SPACE 3D SUBSTRATE EXTRACTION APPLICATION NOTE

S. de Graaf

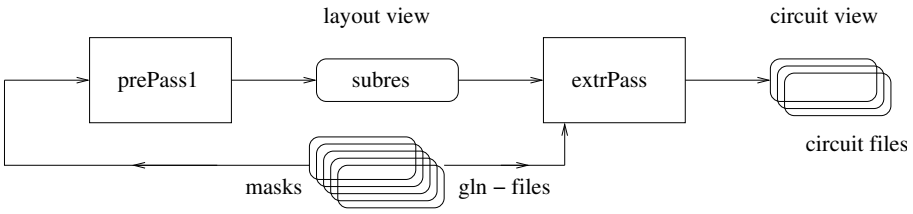
Circuits and Systems Group
Faculty of Electrical Engineering,
Mathematics and Computer Science
Delft University of Technology
The Netherlands

Report EWI-ENS 03-04
October 1, 2003

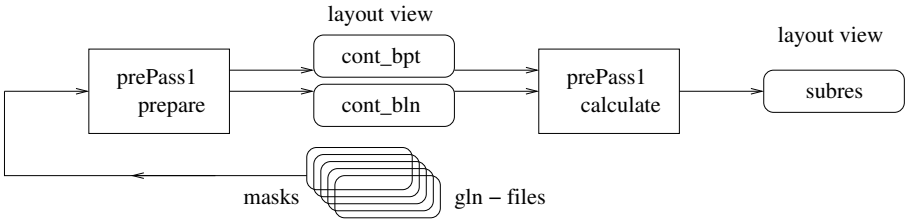
Last revision: June 14, 2004.

1. INTRODUCTION

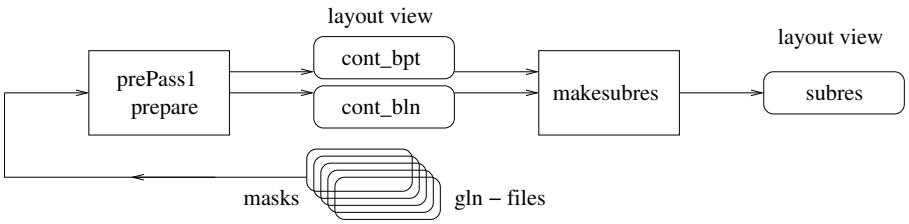
This application note describes an improved *space3d* prepass one for the accurate (3D) substrate resistance extraction method, which is initiated with option **-B**. The method uses a substrate spider mesh to calculate the substrate resistances. The calculation results are written in the cell layout view "subres" file. In the *space3d* extract pass, the results are read again and the substrate resistances are added to the extracted circuit network. See the figure below.



Because the edges of all masks can split the substrate terminals in small tiles, the mesh for the substrate terminals is also depended of all masks. This is an unwanted behaviour. With the improvement, the first *space3d* prepass is split into two passes, whereby the first pass writes the substrate terminal information into two files and whereby the second pass calculates and writes the "subres" file. See the figure below.



For this new special calculation prePass1, *space3d* runs another program, this program was at first moment *space3d* with option **-%1**. In second phase, i have changed this in a call to *makesubres* (see figure below). Currently, *makesubres* is a symbolic link to the *space3d* program. In the future, this part may be replaced by an user program. This program must use the same extraction environment, *space* parameter file and technology file and must know the extracted cell name and project path.



2. IMPLEMENTATION

2.1 The Prepass1 Prepare Pass

The old prePass1 is changed, such that it does the prePass1 **prepare** pass. This prePass1 is done in optSimpleSubRes (**-b**) mode. Variable optSubResSave remembers that prePass1 is the prepare pass. Note that stream "cont_pos" is now written by option **-B**.
The following variables are set: prePass1, optCap3D, optSimpleSubRes, optSubResSave, substrRes, runSpecialPrePass.
The following variables are not set: optSubRes, optRes, optCap.
Note that function prepass (see scan/sp_main.c) does the needed variables switching (for optSubResSave, optSubRes, optSimpleSubRes).
Function run2 is added to start the special prePass1 calculation program. This new run2 function was needed, because run can not use multiple program arguments. There is also added a new function (_dmRun2) to the database library (libddm). Note that the special prePass1 program is not run when using *Xspace* with option **-Z** (optOnlyPrePass), see variable runSpecialPrePass.

Function enumPair (see extract/enumpair.c) shall call function contEdge and contPoint (see extract/contedge.c). Function contEdge (for writing "cont_bln") is called for all non-vertical substrate contact edges. It is also called for interior edges of different substrate contacts, how they lay against each other, but only for not joined (subContJoin) substrate contacts. Function contPoint (for writing "cont_bpt") is called for terminals, which are laying on a vertical edge between two substrate contact tiles (optIntRes must be TRUE). Note that "cont_bpt" is not more written after revision 1 (Rev1) of the source code.

Note that function enumTile (see extract/enumtile.c) shall call function contTile (see extract/contedge.c) for each tile with a substrate contact reference. For distributed contacts the function shall try to remove unwanted edges.

2.2 The Prepass1 Calculate Pass

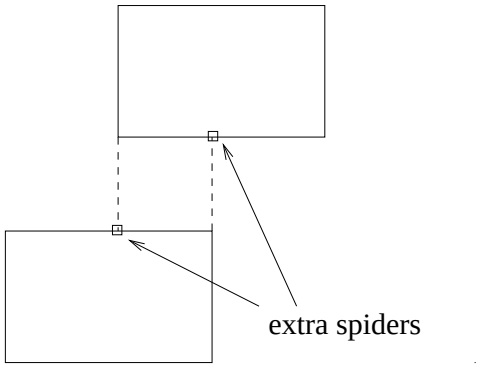
This new special prePass1, the **calculate** pass, reads the substrate contours. If *space* does this pass internal, then function enumPair calls pp1EnumPair to handle this.
The following variables are set: prePass1, optSubRes, substrRes, optCap3D, optOnlyPrePassB1, optNoPrepro, optOnlyPrePass.
The following variables are not set: optSubResSave, optRes, optCap.

The scan input functions are modified for the special prePass1 (optOnlyPrePassB1 is TRUE) to read the substrate files ("cont_bln" and "cont_bpt"). See in "scan/input.c" functions openInput, doFetch and fetchTerm. The "cont_bln" file is read by function doFetch. The substrate record information must be stored in the edge data structure. The gboxlay.chk_type gives additional information about the substrate contour edges. This information is also stored in the new "cc" member of the edge data structure. Only the lower 8 bits are used for storing a causing conductor number. An unique color must be found for each causing conductor. To achieve this, the special prepass uses another

masktable color sceme. Note that the highest substrate conductor number (nrOfConductors-1) is used for substrate terminals of transistors (which have cc# -1). The "cont_bpt" file is used by optIntRes, when terminal points can split the tiles. Function fetchTerm does not read terminals (nrOfTerminals is 0), but reads the terminal points. Note that "cont_bpt" is removed after revision 1 (Rev1) of the source code.

The edge "cc" information is also stored in the tile "known" member. This is done by giving the createTile function an additional argument (see the modifications in "scan/tile.c" and "scan/update.c"). The tile "known" information is used in function pp1EnumPair to set the "distributed" flag and "causing_con" in the substrate contact reference (see extract/enumpair.c). Note that the special prePass1 does not use element recognition. Note that function enumTile (see extract/enumtile.c) does only function spiderTile.

The substrate resistances are calculated with a mesh. In the corners of the mesh are spiders. Some unwanted spiders are still generated on the mesh edges (see figure below).



Function newSpider does also a call to stripAddSpider, which function adds the spiders in a strip. However, after that the spiders are not easy to remove again. Thus, now the spiders are added by function spiderTile (see spider/sptile.c) in the strip. In the special prePass1 spiderTile shall remove unwanted spiders. Function stripAddSpider must also be called on other places where function newSpider is called. There are also made modifications in source file "spider/strip.c".

Other modifications to *space* for the special prePass1 program are:
See also "scan/sp_main.c". First of all, adding the special option **-%1** to set *space3d* in the special prePass1 program mode. The special prePass1 does not use the circuit view (circuitKey). Thus, function initOut (see lump/out.c) cannot dmUNLINK circuit view files. The special prePass1 does not use the functions initHierNames, readTid, endHierNames and disposeTid. Note that in the special prePass1 a number of variables may not be TRUE (like optResMesh). In new *space3d* it is not useful that option **-i** gives cap3d statistics, while optCap3D was only used in the special prepass for optSubRes.

2.3 The New Extract Pass

Note that in the extract pass (extrPass is TRUE) also variable optRes is TRUE. Note that optRes is TRUE for both substrate resistance and/or interconnect resistance extraction. Function resEnumTile is called by enumTile (see extract/enumtile.c). This function handles also the distribution of the substrate contacts (functions makeContact, subnodeSubcontReconnect and subnodeSubcontEmpty). Now it is possible, that distributed substrate contact nodes are joined. Therefor the node "substr" member must be set to type 2. Thus, function subnodeFindSubcNode can find back this node type.

2.4 Using Old Mesh Method

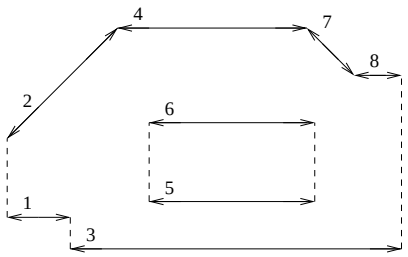
When the old *space3d* **-B** method must be restored, the *space* program had to be compiled with '#define NEW_B1' put off. This define had to be changed in two source files (scan/sp_main.c and extract/enumpair.c). I have changed this in the last source delta's. Now, parameter "sub3d.old_mesh" can be used, to restore the old mesh method.

2.5 Using Internal Makesubres Method

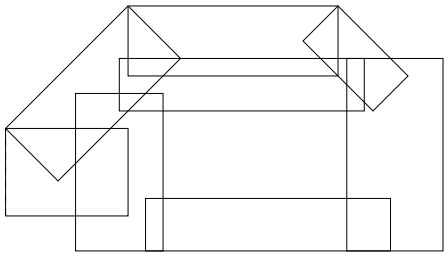
The *makesubres* program is default a symbolic link to the *Xspace* program. Thus, *space3d* is calling itself to perform the special prepass. For using *Xspace* graphic functions with option **-B**, it is important to use the internal *makesubres* (or special prepass) method. Else, it is only possible to use *Xspace* with option **-%1**. I added this feature in the last source delta's. The internal method can also be set with parameter "sub3d.internal".

3. IMPLEMENTATION DETAILS

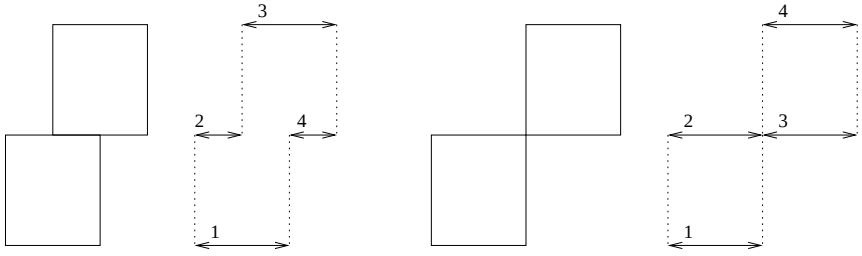
First, we must know something about "mask_gln" streams, which are read by the scan function of the *space* program. A "mask_gln" stream is produced by *makegln* and contains a set general line segments of one mask. These (sorted) non vertical line segments represent the contour of the mask (see the figure).



The *makegln* program produces this "mask_gln" stream by reading all the layout elements of one mask. Most layout elements are boxes (rectangular shapes) read from the stream "mask_bxx" (and "t_mask_bxx" for terminals of an interconnect mask). The non-orthogonal layout elements are read from the stream "mask_nxx". For example, the above mask layout can be build up from the following mask elements:

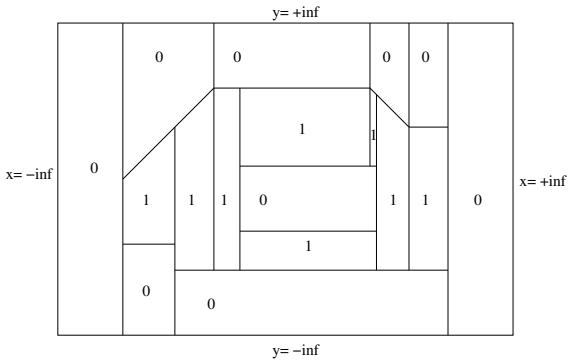


Two other examples of "bxx" to "gln" conversions are given below.

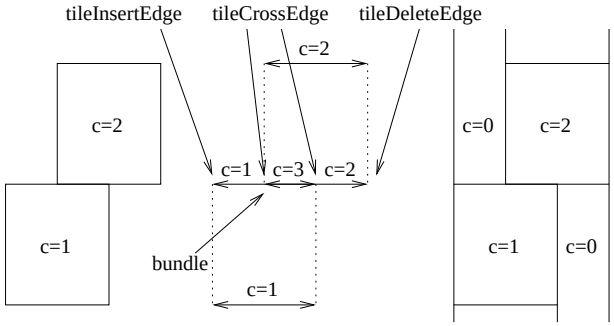


In the last example, the two middle edges are not merged to one edge. In the first example, edges 1 and 4 are start edges, and 2 and 3 are stop edges. In the last example are edges 2 and 4 stop edges. Note that edges for one mask cannot overlap each other.

The *space* program uses a horizontal scan technique. It scans from lowest x position to highest x position all the edges of the "gln" files. The edges read split the layout design surface into tiles. And all these tiles have a color. The tiles above start edges get the color of the start edge mask. The tiles above stop edges remove the color of the stop edge mask. There is always started with a tile, which has no color. An XOR operation is used to add or remove the edge color to the tile color. The first figure is split-up into the following tiles.



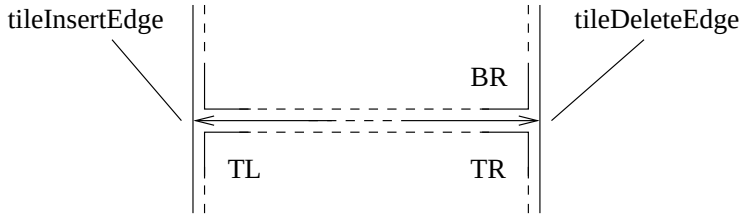
Each mask has an unique color bit. Thus edges of different mask can easy be recognized. Each tile has a color bitmask that represent the existing masks in that layout area. Edges of different masks, which overlap each other, are merged by the scan bundle function. Two edges, which touch each other, are not bundled. For an example, see the following figure:



A tileInsertEdge and a tileDeleteEdge gives always a tile split. But, because of the bundle operation, a tileCrossEdge gives only above the edge a tile split, when there are different colors. The bundle uses an color ADD (OR) operation to set the color for each edge part in the edge bundle. Note that an edge can only be a start or stop edge. Edges with the same color cannot overlap each other. There cannot be interior edges. When we want to use the "gln" scanning method for the contours of the substrate terminals, we must exactly know how the scan function works and what is possible. And we must also know, how we can force horizontal or vertical tile splitting.

3.1 Horizontal Tile Splitting

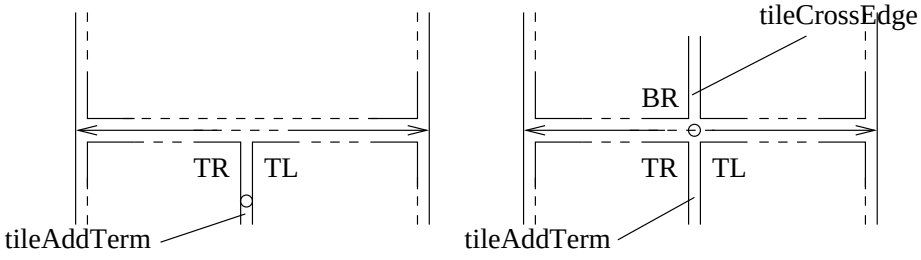
The mesh refinement method forces horizontal tile splitting. It adds a "mesh_gln" stream to the set of "mask_gln" streams. It is important to know, that the user may not define the name "mesh" as a mask name. The "mesh_gln" stream adds interior edges for all high res conductor masks that must be split. The interior edges has no color (no color bit is set). Thus, the edges do not change the color of the tiles. The horizontal splitting starts by a tileInsertEdge and continues till a tileDeleteEdge (see figure).



Thus, an interior edge can also force a vertical split at its edge start and end position. The start and end points force also a state ruler x position. By a possible bundle with other edges, the end position (xc position) can be an interior edge position and shall not split the tile above the edge by tileCrossEdge, because it does not set different colors.

3.2 Vertical Tile Splitting

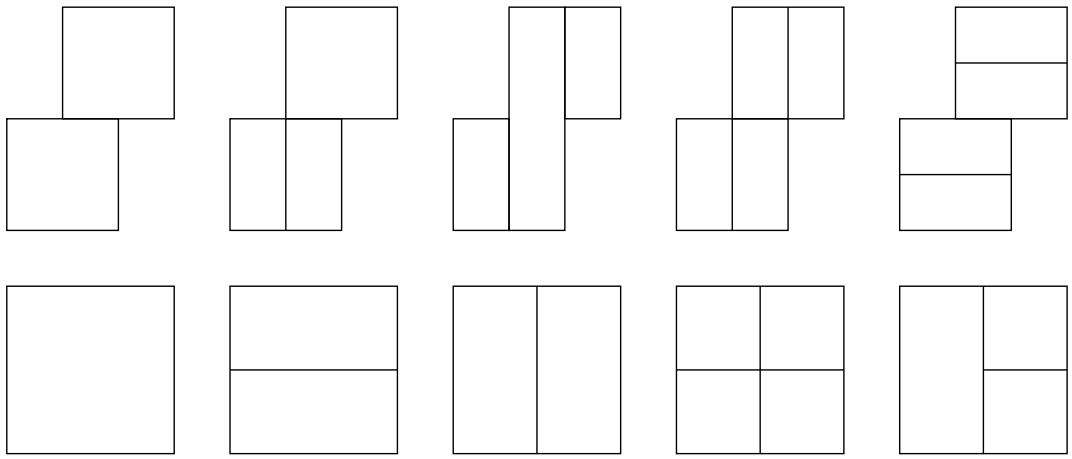
Thus we explained above, edge begin and end points give a vertical tile split, but some only above or below the bundled edge. And each edge gives also a horizontal split of the tile area. This horizontal split can only be avoided, when the edge length becomes zero. This happens with terminal points in the scan function. Terminals split tiles vertically (if needed), because terminal points must be on vertical edge positions. Note that a terminal point on a horizontal edge splits both the tile below and above the edge. The following figure gives two examples.



Function tileAddTerm does the splitting below the edge and tileCrossEdge does the splitting above the edge.

3.3 Making Substrate Terminal Contours

Depending on input mask combinations, a substrate terminal can be build up out of any combination of tiles. The following figure gives a number of examples.



In the extract pass, the tiles are joined depending on the substrate mode chosen. Thus, the prepass must deliver exactly the number of substrate terminals, that the extract pass expect. Thus, the tile joining rules are very important. For example, tiles of distributed substrate terminals are not joined (or merged). Thus, we cannot optimize distributed substrate terminals.

3.4 Substrate Terminal Merging

Substrate terminal merging is dependent of a number of conditions and parameters.
Default, all substrate terminals are merged (var. mergeNeighborSubContacts ≡ true), independent of the causing conductor involved. But with parameter "sep_sub_term" (set to "on") this can be changed. In that case, we want separate substrate terminals for different causing conductors. This is only important for different substrate terminals, which touch each other. Note that tiles of the same causing conductor are merged. Note that only contacts and surface capacitors have causing conductors. Transistors don't have causing conductors and these substrate terminals cannot be distributed.

However, distributed substrate terminal tiles are never merged. A substrate terminal tile is distributed, when the following conditions are true:

1. interconnect res extraction is used (optIntRes ≡ true)
2. sub_term causing conductor res value > low_sheet_res (bit in tile known mask)
3. parameter "sub_term_distr_*xx*" is set for causing conductor mask *xx*

Resumé, two substrate terminal tiles are not merged, when (a) one of the tiles is distributed or (b) they have different cc numbers and "sep_sub_term" is set.

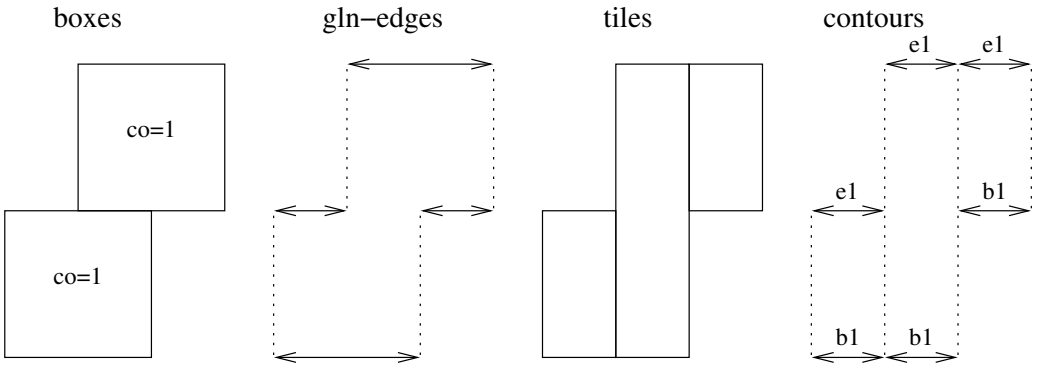
Note that substrate terminal tiles with the same causing conductor number can have different res values. Thus, one can be distributed and the other not distributed.

Thus, when substrate terminal tiles are not merged and have the same causing conductor numbers (because they are distributed), we need possibly an additional splitting strategy to preserve the tiles shapes. Each of these tiles is seen in the extract pass as one separate substrate terminal. Note that this additional tile splitting tricks are needed for the special prepass (the *makesubres* step), which use only the substrate terminal tiles as input. Note that no tiles are used as input, but non vertical edges.

We can only preserve the correct tile shapes for the substrate terminal tiles in the prepass, when we get exactly the same tile splitting in the prepass as we get in the extract pass. Thus, the sub3d and cap3d be_window (bandwidth) must be used in both passes.

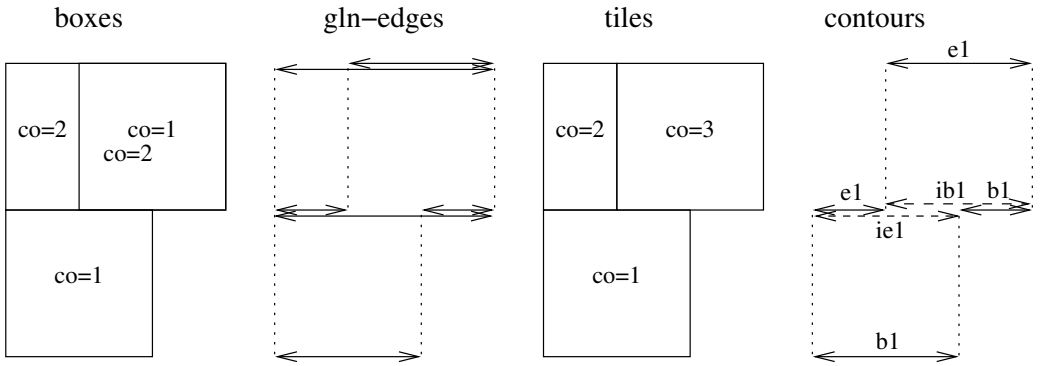
3.5 Substrate Contour Examples

Example 1



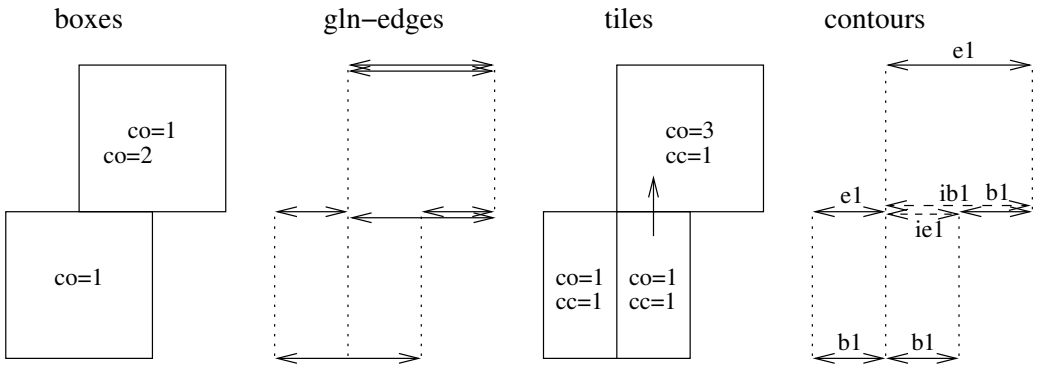
In the most simple case (see above), when there is only one mask (layer), we need only to read the "gln" edges of that single mask (there is only one color). The scan function reads the edges and makes tiles. The edge x-begin and x-end points breaks the mask area into three tiles. Due to this scanning technique, the tiles are as long as possible in vertical direction and can be (depending on the edge begin/end points) very small in horizontal direction.
When the substrate area is distributed, the above contour begin ("b") and end ("e") edges are generated. The number behind the letter "b" or "e" indicates the causing conductor number. These separate set of contour edges give the same tiles back after scanning. In the non distributed case, the begin and end edges are merged. Also in that case the default tile splitting (as above) is done. Note that separate begin / end edges give also a tile split below and above the edges.
Note that only beginning contour edges set or change the tile known mask. This mask is used in the special prepass to find back the substrate cc# and distributed flag. Thus, the scan bundle function must not change the edge→cc flag for end edges.

Example 2



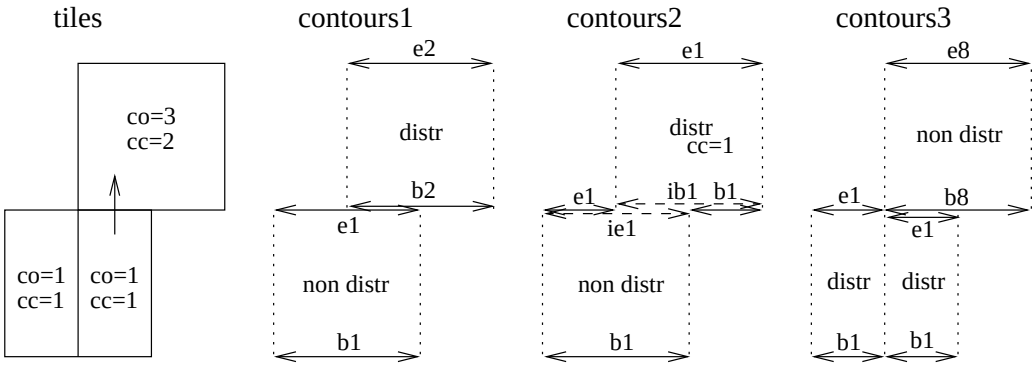
When there is also a second mask (with color "co" ≡ 2), then there are two different "gln" streams to read. The begin edge of the second mask merges with an "e1" edge and a "b1" edge of the first mask. And because of these mergings the tile splitting is completely different. To get the same tile splitting result with the contour edges (by distributed substrate), the above interior ("i") edges are generated. These interior edges overlap each other, thus the edges are merged with each other. Note that an interior edge does not change the tile color. Note that the previous separate "b1" and "e1" edges are not more separate edges. This example shows that another mask can change the tile splitting result. The following example shows another splitting possibility.

Example 3



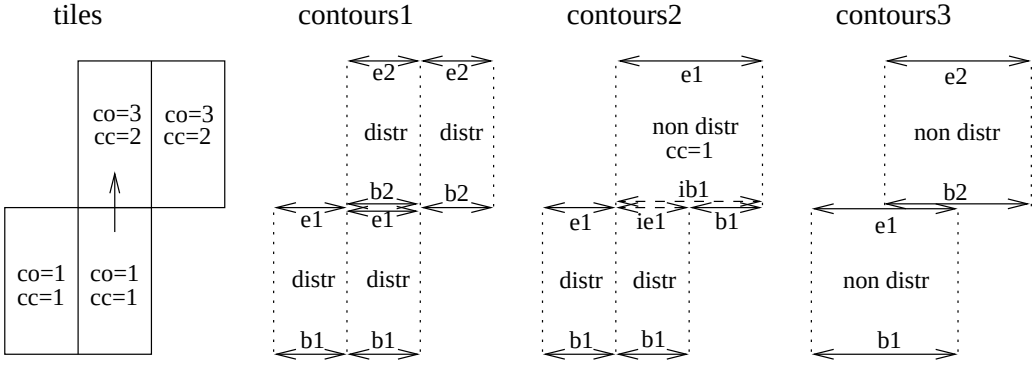
In this example (see above), the box of the second mask begins at the same position as the box of the first mask. The begin edge of the second mask merges only with a begin edge of the first mask. Therefor, the substrate area has another tile splitting as in example 2. As a result the above set of contour edges is generated for distributed substrate. Note that the small interior edge "ie1" is not required, but the contour procedure generates for each tile always a begin and end edge. And the procedure does not merge the begin edges "b1", which are in different tiles. However, edges (of the same type) in the same tile are merged.

Example 4



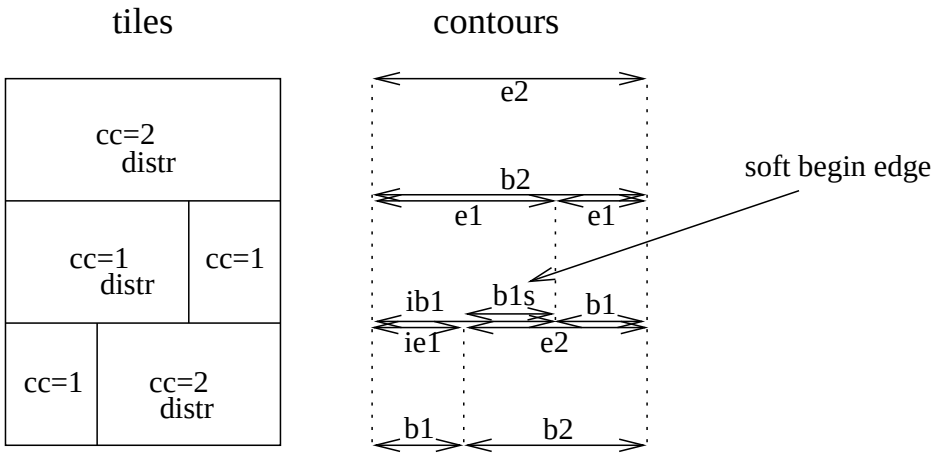
In this example, we see the same tile splitting as in example 3. But now the second mask is responsible for another causing conductor number (cc# ≡ 2). And this causing conductor is maybe not distributed (depending on parameter "sub_term_distr_*xx*"). Note that a second mask can also change the resistivity value of the same causing conductor (depending on technology file) from high res to low res. And in that case, the substrate area becomes also not distributed. Note that the merge of the non distributed area's with different causing conductors is also depending on parameter "sep_sub_term". Thus, the generated set contour edges is different for each case.
Note that for "sep_sub_term off" (the default case) all substrate terminals are merged, except the distributed ones. In that case the highest causing conductor number is used for all merging substrate terminals. In the last contour above (contours3) cc# 8 is used as an example of this case.

Example 5



In this example, we have used the resistance mesh refinement option **-z**. This gives the above tile splitting for conductor tiles. The contour examples 2 and 3 are given for parameter "sep_sub_term on".

Example 6



This last example shall handle the scan function tileCrossEdge problem. When edges are bundled, only tileCrossEdge can split the tile above the edge. But by equal color, this split can only be forced by the termSplit flag. Thus, a beginning new edge (which is bundled) must possibly set the termSplit flag. But not all beginning edges may do this. To handle this situation proper, i added a new s-bit (0x800) to the edge→cc bitmask. When the 's' bit is set (flagging a soft begin), then the split may not be done.

NOTE:
The *dbcat* program (using verbose mode) can list all bits of a "cont_bln" stream.

The following table gives an overview of the bits in "cont_bln" stream record field 5, which is copied to edge→cc by reading.

0x0FF	causing conductor number (8bits)
0x100	interior edge bit
0x200	distributed substrate bit
0x400	end edge bit
0x800	soft begin bit

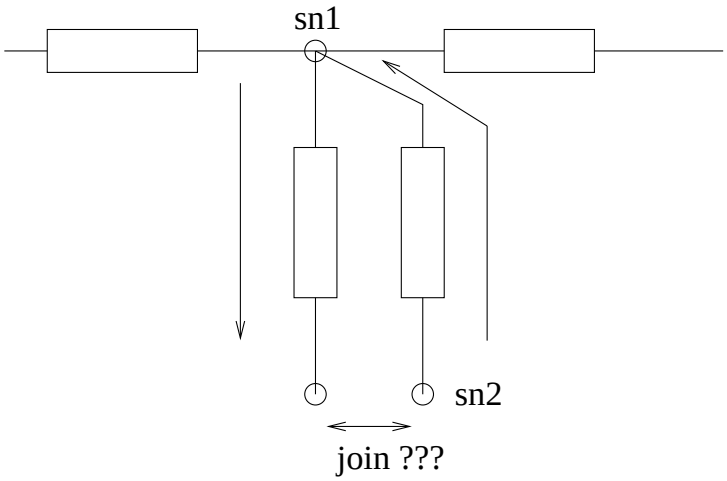
4. OTHER MODIFICATIONS

4.1 Only Center Terminals by optIntRes

See "scan/input.c". Now *space* makes only terminal points in the center of a tile by interconnect resistance extraction and not more for option optCap3D is FALSE. This is better to understand for the users of the *space* program. Note that this center point splits the tile vertically in two pieces.

4.2 Joining Distributed Substrate Contact Nodes

See "extract/enumtile.c" and "lump/lump.c". By identifying distributed substrate contacts (with node→substr ≡ 2), function subnodeFindSubcNode can find via the interconnect node and a contact resistance element another distributed substrate contact node part on the same place and join this nodes. Function readyNode may not delay this nodes. Last change to "lump.c" is made in the 4.91 delta. Function nodeJoin must not decrement the group notReady count, when the node was ready (when it does not have subnodes). And this has nothing to do with the node "ports" pointer.



4.3 Count of Area Nodes

See "lump/init.c" and "lump/lump.c". Option **-i** prints now also a line for all joined equipotential area nodes (variable areaNodes, n→area ≡ 2), like it does for equipotential line nodes (var. equiLines, n→area ≡ 1). This is counted in function makeAreaNode. Besides that, it prints the total number (variable areaNodesTotal) of seen n→area nodes by function readyNode. The printed information text is also changed. Note that the node area flag numbers are swapped after revision 1 (Rev1) of the source code. The highest number is most important.

4.4 Changed Info Text of Substrate Nodes

See "lump/init.c". Changed the text printed by option **-i** for substrate terminal nodes from 'nods' into 'nodes'.

4.5 New Space Options

See "scan/sp_main.c". Besides the new special option **-%1** for running only the special prepass, *space* has get two other new special options.
With special option **-%0** *space* skips re-running prepass zero (re-generation of "mesh_gln") when using **-z** (variable skipPrePass0 is set).
With special option **-%2** *space* runs only the extract pass (variables optOnlyLastPass and optNoPrepro are set). Note that for optSubRes is TRUE and optCap3D is FALSE also functions cap3dInitParam and cap3dInit must be done. Function cap3dInitParam calls get3DWindow for "sub3d.be_window" (the greenCase must be SUBS, see spider/cap3d.c). Function cap3dInit sets only the bounding box variables (Xl_3d, Xr_3d, Yb_3d, Yt_3d). Functions findStripForSubStart and findStripForSubNext (used in function scan) use also this variables.

4.6 Node Random Id's

See "lump/node.c". The node random id generator is now initialized for each *space* pass. Added function srand48 to function nqInit. Thus the nodes have always the same id's in each pass and *space* shall always choice the same node for certain node eliminations (in the extract pass).

4.7 Old Element Hashing Code

See "lump/elem.c". The ELEMHASH macro calculates a hash value based on the difference of the node id's. Therefor node nB→id must be greater (or equal) node nA→id. I have removed the old ELEMHASH code, which was also using a Sort array. I have also removed the calls of function enlargeElemHT from function returnElement, because it can better only be done in function elemNew.

4.8 No Green and No Schur

See "spider/cap3d.c". Option **-U** (var. optEstimate3D) and *Xspace* menu option and parameter "be_mesh_only") was only possible when compiled with #define DEBUG. The #endif DEBUG line is moved, thus it can now be used again.

4.9 Surface Cap Substrate Node Type Check

See "extract/gettech.c". When reading the technology file, surface cap elements are checked for correct node type. Surface caps can not be of type -2 (@sub). See the technology compiler (tecc), @sub nodes are replaced by type -4 nodes. This means that there is a SUBCONTELEM added to the technology file. Note that function enumPair does not need to look for SURFCAPELEM, but this is not yet changed.

4.10 Conductor Mask Colors

See "extract/gettech.c". Each conductor mask must has a color. This is normally true, but additional masks (initiated by new mask technology statement) can also be used as conductor mask. These color id's are never found in the "gln" files of the standard masks. But the special prePass1 scans for conductors and these conductors must have a color. And each conductor (even the substrate conductor) must be in the filterBitmask, else the HasConduc macro does not returns TRUE. And the edges of each conductor can also be displayed by *Xspace*.

4.11 Conductor filterBitmask

See "extract/gettech.c". It is not a good idea to add mask colors of additional masks (which are not a conductor) to the filterBitmask. I have replaced the #ifndef DRIVER line into a #if 0 line.

4.12 Solved ASSERT Failure in nodeJoin

See "lump/lump.c". Added readyGroup test to function nodeJoin with the QAG_inuse flag. This is needed, because nodeJoin can be called while readyGroup is in progress. In that case the node group notReady flag is equal zero. Modified also function nodeJoin for processing ready distributed substrate nodes (substr ≡ 2).

4.13 Added ASSERT Test in placePoint

See "extract/enumpair.c". Replaced the infinity point coordinate test in a ASSERT test, because it is impossible that an infinity tile can have a node point. Because an infinity tile can not have a conductor.

4.14 Fixed subEdge1 Return Problem

See "scan/input.c". In function fetchEdge subEdge1 was not returned (but removed), when the edge left point is equal to the subEdge1 left point. I added also an ASSERT line, because subEdge1 must be always the first returned edge. The problem arises in the special prePass1.

4.15 Changed Macro equalAtX Again

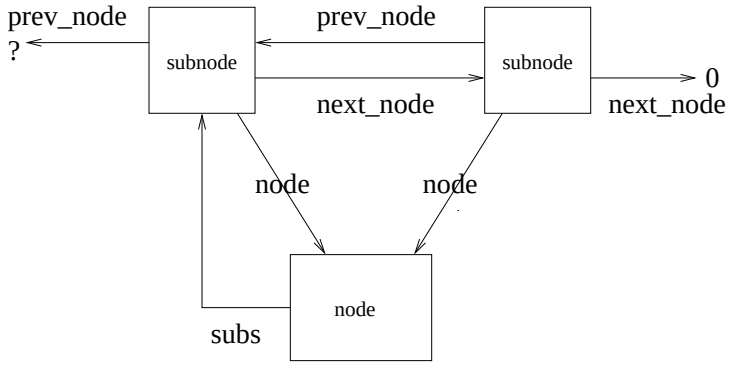
See "scan/scan.c". It was not a good idea to merge (bundle) edges, when the end point of edge e1 was equal to the begin point of edge e2. This scan behaviour is also important for the special prePass1.

4.16 Solved Incorrect cap3dStop Problem

See "scan/scan.c". At the end of the scan function cap3dStop is called. But before this is done, all strips need to be finished. Function cap3dStrip must be called till variable nextStrip is equal to INF.

4.17 Modified subnodeCopy

See "lump/lump.c". The "prev_node" member of the subnode was not set to NULL. Subnodes are not always initialized with subnodeNew. By mode "hasBipoElem" the subnode must have a "pn" member?



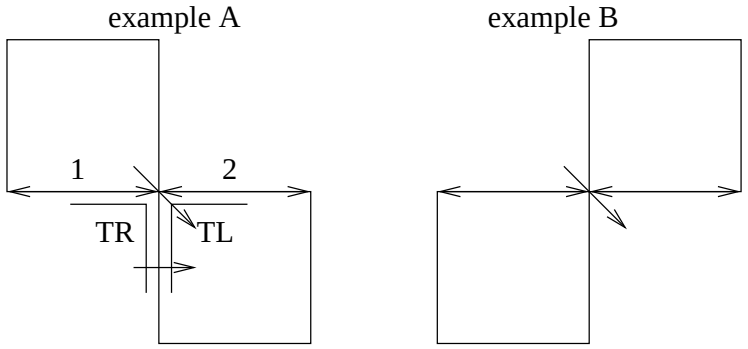
Note that the "prev_node" does not need to be initialized (see application note 03-07). Later on, the code has been changed again (see "lump.c" delta's 4.91 and 4.92).

4.18 Changed lastPass into extrPass

The lastPass flag is not more used in "out.c". Now, there is only output and back-annotation in the extrPass. See "lump/init.c" (delta 4.60), "lump/out.c" (delta 4.96) and "scan/hier.c" (delta 4.35).

4.19 Changed Substrate Terminal Join for Corners

Because of the scan function behaviour, the substrate area's of example A are joined and of example B not (see the figure below).



Because there is done twice an enumPair for the vertical edge (2nd time with blen ≡ 0), therefor the area's of example A are joined. The join is now not more done for blen ≡ 0, see "extract/enumpair.c" (delta 4.95).

5. APPENDICES

APPENDIX A -- List of Modified Source Files

The source files are first checked in at Fri 26 Sep 2003.
Revision 1 (Rev1) is checked in at 12/27 Jan 2004.
Files "input.c", "contedge.c"(4.10) and enumpair.c(4.100) checked in at 6 Feb 2004.

Module	File	New delta's	Rev1 delta's
extract	contedge.c	4.7	4.8-4.10
extract	enumpair.c	4.87-91	4.96,4.100
extract	enumtile.c	4.64-65	
extract	gettech.c	4.75-78	
lump	elem.c	4.41	
lump	extern.h	4.44	
lump	export.h	4.66	
lump	init.c	4.59,60	
lump	lump.c	4.84-87,91,92	4.95(16/1)
lump	node.c	4.71	
lump	out.c	4.93,96	
scan	edge.c	4.14	
scan	extern.h	4.21	
scan	hier.c	4.35	
scan	input.c	4.53,54	4.55(6/2)
scan	scan.c	4.49,51	4.54/4.55
scan	scan.h	4.13	
scan	sp_main.c	4.51,54-57	
scan	tile.c	4.27	
scan	update.c	4.27	
spider	cap3d.c	4.45	
spider	mesh.c	4.24	
spider	refine.c	4.21	
spider	spider.c	4.13	
spider	sptile.c	4.6	
spider	strip.c	4.30	
substr	subcont.cc	4.24	
X11	rgb.c	4.29	

APPENDIX B -- Differences of Source Files

```
Diff: extract/contedge.c (4.6 <-> 4.7)
=====================================================================
30c30
< DM_STREAM * stream_cont_aln;
> extern bool_t optSubResSave;
32c32,53
< void initContEdge (layoutKey) DM_CELL *layoutKey;
---
> typedef struct cEdge {
>     coor_t xl, yl, xr, yr;
>     int cc, di, top;
>     struct cEdge *next;
> } cEdge_t;
>
> static cEdge_t *eListBegin;
> static cEdge_t *eListEnd;
> static int nrOfEdges;
>
> static DM_STREAM *stream_cont;
> static DM_STREAM *stream_cont2;
>
> void initContEdge (lkey) DM_CELL *lkey;
> {
>     stream_cont = dmOpenStream (lkey, optSubResSave ? "cont_bln" : "cont_aln", "w");
>     if (optSubResSave) stream_cont2 = dmOpenStream (lkey, "cont_bpt", "w");
>     eListBegin = NULL;
>     nrOfEdges = 0;
> }
>
> Private int compareEdge (c1, c2) const void *c1, *c2;
34c55,61
<     stream_cont_aln = dmOpenStream (layoutKey, "cont_aln", "w");
---
>     cEdge_t ** e1 = (cEdge_t **) c1;
>     cEdge_t ** e2 = (cEdge_t **) c2;
>     if ((*e2) -> xl > (*e1) -> xl) return (-1);
>     if ((*e2) -> xl < (*e1) -> xl) return ( 1);
>     if ((*e2) -> yl > (*e1) -> yl) return (-1);
>     if ((*e2) -> yl < (*e1) -> yl) return ( 1);
>     return (0);
39c66,142
<     dmCloseStream (stream_cont_aln, COMPLETE);
---
>     cEdge_t *e, **eArray;
>     int i;
>
>     if (eListBegin) { /* optSubResSave */
>     eArray = NEW (cEdge_t *, nrOfEdges);
>     e = eListBegin;
>     for (i = 0; i < nrOfEdges; i++) {
>         eArray[i] = e;
>         e = e -> next;
>     }
>     qsort ((char *)eArray, nrOfEdges, sizeof (cEdge_t *), compareEdge);
>
>     for (i = 0; i < nrOfEdges; i++) {
>         e = eArray[i];
>         gboxlay.xl = e -> xl;
>         gboxlay.yb = e -> yl;
>         gboxlay.xr = e -> xr;
>         gboxlay.yt = e -> yr;
```

```
>         gboxlay.chk_type = e -> cc + (e -> di << 8);
>         dmPutDesignData (stream_cont, GEO_BOXLAY);
>         DISPOSE (e);
>     }
>     DISPOSE (eArray);
>     }
>     dmCloseStream (stream_cont, COMPLETE);
>     if (stream_cont2) {
>         dmCloseStream (stream_cont2, COMPLETE);
>         stream_cont2 = 0;
>     }
> }
>
> void contTileEdge (top, xl, yl, xr, yr)
> int top;
> coor_t xl, yl, xr, yr;
> {
>     coor_t dx1, dx2, dy1, dy2;
>     cEdge_t *e, *p = 0, *e1 = 0;
>
>     for (e = eListBegin; e; e = p -> next) {
>         if (e -> top == top && e -> xl == xl && e -> yl == yl) {
>             if (e -> xr == xr && e -> yr == yr) {
>                 if (e1) {
>                     e1 -> xr = xr;
>                     e1 -> yr = yr;
>                     if (eListEnd == e) eListEnd = p;
>                     p -> next = e -> next;
>                     DISPOSE (e); nrOfEdges--;
>                 }
>                 else e1 = e;
>                 return;
>             }
>             dx1 = e -> xr - e -> xl;
>             dx2 = xr - xl;
>             dy1 = e -> yr - e -> yl;
>             dy2 = yr - yl;
>             if (dx1 * dy2 == dx2 * dy1) {
>                 xl = e -> xr;
>                 yl = e -> yr;
>                 if (e1) {
>                     p -> next = e -> next;
>                     DISPOSE (e); nrOfEdges--;
>                 }
>                 else e1 = p = e;
>             }
>             else p = e;
>         }
>         else p = e;
>     }
>     ASSERT (0);
> }
>
> void contTile (tile) tile_t *tile;
> {
>     if (tile -> subcont -> distributed) {
>         contTileEdge (0, tile->xl, tile->bl, tile->xr, tile->br);
>         contTileEdge (1, tile->xl, tile->tl, tile->xr, tile->tr);
>     }
> }
46c149,232
<     /* non-vertical boundary */
---
>     if (optSubResSave) {
```

```
>         coor_t dx1, dx2, dy1, dy2;
>         cEdge_t *e;
>         int cc, di;
>
>         if (tile -> subcont) {
>             cc = tile -> subcont -> causing_con;
>             di = (newerTile -> subcont && newerTile -> subcont -> causing_con == cc);
>             if (cc < 0) cc = nrOfConductors - 1;
>
>             if (tile -> subcont -> distributed) di |= 2;
>
>             if (!di) {
>                 for (e = eListBegin; e; e = e -> next) {
>                     if (e -> top && e -> xr == bdr -> x1 && e -> yr == bdr -> y1) {
>                         dx1 = e -> xr - e -> xl;
>                         dx2 = bdr -> x2 - bdr -> x1;
>                         dy1 = e -> yr - e -> yl;
>                         dy2 = bdr -> y2 - bdr -> y1;
>                         if (dx1 * dy2 == dx2 * dy1) { /* same angle */
>                             if (e -> di) break;
>                             if (e -> cc != cc && !mergeNeighborSubContacts) break;
>                             e -> xr = bdr -> x2;
>                             e -> yr = bdr -> y2;
>                             return;
>                         }
>                     }
>                 }
>             }
>             e = NEW (cEdge_t, 1);
>             e -> xl = bdr -> x1;
>             e -> yl = bdr -> y1;
>             e -> xr = bdr -> x2;
>             e -> yr = bdr -> y2;
>             e -> cc = cc;
>             e -> di = di;
>             e -> top = 1;
>             e -> next = NULL;
>             if (eListBegin) eListEnd -> next = e;
>             else eListBegin = e;
>             eListEnd = e;
>             nrOfEdges++;
>         }
>
>         if (newerTile -> subcont) {
>             cc = newerTile -> subcont -> causing_con;
>             di = (tile -> subcont && tile -> subcont -> causing_con == cc);
>             if (cc < 0) cc = nrOfConductors - 1;
>
>             if (newerTile -> subcont -> distributed) di |= 2;
>
>             if (!di) {
>                 for (e = eListBegin; e; e = e -> next) {
>                     if (e -> top == 0 && e -> xr == bdr -> x1 && e -> yr == bdr -> y1) {
>                         dx1 = e -> xr - e -> xl;
>                         dx2 = bdr -> x2 - bdr -> x1;
>                         dy1 = e -> yr - e -> yl;
>                         dy2 = bdr -> y2 - bdr -> y1;
>                         if (dx1 * dy2 == dx2 * dy1) { /* same angle */
>                             if (e -> di) break;
>                             if (e -> cc != cc && !mergeNeighborSubContacts) break;
>                             e -> xr = bdr -> x2;
>                             e -> yr = bdr -> y2;
>                             return;
>                         }
```

```
>                     }
>                 }
>             }
>             e = NEW (cEdge_t, 1);
>             e -> xl = bdr -> x1;
>             e -> yl = bdr -> y1;
>             e -> xr = bdr -> x2;
>             e -> yr = bdr -> y2;
>             e -> cc = cc;
>             e -> di = di;
>             e -> top = 0;
>             e -> next = NULL;
>             if (eListBegin) eListEnd -> next = e;
>             else eListBegin = e;
>             eListEnd = e;
>             nrOfEdges++;
>         }
>         return;
>     }
60c246
<     dmPutDesignData (stream_cont_aln, GEO_GLN);
---
>     dmPutDesignData (stream_cont, GEO_GLN);
62a249,258
>
> void contPoint (x, y) coor_t x, y;
> {
>     if (stream_cont2) {
>         gboxlay.xl = x;
>         gboxlay.yb = y;
>         gboxlay.xr = 0;
>         gboxlay.yt = 0;
>         gboxlay.chk_type = 0;
>         dmPutDesignData (stream_cont2, GEO_BOXLAY);
>     }
> }


Diff: extract/contedge.c (4.7 <-> 4.8), substrate prepass improvements
=====================================================================
84a85
>           if (e -> top) gboxlay.chk_type += 0x400;
108,112c109,123
<                   e1 -> xr = xr;
<                   e1 -> yr = yr;
<                   if (eListEnd == e) eListEnd = p;
<                   p -> next = e -> next;
<                   DISPOSE (e); nrOfEdges--;
---
>                   if (e1 -> di == e -> di) {
>                       e1 -> xr = xr;
>                       e1 -> yr = yr;
>                       if (eListEnd == e) eListEnd = p;
>                       p -> next = e -> next;
>                       DISPOSE (e); nrOfEdges--;
>                   }
>                   else if (e1 -> di < e -> di) {
>                       e -> xl = e1 -> xl;
>                       e -> yl = e1 -> yl;
>                   }
>                   else {
>                       e1 -> xr = xr;
>                       e1 -> yr = yr;
>                   }
```

```
114d124
<               else e1 = e;
125,126c135,147
<                   p -> next = e -> next;
<                   DISPOSE (e); nrOfEdges--;
---
>                   if (e1 -> di == e -> di) {
>                       p -> next = e -> next;
>                       DISPOSE (e); nrOfEdges--;
>                   }
>                   else if (e1 -> di < e -> di) {
>                       e -> xl = e1 -> xl;
>                       e -> yl = e1 -> yl;
>                       e1 = e;
>                   }
>                   else {
>                       e1 -> xr = xl;
>                       e1 -> yr = yl;
>                   }
150c171
<       coor_t dx1, dx2, dy1, dy2;
---
>       coor_t dx1, dx2, dy1, dy2, xl, yl;
161c182,185
<           if (!di) {
---
>           xl = bdr -> x1;
>           yl = bdr -> y1;
>
>           if (di < 2) {
169,173c193,203
<                           if (e -> di) break;
<                           if (e -> cc != cc && !mergeNeighborSubContacts) break;
<                           e -> xr = bdr -> x2;
<                           e -> yr = bdr -> y2;
<                           return;
---
>                           if (e -> di > 1 || e -> cc != cc) break;
>                           if (e -> di >= di) {
>                               e -> xr = bdr -> x2;
>                               e -> yr = bdr -> y2;
>                               if (e -> di == di) goto ret;
>                           }
>                           else {
>                               xl = e -> xl;
>                               yl = e -> yl;
>                           }
>                           break;
179,180c209,210
<           e -> xl = bdr -> x1;
<           e -> yl = bdr -> y1;
---
>           e -> xl = xl;
>           e -> yl = yl;
192c222
<
---
> ret:
200c230,233
<           if (!di) {
---
>           xl = bdr -> x1;
>           yl = bdr -> y1;
>
```

```
>           if (di < 2) {
208,212c241,251
<                           if (e -> di) break;
<                           if (e -> cc != cc && !mergeNeighborSubContacts) break;
<                           e -> xr = bdr -> x2;
<                           e -> yr = bdr -> y2;
<                           return;
---
>                           if (e -> di > 1 || e -> cc != cc) break;
>                           if (e -> di >= di) {
>                               e -> xr = bdr -> x2;
>                               e -> yr = bdr -> y2;
>                               if (e -> di == di) return;
>                           }
>                           else {
>                               xl = e -> xl;
>                               yl = e -> yl;
>                           }
>                           break;
218,219c257,258
<           e -> xl = bdr -> x1;
<           e -> yl = bdr -> y1;
---
>           e -> xl = xl;
>           e -> yl = yl;

Diff: extract/contedge.c (4.8 <-> 4.9), added soft-begin flag for special pp1
=====================================================================
# 122a123
# >                     if (!top) e -> di += 8; // soft begin
# 245a247
# >                               di += 8; // soft begin

Diff: extract/contedge.c (4.9 <-> 4.10), removed function contPoint
=====================================================================
43d42
< static DM_STREAM *stream_cont2;
48d46
<     if (optSubResSave) stream_cont2 = dmOpenStream (lkey, "cont_bpt", "w");
92,95d89
<     if (stream_cont2) {
<       dmCloseStream (stream_cont2, COMPLETE);
<       stream_cont2 = 0;
<     }
290,301d283
< void contPoint (x, y) coor_t x, y;
< {
<     if (stream_cont2) {
<       gboxlay.xl = x;
<       gboxlay.yb = y;
<       gboxlay.xr = 0;
<       gboxlay.yt = 0;
<       gboxlay.chk_type = 0;
<       dmPutDesignData (stream_cont2, GEO_BOXLAY);
<     }
< }

Diff: extract/enumpair.c (4.86 <-> 4.88)
=====================================================================
16a17,18
> #define NEW_B1 /* using new prePass1 for -B */
>
47a50,51
> extern bool_t optSubResSave;
```

```
> extern bool_t optOnlyPrePassB1;
58a63
> extern  void contPoint P_((coor_t x, coor_t y));
59a65
> Private void pp1EnumPair P_((tile_t *tile, tile_t *newerTile, int edgeOrien));
182a189,199
> #ifndef PUBLIC
> #ifndef NO_SUB_RES
> #ifdef CAP3D
>     if (optOnlyPrePassB1) {
>         pp1EnumPair (tile, newerTile, edgeOrien);
>         return;
>     }
> #endif
> #endif
> #endif
321a339
> #ifndef NEW_B1
333a352
> #endif /* !NEW_B1 */
345c364,370
<             if (optSubRes && (cx = nsc -> causing_con) >= 0) {
---
> #ifdef NEW_B1
>             if (optSubResSave && (cx = nsc -> causing_con) >= 0)
> #else
>             if (optSubRes && (cx = nsc -> causing_con) >= 0)
> #endif
>             {
>                 ASSERT (newerTile -> cons[cx]);
396a422,433
>         if (optSubResSave && edgeOrien != 'v') {
>             if (tsc -> distributed || nsc -> distributed
>              || tsc -> causing_con != nsc -> causing_con && !mergeNeighborSubContacts) {
>                 contEdge (tile, newerTile, bdr);
>             }
>         }
>     }
>     else if (optSimpleSubRes && (tsc || nsc)) {
>         if (tsc) tsc -> subcontInfo -> perimeter += blen;
>         else     nsc -> subcontInfo -> perimeter += blen;
>         if (edgeOrien != 'v') contEdge (tile, newerTile, bdr);
422a460
> #ifndef PUBLIC
> #ifndef NO_SUB_RES
>         if (optSubResSave && tsc && nsc && optIntRes) contPoint (term -> x, term -> y);
> #endif
> #endif
428,437d465
< #ifndef PUBLIC
< #ifndef NO_SUB_RES
<     if (optSimpleSubRes && (tsc && !nsc || !tsc && nsc)) {
<         if (tsc) tsc -> subcontInfo -> perimeter += blen;
<         else     nsc -> subcontInfo -> perimeter += blen;
<         if (bdr -> x1 != bdr -> x2) contEdge (tile, newerTile, bdr);
<     }
< #endif
< #endif /* PUBLIC */
496c524
<     if (optCap3D) spiderPair (tile, newerTile, edgeOrien, bdr);
>     if (optCap3D && !optSubResSave) spiderPair (tile, newerTile, edgeOrien, bdr);
499a528,574
> #ifndef PUBLIC
> #ifndef NO_SUB_RES
```

```
> #ifdef CAP3D
> Private void pp1EnumPair (tile, newerTile, edgeOrien)
> tile_t *tile, *newerTile;
> int edgeOrien;
> {
>     subcontRef_t *tsc = tile -> subcont;
>     subcontRef_t *nsc = newerTile -> subcont;
>
>     if (newtile && nHasConduc) {
>         subnode_t *sn;
>         newerTile -> subcont = nsc = subContNew (newerTile);
>         nsc -> subn = sn = NEW (subnode_t, 1);
>         if (newerTile -> known & 0x200) nsc -> distributed = 1;
>         nsc -> causing_con = newerTile -> known & 0xff;
>
>         subnodeNew (sn);
>         sn -> node -> mask = -1;
>         sn -> node -> node_x = newerTile -> xl;
>         sn -> node -> node_y = newerTile -> bl;
>         sn -> node -> substr = 1;
>         sn -> node -> term = 1;
>     }
>
>     if (tsc && nsc) {
>         if (!tsc -> distributed && !nsc -> distributed
>             && (mergeNeighborSubContacts || tsc -> causing_con == nsc -> causing_con))
>             subContJoin (tile, newerTile);
>         else
>             subContGroupJoin (tsc -> subcontInfo, nsc -> subcontInfo);
>     }
>
>     if (blen <= 0.0) return;
>
>     spiderPair (tile, newerTile, edgeOrien, bdr);
> }
> #endif
> #endif
> #endif

Diff: extract/enumpair.c (4.88 <-> 4.91)
===================================================================
17,18d16
< #define NEW_B1 /* using new prePass1 for -B */
<
339,340c341
< #ifndef NEW_B1
<                                 if (optSubRes) {
---
>                                 if (optSubRes) { // optSubResOldMesh
352d352
< #endif /* !NEW_B1 */
364,369c364
< #ifdef NEW_B1
<                 if (optSubResSave && (cx = nsc -> causing_con) >= 0)
< #else
<                 if (optSubRes && (cx = nsc -> causing_con) >= 0)
< #endif
<                 {
---
>                 if ((optSubResSave || optSubRes) && (cx = nsc -> causing_con) >= 0) {
406,407c400,401
<
<     i = lastPass;
---
```

```
> if (tsc || nsc) {
>     i = 1;
425c417
<                   || tsc -> causing_con != nsc -> causing_con && !mergeNeighborSubContacts) {
---
>                   || tsc -> causing_con != nsc -> causing_con) {
444a435
> }

Diff: extract/enumpair.c (4.95 <-> 4.96), substrate prepass improvements
*****************************************************************
368a369,370
>                   if (mergeNeighborSubContacts &&
>                       !nsc -> distributed) nsc -> causing_con = -1;
408,409c410
<               && (mergeNeighborSubContacts
<                   || tsc -> causing_con == nsc -> causing_con))) {
---
>               && tsc -> causing_con == nsc -> causing_con)) {
559c560
<               && (mergeNeighborSubContacts || tsc -> causing_con == nsc -> causing_con))
---
>               && tsc -> causing_con == nsc -> causing_con)

Diff: extract/enumpair.c (4.99 <-> 4.100), removed function contPoint
*****************************************************************
61d60
< extern  void contPoint P_((coor_t x, coor_t y));
440,444d437
< #ifndef PUBLIC
< #ifndef NO_SUB_RES
<           if (optSubResSave && tsc && nsc && optIntRes) contPoint (term -> x, term -> y);
< #endif
< #endif

Diff: extract/enumtile.c (4.63 <-> 4.65)
*****************************************************************
55a56,57
> extern bool_t optOnlyPrePassB1;
> extern bool_t optSubResSave;
72a75
> extern  void contTile P_((tile_t *tile));
149c152,155
<     if (optCap3D) spiderTile (tile);
---
>     if (optCap3D && !optSubResSave) {
>         spiderTile (tile);
>         if (optOnlyPrePassB1) return;
>     }
170c176
<     if (optSimpleSubRes && tile -> subcont) /* prePass1 */
---
>     if (optSimpleSubRes && tile -> subcont) { /* prePass1 */
171a178,179
>         if (optSubResSave) contTile (tile);
>     }
1434c1442
<         sn2 -> node -> substr = 1;
---
>         sn2 -> node -> substr = 2;
1442a1445,1446
>         if (cted -> val > 0.1e-12 && (n = subnodeFindSubcNode (sn1, p -> x, p -> y)))
>             nodeJoin (n, sn2 -> node);
```

```
Diff: extract/gettech.c (4.74 <-> 4.75)
=====================================================================
423a424,426
>         if (!IS_COLOR (masktable[i].color)) {
>             COLOR_ADDINDEX (masktable[i].color, procdata -> nomasks);
>         }
426c429
< #ifndef DRIVER
---
> #if 0
812c815
<                 else if (k == -2 || k == -4)
>                 else if (k == -2 && el -> type != SURFCAPELEM || k == -4)
826c829
<                 else if (k == -2 || k == -4)
>                 else if (k == -2 && el -> type != SURFCAPELEM || k == -4)

Diff: extract/gettech.c (4.75 <-> 4.78)
=====================================================================
424,426d423
<           if (!IS_COLOR (masktable[i].color)) {
<               COLOR_ADDINDEX (masktable[i].color, procdata -> nomasks);
<           }
1521a1519,1548
> static mask_t filterBitmaskSave;
> static mask_t * colorsSave;
>
> void pp1SetColors ()
> {
>     ...
> }
>
> void pp1ResetColors ()
> {
>     ...
> }


Diff: lump/elem.c (4.40 <-> 4.41)
=====================================================================
34,36d33
< static int Sort[100];
38,41d34
< /*
< #define ELEMHASH(n1,n2,sort,size)  ((n2 -> id ^ n1 -> id ^ Sort[sort]) % size)
< #define ELEMHASH(n1,n2,size)  (Abs (n2 -> id ^ n1 -> id) % size)
< */
442,450d432
<     static int init = 0;
<
<     if (!init) { int i;
<         init = 1;
<         for (i = 0; i < 100; i++) Sort[i] = (int) lrand48 ();
<     }
462c444
<     if (capHT_size == 0) enlargeElemHT (&capHashTab, &capHT_size);
>     if (capHT_size == 0) return (el);
469c451
<     if (resHT_size == 0) enlargeElemHT (&resHashTab, &resHT_size);
>     if (resHT_size == 0) return (el);
480c462
<     return ((element_t *) NULL);
>     return (el);
```

```
Diff: lump/init.c (4.58 <-> 4.59)
=================================================================
84a85
> int areaNodesTotal;
251a253
>     areaNodesTotal = 0;
772c774
<     fprintf (fp_info, "sub term nods : %6d %6d\n", inSubTerm, outSubTerm);
>     fprintf (fp_info, "sub term nodes: %6d %6d\n", inSubTerm, outSubTerm);
778,779c780,782
<     fprintf (fp_info, "area nodes               : %d\n", areaNodes);
<     fprintf (fp_info, "equi lines               : %d\n", equiLines);
---
>     fprintf (fp_info, "equi area nodes          : %d\n", areaNodes);
>     fprintf (fp_info, "equi line nodes          : %d\n", equiLines);
>     fprintf (fp_info, "total ready area nodes   : %d\n", areaNodesTotal);

Diff: lump/init.c (4.59 <-> 4.60)
=================================================================
564c564
<     if (lastPass && paramLookupB ("histogram", "off")) {
>     if (extrPass && paramLookupB ("histogram", "off")) {
688c690
<     if (lastPass && optHisto) {
>     if (optHisto) {

Diff: lump/extern.h (4.43 <-> 4.44)
=================================================================
118a119
> extern int areaNodesTotal;
235,237d235
< long int lrand48 P_((void));

Diff: lump/export.h (4.65 <-> 4.66)
=================================================================
61a62
> node_t * subnodeFindSubcNode P_((subnode_t * sn, coor_t x, coor_t y));

Diff: lump/lump.c (4.83 <-> 4.87)
=================================================================
34a35
> extern int QAG_inuse;
286c287,293
<     if (nB -> substr) nA -> substr = 1;
---
>     if (nB -> substr) {
>         if (nA -> substr != 2) nA -> substr = nB -> substr;
>     }
336,337c343,347
<     if (nA -> subs) nA -> subs -> prev_node = last;
<     nA -> subs = nB -> subs;
---
>     if (nA -> subs) {
>         nA -> subs -> prev_node = last;
>         nA -> subs = nB -> subs;
>     }
>     else if (nA -> ports) nA -> subs = nB -> subs;
434,435c444,447
<     ASSERT (grA -> notReady >= 2);
<     grA -> notReady--;
---
>     if (!QAG_inuse) { /* if not called via readyGroup */
>         ASSERT (grA -> notReady >= 2);
>         if (nA -> subs || nA -> ports) grA -> notReady--;
```

```
>     }
530a543
>     subnB -> prev_node = NULL;
541,542c554
<     if (hasBipoElem) polnodeCopy (subnA, subnB);
>     if (hasBipoElem && subnA -> pn) polnodeCopy (subnA, subnB);
769a782
>     areaNodes++;
864c877
<     if (n -> area) areaNodes++;
>     if (n -> area) areaNodesTotal++;
899a913,918
>     /* Don't delay distributed substrate nodes,
>      * else they cannot be joinded later on!
>      * See function subnodeSubcontReconnect().
>      */
>     if (n -> substr == 2) return;
1331a1332,1353
> node_t * subnodeFindSubcNode (sn, x, y) subnode_t *sn; coor_t x, y;
> {
>     int i;
>     element_t *el;
>     node_t *nS, *n;
>
>     nS = sn -> node;
>     for (i = 0; i < resSortTabSize; i++) {
>         for (el = nS -> con[i]; el; el = NEXT (el, nS)) {
>             n = OTHER (el, nS);
>             /*
>              * Try to find another distributed substrate node at same
>              * x,y-coordinate (el must be a contact resistance).
>              */
>             if (n -> substr == 2 && n -> node_x == x && n -> node_y == y) return n;
>         }
>     }
>     return 0;
> }
1366,1371c1365,1368
<     if (nS -> con) {
1392,1397c1389,1393
<     if (node -> con) {

Diff: lump/node.c (4.70 <-> 4.71)
=====================================================================
72a72,76
> Private long int lrand48 P_((void));
> Private void     srand48 P_((long));
> #else
> extern  long int lrand48 P_((void));
> extern  void     srand48 P_((long));
156a153
>     srand48 (1L); /* init random node id's */
475,477c472,473
<     /* lran48 returns a long but, if the sequence is truly ...
---
>     /* lrand48 returns a long but, if the sequence is truly random,
>      * it is safe to truncate if an int is of lower precision.
869c863,870
< long int lrand48 ()
---
> static long a_r48;
>
> Private void srand48 (a) long a;
> {
```

```
>     a_r48 = a * 1234567;
> }
>
> Private long int lrand48 ()
871d871
<     static long a = 1234567;
875,876c875,876
<     a = (large_mul (a, b) + 1) % MM;
<     return (a);                         /* 0 <= a < MM */
---
>     a_r48 = (large_mul (a_r48, b) + 1) % MM;
>     return (a_r48); /* 0 <= a_r48 < MM */
878,879c878
<     else
<        return ((long)rand ());
---
>     return ((long)rand ());

Diff: lump/out.c (4.92 <-> 4.93)
=================================================================
41a42,43
> extern bool_t optOnlyPrePassB1;
203,208c205,210
<     if (!dmsCon)     dmUNLINK ("congeo");
<     if (!dmsTermpos) dmUNLINK ("termpos");
<     if (!dmsDevgeo)  dmUNLINK ("devgeo");
---
>     if (!optOnlyPrePassB1) {
>         if (!dmsCon)     dmUNLINK ("congeo");
>         if (!dmsTermpos) dmUNLINK ("termpos");
>         if (!dmsDevgeo)  dmUNLINK ("devgeo");
>     }

Diff: lump/out.c (4.95 <-> 4.96)
=================================================================
179c179
<     if (lastPass) {
>     if (extrPass) {
199c199
<     if (prePass2 || optBackInfo /* lastPass */) {
>     if (prePass2 || optBackInfo /* extrPass */) {
  ...

Diff: scan/edge.c (4.13 <-> 4.14)
=================================================================
56a57
>     edge -> cc = 0;

Diff: scan/extern.h (4.20 <-> 4.21)
=================================================================
65c65
<     mask_t color, tile_t *stb, tile_t *stl));
>     mask_t color, tile_t *stb, tile_t *stl, int cc));

Diff: scan/hier.c (4.34 <-> 4.35)
=================================================================
122c122
<     if (lastPass) {
>     if (extrPass) {

Diff: scan/input.c (4.52 <-> 4.53)
=================================================================
44a45
> extern bool_t optOnlyPrePassB1;
```

```
> extern int * conductorMask;
> static DM_STREAM *ptStream;
118a123,142
>     edges = NULL;
>
>     if (optOnlyPrePassB1) {
>         TERM_index = -1; /* re-init needed for Xspace */
>         i = 0; /* cont_bln mask must be a conductor mask */
>
>         /* by optIntRes: terminal point can split a tile */
>         ptStream = dmOpenStream (cellKey, "cont_bpt", "r");
>
>         glnStream = dmOpenStream (cellKey, "cont_bln", "r");
>         if (glnStream && (edge = doFetch (glnStream, i))) {
>             edges = NEW (struct e, 1);
>             edges -> mask_no = i;
>             edges -> stream = glnStream;
>             edges -> edge = edge;
>             edges -> next = NULL;
>         }
>         return;
>     }
122,123d145
<     edges = NULL;
188c210
<                 if (optIntRes == TRUE || optCap3D == FALSE) {
>                 if (optIntRes) { /* center coord. (tile split) */
192,199c214
<                 else {
<                    /* By putting terminals on left edge of their
<                     * box, we can hide terminals for the BE-mesh generation.
<                       ...  */
---
>                 else { /* terminal on left edge (gives nicer BE-mesh) */
320c335
<             || (subEdge1 -> xl == edge -> xl && subEdge1 -> yl < edge -> yl)) {
>             || (subEdge1 -> xl == edge -> xl && subEdge1 -> yl <= edge -> yl)) {
325,327c340
<         else if (subEdge1 -> xl == edge -> xl && subEdge1 -> yl == edge -> yl)
<             subEdge1 = NULL;
---
>         else ASSERT (0);
379a392,417
>     if (optOnlyPrePassB1) {
>         k = dmGetDesignData (stream, GEO_BOXLAY);
>         if (k > 0) {
>             edge_t *edge;
>             mask_t color;
>
>             if (gboxlay.chk_type & 0x100) /* interior edge */
>                 color = cNull;
>             else {
>                 k = gboxlay.chk_type & 0xff; /* conductor_nr */
>                 ASSERT (k >= 0 && k < nrOfConductors);
>                 color = masktable[conductorMask[k]].color;
>                 if (!IS_COLOR (color))
>                     fprintf (stderr, "doFetch: no color for conductor_nr %d\n", k);
>             }
>             edge = createEdge ((coor_t) gboxlay.xl, (coor_t) gboxlay.yb,
>                 (coor_t) gboxlay.xr, (coor_t) gboxlay.yt, color);
>             edge -> cc = gboxlay.chk_type;
>             return edge;
>         }
>         if (k < 0) say ("cont_bln read error"), die ();
```

```
>         dmCloseStream (stream, COMPLETE);
>         return NULL;
>     }
475,477c512,529
<     tm = NEW (terminal_t, 1);
<     tm -> x = tm -> y = INF;
---
>     if (optOnlyPrePassB1 && ptStream) {
>         int k = dmGetDesignData (ptStream, GEO_BOXLAY);
>         if (!tm) tm = NEW (terminal_t, 1);
>         if (k > 0) {
>             tm -> x = gboxlay.xl;
>             tm -> y = gboxlay.yb;
>         }
>         else {
>             if (k < 0) say ("cont_bpt read error"), die ();
>             dmCloseStream (ptStream, COMPLETE); ptStream = 0;
>             tm -> x = tm -> y = INF;
>         }
>     }
>     else {
>         ASSERT (TERM_index == nrOfTerminals);
>         tm = NEW (terminal_t, 1);
>         tm -> x = tm -> y = INF;
>     }

Diff: scan/input.c (4.53 <-> 4.54)
======================================================================
126a127
>          nrOfTerminals = 0;      // bug fix!
506c507,508
<     static terminal_t * tm;
---
>     static terminal_t TM;
>     terminal_t * tm;
511a514
>          tm = &TM;
514d516
<              if (!tm) tm = NEW (terminal_t, 1);
527d528
<              tm = NEW (terminal_t, 1);

Diff: scan/input.c (4.54 <-> 4.55), removed cont_bpt stream
======================================================================
104,105d103
< static DM_STREAM *ptStream;
<
130,132d127
<       /* by optIntRes: terminal point can split a tile */
<       ptStream = dmOpenStream (cellKey, "cont_bpt", "r");
<
515,530c510,511
<       if (optOnlyPrePassB1 && ptStream) {
<           int k = dmGetDesignData (ptStream, GEO_BOXLAY);
<           if (k > 0) {
<               tm -> x = gboxlay.xl;
<               tm -> y = gboxlay.yb;
<           }
<           else {
<               if (k < 0) say ("cont_bpt read error"), die ();
<               dmCloseStream (ptStream, COMPLETE); ptStream = 0;
<               tm -> x = tm -> y = INF;
<           }
<       }
```

```
<       else {
<           ASSERT (TERM_index == nrOfTerminals);
<           tm -> x = tm -> y = INF;
<       }
---
>       ASSERT (TERM_index == nrOfTerminals);
>       tm -> x = tm -> y = INF;

Diff: scan/scan.c (4.48 <-> 4.49)
=================================================================
99c98
<     compareSlope (e1, ==, e2) && e1 -> xr >= thisX)
>     compareSlope (e1, ==, e2) && e1 -> xr > thisX)
422c421,424
<     if (optCap3D) cap3dStop ();
---
>     if (optCap3D) {
>         while (nextStrip != INF) nextStrip = cap3dStrip ();
>         cap3dStop ();
>     }
446,447c448,449
<     head -> tile = createTile (-INF, -INF, INF, -INF, cNull, NULL, NULL);
<     tail -> tile = createTile (-INF,  INF, INF,  INF, cNull, NULL, NULL);
---
>     head -> tile = createTile (-INF, -INF, INF, -INF, cNull, NULL, NULL, 0);
>     tail -> tile = createTile (-INF,  INF, INF,  INF, cNull, NULL, NULL, 0);

Diff: scan/scan.c (4.50 <-> 4.51), bug fix: bundle edge
=================================================================
504a505
>       e -> cc = newEdge -> cc;
514a516
>       e -> cc = newEdge -> cc;

Diff: scan/scan.c (4.53 <-> 4.54), substrate prepass improvements
=================================================================
34a35,36
> extern bool_t optOnlyPrePassB1;
>
281a284,285
>           termSplit = 0;
>
286a291,292
>               if (optOnlyPrePassB1 && !(newEdge -> cc & 0x400))
>                   if (!(edge -> cc & 0x100)) termSplit = 1;
302d307
<           termSplit = 0;
475c480
<       e -> cc = newEdge -> cc;
>       if (!(newEdge -> cc & 0x400)) e -> cc = newEdge -> cc;
486c491
<       e -> cc = newEdge -> cc;
>       if (!(newEdge -> cc & 0x400)) e -> cc = newEdge -> cc;
509a515
>       if (!(newEdge -> cc & 0x400)) e -> cc = newEdge -> cc;

Diff: scan/scan.c (4.54 <-> 4.55), added soft-begin flag for special pp1
=================================================================
291,292c291
<               if (optOnlyPrePassB1 && !(newEdge -> cc & 0x400))
<                   if (!(edge -> cc & 0x100)) termSplit = 1;
---
>               if (optOnlyPrePassB1 && !(newEdge -> cc & 0xc00)) termSplit = 1;
```

```
Diff: scan/scan.h (4.12 <-> 4.13)
=================================================================
28a29
>     int cc;  /* causing conductor for substrate */

Diff: scan/sp_main.c (4.50 <-> 4.51)
=================================================================
72a73,74
> #define NEW_B1 /* using new prePass1 for -B */
81a84
>        optSubResSave = FALSE,
>        optOnlyLastPass = FALSE,
>        optOnlyPrePassB1 = FALSE,
135a141,144
> #ifdef NEW_B1
> int    runSpecialPrePass = 1;
> #endif
145a155
> bool_t skipPrePass0 = 0;
365c376
<     strcat (optstring, "%L:dfgJOHNMqmKZw");
>     strcat (optstring, "%L:dfgJOHNMqmKZ012w");
447a459,464
>         case '0': skipPrePass0 = TRUE; break;
>         case '2': optOnlyLastPass = TRUE;
>                   optNoPrepro = TRUE; break;
>         case '1': optOnlyPrePassB1 = TRUE;
>                   optSubRes = TRUE;
>                   optNoPrepro = TRUE;
>         case 'Z': optOnlyPrePass = TRUE; break;
474a492,504
>     if (optOnlyLastPass && optOnlyPrePass) {
>         say ("Option -Z ignored, option -%2 does only the last pass.");
>         optOnlyPrePass = FALSE;
>     }
>
> #ifdef DISPLAY
> #ifdef NEW_B1
>     if (optDisplay && !optNoMenus && optOnlyPrePass && !optOnlyPrePassB1) {
>         runSpecialPrePass = 0;
>     }
> #endif
> #endif
717a748,750
> #ifdef NEW_B1
>     if (optCap3D || optOnlyPrePassB1) { /*}*/
> #else
718a752
> #endif
757,759c791,792
<     alsoPrePass = optResMesh || optPrick;
<     if (alsoPrePass || optAllRes) optIntRes = TRUE;
---
>     if (optOnlyPrePassB1) optResMesh = FALSE;
>     if (optResMesh || optPrick || optAllRes) optIntRes = TRUE;
761,762c794,797
<     alsoPrePassSave = alsoPrePass;
<     if (substrRes) alsoPrePass = TRUE;
---
>     if (!optOnlyLastPass) {
>         alsoPrePass = alsoPrePassSave = optResMesh || optPrick;
>         if (substrRes) alsoPrePass = TRUE;
>     }
834c869
```

```
<             if (optResMesh) {
>             if (optResMesh && !skipPrePass0) {
850c885
<                 lastPass = optOnlyPrePass && !optPrick;
>                 lastPass = optOnlyPrePass && !optPrick && !optOnlyPrePassB1;
855c890
<                 if (optSimpleSubRes) processgln (list + i);
>                 if (!optOnlyPrePassB1) processgln (list + i);
922c957,966
<     if (prePass1 && optSubRes) optCap3D = TRUE;
---
>     if (prePass1 && optSubRes) {
>         optCap3D = TRUE;
> #ifdef NEW_B1
>         if (!optOnlyPrePassB1) {
>             optSubResSave = 1;
>             optSubRes = FALSE;
>             optSimpleSubRes = TRUE;
>         }
> #endif
>     }
932a977,981
>     if (optSubResSave) {
>         optSubResSave = 0;
>         optSubRes = TRUE;
>         optSimpleSubRes = FALSE;
>     }
977c1026
<     if (firstPass) {
>     if (firstPass && !optOnlyPrePassB1) {
1004c1053
<     ASSERT (circuitKey);
>     ASSERT (circuitKey || optOnlyPrePassB1);
1059c1108
<     if (optCap3D) {
>     if (optCap3D || optOnlyLastPass && optSubRes) {
1061c1110
<         if (prePass1) {
>         if (prePass1 || !optCap3D) {
1115c1164,1165
<     initHierNames ();
>     if (!optOnlyPrePassB1) initHierNames ();
1119c1169,1170
<     readTid (layoutKey, circuitKey, inScale);
>     if (!optOnlyPrePassB1) readTid (layoutKey, circuitKey, inScale);
1133a1185,1186
>     else if (optOnlyLastPass && optSubRes)
>         (void) cap3dInit (inScale, bigbxl, bigbxr, bigbyb, bigbyt, layoutKey);
1171c1222,1223
<     endHierNames ();
>     if (!optOnlyPrePassB1) endHierNames ();
1177c1229,1230
<     disposeTid ();
>     if (!optOnlyPrePassB1) disposeTid ();
1183a1237
>     else if (optOnlyPrePassB1) dmCheckIn (layoutKey,  COMPLETE);
1311c1366
<     else { /* prePass1: optSimpleSubRes */
>     else if (optSimpleSubRes) { /* prePass1 */
1318a1374,1409
> #ifdef NEW_B1
>     else if (runSpecialPrePass) { /* prePass1: optSubRes */
>         char optbuf[1024];
>         char *av[64], *opt;
```

```
>         int c, i = 0;
>
>         av[i++] = "space3d"; av[i++] = "-%1";
>         opt = optbuf;
>         *opt = 0;
>         if (techFile) sprintf (opt, "-E%s", techFile);
>         else if (techDef) sprintf (opt, "-e%s", techDef);
>         if (*opt) { av[i++] = opt; opt += strlen (opt) + 1; }
>
>         *opt = 0;
>         if (paramFile) sprintf (opt, "-P%s", paramFile);
>         else if (paramDef) sprintf (opt, "-p%s", paramDef);
>         if (*opt) { av[i++] = opt; opt += strlen (opt) + 1; }
>         ASSERT (opt - optbuf < 1024);
>
>         c = paramGetOptionCount ();
>         while (--c >= 0) {
>             sprintf (opt, "-S%s=%s", paramGetOptionKey (c), paramGetOptionValue (c));
>             av[i++] = opt; opt += strlen (opt) + 1;
>             ASSERT (opt - optbuf < 1024);
>         }
>         av[i++] = cellname;
>         ASSERT (i < 64);
>         av[i] = 0;
>
>         verbose ("computing substrate effects for %s\n", cellname);
>         run2 (av[0], av);
>     }
> #endif
1391a1481,1502
> Private void run2 (char *command, char *argv[])
> {
>     char path[1000];
>     char *cellname, *option;
>     int i = 0;
>
>     if (prep_bin == NULL) {
>         sprintf (path, "%s/%s/bin", icdpath, ICDARCH);
>         prep_bin = strsave (paramLookupS ("prep_bin", path));
>     }
>     sprintf (path, "%s/%s", prep_bin, command);
>
>     option = argv[i];
>     while (argv[i + 1]) ++i;
>     cellname = argv[i];
>     Debug (fprintf (stderr, "%s %s ... %s\n", path, option, cellname));
>     if (_dmRun2 (path, argv)) {
>         say ("error in '%s %s ... %s'", command, option, cellname);
>         die ();
>     }
> }

Diff: scan/sp_main.c (4.53 <-> 4.54)
=====================================================================
131a132
> int    makesubres = 0;
267c268
<         if(s != 0) s++; else s = argv[0];
>         if (s) s++; else s = argv0;
268a270,275
>         else if (strsame (s, "makesubres")) { makesubres = 1;
>           optOnlyPrePassB1 = optSubRes = TRUE;
>           optNoPrepro = optOnlyPrePass = TRUE;
>           strcpy (optstring, "E:e:P:p:S:");
```

```
>           goto getopt;
>       }
352a360
> getopt:
499a508,511
>       if (makesubres) {
>           fprintf (stderr, "[-e def | -E file] [-p def | -P file] [-S param=value] cell\n");
>           die ();
>       }
504c516
<       fprintf (stderr, "[-cFTI%suntvhix] [-a time]", OPT_X);
>       fprintf (stderr, "[-cFTI%suntvhix] [-a time] ", OPT_X);
1348,1349c1359
<       av[i++] = "space3d";
<       av[i++] = "-%1";
---
>       av[i++] = "makesubres";

Diff: scan/sp_main.c (4.54 <-> 4.57)
=====================================================================
64,65d63
< #define NEW_B1 /* using new prePass1 for -B */
74a73,74
>        optSubResInternal = FALSE,
>        optSubResOldMesh = FALSE,
133d132
< #ifdef NEW_B1
135d133
< #endif
178a177,179
> extern void pp1SetColors   P_((void));
> extern void pp1ResetColors P_((void));
473,480d470
< #ifdef DISPLAY
< #ifdef NEW_B1
<     if (optDisplay && !optNoMenus && optOnlyPrePass && !optOnlyPrePassB1) {
<         runSpecialPrePass = 0;
<     }
< #endif
< #endif
518,523c508,512
<         if (optSpecial) fprintf (stderr, "[-%%RdfgJOHNMqmKZVw] ");
< #ifdef LICENSE
<         if (LIC_IS_MODEL("optem"))
<             fprintf (stderr, "[-cCl%s%s%s%sFTI%suntvhi%sxy] [-a time]",
<                         OPT_3, OPT_R1, OPT_G, OPT_B, OPT_X, OPT_R2);
<         else
---
>         if (optSpecial) {
> #ifdef CAP3D
>             fprintf (stderr, "[-%%RdfgJOHNMqmKZVw012] [-%%L max_depth] [-%%A step]");
> #else
>             fprintf (stderr, "[-%%RdfgJOHNMqmKZVw02] [-%%L max_depth] [-%%A step]");
525,528c514,519
<             fprintf (stderr, "[-cCl%s%s%s%sFTI%suntvhi%sx] [-a time]",
<                         OPT_3, OPT_R1, OPT_G, OPT_B, OPT_X, OPT_R2);
<         if (optSpecial) fprintf (stderr,
<                 "\n       [-%%L max_depth] [-%%A step] ");
---
>             fprintf (stderr, "\n       ");
>         }
>         s = "";
>         if (LIC_IS_MODEL("optem")) s = "y";
>         fprintf (stderr, "[-cCl%s%s%s%sFTI%suntvhi%sx%s] [-a time]",
```

```
>                 OPT_3, OPT_R1, OPT_G, OPT_B, OPT_X, OPT_R2, s);
587a579,582
>     optSubResInternal = paramLookupB ("sub3d.internal", optDisplay ? "on" : "off");
>     optSubResOldMesh  = paramLookupB ("sub3d.old_mesh", "off");
>     runSpecialPrePass = !optSubResInternal && !optSubResOldMesh;
>     if (optDisplay && optOnlyPrePass) runSpecialPrePass = 0;
727,731c722
< #ifdef NEW_B1
<         if (optCap3D || optOnlyPrePassB1) { /*}*/
< #else
<         if (optCap3D || optSubRes) {
< #endif
---
>         if (optCap3D || optOnlyPrePassB1 || optSubRes && optSubResOldMesh) {
847a837
>                 int prePass1B = optSubRes && (optOnlyPrePassB1 ||
                                                optSubResInternal && !optSubResOldMesh);
863,864c853,854
<                 if (substrRes) {
<                     lastPass = optOnlyPrePass && !optPrick && !optOnlyPrePassB1;
---
>                 if (substrRes && !optOnlyPrePassB1) {
>                     lastPass = optOnlyPrePass && !optPrick && !prePass1B;
868,870c858,859
<                     prePass1 = 1; prepass (list + i);
<                     if (!optOnlyPrePassB1) processgln (list + i);
<                     prePass1 = 0;
---
>                     prePass1 = 1; prepass (obj); prePass1 = 0;
>                     if (optSimpleSubRes || runSpecialPrePass) processgln (obj);
874a864,882
>                 if (prePass1B) {
>                     bool_t optB1 = optOnlyPrePassB1;
>                     bool_t optRM = optResMesh;
>                     lastPass = optOnlyPrePassB1 || optOnlyPrePass && !optPrick;
>                     optOnlyPrePassB1 = 1;
>                     optResMesh = 0;
> #ifdef DEBUG
>                     if(DEBUG) fprintf(stderr, "PASS: starting prepass 1B\n");
> #endif
>                     pp1SetColors ();
>                     prePass1 = 1; prepass (obj); prePass1 = 0;
>                     pp1ResetColors ();
>                     ...
>                     optOnlyPrePassB1 = optB1;
>                     optResMesh = optRM;
>                     if (lastPass) continue;
>                 }
934a943
>     optBackInfo = optNetInfo = FALSE;
938,939c947
< #ifdef NEW_B1
<         if (!optOnlyPrePassB1) {
---
>         if (!optSubResOldMesh && !optOnlyPrePassB1) {
944d951
< #endif
946d952
<     if (!lastPass) optBackInfo = optNetInfo = FALSE;
1049,1050c1053,1058
<     else if (prePass1)
<         verbose ("prepassing %s for substrate resistance\n", cellname);
---
>     else if (prePass1) {
```

```
>         char *note = "";
>         if (optSubResSave) note = " prepare";
>         else if (optOnlyPrePassB1) note = " calculate";
>         verbose ("prepassing %s for substrate resistance%s\n", cellname, note);
>     }
1213,1214c1221,1225
<         dmCheckIn (circuitKey, COMPLETE);
<         _dmAddCellEquivalence (dmproject, cellname, LAYOUT, circuitName, CIRCUIT);
---
>         if (circuitKey) {
>             dmCheckIn (circuitKey, COMPLETE);
>             _dmAddCellEquivalence (dmproject, cellname, LAYOUT, circuitName, CIRCUIT);
>             circuitKey = 0;
>         }
1216,1217d1226
<     else if (optOnlyPrePassB1)
<         dmCheckIn (layoutKey,  COMPLETE);
1353,1354c1363
< #ifdef NEW_B1
<     else if (runSpecialPrePass) { /* prePass1: optSubRes */
---
>     else { /* prePass1: optSubRes */
1387d1395
< #endif

Diff: scan/tile.c (4.26 <-> 4.27)
=================================================================
59c59
< tile_t * createTile (xl, bl, xr, br, color, stb, stl)
> tile_t * createTile (xl, bl, xr, br, color, stb, stl, cc)
62a63
> int cc;  /* for special prepass: init causing conductor */
87c88
<     t -> known = 0;
>     t -> known = cc;
435c436
<     t[j] = createTile (-INF, j, INF, INF, colorNull, NULL, NULL);
>     t[j] = createTile (-INF, j, INF, INF, colorNull, NULL, NULL, 0);

Diff: scan/update.c (4.26 <-> 4.27)
=================================================================
100c100
<         freeTile -> color, edge -> bwd -> bwd -> tile, freeTile);
>         freeTile -> color, edge -> bwd -> bwd -> tile, freeTile, edge -> bwd -> cc);
113c113
<     color, edge -> bwd -> tile, freeTile);
>     color, edge -> bwd -> tile, freeTile, edge -> cc);
175c175
<             freeTile -> color, edge -> bwd -> bwd -> tile, freeTile);
>             freeTile -> color, edge -> bwd -> bwd -> tile, freeTile, edge -> bwd -> cc);
184c184
<             freeTile -> color, edge -> bwd -> bwd -> tile, freeTile);
>             freeTile -> color, edge -> bwd -> bwd -> tile, freeTile, edge -> bwd -> cc);
317c317
<         color, edge -> bwd -> tile, freeTile);
>         color, edge -> bwd -> tile, freeTile, edge -> cc);
367c367
<             oldFreeTile -> color, edge -> bwd -> bwd -> tile, oldFreeTile);
>             oldFreeTile -> color, edge -> bwd -> bwd -> tile, oldFreeTile, edge -> bwd -> cc);
375c375
<             oldFreeTile -> color, edge -> bwd -> bwd -> tile, oldFreeTile);
>             oldFreeTile -> color, edge -> bwd -> bwd -> tile, oldFreeTile, edge -> bwd -> cc);
430c418
<         freeTile -> color, edge -> bwd -> tile, freeTile);
```

```
>         freeTile -> color, edge -> bwd -> tile, freeTile, edge -> cc);
602c590
<             oldTile -> color, edge2 -> bwd -> tile, oldTile);
>             oldTile -> color, edge2 -> bwd -> tile, oldTile, edge2 -> cc);

Diff: spider/cap3d.c (4.44 <-> 4.45)
=====================================================================
172,173c172,173
<     greenCase = prePass1 ? SUBS : DIEL;
<     greenType = prePass1 ? 2 : 0;
---
>     greenCase = (prePass1 || !optCap3D) ? SUBS : DIEL;
>     greenType = (prePass1 || !optCap3D) ? 2 : 0;
248a249
> #endif /* DEBUG */
257d257
< #endif /* DEBUG */
346a347,352
>     Xl_3d = xl; Yb_3d = yb;
>     Xr_3d = xr; Yt_3d = yt;
>     if (!optCap3D) return (0); /* special case for optOnlyLastPass */
359,363d364
<     Xl_3d = xl; Yb_3d = yb;
<     Xr_3d = xr; Yt_3d = yt;

Diff: spider/mesh.c (4.23 <-> 4.24)
=====================================================================
254a255
>     stripAddSpider (sp3);

Diff: spider/refine.c (4.20 <-> 4.21)
=====================================================================
392a393
>     stripAddSpider (spe);

Diff: spider/spider.c (4.12 <-> 4.13)
=====================================================================
95c95
<     stripAddSpider (spider);
>  /* stripAddSpider (spider); /* done somewhere else (SdeG) */

Diff: spider/sptile.c (4.5 <-> 4.6)
=====================================================================
28a29,30
> extern bool_t optOnlyPrePassB1;
37c38
<     spider_t *tr, *sp, *c[4];
>     spider_t *tr, *sp, *sp2, *c[4];
56c57,58
<         for (sp = ccwa (tr, face); sp != tr; sp = ccwa (sp, face)) {
---
>         stripAddSpider (tr);
>         for (sp = ccwa (tr, face); sp != tr; sp = sp2) {
59a62
>             sp2 = ccwa (sp, face);
64a68,87
>                 stripAddSpider (sp);
>             }
>             else if (optOnlyPrePassB1) {
>                 spiderEdge_t *e1, *e2;
>                 e1 = sp -> edge;
>                 e2 = e1 -> nb;
>                 if (e2 -> nb || sp -> nom_x == xl || sp -> nom_x == xr)
>                     stripAddSpider (sp);
```

```
>                 else {
>                     e1 -> oh -> oh = e2 -> oh;
>                     e2 -> oh -> oh = e1 -> oh;
>                     /* Note: e1 and e2 stay in the strip and are later disposed.
>                        The sp is set to zero to flag its status for drawing. (SdeG) */
>                     e1 -> sp = 0;
>                     e2 -> sp = 0;
> #ifdef DEBUG
> fprintf(stderr,"spiderTile: disposeSpider %p (%g,%g,%g)\n",sp,sp->nom_x,sp->nom_y,sp->nom_z);
> #endif
>                     disposeSpider (sp);
>                 }
65a89
>             else stripAddSpider (sp);

Diff: spider/strip.c (4.29 <-> 4.30)
====================================================================
329a325,326
>     if (spider -> strip) return; /* SdeG: already in strip */
553,555d549
<     if (sp1 -> nom_x > stripA -> xl || sp2 -> nom_x > stripA -> xl) continue;
557c551,552
<      * Implement this by comparing spider pointer values
>      * Implement this by comparing spider pointer values.
>      * Note: sp1 is zero, when the edge is unused (SdeG).
559c554,557
<     if (sp1 > sp2) continue;
>     if (!sp1 || sp1 > sp2) continue;
>     if (sp1 -> nom_x > stripA -> xl || sp2 -> nom_x > stripA -> xl) continue;
626c623,625
<     if (sp1 > sp2) continue;
>     if (!sp1 || sp1 > sp2) continue;

Diff: substr/subcont.cc (4.23 <-> 4.24)
====================================================================
42a43
> extern bool_t optSubResSave;
167c166
<     if (optSimpleSubRes && !rsubdevtab) { /* init only once! */
>     if (optSimpleSubRes && !optSubResSave && !rsubdevtab) { /* init only once! */

Diff: X11/rgb.c (4.28 <-> 4.29)
====================================================================
36a37,38
> extern bool_t optOnlyPrePassB1;
175a178,181
>         if (optOnlyPrePassB1) { /* colorindex is conductor_nr */
>             i = colorindex (color);
>             return gcSpiderColor (i);
>         }
197c203,209
<     int newcolor = doConvertColor (color);
---
>     int newcolor;
>
>     if (optOnlyPrePassB1) { /* colorindex is conductor_nr */
>         newcolor = colorindex (color);
>         if (newcolor < 0) return gc_glass;
>         return gcSpiderColor (newcolor);
>     }
198a211
>     newcolor = doConvertColor (color);
```