SPACE PROGRAMMING NOTES

S. de Graaf

Circuits and Systems Group
Faculty of Electrical Engineering
Delft University of Technology
The Netherlands

Report ET-CAS 01-01
April 24, 2001

Last revision: December 3, 2003.

1. INTRODUCTION

This application note describes the internal structure of the *space* program.
It is intended for SPACE programmers.

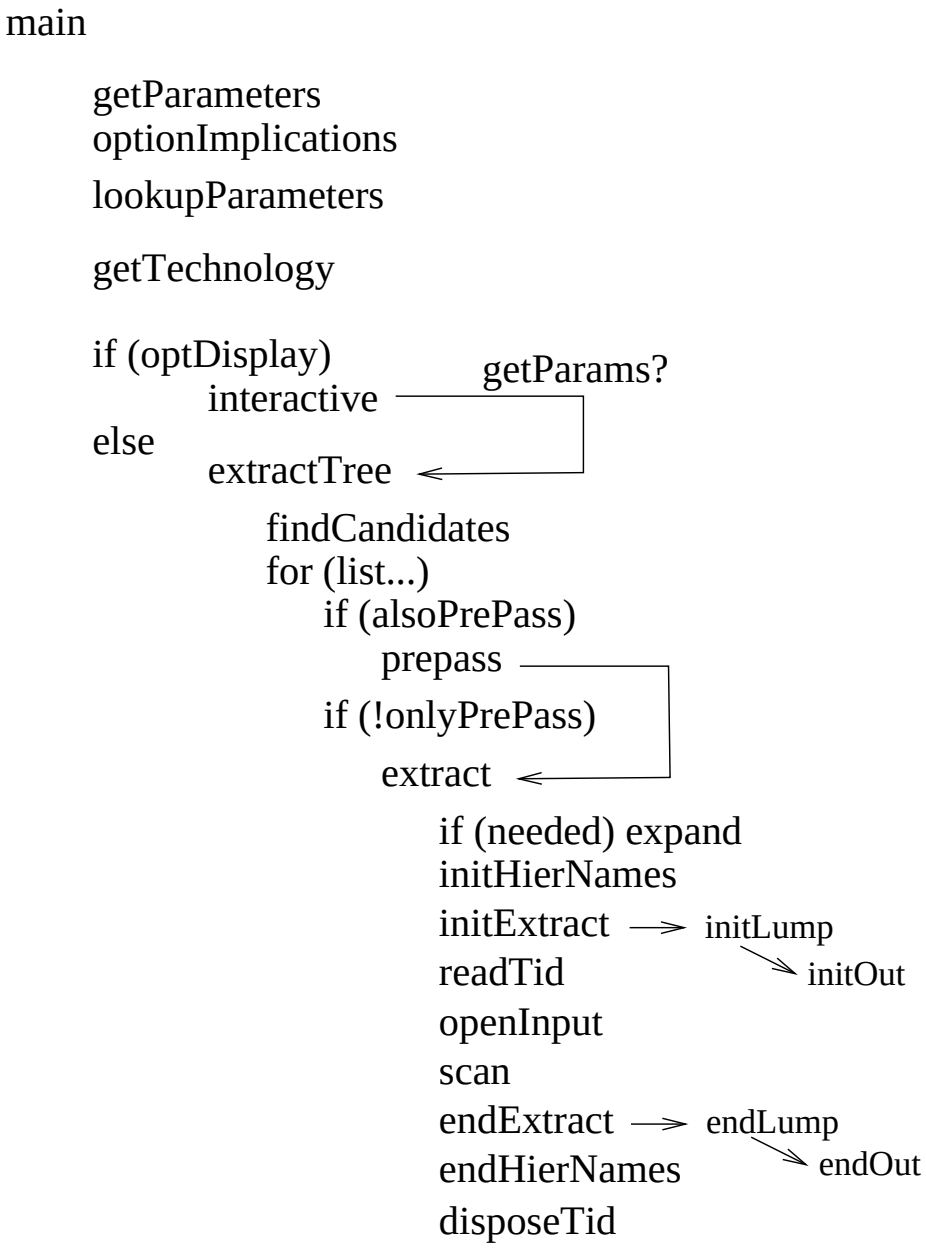
WORKING ENVIRONMENT

The current working environment is on Linux machines.

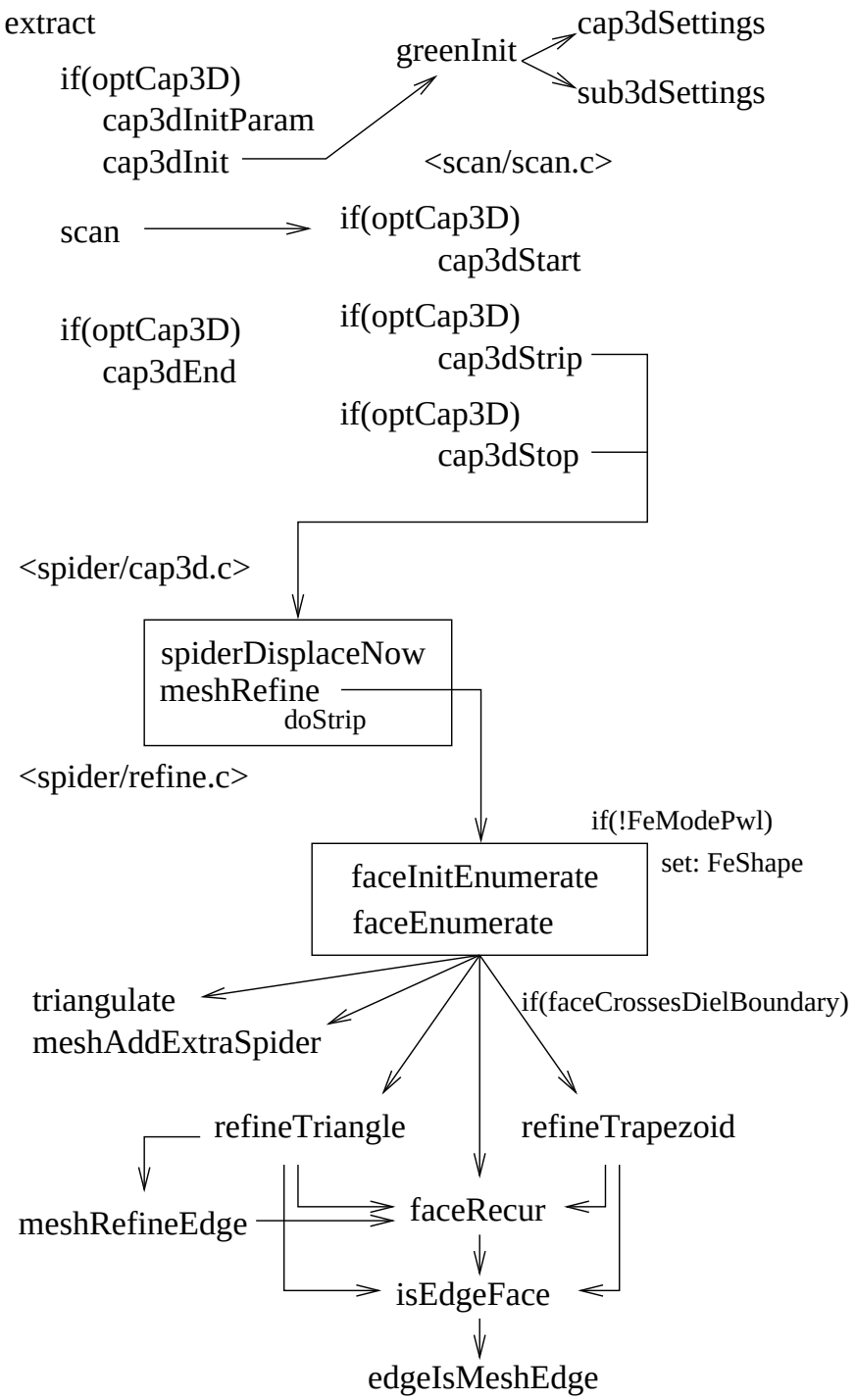
CHECKED IN SPACE SOURCES

The sources are last checked in on Tue 25-Nov-2003.
We are now waiting for release of the 5.0.4 version of the SPACE system.

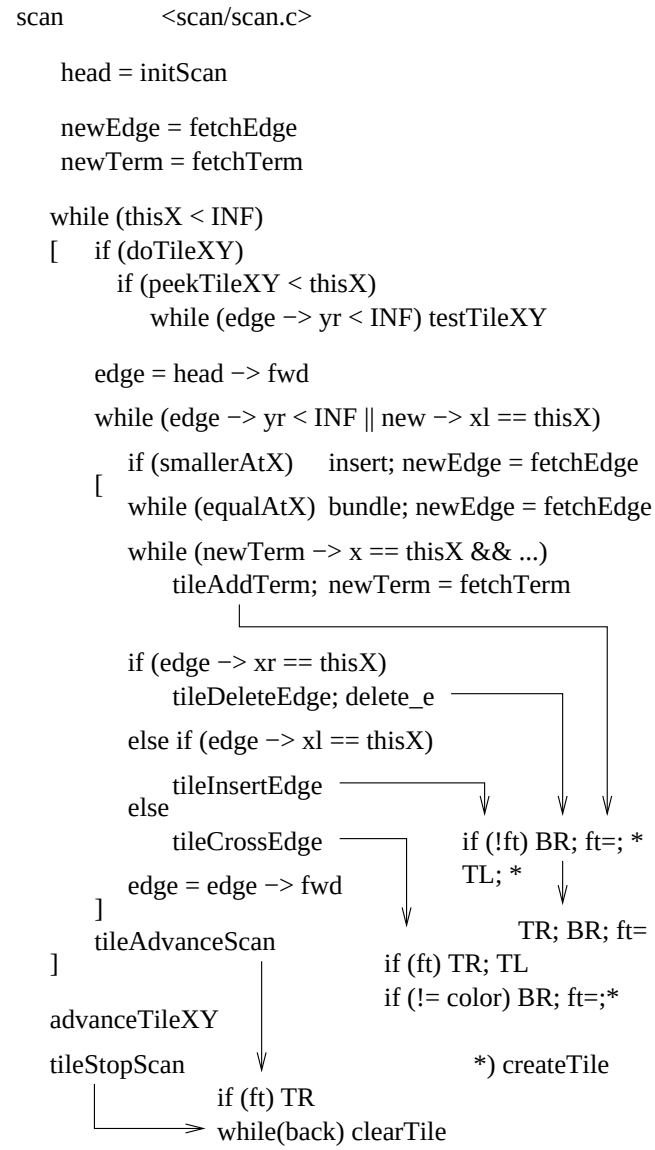
2. SPACE PROGRAM STRUCTURE: MAIN



3. SPACE PROGRAM STRUCTURE: CAP3D



4. SPACE PROGRAM STRUCTURE: SCAN



TL: tile −> tl = tl; if (t_left) enumPair (t_left, tile, 'v')
BR: tile −> xr = xr; tile −> br = br
if (t_bot && t_bot −> xl < xr) enumPair (t_bot, tile, 'h')
TR: tile −> tr = tr
if (t_top −> xl < tile −> xr) enumPair (tile, t_top, 'h')
enumPair (tile, t_right, 'v')
enumTile (tile)
if(bandWidth) inject(tile); else clearTile

5. RUNNING DIFFERENT SPACE VERSIONS

There are three different names used:

```
(1) space
(2) space3d
(3) Xspace
```

The program called with the name *space* is only a 2D extraction version and can not be used for accurate capacitance and resistance extraction and option **-X**.

The program called with the name *space3d* is a symbolic link to *Xspace* and is the complete extraction version. This version must be called with one or more cell arguments.

```
space3d cell1 [cell2 ...]
```

If this version is called with the **-X** option, only one cel argument can be used. In that case a graphic display shows the results of the cel extraction. This graphic display has no menu-buttons (parameter "disp.show_menu" is "off").

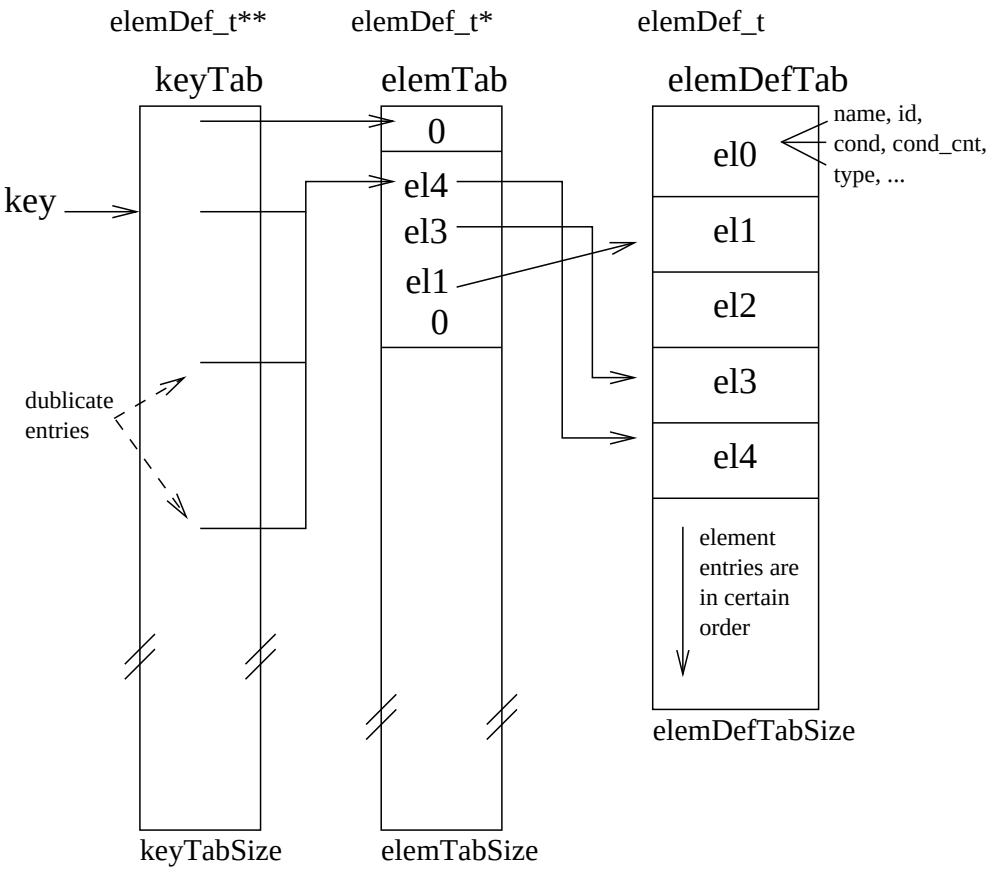
```
space3d -X cell1
```

When parameter "disp.show_menu" is "on", the invocation is equal to a call of *Xspace*. By *Xspace* the extraction is done after clicking on the "extraction" menu-button. The cell on the command line is the selected cell for extraction (is optional).

```
Xspace [cell1]
```

Note that *Xspace* with parameter "disp.show_menu" is "off", does directly the extraction of the first cel argument given (ignoring other cel arguments). No menu’s are displayed. If no cell argument is given, nothing is done. Note that *Xspace* and *space3d -X* use only the **flat** extraction mode! Note that **pseudo hier.** extraction mode also can be used.

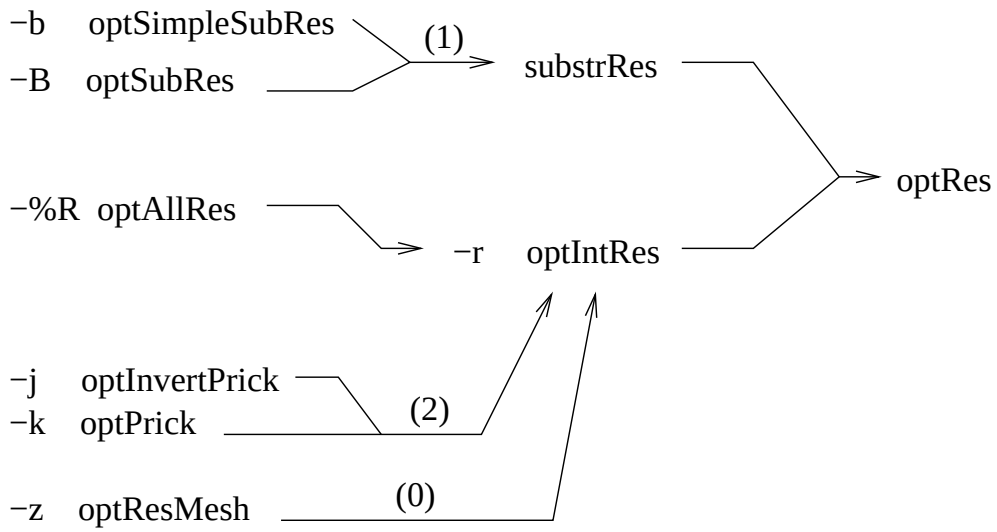
6. RECOGNIZING TECHNOLOGY ELEMENTS



7. RESISTANCE EXTRACTION

Resistance extraction is only available in the NOT PUBLIC version of space. When the program is compliled with the #define's NO_INT_RES and NO_SUB_RES and PUBLIC are OFF, full resistance extraction can be done if requested.
There are two types of resistance extraction, (1) interconnect extraction (optIntRes), and (2) substrate extraction (substrRes). The substrate resistances can only be connected to the interconnect resistances via substrate contacts.
Normally only the high-res conductors are taken into account, depending on the parameter "low_sheet_res" (default 1 Ohm). With option **-R**, all conductors > 0 Ohm are calculated (only possible with optSpecial **-%**). All resistances lower than parameter "min_res" (no default) are not outputted, but are shorts (0 Ohm conductors).

7.1 Resistance Extraction Options



−%Z optOnlyPrePass (seldom used)

Note that standard resistance extraction (**-r**) is not using any prepass. That "low_sheet_res" has precedence over selective resistance extraction. That the "known" bitmap in a tile is only set (with the conductor number) if the resistance must be taken into account. Note that options **-%s** (optSelectiveRes) and **-%w** (optAccWL) are removed.

7.2 Interconnect Resistance Extraction

All conductors with a value ≥ low_sheet_res (and > 0) are taken into account as resistances. Thus, conductors with a value of 0 are always only conductors. Default, all values ≥ 1 Ohm are taken into account. You can specify parameter "low_sheet_res", to

use another value. The special option **-R** is not needed, because **-S** *low_sheet_res=0* gives the same result. Each interconnected net forms a group, which can have many nodes. The tile→known conductor number bitmap is a flag for precense of resistance. The program tecc gives each conductor mask an unique conductor number (≥ 0). Other masks get the number -1. Note that the bitmap can contain atmost 32 bits. If you have more than 32 conductors, put the highest resistance conductors first in the technology file. *Space* cannot handle more than 32 resistance conductors at this moment.

7.3 The Supply Groups

Some groups are special, like the interconnect of the positive and negative power supply. These nets have predefined names in space. The negative supply is default called "VSS" and "GND". The positive supply is default called "VDD". This names are case insensitive. You can add a list of other (case sensitive) names with parameters "neg_supply" and "pos_supply".
Note that short detection is done for the supply groups. Each group has a supply flag, which is set to 1 for neg_supply and set to 2 for pos_supply. But if this flag becomes 3, a short message is given by function supplyShort() and a point (joiningX,Y) of this net is printed (if joiningCon ≥ 0). This can happen when groups are merged by function mergeGrps() or when a terminal or label name (which is a supply name) is added to a group with function nameAdd().

7.4 The GND Node

Only capacitive elements can be connected to this ideal ground node. Thus, no resistors can be connections to GND and also no tor bulk connections can be made with GND. The node pointer is equal to NULL. Thus, there is no real node structure (or group). Ground is also a kind of substrate. The default name is "GND" and is default also used for the negative power supply. With parameter "name_ground" you can specify another name. Elements are connected to "name_ground" by "@gnd" in the technology file. This global predefined name must **not** be used as terminal or label. Note that the GND net only exists by optCap is true.

7.5 The SUBSTR Group

Besides ground, there is another conductor plane, which is called the substrate. The default name for this plane is "SUBSTR". With parameter "name_substrate" you can specify another name. This global predefined name must **not** be used as terminal or label. Elements are connected to "name_substrate" by "@sub" or "%(mask_cond_list)" in the technology file. Note that tecc adds this conductor mask "@sub" to the masklist in the technology file, if one of more elements are connected to "name_substrate".

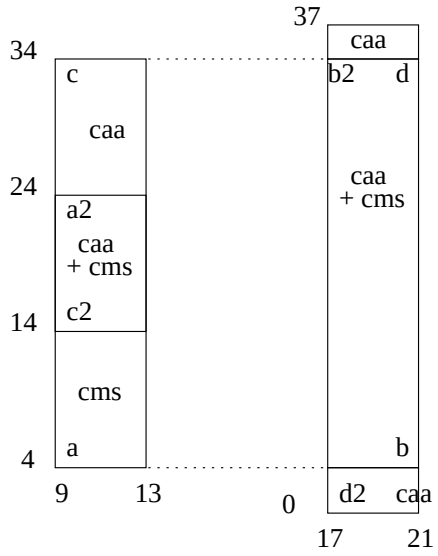
7.6 Substrate Resistance Extraction

For substrate resistance extraction it is required that conductor mask "@sub" is added to the masklist in the technology file. Substrate resistances can only be extracted if there are connections to the substrate. These via’s are called substrate contacts.

7.7 Circuit Reduction

Normally, the network is reduced. Too small resistances are removed from the network. Parameter "min_res" must be set to a minimal reduction value. If you don’t want to have circuit reduction, set option **-n** (optNoReduc) or parameter "heuristics off".

8. RESISTANCE EXAMPLE



term.size 2 x 2

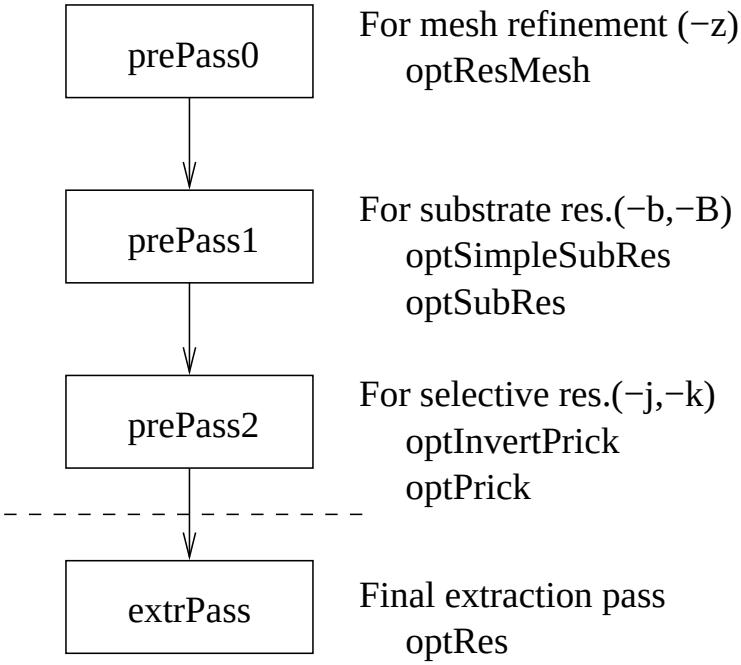
caa−term's: c, c2, d, d2
cms−term's: a, a2, b, b2

sheet res:
caa: 70 Ohm
cms: 0.03 Ohm

defaults:
low_sheet_res = 1 Ohm
min_res = 100 Ohm

```
% space3d -r -Slow_sheet_res=0 -Smin_res=0 cel1
% xsls cel1
network cel1 (terminal d2, c2, d, c, b2, a2, b, a)
{
    res 132.3399m (a, a2);      /* -z: 139.8267m */
    res 308.7931 (c2, c);       /* -z: 326.2622  */
    res 203.9452m (b2, b);      /* -z: 212.1421m */
    res 554.1682 (d2, d);       /* -z: 575.2406  */
}

% cat sel_con
9 13 cms
% space3d -rj -Slow_sheet_res=0 -Smin_res=0 cel1
% xsls cel1
network cel1 (terminal d2, c2, d, c, b2, a2, b, a)
{
    net {a, a2};
    res 308.7931 (c2, c);
    res 203.9452m (b2, b);
    res 554.1682 (d2, d);
}

% space3d -rk -Slow_sheet_res=0 -Smin_res=0 cel1
% xsls cel1
network cel1 (terminal d2, c2, d, c, b2, a2, b, a)
{
    net {c2, c};
    net {b2, b};
    net {d2, d};
    res 132.3399m (a, a2);
}
```

9. SPACE PREPASSES



The *space* program can use a maximum of 3 optional prepasses. The prepasses are used by specific type of resistance extraction. In the prepasses the functions resEnumPair and resEnumTile are not used, but only the functions enumPair and enumTile (see the note). Variable "lastPass" is used to flag the last pass. If option **-Z** is used one of the prepasses can be the last pass. Variable "extrPass" is used to flag the final extraction pass. The final extraction pass is not reached when option **-Z** is used. Note that variable "optCap3DSave" is used to save the initial setting of variable "optCap3D", this value can be changed in prepasses.
Note that, when expansion of the layout is needed, first the program steps *makeboxl* and *makegln* are done, and if needed also *makesize*.

Note: IN ALL PREPASSES
optAccWL = 0, optRes = 0, optLatCap = 0

9.1 Space Prepass0

Prepass0 is only used by option **-z**, when mesh refinement is requested. By the refinement, the conductor tiles are splitted in more tiles. Prepass0 creates an extra gln-stream "mesh_gln" by the *makemesh* program. This extra "mesh_gln" stream is read by all other program passes. It contains edges with color 0, which split up all conductors tiles in more pieces.

Note:	IN PREPASS 0	ALL OTHER PASSES
	prePass0 = 1	prePass0 = 0
	optResMesh = 0	optResMesh = 1
	substrRes = 0	
	optCap3D = 0 (bandWidth = 0)	
	optLatCap = optCoupCap = optCap = optRes = 0	

9.2 Space Prepass1

Prepass1 is only used by substrate res. extraction (option **-b** or **-B**). If both options are specified, only option **-B** is used. Both options set internal variable "substrRes". After prepass1 for **-b** the program *makedela* is called.

Note:	IN PREPASS 1 (-b)	
	prePass1 = 1	optSimpleSubRes = 1 optRes = 0
	substrRes = 1	optSubRes = 0
	optCap3D = 0	optLatCap = optCoupCap = optCap = 0

Note:	IN PREPASS 1 (-B)	
	prePass1 = 1	optSimpleSubRes = 0 optRes = 0
	substrRes = 1	optSubRes = 1
	optCap3D = 1	optLatCap = optCoupCap = optCap = 0

Note that the above settings are also valid for prePass2 (if used).
In that case is prePass2 = 1 and prePass1 = 0.

9.3 Space Prepass2

Prepass2 is only used by selective interconnect res. extraction (options **-j**, **-k**).

Note: IN PREPASS 2

prePass2 = 1 optPrick = 1 (-j, -k) optInvertPrick = 1 (-j)
substrRes = 0
optCap3D = optLatCap = optCoupCap = optCap = optRes = 0

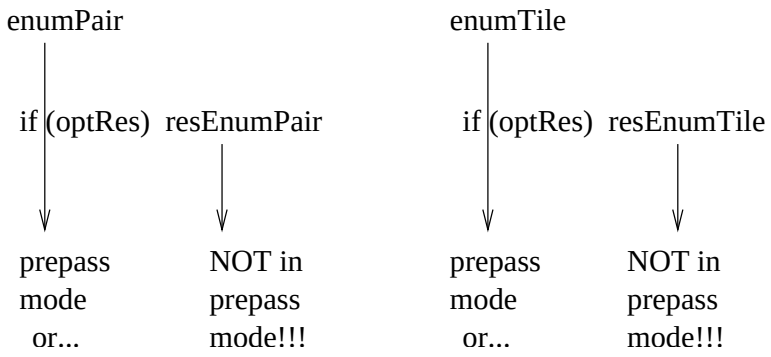
9.4 Space lastPass

The lastPass is normally the extrPass, except when option **-%Z** (optOnlyPrePass) is given. In that case no extrPass is done. Then, the lastPass is depending on given options. Note that some options (**-x**, **-y**) are only working in the lastPass, and that the cell circuit streams (mc, net, term) are only written in the lastPass.

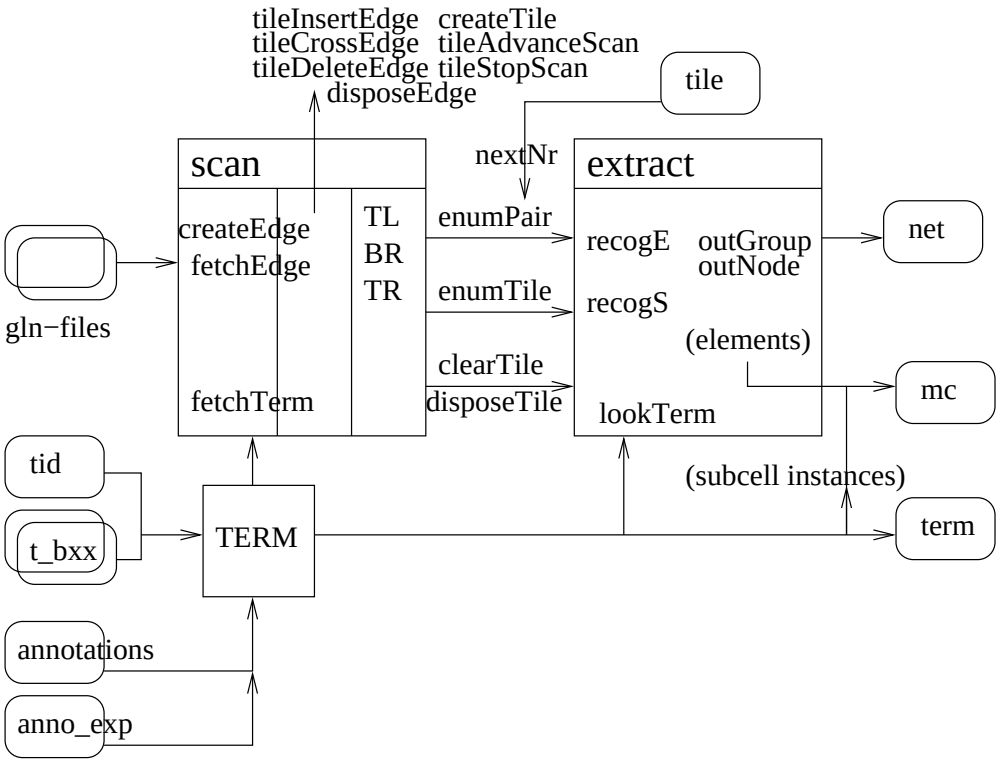
Note: IN EXTRPASS

extrPass = 1 substrRes = 1 (-b|-B)
optSubRes= 1 (-B) optSimpleSubRes = 1 (-b)
optCap3D = 1 (-3c|-3C)
optCap = 1 (-c|-C|-l)
optRes = 1 (-r|-R|-z, -b|-B, -j|-k)

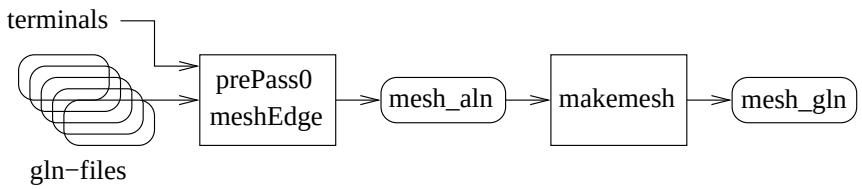
10. IMPORTANT PREPASS NOTE



10.1 The Scan / Extract Interface



11. PREPASS0: MESH REFINEMENT



By option **-z** (mesh refinement), prePass0 is used to make a temporary input stream "mesh_aln" in the layout view for the *makemesh* program. The *makemesh* program adds the stream "mesh_gln" to the set of mask "gln" files. This "mesh_gln" stream is read in all other space passes with the "gln" files. It adds extra edges (using color 0) to the resistance conductor tiles.

A special enumPair function (pp0EnumPair) handles this prePass0 situation.

Function meshEdge() writes:

- two points, if both tiles contain at least one same res-conductor.
- nothing for ’h’ edges, if both tiles contain a res-conductor but don’t overlap.
- the edge, if one (or both) of the tiles contain a res-conductor.

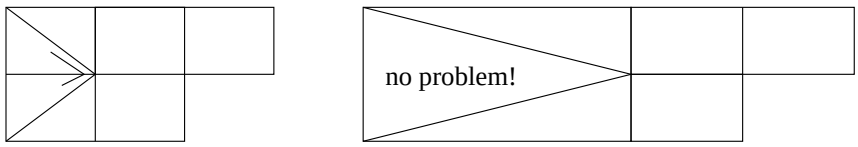
Note that by ’v’ edges also points can be generated for each terminal position on the vertical edge. Note that a terminal can be label, but also a terminal or label of a subcell.

Note that "mesh_gln" adds maximum horizontal strips for conductors to the layout.

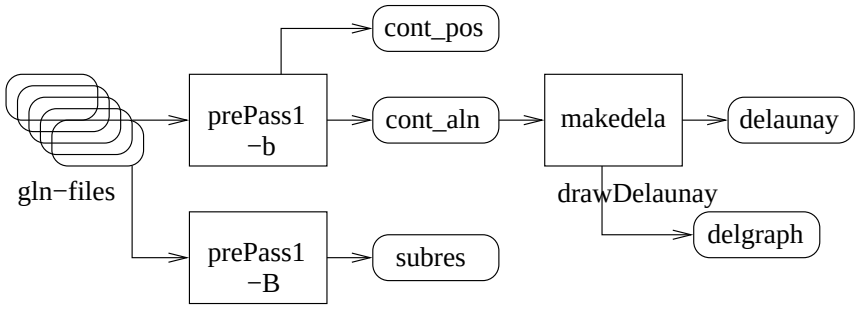
Note that "mesh_gln" contains the data for all high-res conductors. This could be reduced by making first a possible high-res selection (if used).

Note that "mesh_gln" does not need to be generated twice. It can possibly be reused (skipping prePass0). The high-res conditions must be the same (must be stored in a file and checked).

It is maybe possible to make the additional horizontal mesh in the extrPass. Not always are the maximum horizontal strips needed. Goal: The accuracy must be good enough. The triangularization may not get angles > 90 degrees (see figure).



12. PREPASS1: SUBSTRATE RES EXTRACTION



There are two modes of substrate resistance extraction.

- option **-b** (optSimpleSubRes), simple (2D) extraction.
- option **-B** (optSubRes), accurate (3D) extraction.

See functions initSubstr(), subContNew() and subContDel().
Function subContNew() does in prePass1 a call to subContGroupNew(). In the extrPass subContNew() reads info→area and info→perim, and reads next_nr, next_grp, next_xl, next_yb.

Function subContDel() writes stream "cont_pos" (**-b**) or "subres" (**-B**).

Format of "cont_pos":

```
 (cnt)     (id)
<next_nr> <next_grp> <xl> <yb> <area> <perimeter> \n
 ...
```

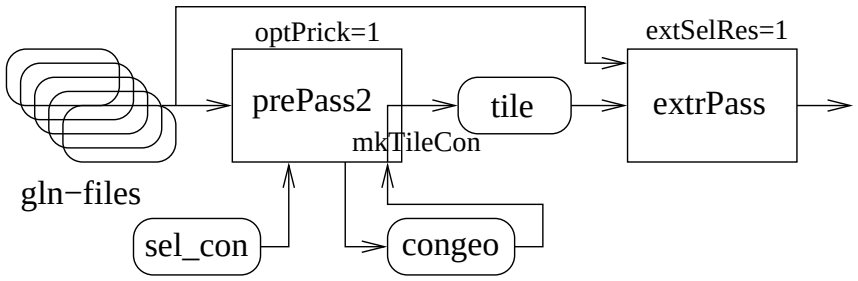
Format of "subres" (c=contact, nc=neighbor_contact):

```
c  %d %d xl %ld yb %ld g_sub %le \n   (term# grp# xl yb res_value)
nc %d g %le \n                       (term# res_value)
 ...
nr_neigh %d \n                (count: nr's used)
```

12.1 Substrate Contacts

See the "Space Substrate Resistance User's Manual". A substrate contact is a contact between the substrate and an interconnect layer. In prePass1 this connections are note made, because space is looking only to the substrate. Note that this connections are also not made in prePass2! Thus interconnect parts are not selected via the substrate, but these connections are made in the extrPass.

13. PREPASS2: SELECTIVE RES EXTRACTION



There are two methods of selective resistance extraction.

- option **-j** (optInvertPrick) and **-k** (optPrick).
- option -%s (optSelectiveRes).

By options **-j** and **-k** a "sel_con" file is used. High resistive interconnect (determined by the value of "low_sheet_res") can be exclusively selected (**-k**) or unselected (**-j**) to have resistance by output. Coordinate points (+ mask_name) records or/and a terminal name record can select groups. The terminal name is possibility not documented. Variable prickName is used to point to the terminal name, it can only point to one terminal (only last specified is used). To specify a terminal name, the name must be preceded with an '=' sign. By scanning, function testTileXY() sets the grp→prick flag. By output, function outTileConnectivity() writes "congeo" records based on this flag. Note that conductor groups never are connected with each other via substrate contacts. However, contacts between interconnect conductor groups join these groups together.

Note that in prePass2 no contacts to substrate are made. On the end of prePass2, function mkTileCon() converts the "congeo" stream into the "tile" stream. In the extrPass, only resistances for tiles found in the "tile" stream are extracted. See function resEnumPair, if (extSelRes) selectSplit = testTileCon (tileCnt).
Variable selectSplit is set to an array of conductor flags, when tileCnt ≡ nextNr.

By option **-s** (obsolete in Dec'02), group tileInfo→rSource and tileInfo→resWeight are used for selection. Function outTransistor() calls accountRSource(), this function sets rSource ≥ 0 if parameter tor_impedance is set ≥ 0. Value rSource becomes the smallest L/W-ratio found in the group. There are three other parameters which must be specified, parameter "imp_res_ratio", "select_res" and "min_connect_width". The value resWeight is set in function estimateRes(), which calls subnodeRes(). It is an accumulation of the resistance of the conductor edge length. For selection resWeight must be ≥ select_res and resWeight * imp_res_ratio ≥ rSource. This selection is done by output in function outTileConnectivity().
Note that, after **-s** is made obsolete, no transistor elements need to be generated in any prepass.

13.1 Another Selective Resist Method

There was also another omitRes / onlyRes method in space (not using a prePass). But this method is become superfluous because of the **-j**, **-k** option. Thus it is put off, because space is not compiled with OMIT_ONLY_RESIST defined.
The method was used in the extrPass (by optRes), when there exists an "omit_resist" file or else an "only_resist" file. In function nameAdd(), the terminals and labels of the top cell are compared against the file list (note that also wildcards may be used in the list).
If a match is found, then the grp→noResis flag is set or reset. In function readyGroup() is this grp→noResis flag tested. If this flag is true, then all interconnect nodes in the grp are joined together.

14. BACKANNOTATION

This mode is enabled with option **-x** (optBackInfo) or parameter "backannotation". Note that **-x** also enables option **-t** (optTorPos, parameter "component_coordinates"). Backannotation is only done in the lastPass.

The effect for **-t** is, that transistor x,y positions are added to the netlist. Note that x,y positions of subcells are read from stream "tidpos". The x,y positions are added as attributes with function addCoorXY().

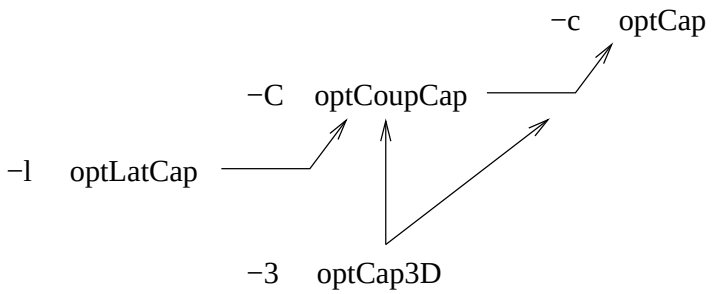
Backannotation generated interconnect and device geometry information for the program *highlay*. This information is written in the circuit view streams "congeo" and "devgeo". Note that the "congeo" stream is also used for temporary results in prePass2 by selective resistance extraction.

For programModel OPTEM, backannotation information can be written in the lastPass to stream "wiredata". This stream is only written for optNetInfo (option **-y** or parameter "netinformation"). The program must be compiled with NO_NETINFO undefined.

15. CAPACITANCE EXTRACTION

There are different types of capacitances. The PUBLIC version supports only the capacitances to ground (substrate). This can be area (surface) and edge capacitances. Other coupling capacitances (between different masks) can be extracted with option **-C**. Also lateral capacitances can be extracted with option **-l**. In 3D-mode (space3d), the capacitances are calculated with vertical dimensions (option **-3**). The program must be compiled with CAP3D defined.

15.1 Capacitance Extraction Options

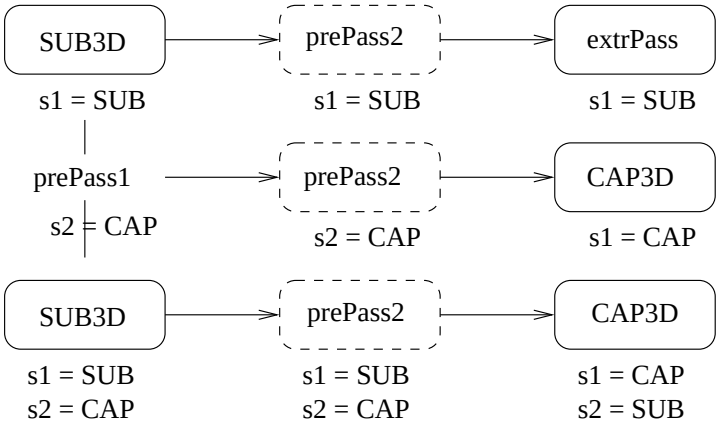


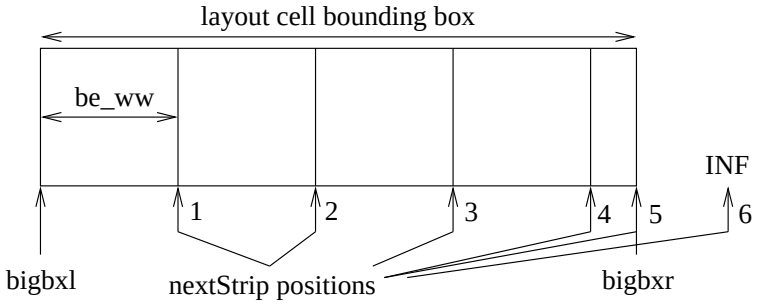
16. CAP3D AND SUB3D

Capacitance 3D extraction (CAP3D) is done in the extrPass. Use option **-3** in combination with the options **-c** or **-C**. This extraction method scans the layout in strips. Parameter "cap3d.be_window" must be specified for the strip width (and height).

Substrate 3D extraction (SUB3D) is done in the prePass1 (option **-B**) and combined in the extrPass with other extract results. This extraction method scans also the layout in strips. Parameter "sub3d.be_window" must be specified for the strip width.

This strip method can split tiles in function tileCrossEdge(), because the strip x-positions are also scanline positions. To get correct results, prePass1 must know the strip x-positions of CAP3D (if used). Also the extrPass must know the strip x-positions of SUB3D (if used). And both strip x-positions must also be used in prePass2 by selective resistance extraction. See the figure.





I detected that the *space* program must also use strip s2 in prePass1 for substrate resistance extraction with option **-b** by CAP3D extraction.
Modified "scan/scan.c", see following "scan.c" code fragment:

```
void scan ()
{
    ...

    if (optCap3D) { /* prePass1 || extrPass */
        nextStrip1 = cap3dStart ();
        while (nextStrip1 <= thisX) nextStrip1 = cap3dStrip ();
        if (prePass1 && optCap3DSave)
             nextStrip2 = findStripForCapStart (thisX);
        else if (extrPass && optSubRes)
             nextStrip2 = findStripForSubStart (thisX);
        else nextStrip2 = INF;
        nextStrip = nextStrip1 < nextStrip2 ? nextStrip1 : nextStrip2;
    }
    else if (!prePass0) {
        if (optSubRes)
             nextStrip1 = findStripForSubStart (thisX);
        else nextStrip1 = INF;
        if (optCap3DSave)
             nextStrip2 = findStripForCapStart (thisX);
        else nextStrip2 = INF;
        nextStrip = nextStrip1 < nextStrip2 ? nextStrip1 : nextStrip2;
    }
    else nextStrip = INF;

    Debug (fprintf (stderr, "nextStrip=%d0, nextStrip));
    ...

    while (thisX < INF) {
       ...

        if (stripSplit) { /* thisX == nextStrip */
            stripSplit = 0;
            if (nextStrip1 == thisX) {
                if (optCap3D) nextStrip1 = cap3dStrip ();
                else nextStrip1 = findStripForSubNext (thisX);
            }
            if (nextStrip2 == thisX) {
                if (extrPass)
                     nextStrip2 = findStripForSubNext (thisX);
                else nextStrip2 = findStripForCapNext (thisX);
            }
            nextStrip = nextStrip1 < nextStrip2 ? nextStrip1 : nextStrip2;
            Debug (fprintf (stderr, "nextStrip=%d0, nextStrip));
        }
       ...
    }

    ...
}
```

17. INSTANCES, TERMINALS AND LABELS

The program *makeboxl* expands the layout data. Information about instances, top and subcell terminals and labels is written to the following streams in the layout view:

```
    tid     = cell_name, inst_name, nx/ny | term_offset, term_name, nx/ny
    tidnam  = full_inst_name ("tid" contains ".")
    tidpos  = inst_bbox_coordinates, dx/dy
    t_M_bxx = term_coordinates, term_offset#
   anno_exp = for hierarchical terminals and labels
```

Note that the topcell labels are in the "annotation" stream. Function readTid() reads the "tid", "tidnam" and "tidpos" streams. Function readTid() writes the top cell terminals to the circuit "term" stream with function addTerminal() and writes the subcell instances to the circuit "mc" stream with function addInstance().
With terminal/label coordinates are the terminal/label names connected to interconnect (conductances) in the netlist. Each terminal/label must have an unique name. Subcell terminals/labels have a leading hierarchical instance name tree (full names). How a long name is build, bepends on some settings.
Note that this subcells are the cells, which are not expanded (also called leafcells). The terminals of this cells are called leaf_terminals. Note that the names can become too long. *Space* gives a message when this happens. The list of truncated names can be found in the "cell_name.nmp" file.

Function newTerminal() adds terminals/labels to the TERM array. First, in readTid(), the root cell terminals and terminals of existing leaf cells are added to TERM[] (with type tTerminal). The terminals of these leaf cells have an instName pointer != NULL.
Second, by useLeafTerminals is true (parameter "leaf_terminals", default "off"), a temporary LABELNAME array is filled with function newLabelName(). And when these terminals are found in the "t_bxx" streams, they are added with full names to TERM[] as type tLabel2 terminals.
Third, if useAnnotations is true (when parameter "no_labels=off", default case), the labels of the root cell from the "annotations" stream are added to TERM[] as type tLabel terminals, by function readLabels().
Fourth, if useHierAnnotations ("hier_labels", default "off") or useHierTerminals ("hier_terminals", default "off"), also intermediate terminals/labels from the "anno_exp" stream are added to TERM[] (with type tLabel2), by function readHierNames().

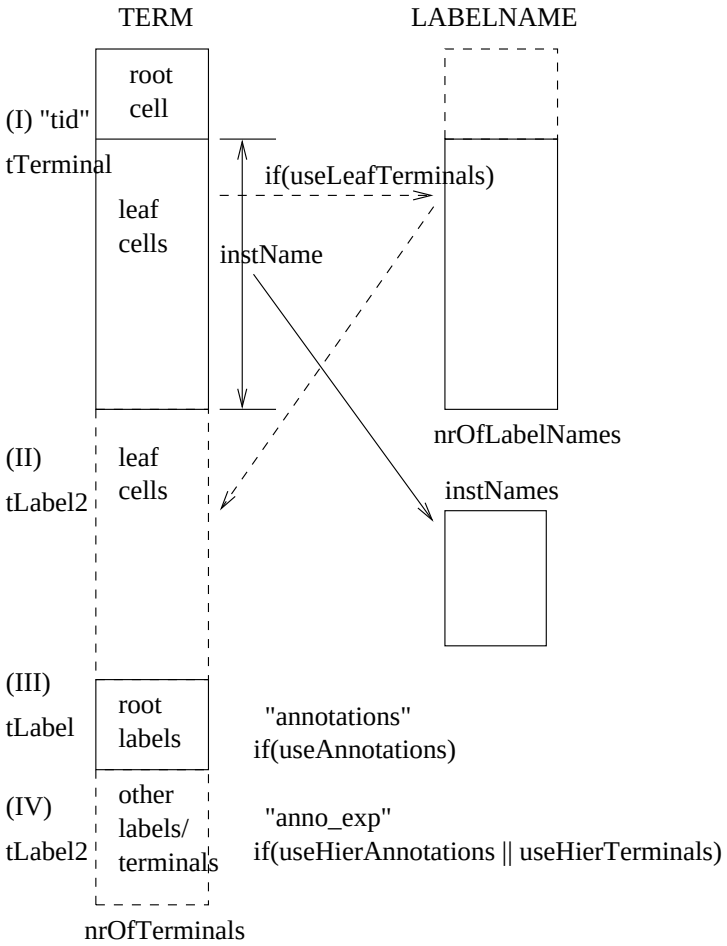
Note that there is made a copy of TERM to the TERMBYNAME array. **The TERM array is sorted by x,y coordinates with sortTerminals**(). The TERMBYNAME array is sorted by name with sortTerminalsByName(). The terminals with an instName pointer != NULL are at the end of TERMBYNAME.

Note, that readTid() initial reads the "tid" stream and adds terminals to the TERM array with INF coordinates and conductor number -1. Later, openInput() reads the "t_mask_bxx" streams and **gives each terminal x,y coordinates and a conductor number**. Each "t_mask_bxx" record contains an unique terminal chk_type offset, which

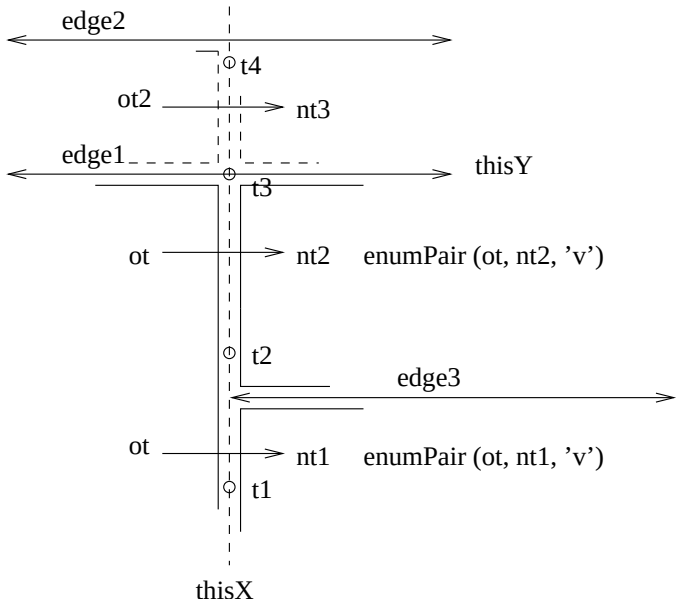
must be used as the TERM[] index. Note that there can be INF terminals at the end of TERM[] after sorting, if terminals are not found. A not connected warning message is given for these terminals.

Note that the TERMBYNAME array is used for (1) duplicate names checking and (2) net names checking. Function findTerminal() checks the net names in outNode(), and looks only to the names with no instName pointer. To know the last no instName entry variable nrOfTermNames is used.
Note, that the instance names are also saved in "instNames" and that the leaf terminals are pointing to this "instNames" space. By disposal, only "instNames" needs to be freed.

Note, that the program *makegln* adds all "t_mask_bxx" data to the "mask_gln" data (except for point terminals, which have no dimension). This is very important for the leaf terminals, to connect the leaf cells in the netlist.



(II) (III) (IV) don’t have copies!
Only tTerminal type can have copies (termX >= 0 || termY >= 0)!



Note that fetchTerm() reads the terminals from the TERM array. On scanline thisX, first terminal t1 is read, second t2 and then t3. **A terminal x position introduce always a scanline x position**. Edges are inserted in the scanline, terminal points not. Terminals can only split the edge tiles in function tileCrossEdge(), when thisX is not the edge→xl or the edge→xr position. All terminals on thisX position and with an y ≤ thisY are fetched. The last terminal t3 is laying on the edge1 (y ≡ thisY), this one shall also request a split of the tile above edge1. Terminal t1 is done first for the vertical edge parts of tiles ot and nt1. Terminals t2 and t3 are done for the vertical edge parts of tiles ot and nt2.

Note that the TERM_index always points to the newTerm, the terminal which is not processed yet. By the current thisY position newTerm is t4, because t4 lays above edge1 (y > thisY). After enumPair() for tiles ot and nt2, function lookTerm() can now use all terminals (t2 and t3) from array TERM for which TERM_look_index < TERM_index. If terminal t3 is not connected to tile ot or tile nt2, then it stays in the TERM array. The TERM_used_index is not updated. If there are more terminals on this position (y ≡ thisY), some can be connected to tile ot or tile nt2. In that case the terminals are swapped in the TERM array. Function newLookTerm() shall reset the TERM_look_index to the TERM_used_index, and shall start with terminal t3 again. Maybe t3 is connected to tile ot2 or tile nt3. Terminals (or labels) connected to tiles are added to nodes by function nameAdd(), and one is maybe used for the net name. Note that a warning message is given, when a terminal can not be connected.

Note that in the above figure both terminals t3 and t4 split the tile above edge1. But note, that after the first split tile ot2 becomes the free tile. The free tile is the tile, which is almost finished. And the tileCrossEdge() for edge2 finishes this tile.

18. NET NAMES

Each net in the network (or circuit) has a name. A net describes which which nodes are connected to each other. Whereby a node can be (a) a terminal or label, (b) an active or passive component pin, (c) a terminal or label of a sub circuit (cell), and (d) global net names. When a net only connects components (elements like resistors, capacitors, transistors), and the net contains no terminals or labels, it is a local net. A local net receives a net number.

19. FUNCTION: enumPair

enumPair

if(both !HasConduc) return

if(optRes) resEnumPair

if(newHasCond && !known)
elem = RecogE
etc.

if(bipoEl) bipoPair
if(..) updateTorEdge
if(..) updateEdgeCap
updateLatEdgeCap

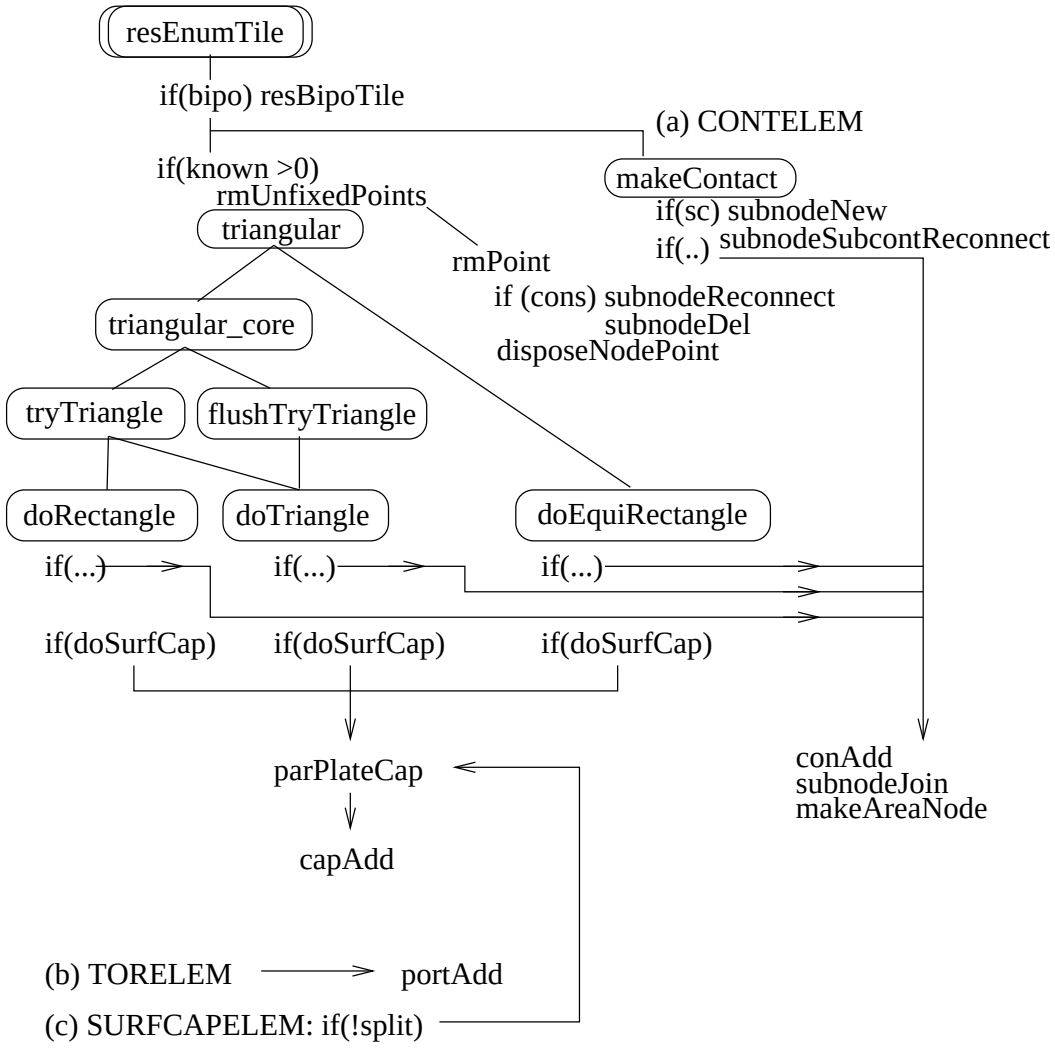
if(optCap3D) spiderPair

if(..) connectPoints
if(..) placePoint
if(..) updateTorEdge
if(..) updateResEdgeCap
updateLatEdgeCap

if(bipoEl) resBipoPair
if(optCap3D) spiderPair

20. FUNCTION: enumTile

```
enumTile

    if(!HasConduc) return
    if(optCap3D) spiderTile

    elem = RecogS
    doSurfCap = optCap && ...

    if(optRes)
        resEnumTile
    else
        if(bipo) bipoTile
        for (elem...)
            if(CONTELEM) subnodeJoin
            if(TORELEM) portAdd
                            tile->tor->surf += surf
            if(SURFCAPELEM)
                    if(doSurfCap) capAdd

        if(optSimpleSubRes && pp1)
                    tile->subcont->area += surf
```


resEnumTile
if(bipo) resBipoTile
(a) CONTELEM
if(known >0)
rmUnfixedPoints
triangular
makeContact
if(sc) subnodeNew
if(..) subnodeSubcontReconnect
rmPoint
if (cons) subnodeReconnect
subnodeDel
disposeNodePoint
triangular_core
tryTriangle
flushTryTriangle
doRectangle
doTriangle
doEquiRectangle
if(...)
if(...)
if(...)
if(doSurfCap)
if(doSurfCap)
if(doSurfCap)
parPlateCap
capAdd
conAdd
subnodeJoin
makeAreaNode
(b) TORELEM
portAdd
(c) SURFCAPELEM: if(!split)

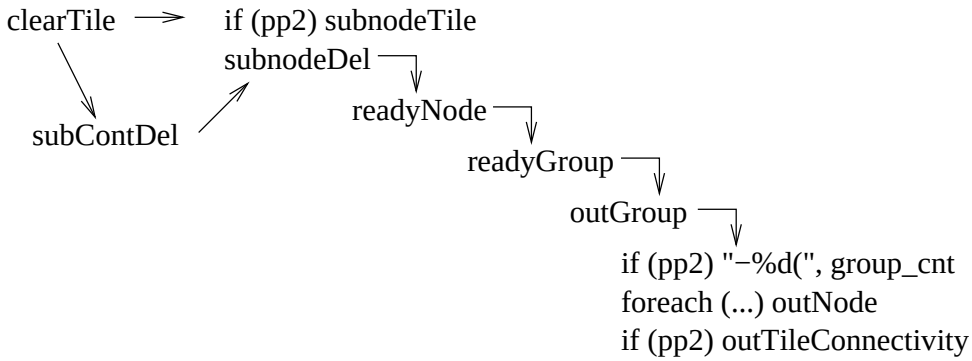
21. FUNCTION: clearTile

Function clearTile() finish the usage of a tile. After that the function disposeTile() is done. If the tile has a tor (transistor element), clearTile() calls subtorDel(). If the tile has a subcont (substrate contact element), clearTile() calls subContDel(). The function does also subnodeDel() for each existing conductor. By optBackInfo (option **-x**) also the functions subtorTile() and subnodeTile() are called. Note that subtorTile() stores tile data in tor→tileInfo→tiles, and subnodeTile() tile data in grp→tileInfo→tiles. The following tile data is stored: cnt, cx, color and coordinates. Note that subnodeTile() is also called in prePass2. Function subtorDel() does outTransistor(). Function subContDel() does subnodeDel() and ... Function subnodeDel() does readyNode() and ...

Note that clearTile() is not needed for tiles connected to infinity boundaries, or for tiles which have no conductors!

```
<scan/update.c>

void tileAdvanceScan (edge, thisX)
{
    if (freeTile)
        TR (INF, freeTile, edge -> bwd -> tile, edge -> tile);

    while (back) {
        if (back -> xr > thisX - bandWidth) break;
        clearTile (back); disposeTile (back);
        back = back -> next;
    }
}

void tileStopScan ()
{
    while (back) {
        clearTile (back); disposeTile (back);
        back = back -> next;
    }
}

private void TR (tr, tile, t_right, t_top)
{
    enumPair (tile, t_top,   'h');
    enumPair (tile, t_right, 'v');
    enumTile (tile);

    if (bandWidth == 0) {
        clearTile (tile); disposeTile (tile);
    }
    else inject (tile);
}
```

```
void clearTile (tile)                       //  <extract/clrtile.c>
{
    if (prePass0) return;

    if (tile -> tor) {
        if (optBackInfo)
            subtorTile (tile, tile -> tor -> type -> s.tor.gCon);
        subtorDel (tile, NULL);
    }

    if (tile -> subcont) subContDel (tile); /* substrate terminal */

    if (!optRes) {
        subnode_t ** cons = tile -> cons;

        for (i = 0; i < nrOfExtConductors; i++) {
            if (cons[i]) {
                if (prePass2  /* optPrick */
                    || optBackInfo) subnodeTile (cons[i], tile, i);
#ifndef NO_NETINFO
                if (optNetInfo) subnodeCoor (cons[i], tile, i);
#endif
                subnodeDel (cons[i]);
            }
        }
    }
    else {
        nodePoint_t *point, *p;
        int flag = 1;

        p = tile -> rbPoints;
        while (point = p) {
            p = point -> next;
            finishPoint (point, (flag ? tile : NULL));
            flag = 0;
        }
        ...
    }
}
```



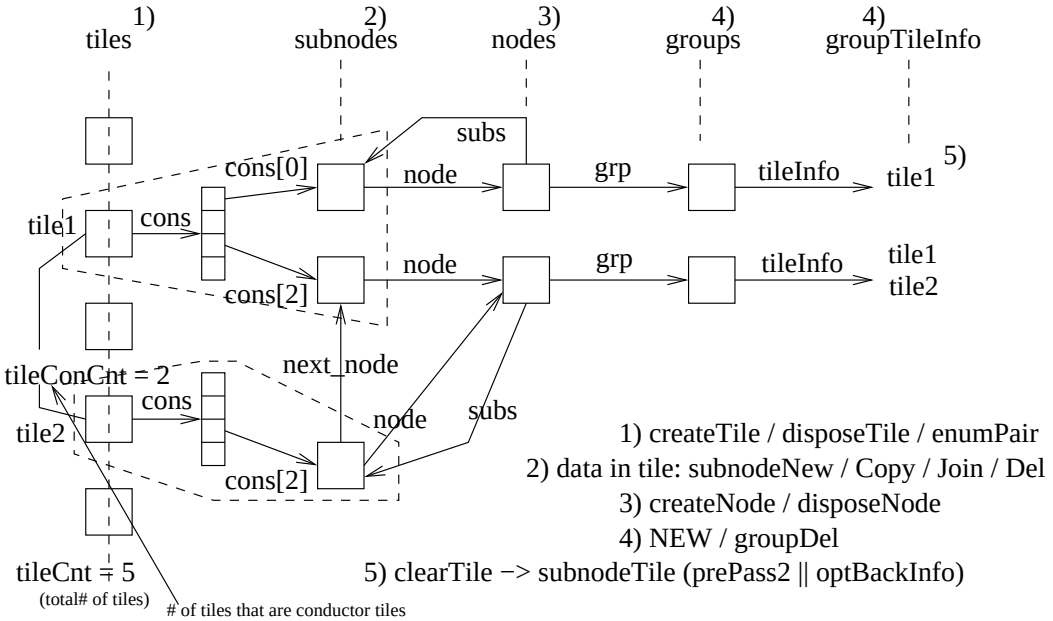
Function subnodeTile() puts tile information for each existing conductor tile→cons[i], where i is a conductornumber ≥ 0 (< nrOfConductors), in the group tileInfo tiles list. This tile information is put in different groups, when there are more conductors in the tile. But it is possible that these conductors are in the same group, when they are connected somewhere to each other.

The group tileInfo tiles list is possibly written to stream "congeo" in function outGroup by function outTileConnectivity. This is done in prePass2, when the group is selected by the prick flag. All groups are written in the extrPass when optBackInfo (**-x**) is true.

Function outTileConnectivity writes a group record (starting with a groupnumber), and tile records (each starting with the tilenumber).

In prePass2, flags substrRes and optCap3D are put off. This has no effect for the correct function of selective res output. Also the work that function outNode does (and its working is maybe changed) has no effect for the correct function of selective res output.

Note that first call to enumPair() for each new tile sets the tilenumber and sets tile→cons[i], when a RESELEM with conductornumber i was found by RecogS(). The joining of the groups is a important matter. Because only the group(s), where the x,y-point is laying in, is selected. Or only the group, where the prickName is in, is selected.
Note that substrate contacts do not connect interconnect parts to each other in prePass2. Thus, shorts via the substrate can not be detected in prePass2. Note that CONTELEM joins are done in enumTile().

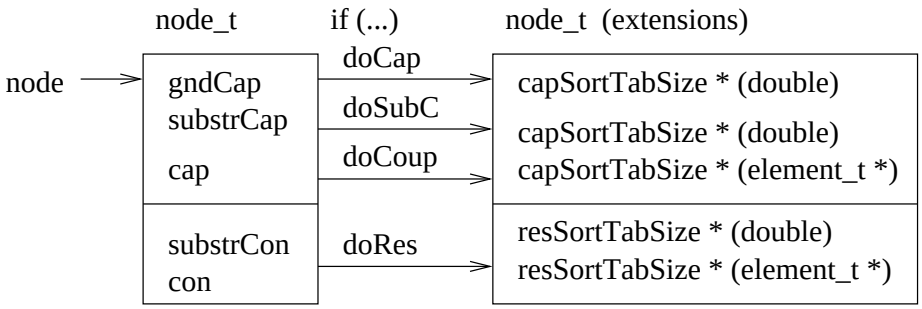


22. NODES AND GROUPS

A new node and group is created by function subnodeNew (see lump/lump.c). This function is called when a RESELEM (conductor) is found by enumPair for a new tile. The tile→cons[i] subnode pointer is set and the subnode points to the new node (see also the figure on previous page).

22.1 Nodes

A node represents a net. Nodes are created with function createNode (see lump/node.c). This function returns a "node_t" pointer. See also "include/node.h" for the data structure specification. The function uses a free node list to get nodes from. If the free_list is empty, it calls function allocNode. The size of the "node_t" data structure is not always the same. The size is determined by its first call. Later on, some parts are maybe not used.



If a pointer to an array is not null, the array members are initialized to 0.

22.2 Groups

A group represents one or more nodes, which represents an interconnect line. If a group contains two or more nodes, then there are resistors between them. A new group is allocated by subnodeNew() with NEW and (if needed) also the tileInfo struct is allocated. No free lists are used for these data structures.
Initial, one node is in the group (grp→nod_cnt is 1 and node→n_children is 0). This first node is the gr_father and points to itself. A group is not pointing to its nodes, but the nodes are pointing to a group. If two different groups are merged (by function mergeGrps), the nodes from grB become nodes in grA. When a new resistance element is made (see function elemNew), the groups of both nodes are merged (and group res_cnt is incremented). Thus, two or more nodes become in one group! Note that the two groups are not merged, when both nodes are in the same group (see function nodeJoin).

23. TILES

Edges are read by *space* (see: scan/scan.c) from a set mask "gln" files. These edges split an INF tile (the INF design space) up into smaller tiles. The created tiles are given to functions enumPair(), enumTile() and clearTile(). Note that *space* deals futher only with tiles. Each tile is an entry-point to the *space* data structures. An initial created tile gets a lower-left point, a color, a zero cnt, a zero res. known bitmap, etc.

23.1 Tile: color

Each edge has a color value. Depending of from which mask the edge is, the color value is (0, 1, 2, 4, ...). Thus, at most one bit is set in the color value. On a scanline thisX position, the tiles receive from the bottom color 0 value another color after edges are found. The second time an edge with the same color is found, it must be a stop-edge, and it resets the color bit to zero. Because the tile→color mask bitmap is currently 64 bits width, *space* cannot handle more than 64 different color values (really used masks).

23.2 Tile: cnt

Each tile that goes for the first time to function enumPair(), as newerTile, receives a unique tile number ≥ 1 (the new value of the tileCnt). If the tile→cnt is 0, we know that it is a newtile, which is coming in for the first time. If this newtile has a conductor, recogE() is done to set the known bitmap. The total number of tiles visited by enumPair() is tileCnt. The tile→cnt is used by selective resistance extraction. In prePass2 and in the extrPass the tiles must get exactly the same tile→cnt numbers to be identified. Selected tiles are written in the stream "tile" with the conductor numbers set (if selected) or zero (if not selected).

23.3 Tile: known

The known field (a 32-bit resistance bitmap) is initial set to 0. If the tile is a conductor tile, only the conductor number bits of the conductors for which resistances must be extracted are set. This is depended of parameter "low_sheet_res". Low resistive conductors are not taken into account. Note that conductors with a value of zero are never taken into account.

23.4 Tile: cons

The cons field points to a array of conductor subnode pointers. A cons[cx] pointer is set to a subnode data structure, if the tile contains a conductor (with number cx ≥ 0). Note that the field is not used by optRes is TRUE. In that case the rbPoints / tlPoints fields are used.

23.5 Tile: subcont

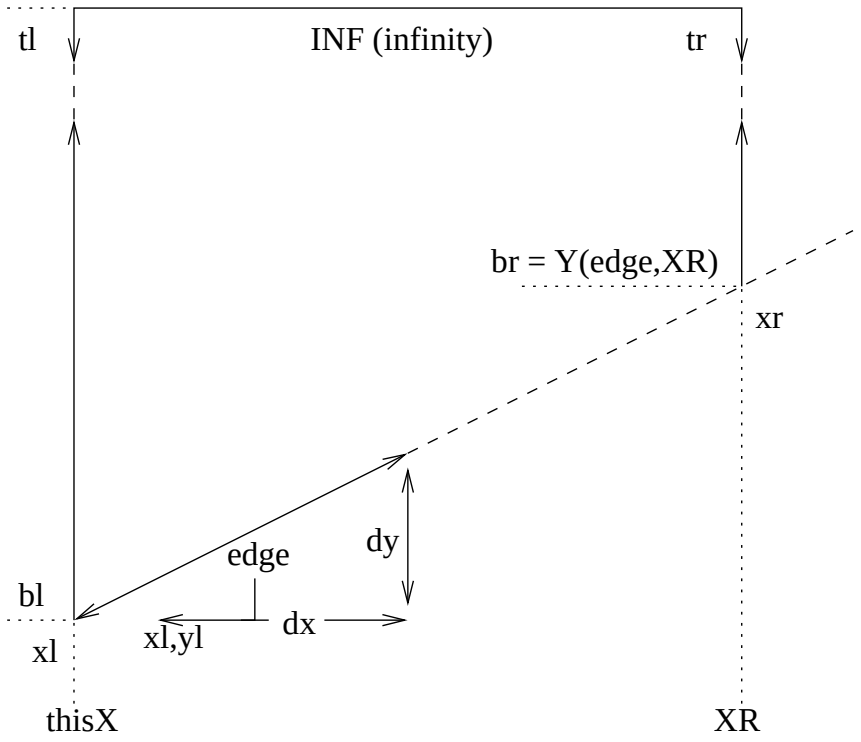
The subcont field indicates that the tile is part of a substrate contant area.

23.6 Tile: tor

The tor field indicates that the tile is part of the gate area of a transistor.

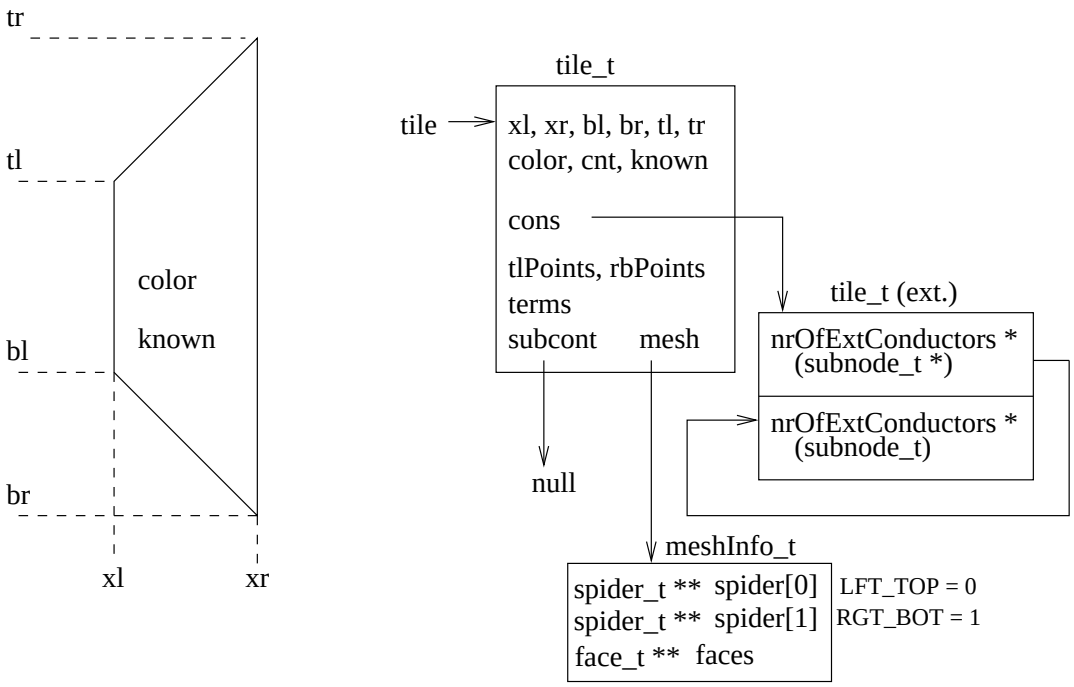
23.7 Tile Coordinates

Initial, the tile lower left coordinate point (xl,bl) is set. The lower right point (xr,br) is also set, but its real xr value is known after a call to BR(). Initial the value XR is used, which is the cell bounding box xr value (ginfo3.bbxr). Function setXr() sets this XR value in function extract() with bigbxr. This bigbxr can have only an extension (sub_ext), if hasSubmask is true. By hasSubmask, this sub_ext is default 10, but can be specified with parameter "substrate_extension" (sub_ext ≥ 1). Note that makesize can have enlarged the ginfo3.bbxr value, it cannot set a smaller ginfo3.bbxr value by shrinking the cell! The tile→br value is calculated with Y(edge,XR). There is one exception, if tile→xl becomes equal to XR, INF is taken for the (last) tile(xr,br). Tile tl and tr are initial always set to INF. Note that function TL() sets the real tl value and TR() the real tr value. The tile→cnt is set after a first call to function enumPair().



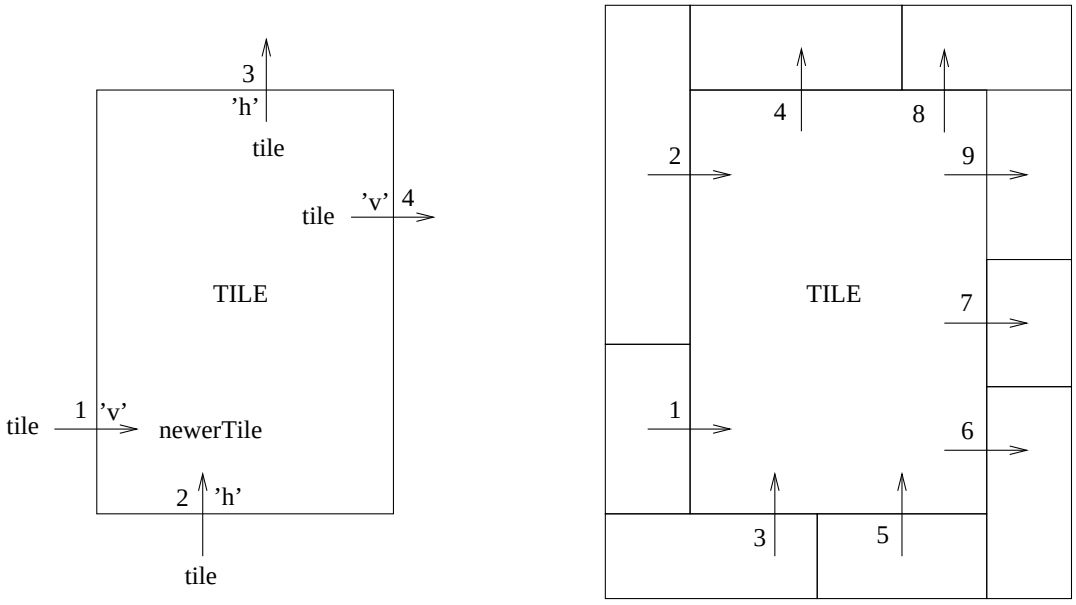
23.8 Tile Creation

Tiles are created with function createTile (see: scan/tile.c). This function returns a "tile_t" pointer. See also "include/tile.h" for the data structure specification. The function uses a free tile list to get tiles from. If the list is empty, it calls function allocTile. The size of "tile_t" is determined by its first call.



The "cons" pointers are initialized to null when optRes is 0. Note that optRes is 0 in all prepasses. Function enumPair sets with macro CONS the pointer to the subnote_t data structure. The indices of the array are conductor numbers. Only the conductor masks have conductor numbers. Note that nrOfExtConductors is equal to nrOfConductors. Only with optAccWL nrOfExtConductors may be larger.

24. VISITING THE TILE EDGES



A tile has four edges. Two vertical edges (which can be of zero length), and two non-vertical edges (which are normally horizontal). Due to the *space* scanning technique, the edges are always visited in certain order. The left vertical edge is visited first from bottom to top (always one or more times). Second, the bottom non-vertical edge is visited from left to right (one or more times). Third, the top non-vertical edge is visited from left to right (one or more times). And at last the right vertical edge is visited from bottom to top (one or more times).

Note that the real visit order depends on the scanning direction, from left to right x-direction and from bottom to top y-direction. And is depending on which tile edge is finished first (see the figure).

These edges are handled by function enumEdge. The orientation flag ('v' or 'h') tells enumEdge, when it is a vertical or non-vertical edge between two tiles. Note that for the left vertical edge and for the bottom non-vertical edge the tile is the newerTile argument to function enumEdge. By the first visit, this newerTile gets a tile number and there is done a RecogS call for finding conductors in this brand new tile.

Note that a tile is at least two times used as the newerTile in a call to enumEdge, but it is also possible that the tile is 10 times the newerTile (see the figures). Note that the same edge part (with length > 0) of a tile is never done twice!

25. TILE CONNECTION PROBLEM

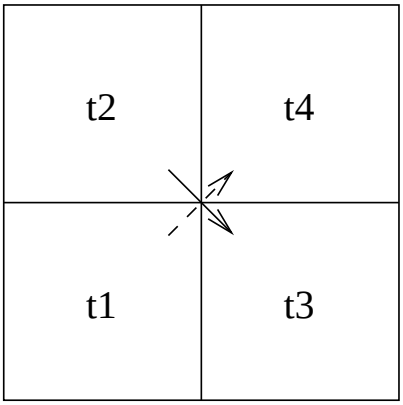


Figure A

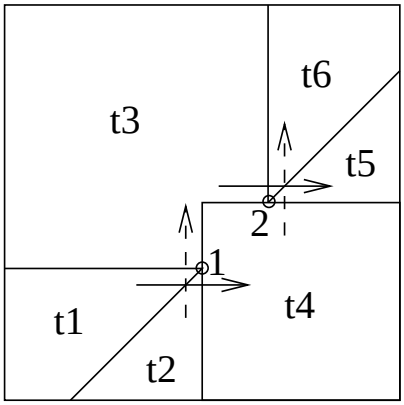


Figure B

A point connection between tiles gives always a problem. Must we decide that it is a good connection (0 Ohm) or that it is not yet a connection. *Space* scanning technique can not see all this bad connections. This can happen in orthogonal layouts, but also in non-orthogonal layouts.
I have decided to make the *space* interpretation uniform and that a point connection is not yet a connection. You must scale (grow) the layout a little to make this connections.
This implies that *space* can not detect shorts for point connections!

25.1 Orthogonal Layouts

In orthogonal layouts (see figure A) it is not possible for *space* to connect tiles t1 and t4. *Space* can connect tiles t2 and t3 together, but *space* skips this connection (because the vertical edge length is zero)!

25.2 Non-Orthogonal Layouts

In non-orthogonal layouts (see figure B) it is not possible for *space* to connect tiles t2 and t3 together. And also can *space* not connect tiles t4 and t6 together. *Space* can connect tiles t1 and t4 together (and t3 with t5), but *space* skips these connections (because the vertical edge length is zero)!

25.3 Point Terminals

When a tile begins with a lower or upper left corner (or ends with a lower or upper right corner) and its vertical edge length is zero, then it is still possible to connect a point terminal to this tile position. But note that a terminal is connected to the first tile found, which contains the same conductor as the terminal.
Thus, the terminal at position 1 (see the little circles in figure B) can possibly be connected to one of the tiles t2, t1, t3 or t4. And the terminal at position 2 possibly to one of the tiles t3, t4, t5 or t6.

26. STREAM: congeo

This stream is generated in the circuit view of the extracted cell by option **-x**. It contains connectivity geometry data, which is used for back-annotation by the program *highlay*. This stream is also generated for options **-j**, **-k** and **-s** in prePass2, for storing temporary data which is converted to the "tile" stream. Note that the net_names are not printed in prePass2, when prePass2 is not the lastPass.

Format:

```
-%d ( [net_names ...] ) %d %e %e \n
D D D  S  6xD \n
D D D  S  6xD \n
   ...
0 0 0 "0" 6x0 \n
```

The initial ASCII record contains a group number (≥ 1) with leading ’-’ character. Between parenthesis are put the net_names in the group. The %d is for the select flag (0 or 1). The other two %e values are informative, are used with option -s for resWeight and rSource.

After the initial record are written a number of tile records in packed format. These tile records represent the interconnect geometry information of the group. Each record contains a tile number (cnt > 0), a conductor number (cx ≥ 0), a mask number (≥ 0), a 64-bit integer string (color bitmap) and the 6 tile coordinates. The last record contains only zero values, to mark the end of the group data. These tiles data comes from grp→tileInfo→tiles.

After that a new group can be written. This all is done by function outGroup(). Note that the group numbers are in sequential order.

Note that the colors are converted to another color bitmap representation as is used by the tiles. Each bit is an indexed mask number, representing the presence of the mask.

27. STREAM: devgeo

This stream is generated in the circuit view of the extracted cell by option **-x**. It contains device geometry data used for back-annotation with the program *highlay* (option **-e**, **-E**). The coordinates of subcells and transistors (TOR and BJT) are written to this stream. Note that the BJT's only are written, when optRes is FALSE.
Functions outLBJT(), outVBJT(), outTransistor() and backInfoInstance() write the records.

Format:

```
-%d ( inst_name ) \n
D D D  S  6xD \n
   ...
0 0 0 "0" 6x0 \n
```

The negative number is the act_dev_cnt (≥ 1). Functions outLBJT(), outVBJT() and outTransistor() write tile records followed with a zero record.
Function backInfoInstance() does not write tile records, but only one record for each instance. This record begins with a zero and contains bounding box information. Note that the subcell instance name may have a copy index.

28. STREAM: termpos

This stream was generated in the circuit view of the extracted cell by option -x. We don't know which program was using it. It is not generated at this moment. Function outNode() did write a record for each name in the 'node→names' list.

Format:

```
%s %d %d %s %d %d %d %d \n
```

A record contains instName (or "$"), instX,Y (inst.copy), the termName (truncated), termX,Y (term.copy) and the terminal/label x,y position.

29. STREAM: tile

The "tile" stream is generated in the layout view at the end of prePass2. Function mkTileCon() converts the temporary stream "congeo" (in view circuit) into this "tile" stream.

The "tile" stream is used in the extrPass (by optRes), to select the tiles which may have resistances. See the section about selective resistance extraction.

Format:

```
<tile#> <grp#0> <grp#1> ... \n
```

The tile number is ≥ 1 and are in sequential order. Only tiles which have at least one group number (cx) > 0 are listed. In each record contains exactly nrOfConductors group numbers. Group numbers equal 0 represent conductors which are not used for resistance extraction.
Function resEnumPair() uses this information to set the conductor number in the newerTile→known bitmap.

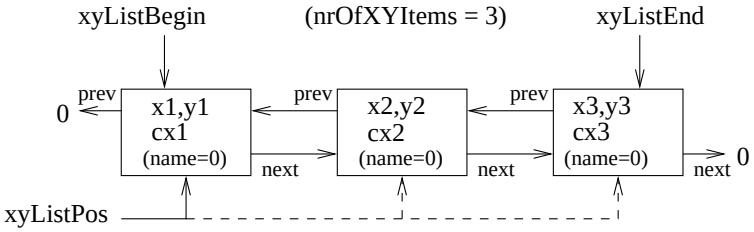
Code:

```
if (newtile) {

    if (extSelRes) selectSplit = testTileCon (tileCnt);

    elem = RecogE (newerTile -> color, tile -> color);

    for (i = 0; (el = elem[i]); i++) {
        switch (el -> type) {
            case RESELEM:
                cx = el -> s.res.con;
                ...
                if (optIntRes && el -> s.res.val > low_sheet_res) {
                    if (extSelRes) {
                        if (selectSplit && selectSplit[cx] > 0)
                            newerTile -> known |= (1 << cx);
                    }
                    else newerTile -> known |= (1 << cx);
                }
                break;
        }
    }
    ...
}
```

30. FILE: sel_con

In prePass2 by optPrick (option **-j** or **-k**) a file "sel_con" must be read from the current directory. This "sel_con" file is used for selective resistance extraction.

Each record of the "sel_con" file can contain a x,y-point and a mask-name. The mask must be a conductor mask. One record may contain also only a prick-name, which can match with a net-name. The prick-name is specified with a leading '=' sign (first character of record).



Function readXYFile() reads the "sel_con" file. The conductor mask-name is stored as a conductor number cx (≥ 0). The list is sorted with function sortXYItems(). The flag doTileXY is set. Before usage, a data struct is read with getXYItem() and stored in nextX, nextY and nextCon. Variable xyListPos points to this data struct. The point position is not yet changed.

If function testTileXY() detects that the point is laying in a tile, then the interconnect group prick flag is set (of tile→cons[nextCon]). Function freeXYItem() removes current xyListPos data from the list. The xyListPos pointer is changed and points to the next data.

Note that in enumPair('v'), in nameAdd() a group prick flag also can be set with the specified prickName (if terminal/label-name matches the prickName).

By output in prePass2, function outTileConnectivity() selects groups with a set prick flag for output. Selected groups are written to stream "congeo". Variable outPrePassGrp counts the number of selected groups for statistics.

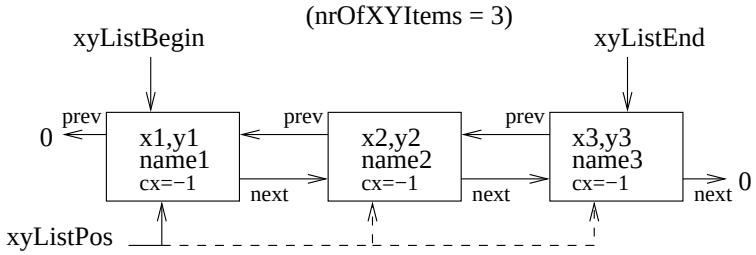
At the end of prePass2, function mkTileCon() converts stream "congeo" (in view circuit) into stream "tile" (in view layout). This stream "tile" is read in the extrPass by function resEnumPair(), by testTileCon().

31. STREAM: torname

In the extrPass by useAnnotations = TRUE (default case) a stream "torname" can be read.

The stream "torname" must be found in the extracted topcell layout view.

Note that flag useAnnotations can be set FALSE with parameter "no_labels on". Each record of this stream can contain a x,y-point and a transistor instance-name. The points and the names are stored in a XY-list by function readXYFile().



The list is sorted with function sortXYItems(). The flag doTileXY is set. The first data is read with getXYItem() and stored in nextX, nextY and nextName.

If function testTileXY() detects that the point is laying in a tile, then the tile has a tor (gate area), the tor→instName is set to the nextName. Can also be used for BJT's. If cons[cx]→pn→t is set, there is a BJT-element. By BJT's, also the instName is set to the nextName. A warning is given, if no transistor at the x,y-position is found. A warning is also given, if the instName is already set or if there are two different gates present at the x,y-position. A set instName replaces the default instance name by circuit output.

Note, that the doTileXY function is also used in prePass2 for selective resistance extraction. That both doTileXY functions are not used at the same time.

32. STREAM: wiredata

In the lastPass a stream "wiredata" can be written in the circuit view. This stream is only written for optNetInfo (option **-y** or parameter "netinformation").

Note that the *space* program must be compiled with LICENSE defined. And also be compiled with NO_NETINFO undefined.
Note that only a licensed user, which has a license file with programModel OPTEM can use this option.

See function outNetInfo(). It prints info about each net. Note that grp→tileInfo is used to store information data.

```
version 1 0                                        (a)
1 nodename1 nodename2 ...                          (b)
3.2e+3 0 0 6.8e-15 7.2e-07 8 32 144 140 0 0 0 0  (c)
 (1)  (2,3)  (4)   (5)     (6,7,8,9)   (10,11,12,13)

record (a): head record: major# minor#
record (b): all node/net names of a group
record (c): data summary of group
(1) maxRes   value in Ohm
(2) groudCap value in Farad
(3) coupCap  value in Farad
(4) totArea  value in m^2
(5) totPerim value in m
(6) xmin (internal units)
(7) ymin
(8) xmax
(9) ymax
(10) number of gates
(11) gate area
(12) number of d/s
(13) contact area
```

33. CIR_NET_ATOM_AVAILABLE

Source file "lump/out.c" use only this #ifdef.
Is defined in file "include/config.h":

```
/* CIR_NET_ATOM_AVAILABLE identifies a capability of "libddm/libcirfmt"
 * for writing a net 'structure by structure'.
 */
#if NCF_RELEASE < 400
#ifdef CIR_NET_ATOM
#define CIR_NET_ATOM_AVAILABLE
#endif
#endif
```

Note that `CIR_NET_ATOM` is defined in "libddm/dmincl.h".
Thus the file "libddm/dmincl.h" must be included before "include/config.h".
The define `CIR_NET_ATOM` is a circuit stream format number.

We have decided to remove the `CIR_NET_ATOM_AVAILABLE` #ifdef's
from the source file "lump/out.c".
Because we cannot be sure that the old code, which is only using `CIR_NET`
works correctly.

Note that for `NCF_RELEASE` $\geq$ `400` also `CIR_NET_ATOM` must
be available.
But that release is not more in use for a SPACE distribution.

34. REFERENCES

For SPACE see:

The usage of Space is described in the user manuals:

- Space Tutorial, October 2000
- Space User's Manual, September 2000
- Space3D Cap. Extraction User's Manual, October 2000
- Space Substrate Res. User's Manual, May 2000

CONTENTS