment sequence of **P** is equal to the minimal direct angle-segment sequence of **C**

or

- minimal indirect angle-segment sequence of **P** is equal to the minimal direct angle-segment sequence of **C**.

In this way, the problem of contour matching has been reduced to the problem of string processing. First, the strings are created (by finding minimal sequences as described above in Sec. 3.1) and then, the strings are subject to comparison. Both of these procedures have the complexity O(n). It should be noted however, that maximal value of n (i.e., the greatest number of vertices in a contour for a given design to be processed) does not change significantly with an increase of complexity of the design. Hence, the cost of matching two contours can be considered to be constant.

The need to compare against both direct and indirect sequences is a consequence of the fact that although each angle-segment sequence is isometry invariant, the reflection of a shape causes any particular sequence to become indirect if it was direct before reflection and vice versa (compare Fig.2). Note, that if we test two contours for matching, we have to create a full signature for one of them only (**P** in our case), the other one may be just coded with the minimal direct sequence.

3.3. Chain matching

One important variation of the ideas presented above is the case when pattern matching has to be done in search for repetitive open chains of vertices. Such a capability may have many applications. We will give two examples. First, the chain may describe a path object, (e.g., "the wire" in CIF format). This allows for improved performance of contour matching on metal interconnect layers. Second, the chain may describe features of a border of a mask shape (e.g. for optical proximity correction). In this case, only the vertices separated by less than a certain threshold distance are considered members of a chain. This is a concept which we use in topography simulation [6].

The chain matching procedure differs from contour matching only at the stage of signature construction. An open chain is represented by exactly two vertex sequences: a direct one and an indirect one. Thus, the smallest direct angle-segment sequence starts at the beginning of the chain and the smallest indirect one starts at the and of the chain. Moreover, the angle segment sequences contain one more segment length value than the angle magnitude value. Having this in mind, we can say that signature matching for chains is analogous to the one described just above for contours.

4. APPLICATION

Using the concept of pattern signature and the matching algorithm introduced above, we organize a pattern database (pDB) and a database of contour instances (iDB) for a given data set. The issues related to these two databases are discussed in Sections 4.1 and 4.2.

Our pattern matching algorithm is formulated for arbitrary isometries and for polygons with arbitrary angles. However, in order to increase efficiency of our implementation, we limited possible rotation angles. We also created a special variant of the signature for regular geometries. We discuss these two issues in Sec. 4.3 and Sec. 4.4.

4.1. Pattern database building algorithm

The pDB stores contour signatures and may also keep some other useful pattern-related data (such as 3D topography profiles for example). The pDB is based on a set of binary trees, roots of which are stored in a hash table. Each binary tree is sorted according to the lexicographical order of signatures. The hashing function is based on the number of contour vertices and dimensions of the bounding box. Such sorting allows for efficient addition of a new pattern **C** to pDB. In other words, we have to perform signature comparison for **C** only against those patterns **P** from the pDB which satisfy necessary conditions for contour equivalence. In the case of our implementation, this means that **C** and **P** have the same value of the hashing function and the actual signature comparison algorithm is invoked for a very small number of patterns only. Note, that the hashing function can be adjusted to the statistical properties of geometrical database characteristic of a given CAD environment and design style. To further improve efficiency of building the pDB, we construct only the direct minimal angle-segment sequence of the "candidate" **C**, not the entire signature (com-

pare Sec. 3.2). Only after **C** is found not to match any contour from pDB, we look for the indirect minimal sequence and thus, we finish constructing the signature to add such a unique contour **C** to pDB. In the opposite case, when **C** is found to match a pattern **P** already existing in pDB, the angle-segment sequence of **C** is discarded but we know if **C** is matched against direct or indirect sequence of **P**. The transformation from **P** to **C** includes reflection in the latter case only. Once we have the pDB entry corresponding to **C**, we can add this contour instance to iDB. We have to determine the parameters of transformation leading from pDB entry to **C**. This is analogous to providing a transformation code for a cell instance in hierarchical designs. As explained in Sec. 4.3, we decided to limit possible rotation angles to the multiples of $\pi/2$.

We assume a convention that the identity transformation of a pDB entry to a contour instance will result in a contour with minimal vertex situated at point (0,0) on the plane. In addition, we specify orientation of what we call the **first vector of the contour**: a vector connecting minimal vertex with the next vertex in the direct vertex sequence. This vector, for the identity transformation makes an angle less than $\pi/2$ with the X axis of the coordinate system.

With this convention in mind, one can easily find translation, and rotation parameters for any contour **C** for which its minimal point is known. Moreover, the reflection must be accounted for if it was detected during the signature matching.

4.2. Pattern instance database and contour recreation

Due to specific needs of CAD applications, the elements of iDB are geographically (e.g., using quad-tree representation) and hierarchically ordered. Note, that iDB may be, when applicable, an existing hierarchical layout database with pattern instances representing selected layout primitives. Alternatively, it may be a totally new database when the hierarchy does not work which is, in general, the case for most metal interconnect layers.

Each contour instance, an entry in iDB, must contain the following minimal information: the transformation code, coordinates of its minimal point (called the **origin**) and reference to the respective pattern from pDB. The rest of contour instance data, such as the coordinates of vertices and the bounding box may be regenerated based on this minimal data and the information contained in the appropriate pattern from pDB. In order to facilitate fast vicinity search and region queries in the iDB, we store the origin point implicitly: we represent the center of gravity of the pattern instance's bounding box (COG). The offset from this COG to origin is stored with the pattern where the dimensions of the bounding box are also kept. This allows for calculation of the bounding box very efficiently on the fly (four additions only) during vicinity search.

Based on the signature, the polygon describing the contour instance **cI** may be easily recreated using following algorithm:

1. Find length of the first vector based on the signature of **cI**.
2. if(the transformation of **cI** does not involve reflection **){**
  3a. Position the first vector at the origin of **cI**, based on transf. code of **cI**, so that beginning of the vector is in origin of **cI**.
  3b. Traversing the minimal direct sequence in signature, starting from the second vertex, generate each next vector from the previous by setting the length and rotating the vector according to corresponding segment length and angle in signature.
}else{ /*Case of indirect isometry*/
  4a. Place the first vector at the origin of **cI**, based on the transf. code of **cI**, so that the end of the vector is in origin of **cI**.
  4b. Traversing minimal direct sequence in the signature starting at the first vertex and then proceeding backwards from the last vertex, generate each next vector the from previous one by setting length and rotating it according to the corresponding segment length and angle in signature.
}

This procedure is illustrated in Fig. 3. It is worth noting that this can be also used to constructively prove the uniqueness of signature discussed in Sec. 3.1.

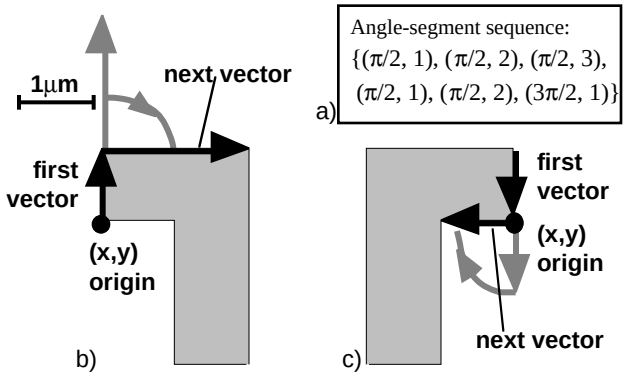

FIGURE 3. Contour recreation: a) the minimal angle-segment sequence, b) for the case of a contour instance which is not mirrored (steps 3a and 3b of the procedure), c) the case of a contour instance which is mirrored (steps 4a and 4b).

Such a pattern-based organization of the geometrical database reduces the memory requirements (in comparison with the databases which do not account for repetitive patterns). Moreover, it allows for adaptive and selective trade-off between memory usage and execution time. For example, if the coordinates of vertices of some contour instances are needed often, they can be stored together with these instances and later, when they are no longer needed, they may be discarded.

4.3. Limitation of transformations of interest

In our current implementation, we decided to limit (Fig. 4) the possible contour instance transformations to translations, reflection with respect to Y axis (traditionally called mirror X or MX), and rotations by an angle which is a multiple of $\pi/2$. This is because all coordinates are snapped to a grid. Hence, a rotated polygon may in general have slightly different edge lengths and angle magnitudes than before rotation. To account for this effect, we would have to use a much less efficient inexact matching techniques. Moreover, the entire concept of contour equivalence would become fuzzy. Within our restricted transformations, a diagonal line is transformed to another diagonal line and a line parallel to grid is transformed to another one parallel to grid. Restriction to eight possible transformations has an additional advantage: it allows for storing the transformation code in a 3-bit long field and speeding up rotation processing due to very simple transformation matrices which include only 0s and ±1s.

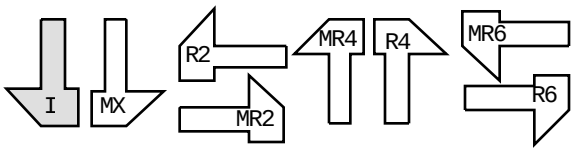

FIGURE 4. The transformations of contour instances to which we restrict our implementation. I - identity, MX - mirror wrt Y axis, R2 - clockwise rotation by $\pi/2$, R4 - clockwise rotation by π, R6 - clockwise rotation by $3\pi/2$, MR2, MR4, MR6 are compositions of MX and R2, R4, R6, respectively.

4.4. The case of $\pi/4$ geometry

The vast majority of layout entities are based on either Manhattan geometry or $\pi/4$ geometry (i.e., Manhattan lines and diagonal lines on a rectangular grid). We have developed a variant of the matching algorithm implementation to take advantage of this fact.

As shown in Figure 5, we can use a very limited number of angle codes to represent angle magnitudes for the $\pi/4$ geometry. Now,

the contour signature can be stored in a more efficient way since the contour codes require only 3 bits. The angle sequence is stored separately from the edge length sequence in this variant of implementation. Moreover, the contour recreation (Figure 3) can be done efficiently due to the use of simple look-up tables based on vector direction codes instead of using the rotation matrices.

The angle code 4 (vertex between two collinear segments) is allowed only in case of colored contours in the point where color changes on a segment, otherwise collinear edge segments are merged in order not to cause ambiguity.

This specific way of handling the $\pi/4$ geometry provides both a significant speedup and reduction of memory usage. This is also very easy to integrate with the general geometry processing code and data structures thanks to the object oriented implementation.

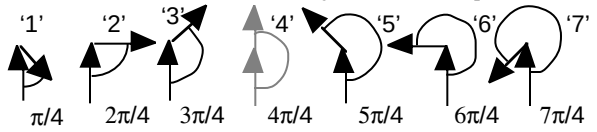

FIGURE 5. The angle codes allowed for $\pi/4$ geometry.

5. EXPERIMENTAL RESULTS

5.1. Repeatability of contours

The goal of our prototype implementation was to verify the concepts of pattern matching and to gain insight into how often patterns tend to repeat in industrial IC designs. We built interfaces between our prototype and two commercial products: the Cadence DFII Opus database and the SiCat (AISS/PDF Solutions) mask engine [12].

Performance of our prototype was about 700 patterns / second consistently on Spark Ultra 5 and on IBM PowerSeries 850 workstations. The examples are shown in Table 1. They are representative for what we obtained after processing of several industrial designs done for three different submicron and deep-submicron technologies and in different design styles (from approx. 15 000 transistors to 70 000 transistors in a flat mode and up to 1 000 000 transistors in a hierarchical mode). First four examples in the Table 1 illustrate repeatability of contours on a large region of polysilicon mask - the designs were flattened and all overlapping or abutting polygons were merged. Next two examples show the same thing for the active region and metal 1 mask. Part of the repeatability for such flat layouts may be attributed to the repetition of cell instances in the design hierarchy, but the result of hierarchical processing demonstrates that the layout features repeat quite often in the hierarchical layout database as well. In this case, we traversed the design hierarchy visiting each cell kind only once and again, merging all overlapping and abutting shapes, but without any local flattening of instances within the cell.

	# of contours	# of different patterns	Mean # of vertices in pattern	Data description
1	30507	1366	11.4	flat, Poly, cache tags+ctrl. logic
2	6802	1057	19.2	flat, Poly, full-custom rand logic
3	71503	2723	11.5	flat, Poly, full-custom rand logic
4	15545	4755	15.9	flat, Poly, std cell random logic
5	8219	246	15.5	flat, Met1, full-custom rand logic
6	23585	126	5.6	flat, active, full-custom rand logic
7	45355	3421	10.9	hierarchical, Poly, semi-custom integer processing unit
8	162699	2339	11.8	hierarchical, Poly, 16bit CISC CPU (standard cell style)

TABLE 1: Repetition of contour patterns in example designs.

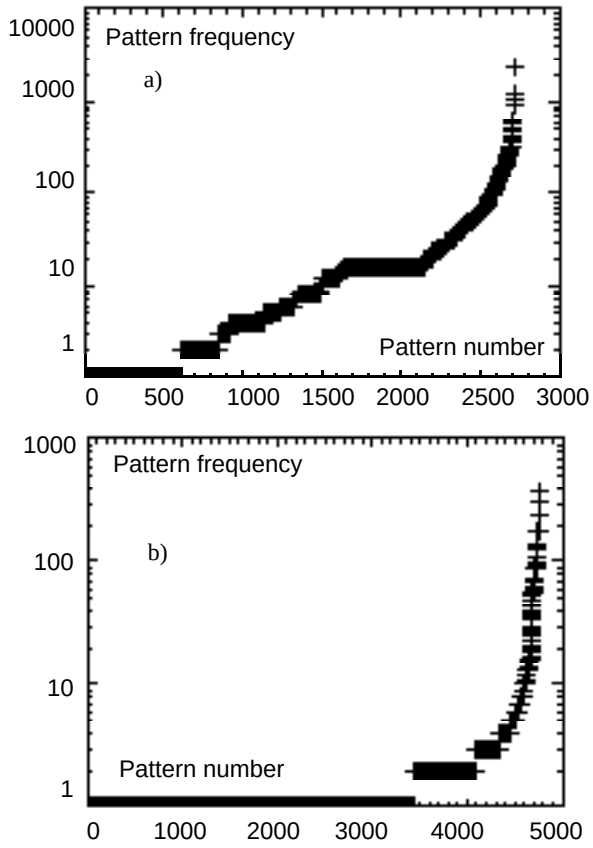

FIGURE 6. The frequency of repetition of contour patterns for polysilicon masks of two different designs (#3 and #4 in Table 1).

Figure 6 shows the repetition characteristics for two different designs. First one (Figure 6a) is an integer processing unit, which includes iterative logic. Thus, the characteristics shows repetition of many patterns exactly the same number of times. This is a feature which can be often directly deduced from the regularity of the design structure. However, repeatability of patterns can be seen even for logic layouts which seem not to contain such a regularity. The typical pattern repetition characteristics is similar to the one shown in Figure 6b where we show the worst case we found in our experiments. There is quite a few contours which do not repeat.

We noticed that for some patterns, significant portion of contour instances (10 - 25%), may be a reflection of the original pattern. In the above case of contours on the poly mask of a standard cell based design, reflection happens much more often than rotations. Note however, that there are many patterns for which we will never need reflections due to their symmetry. In the case of chain instances (see Section 3.3), which are features with “finer granularity”, we noticed that all kinds of transformations for chain instances are usually present. Due to space limitations, we do not show here the results we obtained for such chains. They are presented in [6]. Generally, for maximal vertex spacing of 2µm, there were only several hundred different vertex chains in each of the examples we analyzed. Our experiments have shown, that accounting for rotations and reflections of contours is an important feature of the matching algorithm.

5.2. Memory consumption

Our prototype representation of π/4 geometry pattern with **n** vertices requires 4**n** bytes for edge lengths (a coordinate is represented by 4 bytes), **n** bytes for angle lengths (this could actually be **n**/2 but we chose not to pack two angle codes into one byte for simplicity), 8 bytes for both bounding box dimensions, and 8 bytes for the vector coding the offset from COG to first vertex of the pattern and 2 bytes for coding the number of vertices - total 5**n** + 18 bytes. The memory required to represent pattern instances does not depend from **n** and is: 8 bytes for coordinate of COG, 1 byte for the transformation code, and 4 bytes for the pattern reference - total of 13 bytes.

The memory needed for representation of a polygon is roughly 8**n** for coordinates of vertices, 8 bytes for both bounding box dimensions and 2 bytes for the number of vertices - total of 8**n** +10. For the square, rectangle and trapezoid this cost is smaller: 12, 16 and 24 bytes respectively.

	Memory consumption in bytes					
	polygon			π/4 pattern +instance		
number of instances	1	2	10	1	2	10
3 vertices	34	68	340	46	59	163
4 vertices	16	32	160	51	64	168
5 vertices	50	100	500	56	69	173
6 vertices	58	116	580	61	74	178
7 vertices	66	132	660	66	79	183
8 vertices	74	148	740	71	84	188
9 vertices	82	164	820	76	89	193

TABLE 2: Memory usage for direct representation of polygon layout primitives and for our method.

The above table shows, that it is possible to spare a lot of memory if the contours do repeat and the average number of vertices is larger than 4 (compare Table 1 for mean numbers of vertices). The case of 4 vertices requires special treatment - in such case we can still use the patterns, but we use the bounding box and simple object classification instead of explicit signatures.

6. CONCLUSION

We presented an algorithm for isometry invariant pattern matching tailored especially for processing of geometrical shapes present in IC designs. The algorithm is capable of handling polygons with arbitrary angles, but has also a faster version making use of specific features of data sets in which all angles are multiples of π/4. Due to practical considerations, we restricted the allowed contour transformations by limiting the number of possible rotation angles. Based on the pattern matching algorithm, we presented an approach for construction of geometrical database using repetitive patterns stored in a pattern library.

Our data reduction technique seems to be well suited for very large layouts since the diversity of patterns does not significantly increase with the increase of chip size. Moreover, the signature-based pattern matching enables the trade-offs between execution speed and memory usage. Time required to recreate a contour based on the information contained in the signature does not depend upon the chip size. It is also worth noting that many frequent operations that require quad-tree search can be performed with very small performance penalty if pattern instances were used as layout primitives instead of polygons or paths. Recreation of a bounding box from a pattern instance requires only four additions. This is an example of how our method can provide significant memory usage reduction without an unacceptable performance degradation.

The patterns and pattern attributes (e.g., 3D topography profile) accumulated in a library may be used for more than one design targeted at a given manufacturing line. Moreover, even for multiple analyses for a given design, the building of pattern library and CAD database can be, in principle, carried out only once.

An important feature of our approach, which we are going to exploit in the future, is that it allows to enhance hierarchical representation which is important in case of layers such as metal interconnect which do not follow the design cell hierarchy. This, of course, does not exclude the use of the cell hierarchy for some set of design layers but gives much more freedom in efficient organization of design data. There are many applications, such as e.g. optical proximity correction, for which we can assume that the interactions between the mask shapes can be confined to a single layer or to the two adjacent layers. Our algorithm has already proven useful for chip-scale 3-D topography synthesis [6] and is now a basis for thorough investigation of statistical properties of deep-submicron industrial designs.

7. ACKNOWLEDGMENTS

We would like to thank Pranab K. Nag and Charles Ouyang of Carnegie Mellon University for their helpful discussions. We are also grateful to Thomas Waas and Hendrich Schneider of AISS, Munich, Germany and to Dennis Ciplickas of PDF Solutions, San Jose CA for their feedback and for the SiCat mask processing software. This work was supported in part by the SRC under contract DC-068.

8. REFERENCES

[1] T. Whitney, “A Hierarchical Design Analysis Front End”, in Proc. of First Int. Conf VLSI, Aug. 1981, pp. 217-225
[2] G. S. Taylor and J. K. Ousterhout, “Magics’s Incremental Design Rule Checker”, Proc. of 21stACM/IEEE Design Automation Conf., pp. 160-165, 1984.
[3] D.M.H. Walker, C. Kellen, D. M. Svoboda and A. Strojwas, “The CDB/HCDB Semiconductor Wafer representation Server”, IEEE Trans. on CAD, Vol CAD-12, No.2, pp.283-295, Feb. 1993.
[4] P. K. Nag and W. Maly, “Hierarchical Extraction of Critical Area for Shorts in Very Large ICs”, IEEE Intl. Workshop on Defect and Fault Tolerance in VLSI Systems, pp. 19-27, Nov 1995
[5] N. Hedenstierna and K. O. Jeppson, “The Halo Algorithm - An Algorithm for Hierarchical Design of Rule Checking of VLSI Circuits”, IEEE Trans. on CAD, Vol. CAD-12, No. 2. pp 265-272, Feb. 1993
[6] Mariusz Niewczas, Xiaolei Li, Andrzej Strojwas and Wojciech Maly, “Chip scale 3-D Topography Synthesis”, SPIE Optical Microlithography Conference, Feb 1998.
[7] Neg-Chung Hu, Kuo-Kan Yu, Yung-Li Hsu, “Two-dimensional shape recognition using oriented -polar representation”, Optical Engineering, Vol. 36, No. 10, Oct. 1997
[8] L. Davis, “Shape Matching Using Relaxation Techniques”, IEEE Trans. on Pattern Analysis and Machine Intelligence, Vol. PAMI-1, No. 1, Jan. 1979
[9] T. Pavlidis, “Algorithms for Shape Analysis of Contours and Waveforms”, IEEE Trans. on Pattern Analysis and Machine Intelligence, Vol. PAMI-2, No. 4, July 1980
[10]B. Bhanu and O. Faugeras, “Shape Matching of Two-Dimensional Objects”, IEEE Trans. on Pattern Analysis and Machine Intelligence, Vol. PAMI-6, No. 2, March 1984
[11] S. Z. Li, “Matching: Invariant to Translations, Rotations and Scale Changes”, Pattern Recognition , Vol. 25, No. 6, pp. 583-594 (1992)
[12] information about SiCat can be found at http://www.aiss.com
[13]A. Aho, J. Hopcroft and J. Ulman, “The design and Analysis of Computer Algorithms”, Addison-Weseley, 1974