NAME

mpack – routines for generating semi-regular modules

DESCRIPTION

Mpack is a library of 'C' routines that aid the process of generating semi-regular modules. Decoder planes, barrel shifters, and PLAs are common examples of semi-regular modules.

Using Magic, an mpack user will draw an example of a finished module and then break it into tiles. These tiles represent the building blocks for more complicated instances of the module. The mpack library provides routines to aid in assembling tiles into a finished module.

MAKING AN EXAMPLE MODULE

The first step in using mpack is to create an example instance of the module, called a *template*. The basic building blocks of the structure, or *tiles*, are then chosen. Each tile should be given a name by means of a rectangular label which defines its contents. If the tiles in the module do not abut (e.g. they overlap) it is useful to define another tile whose size indicates how far apart the tiles should be placed.

Templates should be in Magic format and, by convention, end with a **.mag** suffix. With some programs, it is possible to generate the same structure in a different technology or style by changing just the template. If this is the case, each template should have a filename of the
form basename**-***style***.mag**. The *style* part of the filename interacts with the **-s** option (see later part of this manual).

WRITING AN MPACK PROGRAM

An mpack program is the 'C' code which assembles tiles into the desired module. Typically this program reads a file (such as a truth table) and then calls the tile placement routines in the mpack library.

The mpack program must first include the file **~cad/lib/mpack.h** which defines the interface to the mpack system. Next the **TPinitialize** procedure is called. This procedure processes command line arguments, opens an input file as the standard input (**stdin**), and loads in a template.

The program should now read from the standard input and compute where to place the next tile. Tiles may be aligned with previously placed tiles or placed at absolute coordinates. If a tile is to overlap an existing tile the program must space over the distance of the overlap before placing the tile.

When all tiles are placed the program should call the routine **TPwrite_tile** to create the output file that was specified on the command line.

To use the mpack library be sure to include it with your compile or load command (e.g. **cc** *your_file* **~cad/lib/mpack.lib**).

ROUTINES

Initialization and Output Routines

TPinitialize(*argc, argv, base_name***)**

The mpack system is initialized, command line arguments are processed, and a template is loaded. The file descriptor **stdin** is attached to the input file specified on the command line. The template's filename is formed by taking the *base_name*, adding any extension indicated by the **-s** option, and then adding the **.mag** suffix. The **-t** option allows the user to override *base_name* from the command line.

Argc and *argv* should contain the command line arguments. *Argc* is a count of the number of arguments, while *argv* is an array of pointers to strings. Strings of length zero are ignored (as is the flag consisting of a single space), in order to make it easy for the calling program to intercept its own arguments. *Argc* and *argv* are of the same structure as the two parameters passed to the main program. A later section of this manual summarizes the command line options.

TPload_tiles(*file_name***)**

The given *file_name* is read, and each rectangular label found in the file becomes a tile accessible via TPname_to_tile. No extensions are added to *file_name*.

TILE TPread_tile(*file_name***)**
A tile is created and *file_name* is read into it. The tile is returned as the value of the function.

TPwrite_tile(*tile, filename***)**
The tile *tile* is written to the file specified by *filename*, with **.ca** or **.cif** extensions added. See the description of the **-o** option for information on what file name is chosen if *filename* is the null string. The choice between Magic or CIF format is chosen with the **-a** or **-c** command line options.

Tile creation, deletion, and access

TPdelete_tile(*tile***)**
The tile *tile* is deleted from the database and the space occupied by it is reused.

TILE TPcreate_tile(*name***)**
A new, empty tile is created and given the name *name*. This name is used by the routine **TPname_to_tile** and in error messages. The type **TILE** returned is a unique ID for the tile, not the tile itself. Currently this is implemented by defining the type TILE to be a pointer to the internal database representation of the tile.

int TPtile_exists(*name***)**
TRUE (1) is returned if a tile with the given *name* exists (such as in the template or from a call to TPcreate_tile).

TILE TPname_to_tile(*name***)**
A value of type **TILE** is returned. This value is a unique ID for the tile that has the name *name*. This name comes from a call to TPcreate_tile(), or from the rectangular label that defined it in a template that was read in by TPread_tiles() or TPinitialize(). If the tile does not exist then a value of NULL is returned and an error message is printed.

RECTANGLE TPsize_of_tile(*tile***)**
A rectangle is returned that is the same size as the tile *tile*. The rectangle's lower left corner is located at the coordinate (0, 0). All coordinates in mpack are specified in half-lambda.

Painting and Placement Routines

RECTANGLE TPpaint_tile(*from_tile, to_tile, ll_corner***)**
The tile *from_tile* is painted into the tile *to_tile* such that its lower left corner is placed at the point *ll_corner* in the tile *to_tile* . The location of the newly painted area in the output tile is returned as a value of type RECTANGLE. The tile *to_tile* is often an empty tile made by **TPcreate_tile()**. The point *ll_corner* is almost never provided directly, it is usually generated by routines such as **align()**.

TPdisp_tile(*from_tile, ll_corner***)**
A rectangle the size of *from_tile* with the lower left corner located at *ll_corner* is returned. Note that this routine behaves exactly like the routine TPpaint_tile except that no output tile is modified. This routine, in conjunction with the **align** routine, is useful for controlling the overlap of tiles.

RECTANGLE TPpaint_cell(*from_tile, to_tile, ll_corner***)**

This routine behaves like **TPpaint_tile()** except that the *from_tile* is placed as a subcell rather than painted into place. The tile *from_tile* must exist in the file system (i.e. it must have been read in from disk or have been written out to disk).

Label Manipulation Routines

TPplace_label(*tile, rect, label_name***)**

A label named *label_name* is place in the tile *tile*. The size and location of the label is the given by the RECTANGLE *rect*.

int TPfind_label(*tile, &rect1, str, &rect2***)**

The tile *tile* is searched for a label of name *str*. The location of the first such label found is returned in the rectangle *rect2*. The function returns 1 if such a label was found, and 0 otherwise. The rectangle pointer *&rect1*, if non-NULL, restricts the search to an area of the tile.

TPstrip_labels(*tile, ch***)**

All labels in the tile *tile* that begin with the character *ch* are deleted.

TPremove_labels(*tile, name, r***)**

All labels in the tile *tile* that are completely within the area *r* are deleted. If *name* is non-NULL, then only labels with that name will be affected.

TPstretch_tile(*tile, str, num***)**

The string *str* is the name of one or more labels within the tile *tile*. Each of these labels must be of zero width or zero height, i.e. they must be lines. Each of these lines define a line across which the tile will be stretched. The amount of the stretch is specified by *num* in units of **half**-lambda. Stretching such a line turns it into a rectangle. Note that if the tile contains 2 lines that are co-linear, the stretching of one of them will turn both into rectangles.

Point-Valued Routines

POINT tLL(*tile***)**
POINT tLR(*tile***)**
POINT tUL(*tile***)**
POINT tUR(*tile***)**

The location of the specified corner of tile *tile*, relative to the tile's lower left corner, is returned as a point. LL stands for lower-left, LR for lower-right, UL for upper-left, and UR for upper-right. Note that **tLL()** returns (0, 0).

POINT rLL(*rect***)**
POINT rLR(*rect***)**
POINT rUL(*rect***)**
POINT rUR(*rect***)**

The location of the specified corner of the rectangle *rect* is returned as a point. LL stands for lower-left, LR for lower-right, UL for upper-left, and UR for upper-right.

POINT align(*p1, p2***)**

A point is computed such that when added to the point *p2* gives the point *p1*. *p1* is normally a corner of a rectangle within a tile and *p2* is normally a corner of a tile. In this case the point computed can be treated as the location for the placement of the tile.

For example, TPpaint_tile(outtile, fromtile, align(rUL(rect), tLL(fromtile))) will paint the tile *fromtile* into *outtile* such that the lower left corner of *fromtile* is aligned with the upper-left corner of *rect*. In this example *rect* would probably be something returned from a previous TPpaint_tile() call.

Point and Rectangle Addition Routines

POINT TPadd_pp(*p1, p2***)**
POINT TPsub_pp(*p1, p2***)**

The points *p1* and *p2* are added or subtracted, and the result is returned as a point. In the subtract case *p2* is subtracted from *p1*.

RECTANGLE TPadd_rp(*r1, p1***)**
RECTANGLE TPsub_rp(*r1, p1***)**

The rectangle *r1* has the point *p1* added or subtracted from it. This has the effect of displacing the rectangle in the X and/or Y dimensions.

Miscellaneous Functions

int TPget_lambda()

This function returns the current value of lambda in centi-microns.

INTERFACE DATA STRUCTURES

In those cases where tiles must be placed using absolute, (half-lambda) coordinates, it is useful to know that **RECTANGLE**s and **POINT**s are defined as:

```
typedef struct {
   int x_left, x_right, y_top, y_bot;
} RECTANGLE;

typedef struct {
   int x, y;
} POINT;
```

The variable **origin_point** is predefined to be (0, 0). **origin_rect** is defined to be a zero-sized rectangle located at the origin.

OPTIONS ACCEPTED BY TPinitialize()

Typical command line: *program_name* [-t *template*] [-s *style*] [-o *output_file*] *input_file*

-a produce Magic format (this is the default)

-c produce CIF format

-v be verbose (sequentially label the tiles in the output for debugging purposes; also print out information about the number of rectangles processed by mpack)

-s *style* generate output using the template for this style (see TPinitialize for details)

-o The next argument is taken to be the base name of the output file. The default is the input file name with any extensions removed. If there is not input file specified and no -o option specified, the output will go to **stdout**.

-p (pipe mode) Send the output to **stdout**.

-t The next argument specifies the template base name to use. This overrides the default supplied by the program. A **.mag** extension is automatically added. (see TPinitialize)

-l *name* Set the cif output style to *name*. *name* is the name of a cif output style as defined in Magic's technology file. If this option is not specified then the first output style in the technology file is used. (Note: In the old tpack system this option set the size of lambda.)

input_file
The name of the file that the program should read from (such as a truth table file). If this filename is omitted then the input is taken from the standard input (such as a pipe).

-M *num* This option is accepted by mpack, but ignored. It is a leftover from the tpack system.

-D *num1 num2*
The *Demo* or *Debug* option. This option will cause **mpack** to place only the first *num1* tiles, and the last *num2* of those will be outlined with rectangular labels. In addition, if a tile called "blotch" is defined then a copy of it will be placed in the output tile upon each call to the *align* function during the placing of the last *num2* tiles. The blotch tile will be centered on the first point passed to *align*, and usually consists of a small blotch of brightly colored paint. This has the effect of marking the alignment points of tiles. The last tile painted into is assumed to be the output tile.

EXAMPLE

It is highly recommended that the example in **˜cad/src/mquilt** be examined. Look at both the template and the 'C' code. A more complex example is in **˜cad/src/mpla**.

FILES

cad/lib/mpack.h	(definition of the mpack interface)
cad/lib/mpack.lib	(linkable mpack library)
cad/lib/mpack.ln	(lint-library for lint)
cad/src/mquilt/*	(an example of an mpack program)
cad/lib/magic/sys/*.tech*	(technology description files)

ALSO SEE

magic(CAD), mquilt(CAD), mpla(CAD)
Robert N. Mayo *Pictures with Parentheses: Combining Graphics and Procedures in a VLSI Layout Tool,* ®Proceedings of the 20th Design Automation Conference, June, 1983.
\`C' Manual

HISTORY

This is a port of the tpack(1) system which generated Caesar files.

AUTHOR

Robert N. Mayo

BUGS

When a tile contains part of a subcell, or touches a subcell, then the whole subcell is considered to be part of the tile. The same goes for arrays of subcells.