Simulation Standard

Connecting TCAD To Tapeout

A Journal for CAD/CAE Engineers

Real-time DRC in Expert Layout Editor

1. Introduction

In this paper we introduce one of the latest and most advanced features of Silvaco's ***Expert*** Layout processor for Windows NT – DRC Guard. DRC Guard is fully functional real-time design rule checker that works in background mode and makes extensive use of multithreading/multiprocessor capabilities of operating system. Based on a specially optimized and tuned DRC engine, DRC Guard works extremely fast and reports results of local DRC directly into the layout area in convenient user-friendly way. DRC Guard automatically checks modified part of the layout immediately after the modification is made. It also can perform instant DRC in any part of layout at the user's request.

2. DRC Guard

2.1 Real-time background DRC

2.1.1 Minimizing response time

One of the most interesting problems, which arise in real-time DRC system implementation process, is the trade-off of response time and number of DRC checks. On the one hand, to be of use this system should be able to handle DRC scripts that contain substantial number of commands. Often some very useful DRC checks can't be expressed by means of a single check command. Implementation of such checks may include generation of intermediate layers by, boolean or resize operations. Several such not very sophisticated multi-command checks together may create DRC script consisting of hundreds or even thousands of commands which should be executed in real-time.

On the other hand, a real-time DRC system should not get in the way and make the user wait until execution of the script is completed blocking all layout functions. No matter how fast is the internal DRC engine, any non-trivial DRC script being applied in straightforward way to each modification made to layout will noticeably and unacceptably increase the editor response time.

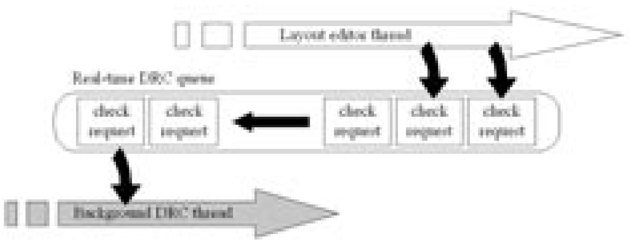


Figure 1. Layout editor and background DRC communication

To resolve this conflict, special techniques have been developed that made it possible to implement the real-time background DRC. Given a multithreaded operating system (such as Windows NT), the main idea is that real-time DRC is separated into a special low-priority background thread while layout editing is performed by a high-priority foreground thread. Instead of calling DRC directly after each modification, the layout editor places a special record called check request into the real-time DRC queue. A check request contains full information about what kind of modification was made, which object was modified, where it is located, etc. Background DRC reads check requests from the queue one by one and performs all necessary checks in the corresponding area of layout. The real-time DRC queue is a communication link between the layout editor and the background DRC (see Figure 1).

Continued on page 2....

INSIDE

It is obvious that DRC system built on these principles will not necessary report all DRC violations immediately after they are introduced. But the time gap between an erroneous edit and the error report can be negligibly small provided the DRC engine is accurately implemented and tuned within the whole system. The key is that care should be taken to minimize the amount of information processed by the background DRC without affecting check accuracy (see “Minimizing system resource usage” section).

Separation of real-time DRC into an independent thread also makes it possible to control its execution priority with respect to the priority of the layout editor. The higher is the priority of DRC the shorter is its response time. The lower is the priority of DRC the less noticeable is its slowing influence on layout editor functions. Another great advantage of a multithreaded architecture is that on multiprocessor system the real-time DRC thread will work on a separate processor. It will result in a substantial increase of the performance of the background DRC as well as the performance of the layout editor.

2.1.2 Database Synchronization

Although our real-time DRC works as a background thread parallel to the layout editor thread, the former is not completely independent from the latter. The most obvious point of competition between these threads is the moment when they try to access the common layout database simultaneously. The execution of DRC on some part of layout takes some period of time while user is free to modify any part of layout at any moment. Under this assumption it appears to be a reasonable technique to make a separate copy of local part of layout to be checked – the layout snapshot. After that the background DRC is executed on that copy rather than on original layout. In this case the problem of database access synchronization will take place only at the moment of snapshot preparation and can be easily solved by means of standard thread synchronization methods provided by the operating system.

Although such technique of snapshot processing of layout works well with batch mode fire-and-forget background DRC, it is not free of drawbacks when used in real-time background DRC. Suppose that user is making some modifications in the part of layout which is being processed by DRC at same time. It means that at any moment the snapshot taken for DRC may become out of date. If it does, chances are that the results of this script execution will be obsolete as well. To prevent real-time DRC from generating obsolete results such, situations should be detected. DRC execution stopped and modified area scheduled for recheck. The same applies to check requests waiting in the real-time DRC queue. Some of them may become obsolete before they are processed and should be excluded from the queue. This technique of exclusion of obsolete check requests leads to substantial increase in performance of real-time DRC.

2.1.3 Minimizing System Resource Usage

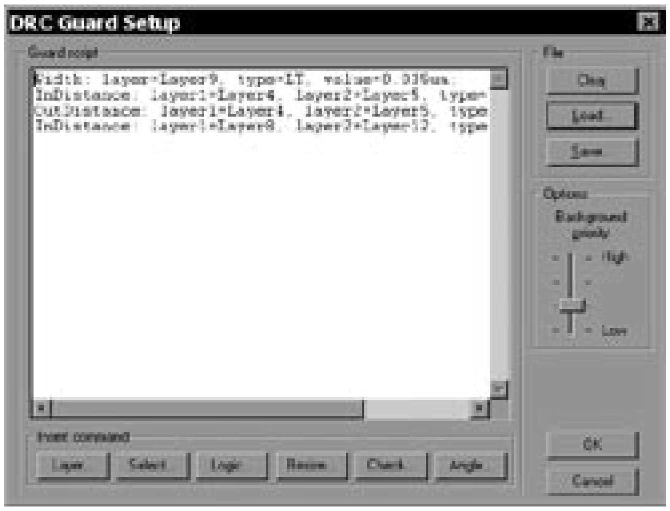
The amount of data handled by conventional DRC can be very large and usually does not fit into the internal memory requiring additional external storage space. However, it makes sense to use only RAM for real-time DRC in order to increase its performance thus reducing its response time. Of course, when real-time DRC is implemented as background thread care should be taken to prevent it from occupying too much system memory because it always leads to the drop of overall system performance. The simplest way is to set a limit for the amount of memory allocated for real-time DRC needs and cancel DRC script execution whenever the limit is exceeded. The corresponding check request from the real-time DRC queue is removed without processing.

Under assumption that the amount of memory background DRC thread can occupy is limited, the problem of most effective utilization of that memory arises. An effective method of optimization of real-time DRC resource usage is developed. It will be referred to as selective DRC script execution. When a designer is editing a layout most of modifications he makes are localized in a particular area and particular layer of the layout. Most frequently used operations such as creation or deletion of object or changing shape of object locally affect contents of only one layer. Hence, to check whether the result of such operation is violating any of DRC requirements it is sufficient to execute only those operations of DRC script, which are related to the modified layer. The problem of extraction of the sequence of the operations to be executed is non-trivial because any check operation can reference layers of the layout directly as its operands as well as indirectly through a chain of layer generation operations (boolean, resize etc.). The informational dependency graph of script commands can be used to solve this problem efficiently [2]. When the sequence of DRC commands is determined, the set of input layers for this sequence can be easily obtained. Only layers from this set and their objects from area of modification should be included in the layout snapshot before executing the DRC script. In most cases such preliminary analysis of the script allows faster checking of the modified portion of layout because it reduces the amount of layout data to be prepared for DRC processing and decreases the number of commands to be executed.

2.2 Using DRC Guard

2.2.1 Operations Recognized by DRC Guard

The background real-time DRC based on the principles described above and called DRC Guard was recently introduced into Silvaco's ***Expert*** Layout Processor. Once set up, DRC Guard watches all modifications which are made to the layout and generates immediate visual report of all detected errors. Internal DRC engine used in DRC Guard is based on very efficient scan-line approach [1]. It allows DRC Guard to deliver check results in true real-time. It is worth to mention that DRC Guard supports all types of error report conventional DRC does including polygon-shaped errors reported by check operations with GT/GE comparison types (see "User interface" section and Figure 5).



```
Width: layer=Layer9, type=LT, value=0.335um
InDistance: layer1=Layer4, layer2=Layer5, type=
OutDistance: layer1=Layer4, layer2=Layer5, type
InDistance: layer1=Layer8, layer2=Layer12, type
```

Figure 2. DRC Guard Setup is through a user defined options menu.

Setup of DRC Guard is very simple and includes ordinary DRC script written in ***Savage*** DRC language (see Figure 2). Not only check operations but also any command of the language can be used in DRC Guard script.

Check requests are placed into DRC Guard queue after each application of any editing operation such as Create Object, Delete Object, Modify Object, Stretch, Move, Copy, Cut, Paste etc. Being built into hierarchical layout editor, DRC Guard is able to handle editing operations not only when they are applied to polygons but also when they are applied to instances of other cells in current one. DRC Guard can also perform quick DRC check of any rectangular area of layout by user's request (see Figure 6). DRC Guard also supports Undo command. Moreover, DRC Guard produces correct and useful results in ***Expert's*** Edit-in-Place mode of hierarchical editing.

2.2.2 Error Report on Hierarchical Layout

Since it is implemented in a hierarchical layout editor, DRC Guard takes into account hierarchical structure of layout when performing DRC checks and displaying errors. The purpose of DRC Guard is real-time detection of DRC violations caused by user's modifications currently made. It aims for finding and reporting of errors of this kind only. If DRC Guard finds some errors that are unlikely to be caused by user modifications, it does not report them to avoid overloading of workspace with irrelevant DRC error marks. To carry out DRC check of entire layout or its particular part with full error report designer can use batch-mode DRC or DRC in Area.

The problem of filtering out irrelevant errors becomes more complicated when DRC is executed on hierarchical layout, because in this case designer can work not only with polygonal shapes but also with instances of other cells. Incorrect placement of the instance may induce DRC violations, which also have to be detected. DRC Guard considers a DRC error as relevant and reports it if the error meets certain conditions. More concretely, when designer is working with cell C DRC Guard reports all errors that would be dispatched to that cell C in accordance with nearest common parent rule used in hierarchical DRC [3]. To get all information necessary for application of this rule DRC Guard makes use of hierarchical information inheritance technique [3].

In simple words, in its search for DRC violations in current cell C DRC Guard checks the following relations:

a) between polygons of cell C;

b) between direct and indirect instances of other cells in cell C and polygons of cell C;

c) between different direct and indirect instances of other cells in cell C.

It is very likely that the actual reason of DRC violation detected by DRC Guard in cell C resides nowhere else but in cell C [3].

2.2.3 DRC Guard in Edit-In-Place

When user is editing cell contents in the Edit-in-Place (EiP) mode the situation is rather different from ordinary editing mode. The nature of difference is that the cell is edited in its particular placement – the instance – and therefore it is surrounded by instance's environment, i.e. objects of its parent cell. At any moment user can change edited cell by diving one level down via instance of another cell as well as by climbing on level up. Chain of instances used by the user to move between levels of hierarchy is called the Edit-in-Place chain (EiP chain). EiP chain begins in the topmost cell and ends in the current edited cell. Any modification made on certain level of hierarchy may induce DRC violation not only on that level but also on any higher level up to the topmost

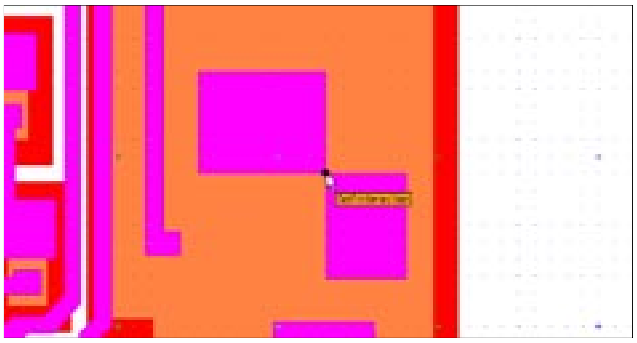


Figure 3. Intersection marker (Intersection check)

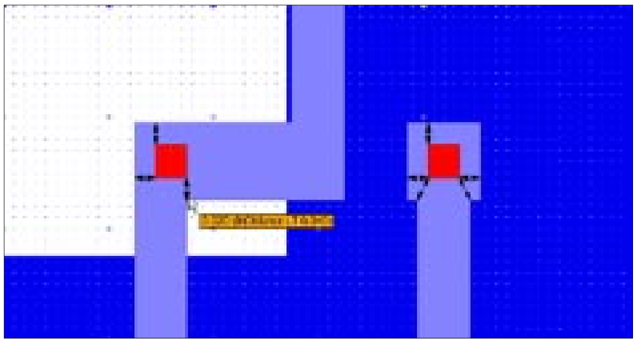


Figure 4. Minimum distance violation markers (InDistance LT check).

one. Moreover, before leaving modified cell user has to choose the way of its integration into project structure. The options are the following:

1. *Explode instance* – the instance of edited cell is replaced with its polygons in parent cell. DRC Guard checks the area of the former instance in parent cell and environments of all instances from EiP chain.
2. *Save Instance as Separate Cell* – DRC Guard checks environments of all instances from EiP chain including environment of edited instance in parent cell.
3. *Change Edited Cell* – DRC Guard checks environments of all instances of this cell in all their parent cells up to the topmost one (including instances from EiP chain).

Such behavior of DRC Guard in EiP mode may seem rather complicated but it leads to effective detection of all DRC violations caused by modifications made in EiP mode. The large numbers of check requests that may be generated after EiP modifications do not represent any problem because DRC Guard is implemented as background thread.

2.2.4 User interface

All DRC errors detected by DRC Guard are immediately displayed on the screen in form of blinking markers. The markers can have different shape depending on type of particular violation. For instance, distance violations are displayed in form of two-headed arrows and intersections are marked by circles. When mouse cursor hovers above particular error mark the label containing information about this error is displayed. The information includes type of violation and DRC rule that is violated (see Figures 3, 4, 5).

Another component which makes DRC Guard's user interface more convenient is the Error Locator. Although the main purpose of DRC Guard is to check correctness of local layout modifications, it can easily handle any non-local editing operation. Note that such operations as, for instance, Move can change location of all currently selected objects. Depending on distribution of selected objects over the layout area, application of this operation may result in local modification as well as in global one affecting the entire layout. In case selected objects are relatively far one from another, operation Move can produce DRC errors outside view area. It means that DRC Guard will actually detect all errors but user probably will not be able to see all error markers on the screen. In such situations Error Locator could be very useful. Error Locator consists of small display containing eight arrows and central dot (see Figure 7). Each of eight arrows represents particular direction (North, Northeast, East etc.) and begins to blink if DRC Guard has detected violation in corresponding area outside the screen. When mouse cursor hovers over blinking arrow in Error Locator, the amount of errors found in that invisible area is displayed. The central dot provides the same information for visible screen area. Clicking on any of the arrows causes the layout to pan so that the nearest error in corresponding direction moves to the center of view area. This provides simple and convenient way of navigating DRC Guard errors.

A number of important additional steps were taken in order to achieve further improvement of DRC Guard performance, reliability and convenience. We only mention some of them here. First, the DRC Guard setup

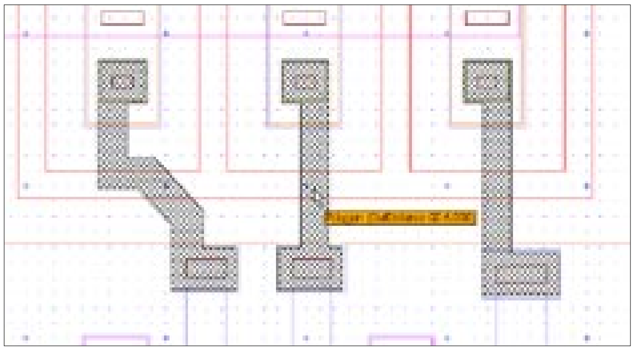


Figure 5. Maximum distance violation markers (OutDistance GE check).

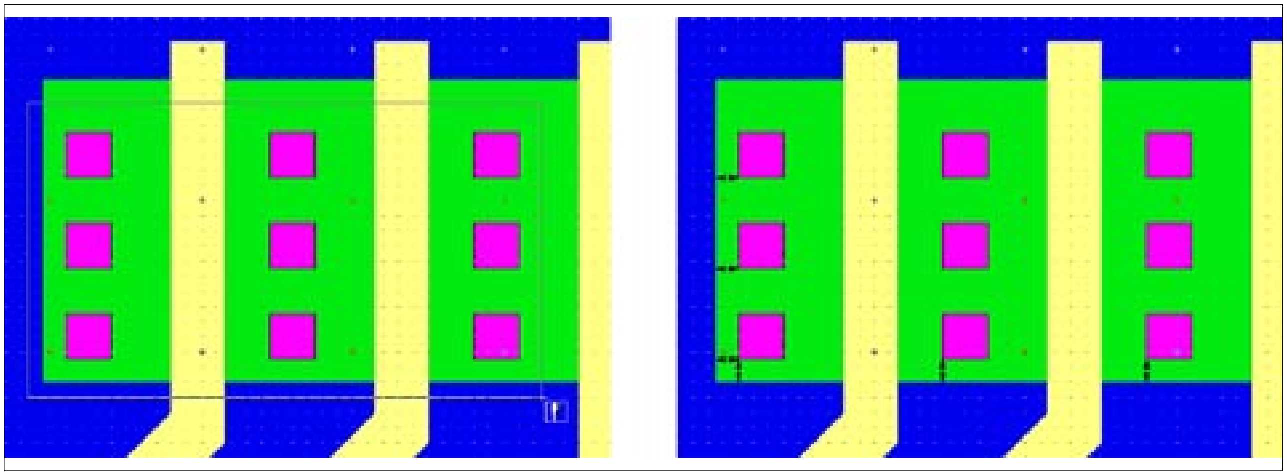


Figure 6. Check Area function of DRC Guard.

dialog box offers three ways to specify the script: manual typing, semi-automatic command generation by means of dialog boxes, loading from file. Second, the DRC Guard script is analyzed, compiled and optimized in special way at setup stage to reduce the initialization time of processing of each check request. Third, DRC Guard tracks the location of user's working area on the layout and processes in first turn check requests that are closer to that area than others. Fourth, the DRC Guard thread is carefully isolated from the layout editor thread so that accidental crash of DRC Guard doesn't result in crash of the entire layout editor and data loss.

3. Conclusion

The real-time DRC the ***Expert*** Layout Processor is evolving very rapidly in direction of introducing new user interface features, new types of DRC commands and speeding up overall DRC performance. In its current state DRC Guard is convenient and powerful component of the layout editor, which improves speed and quality of custom design of integrated circuits.

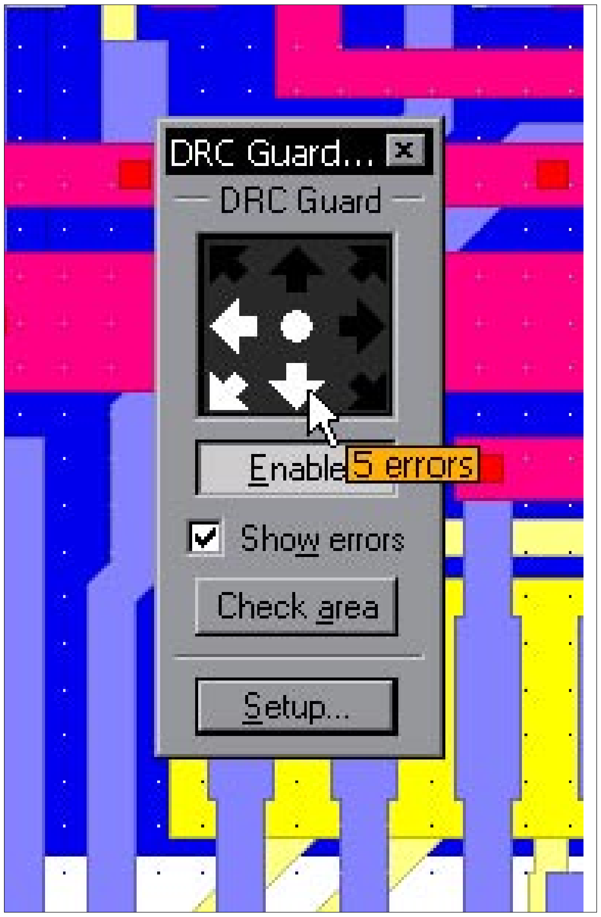


Figure 7. Error Locator.

References

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

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

[3] Savage Enhanced with Recognition and Reporting of Hierarchical Structure of Errors. Simulation Standard. Volume 9, Number 3, March 1998, pp. 1-4.

Recursive Cutting of Rectangular Partitions for VLSI Floorplanning

Rectangular partitionings form a mathematical base for many modern approaches to automation of VLSI design [1-3]. In particular, popular methodologies of hierarchical placement (by cell grouping / merging) as well as procedures of global routing and layout compression deal with partitions that can be hierarchically subdivided into components of bounded complexity.

The article shows that some tight placements cannot be generated by any pairwise merging procedure in the class of isotetically convex blocks.

Isotetically convex blocks (here called blocks, for simplicity) are well described in [4,5]. A block P is a closed polygon whose boundary consists of line segments parallel to coordinate axes, and every horizontal or vertical segment connecting any two points of the polygon P belongs to P. Rank r of block P is the minimal number of rectangles R1, R2, ..., Rr which do not intersect block P and complete it to rectangle. In other words:

P ∪ (∪ Ri, for i=1,2,...,r) is rectangle,

Ri ∩ Rj = ∅ for i,j=1,2,...,r, i != j.

Ri ∩ R = ∅ for i=1,2,...,r.

This definition of rank is equivalent to one introduced in [5]. Rank r of block P and number k of block vertices are related by equation

k = 2r + 4.

Figure 1 shows all possible blocks of rank 0, rank 1, and three blocks of rank 2 (excluding ones made by symmetry transformations).

Let us denote the interior of set A by int(A) and boundary of A by fr(A). A set of rectangles

R = {Ri}, i=1,2,...,n is partition of block P if

1. int(Ri) ∩ int(Rj) = ∅, i != j, i,j=1,2,...,n

2. ∪{Ri, i=1,2,...,n} = P.

There is an evident relation between the structure of block and its partition. So we will use same letter P to denote both the block itself and it partition. Rectangles Ri are called atoms of the partition. The rank of partition is the rank of the partitioned block.

Articles [6-7] propose an approach to find the "best" placement of isotetic blocks based on the sequence of their pairwise merging. This article is intended to show an existence of placements which cannot be constructed by any sequence of pairwise merge operations over isotetically convex blocks of limited rank.

Let us introduce the following notations:

- V is set of vertices of P atoms;
- F is aggregate of atom boundaries:
  F = ∪ {fr(Ri), i=1,2,...n};
- L is partition of F:
  L = {S=[V1,V2] | (S ∩ V) = {V1,V2}}.

Now we can introduce the notion of cutting. An ordered sequence C=(S1,S2,...,St) of segments of partition P cuts partition P if:

- Si ∈ L, i=1,2,...t;
- (Si ∩ Sj) = Vij, Vij ∈ V if
  and only if i=j+1, or j=i+1; i,j=1,2,...t;
- (Si ∩ Sj) = 0 if and only if
  i != j+1 and j != i+1, i,j = 1,2,...,t;
- (C∩ fr(B)) = (Vb, Ve}, where
  Vb = (S1 ∩ fr(B)),
  Ve = (St ∩ fr(B), Vb,Ve ∈ V.

In other words, cutting is a simple path consisting of segments of L such that it first and last points belong to boundary of block P. C is "through" cutting if Vb and Ve belong parallel segments of boundary of P. Otherwise cutting is called as "sidelong".

Every cutting C splits the partition P into non-empty subsets P1 and P2. Herein we will consider only cuttings which intersections with any horizontal or vertical line are connected. In this case ∪(P1) and ∪(P2) are blocks.

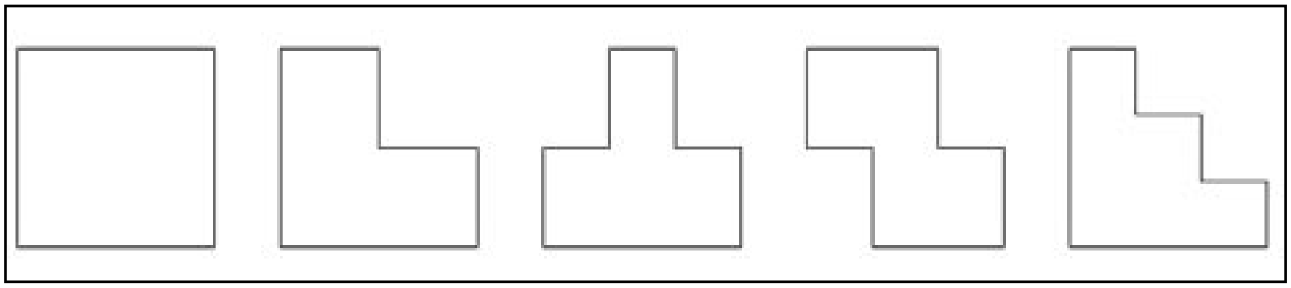


Figure 1. Simplest blocks of bounded rank.

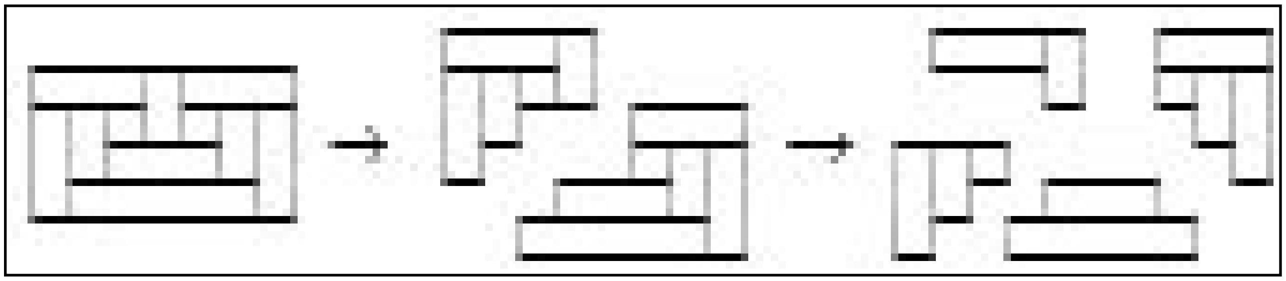


Figure 2. Recursive cutting onto blocks of rank 2.

Cutting C2 intersects cutting C1, if C1 cuts P into partitions P1 and P2, and C2 is cutting either P1 or P2, and just one of end points of C2 belongs to C1.

Let us define the direction of cutting:

$$\text{dir}(C) = \begin{cases} 1, & \text{if sequences of X and Y coordinates of vertices are both monotonous;} \\ 0, & \text{otherwise;} \end{cases}$$

We will say that block (partition) P can be recursively cut into blocks of rank r if partition P contains cutting which splits it into non-empty partitions P1 and P2, ranks of P1 and P2 do not exceed r, and both blocks P1 and P2 also can be recursively cut into blocks of rank r. It is supposed that a single atom can be cut into blocks of any rank. Figure 2 shows three first steps of cutting rectangle onto blocks of rank 2.

A set of all partitions recursively cut only onto blocks of rank >= r is denoted via P**r.

Theorem. The set P**r is not empty for any positive integer r. In other words, for any positive integer r there exists partition which cannot be cut onto blocks of rank r.

To prove the theorem, we will show how to construct an example partition P of square which belongs to P**r for every pre-defined natural r. The construction will be made on 2-dimensional regular integer grid.

Partition P1 from P**1 can be pointed out directly. It is the first "snail" drawn on Figure 3. Any through cutting splits it into blocks of rank 1. Sidelong cutting splits the "snail" into blocks of ranks 0 and 1. So one-dimensional "snail" belongs to P**1.

Partition Pr from P**r can be constructed as a square containing r rows and r columns. Cell Kij located at the intersection of row i and column j, i+j=0(mod2), i,j = 1,2,...r, contains "snail" P1. Cell Kij, i+j=1(mod2), i,j=1,2,...r, contains partition ~P1 which is mirroring of P1 with respect to horizontal or vertical axis. Figure 3. shows example partitions P1, P2, P3 and the structure of cells for example partition P3.

Partition Pr inherits main property of partition P1. Namely, a part of any cutting bounded by bottom (upper) side of cell Kij and bottom (upper) side of source square contains i-1 (r-i) steps. If this part is bounded by cell side and bottom (upper) side of the source square, then it contains i (r-i+1) steps.

Now we are ready to show, that any sequence C^ =(C1,C2,...,Cq) of cuttings Pr onto atoms will create blocks of rank >= r. Blocks created after every cutting from C^ can be considered as central (ones contain symmetry center of source square) and peripheral blocks. Peripheral blocks are out of our interest. We will estimate ranks of central blocks. So cuttings peripheral blocks can be instantly removed out of C^, and, after this operation, C^ can be considered as sequence of cutting details out of central point.

Pr consists of r rows and r columns. As it was noted, every through cutting contains r steps. It is true, because maximal available length of every segment in Pr is 3 units and every cutting started and finished on opposite sides of the initial rectangle contains r steps and, consequently, it creates blocks of rank r. So, let us assume C^ does not contain through cuttings.

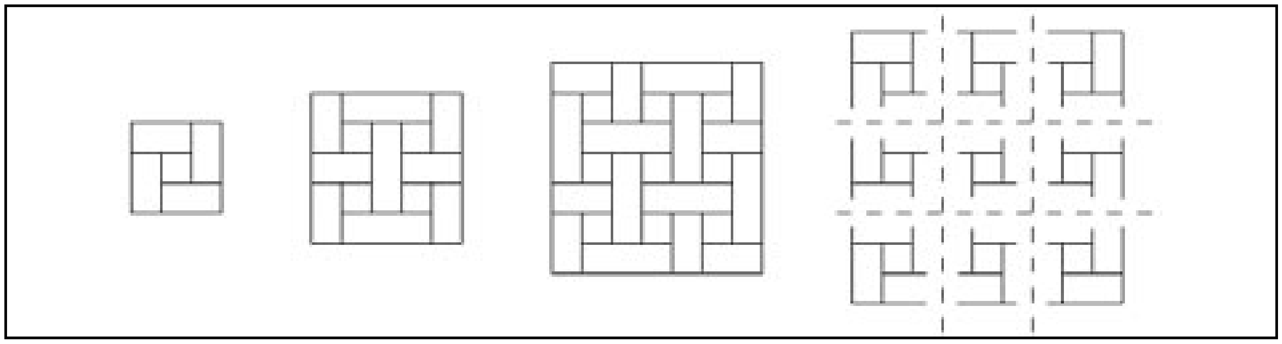


Figure 3. Based "snail" and generated partitions Pr.

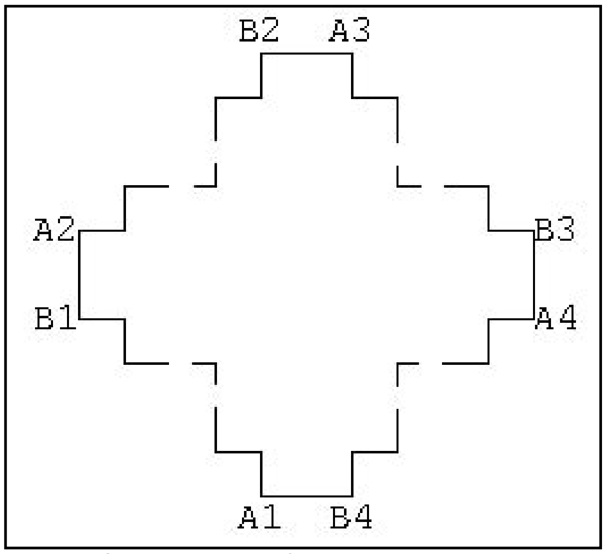


Figure 4. General structure of isotetically convex block.

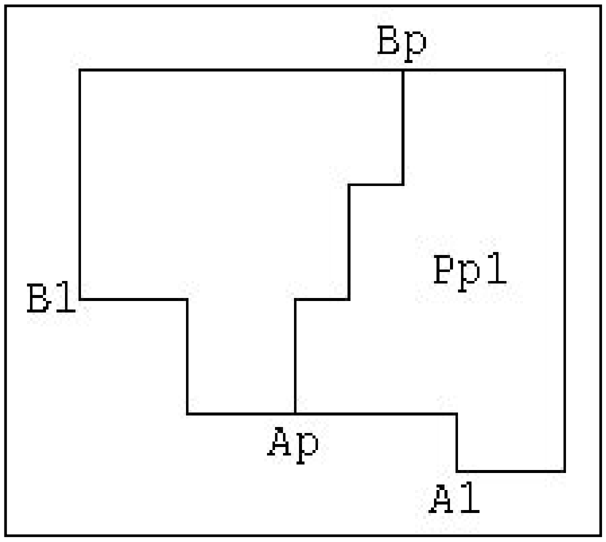


Figure 5. Case 5 of cutting.

The last assumption means that C^ contains at least one sidelong cutting for the original square and one additional cutting which intersects the first one. Otherwise it would be impossible to cut out central atom of source partition. Let denote this additional cutting by Cp:

p = min{k | (Ck ∩ Ck^) != ∅}, where Ck^ = ∪ {Cj, j=1,2,...,k-1}.

As it is noted in [4], after C1,C2...,Cp-1 cuttings the central block is bounded by at most 4 stairways. Its general view is shown at Figure 4. Segments F1=(B 1,A2),...,F4=(B4,A1) are parts of boundary of the initial square. So, without loosing the correctness, we can assume that first p-1 cuttings in C^ can be replaced by 4 ones: C1′, C2′, C3′, C4′, where Cj′=(Aj,...,Bj), j=1,...,4.

Now let us consider structure of blocks being created after all possible configurations of cutting Cp. Let Ap and Bp be its first and last vertices, Ap ∈ C1′, Pp1 and Pp2 are central and peripheral blocks which are cut out by Cp. Taking into account the symmetry of configuration, we can consider only the following 6 different cuttings:

1. dir(Cp) = dir(C1′), Bp ∈ F1;

2. dir(Cp) = dir(C1′), Bp ∈ Ck′, k=2,3,4;

3. dir(Cp) != dir(C1′), Bp ∈ C2′;

4. dir(Cp) = dir(C1′), Bp ∈ F2;

5. dir(Cp) != dir(C1′), Bp ∈ F2;

6. dir(Cp) != dir(C1′), Bp ∈ C3′.

In the first case the block Pp1 is bounded by (A1,...,Ap,...Bp). It is sidelong cutting. The structure of the block Pp1 remains the same as shown at Figure 4. We can assume it is built from the original square with the help of cuttings (A1,...,Ap,...,Bp), C2′, C3′, C4′. If we replace block Pp by block Pp1 and cuttings C1′ and Cp in C^ by cutting (A1,...,Ap,...,Bp), then we can recursively consider sequence C^ again without loosing of correctness.

In the second case the block Pp1 is bounded either by cutting (A1,...,Ap,...Bp) or cutting (B1,...Ap,...Bp). We can reduce this configuration to configuration of the general case again. With this goal let us repair the central block Pp by gluing peripheral block split by C1′ and replacing cuttings C1′ and Cp in C^ by boundary of just created Pp1.

The third case can be reduced to the second case by re-orientations.

In accordance with assumption cutting cannot go through a boundary of the central atom. So reductions made during consideration of first three cases are correct and we will meet in C^ cutting made by one of rest cases.

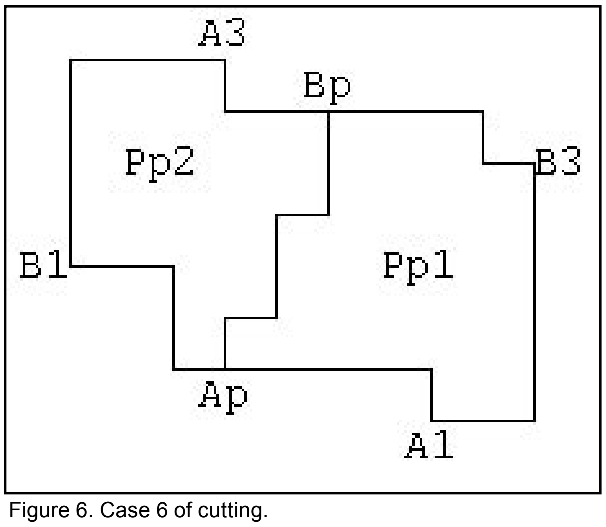


Figure 6. Case 6 of cutting.

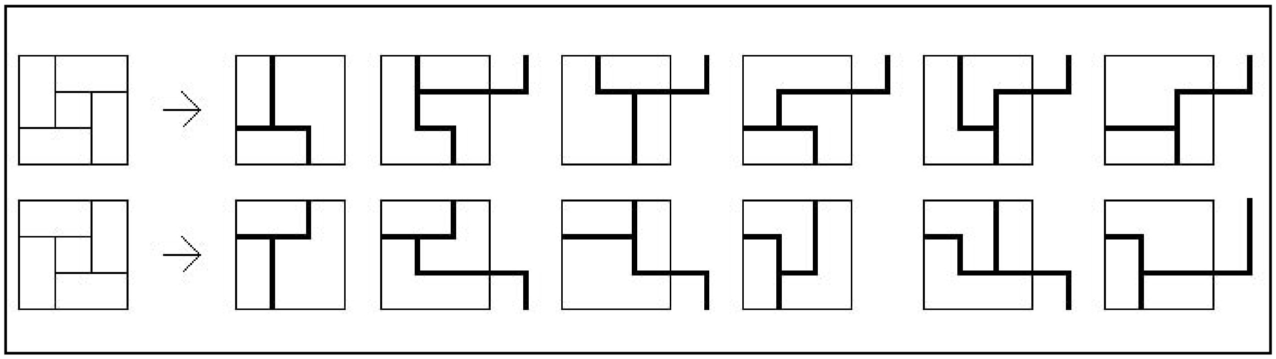


Figure 7. All possible intersections of cutting inside cell.

In the fourth case the block Pp1 will be bounded by (A1,...,Ap,...,Bp). It is through cutting.

Now we consider case 5. Illustrations for it is shown at Figure 5. Let Ap ∈ cell Klm. All available intersections of cutting C1′ and cutting Cp inside Klm are shown at Fig 7. Boundaries of the cell are drawn by thin lines. Parts of C1′ and Cp bounded by l-th row are drawn by thick lines. Let us denote the contribution of the l-th row into the rank of block Pp1 by Sl. It can be easily seen that Sl = 1 (for intersections 1,3,5,6,7,9,10,11,12) or Sl = 2 (for intersections 2,4,8). Now we can estimate the number of steps of the left boundary of block Pp1: (l-1) + Sl + r-l) >= r.

The last case is illustrated at Figure 6. Let us assume that Ap ∈ Klm, Bp ∈ Kkn. Let us denote contributions of rows l and k into rank of the block Pp1 by Sl, Sk, and contributions of columns m and n into rank of the block Pp2 by Sm and Sn. From the properties of the initial "snail" it follows that:

Sl, Sm, Sk, Sn >= 1.

the following conditions are necessary for Pr to be recursively cut into blocks of rank < r:

(l - 1) + Sl + (k - l) + Sk + (r - n) < r;

(m - 1) + Sm + (n - m) + Sn + (r - k) < r.

Incompatibility of these conditions shows that there is no way to build sequence of cutting in class of isotetic blocks with apriory bounded number of vertices. It finishes the proof.

The author wishes to express his gratitude to N.Metelsky for formulation of the problem and kindly advices.

Summary

There exist a partition of the square into rectangles which cannot be synthesized by pairwise block merging procedure in the class of convex isotetic blocks with apropriory bounded number of vertices.

References

[1] Busnyuk N.N., Sarvanov V.I. // Vesci AN BSSR.

[2] Busnyuk N.N., Graphs of partition of rectangle into rectangles. Preprint (36(306)) of the Mathematical Institute, Belarussian Academy of Sciense, 1987.

[3] Klebanovich D.M., Metelskiy N.N. Grid of macro-blocks for channel model of VLSI. Preprint (6(316)) of the Mathematical Institute, Belarussian Academy of Sciense, 1988.

[4] Wood D. // Computational geometry, Toussaint G.T. (editor). North Holland): Elsevier Science Publishers, 1985, P.431-459.

[5] Metelskiy N.N., Krikun V.S. Isotetic boundaries of limited rank, type and kind. Preprint (10(410)) of the Mathematical Institute, Belarussian Academy of Sciense, 1990.

[6] Azarenok A.S., Klebanovich D.M., Krikun V.S. and others. Automation of VLSI design. Method of hierarchical generation VLSI layout on the base of fixed non-standard blocks. Preprint (16(316)) of the Mathematical Institute, Belarussian Academy of Sciense, 1990.

[7] Metelskiy N.N., Krikun V.S. The method of hierarchical placement of isotetic blocks. Preprint (27(427)) of the Mathematical Institute, Belarussian Academy of Sciense, 1990.

Calendar of Events

September	October
1	1
2 SISPAD - Belgium	2
3 SISPAD - Belgium	3
4 SISPAD - Belgium	4
5	5 SOI Conference - Florida
6	6 SOI Conference - Florida
7 ESSDERC '98 - France	7 SOI Conference - Florida
8 ESSDERC '98 - France	8 SOI Conference - Florida
9 ESSDERC '98 - France	9
10 ESSDERC '98 - France	10
11 ESSDERC '98 - France	11
12	12
13	13
14	14
15	15
16	16
17	17
18	18
19	19
20	20
21	21
22	22
23	23
24	24
25	25
26	26
27 BCTM Conference	27
28 BCTM Conference	28
29 BCTM Conference	29
30	30
	31

Bulletin Board

NT-based CAD Design Tools Exhibited at ICCAD

Silvaco will be exhibiting the latest developements in the PC-based NT Design Framework ***CELEBRITY*** at the ICCAD Conference from 8th-12th November in San Jose, CA. New features of the layout editor ***Expert*** including the DRC On-Guard feature described in this issue will be presented.

CAD Division Moves to New Building

Silvaco's CAD Division has moved into their newly refurbished ultra modern building. The CAD building is the fifth Silvaco building at the Santa Clara site. The CAD team was recently bolstered by 6 new development engineers and this dedicated facility gives room for rapid expansion in the coming months.

Korean University Standardize on Expert

Over 60 Korean Universites have decided to standardize teaching of students in layout and DRC techniques using ***Expert*** and ***Savage.*** This effort is organized and supervised by KAIST, a leading science institute. This confirms the leading capabilities of Silvaco CAD products and paves the way for the future of Silvaco tools in the Korean market.

*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. 9, No. 9, September 1998 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

Some questions from previous issues are repeated here since continued enhancements are introduced in ***Expert*** to improve designer's productivity.

Q: I run DRC, find one-two violations, correct them, then I re-run DRC to check whether my corrections worked. However re-running on the whole design is time-consuming. How can I run DRC over a piece of the layout, in the vicinity of the introduced changes?

Recent release of ***Expert*** contain two important features that avoid multiple batch DRC reruns for the whole design.

- "DRC in Area" command allows you to run DRC script to be run it the selected rectangular area of the layout.
- "DRC Guard" allows monitoring of design rule violations on the fly while editing the design.

Please read the release notes for ***Expert*** v. 156 for details. (All release notes are accumulated in the file relnotes.txt).

Q1: When I plot my design on a laser printer some regions are not plotted at all.

Q2. Stipple patterns on screen and on plot look differently.

A: In both cases the reasons are the same. Unfortunately, color capabilities of screen and plotting devices often do not match. In particular, some nonsaturated colors clearly visible on the screen are mapped into white color on the plot.

One more problem arises if you try to plot a layout drawn in stipple mode of filling. When preparing data for plotting the software tries to match RGB values of colors from layout to the available set of plotter colors. To do so, plotters use dot patterns of primary colors. These dot patterns may be either regular or diffuse ones. Stipple mode of filling also uses bit patterns for filling. When plotter's color dot patterns and stipple patterns interact, the results can be totally unpredictable. Therefore color matching has chance to work correctly for solid-filled and wireframe shapes only.

Expert provides two means to alleviate these problems:

- plotting style setup, see Figure 1;

- color libraries.

Plotting style setup, among other possibilities, allows you to assign filling modes and colors for layers different from the ones used on screen. In particular, in troublesome situations is is possible to replace stipple filling by hatching. Hatching is filling by "wireframe" lines rather than by dot patterns, therefore it does not interfere with color matching tricks. However this advantage is linked with a drawback: there are six possible types of hatching, see Figure 1.

Color libraries allow you to define "named" colors. This allows you to predefine a set of "good" colors for further use in setups, see Figure 2.

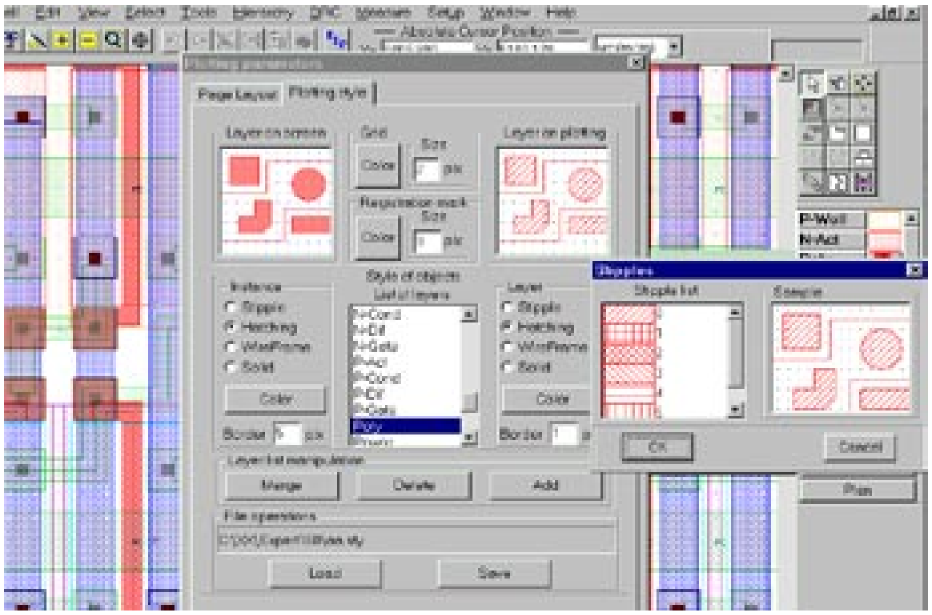
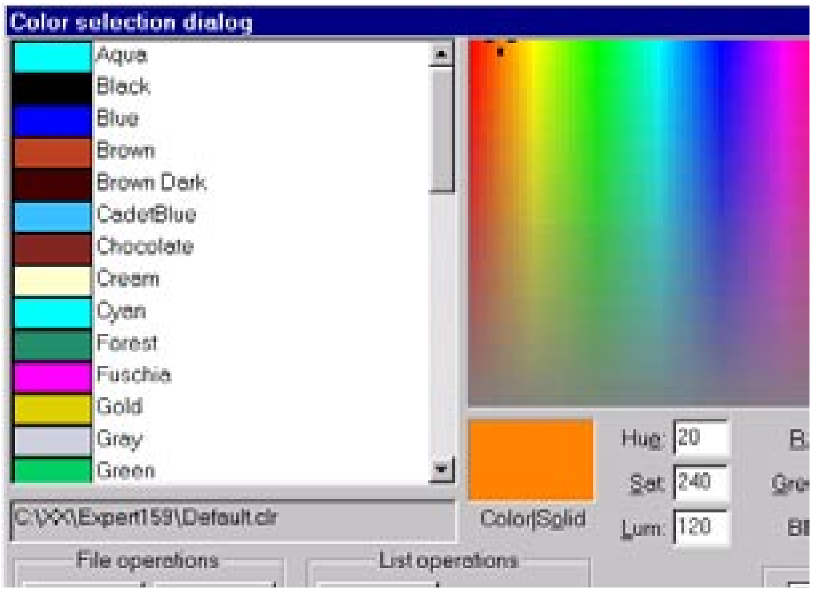


Figure 1. Style Setup.

Moreover, color libraries for plotter and for screen may contain colors with the same name, but with different numeric RGB values, so that the named color on screen will look exactly the same as the color with the same name for printer/plotter. Such separate color definitions may be prepared for all types of plotters you have.

Of course, such kind of color/style setup requires some tedious tuning, including trial printouts. But please remember, color library tuning must be performed only once.



C:\XX\Expert159\Default.clr

Hue: 20
Sat: 240
Lum: 120

Figure 2. Color Setup.

Q: *Expert's* tool bars occupy too much screen space. I would like to have more space for layout.

In ***Expert*** the whole screen may be used for editing in the following way:

- using menu Setup>>Customize assign shortcuts to most frequently used commands and/or add them into the custom menu (the one appearing if you depress right mouse button inside design area).
- remove all unnecessary toolbars;
- assign shortcuts to show/hide necessary toolbars; (suppose you assign key "L" to the Layer bar)
- holding the CTRL key, drag the necessary toolbars so that will not dock to the boundary of ***Expert's*** window.
- then if you press "L" several times, you will see that the layer bar shows and hides, while the layout occupies the whole screen.

Q: How can I report long parallel interconnect segments that run too close to each other?

A: An example check is as follows:

OutDistance:

layer1=m1, layer2=m2, type=LT, value=0.25um, options=(S1,P,O,L=GE 6um,L1=GE 5um,L2=GE 4um);

Actually, it was generated through the user interface of the DRC script panel. Click "Check" at DRC panel, then select a basic operation, then click "Options" button.

The following options were selected at the "Options" panel:

- parallel
- with projections
- length of segments
- length of errors 1
- length of errors 2
- report error subsegments.

Users should try different combinations of these options (and their opposites) and choose what kind of check is required.

A more advanced check, similar to the PLENGTH command, is under development.

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