SPACE EXTRACTION APPLICATION NOTES

S. de Graaf

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

Report ET-CAS 00-01
March 6, 2000

Last revision: December 2, 2003.

1. INTRODUCTION

This report describes the *space* hierarchical layout to circuit extractor and its preprocessors, and other useful programs.

- Describes what these programs are doing.
- Describes which database files are involved.
- Describes the old space situation.
- Describes the new space situation.
- Describes the modifications done.

This work is carried out for the SPACE project.

1.1 REFERENCES

[1] N.P. van der Meijs, A.J. van Genderen, "Space User's Manual", report ET-NT 92.21, 1999.

[2] N.P. van der Meijs, A.J. van Genderen, "Space Tutorial", report ET-NT 92.22, 1999.

[3] S. de Graaf, N.P. van der Meijs, P. van der Wolf, "Data Management Interface Application Notes", report ET-NT 88.1, 1988.

[4] A. van der Hoeven, "Introduction to the NELSIS CAD Framework", report DIMES-DDTC, 1993.

2. EXTRACTION

A typical hierarchy of layout cells can look like this:

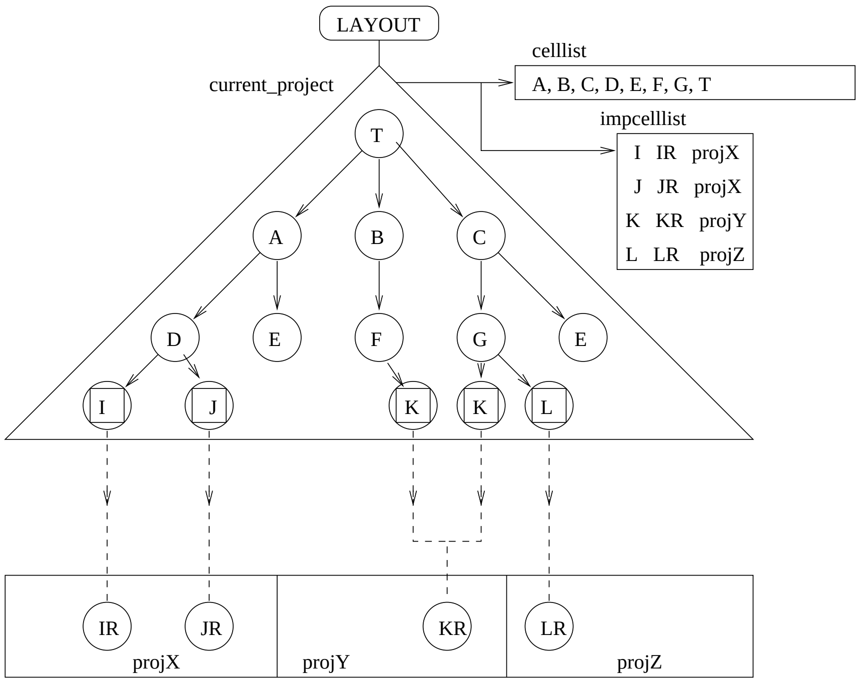


Figure 1. A typical layout hierarchy.

2.1 Hierarchical and Flat Extraction

Space has 2 extraction modes. The hierarchical mode is the default mode. Give option **-F** to use the flat extraction mode. Note that device cells are never extracted. The root cell or topcell is the cell you start with on the command-line. It is the topcell of the cell tree. You can specify more than one topcell.

2.1.1 Flat Extraction

In flat (linear) extraction mode, the topcell T is completely flattened. This means, that **all layout primitives** of all the subcells are stored in the output files of topcell T. Also of imported subcells. To do this, *makeboxl* is called without any options. Note that the layout primitives of pure device subcells are not expanded, but information of instances and terminals is stored in some special files. The flat extraction is also controlled by the macro status of subcells. If the macro status of device subcells is true, they are not more pure devices, and are also expanded (depending on the Free or Freemasks setting). If you extract cell T, *space* tries to find out if the expansion is already done. An indication for flat expansion is the existence of file "spec". If the "spec" file does not exist or when a "box" file is newer than a "gln" file, or the "gln" file does not exist, the expansion must be done. Thus, *space* cannot really detect, when expansion must be done again. It does not inspect imported subcells. Space has no option whereby you can say, please do the expansion always. Do *dbclean* to force the expansion for the cell.

2.1.2 Hierarchical Extraction

Default, in incremental mode (with depth = 1), the topcell T is always hierarchical extracted. But the subcells are only hierarchical extracted if needed. You can change the incremental depth with option **-D** *depth*. If the depth = 0, all cells are only hierarchical extracted if needed, also the specified root cells on the command-line (fully incremental). If you use depth = 2, the topcell T and the children A, B and C are always hierarchical extracted, and the other subcells (D, E, ..) only if needed. If you use depth = 3, all local cells are hierarchical extracted, because there are only 3 levels. You can also put off incremental mode with option **-I**. In that case all local cells are ’always’ hierarchical extracted. This is not completely true when you specify two or more topcells. If the topcells have a number of subcells in common, these subcells are only once extracted. The topcells are normally all extracted. Note that there is also a maximum depth option **-L** *maxdepth*, subcells which are on a level above this maximum depth are not extracted. Option **-T** is equal to option **-L 1**. With option **-T** only the topcells are extracted.

Only cells which don’t have the device status and subcells which don’t have the macro status are possible extracted.
The rules for (re-)extraction are:
- cells given on the command-line are always extracted (except devices).
- subcells are always extracted, if the subcell_level <= incr. and max. depth
  (except device or macro cells).
- other subcells (except device or macro cells) are only extracted, if the
  subcell_level <= max. depth and the following is true:
    - the cell is not yet extracted (no circuit is present), or
    - the (extracted) cell circuit is present, but is out of date, or
    - the cell circuit is present, but was extracted with another mode (X ?)
      (modes are: flat, hierarchical or pseudo hierarchical).

Note that the following items are not a rule for re-extraction:
- which technology file you use and changes in the technology file.
- options for different kind of resistance and capacitance extraction.
- changes in imported subcells are not detected (NEW: only imported top cells).
- newer expansion data is no reason for re-extraction.

When a cell is extracted, *space* tries to find out if the cell needs to be expanded.
The preprocessing programs are only run when option **-u** is not specified.
The rules for (re-)expansion are:
- no previous expansion, file "tid" not found.
- previous expansion out-of-date, cell layout/status newer.
- previous expansion was another type of expansion.
- because different resize was done.

If the technology file used specifies resizing of masks.
The following rules for doing resize are:
- because expansion is done.
- because no resize was done.

2.1.3 Extract Data

Extraction can only be done, if there is expanded layout data (expand data or gln data). This expanded layout data must, of coarse, be up-to-date. If the preprocessing (expansion) programs aborts, also the space extraction must stop. This is done by a *dbclean* operation, by removing the possibly existing expand data. Thus the extractor can not proceed.

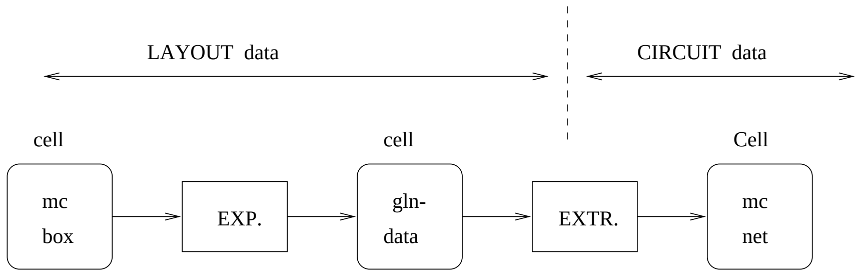


Figure 2. Extraction steps.

Of coarse, the timestamps must be correct. The timestamp of gln-data must be newer than that of the layout (stream mc) and the timestamp of the circuit-data (stream mc) must be newer than that of the gln-data. But the first test is between the circuit-data and the layout-data (mc streams). Because the first question is, must the circuit be extracted. Yes, if there is no extracted circuit or else if it is old (out-of-date), because the layout is modified or the status of a subcell is changed. The extracted circuit is a different circuit than the reference circuit. The reference circuit is the starting point for the routing of the layout and consists of all hierarchical cell connections (netlist) and nothing more. The extracted circuit can have another cell hierarchy and can contain different kind of parasitics (capacitances and resistances) depending of the specified extraction options and accurrency, and the values of these parasitics is depending of the given technology information. Note that a capital first letter is used for the name of the extracted circuit and a lowercase first letter for the name of the reference circuit. The layout can also contain device subcells. These subcells are normally not extracted. If seaching for devices, it is better not to look for a circuit with a capital first letter (an extracted circuit), but for a reference circuit.

2.1.4 Cell Hierarchy Examples

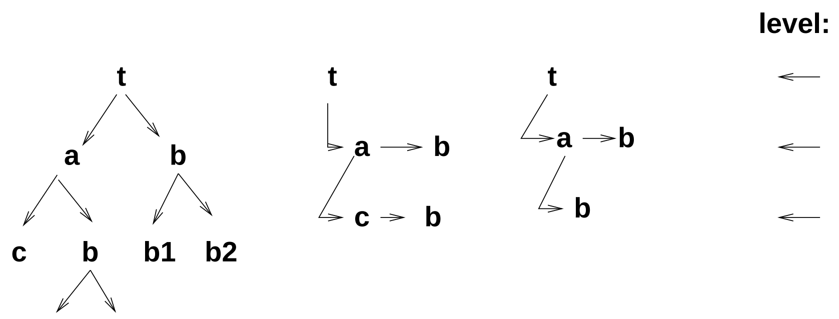


Figure 3. Cell hierarchy examples.

For the order of the extraction of the different cells, it is better to do the children first.

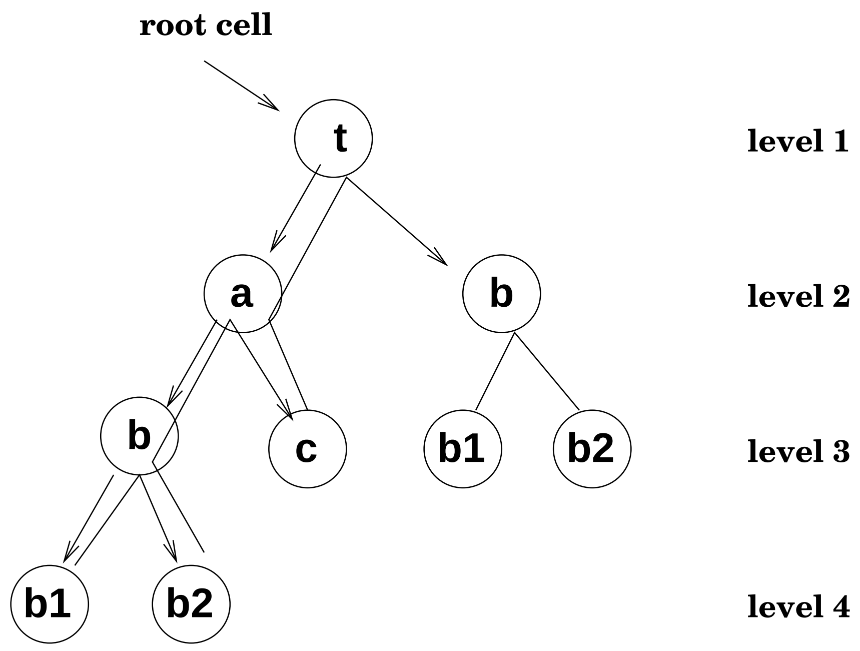


Figure 4. Cell hierarchy used for extraction.

3. FLAT EXTRACTION - THE OLD SITUATION

By flat extraction function extractTree() calls for every root cell (command-line argument) function findCandidates(). For example:

```
% space -F t
```

The root (top) cell "t" contains children "a" and "b". Subcell "a" contains children "b" and "c", and subcell "b" contains children "b1" and "b2". The subcells "c", "b1" and "b2" are leaf cells. The function findCandidates() returns a list (array) of possible candidates. By flat extraction this array contains only the root cell. If this root cell has device or macro status, it is not extracted. This root cell can be a macro cell, if this status is put off for the cell. The macro status has only meaning by usage by another cell. Function findCandidates() does two things: 1) makeTree() 2) makeCandidateArray()

3.1 Function makeTree()

This function traverse the tree of cell "t" depth first. Its makes two lists with the same Cell elements, using different next pointers. The "cells" and "candidates" lists:

```
cells      -> c(3) - b2(4) - b1(4) - b(3) - a(2) - t(1)
candidates -> t(1) - a(2) - c(3) - b(3) - b2(4) - b1(4)
```

The "cells" list is in (reverse) first found order and the "candidates" list is in (reverse) first cell ready (or finished) order. The "candidates" list is very important, because it maintains the unique (reverse) extraction order of the cells. Thus if you want to extract the cells in correct order, you must scan the candidates list. The "cells" list is used for the look-up of the cells. The makeTree() function starts at the ROOT_DEPTH (=1), and goes every time a level deeper (higher). Note that cell "b" is also used a second time on level 2. If you change the order of the MC records of cell "t", you get:

```
cells      -> c(3) - a(2) - b2(3) - b1(3) - b(2) - t(1)
candidates -> t(1) - a(2) - c(3) - b(2) - b2(3) - b1(3)
```

The layout of every unique cell in the tree is checked out, and the cellkey is stored in the Cell struct. The macro status of each cell is checked. The root_c pointer is assigned to the root cell (depth = 1) struct. The device status is only checked for the root cell, and also only for the root cell the following DO_SPACE tests are done:

```
- exist the extracted circuit?  If (NO)  DO_SPACE; else
- exist the "devmod" file?      If (YES) DEVMOD;   else
- layout/mc > circuit/mc ?      If (YES) DO_SPACE; else
- circuit diff. extracted?      If (YES) DO_SPACE;
```

Comments: The timestamps of the "macro" and "devmod" files are not tested. If caps_name mode is ON, the "devmod" file must be in the extracted circuit directory.

The following preprocessing tests are also only done for the root cell, but only if the root cell is not a device and the preprocessing option is ON:

```
    - layout diff. expanded?        If (YES) DO_EXP; else
    - layout/box > layout/gln?      If (YES) DO_EXP;
```

Note that DO_EXP also sets the DO_SPACE status for the root cell. The option optMaxDepth is also used in the test, but this option is not for flat mode. If you don't set this option, it is a very big value (INF). For subcells the following preprocessing test for the root is done:

```
    - layout/box > layout/root/gln? If (YES) DO_EXP(root);
```

Comments: All subcells in the tree are read, thus the above preprocessing test is done for all subcells, this because the device status for subcells is not set. A subcell DO_EXP setting does not set the DO_SPACE status for the root cell. Only the DO_EXP status for the root cell is set. By last resize modifications, the DO_EXP status for some subcells is set by the setting of the DO_RESIZE status. The resize tests can set the DO_EXP status for the subcell itself. The resize tests are not correct implemented for the root cell.

3.2 Function makeCandidateArray()

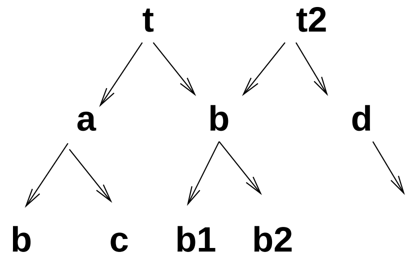
This function searchs for the root_c in the candidates list. In flat mode, this is the only cell which is placed in the array (num = 1). The depth field in the cell struct is not set (= 0), thus the test:

```
    if (c -> depth == ROOT_DEPTH) reset MACRO status;
```

fails. Thus you can't extract a macro cell in flat mode. Normally the top cell is always extracted, because the test:

```
    if (c -> depth <= optMinDepth) set DO_SPACE;
```

is always true (optMinDepth = 1). Thus all tests done in function makeTree() are useless, except for the DO_EXP setting. Comments: The DO_EXP setting is not done for a subcell. Thus subcells given on the command-line cannot be expanded. A DONE root cell is never expanded a second time, because its DO_EXP/DO_RESIZE status is reset. Device subcells are possible extracted, because the device status for subcells is unknown. Because the DONE_SPACE status is always set, the function checks out the cell again. This is not possible for macro cells, because these cells are accidental skipped (not extracted). Thus the timestamp of the layout of all (except imported) cells in the tree can set the expand status for the root cell. If a new root cell (for example "t2") contains some subcells of the previous tree, these subcells cannot set DO_EXP for the new root cell.



4. HIER EXTRACTION - THE OLD SITUATION

By hierarchical extraction function extractTree() calls for every root cell (command-line argument) function findCandidates(). For example:

```
% space t
```

The root (top) cell "t" contains children "a" and "b". Subcell "a" contains children "b" and "c", and subcell "b" contains children "b1" and "b2". The subcells "c", "b1" and "b2" are leaf cells. The function findCandidates() returns a list (array) of possible candidates. By hierarchical extraction this array can contain all cells of the tree. Some of this candidates are skipped, because they are devices or macro cells or already extracted. The candidates list order is used to place the cells in the array in correct order for extraction. Function extractTree() skips cells which have the device or macro status and also skips cells which have not the DO_SPACE status. These cells are already extracted. Note that the root cell, which is a macro, can be extracted (if not a device), because this status is put off for the root cell. Macro status has only meaning by usage by another cell. Which cells of the root tree are extracted or not, is also depended of two depths settings (optMinDepth and optMaxDepth). Cells at depth <= optMinDepth are always extracted and cells above optMaxDepth are never extracted. Note that cells between this two levels are only extracted when needed. Note that by hierarchical extraction the interface can be put in the pseudo mode. This not done by a command-line option, but by the extract definition file.
Function findCandidates() does now four things:
1) makeTree()
2) findCandidateParents()
3) determineDepths()
4) makeCandidateArray()

4.1 Function makeTree()

The makeTree() function starts with the ROOT_DEPTH (=1), and goes every time a level deeper (higher). The layout of every unique cell in the tree is checked out, and the cellkey is stored in the Cell struct. The macro status of each cell is checked. The root_c pointer is assigned to the root cell (depth == 1) struct. For the root cell and not macro subcells the device status is checked. And for these cells are also done the following DO_SPACE tests:

```
- exist the extracted circuit?  If (NO)  DO_SPACE; else
- exist the "devmod" file?      If (YES) DEVMOD;   else
- layout/mc > circuit/mc ?      If (YES) DO_SPACE; else
- circuit diff. extracted?      If (YES) DO_SPACE;
```

Comments: The timestamps of the macro/devmod files is not tested. If caps_name mode is ON, the devmod file must be in the extracted circuit directory. The device status for macro subcells is unknown.

The following preprocessing tests are also only done for these cells, but only when the cell is at lower depth than optMaxDepth, and is not a device, and the preprocessing option is ON:

```
    - layout diff. expanded?        If (YES) DO_EXP; else
    - layout/box > layout/gln?      If (YES) DO_EXP;
```

Comments: Note that only for normal subcells and the root cell the DO_EXP status can be set. The test "depth < optMaxDepth" is incorrect, must be "<=". Thus the DO_EXP status is not set for subcells at too high depth. This is strange, because the DO_SPACE status can be set, but they are not expanded. Note that DO_EXP also sets the DO_SPACE status for the cell. Note that not all subcells in the tree are read (and checked out). If a cell is a pure device (which is checked out) the tree traversal stops. Note that this is not correct for the pseudo hierarchical mode. Note that the timestamp of devices is not checked. There is NO layout vs. circuit test done for macro and device cells, and also NO layout/box vs. layout/gln test. A subcell DO_EXP setting does not set the DO_EXP or DO_SPACE status of its parents or the root cell. This kind of things is done by function findCandidateParents().

4.2 Function findCandidateParents()

This function first calls function invertHierarchy() to add parent pointers to the cells data structure. This is an inefficient method, but it works. If used a second time, it does everthing again, but adds only new parents.

Second, the function examines all cells in the candidates list. This list contains all cells found in the special cells calling order. The root cell is last put in this list, and is the first found in the list. Thus to set the DO_SPACE/DO_EXP status for parents correctly, this list must really be used in reverse order? If parents are set, this setting cannot be used to set the parents of the parents (maybe a feature?). Thus second, childs can set the status of its parents. Also the following case sets the DO_SPACE and DO_EXP status for a parent:

```
    - if the cell has a macro file, and the timestamp for parent_gln
      is older than the "is_macro" file (or if gln is missing).
```

Comments: For a cell already DONE it is not done again, and the DO_EXP status of a cell can be reset by function makeCandidateArray(). Thus it shall not work in all situations if you have more than one root cell. Thus DO_SPACE/DO_EXP of child is a reason for DO_SPACE/DO_EXP of parent.

4.3 Function determineDepths()

First the depth fields of all cells are reset to 0. After that, the cells in the root tree gets new depth values >= 1. Function bfs() is called to make a cell list with the fringe pointers. Every cell in the fringe list is scanned and its children get a new depth value (if not already set) and added to the fringe list. Comments: The fringe pointers are not reset, this can give problems, because function fappend() goes each time to the last point of the list (also inefficient).

```
    fringe list: t(1) - a(2) - b(2) - c(3) - b1(3) - b2(3)
```

4.4 Function makeCandidateArray()

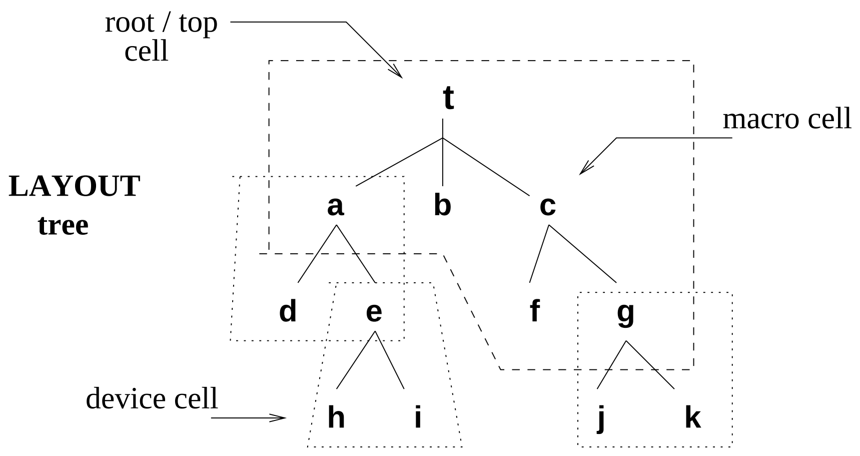
This function searchs the candidates list and shall first count the candidates. It skips cells of which the depth is not set and cells on too high depth. This depth is controlled by option optMaxDepth. If the cells are counted, it looks if the array is big enough.

The function searchs the candidates list a second time with the same constrains to place the cells in the array and does this placement in reverse order.

If the root cell (depth == 1) is a macro cell, its macro status is reset. It resets also the DO_SPACE status for every cell which is DONE before. The cells at a depth <= optMinDepth are always (re-)extracted. If a cell was DONE before, it is checked out again, but is not expanded a second time. Cells at a depth > optMinDepth and not DONE before, of which the DO_SPACE is set, are also extracted (and if requested: also expanded). Of coarse the depth of these cells is <= optMaxDepth. Thus these cells are only extracted when needed! A special case is optMinDepth=0, in that case all cells are only extracted if needed. Comments: Because the DONE_SPACE status is always set, the function checks out the cell again. This is not possible for device/macro cells, which were already skipped before. If a cell has the macro status, it does normally not have the device status, because this test is skipped. Thus cells are first skipped by function extractTree() if they have macro status and second because they have only the device status. The DO_EXP status of macro cells is normally not set, thus if these cells become root cells on the command-line the second time and are in the tree of the first root cell, they are possible not expanded.

5. WHAT IS HIERARCHICAL EXTRACTION?

Hierarchical extraction of a cell "t" means, that alone the layout elements (boxes, terminals, labels, nors) of that cell "t" are taken as input for an extraction.



Because the cell "c" has the macro status, the layout elements of this cell are added to the top cell "t". Note that some masks of a macro cell can be skipped (if specified). You can find the subcells "a", "b", "f" and "g" back as sub-circuit instances in the extracted circuit "t". In capital mode, this extracted circuit is called "T". Because there are sub-circuit instances, you name this type of extraction hierarchical. The sub-circuits must also be extracted to get a complete hierarchical tree. Some subcells contain also other subcells. And other subcells are leaf cells. And one of these leaf cells has the device status. This device layout cell "h" is not extracted. Note that the layout cell "c" with the macro status is also not extracted. Because of the macro cells the circuit tree is not the same as the layout tree. An implementation of the circuit device cell "h" already exists. Note that extracted devices out of the layout don't need to have a circuit device cell, but it can be.

Thus to get a complete circuit tree you must also extract the layout cells "a", "b", "f" and "g". If this is done hierarchical, you must also extract the layout cells "d", "i", "j" and "k". Note that a hierarchical circuit tree for cell "t" is complete, if this circuit can be simulated with a circuit simulator (like *sls* or *spice*). Thus it must contain at the bottom level only devices or functions which can be used for simulation of the logical or analog behavior of the circuit. Each circuit in the hierarchical circuit tree can be simulated separately. This is not possible, if you do a flat extraction of the layout of cell "t", because a flat extracted circuit is not hierarchical: thus it contains no sub-circuits, but only device cells and extracted devices. Of coarse you can extract all layout cells flat and simulate them.

5.1 Hierarchical Extraction of a Cell Tree!

Because a lot of cells must be hierarchical extracted, it should be nice to do this automatically, when you extract the top cell "t" hierarchically. If the cells are extracted only if needed, then this is called the incremental hierarchical operation mode. The default for *space* is, that the top cell is always extracted, and all other cells only if needed (min_depth = 1). But you can make it completely incremental or only incremental above a higher level in the cell hierarchy. Thus you can also make it completely no-incremental. Which means, that all cells are hierarchical extracted. There is also an option **-L** for the specification of a maximum level. Cells above this level are not extracted. See *space* options **-D**, **-I**, **-L** and **-T** for this fine tuning.

5.2 When is Hierarchical Extraction Needed?

The answer seems to be very simple, because if there is no extraction done yet (there is no extracted circuit), the layout cell must be extracted to get one. But when there is an extracted circuit, is re-extraction then needed? Well, when the extracted circuit is out-of-date, yes it is. And this is the case, when you have changed something in the layout, what results in different expansion data. And you know, this expansion data is used for doing the extraction. Note that, when you change the status of a subcell, this results also in different expansion data. Because you are doing a hierarchical extraction, the changes don't need to result in the re-extraction of all cells. Because the influence can be locally in the cell tree. Note that by a flat extraction, you must always re-extract, because every change is a change at the top level. Thus you see, incremental hierarchical extraction is faster, because only the parts which are changed are re-extracted. Thus you can have an up-to-date extracted circuit tree. Note that hierarchical extraction is not needed, in case the existing extracted circuit is extracted with another extraction mode (i.e. pseudo hierarchical or flat). We also don't know, that the existing circuit was extracted with other options. And what, if you use a modified or different technology file? Well, in that case you must know what you are doing. Normally, in that case you must re-extract everything.

New in *space* is, that you now can have a hybrid extraction (circuit) tree. And that the extraction process is also controlled by the existence of up-to-date flat sub-trees. The subcells in these trees are not more consulted for extraction. The general policy however is, that out-of-date extracted circuits are replaced by hierarchical ones.

The extractor use the timestamps of some files in the database. Don't touch the files or don't make a copy of the database, because after that the timestamps can be incorrect. But if you know, how the extractor is using the timestamps, maybe you can control the extraction process by touching some files. You can also use the *dbclean* program, to remove the expand data of a cell. This force not directly the extraction, but if extraction must be done, then it is certain that also the expansion is done again. An important notice about this is, that you must know that the extractor not always completely can see that there is something changed in the layout cell tree and that a new expansion therefor must be done. Note that, when incorrect expansion data is used, the extraction result is also incorrect (and misleading). Why does space not always do the expansion? Answer, because it cost time.

5.3 What is Pseudo Hierarchical Extraction?

Well, pseudo hierarchical extraction is a kind of hierarchical extraction which can give different results. This option can’t be chosen from the *space* command-line, but is a parameter (expand_connectivity) which can be put "on" in the *space* definition file.

It can give better results, when routing thru subcells is done. It costs more expansion time (and disk data), because the layout is completely expanded. The interconnect masks of the subcells are also copied to the ’bb’ mask. In combination with this ’bb’ mask, the interconnect which is laying in the subcells can be separated from the interconnect of the top cell. To use this method, you need a modified technology file for the extraction process.

5.4 Selective Pseudo Hierarchical Extraction

In hierarchical extraction mode it is possible, on bases of the status of a cell, to decide of the subcells must be pseudo hierarchical expanded.

6. DISCUSSION OF CHANGES IN THE SOURCE FILES

6.1 What is changed in sp_main.c?

See the comments for the SCCS delta’s 4.12 to 4.14 in appendix E.

- Changed the Copyright text 1999 into 2000.
- Added my name to the list.
- Removed the duplicate assignment of optResMesh.
- Point 1 in function extractTree(), see appendix C: Changed the order of the tests for ISMACRO and DEVMOD, because in the old version the macro status of a cell does not set the device status for that cell. Thus space says to the user that it skippes the cell, because it has macro status.
- Point 2 in function extractTree(), see appendix C: We now use the ’circuitName’ to check out the circuit for the first time. To do that, the name is reset in extractTree() to a null string. Because function extract() can also be called by prepass() twice. When the ’circuitName’ is null, extract() makes a new ’circuitName’ and shall check out this ’circuitName’ only the first time called. No new memory needs to be allocated for the used modulename. The new ’circuitName’ is the new modulename. Function extract() was also using cktName() to set the first character of the name to uppercase (if needed). This is now directly done with ’circuitName’.
- Point 3 in function extractTree(), see appendix C: There are two new done bits (see also determ.h/.c). In hierarchical mode the space program can tell the user what for type extraction is already done. The normal "already extracted" message is given for the mode you are using.
- The call to _dmAddCellEquivalence() is changed for release 4. The old added equivalence between LAYOUT/modulename and CIRCUIT/modulename looks to be incorrect, must be between cellname <-> circuitName.
- Some of the used buffers are enlarged (were possibly too small).
- Function extract(), expand() and processgln(): Now function strchr() is used to search for the ’#’ character.
- Function expand() and processgln(): The layout versionnumber (release 4) is not more stored in the dObject. Now is looked in the layoutKey for the versionnumber (see determ.c).

6.2 What is changed in determ.h?

See the comments for the SCCS delta’s 4.10 and 4.11 in appendix E.

- Removed the versionnumber out of struct design_object (do_t). This versionnumber is only needed for release 4 and can be found by the lkey.

- Removed the __cplusplus declaration of function cktName(). This function is not more used by "sp_main.c" function extract().
- Added modes DONE_FLAT/DONE_PSEUDO. This new modes are used by "sp_main.c" and "determ.c".

6.3 What is changed in determ.c?

See the comments for the SCCS delta’s 4.38 to 4.41 in appendix E.

- The source code looks to be completely rewritten. This is not realy true, but there have been a lot of modifications. First of all, a great number of functions is removed. This makes the code easier to read and more simple. Function findCandidates() calls now only two functions (see appendix D). A complete list of these removals is:

— addChild(), code is added to makeTree().
— newCell(), code is added to makeTree().
— lookupCell(), code is added to makeTree(). Lookup was not needed for determineDepths().
— reportCell(), code is added to makeTree().
— addParent(), see invertHierarchy().
— invertHierarchy(), see findCandidateParents(). We don’t use parent pointers, we don’t need to add them.
— findCandidateParents(), we use another strategy, see makeCandidateArray().
— fappend(), code can be added to bfs().
— fdelete(), code can be added to bfs().
— bfs(), code added to determineDepths().
— determineDepths(), code added to makeCandidateArray().
— compareDmStreamDate(), now not more used.
— compareGlnDate(), now not more used.
— cktName(), code is added where needed.

- There are also three new functions added (see appendix D):

— traverse(), used by makeCandidateArray() calculates the last modified time correctly for hierarchical mode. By flat it must go deeper in the tree.
— dmGetImpCell(), used by makeTree() gets an imported celllist structure for the imported cell (alias) name. Returns the real cell name and the project path.
— existCircuit(), used by makeCandidateArray() looks if an extracted circuit exist for the layout cell and checks this circuit out for usage (returns the cell key).

- The DRIVER part is also updated and can be used again for debugging and testing. The DRIVER has new options, which are compatible with the space program.
- Other stream names are used for testing the timestamps. For example, stream "mc" in place of stream "box".

- Now another strategy is used to decide when extraction is needed. The parents are not more extracted, because a child is extracted! In hierarchical mode the type of the extraction may be of different type! The timestamps of "devmod" files, the layout of devices and imported cells is now also used for extract detection. The device status is always tested. There is not looked for a "devmod" file in an extracted circuit directory, when uppercase mode is used.
- The Cell structures does not more contain versionnumbers and parents pointers. Memory is now allocated for the cell name (no static buffer of 256 bytes). There are now two timestamps stored in each Cell structure (lmtime and smtime).
- The code is made faster, because the lists are not unneeded inspected anymore. But other things cost more time, such as the visit of imported cells.
- A number of bugs and pitfalls is repaired. A second checkout for the layout cell does not happen again. A macro cell can now be flat extracted. A better test for resize.

7. EXTRACTION - THE NEW SITUATION

Well, the DO_EXP and DO_SPACE status of a cell is not more set by function makeTree(). The root_c pointer is returned by makeTree() and is only needed in makeCandidateArray(). Function makeTree() checks out all layout cells in a tree and determines the macro and device status for every cell. Every cell has two timestamps. The timestamp for the layout (lmtime) is taken from the "mc" stream and the timestamp for the cell status (smtime) is taken from both the "is_macro" file and the "devmod" file (if existing). The lmtime and smtime of the child is used by the parent cell in combination with its own lmtime. Notes that the cell traversal stops, when the cell is a pure device and when not running in pseudo hierarchical mode (this cell is checked in again). The traversal also stops, when the cell is an imported cell (this cell is also checked in). This cell is not placed in the candidates list and its depth value is set to -1, thus it is also never placed in the fringe list. When running in flat or pseudo mode, the timestamps are directly used for the parent cells, thus the top of the tree has always the most latest timestamp. In normal hierarchical mode the timestamp of a cell is computed with function traverse().

Note that a root cell in flat mode shall always be extracted, see makeCandidateArray() in appendix D (except when it is a device). The DO_SPACE status of the candidate cell is set, when extraction must be done. Note that the cells with a depth setting > 0 are the ones, which are placed in the fringe list, and only this candidates are given back in the candidates array. The DO_SPACE status and the depth value must be reset in the cell structure, else it cannot be used a second time (in another root tree).

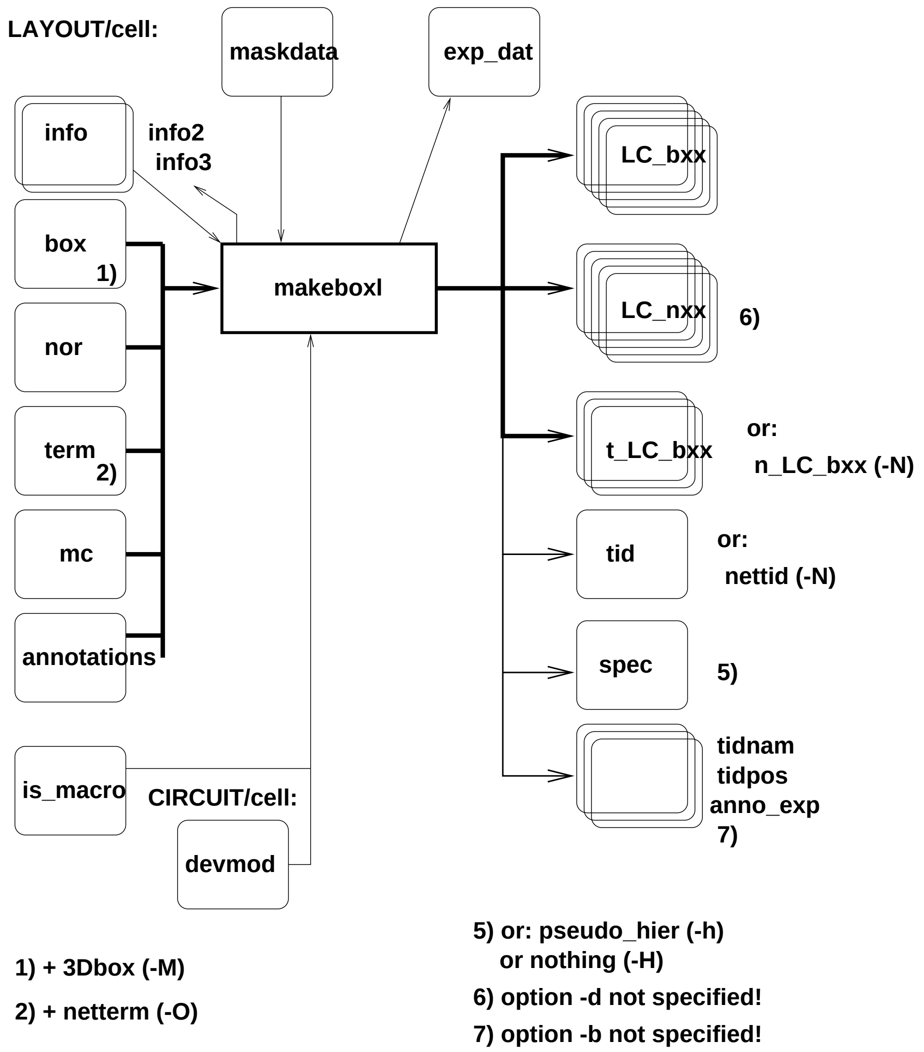
When a cell must be extracted and it is the first time (the DONE_SPACE status is not yet set), then is checked of preprocessing is needed. Note that preprocessing is never done a second time, there is no option to do that. There is also no option to skip the checking and do the preprocessing in any case. You can force that by removing the "gln" files.

Note that the status of the "gln" files is never a reason for extraction.

The *Xspace* program is also using the findCandidates() interface (only for flat mode). *Xspace* can only extract one cell if optMinDepth is set to 1. And when used in hierarchical mode optMaxDepth must also be set to 1 (option **-T**). Note that the interface reads only ones the cell information out of a tree. Thus, don't change the status (or the layout) of a cell while running this interactive program.

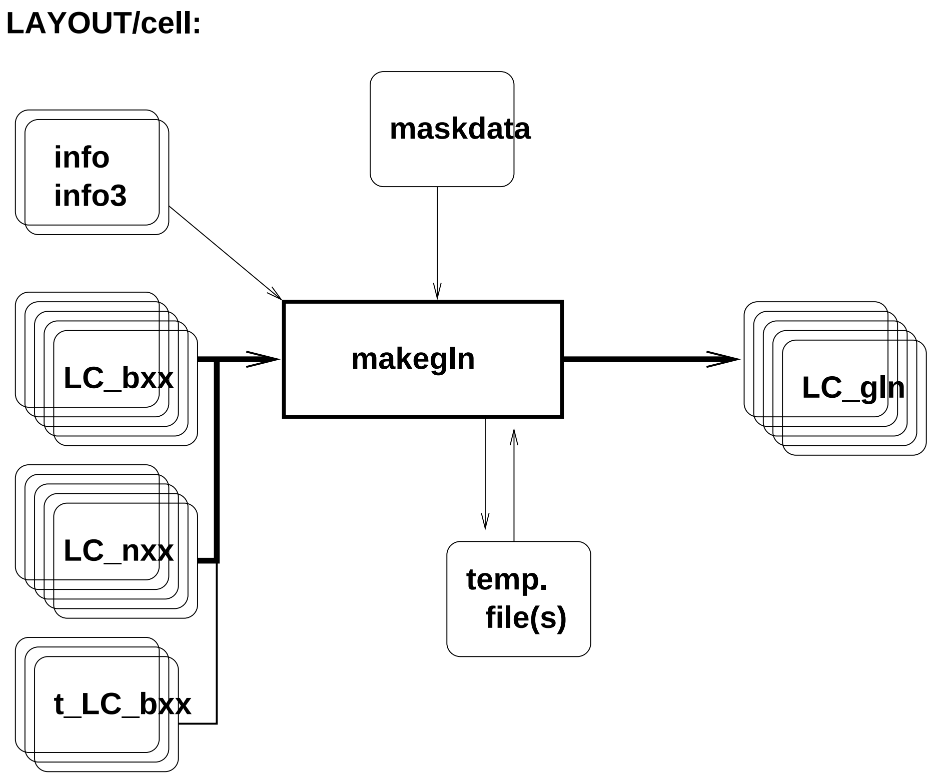
8. PROGRAM makeboxl

The program space starts the program *makeboxl* for the preprocessing phase 1 of an extracted cell. The *makeboxl* program expands the primary data of the layout cell hierarchy to secondary data (boxes and other data) by the top cell. Space runs *makeboxl* without any options in case of flat expansion. *Space* uses the option **-h** in case of pseudo hier_mode and **-H** for normal hierarchical mode. The program *space* starts the program *makeboxl* always in combination with the program *makegln* (if the cell has the DO_EXP status) as one operation.



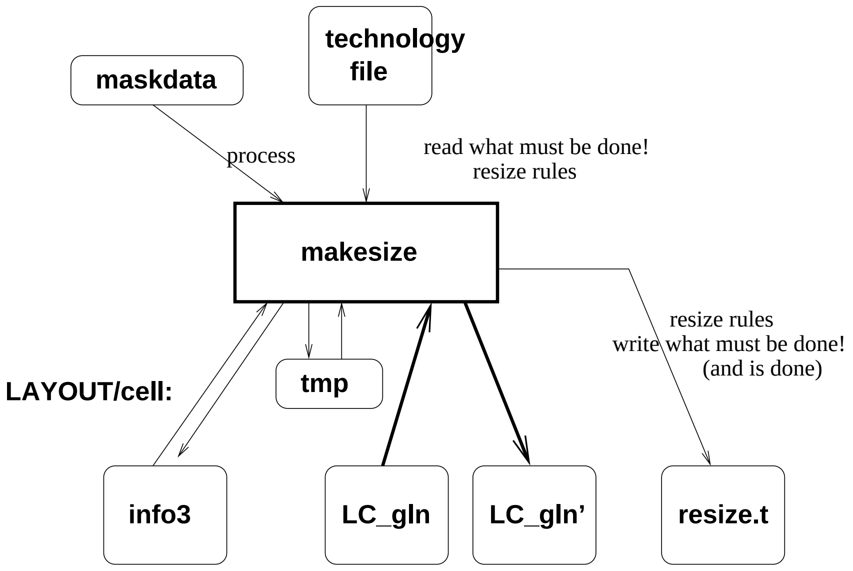
9. PROGRAM makegln

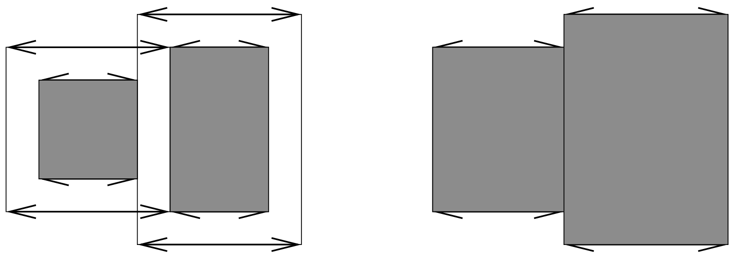
The program *space* starts the program *makegln* for the preprocessing phase 2 of an extracted layout cell, to remove overlap and to make the "gln" data-set. By *makegln* only the option **-D** is given, to remove the intermediate "bxx" and "nxx" files. The program *makegln* is always directly run after the program *makeboxl*.



10. PROGRAM makesize

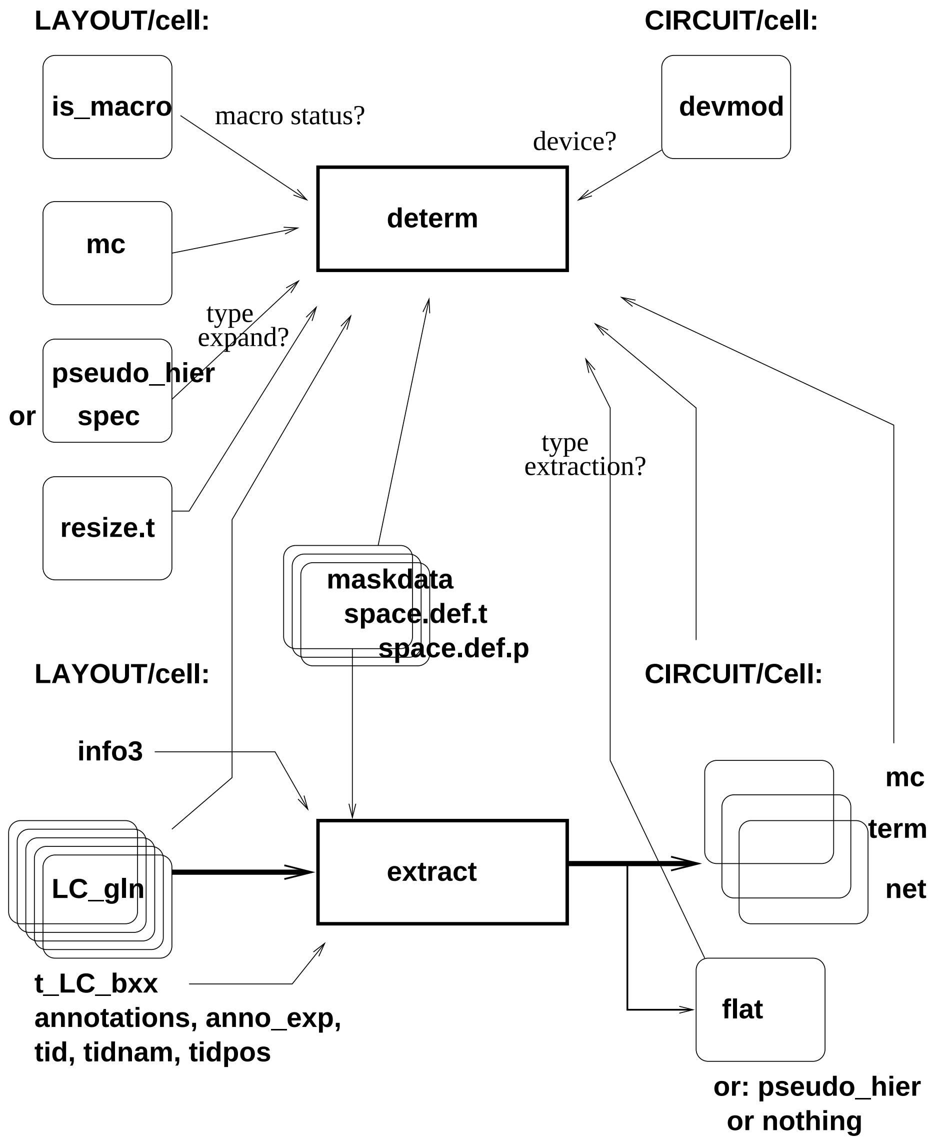
The program *space* starts the program *makesize* after the *makegln* step, if the cell has the DO_RESIZE status. The program resizes the "gln" files as requested by the specified technology file on the command-line (with option **-E**). A positive resize (grow) is quiet simple. By a negative resize (shrink) the "gln" mask must first be inverted. The program *makesize* does an extract scan on the "gln" mask stream and does overlap removal, it creates a new "gln" mask stream. The file "resize.t" tells what for resizes are done to the "gln" files.





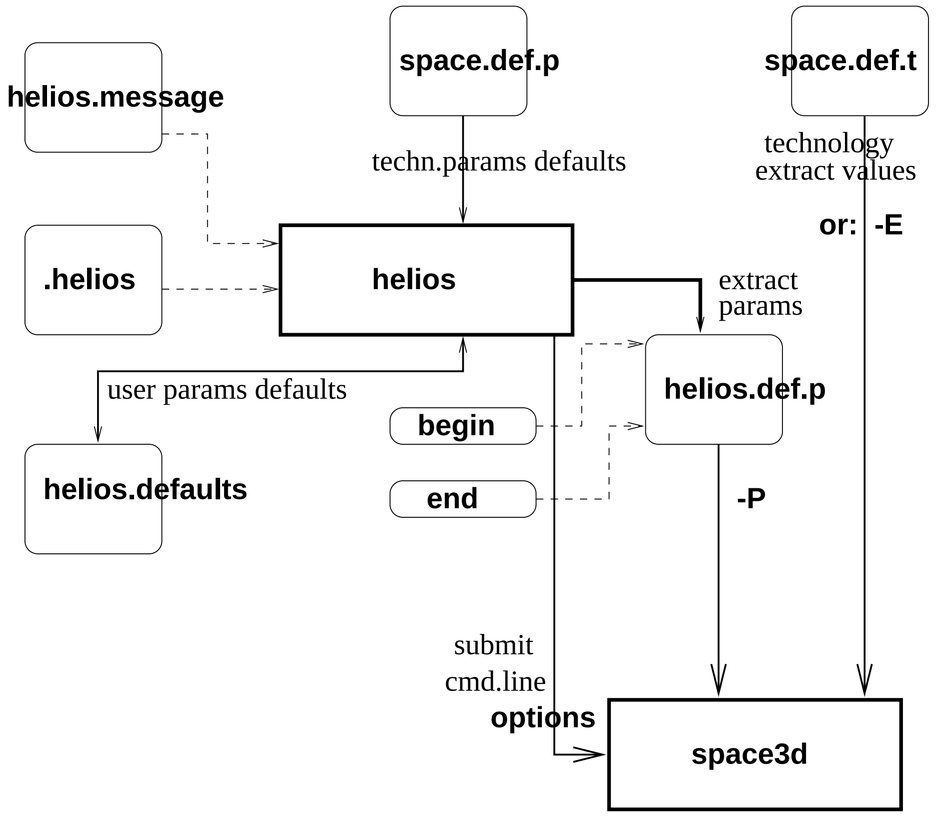
11. PROGRAM space

The module "determ" decides what must be done. The module "extract" does the real layout to circuit extraction after the preprocessing steps are done. There is also a *space3d* version and a *Xspace* version of the *space* program.



12. PROGRAM helios

The program *helios* is a graphical user interface for the space system. You can start a lot of tools with it. But first of all, it is used for running the layout to circuit extractor. The user don't need to know the space options, but can fill in the forms and choices the extraction mode and what must be extracted and the accurrency of the extraction. The user can save the parameters used in a file "helios.defaults". By opening a project, *helios* reads the technology default parameters and the user defaults. *Helios* makes it own parameter file "helios.def.p" for running the extractor *space3d* and gives also the options, which must be specified on the command-line to the extractor. When you start *helios,* it reads its setup from a file ".helios" in your home directory.



13. PROGRAM dbclean

The program *dbclean* knowns what must be cleaned. It removes all secondary data from the layout cell directory (all preprocessing data). The primary layout data of a cell must never be removed. Note that the "annotations" stream and the "is_macro" file are now also primary layout data / information. The timestamps of the "mc" stream and "is_macro" file are used as last modified time of the layout data and layout status. This means, that the other layout data streams may not be modified without touching the "mc" stream. The "info2" and "info3" parts of the "info" stream are also secondary data. The program *dbcat* can be used to list the contents of primary and secondary data. It is for packed (binary) streams the only way to inspect their contents.

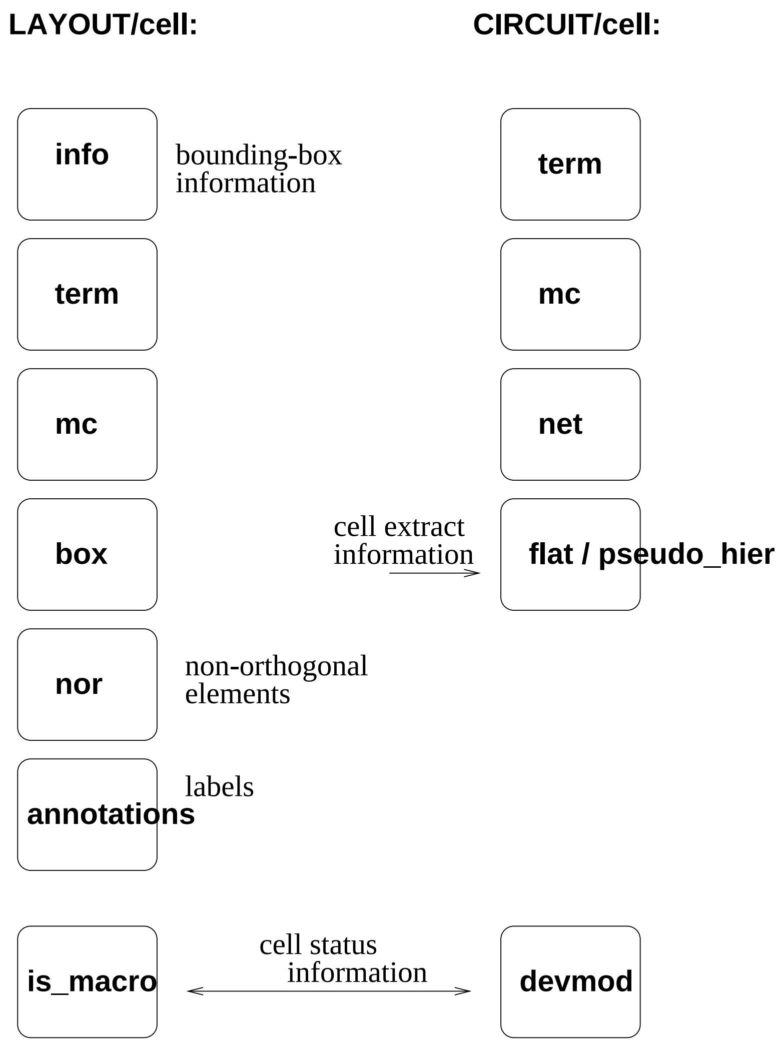
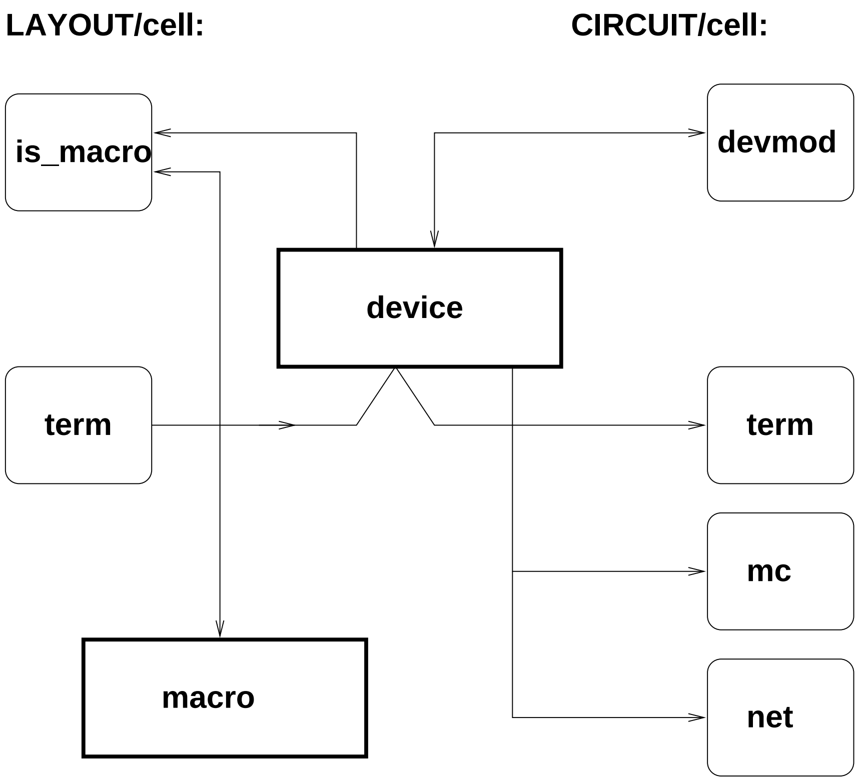


Figure 5. The database primary data files.

14. PROGRAM device

The *device* program sets and unsets the device status for a cell. If the circuit cell directory does not exist, it reads the layout cell "term" stream and writes the "term", "mc" and "net" streams. The "mc" and "net" streams are empty. It sets the device status by writing an empty "devmod" file. It unsets the device status by removing the "devmod" file and touching the "is_macro" file (because a timestamp for changing cell status must be known). Note that the timestamp of an existing "devmod" file is also used. Note that the *putdevmod* program writes a non-empty "devmod" file!



15. PROGRAM macro

The *macro* program sets and unsets the macro status for a cell. It can also sets the used masks for this macro cell. Because of the timestamp of the cell status the "is_macro" file must never be removed (else no status is ever set before).

16. PROGRAM dminst

The program *dminst* shall check all parents of a given cell. The given cell is expected to be changed/modified and that's the reason why the parents must be checked/updated, to re-install the cell in the old tree.

For a circuit view cell only a check is done, nothing is changed. For the "term" stream of the cell its parents "net" and "mc" streams are checked. Maybe there is added and/or removed a terminal, who knows?

For a layout view cell not only a check is done, but the bounding boxes of the parents "mc" and "info" streams can be updated (if needed).

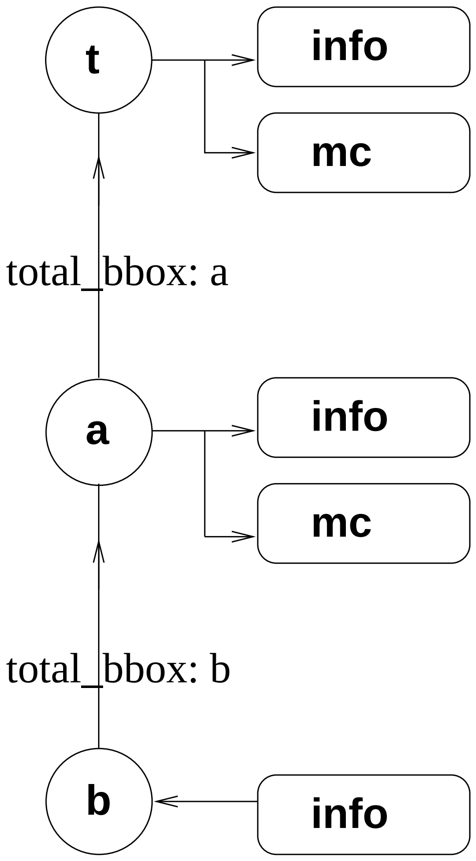


Figure 6. Install cell b (dminst b).

17. RELEASE 4 ISSUES

The *space* program can also be used in a NELSIS release 4 environment. The correct working in such an environment is not yet tested. Because of the different cell versions things are more complicated. Notes:

```
- There is always one WORKING version of the cell.
- The WORKING version is the latest and default version.
- There can also be one ACTUAL version of the cell.
- The ACTUAL version is finished, ready for usage.
- There can also be one or more BACKUP versions.
- All versions have a version number.
- There can be also DERIVED versions.
- A DERIVED version is a special version created by some tool,
  and is computed out of one of more other objects.
  For example an extracted circuit is a DERIVED version.
```

What we call a cell in release 3, is called a module in release 4 (see reference 4). A module is the collection of all design objects that have the same name and the same viewtype. The cells in release 4 are not stored in directories under their names, but in directories with a DO#.

If you checkout a WORKING version with a name and version number, you can get possible the ACTUAL version or one of the BACKUP versions. If you checkout a WORKING version with only a name, you get the latest version. This is most likely the WORKING version, but if no WORKING version exist, it can be the ACTUAL version.

A design object can instantiate another design object, this can be the WORKING, ACTUAL or a BACKUP version of that design object. New instantiates are not made to BACKUP versions. ACTUAL design objects are the appropriate ones to be used in layout design trees, thus WORKING layout cells must instantiate only ACTUAL layout cells. Also imported cells are always ACTUAL versions.

A WORKING version changes not automatically into an ACTUAL version. You must use the dsui or instcell tool to do so. If you replace an ACTUAL version, it becomes a (new) BACKUP version. And thus, you can replace an ACTUAL version in a tree by the WORKING version.

You can ATTACH so called secondary data to a design object with version status WORKING, ACTUAL, etc. A design object contains always primary data. Expand data is for example secondary data. Other cell checkout modes are:

```
READONLY - only use an existing version
CREATE - new version without reading from prev. version
UPDATE - existing and/or create new version
```

Note that CREATE and UPDATE may only be used for WORKING or DERIVED.

APPENDIX A -- OLD SOURCE FILE: sp_main.c

```
main (int argc, char **argv)
{
    if (optFlat && optMinDepth != ROOT_DEPTH) {
        say ("Options -D and -I are ignored with -F");
        optMinDepth = ROOT_DEPTH;
    }

    i = 0;
    while (cellNames[i]) extractTree (cellNames[i++]);
    ...
}

void extractTree (char * cellName)
{
    do_t * list = findCandidates (dmproject, cellName);

    if (!optFlat) PR ("extract hierarchy of %s\n", cellName);

    for (i = 0; list && list[i].name; i++) {
        if (list[i].status & DEVMOD)
            PR ("%s skipped, device model present\n", list[i].name);
        else if (list[i].status & ISMACRO)
            PR ("%s skipped, has macro status\n", list[i].name);
        else if (list[i].status & DO_SPACE) {
            if (alsoPrePass) {
                if (((optResMesh || substrRes) && Step == 0) || Step == 1) {
                    if (!(optSubRes && paramLookupB ("use_subres", "off"))) {
                        prePass1 = TRUE;
                        prepass (list + i); /* does dmCheckIn layout */
                        if ((optResMesh || substrRes) && Step == 0)
                            processgln (list + i);
                    }
                }
                if (((optSelectiveRes || optPrick) && Step == 0) || Step == 2) {
                    prePass2 = TRUE;
                    prepass (list + i); /* does dmCheckIn layout */
                }
            }
            if ((!optOnlyPrePass && Step == 0) || Step == 3)
                extract (list + i); /* does dmCheckIn layout */
        }
        else
            PR ("%s already extracted\n", list[i].name);
    }
}

void extract (do_t * dObject)
{
    if ((dObject -> status & DO_EXP || dObject -> status & DO_RESIZE)
        && Step == 0 && (!alsoPrePass || prePass1 || (!optResMesh && prePass2))) {

        expand (dObject);
    }
    ...
    /* do EXTRACT   and...   dmCheckIn (layoutKey) */
}
```

```
void expand (do_t * dObject)  /* cellname = dObject -> name */
{
    if (dObject -> status & DO_EXP) {
        PR ("preprocessing %s (phase 1%s)\n", cellname, optFlat? "- flattening":"");
        run (MKBOX, cellname, optFlat ? "" : (optPseudoHier? "-h" : "-H"));

        PR ("preprocessing %s (phase 2 - removing overlap)\n", cellname);
        run (MKGLN, cellname, delete_bxx? "-D" : "");
    }
    if (dObject -> status & DO_RESIZE) {
        PR ("resizing masks for %s\n", cellname);
        run (MKSIZE, cellname, mprintf ("-E%s", usedTechFile));
    }
}

void processgln (do_t * dObject)  /* cellname = dObject -> name */
{
    if (optResMesh) {
        PR ("refining mesh for %s\n", cellname);
        run (MKMESH, cellname, NULL);
    }
    if (optSimpleSubRes) {
        PR ("computing substrate effects for %s\n", cellname);
        run (MKDELA, cellname, goptDrawDelaunay? "-wE": "-E");
    }
}

NOTE:
       PR ("refining mesh for %s\n", cellname);
is
       verbose ("refining mesh for %s\n", cellname);



NOTE:
       PRINT ("no gln file or box file is newer");
is
       Debug (fprintf (stderr, "no gln file or box file is newer\n"));
```

APPENDIX B -- OLD SOURCE FILE: determ.c

```
extern int nrOfResizes;
extern resizeData_t *resizes;

struct Cell {
    char name[DM_MAXNAME + 1];
    struct Cell *next;         /* all cells seen so far */
    struct adj  *children;
    struct adj  *parents;
    struct Cell *fringe;
    struct Cell *candidates;   /* extraction candidates in dfs order */
    int depth;
    unsigned int status;
    DM_CELL *lkey;
};

struct adj {
    struct Cell *cell;
    struct adj  *next;
};

static DM_PROJECT *dmproject;

static struct Cell *candidates;
static struct Cell *cells;
static struct Cell *root_c;

/* General strategy:
 * 1. Make a dag with root at the top.
 * 2. Determine depth (distance to the root) of each cell.
 * 3. (Only when R3)
 *    Also handle the parents of each cell to be dealt with.
 * 4. Make a linked list in correct order for extraction,
 *    and return the list.
 */
do_t * findCandidates (DM_PROJECT *project, char *root)
{
    dmproject = project;

    (void) makeTree (root, ROOT_DEPTH);

    if (!optFlat) {
        findCandidateParents ();
        determineDepths ();
    }

    return makeCandidateArray ();
}

struct Cell * makeTree (char *name, int depth)
{
    struct adj *a;
    struct Cell *c;
    DM_STREAM *dsp;
    DM_CELL   *lkey, *ckey;
    int flag;
    for (c = cells; c && strcmp (c -> name, name); c = c -> next);

    if (c) {
        if (depth == ROOT_DEPTH) root_c = c;
        return (c);
    }
    else {
        c = NEW (struct Cell, 1);
        if (depth == ROOT_DEPTH) root_c = c;

        strcpy (c -> name, name);
        c -> children = NULL;
        c -> parents = NULL;
        c -> status = 0;
        c -> fringe = NULL;
        c -> depth = 0;
        c -> candidates = NULL;
        c -> lkey = NULL;

        c -> next = cells;
        cells = c;
    }

    lkey = c -> lkey = dmCheckOut (dmproject, name,
                            WORKING, DONTCARE, LAYOUT, ATTACH);

    if (existDmStream (lkey, "is_macro")) {
        dsp = dmOpenStream (lkey, "is_macro", "r");
        if (fscanf (dsp -> dmfp, "%d", &flag) > 0 && flag == 1) {
            c -> status |= ISMACRO;
        }
        dmCloseStream (dsp, COMPLETE);
    }

    if (!((c -> status & ISMACRO || optFlat) && depth != ROOT_DEPTH)) {

        if ((int) dmGetMetaDesignData
            (EXISTCELL, dmproject, cktName(name), CIRCUIT)) {

            ckey = dmCheckOut (dmproject, cktName (name), ACTUAL,
                            DONTCARE, CIRCUIT, READONLY);

            if (existDmStream (ckey, "devmod")) {
                c -> status |= DEVMOD;   /* device model present */
            }

            if (!(c -> status & DEVMOD)) {
                if (compareDmStreamDate (lkey, "mc", ckey, "mc") < 0) {
                    c -> status |= DO_SPACE; /* layout is newer */
                }
                else if (optFlat) {
                    if (!existDmStream (ckey, "flat")) {
                        c -> status |= DO_SPACE; /* prev was !flat */
                    }
                }
                else { /* hierachical */
                    if (existDmStream (ckey, "flat")) {
                        c -> status |= DO_SPACE; /* prev was flat */
                    }
                    else if (optPseudoHier) {
                        if (!existDmStream (ckey, "pseudo_hier")) {
                            c -> status |= DO_SPACE; /* prev was hier */
                        }
                    }
                    else {
                        if (existDmStream (ckey, "pseudo_hier")) {
                            c -> status |= DO_SPACE; /* prev was pseudo */
                        }
                    }
                }
            }

            dmCheckIn (ckey, COMPLETE);
        }
        else {
            c -> status |= DO_SPACE; /* no circuit present */
        }
    }


    if (optNoPrepro == FALSE && !(c -> status & DEVMOD) && depth < optMaxDepth
        && !((c -> status & ISMACRO || optFlat) && depth != ROOT_DEPTH)) {

        if (optFlat == TRUE && !existDmStream (lkey, "spec")) {
            PRINT ("no previous exp or hierarchical");
            c -> status |= DO_EXP;
        }
        else if (optFlat == FALSE && existDmStream (lkey, "spec")) {
            PRINT ("previous exp was linear");
            c -> status |= DO_EXP;
        }
        else if (optFlat == FALSE) {
            if (!optPseudoHier && existDmStream (lkey, "pseudo_hier")) {
                PRINT ("previous exp was pseudo-hierachical");
                c -> status |= DO_EXP;
            }
            if (optPseudoHier && !existDmStream (lkey, "pseudo_hier")) {
                PRINT ("previous exp was not pseudo-hierachical");
                c -> status |= DO_EXP;
            }
        }

        if (!(c -> status & DO_EXP)) {
            if (compareGlnDate (lkey, "box", lkey) <= 0) {
                PRINT ("no gln file or box file is newer");
                c -> status |= DO_EXP;
            }
        }
    }
    else if (optFlat && depth != ROOT_DEPTH) {
        if (compareGlnDate (lkey, "box", root_c -> lkey) <= 0) {
            PRINT ("no gln file for root or box file of child is newer");
            root_c -> status |= DO_EXP;
        }
    }
    if (existDmStream (lkey, "resize.t")) {
        if (!resizeUpToDate (lkey)) {
            c -> status |= DO_EXP;
            c -> status |= DO_RESIZE;
        }
        else if (nrOfResizes > 0 && (c -> status & DO_EXP))
            c -> status |= DO_RESIZE;
    }
    else if (nrOfResizes > 0) {
        c -> status |= DO_RESIZE;
    }

    if (c -> status & DO_EXP || c -> status & DO_RESIZE) c -> status |= DO_SPACE;

    if (!(c -> status & DEVMOD) || c -> status & ISMACRO) {
        dsp = dmOpenStream (lkey, "mc", "r");
        while (dmGetDesignData (dsp, GEO_MC) > 0) {

            if (gmc.imported == IMPORTED) continue;
            name = gmc.cell_name;

            for (a = c -> children; a; a = a -> next) {
                if (strcmp (a -> cell -> name, name) == 0) break;
            }
            if (!a) {
                a = NEW (struct adj, 1);
                a -> next = c -> children;
                c -> children = a;
                a -> cell = makeTree (name, depth + 1);
            }
        }
        dmCloseStream (dsp, COMPLETE);
    }

    c -> candidates = candidates;
    candidates = c;
    return (c);
}

void findCandidateParents ()
{
    struct adj *a;
    struct Cell *c;
    int status, status2;

    invertHierarchy ();

    status2 = DO_SPACE | DO_EXP;

    for (c = candidates; c; c = c -> candidates) {

        /* must continue, because layout already checked in. */
        if (c -> status & DONE_SPACE) continue;

        status = 0;
        if (c -> status & DO_SPACE) status = DO_SPACE;
        if (c -> status & DO_EXP)  status |= DO_EXP;
        if (status) {
            for (a = c -> parents; a; a = a -> next)
                a -> cell -> status |= status;
        }

        if (status != status2)
        if (c -> status & ISMACRO || (!(c -> status & DO_EXP)
                && existDmStream (c -> lkey, "is_macro"))) {
            for (a = c -> parents; a; a = a -> next) {
                if (compareGlnDate (c -> lkey, "is_macro",
                                     a -> cell -> lkey) <= 0)
                    a -> cell -> status |= status2;
            }
        }
    }
}

void invertHierarchy ()
{
    struct adj *a; struct Cell *c;
    for (c = cells; c; c = c -> next) {
        for (a = c -> children; a; a = a -> next) addParent (a -> cell, c);
    }
}

void addParent (struct Cell * child, struct Cell * parent)
{
    struct adj *a;
    for (a = child -> parents; a; a = a -> next) {
        if (strcmp (a -> cell -> name, parent -> name) == 0) return;
    }
    a = NEW (struct adj, 1);
    a -> next = child -> parents;
    child -> parents = a;
    a -> cell = parent;
}

void determineDepths ()
{
    struct adj *a; struct Cell *c, *f;

    for (c = cells; c; c = c -> next) c -> depth = 0; /* reset */
    c = root_c;
    c -> depth = ROOT_DEPTH;

    do { /* bfs() */
        for (a = c -> children; a; a = a -> next) {
            if (a -> cell -> depth == 0) {
                a -> cell -> depth = c -> depth + 1;
                for (f = c; f; f = f -> fringe) {
                    if (f -> fringe == NULL) { /* fappend() */
                        f -> fringe = a -> cell;
                        break;
                    }
                }
                a -> cell -> fringe = NULL;
            }
        }
    } while (c = c -> fringe);
```

```
}

do_t * makeCandidateArray ()
{
    struct Cell *c;
    static do_t *list = NULL;
    static int size = 0;
    int num = 0;

    for (c = candidates; c; c = c -> candidates) {
        if ((!optFlat && c -> depth <= optMaxDepth && c -> depth > 0)
            || (optFlat && c == root_c)) num++;
    }

    if (num + 1 > size) {
        if (size > 0) DISPOSE (list);
        size = num + 1;
        list = NEW (do_t, size);
    }

    list[num].name = NULL;

    for (c = candidates; c; c = c -> candidates) {
        if ((!optFlat && c -> depth <= optMaxDepth && c -> depth > 0)
            || (optFlat && c == root_c)) {
            do_t * l = list + (--num);
            l -> status = c -> status;
            l -> name   = c -> name;
            l -> lkey   = c -> lkey;

            /* Reset macro status of root. Because the macro status
             * has a meaning only in the context of a father cell.
             */
            if (c -> depth == ROOT_DEPTH) l -> status &= ˜ISMACRO;

            /* In principle, do not re-extract cells in one run.
             */
            if (c -> status & DONE_SPACE) l -> status &= ˜DO_SPACE;

            if (c -> depth <= optMinDepth) {
                l -> status |= DO_SPACE;
                if (c -> status & DONE_SPACE) {
                    l -> lkey = dmCheckOut (dmproject, c -> name,
                        WORKING, DONTCARE, LAYOUT, ATTACH);
                    l -> status &= ˜DO_EXP;
                    l -> status &= ˜DO_RESIZE;
                }
            }

            c -> status |= DONE_SPACE;
        }
    }

    return (list);
}
```

APPENDIX C -- NEW SOURCE FILE: sp_main.c

```
void extractTree (char * cellName)
{
    do_t * list = NULL;
    int i;

    if (!optFlat) verbose ("extract hierarchy of %s\n", cellName);

    list = findCandidates (dmproject, cellName);

    for (i = 0; list != NULL && list[i].name != NULL; i++) {
        Debug (fprintf (stderr,
            "%s, status %#o\n", list[i].name, list[i].status));
/* 1 */ if (list[i].status & ISMACRO) {
            verbose ("%s skipped, has macro status\n", list[i].name);
        }
        else if (list[i].status & DEVMOD) {
            verbose ("%s skipped, device model present\n", list[i].name);
        }
        else if (list[i].status & DO_SPACE) {
/* 2 */     *circuitName = '\0';
            if (alsoPrePass) {
                if (((optResMesh || substrRes) && optStep == 0) || optStep == 1) {
                    if (!(optSubRes && paramLookupB ("use_subres", "off"))) {
                        prePass1 = TRUE;
                        prepass (list + i);
                        prePass1 = FALSE;
                        if ((optResMesh || substrRes) && optStep == 0)
                            processgln (list + i);
                    }
                }
                if (((optSelectiveRes || optPrick) && optStep == 0) || optStep == 2) {
                    prePass2 = TRUE;
                    prepass (list + i);
                    prePass2 = FALSE;
                }
            }

            if ((!optOnlyPrePass && optStep == 0) || optStep == 3)
                extract (list + i);
        }
        else {
/* 3 */     char * type = "";
            if (list[i].status & DONE_FLAT) {
                type = " flat";
            }
            else if (list[i].status & DONE_PSEUDO) {
                if (!optPseudoHier) type = " pseudo";
            }
            else {
                if (optPseudoHier) type = " !pseudo";
            }
            verbose ("%s already%s extracted\n", list[i].name, type);
        }
    }
}
```

APPENDIX D -- NEW SOURCE FILE: determ.c

```
/*
 * ISC License
 *
 * Copyright (C) 1988-2000 by
 *     Arjan van Genderen
 *     Nick van der Meijs
 *     Simon de Graaf
 * Delft University of Technology
 *
 * Permission to use, copy, modify, and/or distribute this software for any
 * purpose with or without fee is hereby granted, provided that the above
 * copyright notice and this permission notice appear in all copies.
 * ...
 *
 * See the LICENSE file.
 */

#include ..........

extern int nrOfResizes;
extern resizeData_t * resizes;

#ifdef __cplusplus
  .........
#endif

struct Cell {
    char * name;
    struct Cell * next;         /* all cells seen so far */
    struct Cell * candidates;   /* extraction candidates in dfs order */
    struct Cell * fringe;
    struct adj * children;
    int depth;
    unsigned int status;
    unsigned long smtime;       /* status last modified time */
    unsigned long lmtime;       /* layout last modified time */
    DM_CELL * lkey;
};

struct adj {
    struct Cell * cell;
    struct adj  * next;
};

static DM_PROJECT *dmproject;

/* The head of a list of all cells that must be extracted (if not
 * restricted by the optMaxDepth option).
 * The list is in reverse depth-first order, after another reversion
 * it is suitable for extraction.
 */
static struct Cell * candidates;
static struct Cell * root_c;

/* The head of a list of all cells seen so far. */
static struct Cell * cells;
static char * gln_file = NULL;
static int checkoutmode;

/* Sole entry point of this module.
 *
 * Given a project and cell name as argument, this function returns
 * an array of do_t’s (not pointers to them) describing all cells
 * in the complete hierarchical tree that must be extracted.
 * The depth of the tree can be restricted by optMaxDepth.
 * The do_t also contains info describing whether the cell must
 * be preprocessed first.
 *
 * Primarily meant for hierarchical mode, but for convenience it
 * also works in flat mode.
 *
 * The end of the list is marked by .name == NULL.
 *
 * General strategy:
 * 1. Make a dag with root at the top.
 * 2. Determine depth (distance to the root) of each cell.
 * 3. Make a linked list in correct order for extraction,
 *    and return the list.
 */
do_t * findCandidates (project, root)
DM_PROJECT * project;
char * root;
{
    dmproject = project;

    if (!gln_file)
        gln_file = strsave (mprintf ("%s_gln", masktable[0].name));

    checkoutmode = (((alsoPrePass || optOnlyPrePass)
        && (optResMesh || optSelectiveRes || optPrick || substrRes))
        || optGenNetTerm
        || (optCap3D
        && paramLookupB ("cbemdata_output", "off"))) ? ATTACH : READONLY;

    root_c = makeTree (root, ROOT_DEPTH, 0);

    return makeCandidateArray ();
}

private struct Cell * makeTree (name, depth, imported)
char *name;
int depth;
int imported;
{
    struct stat buf1;
    struct adj  * a;
    struct Cell * c;
    DM_PROJECT *project;
    DM_STREAM * dsp;
    DM_CELL   * lkey;
    char * versionstatus;
    int equiv_circuit, flag, mode;

    /* Check if cell has already been done.
```

```
     * If so, return immediately
     */
    for (c = cells; c && strcmp (c -> name, name); c = c -> next);
    if (c) return (c);

    Debug (fprintf (stderr, "pre-order visit ’%s’, depth %d\n",
        name, depth));

    c = NEW (struct Cell, 1);
    c -> name = strsave (name);
    c -> children = NULL;
    c -> status = 0;            /* assume circuit present */

    c -> next = cells;
    cells = c;

    if (imported) {
        IMPCELL *icp = dmGetImpCell (name);
        project = dmOpenProject (icp -> dmpath, PROJ_READ);
        name = icp -> cellname;
        c -> depth = -1; /* never a candidate for extraction */
    }
    else {
        project = dmproject;
        c -> depth = 0;
    }

    versionstatus = (depth == ROOT_DEPTH ? WORKING : ACTUAL);

    /* Determine checkout mode.
     * Do read-only if possible to increase potential concurency
     * when running under release 4.
     */
    mode = (imported || optFlat && depth > ROOT_DEPTH)? READONLY : checkoutmode;

    lkey = dmCheckOut (project, name,
                versionstatus, DONTCARE, LAYOUT, mode);

    if (!(c -> lkey = lkey)) return (NULL);

    if (dmStat (lkey, "mc", &buf1) == 0) /* layout last modified time */
        c -> lmtime = buf1.st_mtime;
    else {
        c -> lmtime = 0;
        dmError (name);
    }

    if (dmStat (lkey, "is_macro", &buf1) == 0) { /* stream exist */
        c -> smtime = buf1.st_mtime;
        dsp = dmOpenStream (lkey, "is_macro", "r");
        if (fscanf (dsp -> dmfp, "%d", &flag) > 0 && flag == 1) {
            PRINT ("is macro");
            c -> status |= ISMACRO;
        }
        dmCloseStream (dsp, COMPLETE);
    }
    else
        c -> smtime = 0;
```

```
#if NCF_RELEASE >= 400

    equiv_circuit = 0;

    if (name) {
        char **equiv4;
        char * cp;

        /* Cannot use an argument name of form ’name#3’ */
        if (cp = strchr (name, ’#’)) *cp = ’\0’;

        equiv4 = (char **) dmGetMetaDesignData (CELLEQUIVALENCE,
                                project, name, LAYOUT, CIRCUIT);
        if (cp) *cp = ’#’;
        if (equiv4 && equiv4[0]) {
            equiv_circuit = 1;
            name = equiv4[0];
        }
    }
#else /* NCF_RELEASE */

    /* We don’t use paramCapitalize: cktName (name) anymore! */
    equiv_circuit = (int) dmGetMetaDesignData (EXISTCELL,
                                project, name, CIRCUIT);
#endif /* NCF_RELEASE */

    if (equiv_circuit) {
        DM_CELL *ckey;

        Debug (fprintf (stderr, "equivalent circuit ’%s’\n", name));

        ckey = dmCheckOut (project, name,
                        WORKING, DONTCARE, CIRCUIT, READONLY);
        mode = dmStat (ckey, "devmod", &buf1);
        dmCheckIn (ckey, COMPLETE);

        if (mode == 0) { /* stream exist */
            if (buf1.st_mtime > c -> smtime) /* device status changed */
                c -> smtime = buf1.st_mtime;
            PRINT ("device model present");
            c -> status |= DEVMOD;
            if (!optPseudoHier && !(c -> status & ISMACRO)) {
                dmCheckIn (lkey, COMPLETE);
                lkey = NULL;
            }
        }
    }

    if (imported) {
        if (lkey) dmCheckIn (lkey, COMPLETE);
        goto ret;
    }

    if (lkey) {
        /* In hierarchical mode, recursively visit the children
         * to see whether they must be extracted.
         * In flat mode, recursively visit the children to see
         * if the root cell should be pre-processed.
         */
        if ((dsp = dmOpenStream (lkey, "mc", "r")) == NULL) return (NULL);

        while (dmGetDesignData (dsp, GEO_MC) > 0) {

            name = gmc.cell_name;

            a = c -> children;
            while (a && strcmp (a -> cell -> name, name)) a = a -> next;
            if (!a) { /* add new child */
                a = NEW (struct adj, 1);
                a -> next = c -> children;
                c -> children = a;
                a -> cell = makeTree (name, depth + 1, gmc.imported); /* recursive call */
                if (optFlat || optPseudoHier) {
                    if (a -> cell -> lmtime > c -> lmtime)
                        c -> lmtime = a -> cell -> lmtime;
                    if (a -> cell -> smtime > c -> lmtime)
                        c -> lmtime = a -> cell -> smtime;
                }
            }
        }
        dmCloseStream (dsp, COMPLETE);
    }

    c -> candidates = candidates;
    candidates = c;
ret:
    Debug (fprintf (stderr, "post-order visit ’%s’, depth %d\n",
        c -> name, depth));
    return (c);
}

/* Set the last modified time correctly for hierarchical mode.
 */
private unsigned long traverse (c, ct, flat)
struct Cell *c;
unsigned long ct;
int flat;
{
    struct adj *a;
    for (a = c -> children; a; a = a -> next) {
        c = a -> cell;
        if (c -> lmtime > ct) ct = c -> lmtime;
        if (c -> smtime > ct) ct = c -> smtime;
        if (flat || c -> status & ISMACRO) ct = traverse (c, ct, flat);
    }
    return ct;
}

/* Make an array of do_t’s that must be extracted:
 * OptMaxDepth is for debugging only, it can deliver a non-consistent tree.
 */
private do_t * makeCandidateArray ()
{
    static do_t *list;
    static int size = 0;
    struct adj  *a;
    struct Cell *c, *f;
    struct stat buf1, buf2;
    do_t *l;
    unsigned long ct;
    int num;

    if (!(c = root_c)) return (NULL);

    /* Make candidates fringe list and
     * count the number of cells to be extracted.
     */
    c -> depth = ROOT_DEPTH; /* = 1 */
    num = 1;

    if (optFlat) {
        Debug (fprintf (stderr, "candidate ’%s’, depth %d, status %d\n",
            c -> name, c -> depth, (int) c -> status));
        if (c -> status & DEVMOD) {
            PRINT ("is device");
        }
        else
            c -> status |= DO_SPACE;
    }
    else for (f = c;; c = c -> fringe) {
        Debug (fprintf (stderr, "candidate ’%s’, depth %d, status %d\n",
            c -> name, c -> depth, (int) c -> status));

        if (c -> status & DEVMOD) {
            PRINT ("is device");
        }
        else if (c -> status & ISMACRO && c != root_c) {
            PRINT ("is macro");
        }
        else if (c -> depth > optMinDepth) { /* Extract only if needed! */
            DM_CELL *ckey = NULL;
            int ok = 1;

            if (c -> status >= DONE_SPACE) goto needed;

            if (!(ckey = existCircuit (c -> name)) || dmStat (ckey, "mc", &buf1)) {
                PRINT ("no equivalent circuit");
                ok = 0;
                goto needed;
            }

            ct = c -> lmtime;

            if (existDmStream (ckey, "flat")) {
                PRINT ("previous extraction was linear");
                if (!optPseudoHier) ct = traverse (c, ct, 1);
                ok = DONE_FLAT;
            }
            else if (existDmStream (ckey, "pseudo_hier")) {
                PRINT ("previous extraction was pseudo-hier");
                if (!optPseudoHier) ct = traverse (c, ct, 1);
                ok = DONE_PSEUDO;
            }
            else {
                PRINT ("previous extraction was hier?");
                if (!optPseudoHier) ct = traverse (c, ct, 0);
            }

            if (ct >= buf1.st_mtime) {
                PRINT ("layout is newer");
                ok = 0;
            }
needed:
            if (ckey) dmCheckIn (ckey, COMPLETE);

            if (!ok) {
                c -> status |= DO_SPACE;
                PRINT ("extraction needed!");
            }
            else {
                PRINT ("already correct extracted");
                if (ok >= DONE_FLAT) {
                    c -> status |= ok;
                    if (ok == DONE_FLAT) goto skip; /* children */
                }
            }
        }
        else {
            c -> status |= DO_SPACE;
            PRINT ("extract: depth <= min_depth");
        }

        if (c -> depth >= optMaxDepth) {
            PRINT ("children not extracted: depth >= max_depth");
            goto skip;
        }
        for (a = c -> children; a; a = a -> next) {
            if (a -> cell -> depth == 0) {
                f -> fringe = a -> cell;
                f = a -> cell;
                f -> depth = c -> depth + 1;
                ++num;
            }
        }
skip:
        if (c == f) break;
    }

    /* make sure the list is long enough */
    if (num + 1 > size) {
        if (size > 0) DISPOSE (list);
        size = num + 1;
        list = NEW (do_t, size);
    }

    /* put marker in first position */
    l = list + num;
    l -> name = NULL;

    /* By flat extraction, there is only 1 candidate and
     * this candidate is always extracted (except when it is a device),
     * because c -> depth == 1 and optMinDepth == 1.
     *
     * The candidates are returned in reverse order,
     * the highest level subcells are done first, the topcell last.
     */
    if (num > 1) c = candidates;
    for (; c; c = c -> candidates) {
        if (c -> depth <= 0) continue;

        --num;
        --l;
        l -> status = c -> status;
        l -> name = c -> name;
        Debug (fprintf (stderr, "candidate ’%s’, depth %d, status %d\n",
            c -> name, c -> depth, (int) c -> status));
        c -> depth = 0;

        if (!(c -> status & DO_SPACE)) {
            if (num == 0) break;
            continue;
        }
        c -> status &= ˜DO_SPACE;

        if (l -> status & ISMACRO) l -> status &= ˜ISMACRO;

        if (c -> status & DONE_SPACE) {
            /* If they were done before, check ’m out again.
             */
            l -> lkey = dmCheckOut (dmproject, c -> name,
                    WORKING, DONTCARE, LAYOUT, checkoutmode);
            PRINT ("done before: no preprocessing!");
        }
        else {
            l -> lkey = c -> lkey;
            c -> status |= DONE_SPACE;

        if (!optNoPrepro) { /* preprocessing needed? */
            DM_CELL *lkey = c -> lkey;
            int ok = 0;

            if (dmStat (lkey, gln_file, &buf1)) {
                PRINT ("no gln file, no previous exp");
                goto not_ok;
            }

            if (!(optFlat || optPseudoHier))
                ct = traverse (c, c -> lmtime, 0);
            else
                ct = c -> lmtime;

            if (ct > buf1.st_mtime) {
                PRINT ("previous exp was old");
            }
            else if (existDmStream (lkey, "spec")) {
                PRINT ("previous exp was linear");
                ok = optFlat;
            }
            else if (existDmStream (lkey, "pseudo_hier")) {
                PRINT ("previous exp was pseudo-hier");
                ok = (!optFlat && optPseudoHier);
            }
            else {
                PRINT ("previous exp was hier?");
                ok = (!optFlat && !optPseudoHier);
            }
not_ok:
            if (!ok) {
                PRINT ("previous exp not ok, do exp!");
                l -> status |= DO_EXP;
                if (nrOfResizes > 0) l -> status |= DO_RESIZE;
            }
            else if (dmStat (lkey, "resize.t", &buf2) == 0) { /* exists */
                int resize_valid = (buf2.st_mtime > buf1.st_mtime);
                if (nrOfResizes > 0) {
                    if (!resize_valid || !resizeUpToDate (lkey)) {
                        PRINT ("previous resize not ok, do it!");
                        l -> status |= (DO_EXP | DO_RESIZE);
                    }
                }
                else if (resize_valid) {
                    PRINT ("previous resize, do exp!");
                    l -> status |= DO_EXP;
                }
                else
                    PRINT ("previous exp and resize ok!");
            }
            else if (nrOfResizes > 0) {
                PRINT ("previous exp ok, but do resize!");
                l -> status |= DO_RESIZE;
            }
            else
                PRINT ("previous exp ok!");
        }
        }
        if (num == 0) break;
    }

    ASSERT (l == list);

    /* The list with ’struct Cell’s is not destroyed since these structures
       may be used for the extraction of other cells (if more than one cell
       is specified on the argument list or with Xspace).
    */

    return (list);
}

/* Returns TRUE if stream exists, FALSE otherwise.
 */
bool_t existDmStream (key, streamName)
DM_CELL * key;
char * streamName;
{
    struct stat buf;
    return (dmStat (key, streamName, &buf) == 0 ? TRUE : FALSE);
}
```

```
private int resizeUpToDate (layoutKey)
DM_CELL *layoutKey;
{
    int ok;
    int i, j;
    int maj, min, nr;
    char buf[512];
    double val;
    int condcnt;
    int id, p, a;
    resizeCond_t *resCond;
    DM_STREAM * dsp_rs;

    /* read resize information from the stream "resize.t" to
       decide whether or not it is necessary to run makesize again.
     */

    if (dsp_rs = dmOpenStream (layoutKey, "resize.t", "r")) {
        ok = 1;
        if (dmScanf (dsp_rs, "%d %d", &maj, &min) != 2
            || maj > 1 || min > 1) {
            ok = 0;
            goto end_of_resize_check;
        }
        if (dmScanf (dsp_rs, "%d\n", &nr) != 1
            || nr != nrOfResizes) {
            ok = 0;
            goto end_of_resize_check;
        }
        for (i = 0; i < nrOfResizes; i++) {
            if (dmScanf (dsp_rs, "%s %d %lg %d",
                         buf, &id, &val, &condcnt) != 4
                || strcmp (resizes[i].newmaskname, buf)
                || resizes[i].id != id
                || resizes[i].val != val
                || resizes[i].condcnt != condcnt) {
                ok = 0;
                goto end_of_resize_check;
            }
            resCond = resizes[i].cond;
            for (j = 0; j < condcnt; j++) {
                if (dmScanf (dsp_rs, " %d %d", &p, &a) != 2
                    || p != resCond -> present
                    || a != resCond -> absent) {
                     ok = 0;
                     goto end_of_resize_check;
                }
                resCond = resCond -> next;
            }
        }
    }
    else
        return (0);

end_of_resize_check:
    dmCloseStream (dsp_rs, COMPLETE);
    return (ok);
}
```

```
/* Get imported celllist structure for imported cell (alias) name.
 * Don’t read the imported celllist more than ones.
 */
private IMPCELL * dmGetImpCell (name)
char *name;
{
    register IMPCELL **icl;
#if NCF_RELEASE >= 400
    int i = _dmValidView (dmproject, LAYOUT);
#else
    int i = _dmValidView (LAYOUT);
#endif
    if (i < 0 || !(icl = dmproject -> impcelllist[i]))
        icl = (IMPCELL **) dmGetMetaDesignData (IMPORTEDCELLLIST, dmproject, LAYOUT);
    if (icl)
    while (*icl) {
        if (strcmp ((*icl) -> alias, name) == 0) return *icl;
        ++icl;
    }
    dmerrno = DME_NOCELL;
    dmError (name);
    return 0;
}

/* Look if an equivalent extracted circuit exist for the layout cell name.
 * If the circuit cell exists, check it out for usage.
 * Note that the layout cell name can’t be an imported cell name.
 */
private DM_CELL * existCircuit (name)
char * name;
{
    DM_CELL *ckey;
    char *cp, c;
    int exist;
#if NCF_RELEASE >= 400
    char **equiv4;

    /* Cannot use an argument name of form ’name#3’ */
    if (cp = strchr (name, ’#’)) { c = *cp; *cp = ’\0’; }

    equiv4 = (char **) dmGetMetaDesignData (CELLEQUIVALENCE,
                                dmproject, name, LAYOUT, CIRCUIT);
    exist = (equiv4 && equiv4[0]);
    if (exist) name = equiv4[0];
#else
    cp = 0;
    if (paramCapitalize) { cp = name; c = *cp; *cp = toupper (*cp); }
    exist = (int) dmGetMetaDesignData (EXISTCELL, dmproject, name, CIRCUIT);
#endif
    if (exist)
        ckey = dmCheckOut (dmproject, name,
                        WORKING, DONTCARE, CIRCUIT, READONLY);
    else
        ckey = NULL;
    if (cp) *cp = c;
    return ckey;
}
```

APPENDIX E -- SOURCE FILES SCCS DELTA's

SCCS/s.determ.h:

```
D 4.11 00/02/24 16:58:49 space 11 10
COMMENTS: Added modes DONE_FLAT/DONE_PSEUDO. (SdeG)
D 4.10 00/02/18 13:59:45 space 10 9
COMMENTS: Removed versionnumber and obsolete function cktName. (SdeG)
D 4.9 00/01/28 16:42:43 space 9 8
COMMENTS: added resizing option with makesize (AvG)
D 4.8 93/12/15 10:29:36 space 8 7
COMMENTS: mainly header comment, but some other small changes (NvdM)
```

SCCS/s.determ.c:

```
D 4.41 00/02/24 17:38:30 space 43 42
COMMENTS: Parent not more extracted, if a child is extracted!
In hier.mode the type of extraction may be of diff. type! (SdeG)
D 4.40 00/02/23 10:32:10 space 42 41
COMMENTS: Use fringe list to get correct cells order. (SdeG)
D 4.39 00/02/18 14:01:06 space 41 40
COMMENTS: Bug fix and changed some stream names. (SdeG)
D 4.38 00/02/17 16:58:26 space 40 39
COMMENTS: Rewritten. Also the DRIVER part updated. (SdeG)
D 4.37 00/02/04 14:34:12 space 39 38
COMMENTS: in case of resize, also perform extraction (AvG)
D 4.36 00/01/31 15:06:11 space 38 37
COMMENTS: improved algorithm to determine when resizing is done (AvG)
D 4.35 00/01/28 16:42:44 space 37 36
COMMENTS: added resizing option with makesize (AvG)
D 4.34 00/01/28 10:28:24 space 36 35
COMMENTS: Bug fix: (lkey, "flat") must be (ckey, "flat");
made some other code improvements. (SdeG)
D 4.33 99/06/29 14:25:38 space 35 34
COMMENTS: bug fix for previous change (AvG)
D 4.32 99/06/29 12:31:44 space 34 33
COMMENTS: added parameter expand_connectivity (AvG)
D 4.31 99/05/25 10:25:51 arjan 33 32
COMMENTS: update for more portability (AvG)
```

SCCS/s.sp_main.c:

```
D 4.14 00/02/24 17:23:39 space 14 13
COMMENTS: Tells type of extraction done by hier mode. (SdeG)
D 4.13 00/02/23 10:23:56 space 13 12
COMMENTS: MACRO status test must be before DEVICE test! (SdeG)
D 4.12 00/02/17 16:55:40 space 12 11
COMMENTS: Updated for changes in determ.c  (SdeG)
D 4.11 00/01/28 16:42:46 space 11 10
COMMENTS: added resizing option with makesize (AvG)
D 4.10 99/08/20 09:06:26 space 10 9
COMMENTS: updates for making public space (AvG)
```

CONTENTS

LIST OF FIGURES