Simulation Standard

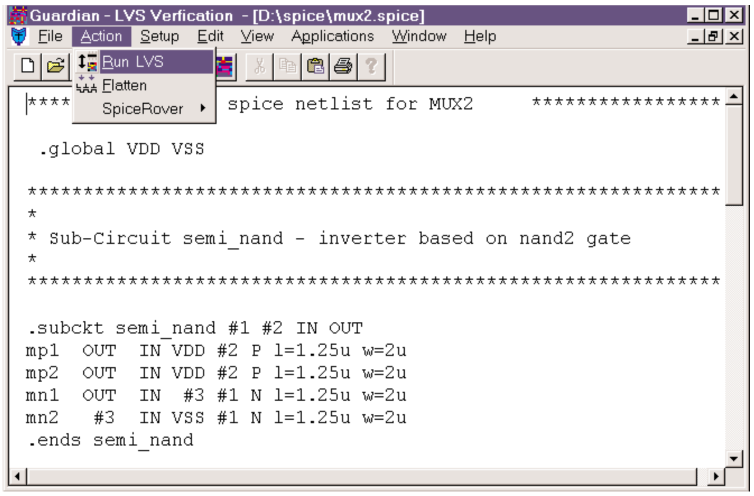
TCAD Driven CAD

A Journal for CAD/CAE Engineers

Introducing *Guardian* - LVS Verification for PC-based Platforms

Introduction

Guardian is a (state-of-the-art) hierarchical netlist comparison system, which eliminates many of the disadvantages of existing programs. Running on PCs under Windows NT, it easily compares circuits with a large number of devices. The advanced algorithms implemented in ***Guardian*** allow a substantial reduction of execution time and also detect discrepancies between two netlists more precisely. ***Guardian*** generates a comprehensive hierarchical report, which is easy to read. An embedded tool, called the Spice Netlist Rover, links report files with source netlists to make inspecting correct matches and errors simple.



```
|****                 spice netlist for MUX2      *****************

 .global VDD VSS

*****************************************************************
*
* Sub-Circuit semi_nand - inverter based on nand2 gate
*
*****************************************************************

.subckt semi_nand #1 #2 IN OUT
mp1  OUT  IN VDD #2 P l=1.25u w=2u
mp2  OUT  IN VDD #2 P l=1.25u w=2u
mn1  OUT  IN  #3 #1 N l=1.25u w=2u
mn2   #3  IN VSS #1 N l=1.25u w=2u
.ends semi_nand
```

Figure 1. Running LVS for open spice file.

Guardian Functionality

Guardian is used for comparing two circuits described by their netlists. Most often one of these netlists corresponds to the schematic of a circuit produced by a schematic editor. It represents the logic of the circuit. The other one is extracted from mask layout by ***Maverick*** netlist extractor. It represents the actual physical layout of the circuit as produced by ***Expert*** layout editor. This kind of circuit verification is called LVS (layout versus schematic comparison). It is not uncommon to compare two schematic netlists (SVS, or schematic versus schematic) or two layout netlists (LVL, or layout versus layout).

Guardian compares circuits, which may contain both primitive devices (flat circuits) and cell or hierarchical blocks of different kinds (hierarchical circuits).

Guardian does not require any identification marks in both netlists to start comparison. However, it takes advantage of such marks if present, allowing comparison with initial correspondence node pairs defined in the prematch file.

Guardian uses Spice netlist format for comparison. Spice netlists describe circuits at the device level i.e. in terms of transistors, diodes, resistors and capacitors.

If a spice netlist contains only subcircuit definitions but neither subcircuit calls nor element statements, one of these subcircuits serves as a main subcircuit. In fact, this subcircuit will contain other subcircuit calls and / or element . In this case, the main subcircuit will be treated as the top-level subcircuit.

If there are subcircuit calls and / or element statements in the spice netlist, they will be considered as elements of the top level subcircuit. The default subcircuit name "top" will be used for it.

The basic flow of the netlist comparison is as follows.

Continued on page 2....

INSIDE

Volume 10, Number 3, March 1999

SILVACO INTERNATIONAL

First of all, ***Guardian*** performs netlist transformation. At this stage, series and parallel devices are merged and unused devices are filtered out.

After that ***Guardian*** performs logic gate recognition and reduces groups of transistors into "boxes" with permutable (interchangeable) groups of terminals.

Finally, the actual comparison takes place, during which ***Guardian*** attempts to match elements of the two netlists against each other.

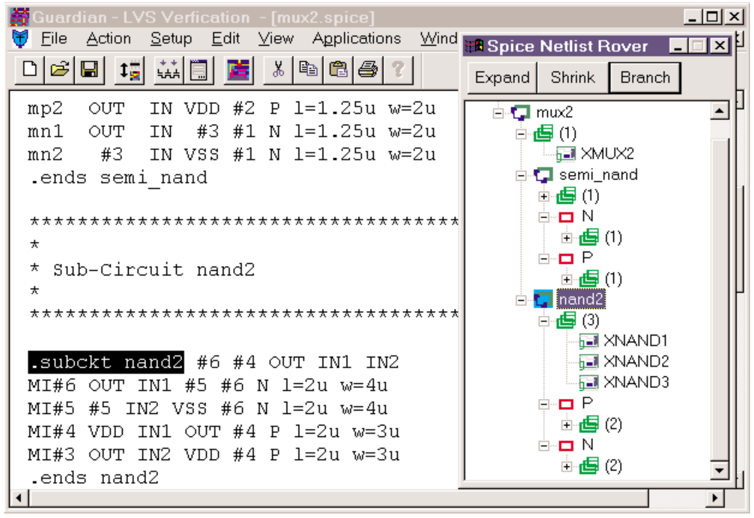
Both preprocessing and comparison are controlled by various options and conditions. Some options may be specified through the user interface, others are specified by control files. An example of the latter case is the list of prematched nets, devices, or subcircuits.

The results of netlist comparison (filtered, merged and reduced devices, matched pairs, and discrepancies) are stored in report files. While reporting unmatched nodes ***Guardian*** gives suggestions about node pairs that could be correct matches if discrepancies between netlists were fixed.

Running *Guardian*

Using ***Guardian*** is very simple and convenient. There is an embedded text editor, which allows viewing of netlists and reports. You can start netlist comparison while viewing an open spice file by selecting the **Run LVS** menu command. A dialog box prompting the user to choose the second spice file for the comparison will appear.

The second way to start comparison is to use **Run LVS** menu command when there is no open spice file. The **Guardian Settings** panel will open for selecting input files and options.

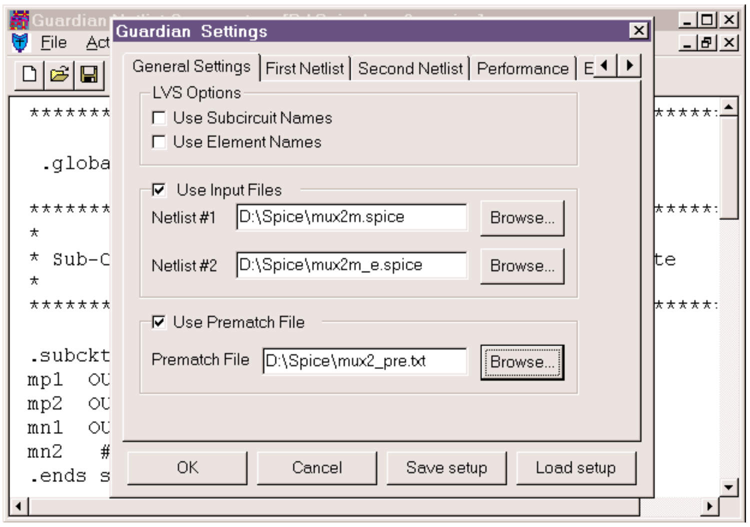


```
mp2   OUT  IN VDD #2 P l=1.25u w=2u
mn1   OUT  IN   #3 #1 N l=1.25u w=2u
mn2    #3  IN VSS #1 N l=1.25u w=2u
.ends semi_nand

*************************************
*
* Sub-Circuit nand2
*
*************************************

.subckt nand2 #6 #4 OUT IN1 IN2
MI#6 OUT IN1 #5 #6 N l=2u w=4u
MI#5 #5 IN2 VSS #6 N l=2u w=4u
MI#4 VDD IN1 OUT #4 P l=2u w=3u
MI#3 OUT IN2 VDD #4 P l=2u w=3u
.ends nand2
```

Figure 3. Spice Netlist Rover in action.



LVS Options
- [ ] Use Subcircuit Names
- [ ] Use Element Names

[✓] Use Input Files
Netlist #1 D:\Spice\mux2m.spice
Netlist #2 D:\Spice\mux2m_e.spice

[✓] Use Prematch File
Prematch File D:\Spice\mux2_pre.txt

Figure 2. Main page of Guardian Settings

All options for running LVS are set through the user interface. The only exception is for defining any initial corresponding node pairs in the prematch file.

There is an option to save all current settings into the context file. Saved settings can be restored at any time from the file for use in comparison.

Netlist Transformation

While preprocessing ***Guardian*** can optimize input netlists in order to reduce the number of devices and to replace groups of primitive devices by logical gates. Netlist transformation has the following steps:

- All unused devices such as MOS transistors with shortened drain and source, devices with floating pins, MOS transistors with gate connected to ground or power nets are filtered out.
- Devices of the same type connected in parallel or in series are merged into a single element. MOS transistors connected in series must have a common gate.
- Groups of transistors which represent most common logic gates (inverter, NAND, NOR, etc.) are recognized and replaced with "black boxes" with permutable terminals.
- Groups of transistors, which do not form common logic gates, are combined together into a single element with permutable terminals as well.
- Two terminal devices (resistors, capacitors, etc.) connected in parallel or in series but found in different cells (even at different levels of hierarchy) are across-hierarchically reduced into a single device.

All components of netlist transformation listed above are optional and can be switched off by the user.

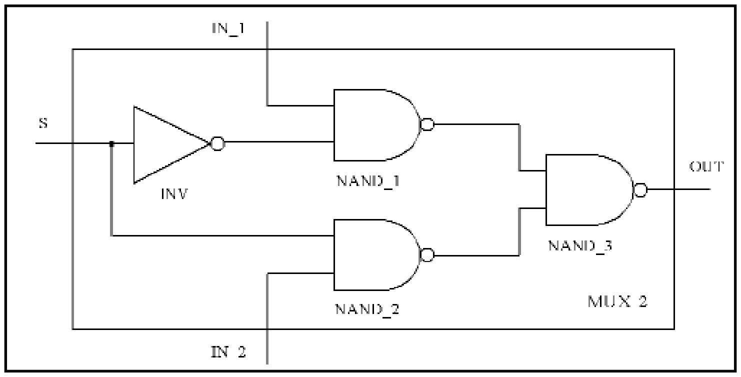


Figure 4. Two input multiplexer.

Advanced *Guardian* Comparison Algorithm.

The mathematical aspects of the netlist comparison problem have been extensively studied, and many different algorithms have been developed. Most of them are based on classical graph isomorphism (GI) techniques. These techniques require flattening compared netlists, assigning labels to nodes (devices and nets) and then refining the labels according to the node neighborhood. As a result, GI-based algorithms compare successfully only simple circuits (netlists) with little or no symmetry. On the other hand, they have several serious disadvantages when applied to realistically large circuits with high symmetry. In particular, the loss of information about circuit hierarchy as well as circuit symmetry may tremendously increase execution time. Furthermore, algorithms using GI methods cannot prevent single error propagation, which results in the generation of many spurious errors far from the original one.

The power of ***Guardian*** is a result of using advanced, fast search–oriented subgraph isomorphism techniques. It helps overcome drawbacks associated with standard GI–based algorithms. In other words, the ***Guardian*** comparison algorithm combines both the graph isomorphism and the special subgraph isomorphism methods and dynamically switches between them depending on an analysis of the validation process. Briefly, the ***Guardian*** algorithm can be presented as follows. The partitioning techniques are used as the primary basis of the algorithm. Compared circuits are represented by bipartite graphs with devices and nets as graph nodes. These nodes are partitioned into classes based on assigning labels according to node invariants. Obtained labels are repeatedly refined using the principles of local and global matching and some kinds of heuristics. This avoids avoiding label collisions and leads to singular partitions with high probability.

However, the basic refinement algorithm has some disadvantages. For example, in the presence of symmetry the relabeling process requires too many iterations to obtain unique labels. Further, in some cases there is no sufficient information available for label recalculation. It leads to creating non-singular partitions. Combination of the above GI – based algorithm with special fast search – oriented subgraph isomorphism (SGI) techniques helps avoid such problems. First of all, these SGI techniques are used to form original patterns of gate level subcircuits and then quickly find and match these patterns in both netlists anywhere it is possible. If the matching of whole patterns is impossible owing to errors, we dynamically form pseudo gates and match them to the most appropriate original gate patterns. Such a pseudo matching procedure avoids label collisions and error spreading, enhancing error classification. Moreover, the more detailed analysis with pseudo matched gates often allows matching nodes in a graph "region" neighboring error node. If pseudo matching was erroneous, it does not lead to extra errors due to analysis of netlist hierarchy.

Thus, the advanced ***Guardian*** comparison algorithm yields comparison results that are superior comparing with other algorithms.

Flatten Function

Sometimes a user may need to work with a flat rather than hierarchical netlist. One of the features of ***Guardian*** is a possibility to convert hierarchical netlists into flat ones. The flatten function performs this task. The resulting netlist is stored in a spice format file with comments, which explain the conversion details. Device and net names in the resulting flat netlist contain information about the whole path from the top level of circuit hierarchy to the node.

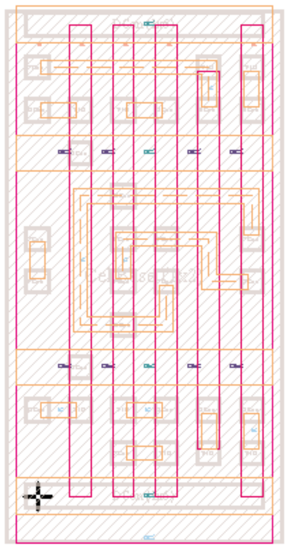
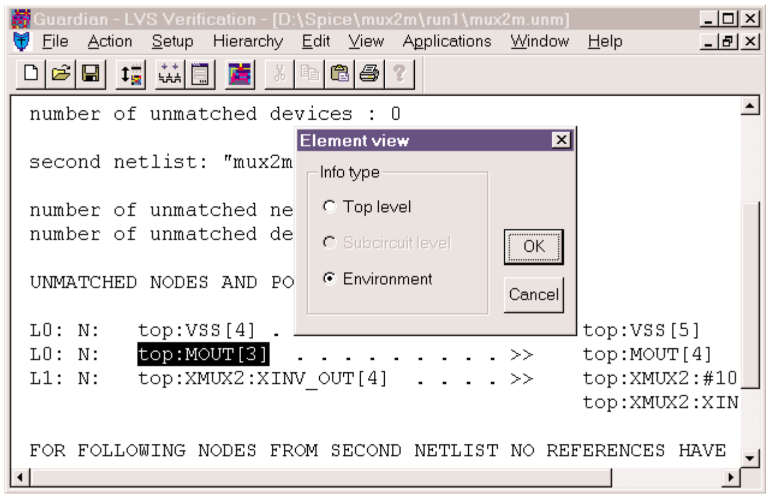


Figure 5. Layout display for two-input multiplexer.


Guardian - LVS Verification - [D:\Spice\mux2m\run1\mux2m.unm]
File Action Setup Hierarchy Edit View Applications Window Help
Element view
OK
Cancel

```
number of unmatched devices : 0

second netlist: "mux2m

number of unmatched ne
number of unmatched de

UNMATCHED NODES AND PO

L0: N:   top:VSS[4] .                              top:VSS[5]
L0: N:   top:MOUT[3]  . . . . . . . . . . >>       top:MOUT[4]
L1: N:   top:XMUX2:XINV_OUT[4]  . . . . >>         top:XMUX2:#10
                                                   top:XMUX2:XIN

FOR FOLLOWING NODES FROM SECOND NETLIST NO REFERENCES HAVE
```

Info type: ( ) Top level ( ) Subcircuit level [✓ Environment]

Figure 6. Report browsing.

Integrated Spice Netlist Rover

One of the advanced features of ***Guardian*** is the Spice Netlist Rover that is seamlessly integrated within it. Spice Netlist Rover is a tool for navigating through the hierarchy of the netlist represented in the form of a tree. In addition to hierarchy tree navigation, Spice Netlist Rover provides a convenient visual supplement for other operations that are somehow related to netlist hierarchy. It highlights selected cell instances, visualizes the results of the LVS verification (e.g. coloring matched and unmatched instances). It also allows the user to specify and show instances placed anywhere in the netlist's hierarchy, and to find and select a branch of any instance from the spice netlist.

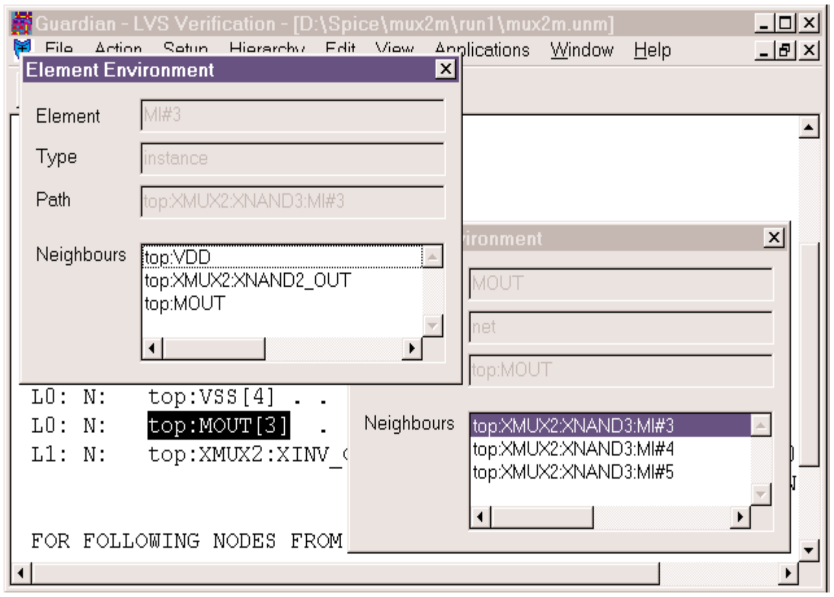
Depending on the Spice Rover actions the netlist hierarchy is shown in two different forms: Basic Tree View and Branch Tree View.

Having two forms of view is essential in case of "cell hierarchy", when a circuit consists of hierarchically nested cell instances. When hierarchy becomes deeper and more instances of the same cell appear, the hierarchical tree becomes enormous and incomprehensible.

For this reason Basic Tree View branches out only along cells. The instances of the same cell in a common parent cell are grouped together and do not branch further.

Spice Rover performs the following operations between the hierarchy tree and the spice netlist:

- If you double click on a cell item you switch to the window containing the spice netlist at the place where this cell is defined.


Guardian - LVS Verification - [D:\Spice\mux2m\run1\mux2m.unm]
File Action Setup Hierarchy Edit View Applications Window Help
Element Environment
ironment

Element: MI#3
Type: instance
Path: top:XMUX2:XNAND3:MI#3
Neighbours: top:VDD
top:XMUX2:XNAND2_OUT
top:MOUT

MOUT
net
top:MOUT
Neighbours: top:XMUX2:XNAND3:MI#3
top:XMUX2:XNAND3:MI#4
top:XMUX2:XNAND3:MI#5

```
L0: N:   top:VSS[4] . .
L0: N:   top:MOUT[3]  .
L1: N:   top:XMUX2:XINV_(

FOR FOLLOWING NODES FROM
```

Figure 7. Sequential viewing element's neighbors.

- If you double click on a cell instance you switch to the window with the spice netlist and highlights the place where this cell is called.
- Double clicking on a device highlights a device statement in the spice netlist.

Spice Netlist Rover may show several trees:

Active Cell Tree shows the hierarchy of the cell in the current active window.

Project Tree shows the hierarchy of the whole netlist.

Branch Tree allows Spice Rover to access cell instances laying deep in the hierarchy of the active cell.

Advanced Hierarchical Report

When hierarchical netlists are being compared, the output files are desired in hierarchical form as well. It is also very important not only to have a list of unmatched nodes, but to also point out causes of these discrepancies. Unlike other LVS systems, ***Guardian*** performs both tasks.

Guardian reports matched node pairs and unmatched nodes starting from the top-level subcircuit, which corresponds to the zero hierarchy level. After that nodes from the most complex subcircuit, which belongs to the top level, are reported, and so on. As a result all reported nodes will be composed of a hierarchical tree. Increasing level numbers reflects a moving down in the hierarchy. Since the netlists being compared may have a different hierarchy structure, the level number is defined according to the first netlist hierarchy structure.

In most cases netlists differ from each other. In this case ***Guardian*** tries to match elements of one netlist to elements of the other as often as possible. An advanced approach is used to classify discrepancies and locate the errors. Information about the node environment is collected during the comparison. It is used in an attempt to find node pairs that could have matched.

As a part of the ***Guardian*** report, a file will list all nodes that have not been matched. This file contains three sections:

Unmatched nodes and possible matches

This section contains unmatched nets and devices from the first netlist that have unique neighborhoods. However no corresponding net / device is found in the second netlist. Wherever it is possible the matching algorithm gives a reference to a node (or nodes) from second netlists that did not match because of having different features (different device type or net degree). If a net from the first netlist has more than one possible match in many cases it may indicate a broken connection in the second netlist. For some nodes no references may be found. Missing devices in the second netlist might cause such an outcome.

Following group(s) of unmatched nodes might have appeared in the case of missing corresponding node(s) in either netlist

In this section, groups with the same feature nodes in both netlists are listed. Node features are defined by node environment. Nodes inside a group are similar and can not be distinguished from one another (e.g. resistors connected in parallel or nets with similar connections). The number of nodes in corresponding groups is always different because of discrepancies between two netlists. For this reason the algorithm is not able to match nodes from one group to nodes from the other. In many cases such groups might result from missing devices or nets in either netlist.

For following nodes from the second netlist no references have been found

This section reports all unmatched nets and devices from the second netlist that have not been listed in the previous two sections. No references from the first netlist were found by the matching algorithm. In many cases these nodes are missing in the first netlist.

An example of the unmatched report file is given for two input multiplexer:

```
* * * * * * * * * * * * * * * * * * * * * *
*    GUARDIAN UNMATCHED NODES REPORT    *
* * * * * * * * * * * * * * * * * * * * * *

first netlist: "mux2m"

number of unmatched nets    : 3
number of unmatched devices : 0
second netlist: "mux2m_e"

number of unmatched nets    : 4
number of unmatched devices : 1

UNMATCHED NODES AND POSSIBLE MATCHES:

L0: N: top:VSS[4]   . . . . . >> top:VSS[5]
L0: N: top:MOUT[3]  . . . . . >> top:MOUT[4]
L1: N: top:XMUX2:XINV_OUT[4] >> top:XMUX2:#10[2]
                              top:XMUX2:XINV_OUT[2]

FOR FOLLOWING NODES FROM SECOND NETLIST NO REFERENCES
HAVE BEEN FOUND:

L?: R: No reference found  . >> top:RLOAD

* * * * * * * * * * * * * * * * * * * * * *
```

All three sections may not appear in every the unmatched file as it depends on discrepancies between the two netlists. However the first section is generated whenever any errors have been detected.

Guardian Report Browsing

Guardian has an efficient viewer which links output files to netlists and makes easy to browse reports.

If one double clicks on a (hierarchical) name of an instance or a net in the ***Guardian*** match or discrepancy file, this name will become highlighted and the **Element View** panel will appear as shown at Figure 6.

Depending on where a node is located in the netlist (top level or subcircuit) a corresponding viewer option will be available. Two ways of browsing are available:

- If you select Top level or Subcircuit level option and press OK button, the corresponding spice netlist will be opened and the location of first appearance for this node name will be highlighted. This place will be either at the top level of netlist or in the subcircuit.
- If you select Environment option and press the OK button, all node neighbors can be inspected. The panel Element Environment pop-up will appear.

This panel shows the name, type, full path of the selected node and the list of its neighbors in the hierarchical netlist. All neighbors of a device are nets. All neighbors of a net are devices.

Now you can "travel" through the entire netlist by double clicking on one of the element neighbors. Each time a similar panel will appear for that neighbor.

Advanced Pairwise Merging Algorithm for VLSI Floorplanning

Introduction

This paper concerns the problem of determining optimal placement of rectangular blocks within a rectangular area known as the packing or cutting-stock problem. This problem arises at then floorplanning stage of VLSI design.

Assume that one is given n different types of rectangular blocks $\mathbf{R_0}, \mathbf{R_1}, \dots \mathbf{R_{n-1}}$, where $\mathbf{R_i}$ is rectangle $\mathbf{h_i} \mathbf{x} \mathbf{w_i}$. The goal is to find non-overlapping placement of the blocks within rectangular area **A** so that uncovered part of the area is as small as possible. The rectangle **A** is referred to as *target area*. Without loss of generality one can assume that a block of any type $\mathbf{R_i}$ can fit in the target area. This problem is referred to as *packing problem*. The problem is also known as *cutting stock* problem. (It can be easily reformulated as problem of cutting rectangular sheet of material into rectangular pieces. We will use this name to refer to the problem as well.)

If some resultant placement **P** contains $\mathbf{c_i}$ blocks of type **i**, square of a block of type **i** is $\mathbf{S_i} = \mathbf{h_i} * \mathbf{w_i}$ and target area **A** is a rectangle with dimensions **h** and **w**, then square of uncovered part of **A** can be obtained from the following formula:

$$TW_P = h * w - c_0 * S_0 - c_1 * S_1 - \dots - c_{n-1} * S_{n-1}$$

Value TW_P is called *total waste* of placement **P**. Placement **P** is considered optimal if and only if $TWP \leq TW_Q$ for any other placement **Q**.

In general case optimal solution is very difficult to find and requires an algorithm based on exhaustive search approach. But in most practical cases a acceptable approximate solution can be obtained much faster and easier. There are number of purely heuristic algorithms designed to solve the packing problem which work very fast. There are also some heuristic modifications of exhaustive search algorithms, which substantially increase algorithms' performance at cost of quality of result.

Pairwise Merging Algorithm

One of the popular approaches here is the *pairwise merging technique.* An algorithm based on this technique works in the following manner. Starting with initial blocks it generates new ones by merging them in pairwise manner. Blocks in each pair are attached to each other without overlapping at different relative positions thus creating a number of combined blocks. Such combined blocks are referred to as clusters. If cluster still fits in target area it is called feasible cluster and participates in subsequent mergers to form larger clusters and so on. Note that any *feasible cluster* as well as any initial block alone represents more or less satisfactory solution of the problem. When the algorithm is not able to generate any new cluster the best solution found so far is reported as final result. The following is the sketch of algorithm based on pairwise merging technique:

```
S, L and N – sets of feasible clusters

feasible(X) – functions which returns set of all feasible
    clusters from set X, feasible(X)    X

merge(s, l) – operation which generates finite set of
    new clusters by merging clusters s and l at different
    relative positions.

Step 1. $S := \varnothing$; $L := \{R_1, R_2, \dots, R_{n-1}\}$

Step 2. $N := \text{feasible}((\cup_{s \in S, l \in L} \text{merge}(s, l)) \cup (\cup_{s \in L, l \in L} \text{merge}(s, l)))$

Step 3. if $N = \varnothing$ goto step 5

Step 4. $S := S \cup L$; $L := N$; goto step 2

Step 5. $P\text{-}^* = \arg\min_P TW_P$; output(P*)
```

To avoid generation of identical clusters again and again the algorithm performs merging (step 2) only in such pairs of clusters that consist of at least one cluster generated at previous iteration (set **L**). However, it doesn't guarantee that each cluster will be built only once. It is easy to show that in the general case a cluster consisting of more than two blocks can be built by pairwise merging operation in several ways. By elimination of repeatedly generated clusters from set **N** on step 2 of the algorithm its performance can be substantially increased.

Operation *merge*

The main obstacle in the way of practical implementation of this algorithm is merging operation (*merge* is the sketch above). A number of problems complicate the implementation of this operation and the whole algorithm. The most interesting of them is the following.

Any two geometrical objects, even pair of rectangles, can be merged into a new one in an infinite number of ways. While initial blocks are rectangles and are relatively easy to deal with on first iterations of the algorithm, clusters that appear later can have very complex shapes. From the other side, it is easy to see that all initial blocks and intermediate clusters are isothetic objects. There are theoretical results which show that in order to find an optimal solution in this case we can limit ourselves to consideration of a finite number of relative positions of such objects – so called *contact concurrent positions* [4]. However, the number of such positions can still be very large, which results in generation of large amounts of new feasible clusters. In practice the amount of intermediate data generated by the algorithm can easily overflow memory in any modern computer. Not to mention that algorithm's iteration time grows at a very high rate.

A simple way to reduce the number of variants generated by merge function is the following. First, the particular class of geometrical shapes **H** is chosen and operation *merge* is defined as taking two parameters from class **H** and returning a subset of class **H**:

$$\text{merge}: (\mathbf{H}, \mathbf{H}) \rightarrow 2^{H}$$

It means that each variant of merging of two objects form class **H** is approximated by shape from class **H.** Of course, such approximation will generally increase the square of resultant clusters by introducing some unused dead space into them. Square of dead space within cluster **c** is called *internal waste* of the cluster – $\mathbf{IW_c}$. It means that total waste $\mathbf{TW_P}$ for any placement **P** generated by algorithm which uses such *merge* operation will consist of two components: external waste $\mathbf{EW_P}$ – square of target area not covered by **P** – and internal waste $\mathbf{IW_P}$ introduced in **P** by all merge operations that built **P**.

$$\mathbf{TW_P} = \mathbf{EW_P} + \mathbf{IW_P}$$

Now when clusters can contain internal waste the notion of cluster domination can be introduced. Consider two clusters s and t, each one consisting of cs0, cs1, … , csn – 1 and ct0, ct1, … , ctn – 1 blocks of each type respectively. Cluster s is said to be dominating over cluster t if and only if the following two conditions are met (see Figure 1):

1. $\mathbf{c}^{s}_{i} \geq \mathbf{c}^{t}_{i}$, for any $\mathbf{i} = 1, 2, \dots, \mathbf{n}$;
2. cluster **s** can be enclosed within cluster **t**.

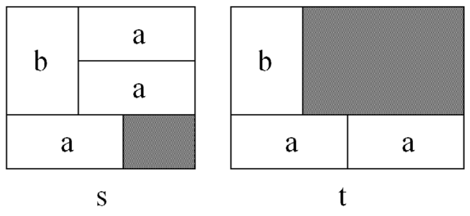


Figure 1. Domination: cluster s dominates over cluster t

When some cluster **s** is generated at particular iteration of the algorithm, all previously built clusters **t**, such that cluster s dominates over cluster t, can be discarded without compromising the quality of final solution. In fact, in any placement that contains cluster **t** it can be naturally replaced with cluster **s**. Such replacement can only increase the quality of the solution. Note that according to definition of domination relation any cluster s dominates over itself. It means that by discarding dominated clusters the algorithm discards identical clusters as well. The discarding results in substantial decrease in number of clusters built by the algorithm thus reducing the number of variants to be checked and increasing the overall performance of the algorithm.

Heuristic Criteria for Cluster Discarding

Amount of internal waste in cluster can be used in two simple heuristic criteria which significantly reduce number of variants processed by the algorithm [2]. The main idea here is to detect and discard intermediate clusters that are unlikely to be part of optimal solution. The criterion is applied to clusters generated by merge operation. If a cluster meets the criterion it participates in subsequent mergers, otherwise the cluster is discarded because of high percentage of internal waste. The first criterion is formulated as follows:

$$\mathbf{IW_c} \leq \mathbf{B_a} * \mathbf{H} * \mathbf{W}, \quad 0 \leq \mathbf{B_a} \leq \mathbf{1}$$

This criterion is set to be *absolute criterion*. Coefficient $\mathbf{B_a}$ is parameter of the algorithm. The first criterion is formulated as follows:

$$\mathbf{IW_c} < \mathbf{B_r} * \mathit{square}(\mathbf{C}) , \quad 0 < \mathbf{B_r} < \mathbf{1}$$

This criterion is set to be *relative criterion.* While relative criterion discards "unpromising" clusters evenly on all iterations of the algorithm, absolute criterion tends to be more tolerant to clusters generated on first iterations.

Obviously, these criteria are heuristics, and on particular input data they can "cut off" optimal solution of the problem. But with algorithms described below they provide excellent results in terms of performance and quality of solution. There are also more complex and more precise criteria for discarding clusters that cannot be part of optimal solution [3]. Branch-and-bound technique can be naturally used in pairwise merging algorithms.

Wang's Algorithm

A good illustration to this approach is Wang's algorithm for two-dimensional cutting stock problem [2]. The algorithm uses class of rectangles with their sides parallel to coordinate axes as class of geometrical shapes **H.** In this case operation *merge* is easy to define. Two rectangles are be combined in two different ways called *horizontal* and *vertical* composition and after that resultant shapes are approximated by their bounding boxes (see Figure 2).

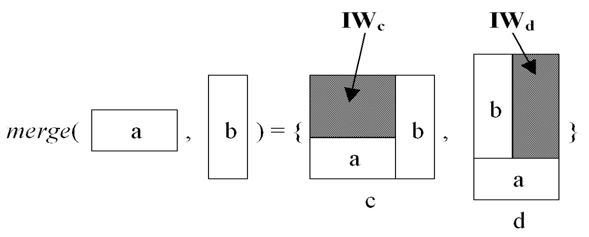


Figure 2. Horizontal and vertical composition

The obvious disadvantage with Wang's algorithm is that approximation with bounding box in many cases will substantially increase amount of internal waste in intermediate clusters generated by *merge* operation and, consequently, in final placement. Moreover, it is easy to show that generally this algorithm is not able to find optimal solution of the problem. All placements generated by Wang's algorithm are so called *guillotine* ones. In terms of cutting problem, guillotine placement allows cutting of rectangular sheet of material into rectangular pieces by consequential side-to-side cuts. Placement shown on Figure 3 is optimal for corresponding problem but is not guillotine one. It cannot be built by Wang's algorithm.

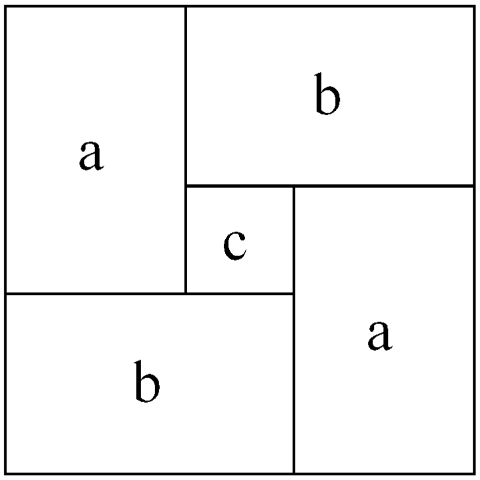


Figure 3. Non-guillotine placement

L-block-based Algorithm

In this paper an improved algorithm based on pairwise merging technique is introduced. The main difference in the algorithm is the class of so called L-blocks [1] is used as class H and operation merge is defined on class of L-blocks. There are five types of objects in this class (see Figure 4).

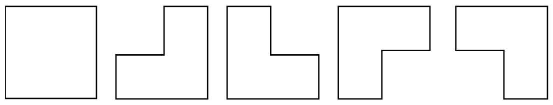


Figure 4. Five types of L-blocks

In sequence of classes of isothetic convexity **ICONVEX**^{0}, **ICONVEX**^{1}, **ICONVEX**^{2}, ... [5] class of rectangles and class of L-blocks occupy first two places (**ICONVEX**^{0} and **ICONVEX**^{1} respectively). Intuitively, the fact that these classes are found next to each other in this sequence means that L-blocks are "just a little more complex objects" than rectangles. Therefore one can expect that data structures and *merge* operation for L-blocks are not much more complex than for rectangles. Moreover, class of rectangles is included into class of L-blocks which means that quality of result of L-block-based algorithm can only be better than quality of result of Wang's algorithm on the same input data. Another important advantage of the algorithm is its ability to process input consisting of L-blocks as well as rectangles.

Operation merge on class of L-blocks can be easily defined through five elementary operations:

- $\neg$ – complement. This unary operation complements L-block to its bounding box (which is L-block too).
- $*_1$, $*_2$, $+_1$, $+_2$ – four types of build. These binary operations combine two input L-blocks to form a new one (see [1] for detailed description of these operations).

Completeness of this system of operations is proved by the following theorem [1].

Theorem. Let **L** be class of L-blocks and **f: (L, L) → L** is any function that combines pair of L-blocks without overlapping into a new L-block.

Then for any pair of L-blocks a and **b, a, b** ∈ **L**, either L-block (A ∘ B) or (B ∘ A), where **A** ∈ {**a**, ¬**a**}, **B** ∈ {**b**, ¬**b**}, ∘ ∈ {$*_1$, $*_2$, $+_1$, $+_2$}, can be enclosed within L-block **f**(**a, b**) (see [1] for proof).

Using this result one can build operation merge for the algorithm:

$$\begin{aligned} \text{merge}(\mathbf{s}, \mathbf{l}) = \{ \; & \mathbf{s} +_1 \mathbf{l}, && (\neg\mathbf{s}) +_1 \mathbf{l}, && \mathbf{s} +_1 (\neg\mathbf{1}), && (\neg\mathbf{s}) +_1 (\neg\mathbf{1}), \\ & \mathbf{s} +_2 \mathbf{l}, && (\neg\mathbf{s}) +_2 \mathbf{l}, && \mathbf{s} +_2 (\neg\mathbf{1}), && (\neg\mathbf{s}) +_2 (\neg\mathbf{1}), \\ & \mathbf{s} *_1 \mathbf{l}, && (\neg\mathbf{s}) *_1 \mathbf{l}, && \mathbf{s} *_1 (\neg\mathbf{1}), && (\neg\mathbf{s}) *_1 (\neg\mathbf{1}), \\ & \mathbf{s} *_2 \mathbf{l}, && (\neg\mathbf{s}) *_2 \mathbf{l}, && \mathbf{s} *_2 (\neg\mathbf{1}), && (\neg\mathbf{s}) *_2 (\neg\mathbf{1}) \; \} \end{aligned}$$

In general this operation generates 16 variants of merging of two L-blocks. But in each particular case number of generated L-blocks can be substantially less since some of them can be the same. For example, if L-block **a** is rectangle then ¬**a** = **a**, which means that for two rectangular input blocks operation *merge* will not produce more then four new blocks.

However, this L-block-based algorithm is not guaranteed to find optimal solution of packing problem. As Wang's algorithm is not able to generate non-guillotine placements, this algorithm is not able to produce placement shown below (see Figure 5). Moreover, counterexample can be built for any pairwise merging algorithm which works within one of the classes of isothetic convexity [6].

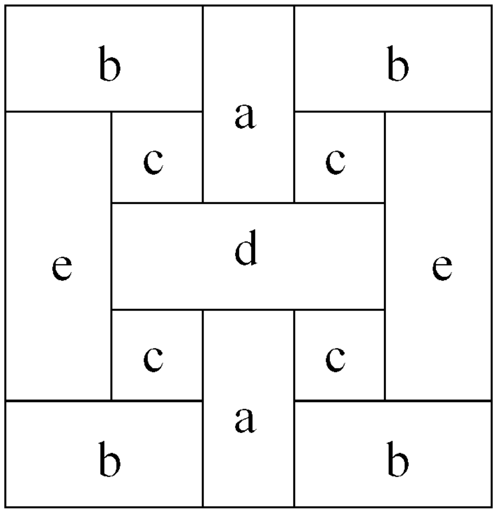


Figure 5. Counterexample for L-block-based algorithm

Computational Results

A number of placements generated by L-block-based algorithm are shown below (see Figure 6). The problem solved in all three cases is constrained cutting stock problem on input data from [2] and [3].

It is important to note that in examples a) and c) the L-block-based algorithm produced non-guillotine placement. In this two cases quality of solutions is substantially better than that provided by Wang's algorithm (see [2]).

References

1. D. F. Wong, H. W. Leong, C. L. Liu. Simulated Annealing for VLSI Design. Kluwer Academic Publishers. 1988. p. 202.
2. P. Y. Wang. Two Algorithms for Constrained Two-Dimensional Cutting Stock Problems. // Operation Research, 1983, Vol. 31, No. 3, pp. 573 – 586.
3. J. F. Oliveira, J. S. Ferreira. An Improved Version of Wang's Algorithm for Two-Dimensional Cutting Problems. European Journal of Operational Research. 1990, vol. 44. pp. 256-266.
4. N. N. Metelski, V. S. Krikun. Method of Hierarchical Placement of Isothetic Blocks. Minsk, Institute of Mathematics of Belarussian Academy of Science. 1990.
5. N. N. Metelski, V. N. Martinchik. Basic Classes of Generalized Convexity for Isothetic Regions. Minsk, Institute of Mathematics of Belarussian Academy of Science. 1991.
6. Recursive Cutting of Rectangular Partitions for VLSI Floorplanning. Simulation Standard. 1998, vol. 9, n. 9, p. 6.

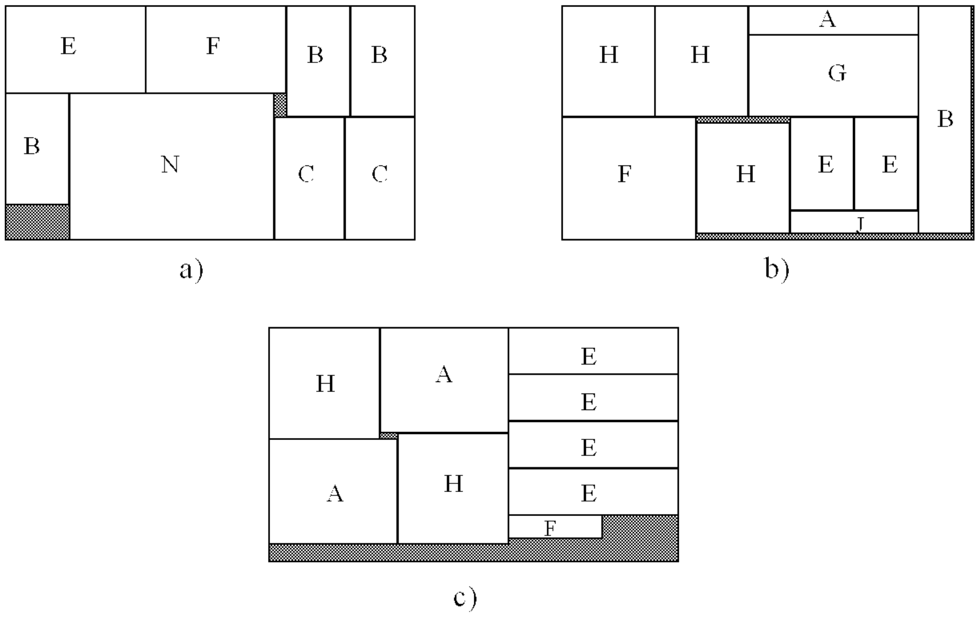


Figure 6. Placements generated by L-block-based algorithm.

Calendar of Events

March

1
2
3
4
5
6
7
8 GOMAC - Monterey
9 GOMAC - Monterey
DATE 99 - Munich, Germany
10 GOMAC - Monterey
DATE 99 - Munich, Germany
11 GOMAC - Monterey
DATE 99 - Munich, Germany
12 DATE 99 - Munich, Germany
13
14
15
16
17
18
19
20
21
22 IRPS 99 - San Diego, CA
23 IRPS 99 - San Diego, CA
24 IRPS 99 - San Diego, CA
25 IRPS 99 - San Diego, CA
26
27
28
29
30
31

April

1
2
3
4
5 MRS Meeting - San Francisco
6 MRS Meeting - San Francisco
7 MRS Meeting - San Francisco
8 MRS Meeting - San Francisco
9 MRS Meeting - San Francisco
10
11
12 High Perfromance
Conference - Amsterdam
13 High Perfromance
Conference - Amsterdam
14 High Perfromance
Conference - Amsterdam
15 High Perfromance
Conference - Amsterdam
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30

Bulletin Board

East Coast Presence Expands

Silvaco will fill the training halls of SUN Microsystems on Friday March 5th. The training session will be handled out of our Massachusetts Office to provide further service and value to our East Coast customers.

Silvaco Attends the American Vacuum Society Meeting

This one-day conference provided a platform for Silvaco's Misha Temkin and Ivan Chakarof to demonstrate our broad set of tools. Look for us at future meetings of the American Vacuum Society.

Meet Silvaco at the MRS Spring Meeting

Booth 220 will be the site for our demonstration. From April 6-8 at the San Francisco Marriott, Silvaco will be on hand with personnel and software in support of a prosperous partnership with the members of the Materials Research Society.

Silvaco Storms the Shores of Euroland at DATE 99

Munich hosts Silvaco at the DATE 99 show March 9-12. Members of Silvaco from the U.K, France, Munich and U.S. Offices will exhibit at the largest Design conference in Europe.

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

The Simulation Standard, circulation 17,000 Vol. 10, No. 3, March 1999 is copyrighted by Silvaco International. If you, or someone you know wants a subscription to this free publication, please call (408) 567-1000 (USA), (44) (1483) 401-800 (UK), (81)(45) 341-7220 (Japan), or your nearest Silvaco distributor.

Hints, Tips and Solutions

Mikalai Karneyenka, Applications and Support Engineer

Q: I often use the temporary reference point, but I would like to see both absolute coordinates and relative coordinates, without toggling the reference point on and off.

A: The menu command Measure>> Coordinate Axes toggles the display of the coordinate axes. They are shown along the upper and left borders of the layout window. The current cursor position is shown by running marks on these axes. Therefore if you switch the axes on and set a reference point, you may see the relative cursor position at the Metric bar, as well as the absolute cursor position at the coordinate axes, see Figure 1. Cursor cross-hairs may be helpful in this case as well.

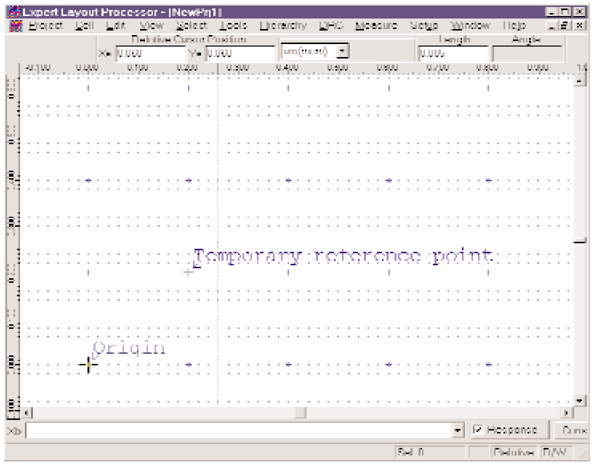

Expert Layout Processor - [NewPrj1]
Temporary reference point
Origin

Figure 1.

Q: Can I perform a check for polygons touching by corners using your Savage DRC system? (See Figure 2.)

A: The required check looks somewhat complex:

```
(* CHECKS FOR CORNER TOUCHING ERRORS *)

    (* NOTE: merge the IN layer only if it is not
    already merged *)

Merge:
   Layer  =    IN,
   LayerR =    &IN_Merged(1);
OutDistance:
   Layer  =    &IN_Merged,
   Type   =    LE,
   Value  =    0.00um;
Width:
   Layer  =    &IN_Merged,
   Type   =    LE,
   Value  =    0.00um;
```

The reason is that there is some discrepancy in merging operation. In ***Expert*** two touching boxes are never merged into one figure. In ***Savage*** they are sometimes merged into an 8-shaped polygon, so you need to check both for zero outdistance and for zero width. This discrepancy will be fixed in one of the subsequent releases of Savage system.

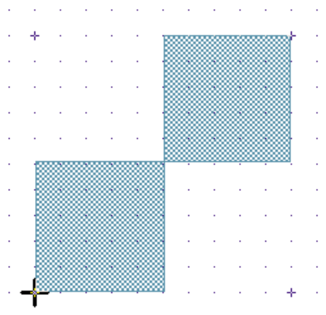


Figure 2.

Q: Windows NT task manager shows me that DRC Guard takes almost 100% of CPU time. Naturally, this worries me.

A: When DRC Guard performs checks, it is supposed to use 100% of processor time. When DRC Guard enabled but idle, you can still notice that processor is about 100% busy. This happens because the error mark drawing function of DRC Guard runs in background.

Even if there are no errors, this function can load the processor to 100%. But this is not to be worried about. This cycle works only when Expert and other WinNT applications are idle. When you start using Expert, DRC Guard drawing is stopped and Expert performance is not affected. Moreover, this drawing is stopped when some other application is activated, so the performance of WinNT and other applications are not affected.

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!

Silvaco Wants You!

- Process and Device Application Engineers
- SPICE Application Engineers
- CAD Applications Engineers
- Software Developers

fax your resume to:
408-496-6080, or
e-mail to:
jobs@*silvaco.com*

Opportunities worldwide for apps engineers: Santa Clara, Phoenix, Austin, Boston, Tokyo, Guildford, Munich, Grenoble, Seoul, Hsinchu. Opportunities for developers at our California headquarters.

SILVACO
INTERNATIONAL

USA HEADQUARTERS

Silvaco International
4701 Patrick Henry Drive
Building 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

e*ECAD
Vendor Partner