# Simulation Standard

Connecting TCAD To Tapeout

A Journal for CAD/CAE Engineers


## *Maverick:*
## Hierarchical Netlist Extractor for PC Platforms

*Alex Azarenok and Valeri Feinberg, Silvaco International*

### Introduction

***Maverick*** is a modern hierarchical netlist extractor, providing extraordinary efficiency as well as comprehensive features and ease of use. It runs on PC under Microsoft Windows NT providing unique productivity in processing of virtually any size VLSI/ULSI designs. The extractor is fully integrated with state-of-the-art ***Expert*** Layout Editor, and represents important part of its verification shell along with ***Savage*** Design Rule Checker, and ***Guardian*** LVS System.

***Maverick*** is used to restore circuit netlist from hierarchically designed mask layout. It accomplishes that by:

- accurately identifying standard semiconductor devices (transistors, diodes, resistors, capacitances) as well as generic (user-defined) constructions
- accurately estimating most common parameters for each device
- extracting hierarchical connectivity with fast runtime and low memory requirements

Figure 1. Integrated GUI of ***Maverick***.

### Main Features

- full-chip hierarchical extraction for designs containing arbitrary number of elements
- user- definable and technology- tunable extraction process
- fully hierarchical processing and reporting
- maximum preserving of original hierarchy for layout pathologies
- handling of arbitrary-shaped geometry
- essential advantage in processing speed in comparison with existing tools

***Maverick*** is a core part of the ***CELEBRITY*** framework providing input to Layout Versus Schematic (LVS), a comparison tool for logical and physical networks.

### Advantages

At the present time there are no PC-based tools in the market capable of performing netlist extraction for comparably large (several millions devices) layouts. At the same time the demand for such CAD tools is dramatically increasing.

*Continued on page 2....*

INSIDE





SILVACO INTERNATIONAL

However, even UNIX-based tools which are capable to do full-chip verification of large designs need to improve both productivity and level of comprehension in use.

Analysis of current extractors shows that they suffer from one or more of these typical problems

- Too slow
- Can not handle large design
- Either act only over flatten design or assume essential restrictions on hierarchy structure.

  Most those tools which claim hierarchical extraction have weaknesses in handling of hierarchically distributed devices (i.e. devices whose elements are not concentrated inside primary primitives of a single cell). The typical behavior is one of the following:

- Do not detect such a devices at all,
- Detect overlapping of hierarchically disjoined primitives from device-making layers but do not report distributed devices, advising to user to "improve" hierarchy structure.
- Detect the hierarchical overlapping, but flatten all instances of those cells whose any instance participates in hierarchical overlapping.
- Redistribution of device-making layout elements between hierarchy levels (to collect them on common level), followed by rebuilding the hierarchy structure of design. Extraction results in this case are output in terms of new hierarchy which is difficult to interpret.

To be competitive, ***Maverick*** delivers unique features:

- incomparable efficiency by using

  the most advanced mathematical methods for hierarchical geometry operations and pattern recognition supplied by

  efficient access to proprietary layout data base common for ***Expert*** layout editor and ***Savage*** design rule checker
- most comprehensive and easy use through:
  - true hierarchy reporting. Presenting of the all extracted devices in the most convenient for user form, where their components enumerated in the terms of source design hierarchy
  - integration of ***Maverick*** into state-of-the-art environment of ***Expert*** (see Figures 1 and 2).
- Multi-technology (Mixed-technology) design handling
- Automatic recognition and assignment of contacts
- Reflection of device distribution on hierarchy structure
- Flexible tools for connectivity setup and device definition
- On-the-fly (on guard) control of layout's netlist consistency during editing
- Incremental and selective modes (preserving already extracted components, re-extraction on edited areas only)

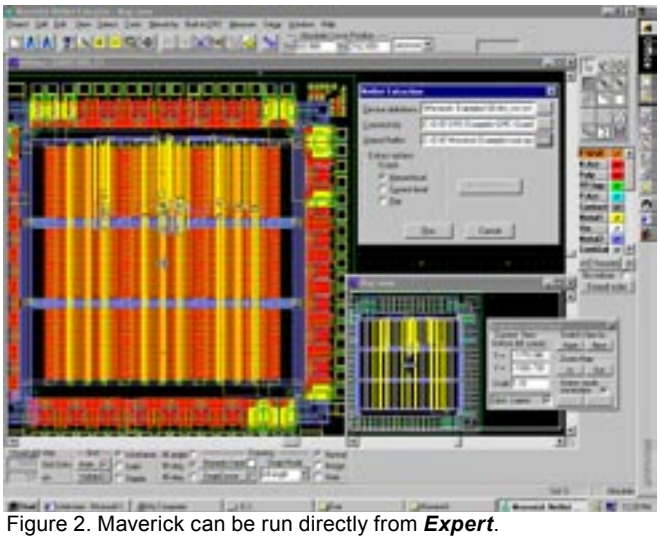

Figure 2. Maverick can be run directly from ***Expert***.

In addition an interface with the ***DISCOVERY*** family of physics-based Interconnect Parasitic Extraction Tools is planned.

## True Hierarchical Nature of Processing

**Maverick** efficiently handles traditionally difficult kinds of designs where hierarchical units are strongly overlapped. A typical example is presented in Figures 3 through 5.

In contrast to other hierarchical extraction tools, ***Maverick*** provides:

- efficient hierarchical recognition and reporting of distributed devices whose elements are not concentrated inside a single cell
- fast and reliable hierarchical deriving of connectivity regardless of existence and nature of hierarchically overlapped components

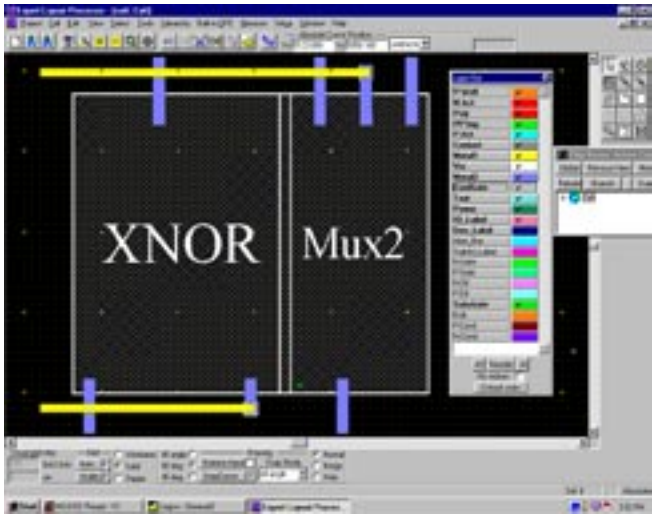


Figure 3. Sample hierarchical layout with overlapped cells.

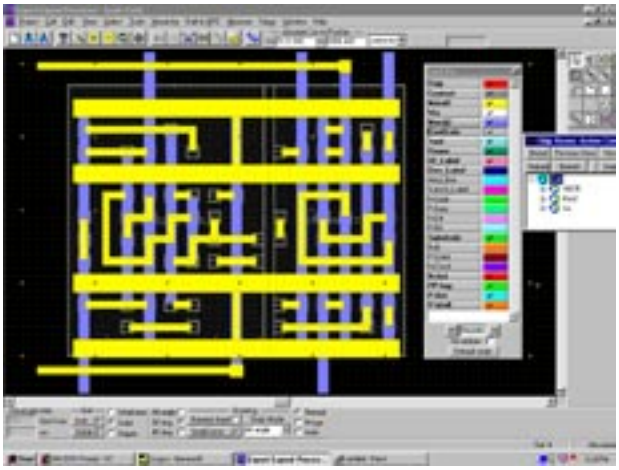

Figure 4. Sample from Figure 3 with one level of hierarchy expanded.

Figure 5. Sample from Figure 3 with all hierarchy expanded.

## Basic Methodology

### Core Problem of Hierarchical Extraction.

Hierarchy must be utilized in design of VLSI circuits to handle the large amount of layout data efficiently. Entry of the circuit schematic as well as layout description is simplified and the amount of data to be stored for the circuit description is substantially reduced. Circuit modifications are easily performed as a changes in a basic cell are automatically repeated on all multiple uses of the cell. This hierarchy in circuit description should also be exploited to gain speed and efficiency in VLSI circuit verification and element extraction. The handling of hierarchy on layout verification would have been no problem if the layout were truly hierarchical and the cell instances did not interact through layout overlap. However, cell instance interaction can hardly be avoided since the designer can easily find a number if reasons why cells should be allowed to overlap. From the designers point of view complete release of any constraint is preferable. However, this freedom for the designer complicates design verification.

The usual solution to this problem is either:

a) to flatten the design in overlapping areas thereby letting these problems surface at the next hierarchical level,

b) to form a new disjunct hierarchy.

However both of this schemes are unacceptable since they produce a different hierarchy than the original hierarchy specified by designer. Instead the hierarchy of the design must be preserved as much as possible. Implicitly, the designer violates the hierarchy and creates a new one if he allows circuit elements to be formed when cell instances overlap.

The core problem is:

Does the element belong to the hierarchical level where the cells overlap or does it belong to one of the cell instances?

The same is true for a DRC. If design rule violations appear when cell instances overlap the question is:

Which hierarchical level should be chosen to display these errors to make them as simple as possible for the designer to correct?

To solve these problems we need to define a set of rules specifying the most correct hierarchical level in the original circuit description to which elements in overlapping layout should be referred. In short, this level is the lowest hierarchical level where an element always appears.

The model of dual layout forest is implemented as a mean of performing fully hierarchical design verification without any restrictions on cell instance overlaps. This provides us with fast and general method of marking design rule errors or extracted devices at the correct hierarchical level. The dual layout forest for each element is built up as a layout data is processed from the bottom up. When layout processing is completed we can determine the most appropriate cell for each element. New elements are formed as a layout from different cell overlaps can now be placed at the lowest level of hierarchy where they always appear instead of being indiscriminately incorporated in the parent cell. Thereby, the method using dual layout forests preserves the original hierarchy to the largest possible extent.

### Layout Pattern Recognition

The capability of the dual layout forest by looking at the way it handles hierarchical mask operations used in extraction process. These Boolean mask operations between layers in a layout must be carried out before circuit extraction in order to create device specific rectangles. If the design is described in actual mask layers we must form abstract layers describing electrical elements such as transistor gates and N+ an P+ diffusions. In a polysilicon gate CMOS process, transistors are formed by the overlap of active

(thinoxide) and polysilicon rectangles. Gate rectangles are formed by Boolean AND operations between these layers. After formation of the transistor the active rectangle belongs to two different electrical nodes, the source and the drain. Since the circuit extractor cannot handle rectangles that are shared between two nodes these have to be divided into two different rectangles. This is done with the Boolean DIFF operation.

**Dual Forest of Hierarchy.**

The dual layout forest is a major mean of accomplishing fully hierarchical design verification according to the predefined rules. The rule set or dual layout forest allow the original hierarchy to be preserved without any restrictions on cell instance overlaps. When designer himself endangers the hierarchy by letting new devices form as cell instances overlaps, this is handled by the rule system embedded in the dual layout forest.

During hierarchical design verification or device extraction process, the layout data is processed twice:

1) We need to carry out some mask operations before the actual design rule checking and circuit extraction takes place.

2) The circuit extraction is performed such a way that layout data is processed cell by cell. For each cell its cell instances has already been checked.

In the first run we form new elements when layers overlap. In the second run we look for design rule errors or couplings between nodes.

Each of these new elements is assigned to a certain rectangle in the layout and dual layout forest. New elements can be formed when cells overlap. Elements can also disappear when cells overlap if, a design rule error is corrected by the way the cell is used. In all these cases we make notes if these changes by adding or removing branches in an dual layout forests of these elements. After each run is completed we can go through the layout data again and use the dual layout forests to place rectangles or design rule errors and couplings, respectively, in the most appropriate or correct cell.

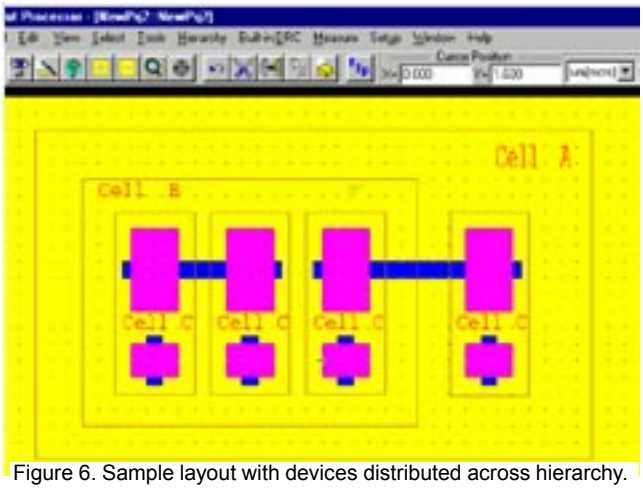

Figure 6. Sample layout with devices distributed across hierarchy.

A hierarchical layout can then be modeled as a rooted directed forest whose leaf cells represent flat layouts. As an example, a simple layout is shown on Figure 6.

A leaf cell, C, that contains two active rectangles and one polysilicon rectangle. One of the active rectangles overlaps the polysilicon rectangle forming a transistor. This transistor belongs to cell C. Cell C is instantiated three times in cell B at the next hierarchical level and once in the main cell, A, at the topmost hierarchical level. Cell B is used one time in cell A. Cell B contains polysilicon overlapping the right active rectangle in cell C at two of three instances. This rectangle was not overlapped by the polysilicon in cell C itself so new transistors are formed. At the top most level, in cell A, there is also a polysilicon, forming new transistors both in instance of cell B and the directly instantiated cell C. There is also N+ implant rectangle in cell A, changing the transistor in cell C from P-channel to N-channel at one of the instances. This layout will be used to show the generality of the dual layout forest in hierarchical AND and DIFF operations. The rooted directed tree of this layout is shown in Figure 7a. The main cell, A, is a root of the forest. Cell C is the only leaf cell in this example. When the Boolean mask operations are performed we traverse the layout tree from the bottom up. To keep track of the different paths from cell C to the main cell we can draw a dual layout forest with cell C as a root as shown in Figure 7b.

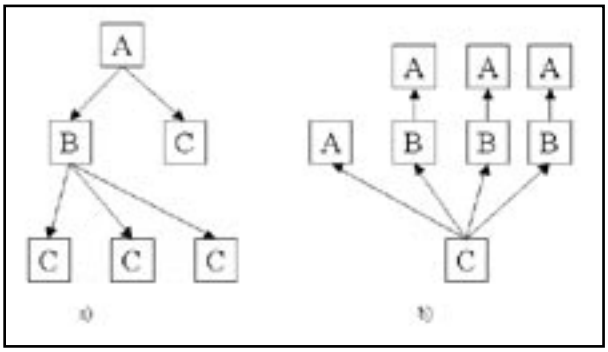

Figure 7. Cell hierarchy tree and its dual forest.

When we start the Boolean mask operations, gate rectangles belonging to cell C are formed where the active and the polysilicon rectangles overlap. However, as we go on with the mask operations with cell B, a transistor is formed through the polysilicon overlap at two of the instances of cell C. A gate rectangle is now formed too through the AND operation. Since we do not know at how many instances in the cell C this overlap will occur until the mask operations are completed we must mark those instances in the dual layout forest where it does appear and save them until the mask operations of the chip are completed. This is shown in Figure 8a. To simplify the dual layout forest we only need to record those hierarchical levels where the status of rectangle is changed. Such a reduced dual layout forest is shown in

Fig. 8b. We now see that even if the gate does not exist in cell B at the rightmost instance it exists at all instances of this B cell in A. We can then reduce the dual layout forest further by deleting the three A vertices and mark the rectangle true in cell B shown in Fig. 8c. Hence, the rectangle exists at all the instances and using the same rule again the dual layout forest is reduced to the root cell as shown in figure 8d.

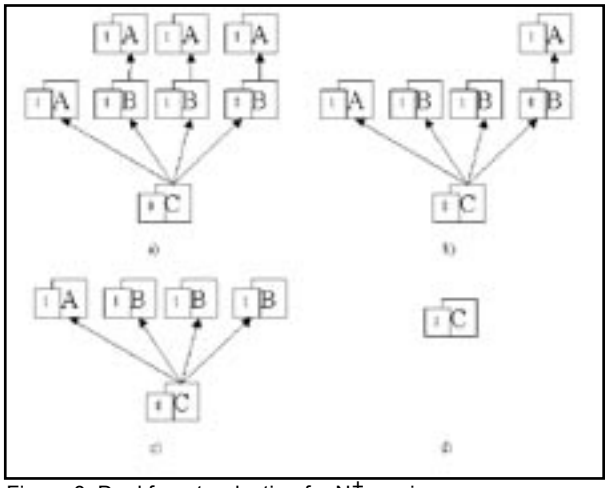


Figure 8. Dual forest reduction for $N^+$- regions.

The active rectangle overlapped by polysilicon will be considered to form a gate rectangle at the cell C level. This means that even if the element did not exist in the cell but exists in all the instances of the cell it will be transferred to the cell. For the left gate rectangle, formed when active and polysilicon overlap in cell C, the dual layout forest only consists of the root since the rectangle status is not changed at higher hierarchical levels. This is true for all the rectangles in a strictly non-overlapping hierarchical layout, where each rectangle belongs to the cell where it appears as a flat layout. This example shows the use of the of the dual layout forest for rectangles formed by the AND operation between mask layers. Such elements might be correctly moved down to the lowest possible hierarchical layer instead of simply remaining at the level where they are formed. However, in the case of DIFF-operations the use of an dual layout forest is an absolute necessity for hierarchical operations because we may need to go back to the cell instances and selectively divide already processed rectangles. This is possible with the dual layout forest since it keeps track of the instances where the rectangle's status is changed. If we return to the layout of cell C in Fig. 6 the right active rectangle is not overlapped by polysilicon is used to form a P+-diffusion rectangle in cell C. However, as we go on with the mask operations to cell B the transistor is formed through the polysilicon overlap at two of the instances of cell C. The P+-rectangle is already stored in cell C should now be divided into two smaller P+-rectangles through the DIFF operation. Since information is to be removed from cell C we must go through all its instances before we can do this. The dual layout forest for each of these source/drain P+- rectangles is shown in Figure 9.

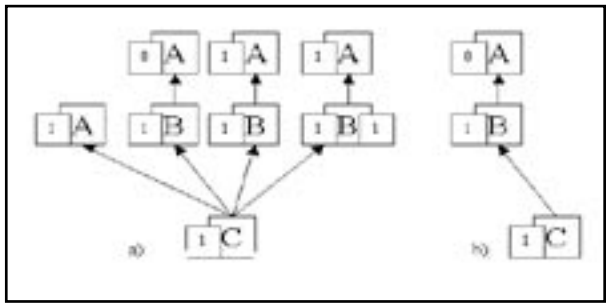


Figure 9. Dual forest reduction for $P^+$- regions.

instance (marked false in a dual layout forest).

The following procedure is used in the build-up of the dual layout forests:

When element changes status the program determines which rectangle the element is associated with and a list with the path of instances from this rectangle to the current cell. Three parameters are sent when procedure is called: 1) new status of the element; 2) path; 3) dual layout forest associated with the element.

The procedure then recursively modifies dual forest. It presumes that cell instances of the cell are checked before the cell itself. The dual layout forests are formed as mask operations proceed from leaf cells to the main cell. When changes in a rectangle's status are encountered new branches are added to or removed from the dual layout forest. This is easily done since the path between the involved cells is returned by the search procedure.

Since no operation is performed until changes are encountered no extra CPU time or memory storage space is wasted in dual layout forests for strict hierarchical designs. When mask operations are completed the layout data is processed again and rectangles placed in appropriate cells according to their dual layout forests. Finally the dual layout forests are disposed of as they are not needed anymore.

So, the dual layout forest provides us with possibility of going back to cell instances and dividing already processed rectangles in to smaller rectangles with DIFF operation. It also makes it possible to move rectangles formed by AND operations to a lower and more correct hierarchical level. Embedded in the dual layout forest are rules for determining the correct hierarchical level for the elements formed by cell overlaps. We have defined the correct level as the lowest level of hierarchy along the path from the leaf cell to the main cell where the element always appears in all of its instances. This means that an element that appears at all instances of the cell belongs to this cell and is located only here. It

also means that if an element exists at all of the instances of the cell, but not the cell itself, it will be transferred to the cell anyway. An element that disappears at higher levels is removed from abstract layout. An element that is only removed at some of its instances is maintained at the other instances. An element that appears at some of its instances is only marked at these instances.

**NOTE:** Using the dual layout forest to display errors in DRC makes it possible to display only true errors. Errors occurring in leaf cells but which are removed later no longer have to be displayed in the leaf cell. The designer no longer has to be follow how errors are created and removed through the error printout.

#### Assembling of Connectivity.

When all designed devices are extracted, the most efficient way to obtain interconnections between their pins is to apply appropriate dialect of classical scan-line algorithm ([3]). Being enriched by specifically tuned set merging algorithm ([2]) and reduced model of circuit hierarchy handling ([4]), it provides the optimal efficiency for deriving of interconnections. Even in worst case (flatten layout) the algorithm requires the time proportional NlogN, where N is the overall quantity of vertices of polygons from layers using for interconnections.

## Interface Overview

Both batch and built-in **Expert** GUI modes are available for **Maverick.** Note, built-in version is supplied by advanced viewer for convenient displaying of circuits and comprehensive representation of hierarchy of distributed devices.

**Maverick** works with layout presented in EIF (**Expert** Internal Format), utilizing very efficient access functions using by **Expert** and specially designed hierarchy handlers.

It also is able to accept data prepared in GDSII and CIF formats.

**Maverick** does not need any kind of labels to be used for layout components, but labels can be used when appropriate regime is given .

To restore correct electrical circuits, extractor utilizes specifications of devices, basic circuit elements (e.g., transistors, diodes, resistors, capacitors), customized configurations and functional knots (subcircuits).

Every device is described by device attribute block (DAB).

Device attribute blocks are collected either in technology file or in the special Device Attribute File. It is also possible to provide correction of device attributes as well as creating of new device descriptions via GUI.

Connectivity info is specified as sets of layers connected by a connecting layer (via layer):

CONNECT=(“layer1”, “lay2”, “lay3”,...,”vialayer”)

Layers, mentioned in DABs or connectivity statements can be as basic (technology) layers as generated ones (i.e. obtained with help of layer construction statements).

There are 3 sources of layer descriptions:

1) Technology file; 2) Rule script file; 3) GUI forms

Connectivity also can be stated using all enumerated ways

Construction and connectivity statements used both in technology and rule script files have the same syntax basically inherited from **Expert/Savage** DRC script language.

Run options are located in special configuration file. They also can be assigned/corrected using **Maverick’s** Extraction Setup. Several major options (concerned, basically, hierarchy processing/reporting) are presented below:

TRUE_HIERARCHICAL
FULL_REGARDLESS_HIERARCHY
PARENT_LEVEL_ONLY
HIERARCHICAL_IGNORING_OVERLAP
FLAT_OVERLAP
STOP_IF_OVERLAP
RECONSTRUCT_HIERARCHY
REPORT_IN_SPICE

Maverick generates two types of netlist: one in internal format used for internal LVS comparison, and the other in standard SPICE format used for running external LVS programs and circuit simulation.

Note also, that devices, whose elements are distributed through hierarchy, are reported in special file, containing hierarchy history of components. In addition to this report file, GUI solution is designed for convenient visual highlighting of distributed device genesis.

#### References


[1] L.K. Scheffer, R. Soetarman “Hierarchical Analysis of IC Artwork”, Proc. 22nd Design Automation Conference, p. 293-298.

[2] J.D. Ullman. “Computational Aspects of VLSI”, Morgan-Kaufmann, 1992.

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

[4] A.Azarenok. V.Sarvanov. Extremal realizations of hypergraphs in VLSI design. Reports of the National Academy of Science. Minsk, Vol.30, No. 10, 1989.

# Optimum Standard Cell Layout Using Weighted Cycle Linear Placement

*Mikalai Karneyenka, Silvaco International,*
*Victor Lepin, Institute of Mathematics, Belarusan Academy of Sciences*

## Introduction

Placement and routing problems for integrated circuits are inherently interrelated and extremely complex from the algorithmic point of view. This complexity grows exponentially as the scale of integration increases. To cope with their complexity within fully automated design methodologies, a common approach is to restrict the design framework both in terms of the structure of the target circuit and design algorithms.

A widely used approach is standard cell design (Fig. 1), in which almost all layout cells are of the same height, so it is convenient to arrange them in rows. If there some cells can be grouped in such a way that interconnects between the groups may be ignored or accounted for in some hierarchical way, then a locally-optimal placement of cells within each group may be approximated by the model of linear placement as shown in Figure 2. The quality of placement is estimated by some measure of interconnectivity provided that all wires run horizontally.

In many cases the latter problem may be reduced to a problem of optimal placement of a weighted graph on a line. This problem is known to be NP-hard for many common optimality criteria [GJ]; therefore to handle this problem, a common approach is to consider its special cases affording a polynomial-time solution.

This paper proves NP hardness of a refined special case of linear placement and proposes a polynomial-time algorithm for some classes of placement, based on dynamic programming approach.

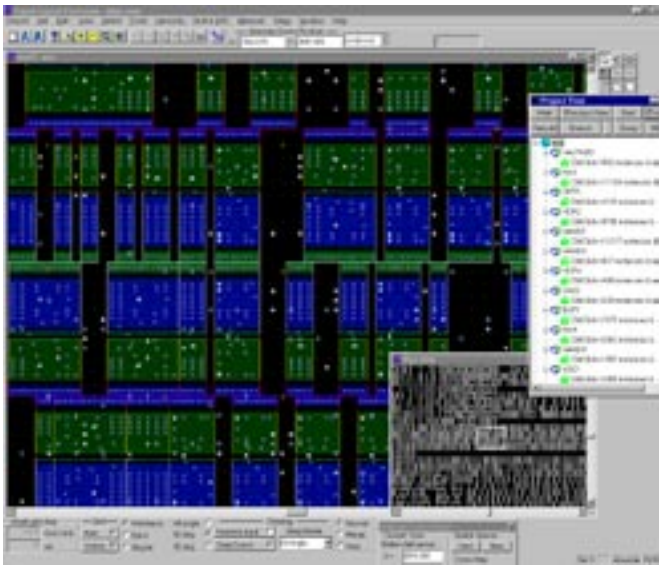

Figure 1. A standard-cell design (wiring hidden) in Expert.

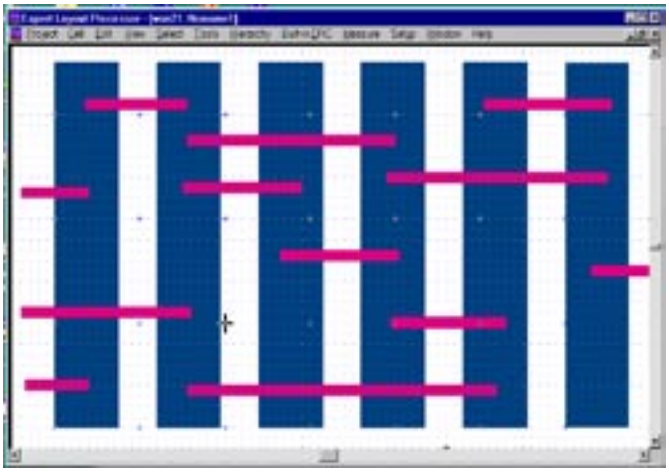

Figure 2. Standard cell linear placement model.

## Weighted Cycle Linear Placement Problem

Let *G(V,E)* be a graph with vertex set *V = V(G)* and edge set *E=E(G)* and let *n* be the number of its vertices, *n = |V|*. A graph G is said to be *edge-weighted* if a weighting function

W: E(G) --> R+

is defined upon its edge set. A one-to-one correspondence

F: V(G) --> {1,2,…,n}

is called *linear placement*, or *numbering*, of G. If we assume that V(G) = {1,2,…,n}, then f is a permutation from the symmetric group Sn over the set {1,2,…,n}; therefore in this paper the set of all numberings (linear placements) of an n-vertex graph will also be denoted by Sn.

This paper will consider the following criteria for placement:

Bw(G,f) = max{w(u,v)|f(u)-f(v)| : (u,v) \in E},

Tw(g,f) = max(1<=I<n) Sum(w(v,u)),

where the sum is over (v, u) \in E, such that f(v)<=I<f(u).

Lw(G,f) = Sum(w(u,v) |f(u)-f(v)|

where the sum is over (v, u) \in E.

The values Fw(G) = min{Fw(G,f) : f \in Sn}, where F is one of B, T, L, will be called the width, thickness, and length of G, respectively.

The problems of finding Bw(G), Tw(G), Lw(G) are known to be NP-hard, see [GJ], where the problems of finding B, L and T are called bandwidth minimization, optimal linear arrangement and min-cut linear arrangement, respectively. Polynomial-time algorithms for finding the length of an unweighted tree are known [2], [3]. On the other hand, the problem of finding the width of an unweighted tree is NP-hard even for the case when vertex degrees of a tree do not exceed 3, see [4]. An NP-hard problem is the one of finding the thickness of the weighted star graph K(1,n) (the PARTITION problem [1J] is easily reduced to it). Therefore polynomial algorithms are sought for narrower classes of graphs: special subclasses of trees [5], [6] and weighted trees [5]. This paper is devoted to the evaluation of computational complexity of linear placement of weighted cycle graphs.

Let Cn be a cycle graph with vertex set {1,2,..,n} and edge set {(1,2), (2,3),…,(n-1,n), (n,1)}.Without loss of generality, assume that the edge (n,1) of Cn has minimal weight. Then it is immediately seen that the placement f(i)=i, i=1,2,..,n, is minimal both with respect to its length and thickness. However this is not the case with respect to width.

**Theorem 1.** The problem of finding a minimum width linear placement of a weighted cycle is strongly NP-hard.

See the book [1] for the notion of strong NP-hardness.

**Proof.** We shall make use of the following strongly NP-complete recognition problem 3-PARTITION from [1].

**3-PARTITION:** Given a set A of size 3m, a positive integer threshold B and positive integer sizes s(a_i) for all elements a_i from A, such that

$$B/4 < s(a\_i) < B \text{ and } \mathrm{Sum}\,(s(a\_i)) = mB,$$

is it possible to partition A into m disjoint subsets A_1,.., A_m such that Σ(s(a_j)) =B, where the Σ is over all a_j in A_j for all j, 1≤ j≤m?

We shall demonstrate that this problem is pseudo-polynomially reducible to the following problem of recognition of the width of a weighted cycle under linear placement:

WEIGHTED CYCLE LINEAR WIDTH: Given a cycle Cn, weights (w_1,…, w_n) of its edges (1,2),…(n,1), and a positive integer K, does there exist a linear placement of this weighted cycle of width at most K?

It will be convenient to use the following intuitive casting of this problem. The cycle is considered as nonexpandable thread with knots which correspond to the vertices of the cycle graph. The length of the thread between knots j and j+1 (edge (j,j+1)) is equal to L_j = K/w_j for all j=1,…,n. The question is whether it is possible to place this thread along the line in such a way that the knots populate all points 1,2,…,n, see Figure 3. Notice again that our thread can bend, but cannot expand.

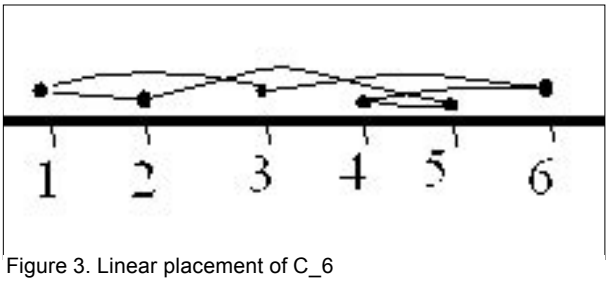

Figure 3. Linear placement of C_6

The required reduction will be constructed as follows. For each instance of the 3-PARTITION problem we shall take the thread with 3mB+3m+3 = 3M knots with the following edge lengths:

L_j =1 for j in intervals [1, M-1], [M+1, M+s(a_1) -1],

[M+s(a_1)+1, M+s(a_1)+s(a_2)-1],…,[M+mB+1, 2M+mB-1] .

L_j = mB+m for j = M, M+s(a_1), M+s(a_1)+s(a_2),…

L_j = mB+m+1 for j = 2M+mB and j = 3M.

The chains of knots [1,…,M], [M+1,…, M+s(a_1)],…, [M+mB+1,…, 2M+mB] (i.e., the chains of edges of length 1) will be called rigid, because in any feasible placement they must occupy consecutive positions. Further, the first and the last rigid chain may occupy only the starting and the ending segments of the segment [1, 3M], i.e., all positions [1,…,M] and [2M+1,…, 3M], because our thread has only two sufficiently large edges that may pass over these chains. As a result, the subchain of m edges of lengths B+1 may be placed in the unique way, leaving m groups of consecutive positions unoccupied. Each of the remaining rigid chains with lengths s_j, j=1,…, 3m, may be placed in any of these groups, thanks to the sufficiently long edges between them. Now it is easily seen that an instance of 3-PARTITION problem is solvable if and only if the corresponding thread has a linear placement.

The remaining three conditions for a reduction to be pseudopolyniomial [1J] are readily verified. It is also easily seen that WEIGHTED CYCLE LINEAR WIDTH problem belongs to class NP, hence the theorem is proved.

## Polynomial Time Solvable Special Case

A placement f of a graph G is said to be k-layer, if

$$T\_1(G,f) = k.$$

It is easy to see that

$$T\_1(G,f) = \max |\{(v,u) \in E(G) : f(v) \le j < f(u)\}|,$$

where maximum is taken over all j, $1 \le j < n$. For example, a placement at Fig. 3 is a 4-layer one. Notice that the mentioned earlier minimum-length and minimum-thickness placements of a weighted cycle is two-layer. Let us consider the problem of finding the minimum-width two-layer placement of a weighted cycle.

The following statement is evident.

**Lemma.** Let f be a two-layer placement of a cycle C_n. Then for any number k, $1 \le k < n$, the vertices with numbers 1, 2, …, k (and similarly, k+1,… n) generate a chain and and the vertex placed at k (at k+1, respectively) is the endpoint of the chain.

In what follows, "+" and "-" will denote addition and subtraction modulo n, respectively.

Let (v-1,v) and (u, u+1) be edges of C_n (v = u is a possible case). After the deletion of these two edges the cycle is decomposed into two chains, denoted by c1 = (v, -u) and c2 = (u+1, -v-1). Let the first one has k vertices. Let further

$$S((v,a)(u,b)(u+1, c)(v-1, d))$$

Denote the set of all two-layer numberings (i.e., linear placements) of C_n in which the vertices of the chain (v, -u) are numbered by 1,2,…, k, and the vertices v, u, u+1, v-1 are numbered by a,b,c,d, respectively. The Lemma above implies that there are only 2(k-1) admissible pairs of numbers (a,b) and 2(n-k-1) admissible pairs (c,d).

Let S((v,a)(u,b)) and S((u+1, c)(v-1, d)) denote the sets of numberings of chains c1 and c2 such that for any

$$h \in S((v,a)(u,b)) \text{ and } g \in S((u+1, c)(v-1, d))$$

the numeration f of C_n such that f(j) = h(j) if j is in c1 and f(j) = n-g(j) + 1, if j is in c2 belongs to S((v,a)(u,b)(u+1, c)(v-1, d)).

Denote

$$B2\_w((v,a)(u,b)) = \min \max w(j, j+1) |f(j)-f(j+1)|,$$

where the minimum is over all f from S((v,a)(u,b)) and the maximum is over $v \le j < u$. We can derive recurrent formulae for the functional B2_w by unfolding the max expression at point a. then the width of minimum-width two-layer placement of a weighted cycle is

$$B2\_w(C\_n) = \min\_\{(v, -u)\} \min\_\{a,b,c,d\} \max\{n1, n2, n3, n4\},$$

where n_1 = B2_w((v,a)(u,b)), n_2 = B2_w((u+1,c)(v-1,d)), n_3= w(u,u+1)(c-b+floor(n/2)), n_4= w(v-1,v)(d-a+floor(n/2)), and the minimum is sought over all chains (v,-u) with the number of vertices floor(n/2) and over all quadruples a,b,c,d. Given the boundary conditions B2_w((j, 1)(j, 1)) = 0, the value B_2(w(C_n) may be found by the dynamic programming approach. Time complexity of the evaluation of the table of values B(()()) using recurrent relations is $O(n^4)$. It is necessary to enumerate n chains and $n^2$ admissible quadruples in order to evaluate B2_s(C_n) by the formula above, so its time complexity is $O(n^3)$. In total the time complexity of the dynamic programming algorithm for finding minimum-width linear two-layer placement of a weighted cycle is $O(n^4)$.

Finally, notice that there are cycles for which minimum-width linear placement is not two-layer. An example is the 14-cycle:

(12, 14, 84,84, 84,84, 21, 84, 42, 21, 84,84, 84,84).

Its minimum-width linear placement is (1, 8, 14, 13, 12, 11, 10, 6, 7, 9, 5, 4, 3, 2) of width 84.

### References


[1] [GJ] M. R. Garey, D. S. Johnson, Computers and Intractability, 1978.

[2] [GK] M. Goldberg, I.Klipker, Notices of Acad. Sci. of Georgian SSR, 1976, Vol. 81, No. 3, 553--556.

[3] [C84] Chung F.R.K. , Computers and Math., with Appl., 1984, Vol.10, No.1, 43--60.

[4] [GGJK] M.R. Garey, R.L. Graham, D.S.Johnson, D.E.Knuth, SIAM J. Appl.Math., 1978, Vol. 34, No.3, 477--495.

[5] [L85] Lepin V.V., Notices of Belarusan Acad. Sci., Ser. Match Phys. 1985, No.2, 16--21.

[6] [SZ82] M.M.Syslo, J.Zak, Ann. Descrete Math., 1982, Vol.16, 281--286.

# Calendar of Events

## June

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15 DAC 98 - San Francisco, CA
16 DAC 98 - San Francisco, CA
17 DAC 98 - San Francisco, CA
W/S - Munich, Germany
18
19
20
21
22
23
24 W/S - Grenoble, France
W/S - UCLA, Los Angeles, CA
25 W/S - UCLA, Los Angeles, CA
26
27
28
29
30 W/S - Guildford, UK

## July

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

## Bulletin Board

*Maverick, Dragon and Guardian*
*New Products in CELEBRITY Framework*

Three new and exciting products have been added to the ***CELEBRITY*** framework of CAD products:

- ***Maverick*** - Hierarchical Netlist Extractor
- ***Dragon*** - Hierarchical DRC
- ***Guardian*** - Hierarchical LVS

These products combined with ***Expert*** and ***Savage*** provide an integrated design environment for custom IC layout and design on NT.

*Silvaco signs COMPAQ's Solutions Alliance (CSA) Program*

In April 1998 Silvaco has joined COMPAQ's Solution Alliance Program. This membership represents a very crucial milestone for Silvaco as it will allow the company to reap the benefits of marketing, sales and support of this large company. The primary mutual interests are parallel NT ***SmartSpice*** and ***CELEBRITY*** group of products.

*Successful DAC '98 Tradeshow*

DAC '98 tradeshow was a complete success. The customer's interest in all products, old (TCAD) and new ***(CELEBRITY)*** was equally strong. Participation from European customers was noticeably strong. At the Exhibitors Presentation on Monday June 15th, several new products were introduced. Parallel ***SmartSpice*** on NT (multi threaded) was touted as the first of its kind (first parallel NT simulator).

*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: Can I draw tangents to circles in *Expert*?**

A: **Expert** supports several options for drawing sophisticated shapes. There are three ways to accomplish this task.

(1) If your circles are of the same radius, you may draw a single-segment wire with end style set to ROUND. In addition, you may convert the wire to polygon.

(2) For drawing any types of circles connected by common tangents, try Rolled Region creation (program must be in all-angle mode). In this mode one can create complicated forms consisting of lines and circular arcs. However, you could refer to the user manual, Sect. 5.3.4.

(3) You may create any sophisticated shape using xi (eXpert Interface) language, and even parameterize it. Below is a simple example of spiral polygon drawing written in Xi.

```
do begin
      seq = {};
      xx0 = x0;
      yy0 = y0;
      xx1 = (xx0); yy1 = (yy0);
      SEQ_ADD_LAST(seq, (xx1));
      SEQ_ADD_LAST(seq, (yy1));

      i = 0;
      loop begin
          if (i EQL len) then (leave loop);
          dSQ = sqrt(xx0 * xx0 + yy0 * yy0) ;
          xx1 = xx0 + arc * yy0/dSQ;
          yy1 = yy0 - arc * xx0/dSQ;
          xx2 = xx0 + wid * xx0/dSQ;
          yy2 = yy0 + wid * yy0/dSQ;
          xx0 = xx1;
          yy0 = yy1;

          SEQ_ADD_LAST(seq, (xx0));
          SEQ_ADD_LAST(seq, (yy0));
          SEQ_ADD(seq, 1, (xx2));
          SEQ_ADD(seq, 2, (yy2));
          i=i+1;
       end;

       xx2 = xx0 + wid * xx0/dSQ;
       yy2 = yy0 + wid * yy0/dSQ;
       SEQ_ADD(seq, 1, (xx2));
       SEQ_ADD(seq, 2, (yy2));

       polygon (seq);
end;
```

**Q: Could I change the layer of an object?**

A: There are three ways to do this:

(1) If you want to do this to a single object, set "Numeric Input" option on, select "Modify" tool and pick the required object. In the Numeric Input panel you will see the drop-down list, where you may select the required layer

(2) If you must put several objects into one layer, you may do the following:

- select an object (or several objects);
- make the layer into which you want to move the object active;
- apply menu command: Select>>Save Selected>>To Layer

  This will copy the selected objects into the current layer. (the objects remain selected in the old layer)
- Delete selected.

(3) A slightly different routine for multiple objects:

- select an object (or several objects);
- apply Edit>>Cut command (e.g., using "Cut" button, labeled with scissors);
- make the layer into which you want to move the object active;
- apply Edit>>Paste into Layer command.

For multiple objects routine (3) is easier (less mouse clicks). However, when there are many objects, it is executed slower since it takes some time to copy objects into buffer.

A key advantage of routine (3) is that it allows the user to copy between windows, or between cells.

### 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**

*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*