Simulation Standard

Connecting TCAD To Tapeout

A Journal for CAD/CAE Engineers

Savage Enhanced with Recognition and Reporting of Hierarchical Structure of Errors

Introduction

This article describes a method of reporting DRC errors implemented in **Savage,** applicable to multi-million transistor layouts. The method of hierarchical information inheritance is a perspective approach in an extension of capabilities of flat DRC systems. This technique makes it possible to report hierarchical errors in ordinary flat DRC systems.

Advantages of Hierarchical Error Reports

It is known that fully hierarchical DRC systems compare favorably with flat DRC systems because of some of their capabilities flat systems do not have. The higher performance on hierarchical designs and the more compact and informative error report are probably the two most important features of hierarchical systems. Of course, the former is due to their very nature. However, it is possible to implement the latter on the basis of flat DRC systems provided the input data is hierarchical. The main benefits of hierarchical error report are as follows.

First, it allows merging of multiple replications of the same error in hierarchical design, so the amount of errors reported is close to the amount of layout corrections need to be carried out in order to get rid of those errors. It is worth mentioning that substantial reduction of amount of DRC errors detected yields noticeable gain in performance in flat DRC systems with hierarchical error report capability.

Second, each DRC error detected is filled with information on its hierarchical nature in addition to geometrical information. Being included in description of particular error this information helps designer to figure out the reason of the error and the method of its correction (see Figure 1).

Third, unlike flat DRC errors, hierarchical ones are not mixed in one huge melee but distributed over project hierarchy in accordance with their origin. The information on error distribution in the hierarchy tree aims to reveal and localize erroneous parts of project.

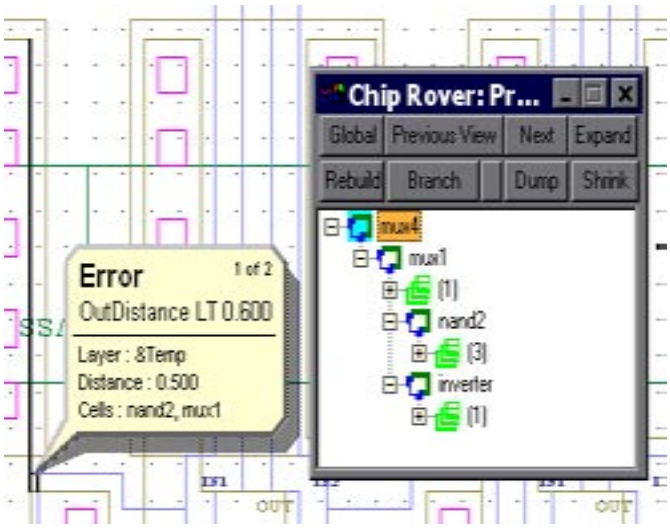

Figure 1. Hierarchical error display during a flat DRC run in **Savage**.

Implementation Issues

Hierarchy Information Inheritance

The implementation of hierarchical error report in flat DRC system is based on hierarchy inheritance technique, which is described below. In order to be processed by flat DRC the input layout should be flattened. During the flattening process every polygon of the result is provided with reference to the instance tree of initial hierarchical layout [1]. The reference (called hierarchical reference) points to the instance this object belongs to in original design. Such objects are referred

Continued on page 2....

INSIDE

SILVACO INTERNATIONAL

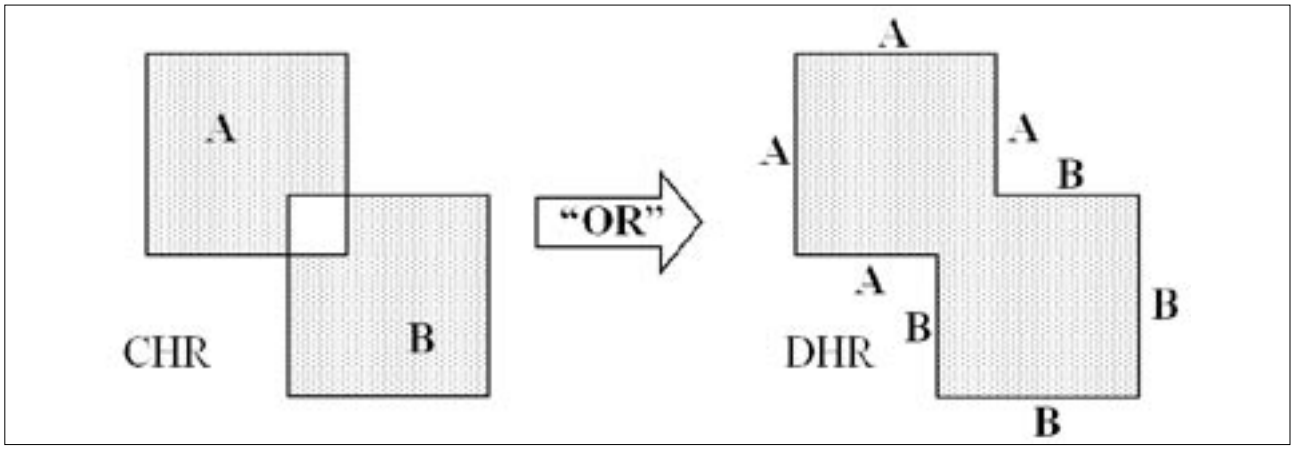

Figure 2. Hierarchical reference distribution.

to as objects with compact hierarchical reference (CHR). Objects from the same instance have equal hierarchical references. In the simplest case we perform only metric operations (CHECK group) on flat layout with hierarchical references. Every time the DRC algorithm finds an error it can obtain all necessary hierarchical information about this error from hierarchical references of polygons involved (see "Virtual hierarchy tree" section). We assume here that any metric operation in DRC checks distances between edges of polygons. Hence, any error detected has the form of two segments that lie on wrong distance from each other. The situation with layer generation operations is more complex. The main idea here is that any object in generated layer is some transformation and/or combination of objects in source layers. Hence, it should inherit hierarchical references from its parents. Operations of group SELECT are easy to deal with because they perform only selecting of objects from incoming layers. From some point of view, they do not generate new objects. Thus, objects in generated layers receive the hierarchical references from their prototypes in straightforward way. At the same time, operations of LOGIC group are very likely to generate new objects by combining polygons from source layers. In case when the new polygon is a combination of polygons with identical hierarchical references the resulting polygon is assigned the same hierarchical reference. The most interesting case takes place when the new polygon represents a result of combination of polygons with different hierarchical references. While it has no definite prototype in source layers, every edge is an image of certain edge of some source polygon. It is either exact copy or part of some source edge. It allows us to apply reference distribution technique. The reference distribution means that hierarchical reference in this case are distributed over the edges of resulting polygon. Each edge of such polygon has its own hierarchical reference inherited from the corresponding source edge (see Figure 2). We assume here that edges of polygon with compact hierarchical references have the same references as the polygon. Such polygons are referred to as polygons with distributed hierarchical reference (DHR). When a polygon with distributed hierarchical reference participates in DRC error detected by some metric operation the hierarchy information for this error is taken from that edge which represents error segment.

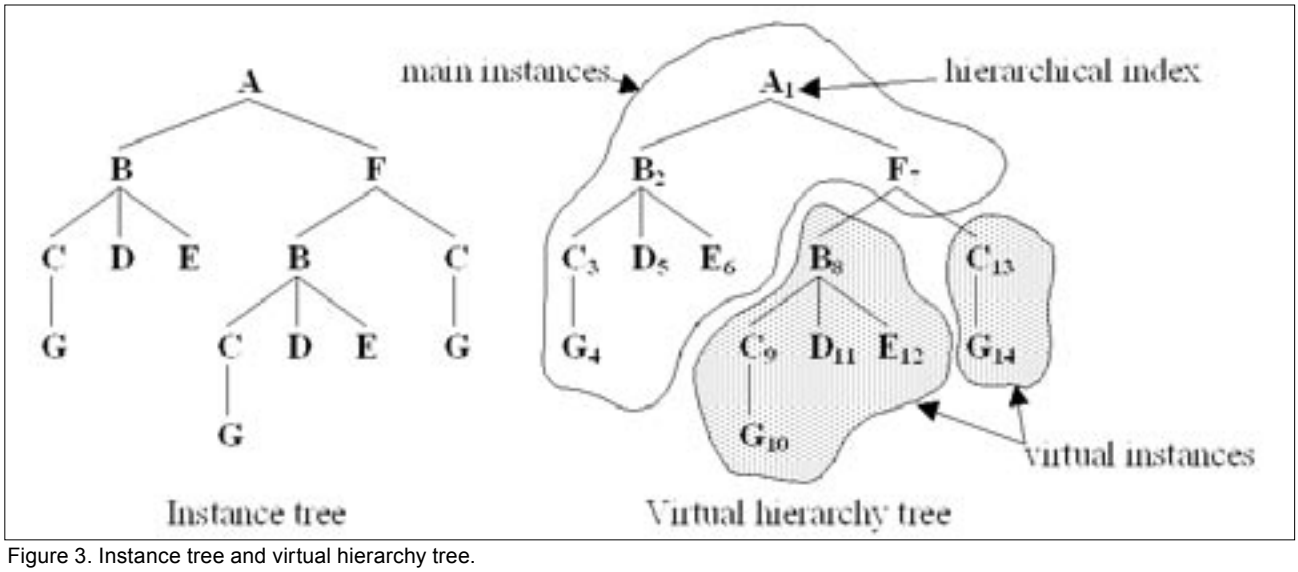

Figure 3. Instance tree and virtual hierarchy tree.

Consequent layer generation operations of DRC script can mix objects with compact and distributed hierarchical references in any combination. In general, it results in growth of number of polygons with DHR. However, at some point of DRC script execution it may happen that some polygon with DHR has equal hierarchical references on all of its edges. Provided this situation is easy to detect, it makes sense to perform the reference compacting replacing all distributed hierarchical references on the edges with single reference associated with the whole polygon. It turns polygons with DHR into polygons with CHR.

Virtual Hierarchy Tree

Our implementation of hierarchical error report in flat DRC system is based on special data structure, which is called virtual hierarchy tree. To anticipate a little, references to nodes of this tree play the role of hierarchical references described above. To define the virtual hierarchy tree for given project let's first consider the well known instance tree associated with the project [1] under assumption that the project has only one top cell (top cell is a cell that is not instanced in any other cell). For each cell of the project we choose one of its instances in the tree and mark it as main instance of the cell. All other instances are marked as virtual. Care should be taken to obey the following rule: all direct or indirect parents of main instance in the tree must be main instances. Such marks could be easily placed by means of pre-order tree traversal algorithm [2].

Each node of the tree in question (no matter main or virtual) is assigned a unique integer number - hierarchical index. Steps should be taken to ensure such arrangement of hierarchical indices as any given index could be efficiently found in the tree. The pre-order tree traversal algorithm mentioned above defines acceptable arrangement as well. The corresponding index search technique is well known. An instance tree with placed main/virtual marks and hierarchical indices is called virtual hierarchy tree (see Figure 3). During the flattening process hierarchical indices from virtual hierarchy tree are copied to polygons of corresponding instances of layout. These indices represent hierarchical references described in section "Hierarchy info inheritance". Each DRC error inherits hierarchical indices from corresponding edges of polygons involved. Section "Error classification and dispatching" describes what happens with DRC errors next.

Error Classification and Dispatching

Although all errors generated by metric operations in DRC are similar from geometrical point of view and consist of two segments, they may differ in their hierarchical nature. We distinguish two general classes of hierarchical errors - proper and cross errors. Error is defined as proper error if it occurs between two polygons that belong to the same instance of the same cell. And error is defined as cross error if polygons involved belong to different cells or to different instances of the same cell.

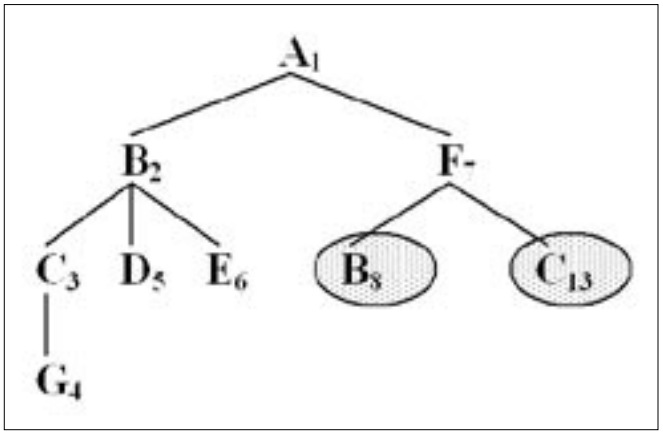

Figure 4. Compact virtual hierarchy tree

As our main purpose is to provide the most informative error report, we perform error dispatching during DRC script execution. The purpose of error dispatching is to find for each particular DRC error the most natural place in project hierarchy where this error can be reported. In plain words, it finds the cell that is most likely to contain the reason of the error and relates the error to this cell. Proper errors have equal hierarchical references on both segments and are dispatched in straightforward way: the proper error is reported in the cell it refers to. The reason of proper error in most cases is incorrect placement of primitives of the cell it is dispatched to. Cross errors are dispatched in accordance with the nearest common parent rule. It means that the root instance of minimal subtree of virtual hierarchy tree is found that contains both hierarchical references of the cross error. This instance is called the nearest common parent and the error is dispatched to corresponding cell. It may happen that nearest common parent for some cross error coincides with one of instances hierarchical references of this error point to. In this case we call this cross error the semi-proper error. Otherwise it is the pure cross error. The plausible reason of a semi-proper error is incorrect placement of instance in relation to primitives of its parent cell. And the likely reason of a pure cross error is incorrect relative placement of different instances in their parent cell.

Regardless of class of hierarchical error the following simple algorithm can perform the dispatching. Every time metric operation reports an error the algorithm starts to search both hierarchical references of the error in the virtual hierarchy tree simultaneously. The search begins at the root and skips from parent to child nodes building two paths from the root to the instances with desired hierarchical references. Either of two following events causes the algorithm to stop the search: the paths to the references diverge or at least one of the references is found. The error is dispatched to the cell at which instance the algorithm stopped. This instance is the nearest common parent for this error.

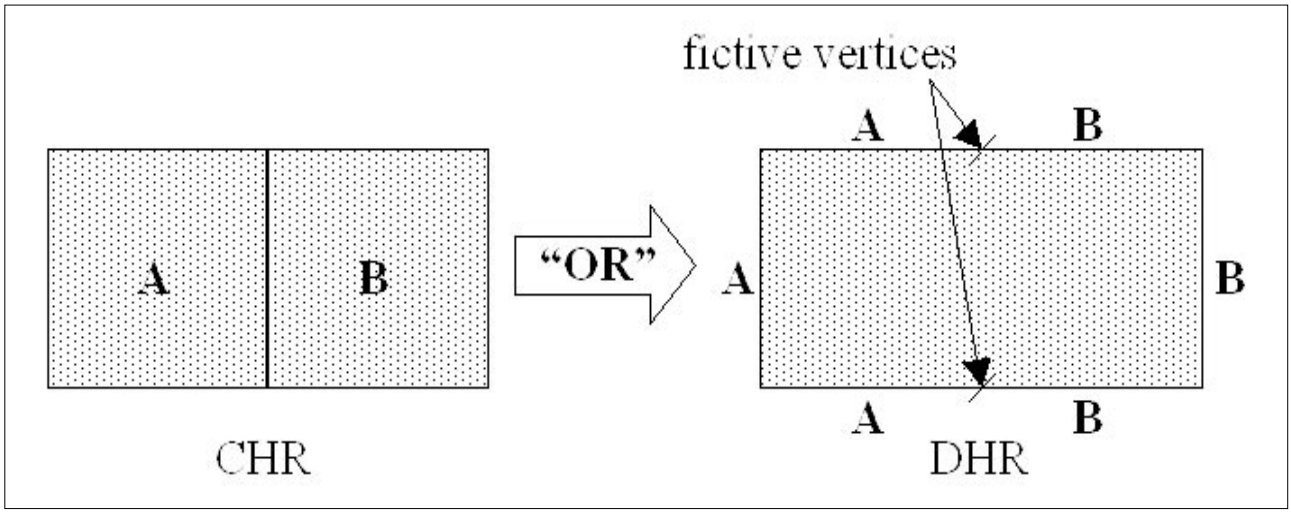

Figure 5. Fictive vertices

In order to eliminate multiple replications of the same error detected in different places of flat layout we just have to check whether the nearest common parent is represented by virtual or by main instance. If it proves to be a virtual one then the error should not be reported at all. It results in DRC errors dispatching to a cell only through its main instance in virtual hierarchy tree. Moreover, there is no necessity to build and keep in memory whole tree. In accordance with virtual hierarchy tree definition all direct and indirect children of virtual instance are virtual instances. It means that all virtual instances in the tree constitute a number of virtual subtrees. Keeping in mind our dispatching algorithm it is easy to see that it can get along with only a root instance of each virtual subtree. That's why the tree is referred to as the virtual hierarchy tree. No doubt, it takes much less memory to store virtual hierarchy tree instead of whole instance tree. However, note that hierarchical indices must be assigned under assumption that all instances from instance tree present in virtual hierarchy tree (see Figure 3 and Figure 4).

Related Overhead

Although the approach described above represents very substantial improvement for any flat DRC system, it has a number of more or less obvious drawbacks. First, it is necessary to build, store and maintain virtual hierarchy tree and hierarchical references, which consumes some computing time. Second, it requires the data structures of polygon and segment to be extended to contain hierarchical reference for polygons with CHR and DHR respectively, which increases amount of memory occupied by these structures. Third, the hierarchy info inheritance technique may lead to appearance of so called fictive vertices in polygons. Sometimes, when some layer generation operation is performed two edges of different polygons lie on the same line and should be merged into single edge of resultant polygon. But if these edges have different hierarchical references the merging should be avoided to preserve inherited hierarchical info. The vertex that separates such edges is called the fictive vertex (see Figure 5). In DRC algorithms based on scan-line technique [3, 4] fictive vertices sometimes are replaced with fictive edges of zero length.

Still another problem is layer generation operations of RESIZE group. These operations transform input polygons into new ones which may differ very substantially from their originals. It complicates the implementation of hierarchy info inheritance in such operations, especially when RESIZE operations with some non-trivial join styles are used.

Conclusion

Although this approach cannot turn a flat DRC system into fully hierarchical one, it enhances the effectiveness of DRC if flat system must be used. On small and medium size designs a flat DRC system with hierarchical error report feature can successfully compete with any hierarchical DRC system, which is proven by our experimental results.

References

[1] Chip navigation in Expert. TCAD Driven CAD. Volume 8, Number 9, September 1997, pp. 3-5.

[2] Alfred V. Aho, John E. Hopcroft, and Jeffrey D. Ulman. The Design and Analysis of Computer Algorithms. Addison-Wesley, 1974.

[3] V. Feinberg. Geometrical problems of VLSI computer graphics, Radio i Sviaz, Moscow, 1987.

[4] Application of Scan Line Methodology to Perform Metric Operations in DRC. Simulation Standard. Volume 8, Number 12, December 1997, pp. 7-9.

Scanbox Approach to Shape Reconstruction for Automated Reticle Inspection

Abstract

The paper describes a generalization of the scanline approach [1] to reconstruction of the shape of planar object represented by a discrete point set with a given distance threshold d. This problem arises in applications of VLSI layout image processing, e.g., during automated reticle inspection.

Two algorithms are presented, with time complexities O(min {n^2, (n + t)log n}) and O(n log n), respectively, where n is the number of input points, t is the number of fixed-radius neighbor pairs. The first one is applicable for sparse point sets, e.g., for almost-grid point sets. In addition, ideas from the first algorithm, combined with the scanbox approach lead to the second algorithm efficient in the general case.

Introduction

Determining the shape of a planar object represented by a finite set of its points is a well-known problem of image processing. Among the major reasons for its complexity is lack of formalization of the notion of "shape". A natural approach to computationally hard problems is to design efficient algorithms treating special cases suitable to specific application areas. Various applications pose different requirements to "shape". Examples of simplest, or coarsest definitions of shape are the bounding box of an object and the convex hull. These definitions of "shape" deliver a reasonable trade-off for goal vs. efficiency in packaging, VLSI floorplanning, etc. More sophisticated "shape" definitions may be found in [2] and [3], Sect. 43.2 "Extracting Shape from Dot Patterns". A number of known algorithms in fact construct a whole family of shapes depending of a parameter of the algorithm, usually related with mutual proximity of points.

In image processing applications a common special case is shape reconstruction for a set of points approximately placed on a regular grid with step d. Examples of such grids are pixels of the scanned image or probe locations under regular probing. In these cases the following definition is often suitable.

The shape of a finite point set is a maximal-area polygonal region with side lengths not exceeding d that contains the given point set. Variations of this basic definition allow for "rectification" of shape boundary according to some angle or deflection thresholds for data compression. This definition is stated to grasp the notion of the external shape, however it may be generalized to allow for non-simply-connected regions, i.e., regions with holes.

Two algorithms are presented for shape reconstruction under the above definition of shape for a planar point set. The time complexity of the first algorithm, unlike the second one, depends on the sparseness of the input point set.

Both algorithms have the following I/O specifications:

INPUT: A set of points {a_1, a_2, ..., a_n} in the plane and proximity threshold d.

OUTPUT: Polygonal regions defining the shape of the given point set according to the above definition.

Sparseness-Sensitive Algorithm

Definition

The limited-distance neighbor graph, or d-near neighbor graph for a point set {a_1, a_2, ..., a_n} is the graph with vertex set {a_1, a_2, ..., a_n} in which two vertices are connected by the edge if and only if the distance between them is at most d.

Sketch of Algorithm 1

Step 1. Construct the d-near neighbor graph for the input point set. The graph is represented by its adjacency structure.

Step 2. For each point a_i do:

Step 2a. Sort the neighbors of a_i counterclockwise.

Step 2b. If for a pair (a_j, a_k) of consecutive neighbors in the sorted list from 2a one of the following conditions hold:

Condition A: Vectors (a_i a_j) and (a_i a_k) form an angle greater than Pi;

Condition B: The distance between a_j and a_k exceeds d;

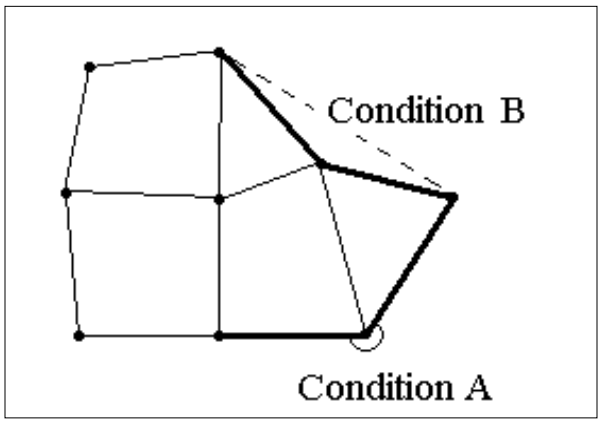

Figure 1. Conditions for Algorithm 1.

then the point a_i and the edges (a_i, a_j) and (a_i, a_k) are marked as exterior.

Step 3. Link marked points and edges into simple polygons.

Step 4. Delete polygons of length 4.

Step 5. Sort the remaining polygons into external boundaries and hole boundaries.

To estimate time complexity of Algorithm 1, we will go into further details.

Step 1 is equivalent to the problem of construction of the intersection graph for the set of circles in the plane of radii d/2 centered at the points {a_1, a_2, ..., a_n}. We propose the following algorithm.

Step 1a. Construct the intersection graph of squares centered at points {a_1, a_2, ..., a_n} with sides of length d and with sides parallel to coordinate axes.

Step 1b. For every pair of intersecting squares, check whether their inscribed circles intersect.

It is well-known that Step 1a may be executed in time O(n log n + k), where k is the number of intersections of squares.

Time complexity of the whole Step 1 is determined with the help of the following theorem.

Statement 1

Let {R_1, R_2, ..., R_n} and {S_1, S_2, ..., S_n} be the sets of squares of the same size with sides parallel of coordinate axes and the respective inscribed circles and let k and t be the numbers of intersections in the sets {R_i} and {S_i} respectively. Then k = O(t + n).

As a corollary, the complexity of Step 1 of Algorithm 1 is equal to O(n log n + t), where t is the number of edges of the d-near neighbor graph.

Time complexity for Step 2 is O(min {n^2, t log n}), [4].

Time complexity of steps 3, 4 is O(n).

In general case, it is known that time complexity of Step 5 (sorting polygons into external and internal boundaries) is O(n log n) [1]. However in our particular case the following statement allows us to reduce time complexity to O(n).

Statement 2

Let a_0 be the lowest leftmost vertex of a polygon constructed at Step 3 of Algorithm 1. If this polygon is an external boundary then Condition A was true for point a_0 at Step 2b, otherwise Condition B was true for a_0 at Step 2b.

Summing the estimates of all steps, we obtain that the worst-case time complexity of Algorithm 1 is

O(min {n^2, (n+t)log n}).

This algorithm is efficient for sparse point sets, for which t = Omega(n). In this case time complexity of the algorithm is reduced to O(n log n).

Scanbox Algorithm

For "dense" point sets, when the threshold d is taken too large resulting in large t, the following algorithm is proposed.

Consider the plane partitioned into square boxes with side d, i.e., into the sets

```
Bij = {(x,y): id <= x <  (i+1)d, jd <= y <  (j+1)d}.
```

Let P_ij be the subset of the input points falling inside B_ij and let S be the set of all nonempty boxes B_ij.

Define the d-near neighborhood graph on the set S to be the graph with vertex set S in which two vertices S_i and S_j are connected by an edge if there are two points a_i, a_j from {a_1, ..., a_n} such that a_i is in S_i and a_j is in S_j and the distance between a_i and a_j is less than d. The following simple statement will be useful in the construction of the algorithm.

Statement 3.

If S_i and S_j are adjacent vertices in the d-near neighborhood graph of set S, then these boxes are 8-adjacent in the plane (i.e., adjacent in the horizontal, vertical, or diagonal direction).

Continued on page 9....

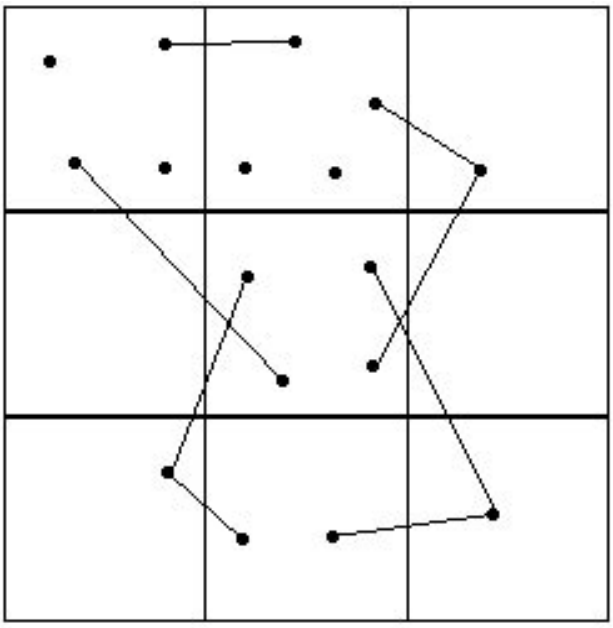

Figure 2. Algorithm 2, Step 3.

Circuit Verification via Hypergraph Realization

One of the most challenging and time consuming tasks in VLSI design automation is **Layout Versus Schematic (LVS)**. The problem is to test the consistency between the actual circuit, represented by the layout, and the nominal circuit upon which the design was based. From mathematical point of view the core problem is the **Hypergraph Isomorphism Problem (HIP)**.

Unfortunately, it is known as NP-hard, so (as for most of CAD problems) it is extremely important to investigate the frequently encountered particular cases. It is well-known fact that particular case of HIP - Graph Isomorphism Problem - becomes easy solvable when input graphs are planar or have bounded parameters . Following this association, the series of investigations were done for estimation of complexity of the HIP in case of planar hypergraphs. Despite of this problem was proven as NP-hard even for very restricted input parameters, some polynomial-time algorithms were invented for cases when most of hyperedges (they correspond to particular VLSI subcircuits) have the bounded capacity. Fortunately, the classified polynomially-solvable for planarity test hypergraphs are in the same time (being planar) easy checkable on isomorphism. This provides the real breakthrough in efficiency of LVS tools used in **Celebrity** (Silvaco suite for VLSI Design Automation) for wide class of integrated circuits.

The presented approach is based on investigations presented in [1-5]. First of all, we need to present some formal mathematical definitions.

Let **H=(X, R)** is hypergraph with set of vertices **X={x(i)},i = 1,..,n** and

set of edges **R={R(j)}, j = 1,..,h,** where every edge is arbitrary subset of vertices.

The *realization* of hypergraph **H** is such a graph **G=(X, E)** for which:

1) every induced subgraph **<R(j)>** in graph **G** is connected,

2) for each edge **xy** of graph **G** there exists the edge **R(j)** of hypergraph H which contains both **x** and **y**.

The *degree* **d(x)** of vertex **x** is the number of edges from **H** containing **x.** Hypergraph **H** is called *linear* if every two edges have no more than one common vertex.

The **Hypergraph Planarity Problem (HPP)** is formulated as follows: Given a hypergraph **H=(X, R),** to determine, is there exists its planar realization.

Hypergraph **H** is called *planar,* if such (planar) realization exists for **H.**

NP-completeness of HPP is established independently in [1] and [2]. More over, there was proven [3] that problem remains NP-complete even in cases when the following hypergraph parameters are bounded: cardinality of every edge **(|R(j)|<= 8)** or maximal degree of vertex **(d(x)<=2).** Besides, it is not difficult to show NP-completeness of HPP even for linear hypergraphs. Obviously, if **|R(j)| <= 2** then HPP becomes polynomially solvable because it is reduced to the simple graph planarity test. From other side, beginning from **|R(j)|<= 8** this problem becomes NP-complete. In the cases when upper bound on edge size takes values from the set {3,4,5,6,7}, the question of problem complexity remains open.

The main goal of this article is to present additional conditions, providing polynomial complexity for Hypergraph Planarity Problem solution.

Hypergraphs of Bounded Degree

Let **|R(j)| <= 3** for all **j=1,..,h** . As it was mentioned, namely from this moment the HPP becomes non trivial. Besides, for wide class of VLSI circuits about 90% of signal subcircuits (corresponding to edges of hypergraph) contain not more than three vertices. More than half from these subcircuits are, as rule 2-contact, i.e. contain exactly 2 vertices. These give us the background to introduce in consideration the *set of 2-edges* **P** - all edges consisting of 2 vertices, and *graph of 2-edges* **G=(Y, P) , (Y is subset of X)** induced from **H** by set of 2 edges **P**. This graph is subgraph of any realization of hypergraph H. Note, the analysis of graph of 2-edges has a key importance while creating of restricted exhaustion algorithms for detecting of hypergraph planarity.

It is clear, that if **G** is non-planar, then **H** is non planar. So, we can consider graph **G** planar without loss of generality.

Now suppose, that hypergraph **H** does satisfy the following:

Condition A. *Every edge consists of three or less vertices, graph of 2-edges is 3-connected and set of its vertices consists of all vertices of hypergraph (Y=X).*

Now we will show how this condition provides polynomial feasibility of HPP.

Let us **RR=R-P**. Without loss of generality, consider, that **RR={R(i)}, i=1,..,t** and **R(i)={x(i),y(i),z(i)}.** For each **R(i),** put in correspondence the triple of edges **T(i) ={x(i)y(i), x(i)z(i), y(i)z(i)}**. Then, graph **A=(X, Union(E,P))** will be realization of hypergraph **H = (X, R),** if it contains minimum two edges from each triple **T(i)**.

Assign now to each edge e(j) from set **T = Union{T(i)} , i=1,..,t** the Boolean variable **a(j)**.

It takes the value 1 , if **e(j)** belongs to **E**, and value 0 otherwise. Then, for every triple **T(i)={e(k),e(l),e(m)}** build the set of disjunctions **S(i)={a(k) V a(m), a(l) V a(m), a(k) V a(l)}.** All true-implying sets for **S(i)** directly correspond namely to those subsets from **T(i)** which consist pf more than one element. This is why graph **A** will be realization of **H** if and only if when **E** consists of precisely from those **e(j)**, which correspond to some true implying set from set of disjunctions **S=Union{S(i)}**.

Because of graph **G** is 3-connected, the addition of edge from **E** to **G** does not violate the planarity only if vertices jointed by this edge belong to the common graph face. Assume **a,b,c,d** are vertices of bounding cycle for some face **C** of graph **A** (written in original cycle-resided succession). Then, a pair of edges (**ac, bd**) of graph **A** will be called conflict pair in relation to face **C**. Obviously, graph, obtained from **A** by adding of edge set {**ac,bd**}, is non planar.

To take into account the planarity of **A**, it is necessary to extract the set **K={(e(l),e(m))}** consisting the all conflict pairs of edges from **T** in relation to faces of graph **G**. The condition of planarity is equivalent to the presence in **E** not more than one edge from each pair belonging to **K**. This, in turn, corresponds to satisfying the disjunction set **B={Union{a(l) V a(m)}}**.

So, HPP is reduced to the problem of construction of satisfiable set on the set **{a(j)}** for set of disjunctions **Union(S, B).**

Note, each disjunction consists of not more than two literals. Therefore, we have the well-known task 2- Satisfying of for which the polynomial-time algorithm exists [7]. The time, required for this algorithm is estimated in our case as **O(|T| * |UNION(S, B)|)**. The construction of **K** (set of conflict edges) requires, obviously, not more than **O(|CC| * |R| ^2)**, where **CC** is the set faces of graph **G**. Taking in account, that **|CC|=O(|X|)** and **|Union(S, B)|=O(|R|),** we will obtain the worst-case estimation for testing of planarity of graph satisfying the Condition A as low as **O(|X| * |R| ^2+|R|^3).**

Linear Hypergraphs

Consider now linear hypergraphs. Obviously, graphs are linear hypergraphs and (in certain sense) it is possible to consider linear hypergraphs as closest extension of graphs. Besides, namely linear hypergraphs are appeared most frequently while designing, for example, layouts of circuits with one layer of commutation. These circumstances explain the interest to investigation of conditions providing the polynomial feasibility of HPP for planar hypergraphs.

Note, that similar to previous point, the central role here plays the condition of 3-connectivity of 2-edge graph **G=(X, P)** of hypergraph **H=(X, R).** Considering the graph **G** planar , we introduce the set **U(G),** assembling all such pairs of vertices **xy** which belong simultaneously both to common edge of hypergraph **H** and common face of graph **G**. Note, this set includes the edges of any planar realization of **H** (if such realization exists). As for previous point, assume **K** as set of conflict pairs of edges in relation to graph **G.**

Condition B. *, **H=(X,R)** - hypergraph with 3-connected graph of 2-edges and no one edge from **U(G)** participates in more than one pair from conflict set **K**.*

Now we will show that Planarity Problem for linear hypergraphs satisfying the above condition is reduced to known polynomially-feasible task [8] of construction the set of edge-disjoint bases for special graphical matroids.

Consider the following sets:
I = {(i,j)} is the set of pairs of indices, corresponding to pairs of conlict edges **(e(i), e(j));**

I_m is the subset of I including pairs of indices of such conflict edges, one of which connects vertices of hyperedge **R(m).**

Now each pair of edges **(e(i), e(j))** from set **K** is exchanged by set from 5 edges:

e_1i = x(i)a(i,j), e_1j = x(j)a(i,j), e_2i = y(i)b(i,j), e_2j = y(j)b(i,j), e_ij = a(i,j)b(i,j).

As result of this transformation, we will obtain the graph **G1=(X1,E1)** with set of vertices

X1=Union(X,{a(i,j)},{b(i,j)})

and set of edges

E1=U(G)-Union({e(i),e(j),e_1i,e_1j,e_2i,e_2j,e_ij}),

where **(i,j)** are all pairs from **I**.

Then, put in correspondence to each hyperedge **R(i)** from set **R- P** the subgraph **G1_i=(X_i, E_i)** of graph **G1,** induced by the set of vertices **X_i= Union(R(i),Union({a(p,q),b(p,q)})),** where **(p,q)** are taken from **I_i.**

Now it is easy to see, that planar realization of hypergraph **H** exists if and only if there exists the set of edge-disjoined trees **{T(i)}**, where each **T(i)** is spanning tree for corresponding graphs **G1_i** (i.e. **{T(i)}** are disjoined bases of respective graphical matroid).

Really, the condition of non-intersection for edge sets of mentioned trees means that edge **(a(i,j) b(i,j))** taken from intersection of **E_k** and **E_l,** belongs only to one of trees **T(k)** and **T(l)** (spanning for **G1_k** and **G1_l** correspondingly).

Suppose, that for conflict pair **(e(i)=x(i)y(i), e(j)=x(j)y(j))** vertices **x(i),y(i)** belong to hyperedge **R(k),** while vertices **x(j),y(j)** belong to hyperedge **R(l)**. Then inclusion of edge **a(i,j)b(i,j)** to tree **T(k)** means the choice of edge **e(i)** from conflict pair **(e(i),e(j))** for inclusion to the set of edges of planar realization for hypergraph **H.**

Note, to construct the target matroid for hypergraphs fitted with Condition B we need as low as **O(|E(G)| * |K|)** operations.

References

[1] Johnson D.S., Pollak H.O. Hypergraph Planarity and the Complexity of Drawing Venn Diagrams. J. of Graph Theory. 1987.- Vol.11, No. 3., pp. 309-325.

[2] Azarenok A., Sarvanov V. The Complexity of Hypergraph planar realization. Notices of the National Academy of Science, Minsk, Ser. Phys. And Math. Sciences.-1987.- No.4., pp. 10-12.

[3] Azarenok A. Computational Complexity of Graph Problems in VLSI Layout Design. Preprint (5(405)) of the Mathematical Institute, Belarussian Academy of Science, 1990.

[4] Volochina A., Feinberg V. About Hypergraph Planarity. Proceedings of the National Academy of Science, Minsk. - 1984.- Vol.28, No.4, pp. 309-311.

[5] Feinberg V., Levin A., Rabinovich E. VLSI Planarization: Methods, Models, Implementation. Kluwer Academic Publishers (Dardrecht/Boston/London). - 1997.

[6] Zemlyachenko V., Karneyenka M., Tyshkevich R. Graph Isomorphism Problem. J.Soviet Mathematics.- 1985.- pp. 1426-1481.

[7] Cook S.A. The Complexity of theorem-proving procedures. Proc. 3rd Ann. Symp. On Theory of Computing.-1971.- pp.151- 158.

[8] Roskind J., Tarjan R. A Note on Finding Minimum-Cost Edge-Disjoint Spanning Trees. Math. Of Oper. Research.- 1985.-Vol.10, No. 4, pp.701-708.

...continued from page 6

Sketch of Algorithm 2

Step 1. Construct the d-near neighbor graph for boxes of S using the following sub-steps.

Step 1a. Sort input points into boxes B_ij.

Step 1b. Inside each nonempty box S_i construct the convex hull of points lying inside it.

Step 1c. For each pair of 8-adjacent boxes find the distance between the corresponding point sets.

Step 2. Construct the "shape" for the set S, e.g., using an algorithm similar to Algorithm 1 or by scanbox approach.

Step 3. For each pair of consecutive boxes from the boundary of the shape from Step 2 construct an appropriate common tangent for convex hulls constructed at Step 1b.

Step 4. If the length of some tangent exceeds d or a pair of tangents intersect, remove the violation by moving the "touch" points along the polygons. Finally, collect polygons from tangents, moved tangents and chains from convex polygons from step 1b.

To estimate time complexity, note first that this partition into steps is made only for conceptual clarity. In fact most of them may be implemented in the course of a single scan through the plane by a pair of adjacent boxes. Bearing this in mind, let us estimate the separate operations of the algorithm.

Step 1a takes O(n) time. Step 1b is estimated by O(n log n) time [3]. At step 1c, the distance between a pair of convex polygons may be found in time O(log n) [3], totalling to by O(n log n) for the whole step.

At Step 2, constructing the neighborhood graph may be performed in time O(n log n) using Algorithm 1, since the corresponding vertex set is inherently "sparse" (every vertex may have at most 8 neighbors). However, as it is already noted, a simpler approach is possible using scanbox approach.

At Step 3, a common tangent for a pair of convex polygons may be constructed in time O(log n) using dichotomy approach described, e.g., in the divide-and-conquer approach for convex hull construction [5].

Finally, Step 4 is clearly O(n), which gives the total estimate of O(n log n) for time complexity of the Scanbox Algorithm.

References

[1] V. Feinberg. Geometrical Problems of VLSI Computer Graphics, Radio i Sviaz, Moscow, 1987.

[2] Generalized Convexity Approach to Cell Boundary Shape Approximation, TCAD Driven CAD, 8 (1997), no. 9, pp. 8-9.

[3] Handbook of Discrete and Computational Geometry, eds. J.E. Goodman and J. O'Rourke, CRC Press, 1997, 991p.

[4] J. E. Goodman and R. Pollack, Multidimensional sorting, SIAM J. Comput. 12 (1983) 484-507.

[5] F.P. Preparata and M.I. Shamos, Computational Geometry: An Introduction. Springer-Verlag, New York, 1985.

Calendar of Events

March

1
2
3
4
5 W/S - Munich, Germany
6
7
8
9
10
11 W/S - Guildford, UK
12
13
14
15
16
17
18
19 W/S - Grenoble, France
20
21
22
23
24
25 W/S - Scottsdale, AZ
26 W/S - Scottsdale, AZ
27 W/S - Yokohama, Japan
28
29
30
31 W/S - Grenoble, France

April

1
2
3
4
5
6
7 W/S - Guildford, UK
8
9
10
11
12
13
14
15
16 W/S - Munich, Germany
17
18
19
20
21 W/S - Scottsdale, AZ
22 W/S - Scottsdale, AZ
23
24
25
26
27
28 W/S - Grenoble, France
29
30

Bulletin Board

New CAD Development Building #8 Remodeling Completed!

The fast growing CAD development group will relocate to an ultra modern development center. This facility will enable fast pace development of Legacy group of products: STORM and CELEBRITY. Come and visit us for a demonstration of the most advanced NT based CAD design solutions.

Very Successful Product Launch at Asia-Pacific DAC!

Silvaco has successfully launched new product groups STORM and CELEBRITY at the DAC show held in Yokohama on February 12th and 13th. During the two day show, over 600 demonstrations of Expert, Savage, EXACT and CLEVER were given. Detailed technical presentations were delivered to a packed audience. The attendees at the presentations and at the show have shown strong acceptance of these unique NT based CAD tools.

See Silvaco at San Francisco DAC!

Silvaco will be exhibiting the latest in TCAD Driven CAD tools at the Design Automation Conference. The exhibition will be 15-17th June at the Moscone Center, San Francisco. Silvaco will present the latest advances in:

- SmartSpice
- NT-based Layout and Verification Tools
- Technology based Parasitic Extraction
- 3D TCAD

For more information on any of our workshops, please check our web site at ***http://www.silvaco.com***

Hints, Tips and Solutions

Mikalai Karneyenka, Applications and Support Engineer

Q: Is it possible to perform a DRC check on portions rather than on the whole circuit useful when only 2 or 3 errors to fix ?

A: You may clip out the required piece of layout (menu Tools>>Clip Out). This operation preserves coordinates and hierarchy of instances that were not cut during clipping. Then you may run DRC on the clip and see if there are any errors. There are two problems:

- at the border fringe errors appear that are not actually present in the original layout;
- there is no direct access to erratic objects in the original layout: you must look at the coordinates where the error is reported in the clip and then find the same place in the original (You may use the operation Edit>>Position Cursor). Therefore you must proceed as follows.
- Open two windows: for original layout and for clipped piece;

  Go to Clip window, run DRC script and load errors for it.
- Go to next error, place cursor at an appropriate point and read the coordinates of cursor position;
- Go into the window with the original layout, make the zoom appropriate to see details;
- Press Ctrl-P and enter the cursor position read in the clip window and press Enter. This will place cursor in the same position and you will easily find the objects generating the error shown in the Clip window.

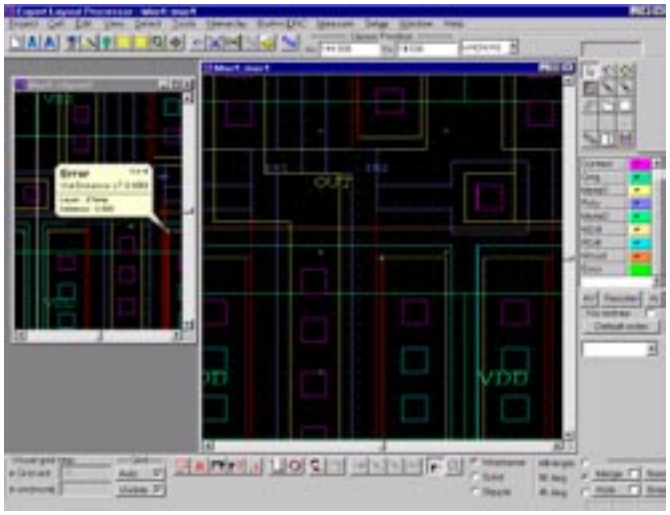

Figure 1.

Q: In the Layer Setup I added a new layer and placed it first. But the layout window is not redrawn, and in fact the new layer is shown last at the Layer Bar.

A: The Layer bar shows the order of layers in the current view. It may be different from the default order defined in the technology file (and shown in the Layer Setup list). To restore the order of layers to the technology default, click the "Default order" button at the layer bar.

Q: Our department purchased 9 copies of Expert for designers and 2 copies of Savage for verification team to run full-chip DRC. Can I run Savage on the server and then correct DRC errors on my own PC?

A: Yes, you can, provided the following requirements are satisfied.

- the projects on both computers must be in eif format;
- their cell hierarchy must be identical (but geometry in cells may be edited, e.g., some DRC errors corrected).

The result a of DRC run is an error database with extension .edb . It lies in the subdirectory with the name <ProjName>.DRC, where <ProjName> is the name of the checked .eif file.

You must create the same <ProjName>.DRC subdirectory in the directory of your computer where your gds/eif file lies and copy the .edb file from **Savage** server into it.

After that you may process DRC errors within **Expert** as usual, i.e.,

- load <ProjName>.eif
- load error database by menu command Built-in DRC>>Load Errors

Call for Questions

If you have hints, tips, solutions or questions to contribute, please contact our Applications and Support Department

Phone: (408) 567-1000 Fax: (408) 496-6080

e-mail: ***support@silvaco.com***

Hints, Tips and Solutions Archive

Check our our Web Page to see more details of this example plus an archive of previous Hints, Tips, and Solutions

http://www.silvaco.com

Join the Winning Team!

Standardize your process/device and CAD design using Silvaco's

"TCAD Driven CAD™"

To get a demo and product description contact or visit a Silvaco office near you:

- Santa Clara
- Phoenix
- Austin
- Boston
- Guildford
- Grenoble
- Munich
- Tokyo
- Seoul
- Hsinchu

SILVACO INTERNATIONAL

USA Headquarters:

Silvaco International
4701 Patrick Henry Drive, Bldg. 2
Santa Clara, CA 95054 USA

Phone: 408-567-1000
Fax: 408-496-6080

sales@silvaco.com
www.silvaco.com

Contacts:

Silvaco Japan
jpsales@silvaco.com

Silvaco Korea
krsales@silvaco.com

Silvaco Taiwan
twsales@silvaco.com

Silvaco Singapore
sgsales@silvaco.com

Silvaco UK
uksales@silvaco.com

Silvaco France
frsales@silvaco.com

Silvaco Germany
desales@silvaco.com

*Products Licensed through Silvaco or e*ECAD*