SPACE APPLICATION NOTE
ABOUT
TID FILES

S. de Graaf

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

Report ET-CAS 00-03
April 4, 2000

Last revision: December 2, 2003.

1. What Does Space With The Tid Files?

The "tid" stream contains cell records and terminal records. A cell record must always be first. Each cell record can have zero or more terminal records. A cell record contains the cell_name, instance_name and the number of cell copies (m_nx and m_ny are 0 if there are no copies).

A terminal record contains the terminal identification number, the terminal_name and the number of terminal copies (t_nx and t_ny are 0 if there are no copies). *Space* assumes that the terminal identification number starts with 0 and that each terminal is sequential numberred (terminal y,x-copies are done first, and after that the possible cell instance y,x-copies). A terminal record of the next cell instance must start with the correct next terminal identification number. This numbers are also written to the "t_LC_bxx" streams and can only with this assumtion correctly be matched.

Example:

```
total  $  0  0   (cell,inst,m_nx,m_ny)
0 t1 0 0  (t_nr,term,t_nx,t_ny)
1 t2 0 0
2 t3 0 0
  ...
9 t10 0 0
subA  ia  2  1
10 in  0 3
34 out 0 3
subB  ib  2  1
58 in  0 3
```

An expansion for terminal ’in’ gives the following numbering:

```
10 = ia[0,0].in[0]
11 = ia[0,0].in[1]
12 = ia[0,0].in[2]
13 = ia[0,0].in[3]
14 = ia[0,1].in[0]
15 = ia[0,1].in[1]
16 = ia[0,1].in[2]
17 = ia[0,1].in[3]
18 = ia[1,0].in[0]
19 = ia[1,0].in[1]
20 = ia[1,0].in[2]
21 = ia[1,0].in[3]
22 = ia[1,1].in[0]
23 = ia[1,1].in[1]
24 = ia[1,1].in[2]
25 = ia[1,1].in[3]
26 = ia[2,0].in[0]
27 = ia[2,0].in[1]
28 = ia[2,0].in[2]
29 = ia[2,0].in[3]
30 = ia[2,1].in[0]
31 = ia[2,1].in[1]
32 = ia[2,1].in[2]
33 = ia[2,1].in[3]
```

Space writes only for the top cell (cell with instance_name ’$’) the terminals with addTerminal() to the "term" stream.

Space writes all other cell records with addInstance() to the "mc" stream of the top cell circuit. If paramCapitalize is true, then the cell_name is capatalized! This records can have x,y-attributes, if optTorPos is true. This xl,yb-coordinates are read from the "tidpos" file. The "tidpos" file must exactly contain the same number of subcell records as the "tid" file has.

If the cell instance_name is equal to ’.’, then *space* tries to read the full instance_name from the "tidnam" file. The "tidnam" file must contain for each instance_name ’.’ a full instance_name record. If this full instance_name is also a ’.’, then "_I%d" is used. If this full instance_name is too long for the database, then the name is truncated by truncDmName() and a mapping record is written to the "nmp" file.

See source file "scan/hier.c" function readTid().

All terminals are expanded and written to a TERM[] array with newTerminal(). The type of this entries is ’tTerminal’. For each entry a terminal_t element is allocated. The conductor information (x,y) is later on read from the "t_LC_bxx" stream. Each array element has pointers to the termName and instName. Note that for cleanup the instNames are stored in the instNames[] array.

By useLeafTerminals, all terminals are also written to the LABELNAME[] array with newLabelName(). This array contains nrOfLabelNames entries. The indices of both

TERM[] and LABELNAME[] array must be pointing to the same terminal cq. label_name. The label_names for the top cell terminals are NULL. Else the label_name can be:

a) if (useCellNames || noRealInstName) " cellName_termName[tx|ty]"
b) "fullInstName[ix|iy].termName[tx|ty]"

Note: [x|y] can be "", "[x]", "[y]" or "[x,y]". Note: inst_term_sep '.' or other character! Note: convert fullInstName '/' to hier_name_sep character!

The label_name is only used in openInput(), see file "scan/input.c". There the "t_LC_bxx" are read. The gboxlay.chk_type must be < nrOfTerminals and also < nrOfLabelNames. There is set TERM[].x and TERM[].y and TERM[].conductor. Note that the conductor is an integer value from the technology file. If useLeafTerminals, for subcells the label_name is read from LABELNAME[]. If the label_name starts with a space (see a), then it is converted to:

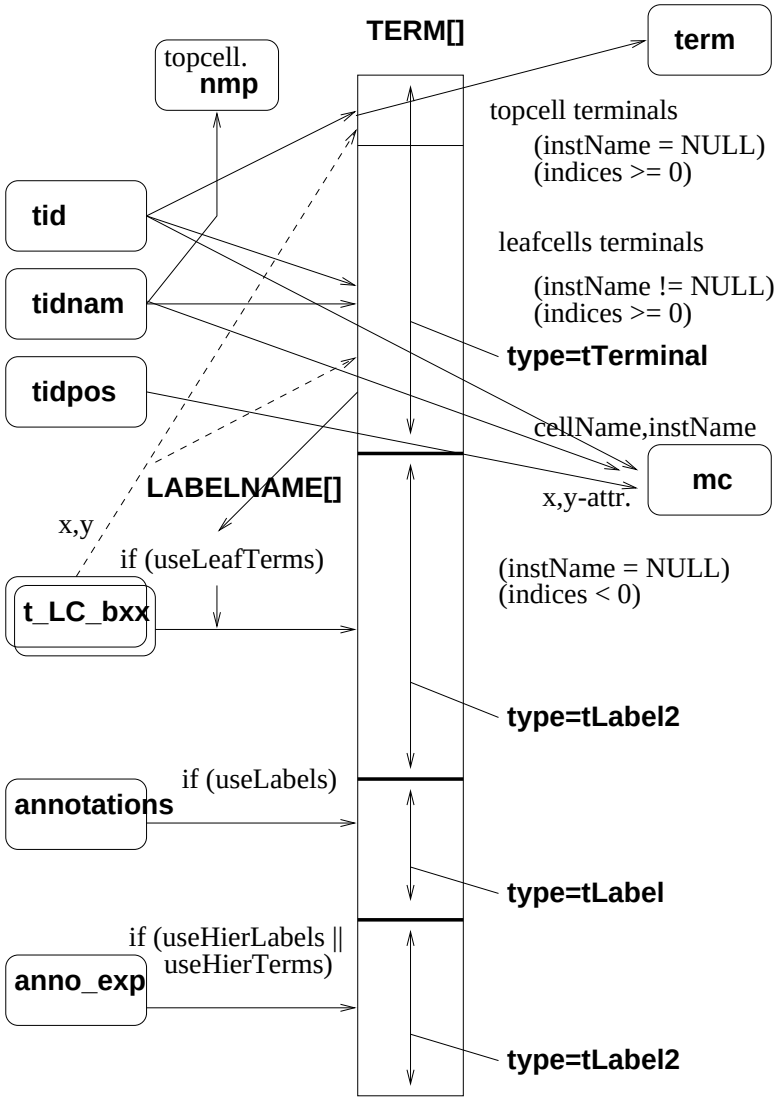
```
"cellName_termName[tx|ty]_x_y"
```

The label_name is written to the TERM[] array (with type 'tLabel2'). The LABELNAME[] array is a temporary used array, its entries are added to TERM! Note: Maybe this is not needed, the labels can be contructed later on! Note that the truncation of all these label names is done later on in outNode().

Note that openInput() reads (if ANNOTATE is defined) also the files:

a) if (useAnnotations)
    readLabels() --> "annotations" --> newTerminal(tLabel, ...)
b) if (useHierAnnotations || useHierTerminals)
    readHierNames() --> "anno_exp" --> newTerminal(tLabel2, ...)

Note: useHierAnnotations is for reading of hier. labels and (default "off")
useHierTerminals is for reading of hier. terminals (default "off")
useAnnotations is for reading of labels (default "on")



Note that an instName of a leafcell is a hierarchical name by level >= 2.

After all terminals and labels are placed in the TERM[], this array is sorted:

```
sortTerminals (TERM, nrOfTerminals);
```

And also sorted by name in the TERMBYNAME[] array:

```
addTerminalNames ();
sortTerminalsByName  (TERMBYNAME, nrOfTerminals);
```