# Simulation Standard

Connecting TCAD To Tapeout

A Journal for CAD/CAE Engineers


## *HIPEX-Net*: Interface From Expert

### 1. Introduction

***HIPEX-Net*** is a hierarchical layout extractor. It can create both hierarchical and flat SPICE formatted netlists of the extracted layout. ***HIPEX-Net*** also performs ERC (Electric Rule Checking) on the extracted netlist. This checks for connectivity errors in a chip design such as opens, shorts, and dangling nodes.

***HIPEX-Net*** is a sophisticated script-driven tool. To invoke the extraction, the user must provide a number of input script files in LISA: Language for Interfacing Silvaco Applications. These are option file, layer mapping files, and technology file. ***HIPEX-Net*** technology is defined by a list of various ***LISA*** statements that build derived layers, connectivity, and devices. However, the input generation is easy when running ***HIPEX-Net*** from ***Expert***.

***Expert*** automatically creates all the input files needed by ***HIPEX-Net***. You use the ***Expert*** GUI to define technology and the extractor settings rather than writing ***LISA*** scripts manually. You can also use Dracula technology converter ***Expert*** provides.

***HIPEX-Net*** introduces in ***Expert*** hierarchical Node Probing feature.

### 2. *HIPEX-Net* Options Dialog Box

The user controls the extractor run-time settings in the ***HIPEX-Net*** Options dialog box. It is composed of Layout, Node names, ERC/Pins, Explosion, Ports, Output, and Netlist Technology Definition pages.

#### 2.1. Layout Settings

Here you choose which cell is to be extracted. You can specify any cell in the layout hierarchy as a top level. The **Flatten Layout** option makes ***HIPEX-Net*** flatten the layout before extracting the netlist. This allows you to process safely layouts with hierarchy violations (devices built partly in one cell and partly in another) at the cost of performance. The **Rebuild Derived** Layers checkbox should be checked if there have been any changes in the layers related to device/connectivity definition. The **Annotate Layout** checkbox makes the Node Probing feature available after ***HIPEX-Net*** completes the netlist

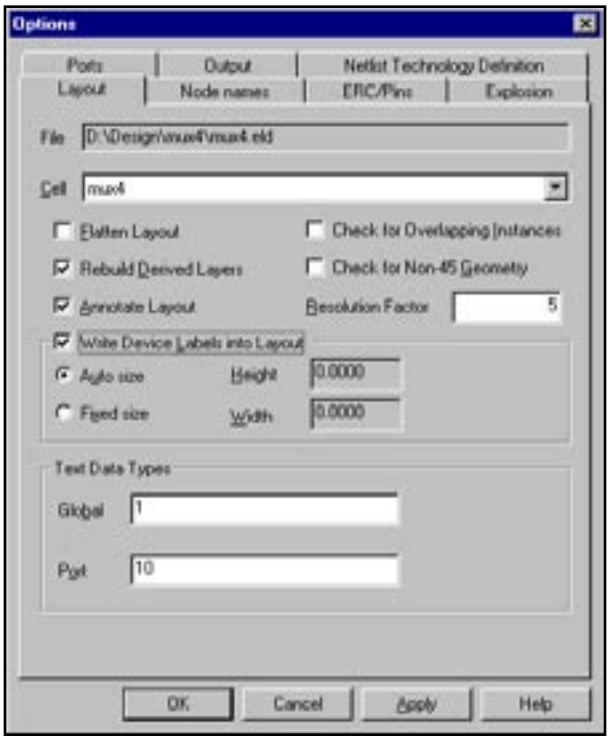


Figure 1. Layout page of the Options dialog.

*Continued on page 2 ...*

*INSIDE*





SILVACO INTERNATIONAL

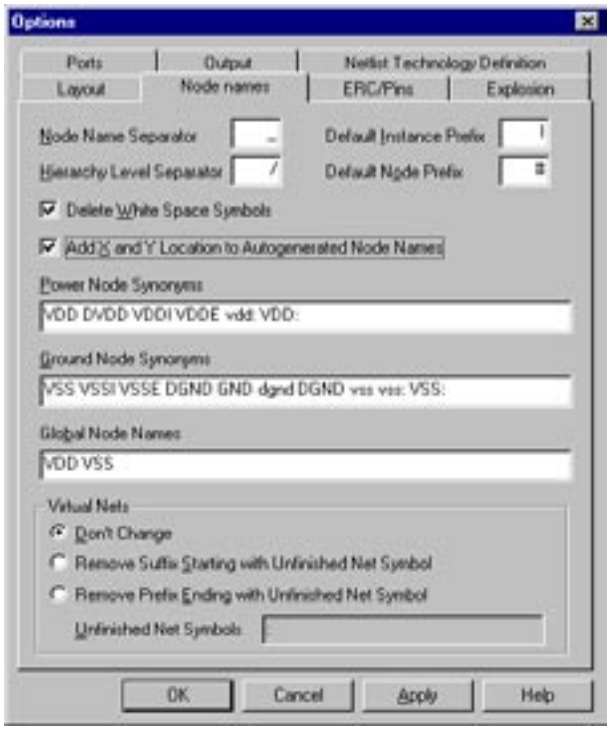

Figure 2. Node Names Options Page.

extraction. Checking this box can decrease ***HIPEX-Net*** performance, but the benefit is that you can search by name or point by mouse nets, devices, and instances to highlight and traverse them directly in the layout editor window.

The two **Check for "..."** options check the layout for multiple identical placements of instances and for non-45 angle geometry. The **Resolution Factor** text box defines the minimum distance that separates two distinct points of the layout.

Using the **Write Device Labels into Layout** group of options, you set up text labels that ***HIPEX-Net*** can add to devices found in the layout. The text is the SPICE statement of the device.

The **Text Data Types** group box contains the two text fields, **Global** and **Port**, for setting the global and port text datatypes. ***HIPEX-Net*** considers text with any other datatype as local text. Local text are labels for a node in a cell, while global text are labels for nodes for the entire layout (e.g., VSS and VDD). Port text are intended to label ports only.

### 2.2. Node Names Settings

***HIPEX-Net*** extracts text from the layout and uses it to: (1) assign names to nodes in the output netlist and (2) check for opens and shorts in the layout. ***HIPEX-Net*** follows a set of rules when processing layout text for node names. For example, global text always takes precedence over local text. You can fine tune node naming in the Node Names page of the Options dialog (see Figure 2).

Here you define the separator and prefix characters ***HIPEX-Net*** uses in node names. If ***HIPEX-Net*** cannot find text anywhere in the hierarchy for a node, it creates a name for the node automatically based on the serial number. You can make ***HIPEX-Net*** add the X, Y layout local coordinates to autogenerated names by checking the appropriate box.

The two text boxes, **Power Node Synonyms** and **Ground Node Synonyms**, allow you to define the synonym names to the power and ground node. ***HIPEX-Net*** substitutes all the listed names by the first one. The Global Node Names text box lists labels you want to make global in spite of their actual datatype in the layout.

The **Virtual Nets group** box provides options to make ***HIPEX-Net*** consider separated virtual (unfinished) nets as the same net.

### 2.3. ERC/Pins Settings

***HIPEX-Net*** uses extracted text from the layout to check continuity and reports possible shorts and opens. If ***HIPEX-Net*** finds open or shorted nodes, it flags them by writing error messages to the summary file. ***HIPEX-Net*** can also report dangle nodes that are not attached to any devices.

You control various ERC options in the ERC/Pins page of the Options dialog shown in Figure 3.

Here you choose whether ***HIPEX-Net*** reports in the summary file all the dangles it encounters. Using the

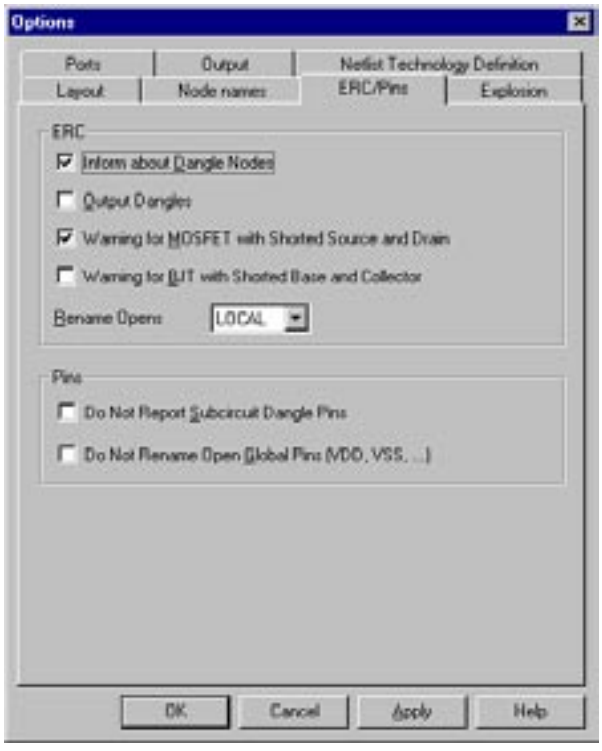

Figure 3. ERC/Pins Options Page.


Figure 4. Explosion Options Page.

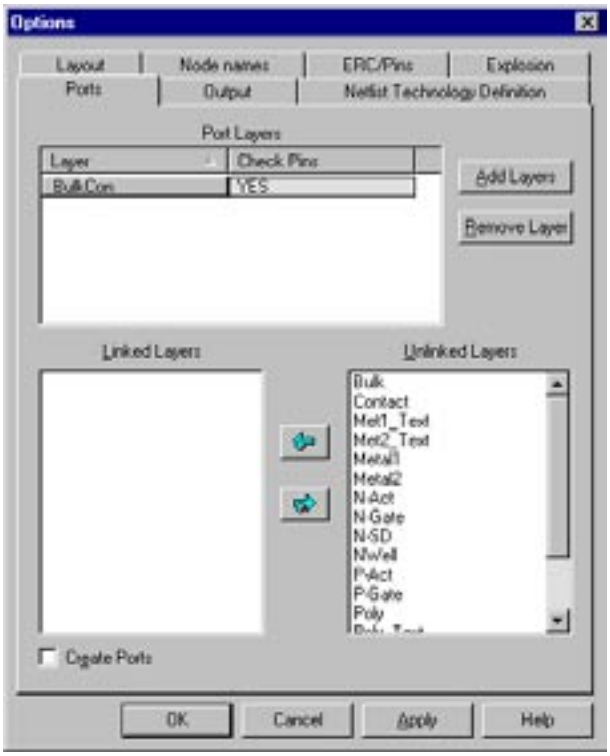


Figure 5. Ports Options Page.

**Output Dangles** checkbox, you can dump flat layout of the dangle nodes into the separate GDSII file. The two **Warning for "..."** checkboxes make ***HIPEX-Net*** write in the summary file the warning messages about improperly connected MOSFETs and BJTs. The **Rename Opens** drop-down list determines whether ***HIPEX-Net*** assigns different names to the nodes having the same local (in a given cell) or global (in the entire layout) text label.

The **Pins** options determine whether ***HIPEX-Net*** ignores dangle pins of subcircuits and renames open global pins.

### 2.4. Explosion Settings

In the Explosion page of the Options dialog, you can define various operations on design hierarchy, such as cell explosions and ignoring particular cells during the extraction. Figure 4 shows the Explosion page.

Here you can explode all the instances of cells containing only wiring, raising their content up one level before the cell is processed. The other checkbox forces ***HIPEX-Net***- to ignore all text at lower levels of the hierarchy except the top one.

The table in the bottom of the Explosion page allows you to define operations on individual cells. The allowable operations are: EXPLODE, FLATTEN, SMASH, and IGNORE. The AUTO option means do nothing.

### 2.5. Ports Settings

One of the benefits using ***HIPEX-Net*** is that it doesn't force you to declare the cell pins or assign names to cell pins. ***HIPEX-Net*** creates pins automatically when it finds any hierarchical connection (that is, between cells from


Figure 6. Output Options Page.

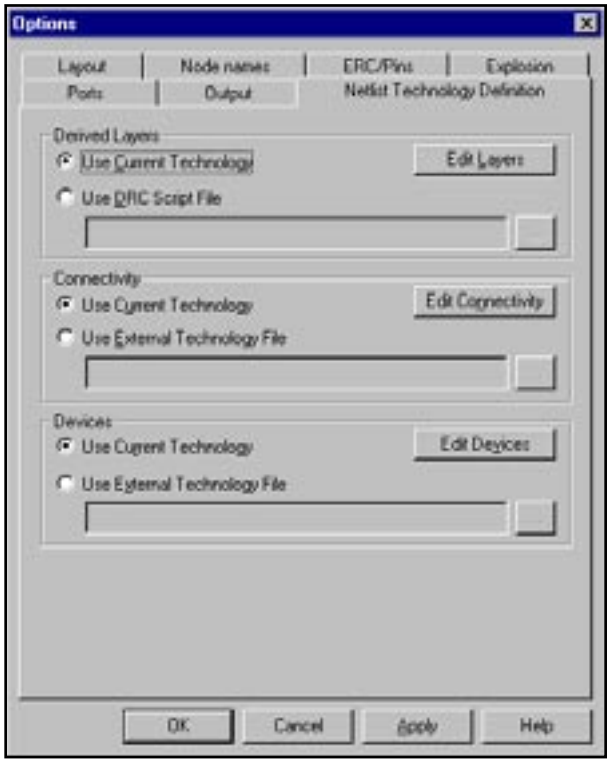

Figure 7. Netlist Technology Definition Page.

different hierarchy levels). Nevertheless, ***HIPEX-Net*** allows you to predefine names and locations of cell pins. These predefined pins are called ports. There are a couple of advantages using ports. One advantage is you can define pins for the top cell in the layout. The other advantage is you can verify hierarchical connections (i.e., pins) by ports. If you do so, ***HIPEX-Net*** creates pins only in those locations where both pin and port are present.

You define ports in the Ports page of the Options dialog box (see Figure 5).

Use a special port layer in your layout (you can also use several port layers if needed). Give your ports names using port datatype text. Then, set this layer as port layer using the **Add Layers** button. Once you define port layer(s), you can link/unlink conductor layers to the selected port layer. Then, the port-linked conductor layers can be verified for hierarchical connections that go through them. You can force HIPEX-NET to create subcircuit pins in the port locations, even if there are no actual hierarchical connections, by checking the **Create Ports** box.

### 2.6. Output Settings

In the Output page of the Options dialog, you define filenames of the output SPICE netlists and control SPICE device parameters to be written to those netlists. Figure 6 shows the Output page.

The two **Write "..."** allow you to output additional SPICE parameters for MOSFETs and capacitors.

## 3. Netlist Technology Definition

The Netlist Technology Definition page of the Options dialog allows you to define technology information needed by ***HIPEX-Net***. ***Expert*** saves the user technology definitions in the project file (.eld file). Then, the layout editor converts the technology data to the ***HIPEX-Net*** technology file once you run the extractor.

***HIPEX-Net*** deals with three types of technology information (see Figure 7): Layer Derivation Statements **(Derived Layers),** Connectivity Statements **(Connectivity),** and Device Definitions **(Devices).** You can modify these types of technology parameters separately.

For technology definition, you can use GUI controls and dialogs accessible from the page above. The alternative way is to load technology data from external ***Expert*** technology files (.tcn files). You can create .tcn files in any text editor manually or use the **Setup>>Technology>> Import Technology"..."** menu command to convert Dracula rule files.

# "Shortest Path" Option for Flight Line Style (Net Bar)

## Introduction

There is the recent improvement in Expert Layout Editor, the "Shortest path" option was added to Flight line styles (Net Bar). This option allows to build the shortest tree (network) connecting all selected nodes.

"Tree" means the network of vertices (nodes) connected with undirected edges (lines) which has two basic properties:

1. Any two vertices are connected with some path.
2. There are no unnecessary edges, in other words if any edge is removed then the first property will be violated.

"Shortest tree" means the tree with minimal summarized length of edges. It is also called the Minimum Spanning Tree (MST). Minimum Spanning Tree of the given set of points P on Euclidean plane is called Euclidean Minimum Spanning Tree (EMST).

## Algorithm

There are a lot of algorithms for finding the Minimum Spanning Tree (one of them is well-known Prim's algorithm, which is described below), but their common problem is bad productivity.

Most of these algorithms require at least $O(n^2)$ time (where n is number of vertices), because the first step for these algorithms is building of the complete graph with all possible edges, where each vertex is directly connected with any other vertex. This operation takes $O(n^2)$ time.

Such slow algorithms can be used with small number of vertices but are absolutely inefficient with huge number of vertices. For example, processing of 10,000 vertices will be about 1,000,000 times longer than processing of 10 vertices.

To avoid this problem the set P of points on Euclidean plane is processed in two steps.

The first step is finding the Delaunay triangulation (Figure 3) for n points on Euclidean plane using the "divide and conquer" algorithm (presented in [1]).

During second step we run the Prim's algorithm on the Delaunay triangulation (Figure 4) to find the Minimum Spanning Tree.

The total processing time is $O(n \cdot \log(n))$, so processing time grows almost linearly with number of processed points.

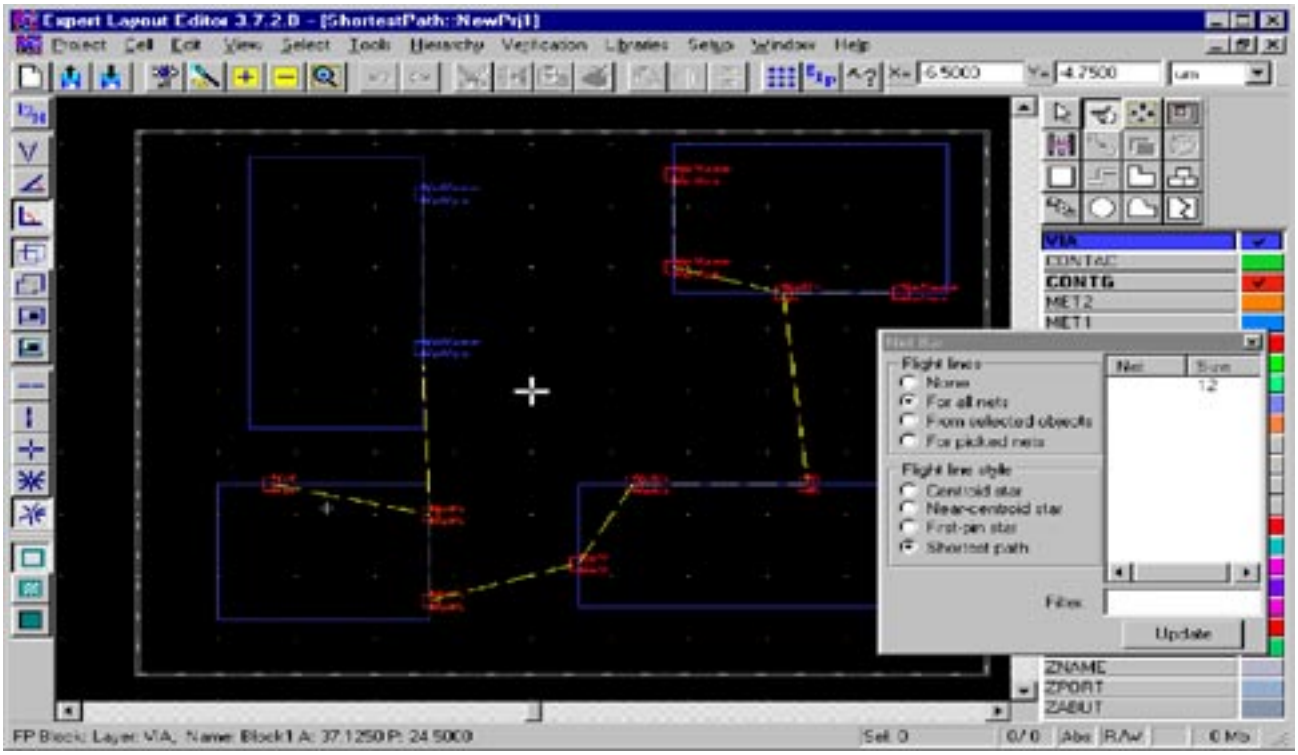

Figure 1. Layout Expert Editor screenshot.

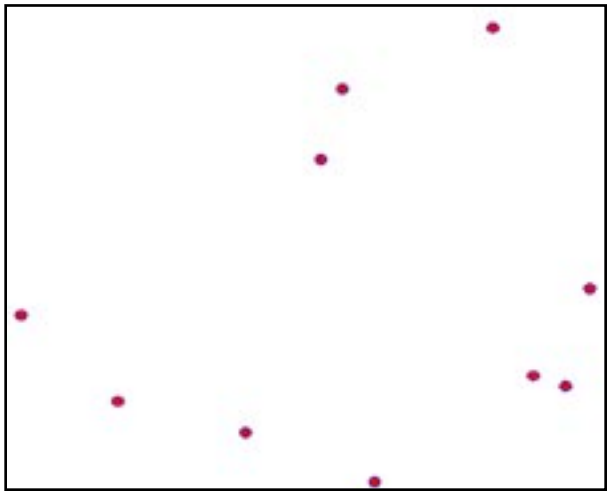

Figure 2. There are no edges.

The results of described two-steps algorithm is EMST because of well-known theorem

EMST of given set of points P on plane is a subgraph of Delaunay triangulation of P.

## Delaunay Triangulation and "Divide and Conquer" Algorithm

The Delaunay triangulation is a triangulation in which every circumcircle of a triangle is an empty circle. Figure 3 shows the example of Delaunay triangulation.

- The common idea of the "divide and conquer" algorithm is to process the set of vertices recursively.
- The original set of vertices $V = \{v_1, \ldots, v_n\}$, $v_i = (x_i, y_i)$ is sorted by x and y coordinates so that $v_i < v_{i+1}$, that means $x_i \le x_{i+1}$, and if $x_i = x_{i+1}$ then $y_i < y_{i+1}$.
- The total set of vertices V is divided into two parts $V_1 = \{v_1, \ldots, v_k\}$ and $V_2 = \{v_{k+1}, \ldots, v_n\}$, $V_1 \cup V_2 = V$ according to sort order (left part and right part).

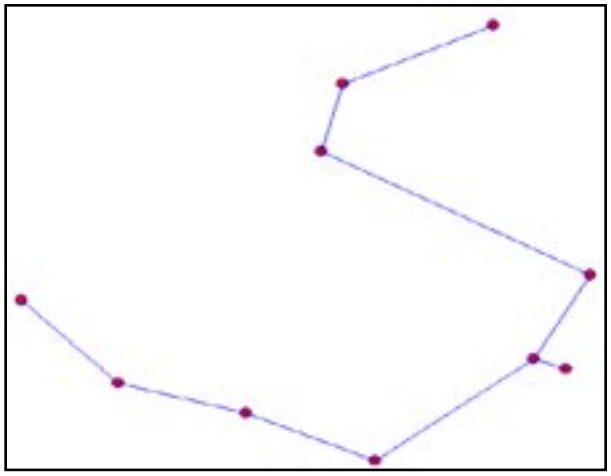

Figure 4. Step 2 – finding the Minimum Spanning Tree using Prim's algorithm

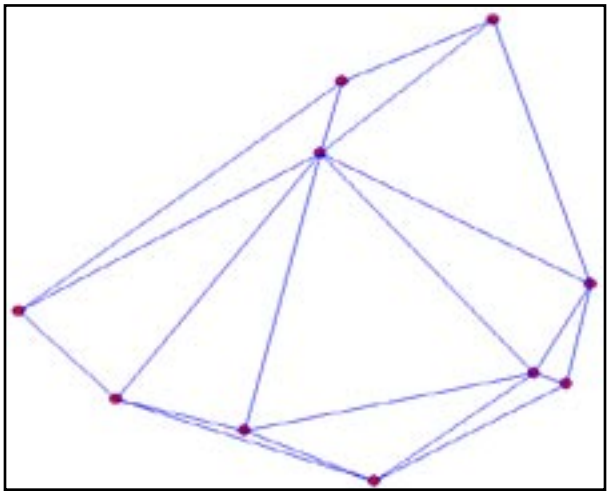

Figure 3. Step 1 – creating the Delaunay triangulation using "divide and conquer" algorithm

- Then "divide and conquer" algorithm is recursively applied to both parts (independently).
- The resulting Delaunay triangulations of two parts are joined (zipped) into one Delaunay triangulation.

## Prim's Algorithm

Prim's algorithm processes graph G = (V, E) of n vertices $V = \{v_1, \ldots, v_n\}$ and m undirected edges $E = \{e_1, \ldots, e_m\}$ and removes unnecessary edges.

- First step is to choose an arbitrary vertex $v_1 \in V$ and to build the set of processed vertices $P = \{v_1\}$, consisting only of $v_1$.
- Then on every step one new vertex $v_i$ is added to the set of processed vertices P. To choose this vertex we look for smallest edge $e = (v_j, v_k)$ connecting the processed vertices with unprocessed, $v_j \in P$, $v_k \in V \setminus P$.
- Process is repeated until there are no unprocessed vertices more.

Figure 4 shows the results of Prim's algorithm.

### References


[1] Guibas, L. and Stolfi, J., "Primitives for the Manipulation of General Subdivisions and the Computation of Voronoi Diagrams", ACM Transactions on Graphics, Vol.4, No.2, April 1985, pages 74-123.

[2] Okabe, A.; Boots, B.; and Sugihara, K. Spatial Tessellations: Concepts and Applications of Voronoi Diagrams. New York: Wiley, 1992.

[3] Lee, D. T. and Schachter, B. J. "Two Algorithms for Constructing a Delaunay Triangulation." Int. J. Computer Information Sci. 9, 219-242, 1980.

# Expert Layout Editor Recent Improvements – Over-the-Point Drawing of Orthogonal Shapes

This article presents an overview of new features added to Silvaco's ***Expert*** layout editor. Silvaco has taken a careful look at the real-world obstacles faced daily by semiconductor layout designers and design engineers. Silvaco has augmented ***Expert*** with features that dramatically increase flexibility and also help to reduce constraint violations and other roadblocks inherent to complex layout design.

***Expert*** is a fast, robust, PC-based editor that is designed to manage the complex issues inherent to advanced, deep-submicron Ultra Large Signal Integration (ULSI) designs. ***Expert's*** proprietary database efficiently processes (ULSI) without restrictions on layout complexity or type of geometry. These improvements once again secure ***Expert's*** place as an industry standard without sacrificing the stability and ease-of-use that seasoned layout experts have come to expect and rely on.

## New Features, Better Results

***Expert's*** new features help users avoid the geometrical constraint violations that result from non-orthogonal cursor movements during the creation of polygon and 90-degree Angle Mode wires. ***Expert*** now defines authorized cursor movements with specific axis points. The software automatically adds extra vertices in order to keep shapes consistent within the allotted space. Seasoned users of ***Expert*** have more flexibility than in previous versions that automatically restrained the cursor to orthogonal movements while in Direction Snap Mode.

The orthogonal-movements-only option is still available in the Edit > Direction Snap Mode > Orthogonal menu. In this mode, each defined vertex requires a single mouse click.

New menus simplify and augment project setup. Angle Mode easily sets default geometrical constraints for 90- and 45-degree custom layouts. Direction Snap Mode restricts potentially dangerous cursor movement to user-defined behavior patterns during object creation and modification. These features are accessed from either Edit > Angle Mode or Edit > Direction Snap Mode menus, by clicking the corresponding Drawing Bar, or by pressing an assigned hotkey.

Over-the-Point Mode simplifies the shape definition process and then quickly renders orthogonal polygons and wires. When this mode is active, a user-specified definition is needed for every second vertex (1st, 3rd, 5th, etc). Two consecutive orthogonal polygons or wires are created with a single mouse click in projects where Angle Mode is set to 90 degrees and Direction Snap Mode is set to "None" (no cursor movement) or to "Diagonal" (diagonal and orthogonal movements only).

The Vertical First option simplifies the drawing of complex shapes with Over-the-Point input. If the Edit > Region Mode > Vertical First option is checked, the first segment is created vertically and the second one, horizontally. If the option is not checked, the reverse is true. The Vertical First shortcut enables or disables this feature during object drawing.

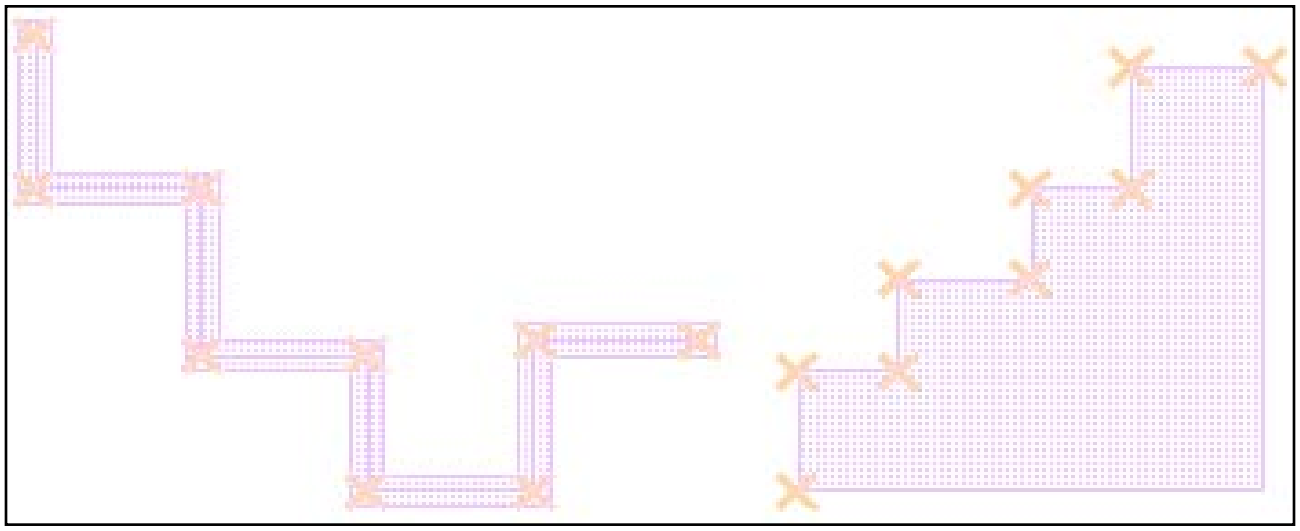

Figure 1. Orthogonal cursor movements only.

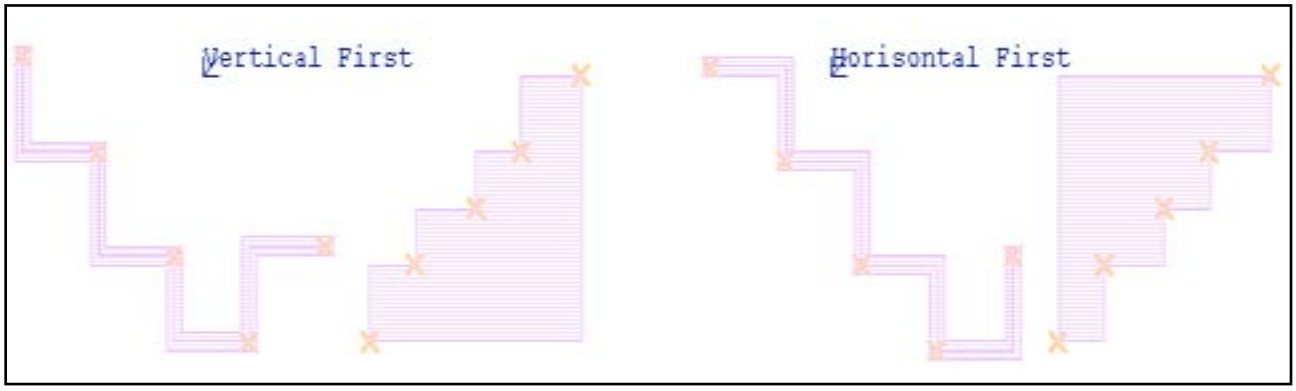


Figure 2. No restriction on cursor movements.

## Search and Properties Dialogs No Longer Block Editing

The Edit >Search and Edit >Properties dialog boxes are reimplemented as non-modal, i.e., ***Expert*** window is available for editing, while search results are available for traversing. This allows to make search only once to perform any legal operation with each found layout object.

## Show Critical Zone Option in *DRC Guard*

***DRC Guard*** checks geometrical design rules during construction, in other words "on the fly". Rules for checking are specified by an ordinary DRC script. The violations are not saved in the DRC error database, but are reported by blinking markers on the screen. If the cursor is placed at a blinking error marker, a short description of the error appears as a small pop-up label. To fix indicated DRC rules violation, objects with violating segments should be modified or moved to make blinking marker disappear.

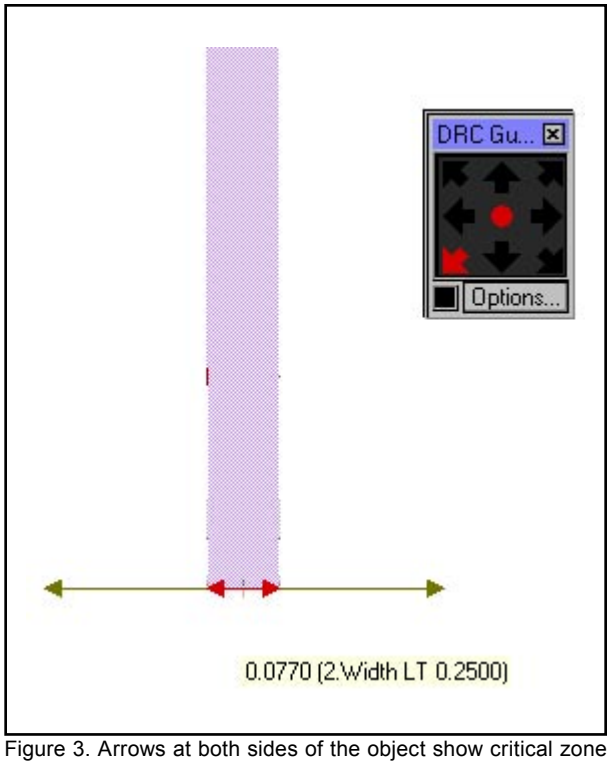


Figure 3. Arrows at both sides of the object show critical zone for width error.

***Expert DRC Guard's*** Show Critical Zone option displays or hides critical zone indicators in the design layout. The critical zone clearly highlights the minimal shifts of violating segments that are critical when troubleshooting potentially detrimental DRC violations. The option is available only when the Show Error Flags menu option is enabled.

## Conclusion

This paper provides a brief sketch of some of the new features in Silvaco's ***Expert*** layout editor. These features simplify the layout design process and enhance the flexibility that ***Expert's*** satisfied users are familiar. Additional information about these and other features is found in the Expert product documentation or from Silvaco International.

# Guardian DRC – Recent Development

## Edge-Based Operations

Recent versions of Guardian DRC introduce operations that involve edge layers, either on input or on output.

An edge layer is a layer that contains line segments that are edges or parts of edges of shapes from other layers. These edges have "inner" and "outer" surfaces defined according to the shape layers they were derived from.

### Edge Selection by Topological Relations

```
select_edges:  relation={inside | outside}
               [, options=(not)],
               layer1=LayerA,Layer2=LayerB,
                  LayerR=EL1;
```

These commands (without "Not" option) select edges or parts of edges from Layer1 that lay strictly inside (respectively, outside) of shapes from Layer2.

"Not" option is for inverting the selection.

```
select_edges: relation = {coincide | touch}
  [, options=(not[,{inside | outside}])],
  layer1=LayerA, Layer2=LayerB,LayerR=EL1;
```

"Coincide" relation selects parts of edges from Layer1 that coincide with (parts of) edges from Layer2.

"Touch" relation selects complete edges whose parts coincide with (parts of) edges from Layer2.

"Inside" option means that at points of coincidence the insides of edges overlap.

"Outside" option means that at points of coincidence the insides of edges do not overlap.

"Not" option is for inverting the selection.

### Shape Selection by Edges

```
select: relation=touch,
  [, options=(options_list)],
        layer1=LayerA, Layer2=LayerB,
  LayerR=EL1;
```

Layer2 is allowed to be edge layer. In this case, the operation selects shapes from Layer1 whose edges touch edges from layer2. In this case "Touching" can be both inside and outside touching. (When both layers are shape layers, touching is considered only from the outside.)

### Edge Layers on Input of Old DRC Operations

- All distance check operations accept edge layers on input
- Substrate (new syntax allows input layers)
- Copy (if input is edge layer, then LayerR is edge layer)
- Delete
- Select...Touch may have Layer2 to be edge layer.

### Size_Edge Operation

```
Size_edge: layer = <L1>,
[{In | In_Factor} = <NIn>,] [{Out | Out_
  Factor} = <NOut>,]
[{Extend | Extend_factor} =<NExt>,]
layerr = <L2>;
```

"*_Factor" parameters mean that the numeric parameter is the factor by which the edge length must be multiplied to obtain the actual sizing parameter.

"In" parameter specifies the amount of expanding the edge in the "inside" direction. This parameter must be nonnegative.

"Out" parameter specifies the amount of expanding the edge in the "outside" direction. This parameter must be nonnegative.

"Extend" parameter specifies the amount of elongating the edge from both sides. This parameter may be negative, meaning edge contraction.

An edge produces no output, if its length does not exceed twice the negated extension value (e.g., in the case "Extend_factor = -0.5").

NOTE: If all in/in_factor/out/out_factor parameters are zero, the output is an edge layer, otherwise it is a shape layer.

### Edge Selection by Slope

The following command selects edges with slopes within specified limits.

The limits are specified in degrees, with values between 0 and 90.

Slope: Layer=<name>, LayerR=<name>, Limits <range>;

The output is an edge layer. The input layer may be shape or edge layer.

Example:

Slope: layer= M1, layerR=M1Acute, limits >0 <=45;

### Edge Selection by Adjacent Corners

Select_Edges: Relation=Corners,

[convex=<012-limits>,] [length <limits>,]

[angle1 <0.0-360.0>,][Length1 <limits>,]

[angle2 <0.0-360.0>,][Length2 <limits>,]

layer=<shape-layer>, layerR=<edge-layer>;

This command selects edges basing on the parameters of the adjacent corners: angles and side lengths.

convex specifies limits for the number of convex angles adjacent to the edge;

length specifies limits for the length of the segment itself

angle1,2 specify limits for the first/second adjacent angle, in range between 0 and 360 degrees.

length1, 2 specify limits for length of the first/second adjacent side.

Examples:

Select_Edges: Relation=Corners, convex <2, length >10, layer=m1, layerr = outbendM1;

Select_Edges: Relation=Corners, convex <2, angle1 = 90, angle2 == 90, length <0.3, layer=m1, layerr = m1Loop;

## News for Antenna Checks

### Notion of Layer of Origin

If LayerB is produced from LayerA by the following operations:

- Select operations with merged selection layer
- Select_edge operations
- Check operations with different LayerR1 and LayerR2 values
- Check_Node_Params
- Copy

then LayerA is called layer of origin for layerB.

### Antenna Checks with Accumulation

'LayerA' keyword is introduced for Check_Node_Params command:

Check_Node_Params: Formula = (expression),

Value=<value1[:value2]>, Type=<check type>,

Layer = <layer identifier>, LayerR = <output layer identifier>,

LayerA = <accumulative layer identifier>;

LayerA must be the output layer of a previously executed Check_Node_Params operation. LayerA and Layer must have the same layer of origin.

If LayerA is not specified, then Check_Node_Params command attaches to each output polygon the corresponding value of computed Formula.

If LayerA is specified, then for each polygon from Layer (which belongs to an electrical node present in input parameter files) the following is done:

- If the polygon is not present in LayerA, then the calculated value of Formula itself is checked against the constraints.
- If the polygon is also present in LayerA, then the calculated value of Formula for its node is increased by the value attached (by the preceding Check_Node_Params operation) to the polygon in LayerA, and the result is checked against the specified constraints.

If the checked accumulated value meets the constraints, then the polygon is output to LayerR with the accumulated value attached.

Note: If LayerR is written into Expert layout database, the attached values are represented as used-defined properties with name specified in "Update_layout" command, parameter "Antenna".

## Antenna Log Files

The 'LogR' and 'LogA' keywords are introduced for Check_Node_Params command:

Check_Node_Params: Formula = (expression),

Value=<value1[:value2]>, Type=<check type>,

Layer = <layer identifier>, LayerR = <output layer identifier>

[, LayerA = <accumulative layer identifier>]

[, LogR = <output geometry log-file name>],

[, LogA = <antenna violation log-file name>];

'LogR' keyword allows to generate a log-file containing the following information for each polygon in the output layer:

1) lower left vertex of the polygon (coordinates are in internal database units)

2) (accumulated) antenna value corresponding to the polygon (the value are in script measurement units).

The log-file also contains information about the operation performed, the internal database unit and the script measurement unit.

If file with specified name exists, the operation rewrites it.

'LogA' keyword allows to generate a log-file containing the following information for each electrical node failed the antenna check:

1) node number

2) computed formula value corresponding to the node (the values are in script measurement units).

The accumulation of antenna values (provided by 'LayerA') are not reflected in this log-file.

The log-file also contains information about the operation performed and the script measurement unit.

If file with specified name exists, the operation appends it.

### 'Shapes' Parameter

A new geometric parameter name, 'Shapes', can be used in Get_Node_Params and Check_Node_Params commands. It denotes the number of polygons that belong to a node. For this node parameter, all actions (Min, Max, Sum) are equivalent, so you can always use this parameter without specifying the action.

### "Not" Operation

Unary ! (Not) mathematical operation can be used in Formula in Check_Node_Params command. The result of the operation is zero if its argument is non-zero and 1 if its argument is zero.

## Preprocessor Directives

In addition to #ifdef and #ifndef, a new directive, #if is added.

Syntax:

```
#if <Boolean_expression>
//...
[#else]
//...
#endif
```

Boolean_expression can consist of preprocessor variables and constants, parentheses, Boolean operations (NOT, AND, OR, XOR, EQV (case insensitive)), and comparisons of expressions (==, != <=, >=, <,>).

Constant's Boolean value is "true" if it is nonzero and "false" otherwise.

Preprocessor variable's Boolean value is defined as before.

Recall that a preprocessor directive must be written in a single line.

Example:

```
#define A
#define B 1
#undef B

// at this point a=true, b=c=d=false
#if (A or B or C or D)
//...
#else
//...
#define C 12
// now C is true
#if (((C >= 1) and NOT A) or D)
//...
#endif
//...
#endif
//...
```

## Input/Output of Layout Data

### Input Merging

Merge_Input: {on | off};

If this setting is on, then all input layers are merged when layout data is read from them into the DRC system.

The default setting is off, according to the old behavior of the DRC engine.

### Input Snapping to Grid

Snap_Input: {on | off};

If this setting is on, then all input layers are snapped onto the corresponding grids when layout data is read from them into the DRC system. If Merge_Input is on, input snapping is performed before input merging.

Grid setting is specified by Grid_Resolution directive.

The default grid setting is 1 Database unit.

The default setting is off, according to the old behavior of the DRC engine.

### Returning Layers from DRC to Expert Layout Database

The following setup command controls writing layers from DRC layout data into ELD files.

Update_Layout:

```
[  input=yes|no]
[, new=yes|no]
[, ampersand="string"]
[, technology=Yes|No|Combine]
[, nonempty=  Yes|No|Combine]
[, layers=(<list>)]
[, no_layers=(<list>)]
[, combine_layers=(<list>)]
[, warning=Yes|No|Error];
[, Antenna_Attr_Name = <text>]
[, Node_Attr_Name = <text>…;]
[, output_cell= "name"]
```

No : do not write into ELD file.

Yes : overwrite existing data.

Merge: merge with existing data.

Input: Layers from "Layers" list

New: New (non-temporary) layers

Ampersand="string": Write back temporary layers by name, with '&' replaced by "string" (1st '&' is forbiddedn in Expert)

Technology: Non-scratch layers from ELD technology (by name)

Nonempty: Any nonempty ELD layers

Layers=(list): list of layers allowed to be written back

(overrides settings of switches)

Merge_Layers=(list): list of layers allowed to be merged by writing back

(overrides settings of switches)

No_Layers=(list): list of layers forbidden to be written back

(overrides settings of switches)

Warning=Yes|No|Error : if a DRC command outputs a layer that confilicts with the settings of the command, then write/nowrite warning into log or stop execution.

Default setting (chosen to match the old behavior of DRC):

If this command is not specified then input=no, all remaining switches =yes

If several switches apply to a layer (e.g. a layer is input and nonempty), then option "NO" takes precedence (for data safety).

Conflicts between "...Layers" lists are treated as error.

Passing Connectivity and Antenna Information into ELD Format

If a layer has node info and/or antenna info, then the corresponding shapes written into eld file will have user-defined attributes with the names specified in the Update_layout command and the corresponding values.

Antenna_Attr_Name specifies the name of a user-defined attribute attached to a shape when it is returned from DRC to Expert. The value of this attribute is the value calculated by Check_Node_Params operation.

Node_Attr_Name specifies the name of a used-defined attribute attached to a shape when it is returned from DRC to ***Expert***. The value of this attribute is the node name for the shape.

Cell for DRC-Generated Layers

output_cell= "name"

This parameter specifies the name of the cell to write the DRC-generated layers instead of the cell for which DRC was run.

If there is no such cell in the layout, it will be created.

## Numeric Parameters of DRC Commands

This version allows you to write flexible checks using arithmetic expressions to specify check constraints, as shown in the example below.

variable: Gap = 0.3;

size: layer= AVDF, value = Gap/100, layerr= AVs;

merge: layer = AVs, layerr = AVm;

Outdistance: layer = Avm, limits >=Gap/100 < Gap*1.5, ID= "AVD3.1";

### Check and Selector Limits

A simpler syntax for check constraints is introduced to replace the likes of "type=LT, value=0.3".

```
Limits <limit>,

Limits <range>

where <limit> is one of :

!= 'value'

== 'value'

 = 'value' (the same as ==<value>)

 > 'value'

 < 'value'

>= 'value'

<= 'value'
```

'value' cannot have measurement unit suffix. You must use "unit" statement, if necessary.

<range> is <limit1> <limit2>, where one of limits is the lower limit of the range and another one is the upper limit. The lower one must not exceed the upper one.

Examples

(1) width: layer=m1, limits !=1;

(2) width: layer=m1, limits <=2>=1;

(3) width: layer=m1, limits >1 <=1.5; // run time error: the upper limit less than the lower one

(4) width: layer=m1, limits >=1 <=1; // equivalent to (1)

(5) width: layer=m1, limits >1<1; // run time error: bad range

A similar usage is possible for count limits in select operations.

Examples

Select: relation= overlap, options=(shapes !=2),...;

Select: relation= overlap, options=(nodes >1 <=4),...;

## Variables

Variables are components of arithmetic expressions.

variable: <name> = <value>;

<name> is the name of the variable. It is case-insensitive.

<value> is an arithmetic expression built from numbers and previously defined variables.

Examples

```
VARIABLE: MinW  = 0.2;

VARIABLE: MinS =0.301;

VARIABLE: Unders=MinW/2 - 0.001;

VARIABLE: Step = 2*Unders + MinS;
```

## Arithmetic Expressions

Arithmetic expressions are constructed from numbers and variables, parentheses, arithmetic operators (+,-,*,/) and functions:

MIN(expr, expr), MAX(expr, expr) minimum and maximum of the two expressions

INT(expr) integer part of the numer

SQRT(expr), square root

POWER(expr1, expr2) expr1 raised into expr2 power.

Arithmetic expressions may be used in the following places:

- Variable definitions

- With "Limits" keyword in checks

- With "Value" keyword in sizing operations

- With "shapes" and "Nodes" keywords in select options.

## Substrate Command: New Options and Parameters

substrate: [layer = <LName>,]

[T=<Ymax>,][ B=<Ymin>,][ L=<Xmin>,][ R=<Xmax>,] layerr= <RLName>;

substrate: [layers = (Lnam1, ..., LnamN),]

[T=<Ymax>,][ B=<Ymin>,][ L=<Xmin>,][ R=<Xmax>,] layerr= <RLName>;

substrate: [options=([Nosize][,][Input]),]

[T=<Ymax>,][ B=<Ymin>,][ L=<Xmin>,][ R=<Xmax>,] layerr= <RLName>;

T, B, L, R parameters specify the top, bottom, left and right clipping coordinates for the produced box. In particular, the commands

variable: BoxSize = 0.1;

substrate: b=1, t=1+BoxSize, l=3, r= 3+BoxSize, layerr=Box13;

build 'Box13' layer with the specified box, if it is inside the "normal" substrate.

"Input" option means that the substrate is based only on the layers actually taking part in DRC operations (regardless technology file or layer table).

**Note**:

Compatibility with Batch Calibre:

substrate: options=(input, nosize),...

Compatibility with Interactive Calibre:

substrate: options=(nosize),...

substrate: [layer = <LName>,] layerr= <RLName>;

substrate: [layers = (Lnam1, …, LnamN),] layerr= <RLName>;

substrate: [options=(Nosize),] layerr= <RLName>;

If Layer or Layers parameters are present, then the command builds the bounding box for the listed layers only.

If "Nosize" option is absent, this box is resized by 1um from the minimal bounding box. "Nosize" option cancels this resizing.

NOTE: In the first two forms of syntax there is no resizing.

# *Guardian LVS*: New Platform-Independent GUI

## Introduction

We introduce platform-independent version of ***Guardian LVS***. At present, very important feature of any tool is its capability to operate on various platforms and to support the internationalization. Now ***Guardian LVS*** is a multiplatform application operating on the following platforms: Windows, Linux, Solaris (32- and 64-bit versions), HP-UX. ***Guardian*** User Interface is realized on the base of Trolltech's Qt that is a C++ toolkit used by numerous companies and organizations for development of multiplatform GUI and applications.

## *Guardian* User Interface Using Qt

***Guardian LVS*** is equipped with Silvaco Text Editor which is also based on Trolltech's Qt toolkit. This Text Editor has advanced memory management that allows you to operate with files up to 2Gb. The system optimally uses RAM by "dynamic loading-unloading" the contents of file. The Text Editor has all the basic features, such as:

- File features for creating, opening, and saving text files.
- Printing features with possibilities of choosing printer, specification of size, source, orientation of the paper, choosing color or grayscale mode for printing, setup of page margins, print preview.
- Editing features with multi-level undo and redo; cut, copy, paste, delete and select all support.
- Search features that allows to find, find next, find previous, replace a given text string, quickly go to a given line number, to set the bookmarks and travel from a bookmark to another one in the forward and backward directions.

Moreover, ***Guardian*** User Interface has the features for management of LVS verification process:

- Project file features for loading and saving LVS project files.
- Action features for performing LVS verification, flattening hierarchical netlists, viewing hierarchical structure of netlist, loading "LVS Navigator" tool after the verification is done.
- Window management features for arrangement of netlists and reports. ***Guardian LVS*** allows you to view simultaneously all LVS reports as well as to link these reports with the schematic and layout netlists for easy inspection of LVS results.
- Setup features for setting options of LVS verification, some options of user interface, font setup.
- ***Guardian LVS*** on-line help system that uses Qt text browser for hypertext navigation.

***Guardian LVS*** also performs the spice syntax highlighting for schematic and layout netlists. It can recognize names of devices and subcircuit instances, spice commands, parameter keywords, and comments (see Figure 1).

Frequently used commands are assigned to buttons of four toolbars: main, action, window and edit. You can move these toolbars and show or hide any one from them.

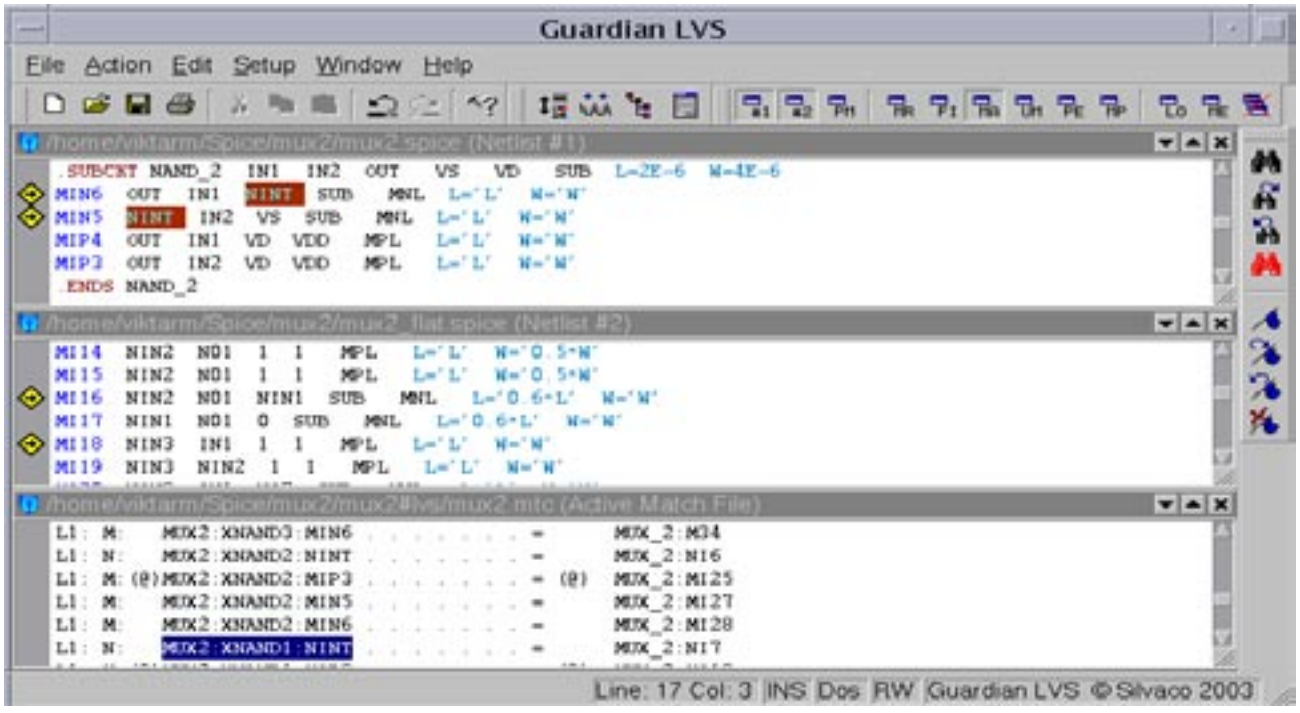


Figure 1. Spice syntax highlighting in platform-independent Guardian LVS.

## Inspection tools in platform-independent *Guardian LVS*

After a LVS run has been performed, ***Guardian LVS*** creates the reports that are used for inspection of LVS results. Platform-independent ***Guardian LVS*** is equipped with a set of tools that link the report files with the schematic and layout netlists and make the inspection of results maximum easy. ***Guardian LVS*** has the following inspection tools:

- Netlist Hierarchy tool activates the Netlist Rover that contains the full information about netlist hierarchy represented in the form of the tree (see Figure 2). You can see in the tree all instances belonging to a subcircuit as well as the names of subcircuits corresponding to these instances. You can highlight the instance or subcircuit in netlist by double click on its name in the tree of Netlist Rover.
- Connectivity Traversing tool shows all nodes to which selected node is connected. Double click on any of the neighboring nodes opens the same panel with connections for new node.
- Netlist File Highlighting tool allows you to find a node in netlist after double click on its name in a report. The corresponding spice file will be open and all appearances of this node name will be highlighted at the top level or in a subcircuit (see Figure 1).
- LVS Navigator tool searches and inspects matched and unmatched nodes, discrepancies and parameter errors. These nodes will be highlighted in reports and both netlists (see Figure 2).
- Node Walker tool shows you the neighborhood list of a node and match or unmatch information for these neighborhoods in the Node Walker panel. Double click on a node name in this panel activates neighborhood list of this node (see Figure 2).
- Merge Hierarchy tool displays the information about merge and reduction for a selected hierarchical name in the merge, match, unmatch, parameter error, and parameter match reports (see Figure 2).
- Instance-Subcircuit Info tool shows the hierarchy of a selected node name in filter, match, unmatch, parameter error or parameter match reports. You can highlight in netlist any instance and the corresponding subcircuit which contains a selected node by double click in Instance-Subcircuit panel.

***Guardian LVS*** can process huge netlists containing millions of devices. These netlists have the nets connecting to hundreds of thousands neighbors. Some from above-listed tools, for example Connectivity Traversing, Node Walker, must contain and represent the information about all connections of these nets. Moreover, the tree in Netlist Rover panel must keep all names of instances, subcircuits of these instances, and devices of the netlists. It's necessary to note that the representation of information in dialog panels in ***Guardian LVS*** on the base Trolltech's Qt takes much less time than ***Guardian LVS*** on the base MFC. For example, representation of the tree of Netlist Rover for netlist with 120000 devices in ***Guardian LVS*** with Qt is performed about 50 times faster than in ***Guardian LVS*** with MFC.

## Conclusion

At present ***Guardian LVS*** is a platform-independent application. It operates on Windows, Linux, Solaris, HP-UX and can be built for many other Unix-variant platforms.

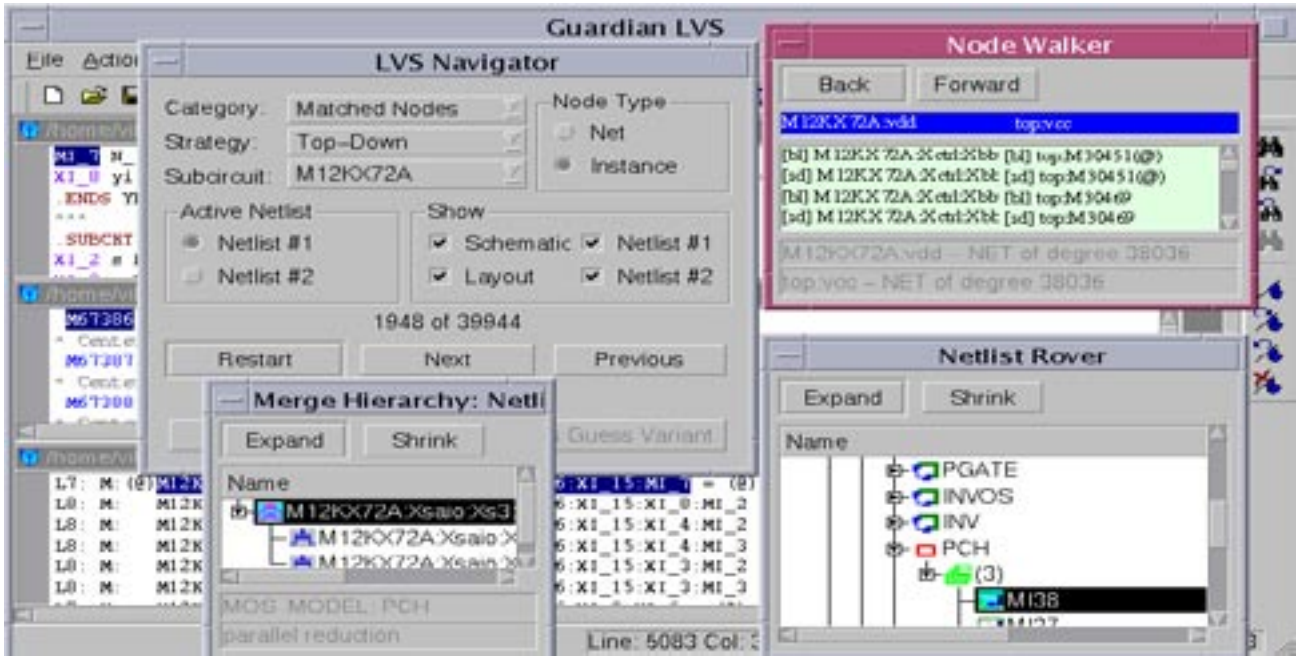

Figure 2. Inspection tools in Guardian LVS (Linux example).

# Calendar of Events

## June

| | |
|---|---|
| 1 | |
| 2 | |
| 3 | |
| 4 | |
| 5 | |
| 6 | |
| 7 | |
| 8 | |
| 9 | |
| 10 | |
| 11 | |
| 12 | IPRM - Santa Barbara,CA |
| 13 | IPRM - Santa Barbara,CA |
| 14 | IPRM - Santa Barbara,CA |
| 15 | IPRM - Santa Barbara,CA |
| 16 | IPRM - Santa Barbara,CA |
| 17 | |
| 18 | |
| 19 | GaAs MANTECH - Scottsdale, AZ |
| 20 | GaAs MANTECH - Scottsdale, AZ |
| 21 | GaAs MANTECH - Scottsdale, AZ |
| 22 | GaAs MANTECH - Scottsdale, AZ |
| 23 | |
| 24 | |
| 25 | |
| 26 | |
| 27 | |
| 28 | |
| 29 | |
| 30 | |
| 31 | |

## July

| | |
|---|---|
| 1 | |
| 2 | |
| 3 | |
| 4 | |
| 5 | |
| 6 | |
| 7 | |
| 8 | |
| 9 | AM-LCD - Kogakuin Unv, Tokyo |
| 10 | AM-LCD - Kogakuin Unv, Tokyo |
| 11 | AM-LCD - Kogakuin Unv, Tokyo |
| 12 | |
| 13 | |
| 14 | |
| 15 | |
| 16 | IMFED - Osaka, Japan |
| 17 | IMFED - Osaka, Japan |
| 18 | IMFED - Osaka, Japan |
| 19 | |
| 20 | |
| 21 | NSREC - Monterey, CA |
| 22 | NSREC - Monterey, CA |
| 23 | NSREC - Monterey, CA |
| 24 | NSREC - Monterey, CA |
| 25 | NSREC - Monterey, CA |
| 26 | |
| 27 | |
| 28 | |
| 30 | |

## Bulletin Board

### *Impact of Silvaco SmartSpice-64 to SoC Design Problems*

SmartSpice-64™ Circuit Simulator gives true SPICE accuracy for simulating large circuits beyond the capability of 32 bit SPICE. One major application of this technology is the SPICE simulation of an SoC clock tree with full parasitics to validate signal integrity of the clock signal. Another major application is to simulate complete megabit memories for sense amp margins with full parasistics or to measure static, dynamic, and leakage power—especially at 130 nanometers and below. These two applications share the common simulation requirements for uncompromised accuracy, large capacity, and simulation results in a reasonable timeframe. SmartSpice-64 meets these needs with true SPICE accuracy, virtually unlimited address space, efficient memory compression algorithms, multiple solvers for convergence, and parallel processing to decrease run-time.

*If you would like more information or to register for one of our our workshops, please check our web site at* ***http://www.silvaco.com***

# Hints, Tips and Solutions

Galina Makovsky, Applications and Support Engineer

**Q:I have some trouble counting the number of instances in a layout using Expert. I have tried to use the Chip Rover feature, but it does not give the right answer. I have the same cell instanced repeatedly in an array. What is the procedure for counting the number of occurrences of an instance in a top cell?**

A: The Chip Rover>>Active Cell Tree shows the number of instances and arrays in the current cell. To see the number of instances in each array, expand the tree as in Figure 1. If there are arrays in it, multiply number of rows by number of columns to get cell count within the array.

The Edit>>Search command allows to find all cell instances and arrays in the Cell Hierarchy. To calculate the number of instance's occurring in a cell, check the number of rows and columns for each found object.

For complex cell hierarchy or if this operation is frequently performed, it would be more efficient to use LISA script to automate the routine as shown below:

```
-------------------------
listCells = "";
aCellNames = get_cell_list("");
nCountCells = aCellNames.size;
i = 1;
LOOP BEGIN
  IF (i GTR nCountCells) THEN (LEAVE LOOP);
  $$objs = (find objects (SEARCH_INSTANCE)
          /criteria = ({search_criterion_create(OA_
   CELL, aCellNames[i], EQ)})
          /visible);
  IF ($$objs.size GTR 0) THEN BEGIN
    IF (listCells EQL "") THEN ( listCells =
   (aCellNames[i]) )
    ELSE ( listCells = listCells & "\t" &
   (aCellNames[i]) );
  END;
```

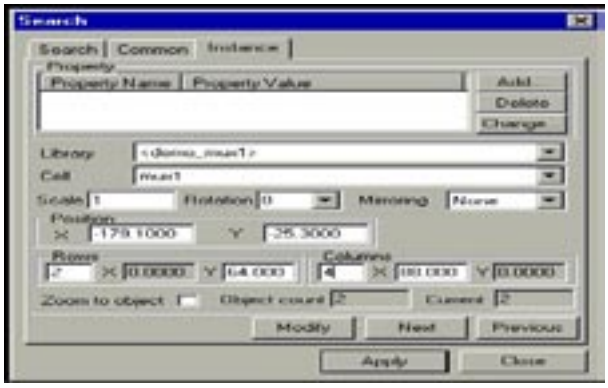


Figure 2. Counting the number of instances, using Search

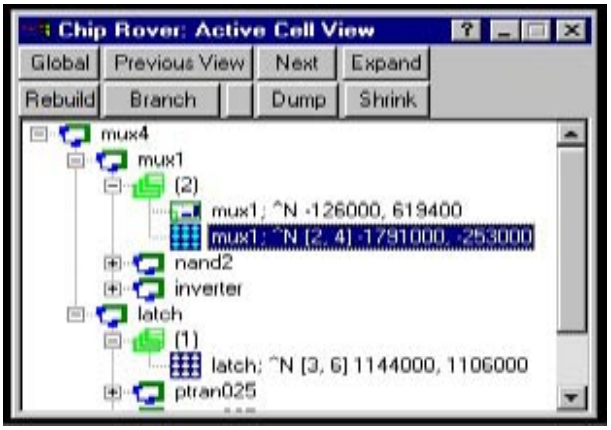


Figure 1. There are 9 instances of mux1 in the current cell ( [2,4] = 8 instances )

```
  i = i + 1;
END;
varparams = {{listCells, "Instance of cell"}};
dlgparams = {"Instance count"};
vars = form_create(varparams, dlgparams);
if (vars.size GTR 0) THEN BEGIN
  display(vars[1]);
  $$objs = (find objects (SEARCH_INSTANCE)
          /criteria = ({search_criterion_create(OA_
   CELL, vars[1], EQ)})
          /visible);
  len = $$objs.size;
  display(len);
  $$cur_obj = $$objs.first;
 count = 0;
 LOOP BEGIN
  IF ($$cur_obj EQL nil) THEN (LEAVE LOOP);
    display($$cur_obj.shape & "("&&$$cur_obj.cols &
   ", " & $$cur_obj.rows&")");
    count = count + $$cur_obj.cols * $$cur_obj.rows;
    $$cur_obj = $$objs.next;
  END;
    display(count); ! Display count or write to file

END;
------------------------------
```

**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


# 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*