NEWOL User's Guide

Massimo A. Sivilotti

November 11, 1991

1 Introduction

NEWOL is a redesign of the IC design system originally written by Carver Mead at Caltech. It is a Manhattan style IC graphics editor, with composition capability. While keeping some of the better features of the original WOL, NEWOL attempts to make designing cells and compositions easier, and hence faster.

This manual attempts to describe all of the features of the design system in a concise, easy to understand fashion. It is assumed that the reader has some idea of cell design, and it would be helpful also to have seen some other graphical editors.

This manual is broken into many sections. The first of these is the design philosophy and how this relates to using NEWOL (of course, the best way to get to know how to use NEWOL is simply to sit down and use it!!). The next sections deal with the menu system, and what each menu square does. Following the description of NEWOL itself, the WOLCOMP composition system is documented. The last section contains extra material which will become useful for experienced NEWOL users (such as Technology files and NEWOL's special CIF).

One last note before you go on to the details of NEWOL. The system has been designed to be useful, friendly, and fast. If you find yourself frustrated or confused with some part of NEWOL, then ask for help!! There are surely many people around who can, and will be able to help. Enjoy!!

2 Design philosophy

The philosophy of NEWOL is really very simple—design simple geometry cells as quickly and painlessly as possible, and then connect these cells up with a simple composition system. Composition is done by abutment, which means that the user better be careful about how wires connect!!

The NEWOL system has two different types of cells that it deals with - geometry and composition. Geometry cells only contain wires, cuts and other geometrical data. Therefore, all geometry is represented on the screen by boxes. The color of the box (see sec. 3b) determines what layer the wire is. These colors can be set by the user, and 16 different colors are available.

The second type of cell is a composition cell. The contents of a composition cell are other cells. A composition cell can be made up of any combination of geometry cells and composition cells. These cells can be rotated, mirrored, and translated any way you see fit. This allows for a very natural, hierarchical composition, which is what you want for large, complicated cells.

The NEWOL system differentiates between the two types of cells at every level. All operations are done on only one type of cell at a time. Furthermore, certain operations apply to only one type of cell.

Design in the NEWOL system is done by a method of buffers. At all times while running NEWOL, there are two main buffers available to the user. One is a geometry buffer, and the other is a composition buffer. They are named *GEOM* and *COMP* respectively. All editing functions are performed on one of these two buffers, and they are the only graphical interface the user has to all of NEWOL's memory.

While very important, these two buffers are not the entirety of NEWOL's world. Far from it!! NEWOL can have loaded at any time as many cells as the user desires (within memory limits, of course). These cells can be used for composition, and also may be edited themselves. The important thing to realize is that to edit a known cell, the cell must first be copied to the main buffer. After the editing, the changed cell can be saved back under the desired name. *All editing is done on one of the main buffers!!!*

Enough of this, however, now on to how to use the system!

2.1 NEWOL's graphics

A large measure of the popularity that NEWOL has enjoyed is attributable directly to the fast response of the system, which in turn is largely due to fast graphics. NEWOL permits several display styles, including outline graphics, filled-box graphics, and several specialized modes. These styles are selected by pressing the TAB key on the keyboard, at any time. This command mechanism is an example of keyboard commands; other such commands are described later.

3 Running NEWOL - The menu system

NEWOL originated on the HP 9836C "Chipmunk" series of computer with the HP 9111A graphics tablet attached. A version that runs on the "Bobcat" machines is also available, and can use either their graphics tablets or the mouse.

The command-line syntax of the NEWOL command is:

```
wol [-u<username>] [-t<tech>] [-q] [file1 [file2 ...]]
```

where

`-u<username>` NEWOL's first action is to load a configuration file specifying technology, size of "lambda", and several internal mode settings. The "-u" flag loads the initial configuration file belonging to user <username>.

`-t<tech>` Load the technology file `<tech>.WOL`. Only layers and colors are defined in this file; the size of "lambda" may have to be set explicitly.

`-q` Initiate quiet start-up. The message-of-the-day is not displayed, and neither is the status of CIF I/O operations.

NEWOL uses menus extensively to display and select all functions, and understanding the menu system is crucial. When invoked with no filename arguements, NEWOL comes to life displaying the OPTS menu (which contains the commands to load and save project files). If a project filename was specified on the command-line, NEWOL starts off in the MAIN menu.

The NEWOL display is really quite simple. There are three areas on the screen. The first of these is the menu area, consisting of the group of cyan colored boxes at the top of the screen. Moving the cursor (which follows the pen on the pad) to one of these menu squares, and pushing the pen down causes NEWOL to initiate whichever function the menu box represents. Some of these functions are *actions* (the function executes immediately, then returns); others are *modes* (which affect the operation of a subsequent action or pen operation). When you select a mode menu square, it is turned to purple—to show you what you have selected. The color of the printing in the menu boxes also has meaning. Selecting a box with red writing will move you to another menu, and yellow boxes represent file operations.

The second area of the screen is the large area underneath the menu boxes. This is the so-called λ-grid, which is the working area on which NEWOL draws its pictures. Notice that as you move the pen around the grid that it seems to jump around. This is because the grid is in fact a real grid. There are distinct points on the grid, and the cursor (and the corners of all NEWOL objects) can only be located on a grid point. NEWOL allows you to zoom in and out, and therefore when you move closer, the grid points are further apart. The zoom factor can be set by various "keyboard commands".

It is worth mentioning here that NEWOL uses a "window" system for the λ-grid. What you see in the grid area at any one time is only a small portion of the whole area that can be filled with cells. The ZOOM and SCROLL functions allow you to move the window, to view different parts of your work.

The last area on the screen is the status area. This is the little box in the upper right hand section of the screen. NEWOL uses this area to communicate the state of the system. Right now, there should be a colored square (the current drawing layer), and the name of the current menu (Main or Opts). There will be more on the status area later. Notice that the cursor can not be moved into the status area. This is a special reserved area, and *cannot* be used for drawing.

There are several menu squares which have no writing in them. These are extras and will be filled with functions later. For now, selecting them will have no effect.

Moving around to the various menus in NEWOL is really very simple. Menus are hierarchically organized by function of their contained operations, and all of the sub-menus have a menu square called 'MAIN' which will send you back to the main menu. Thus, you are always within a few pen pushes of any menu in NEWOL.

3.1 Text pull-down menus

Specification of cell and file names is done textually, with the optional assistance of pull-down menus listing possible choices. The text prompt for the name contains several fields which can be highlighted by using the pen. In particular, a default response is supplied (generated by the last cell/file name provided by the user) and is selected either with the pen or by pressing ENTER. An ABORT option is also provided. The last option is to press or type "?", which will pull down the full menu. Here, names are selected with the pen, or, if you wish to supply a name which is not in the menu, simply by typing the name at the keyboard (followed by ENTER). An ABORT option is also located in the menu.

Rather than type "?" at the first text prompt, the user may wish to directly enable the pull-down menu as a default action. This is specified in the INFO.WOL configuration file, by setting the short file/cell name prompt variables appropriately.

3.2 Keyboard commands

Several commands, in particular those that affect the graphics state of NEWOL are available at all times by pressing keys on the keyboard. This provides a shorthand mechanism for sophisticated users to quickly perform common operations. Typing "?" will print a short help screen describing most of the keyboard commands.

3.2.1 Selection commands

The NEWOL algorithm for selecting objects on the screen is based on a hierarchical data representation in the geometry buffer. Boxes are considered to own boxes which they entirely contain. Many operations (move, delete, etc.) can be performed on the entire hierarchy at once. The selection algorithm is concerned with identifying the subtree of boxes that correspond to the pen being pressed at a particular location. The algorithm is straight-forward: NEWOL tries to grab the *smallest* object which completely surrounds the cursor point (and of course, any children of that object), *unless* the cursor falls on an *edge*, in which case NEWOL tries to grab the *largest* box with an edge on the cursor. A few experiments will quickly teach the user how this mechanism can be used to operate on various structures.

An ambiguity exists in the case of two partially overlapping boxes; neither one contains the other (they are siblings in the hierarchy). In this case, an arbitrary (but deterministic) choice is made by the selection algorithm. In these cases, when it is difficult or inconvenient to unambiguously select a given object, two mechanisms are provided to help steer the selection algorithm:

1. The algorithms can be made principally selective to edges (as opposed to interiors of boxes) by pressing the "E" key. If the search for an edge fails, the selection algorithm reverts to considering interiors of boxes, as before. The "E" key toggles this edge-priority selection mode.
2. The select algorithm can by cycled repeatedly, indicating in sequence ALL of the selectable objects at a given point. When the desired object is highlighted, the pen is pressed (as in a normal selection operation), to actually perform the selection. Lifting the pen at any point aborts the select process altogether. Running down the list of selectable objects is performed by pressing the EXECUTE key; the list itself is cyclic. The EXECUTE key sends a CTRL-C character.

Because it may not always be desirable to operate on all members of the subhierarchy at once, various means are provided to disable its effect. If the select algorithm is run in ONE mode, only a single box is selected; all of its children are left untouched. This mode may be made the default by selecting the disable heirarchy option in either the OPTS menu of the INFO.WOL configuration file. The EXECUTE key and edge-priority select mode are still operational in ONE mode.

The converse is also true: sometimes the heirarchy is inadequate, as one may want to operate simultaneously on several unrelated objects. To this end, a SELect box can be drawn; all boxes that are entirely contained within the box are selected. Boxes are never broken; boxes that partially overlap with the SEL box are not selected.

3.2.2 Viewing commands

NEWOL has two main buffers (see *geometry buffer* and *composition buffer*); each buffer has its own window representing the current viewing transformion. Moving between the buffers in no way changes the windows. NEWOL maintains a *stack* of "graphics state" records, including such information as windowing transformations and zoom factors. Separate stacks are maintained for the geometry and composition buffers.

Z – zoom to window. NEWOL will ask the user to draw a box, which will then be used as the next viewing window (it will be fit as best as possible on the screen, without distorting the aspect ratio). As usual, lifting the pen while drawing the box aborts the operation, and leaves the window unchanged. The old transform is pushed onto the stack.

"+","–" – double and halve the zoom factor, respectively, leaving the coordinate of the center of the screen unchanged.

5 – zoom back to top level. Fits the entire contents of the buffer on the screen at once, setting zoom and center appropriately.

@ – center viewing window. Prompts for lambda coordinates on which to center the window.

arrow keys (and wheel) – scroll the screen in increments of 1/20 screen.

8,2,6,4 – scroll the screen by one page in N,S,E,W directions, respectively.

")" – explicitly PUSH a viewing transform onto the stack. The user is asked to draw a zoom box (as for the Z command) for the new transform, but the current transform is left unchanged.

"^ " – POP the viewing tranform stack, redrawing the screen as required.

"(" – exchange the top two tranforms. This is especially useful for jumping back and forth between two widely separated sections of the chip. Remember that the current transform is usually not yet on the stack.

"." – PUSH the current transform onto the stack. Usually used prior to the ')' command.

3.2.3 Display commands

R, *space* – force a screen refresh.

F – toggle display of boxes as filled regions. This is very convenient for examining connectivity, but outline boxes are often more useful for editing (where you need to see the edges).

A – toggle auto-refresh mode. NEWOL tries to minimize the number of screen refreshes that are made, but even so this may slow down experienced users (especially when doing several edits in rapid succession). In these cases, it is convenient to turn off the auto-refresh, and live with a somewhat inaccurate or messy display. When required, refreshes can be forced by pressing the space bar.

Y – draw a single ruler (Yardstick). The user is asked to select two points; a rubber-band ruler is drawn between them, and the distance in lambda is printed (if a decimal point is present in the distance value, a non-integral distance was measured). One ruler is drawn for each time "Y" is pressed; the rulers remain on the screen until the next refresh.

TAB, BACKTAB – cycle through the different display styles.

= – display coordinates. The (x,y) coordinates of the pen in lambda units are displayed in the lower right corner of the screen. This feature (and the "@" command described above) is particularly useful in locating features found by some other program (e.g. an extractor, or DRC).

D – toggle display ports mode. When enabled, ports are drawn as dotted sides on the appropriate box.

W – toggle wide routing mode. When enabled, routing wires are drawn actual width; when disabled, only centerlines are drawn (more convenient for editing).

M – toggle Manhattan routing mode. When disabled, arbitrary angle routing is permitted.

C – clear alpha screen.

G – regenerate graphics screen. This is more extensive than a simple refresh, in that the menus are regenerated as well.

3.2.4 Filing operations

L – load a CIF file. The user is prompted for the name of the CIF file (the pull-down filer menu is NOT available here).

S – save a CIF file. The user is prompted for the name of the cell to save, and the file to save it in.

P – enable the pull-down menus as defaults for LOAD, SAVE, NEW, MAKE operations.

3.2.5 Other operations

! – shell escape. Puts the user in a sub-shell. When the sub-shell is exited (by typing *exit* or CTRL-D), the user is back in NEWOL.

– execute a program from inside NEWOL. The user is prompted for the program name.

CTRL-G – enter the gripe system, to submit/examine bug reports to the NEWOL maintainer.

CTRL-W – make suggestions for high-level improvements to design tools (i.e. the next generation of NEWOL).

3.3 The Main menu

The main menu is the simplest menu in NEWOL. There are only two actions the user can take from this menu. The first is to press the far right menu square and leave the program.

The other is to press one of the other filled menu squares. This will cause NEWOL to send you to the appropriate sub-menu. These sub-menus, and their basic functions, are:

LAYER	Specify a mask layer to draw geometry
BUILD	Edit geometry cells
ELEC	Specify electrical node names and ports of geometry cells
CELL	Maintain the list of defined geometry cells
COMP	Edit and manage composition cells
OPTS	File loading and saving, and setting NEWOL's internal state
DBUG	Provide support for program maintainers
EXIT	Leave NEWOL. If changes have been made, you will be warned

3.4 The Layer menu

This menu allows the user to select the layer of the next box(es) to be drawn. Each of the layers available is shown, and the text for each is in the color of the layer. Note: These layer text names and colors can be set by the user (see *technology files*). The number of available layers is also determined by the technology being used.

Pressing one of the layer boxes has two effects. First, it changes the current drawing layer to the one selected (this is denoted by the color box in the status area changing color). Also it moves the user from the Layer menu to the Build menu and puts NEWOL into ADD mode, ready to draw a box.

The grid area shows the current geometry buffer while in the layer menu.

3.5 The Build menu

The Build menu is the menu which allows you to draw and edit *geometry cells*. The menu squares perform the following functions:

ADD This menu square puts you in ADD mode, allowing you to add boxes to the geometry. While this menu square is selected (purple in color), pressing the pen down anywhere on the pad will select the first corner of the new box. After you pick the pen up, moving the pen around will make NEWOL create a "rubber-band box" (RBB). This rubber-band box is not permanent, but is there to help you decide where it is that you want the other corner of the box to be. When you have put the pen where you would like the other corner of the box, simply press the pen down again. The box will then become a permanent part of the geometry of the cell. If, while drawing the RBB for the box, you decide that you would like to forget about drawing this box, simply lift the pen away from the tablet. This operation *aborts* most commands, not just ADD. Alternately, pressing the pen down in the menu area will also abort the operation.

MOVE This menu square allows you to move boxes around the screen. While in MOVE mode, pushing the pen down on the grid area will select an object to move. When you lift the pen, the object will follow the pen around until the pen is pushed down again. If you do not wish the object to follow the pen around while MOVing (e.g. if the screen refresh is too slow), use "point-to-point move" by pressing the "V" key on the keyboard. In this mode, select the object to be moved by pressing the pen as before, then go to the new location of the object and press the pen again.

1. Note that while in MOVE mode, a word appears in the right of the status area. This shows the MOVE sub-mode. There are three sub-modes: ALL, ONE, SEL. Repeatedly pressing the MOVE menu square will cause NEWOL to change its sub-mode. Do it several times and see what happens. The sub-modes have the following meanings: ALL means that the selected shape *and all shapes which are entirely inside of it* will be moved around the screen. ONE mode means that just the one shape selected will be moved. SEL (Select) is a bit more tricky. If you are in SEL mode, NEWOL will wait for you to go to the pad area and press the pen down twice - drawing, in the process, a rubber-band box. Everything which is completely inside of this select box will be moved with the cursor.
2. If the *corner* of a box is selected in MOVE ALL or ONE mode, no move is performed. Rather, a rubber-band edit of that corner of the box is done (this feature reduces the number of times you have to enter EDIT mode (below) for little corrections).
3. MOVE mode has one further subtlety: when in MOVE ALL or ONE modes, if the pen is dragged while depressed in selecting an object, a SEL rubber-band box will be drawn, and the system is (temporarily) in SELect mode.

DEL This mode allows you to delete boxes from the geometry buffer. Like MOVE, DEL has three modes (ALL,ONE,SEL) and they allow you to delete a shape and all of its contained shapes, just the shape itself, or a selected region of the buffer.

COPY Copy allows you to make a copy of a box or boxes and then move the copy around the screen. Like the previous two options, you can COPY a shape and all of its included shapes, just the one shape, or all of the shapes in a select box. Once you have picked up the object to copy, you drag it around by moving the pen, and put down copies by pressing the pen. Multiple copies can be placed with multiple presses; lift the pen from the tablet when you are finished copying that object.

EDIT This option allows you to select one side of a box, and move that *whole* side in or out. While the menu square is selected, to use EDIT you move to the *edge* of one of the shapes on the grid. Pressing the pen down selects that edge. Now the side will follow the pen. Pushing the pen down again selects the new position of the edge. See the MOVE command for a rubber-band edit of the corner of a box.

UNDEL This option allows you to bring back objects that had been deleted with the delete option. Moving the pen up to the UNDEL menu square and pressing it causes the last deleted object(s) to reappear (in their last position). It does not matter how many boxes were deleted (or with which sub-mode), they all come back. Pressing the UNDEL box again will cause the boxes deleted before that to come back. In all, five objects can be brought back with the UNDEL. So if you delete six objects in a row, only the last five to be deleted can be brought back.

ZOOM This option allows you to select how fine the λ-grid is. When ZOOM is selected, NEWOL will tell you what the current zoom factor is (i.e. how many screen pixels correspond to 1λ), and will then ask for the new zoom factor. Zoom factors are most useful when integers ≥ 1, but non-integral zooms are permitted (primarily for viewing purposes; it is virtually impossible to select objects when the pen grid and the λ-grid do not line up). NEWOL will then redraw the buffer for you, zooming around the center of the screen. If, instead of typing a number at the text prompt, the user draws a box on the screen, NEWOL will try to zoom to that window (this operation is the same as the "Z" keyboard command).

MAKE This is how you define the geometry buffer into a cell name. After you MAKE a cell, it exists in memory, independent of the GEOM buffer and can be used both in geometry and composition operations. *Until MAKE is done, your edits in the GEOM buffer are transient, as they were done on a* copy *of the cell; the MAKE operation overwrites the old entry with the GEOM buffer contents.* When you press the MAKE menu square, NEWOL asks for the name of the cell. NOTE: Only one cell can have a given name. Cells names can be up to 12 characters long, so use names that mean something!! After NEWOL asks for the name of the cell, it offers you two options for making the cell. If you want this cell to have the minimum bounding box (MBB) defined by its geometry, press the pen down again in the MAKE menu square. If you wish to define your own bounding box (which must be *larger* than the MBB), then move the pen to the grid, and push the pen down twice to denote the corners of the bounding box. Like before, if you decide not to make the cell (or you don't like the bounding box), and you are drawing the RBB of the bounding box, just lift the pen or move the pen up to the menu area and press it down. Like before, the box will go away, and the cell will not be made.

$\boxed{\textbf{LAYER}}$ Go to the layer menu. Once you select a layer, you will be returned to the BUILD menu.

$\boxed{\textbf{MAIN}}$ Go to the main menu.

3.6 The Cell menu

This menu allows you to define geometry cells, and to manipulate them. Since it deals with geometry cells, it shows the current geometry buffer in the grid area. The menu squares have the following effects:

$\boxed{\textbf{MAKE}}$ Same as MAKE in the Build menu.

$\boxed{\textbf{LIST}}$ Pressing this menu square will cause NEWOL to list the geometry cells (including the GEOM buffer) that it knows about. For each cell, it will list the name of the cell, the cell type (GEOM), the height, and the width. If there are more cells defined than will fit on the screen, don't worry, just hit return and the rest of the cells will be shown.

$\boxed{\textbf{APPEND}}$ Pressing this menu square will cause NEWOL to ask you for the name of a geometry cell. It will then ask you to press the pen down on the grid on the location of the lower left hand corner of the cell. NEWOL will then take a copy off that cell and add it to the buffer. NOTE: This is *not* composition. Appending a cell to the buffer takes a copy of the geometry in a cell and integrates it into the buffer's data tree. Once this operation is over, there is no way to tell that any of the boxes in the buffer came from another cell. It is exactly as if the user had ADDed each box by hand in the BUILD menu.

$\boxed{\textbf{NEW}}$ This operation is done immediately prior to editing a cell, and causes the cell to be copied into the GEOM buffer. NEWOL will ask for the name of a geometry cell, using the cell lister menu. NEWOL will then clear out the GEOM buffer, and move a copy of the named cell into the buffer. It also will change the viewing window to scale the new cell to the proper size. Remember that after the new copy of the named cell has been moved to the GEOM buffer, it has no relation to the cell in memory. Editing the buffer will only change the buffer, not the cell from which the copy was made.

$\boxed{\textbf{ERASE}}$ This allows you to erase the cell definition of a cell in memory. The cell's space in memory will be collected for future use. NOTE: A cell cannot be erased if it is used in the definition of any composition cells (i.e. if it is called by any other cell).

$\boxed{\textbf{CLEAR}}$ Clears the geometry buffer. The user is warned if the buffer contains a cell which has been modified but not saved.

$\boxed{\textbf{PLOT}}$ Plots the contents of the GEOM buffer to a HPGL plotter.

$\boxed{\textbf{ZOOM}}$ Same as ZOOM in Build menu.

$\boxed{\textbf{MAIN}}$ Return to main menu.

3.7 The Elec menu

This menu provides means to geometrically transform boxes, and to name nodes and ports. The latter two features are primarily used by subsequent tools (extractors, automatic routers, DRCs), as means to identify boxes.

$\boxed{\textbf{ELEC}}$ Names a box, prompting for a string. The name is saved in the .CIF file using the "`86 <name> ;`" extension.

$\boxed{\textbf{XFORM}}$ Similar in concept to XFORM in the composition buffer. See the description there. There are no XFORM ONE or SEL modes, transformations apply to ALL the selected object(s).

$\boxed{\textbf{RULER}}$ Enter yardstick mode; press a menu square to exit.

$\boxed{\textbf{PORT}}$ Define one edge of a box as a *port*. When display ports mode is set, ports are visible as dashed edges in the geometry buffer, and as solid lines in the composition buffer (in the latter, they are visible even when the geometry cell is poked shut). The PORT command will wait for a box to be selected, then prompt for which edge (N,S,E,W) and a port name.

$\boxed{\textbf{DRC}}$ Enter the design-rule checker.

$\boxed{\textbf{BUILD}}$ Return to Build menu.

$\boxed{\textbf{MAIN}}$ Return to Main menu.

3.8 The Composition Menu

This menu allows the user to compose (by abutment) as many cells as he/she would like. They can have any orientation, and can be repeated many, many times. The composition menu has its own buffer (*COMP*), and its own window to view it with. These are independent of the GEOM buffer and window.

It is very important to understand how NEWOL does its composition. Each composition cell (including the COMP buffer) is made up of $0 - n$ instances of other cells. Each instance is an exact map of that other cell. Therefore, if you later go and change the cell from which the instances get their definition, you will also change these instances, and therefore change the composition cell. The menu squares perform the following actions:

ADD This option allows you to add an instance of either a composition or geometry cell to the buffer. NEWOL will ask you for the name of the cell, and then will ask you to push down the pen on the grid where you would like the lower left corner to be. Don't worry if it's not quite right, you can always move the instance later. NEWOL tries to fit as much of the name of the cell as possible into the box on the screen.

MOVE This allows you to move instances around the screen. As long as the MOVE menu square is lit, pressing the pen down on the grid will select an instance to move. Once the pen is lifted, the instance will follow the pen around until it is pressed down again. Unlike the build menu, you can only move one shape at a time. However, the EXECUTE select mechanism still works in the composition buffer.

DEL This allows you to delete instances that are on the screen. While the DEL menu square is lit, selecting a shape on the grid deletes it from the COMP buffer. NOTE: There is *no* undelete buffer for composition. You can simply ADD the cell back.

COPY Allows you to copy an instance, and move the copy around. When you press the pen down again, the copy stops moving, and is added to the buffer. Again, successive presses put down multiple copies; lift the pen to abort.

POKE Pressing the POKE menu square takes you to the POKE sub-menu, which is used to view the contents of subcells in the composition buffer. If the cell is a geometry cell, you will see the geometry in the cell. If it is a composition cell, you will see the cells which make up the composition cell. This operation is recursive, and can be repeated on the newly revealed cells. NOTE: This is just for looking at how the compositions line up, *NO* editing can be done on the contents revealed with a POKE.

MAKE When pressed, MAKE asks you for the name of the cell you wish to make. A copy of the composition buffer is then put into memory with that name. NOTE: The bounding boxes of compositions are always the minimum size. Unlike geometry cells, they cannot be set.

DRAW Enter the point-to-point routing menu (see below).

NEW This does just the same as the NEW function on the Cell menu. It clears the composition buffer and moves a copy of a new cell into the buffer. Also, it scales the window to the proper size. This is the first step to editing an existing cell.

OTHER Since there are too many things to do for composition, there are two comp menus. Hitting OTHER causes NEWOL to alternate between them.

MAIN Returns back to main menu.

OTHER menu

XFORM This menu square allows you to rotate and mirror the instances in the COMP buffer. Note that when this menu square is lit that a word appears in the right of the status area. This shows the current transform, and can be changed by repeatedly hitting the menu square. The transforms are: Rotate by 0, 90, 180 and 270 degrees, and mirror the X coordinates, mirror the Y coordinates, mirror in the line X=Y, and mirror in the line X=-Y. The transforms are all done around the center point of the instances, and the transform is applied by pressing the pen in the grid area inside of one of the instances. Since all of the transforms look like one of two boxes, NEWOL uses different colors to denote the different transforms. Here are the transforms (and the strings displayed):

```
Rotations:   null = 0 = white;  R90  = red;
                 R180 = green;  R270 = amber;
Reflections: m(x=0) = MX = blue;  m(x=-y) = M-XY = purple;
             m(y=0) = MY = cyan;  m(x=y)  = MXY  = brown;
```

ERASE When called, ERASE asks for the name of a cell to erase. It will remove the cell entirely from memory. NOTE: Like before, if the named cell is used elsewhere, NEWOL will not let you erase it.

ZOOM Just like before, but it zooms the composition window.

$\boxed{\textbf{ARRAY}}$ A fast means to lay down multiple instances of a cell. ARRAY asks for a "basis": two cells must be selected (by pressing the pen on them), and a replication factor, N, is requested. Then, N-2 instances of the *second* basis cell will be placed, separated by the distance between the two basis cells.

$\boxed{\textbf{NAMES}}$ Analogous to the NODES command in the geometry buffer, NAMES allows unique names to be given to different instances of the same cell. The instance name is written out in the .CIF file using the `85 <name> ;` extension.

$\boxed{\textbf{LIST}}$ This option lists all of the composition cells that NE$_{\text{W}}$OL has stored in memory. The listing is different than that for geometry cells, though. For each cell, the name, cell type, height, and width is listed. Also is listed the instances that are in the cell. The instances are listed in cyan, and are inset. Also, for each instance, its transform is also listed.

$\boxed{\textbf{PLOT}}$ Plot the contents of the composition buffer.

$\boxed{\textbf{OTHER}}$ Switches to the other comp menu.

$\boxed{\textbf{MAIN}}$ Back to the main menu.

3.8.1 The Poke sub-menu

The Poke menu provides a means to view the contents of subcells within the composition buffer. It is important to remember that cell *editing* is only performed at the *top* level of the hierarchy. Thus, you cannot modify the contents of cells that you "open up" via the poke command; rather, you must "NEW" those cells into the appropriate buffer, and modify them there.

$\boxed{\textbf{POKE}}$ This is the default action; the first time a cell is POKEd, it is opened up by one level in the hierarchy, revealing any cells that it contains. If these sub-cells are POKEd, *they* open up to reveal their contents. Finally, if a geometry cell is POKEd open, and is then POKEd again, it closes itself; this repeats all the way up the hierarchical tree (on successive POKEs), until the original cell is POKEd shut.

$\boxed{\textbf{TOP}}$ Immediately closes all poked open cells.

$\boxed{\textbf{BOTT}}$ When pressed the first time, all cells are opened *down to the geometry level.* The geometry cells are not opened until BOTT is pressed again; at this point, all geometry is displayed. Subsequent presses alternate between these two states.

RULER Enters yardstick mode; press any menu button to exit.

PLOT Plot the contents of the composition buffer on an HPGL plotter. The "poked" status of the cells is the same as they appear on the screen. Also, the current directory is searched for a LABELS.TEXT file, which contains the (x,y) position (in lambda) and a label string to plot.

EDIT This command provides a shortcut to editing geometry cells. After pressing the menu button, select a geometry cell to be edited. The cell will automatically be "NEW"ed in the geometry buffer, which will be made the current buffer. When the editing is complete, press "POP" in the Build menu (in the menu square which used to be occupied by "MAKE"); the cell will automatically be Made, and the user is returned to the composition buffer. Should the user wish to abort this editing sequence, going from Build to Main causes the POP button to revert to MAKE.

LAYER Causes a sub-menu to be entered in which only selected layers are displayed. This is often convenient for design rule checking. Pressing the pen in the main display window causes a return to the Poke menu. NOTE: any cells saved to a .CIF file while in this menu will only contain the mask layers that were being displayed.

XTRACT Enters the network extraction sub-menu.

COMP Return to the composition menu.

MAIN Return to the main menu.

3.8.2 The NEWOL extractor

To provide connectivity and topology verification, NEWOL contains an extractor that locates all transistors and generates a netlist of their interconnection. This netlist can then be compared, via the NETCMP program, to that generated via a schematic capture system.

In order to maximize its versatility, NEWOL's extractor operates on a *flattened* representation of the geometry. Memory limitations consequently limit extraction to chips of under 20K transistors. For chips within this limit, the extractor is exceptionally fast, making use of box-level and cell-level information together with a bin-sorting algorithm to yield a speedup factor of 100+ over pixel-based extractors.

Once the chip has been extracted, the connectivity information that has been generated can be displayed interactively. Entire electrical nodes can be highlighted, and estimates of their area and electrical capacitance can be made.

XTRACT Extract the cell currently loaded into the composition buffer. It is *vital* that this cell be freshly loaded (via a "NEW" command) into the buffer. You will be prompted for a file name for the .NTK output file.

FLATTEN Write a .CIF file with a flattened representation of the cell loaded in the composition buffer.

NODES Toggle the display style *after* the extraction. When enabled, the actual node number of each box is displayed. When disabled, a conventional display style is selected.

FLASH A convenient way to find a particular node, FLASH prompts for a node number, then flashes that electrical node. Nodes can also be flashed by pressing the pen down on the geometry itself. Whenever a node is flashed, some information about the node (fanout to n- and p-transistors, and optionally, area and capacitance) is also displayed.

AREA Enables the computation of area and capacitances of FLASHed nodes.

SHORT Treats all P+/N+ diodes as short circuits. These structures arise (legally) in CMOS ohmic conacts. Extracting the chip twice (once with this switch set), and comparing it against itself, ensures that all such contact structures have both sides connected to a power source.

DRC Enter the composition buffer DRC facility.

COMP Return to composition menu. All extraction data structures are freed when the Xtract menu is exited.

MAIN Return to main menu.

3.8.3 The point-to-point router

NEWOL has a manual pont-to-point router, intended mainly for high-level routing such as pad-wrapping, where density considerations are offset by a desire for fast and easy wiring. The router is available only in the DRAW sub-menu of the COMP menu.

The router uses *wires* as a primitive data structure. They require separate generation and editing paradigms than the boxes used everywhere else.

The display and specification properties of routing wires are set primarily by keyboard commands, including "W" (toggle fleshed-out routing display), "M" (toggle Manhattan routing).

DRAW Draw a wire by specifying end points of line segments that make up the centerline of the wire. The wire is completed by lifting the pen (the end of the wire is the last point pressed). By definition, a wire is the set of points within 1/2 of the *width* from the centerline. In Manhattan mode, only right-angle bends are permitted; otherwise, arbitrary angles are permitted (the extractor, however, will ignore any non-Manhattan geometry).

LAYER Select the layer for DRAW.

DEL Delete an entire wire, by pressing the pen on one of the line segment end points.

EDIT Edit a routing wire. In Manhattan mode, points or centerlines can be selected and moved; in non-Manhattan mode, only vertices can be edited. The connectivity of the wire is maintained.

WIDTH Set the width of the wire in lambda.

3.8.4 The NEWOL design rule checker

NEWOL provides a simple geometric design rule checker. Like the extractor, it does not operate continuously in the background, but rather, exploits the high-level box internal representation to operate quickly in a "batch" mode (typical performance is approximately 2500 square lambda per second). Unlike the extractor, the DRC is *NOT* guaranteed to find all the errors the user may have made (design-rule checking is a difficult, and presently open, research topic). The philosophy for the DRC is the following: (i) finding some errors is more useful than finding none, and (ii) only true errors should be reported. Consequently, the DRC fails by "errors of omission" (i.e. not finding or reporting particular errors), rather than by errors of commission (reporting a valid construct as an error). The DRC is presently being improved, so this documentation is possibly incomplete. For a more complete specification of the DRC configuration file, see the System Maintainer's Guide.

There are four categories of rules that are required. These are, and NEWOL's current status regarding them:

Separation Rules. These rules specify some minimum distance between boxes on two particular layers. A typical example is that disconnected pieces of polysilicon must be separated by 2λ. NE$_{\mathrm{W}}$OL reads a configuration file of the design rules. The syntax of this rule is:

```
separation <layer1> <layer2> <distance>
```

A variant on this rule provides a logical-AND between two layers to generate "<layer1>" above. For example:

```
separation2 diff select well 5
```

guarantees that n-diffusions (green (diff) surrounded by select) are at least 5 lambda from a well edge.

Overlap rules. These rules specify that some layer must be entirely contained within some other layer. For example, contact cuts must be surrounded by boxes corresponding to the two layers which are to be electrically connected:

```
overlap poly polycut 2
```

guarantees that a poly surround of 2λ around each cut to poly. Again, binary interactions are specified by:

```
overlap2 diff well well 5
```

which specifies that all active within the well must lie within the well by at least 5λ. *Note that this applies to ohmic contacts to the well, in addition to any p-diffusions.*

Width rules. These are single-layer rules, and specify the minimum width of a particular layer. For example:

```
width diffcut 2
```

specifies that $2\lambda \times 2\lambda$ is the minimum size for a cut to active.

Extension rules. These rules specify composite rules such as the distance that poly must extend beyond a transistor channel in order to guarantee the integrity of the channel in the case of mask alignment errors. This rule is currently not tested in NE$_{\mathrm{W}}$OL.

The DRC has two user interfaces: it is accessible through both the geometry buffer and the composition buffer. In general, it is executed by drawing a box around the region to be checked. Any errors found are reported as white boxes drawn at the location of the error. These boxes can then be examined for information regarding the nature of the error. Errors are also reported in a file created in the current directory called `drc.out`.

The menu squares common to the two DRC interfaces are:

RUN DRC Execute the DRC. The user is prompted to draw a box enclosing the area to be checked.

$\boxed{\textbf{CLEAN}}$ Clean up the data structures generated by the DRC. In the geometry buffer, DRC errors are incorporated into a cell's data structure as boxes on a special layer. The node names of these boxes (visible through the ELEC command, below) describe the error. In the composition buffer, special boxes are maintained as part of the buffer, and are automatically cleared when the DRC menu is exited.

$\boxed{\textbf{WELLS}}$ Perform the additional tests required by the *separation2* and *overlap2* rules above. These tests are required for testing well-related rules, but are rather slow to execute.

$\boxed{\textbf{FLASH}}$ Highlight the errors found by the DRC by flashing the white error boxes. As in the extractor, FLASH is terminated by lifting the pen from the tablet.

In the geometry buffer, there is an additional command to inquire about the error in question:

$\boxed{\textbf{ELEC}}$ A short summary of the error is saved as the electrical node name of the error box. Thus, ELEC can be used to display the name.

In the composition buffer, a special command must be used to display the results of a previous DRC:

$\boxed{\textbf{SHOW DRC}}$ When the DRC is first run, this function is executed automatically. However, as the screen is refreshed, the DRC boxes also disappear (since, technically, they are not part of the data structure); SHOW DRC is used to re-display them.

3.9 The Options menu

The last menu accessible from the main menu is the OPTS menu. This menu contains a collection of file management functions, as well as provisions to monitor and modify NEWOL's operation. This is the menu that allows you to save and recover your work, from session to session. NEWOL saves its data in Caltech Intermediate Format (CIF – see Chapter 4 of Mead & Conway), which is directly compatible with the suite of design tools (**cifp**, **cifplot**, **extract**) used at Caltech, as well as the verification/PG-tape generation software used by MOSIS.

$\boxed{\textbf{LOAD}}$ This allows you to read in a file containing one or more cells. These cells may be either geometry cells, in which case they are appended to the list in the *GEOM* buffer, or composition cells, in which case the composition information will be added to the *COMP* buffer list. Any sub-cells in a composition cell will also be read in and added to the appropriate buffer lists (remember that you have to enter these buffers and NEW the cells before they will appear on the screen for editing). Thus, if your finished project is in the form of a single cell (which is required by MOSIS), it can be read in simply by pressing LOAD, and typing in the name of the file in which it is found. Filenames of up to 12 characters are permitted in the CHIPmunk operating system. NEWOL will automatically append a `.CIF` extension to the name you supply, if no extension is given. By default, only the current directory will be searched; full pathnames must be supplied if the file is not in this directory.

SAVE This command saves NEWOL's internal data representation in a file as CIF. Cell revision numbers and last modification dates are also updated as required and written to the file. NEWOL will prompt for a filename to write into (again, 12 characters maximum), and a cell name (either composition or geometry) to save. If the file already exists, NEWOL will rename the original to file.CFO, as a backup feature. Please note: only cells which you save explicitly will in fact survive when you exit NEWOL (however, the program *will* keep track of your work, and warn you if you try to exit with unsaved changes). To save the state of the entire system, instead of a single cell, use the DUMP command (below).

PORTS This command creates a file PORTS.WOL that contains the (heirarchical) portnames of all instanced cells, as well as their absolute (x,y) coordinates (in lambda, with (0,0) in the lower left corner of the chip). This is primarily useful for interfacing with an external router.

MEM Prints the current state of NEWOL's memory manager, including the (user-set) memory threshold.

DIR Enters the file-lister environment (see above), to display the `.CIF` files found in a particular directory. This feature has been largely replaced by the file-lister in LOAD/SAVE, and by the ability to perform shell-escapes.

OPTS Enters a CAGED-style textual options specification sub-menu. These are the options that are written to the INFO.WOL configuration file. The options, some typical values, and their meaning, are:

Lambda:	150	sets λ in hundreths of microns
Display style:	0	see *display styles*
Short cell name prompt:	Off	if On, enter cell-lister environment immediately
Short file name prompt:	On	if On, enter file-lister environment immediately
Auto refresh:	On	see *auto-refresh*
Edge-priority selection:	Off	see *edge-priority selection*
Point-to-point move:	Off	set point-to-point move as default
Disable Hierarchy:	Off	if On, disable hierarchy for select mechanism
Display Bounding Boxes:	On	
Display Ports:	Off	set display ports mode
Manhattan routing:	On	if On, limits DRAW to rectangular geometry
Fleshed-out routing:	Off	If Off, draws centerlines only
Background color:	0	sets gray level of screen background
Memory threshold:	20000	NEWOL will warn when this is reached
CIF symbol start:	1	Symbol number to start numbering at
Technology file:	SCG	

CNFG Writes an INFO.WOL file into the current directory. When NEWOL begins execution it searches for this configuration file in (i) the current directory, (ii) the user's home directory, and (iii) the WOL library directory (/LIB/WOLLIB).

TECH Prompts for the name of a technology file to read in. The default file is specified in the INFO.WOL configuration file. Since different technologies often have different numbers of layers, it is often not possible to change technologies from within NEWOL. The preferred mode is to use the MOSIS program to perform layer conversion, and/or use the `-t<tech>` command-line option to NEWOL.

DUMP Causes all cells, geometry and composition, to be written to a single file, with no top-level call. This is a convenient way to save all of your work at intermediate points in your editing session. For submission to MOSIS, however, CIF must be generated via the SAVE command.

MAIN Return to MAIN menu.

3.10 The DBUG Menu

This menu, accessible from the main menu, is primarily intended for emergency debugging and problem identification. There are very **DANGEROUS** operations in this menu, and they should not be fooled around with, unless you think you know what you are doing. The exception to this caveat is the WCOMP command; it is located in this menu because of its fairly specialized nature.

GEOM Dump the data structures of the geometry buffer.

$\boxed{\textbf{GCELL}}$ Dump the data structures of the geometry cell dictionary list.

$\boxed{\textbf{COMP}}$ Dump the data structures of the composition cell dictionary list.

$\boxed{\textbf{CCELL}}$ Dump the data structures of the composition buffer.

$\boxed{\textbf{EMERG}}$ Take emergency buffer action. The entire contents of the geometry or composition buffers can be destroyed. No attempt is made to reclaim any memory. This command is only intended for use in truly dire straits, and to buy enough time to save what you can, then exit NEWOL.

$\boxed{\textbf{OPTS}}$ Another path into the OPTS text sub-menu, should the primary path (through the Options menu) become corrupted.

$\boxed{\textbf{MAKE}}$ Another path to MAKE either the geometry or composition buffer.

$\boxed{\textbf{WCOMP}}$ A means to individually save all the cells loaded in NEWOL. This is useful for the WOLCOMP composition system, which requires files to contain single geometry cells. The user is prompted if only newly modified cells are to be dumped, and if only geometry cells should be considered.

$\boxed{\textbf{FLUSH}}$ Re-initializes the memory manager. Effectively, this wipes out the NEWOL data structures, clearing the cell dictionaries and buffers, and basically resetting NEWOL to a power-up state.

4 The NETCMP network comparator

The netlist files generated by the NEWOL extractor can be compared to a reference netlist (generated, for example, by the LOG/ANALOG schematic capture system) using the NETCMP network comparison program. This program is external to NEWOL, and must be run separately. NETCMP prompts for two .NTK files, and for a file into which to write the results of the comparison.

The comparator works by alternately considering transistors and nodes. These are divided into *equivalence classes*; initially, transistors are divided by type (p and n), and nodes are divided by fanout (number of n and p gates and drain/sources). The algorithm then subdivides the transistors into classes whose terminals connect to nodes which are themselves in the same classes. The node classes are then subdivided according to the transistor classes that connect to them. This procedure is repeated until no further subdivision is possible. Ideally, at this point, all equivalence classes have exactly two elements (one from each circuit being compared). In some cases, *automorphisms* exist where an equivalence class contains an even number of indistinguishible elements (e.g. parallel driver transistor structures) – by default, automorphisms are not flagged as errors. A true error exists if the subdivision algorithm wants to fracture an equivalence class into classes with odd numbers of elements. These error classes are printed *in their entirety* when the algorithm reaches its fixed-point. Nodes are printed with their fanouts (number of n-gates (ng), n-source/drains (ns), p-gates (pg) and p-source/drains (ps)).

The most effective way to use the comparator output is the following:

1. Make sure that both circuits contain the same number of transistors.
2. Make sure that both circuits contain the same number of nodes.
3. Get the fanout of the power nodes right. These are easy to find: V_{dd} connects to a *large* number of p-source/drains, and *Gnd* connects to a similarly large number of n-source/drains.
4. Look for patterns in the outputs. In general, it is easier to locate nodes with different transistor fan-outs. It is often possible to identify two nodes in one circuit which, if they were connected, would sum up to one node in the other circuit. Then go looking for missing wires in the geometry or schematic.
5. Look for differences between topologies and logic. For example, on a logic diagram, the two inputs of a NAND gate are equivalent. In a VLSI layout, however, they are not at all the same (think about the pull-down chain).

6. The hardest problems to find are in highly symmetric networks. For example, taking the wrong output from a flip-flop is a serious logical error, but a subtle topological problem.

Sample session with NETCMP:

```
NETCMP #1.04; compiled Sep 5, 1985 5:12 pm
  Enter output file name [/spool/apple.text/MASS]: (cr)
Enter input filename 1 : [.NTK] csrltcell
Parsing file: csrltcell.ntk
Found:  9 N Transistors,    9 P Transistors, 18 Total Transistors,
        11 Non-orphan Nodes, 0 Orphans,       11 Total Nodes.
Enter input filename 2 : [.NTK] csrltcell.lntk
Parsing file: csrltcell.lntk
Found:  9 N Transistors,    9 P Transistors, 18 Total Transistors,
        11 Non-orphan Nodes, 0 Orphans,       11 Total Nodes.
netcmp> ?
NETCMP command summary
   auto -- toggle display of automorphisms
   comp -- compare the networks
   quit -- exit the program
   help -- print this list
   dbug -- toggle debug mode
   slow -- ALWAYS consider classes for subdivision
   long -- print extra long results
   dump -- dump data structure
   node -- print info about node
   read -- read .ntk files
netcmp> comp
Iteration: 1; Transistor classes = 9 (+7); Node classes = 7 (+0)
  Memory left = 1762K; used = 23K

Iteration: 2; Transistor classes = 9 (+0); Node classes = 7 (+0)
  Memory left = 1762K; used = 23K

************  results of fixed-point iteration **************
1 difference found.
netcmp> quit
```

Sample output from NETCMP:

```
Parsing file: netcmp.NTK
```

```
Found:  9 N Transistors,    9 P Transistors, 18 Total Transistors,
        11 Non-orphan Nodes, 0 Orphans,       11 Total Nodes.

Parsing file: netcmp2.NTK
Found:  9 N Transistors,    9 P Transistors, 18 Total Transistors,
        11 Non-orphan Nodes, 0 Orphans,       11 Total Nodes.

Iteration: 1; Transistor classes = 9 (+7); Node classes = 7 (+0)
Iteration: 2; Transistor classes = 9 (+0); Node classes = 7 (+0)
************  results of fixed-point iteration **************

-----------------------------------------------
   (2)  Node: _5 fanout:     pg=0 ng=0 ps=1 ns=2
   (1)  Node: MINUS fanout:  pg=0 ng=0 ps=1 ns=1
   (1)  Node: PLUS fanout:   pg=2 ng=2 ps=2 ns=2
   (2)  Node: _7 fanout:     pg=2 ng=2 ps=2 ns=1

1 difference found.
```

(Interpretation of above results: a source/drain of a transistor connecting to node 5 in the layout SHOULD have been connected to node 6 instead. This assumes that circuit (1) is the schematic representation and is correct.)

5 Sending CIF files to MOSIS

MOSIS is the DARPA silicon broker, and accepts CIF files directly. Certain administrative information must be pre-pended to the file, to confirm authorization and to identify projects. To generate this information automatically, a filter program called MOSIS exists. MOSIS accepts `.CIF` files as input, and generates `.MOSIS` files which are ready to be e-mailed to MOSIS.

The syntax of the program is:

MOSIS <-flags> <cif file name>[.cif]

The program will automatically append .CIF to the input file name, and generate an output filename (<input file name>.MOSIS) if none is supplied. In general, no flags need be supplied, and the program will automatically generate a reasonable set of defaults for the user name, identification, e-mail address, etc. The flags are used to change these defaults:

-m<string> – by default, the mailing address is `ces@csvax.caltech.edu`. The "m" flag replaces that with <string>.

-f<string> – by default, the user's full name is <chipmunk user name>. The "f" flag replaces that with <string>.

-n<string> – by default, the project name is take from the filename, and is <cif file name>. The "n" flag replaces that with <string>.

-d<string> – The "d" flag sets the one-line project description to <string>. By default, no description is generated.

-i<string> – set input technology (SCG, SCGE or CBPM2), if different from output technology.

-t<string> – by default, the technology is generic CMOS-bulk (SCG). If translation to/from another process is desired, use "-i<tech name>" to specify the input technology, and "-t<tech name>" to specify the output technology. Valid technologies are SCG, SCGE, and CBPM2. Talk to the system administrator to find out what these are.

-q integer – Makes a special request for a certain number of bonded parts. Large requests should be cleared with MOSIS first.

-b<string> – Tell MOSIS to bond this project using the pinout of project <string> (specify either the project-id or fab-id of the previous project).

The `.MOSIS` file that is generated is then ready to mail to MOSIS. Currently, the way to do this is to transfer the file to a UNIX machine with ARPANET access, and type **mail mosis@cheeta.isi.edu < file.MOSIS**

6 The WOLCOMPII composition system

WOL provides a textual composition system, embedded in the Pascal language. The user writes a program that loads the component geometry cells, then composes them together. The WOLCOMPII composition system is intended to be a more user-friendly interface to an earlier composition system called WOLCOMP (written by Carver Mead; see the next section). In order to use these routines, a PASCAL program must search and link the following modules:

```
import
   $search '/LIB/WOLLIB/WOLCOMP/WOLCOMP'$  wolcomplib,
   $search '/LIB/WOLLIB/WOLCOMP/WOLCOMPII'$  wolcompii ;
```

A brief description of these routines follows:

PROCEDURE wolcomp_init(lamb : integer);

Initialize the WOLCOMP and WOLCOMPII libraries to generate CIF with λ = lamb / 100 microns. This must be the first call to the system, and is generally followed by commands to load the various geometry cells:

PROCEDURE load (filename : strg);

Reads a file containing a composition cell, and inserts all contained geometry cells into an internal dictionary. The global string variable `prefix` is prepended to `filename`; this is convenient while searching some library directory. The `load` command is identical to the WOLCOMP `load_library` procedure. Once all the cells are loaded, they can be composed into new cells:

PROCEDURE define_cell (cell_name : strg);

Begin the definition of the cell "cell_name". The current cell placement origin (x0,y0) is set to the lower left corner (0,0), and transformation, composition, and overlap modes are reset. The definition is terminated with the **endef;** statement.

PROCEDURE place (cell_name : strg);

Within a cell definition, position an instance of cell "cell_name" at position (x0,y0). The user can manually update (x0,y0) by hand as in WOLCOMP (perhaps making use of the "`cell_width`" and "`cell_height`" procedures below), *or* they can be updated automatically according to the current *composition mode*, and "overlap" distance (see below). For example, if we are currently composing horizontally, `x0` will be updated to `x0+cell_width-overlap`, and `y0` will be left undisturbed. If the global variable `include_geometry` is set to `FALSE` before a call to `place`, the coordinate (x0,y0) is updated, but *no geometry is actually drawn*; this is often useful in debugging to reduce the amount of output CIF generated.

PROCEDURE hcompose;

Set the current composition mode to horizontal. Successive calls to `place` will create a horizontal row of cells.

PROCEDURE vcompose;

Set the current composition mode to vertical.

PROCEDURE nocompose;

Set the current composition mode to "do nothing". This means that x0 and y0 are unaffected by a call to `place`. This is the default initial composition mode of WOLCOMPII.

PROCEDURE overlap (lambda : integer);

Sets the overlap between adjacent cells. For example, if `vcompose` is set, and `overlap(10)` is in effect, then every call to `place` will result in `x0` (the position of the *next* cell to be placed) to be 10 lambda to the left of the new cell's right side.

PROCEDURE make_array (cell_name : strg; count : integer);

Equivalent to the loop: `for i := 1 to count do place(cell_name);`

PROCEDURE transform (xfrm : integer);

Sets the current transform to `xfrm`, which can take the value of the predeclared constants:

```
R0 = 0;
R90 = 1;
R18 = 2;  R180 = 2;
R27 = 3;  R270 = 3;
M0  = 4;              MX  = 4;
M90 = 5;
M18 = 6;  M180 = 6;    MY  = 6;
M27 = 7;  M270 = 7;    MXY = 7;
```

This transform *remains in effect* until another transform is specified, or another cell is defined; all calls to `place` will employ the current transform. The initial transform in WOLCOMPII is R0

PROCEDURE vpad (cell1, cell2, cell3 : strg; to_width : integer);

Takes one instance of `cell1` and vertically adds as many instances of `cell2` as are required to give a cell of height `to_width` in lambda. It is legitimate to partially overlap instances of `cell2`. The resulting cell is defined as cell3. This function is useful in generating parameterized cells that must "stretch" to abut against some constraining cell. This function must not be called within a `define_cell` command, as cell definitions cannot be nested.

PROCEDURE hpad (cell1, cell2, cell3 : strg; to_width : integer);

Same as `vpad`, but the composition is horizontal. (identity transform).

Once the project composition is complete, WOLCOMP is terminated with a call to:

PROCEDURE shut_down;

Other functions that are often useful include:

FUNCTION cell_width (cell_name : strg) : integer;

Returns the width of the cell in internal WOLCOMP units, which makes it suitable for addition/subtraction from variables such as (x0,y0).

FUNCTION cell_height (cell_name : strg) : integer;

Similar to "cell_width", but returns the height of the cell.

FUNCTION make_cell_name (root : strg; i : integer) : strg;

Returns a string which is the concatenation of the string `root`, and a string containing the integer `i`.

PROCEDURE push (xi, yi : integer);

Push a pair of integers onto a stack. This is very useful for recording some (x0,y0) position that you want to return to after going off and doing something else.

PROCEDURE pop (var xi, yi : integer);

Pop a pair of integers off the stack, into the variables xi and yi.

PROCEDURE include_file (fname : strg);

Include the contents of the file `fname` directly into the output CIF file. This operation can be safely performed only while no cell definition is in progress.

6.1 Sample program

See the PADGEN portion of this manual.

7 The WOL pad library and pad generator

A set of library pads exist for NEWOL. A complete pad family, containing both analog and digital pads is located in /LIB/WOLLIB/PADLIB. These pads are implemented in double-metal 3u feature-size CMOS, and provide a 150u square bonding area. The pads, and their applications, are:

INPAD	Input pad, 200Ω resistance, and static protection.
INORPAD	Input pad, static protection, no resistor.
METALPAD	Bare metal pad, no static protection.
OUTPAD	Non-inverting digital output pad, 50Ω drive.
FOLLOWPAD	Wide range follower to drive analog signals
VDDPAD	Provide Vdd to chip core and pad ring
GNDPAD	Provide Gnd to chip core and pad ring
CVDDPAD	Provide Vdd to chip core only
PVDDPAD	Provide Vdd to pad ring only
CGNDPAD	Provide Gnd to chip core only
PGNDPAD	Provide Gnd to pad ring only
CORNPAD	Corner of the pad ring
SPCPAD	No pad, just run pad ring and protection

A WOLCOMP module that implements a pad generator is also available: link your program with /LIB/WOLLIB/PADLIB/PADGEN. A sample program is shown below (it is taken from /LIB/WOLLIB/PADLIB/PADSAMPLE.TEXT):

```
program sample (input, output);
import general_2, graphics, misc, hpib_2, newci,

   { N O T E:   you will need to 'perm /lib/wollib/wolcomp'
                             and 'perm /lib/wollib/padlib/padgen'

               BEFORE trying to run this program!!!
    }
   $search '/LIB/WOLLIB/WOLCOMP'$  wolcomplib,
   $search '/LIB/WOLLIB/PADLIB/PADGEN'$  pad_gen ;

var my_pads  :  pad_spec;    { this is the structure used to specify to
                               the pad generator what pads you want, and
                               where they should go}

begin
   set_up;  lambda:=150;

   prefix:= '/lib/wollib/padlib/';
   getfile('cornpad');
   getfile('padwire');
```

```
  getfile('outpad');            { this is a digital output pad }
  getfile('inpad');             { input pad with static resitor --
                                  use this as an analog output pad }
  getfile('inorpad');           { input pad with no resistor }
  getfile('cvddpad');           { VDD pad for chip core - your circuit }
  getfile('pvddpad');           { VDD pad for PAD frame }
  getfile('vddpad');            { VDD pad for core and PAD frame }
  getfile('cgndpad');           { GND pad for chip core - your circuit }
  getfile('pgndpad');           { GND pad for PAD frame }
  getfile('gndpad');            { GND pad for core and PAD frame }
  getfile('followpad');         { Analog follower pad }
  getfile('plnpad');
  getfile('spcpad');            { "space pad" = no pad at all }

  with my_pads do begin
     xnum:= 2;                   {  NUMBER OF PADS ON TOP/BOTTOM }
     ynum:= 6;                   {  NUMBER OF PADS ON LEFT/RIGHT }
     olap:= 10;                  {  magic number -- do not change}
     corner:= cell('cornpad');   {  also magic }

{ define pads along each side:  north and south lists are left to right }
     north[1]:= c('pvddpad');
     north[2]:= c('cgndpad');

     south[1]:= c('cvddpad');
     south[2]:= c('pgndpad');

{ east and west lists are bottom to top }
     east[1]:=  c('plnpad');
     east[2]:=  c('inorpad');
     east[3]:=  c('plnpad');
     east[4]:=  c('outpad');
     east[5]:=  c('plnpad');
     east[6]:=  c('inpad');

     west[1]:=  c('inpad');
     west[2]:=  c('plnpad');
     west[3]:=  c('outpad');
     west[4]:=  c('plnpad');
     west[5]:=  c('inorpad');
     west[6]:=  c('plnpad');
  end;

  gen_pads(my_pads, 'samplepads');   { this actually does the work,
                                       and returns a cell called 'samplepads'}

  {alternate procedure for WOLCOMP users:

  surround_with_pads('mycore', my_pads, 'newcell');

          -- creates cell 'newcell' by wrapping pads around 'mycore'

  OR

  pad_wrap ('mycore', my_pads, 'newcell', 'routing');
```

```
          -- creates cell 'newcell' by wrapping pads around 'mycore',
          PLUS reads the routing wires in file 'routing' and incorporates
          them into cell 'newcell'.  This is most useful when repeatedly
          compiling some system in which the routing remains fixed.
          After creating the routing by hand the first time (in NEWOL),
          edit by hand the CIF file, putting the routing wires into
          file 'routing'.  You can now re-compile the entire project
          without ever having to re-route those wires.

   }


shut_down;
end.
```

8 The WOL PLA tools

A simple PLA generator has been built using WOLCOMP. A textual PLA personality file is written, and a PLA generated in a user-specified technology. Two interfaces are possible: a simple precompiled program that reads the `.pla` file and generates a `.CIF` file containing the PLA, and a Pascal procedure interface for use in the user's own WOLCOMP programs.

The PLAs generated have a simple form. The AND-plane is on the left, with inputs along the bottom. Each literal is presented in uncomplemented form (and is buffered and complemented in the PLA). The OR-plane lies on the right, with its uncomplemented outputs along the bottom. The reader is referred to documentation within the directory `/LIB/WOLLIB/PLAGEN` for specific details regarding the different PLA technologies (i.e. location of power rails, clock requirements, etc.).

8.1 The WOLCOMP interface to PLAGEN

To access this module, your program must contain the following import text:

```
import
   $search '/LIB/WOLLIB/WOLCOMP/WOLCOMP'$  wolcomplib,
   $search '/LIB/WOLLIB/WOLCOMP/WOLCOMPII'$  wolcompii,
   $search '/LIB/WOLLIB/PLAGEN/PLAGEN'$  plagen;
```

The interface to the module is the following (for an example of the use of these procedures, see the WOLPLA program below):

```
EXPORT

const
      MAXPLAROWS = 50;   { maximum number of rows in pla }
      MAXPLACOLS = 50;   { max. number of cols. in and/or planes }

type
      pla_plane =  array [1..MaxPlaRows,1..MaxPlaCols] of char;

var
   num_pla_rows, or_width, and_width  :  integer;
   or_plane, and_plane :  pla_plane;
```

```
PROCEDURE load_placells (tech : strg);
   { load cells from directory /lib/wollib/plagen/<tech>  }

PROCEDURE make_pla (cellname, filename : strg);
   { read <filename> for PLA personality, generate cell <cellname> }

PROCEDURE read_pla_file (filename : strg; var and_plane, or_plane : pla_plane);
   { read PLA personality into 'and_plane' and 'or_plane', set
     global variables 'num_pla_rows', 'or_width', and 'and_width' }

PROCEDURE build_pla (cellname : strg; and_plane, or_plane : pla_plane);
   { use 'and_plane', 'or_plane', and global variables 'num_pla_rows',
     'or_width', and 'and_width' to generate PLA in cell <cellname> }
```

8.2 PLA personality file format

A very simple file format is used to program the PLAs. The first line contains an integer representing the number of rows (implicants) in the PLA. A textual block then follows, one row per implicant, one column per literal, to define the AND plane. A "1" represents an uncomplemented literal, a "0" a complemented literal, and an "X" a dont-care (that literal does not appear in a particular implicant). A blank line separates the OR-plane specification, which is similar, but only "1" and "X" are allowed. Obviously, all rows in the AND-plane specification must contain the same number of characters, as must all rows in the OR-plane specification.

As an example:

```
4
.001
0.11
10..
1.00

1..
.1.
..1
11.
```

8.3 The WOLPLA program

This program is executed by typing `/LIB/WOLLIB/PLA/WOLPLA` (you may need to type `USEWOLCOMP` first). The user is prompted for (i) the technology for the PLA, (ii) the programming file for the PLA, and (iii) the cell name and CIF file name for the generated PLA.

The listing of the program follows:

```
program wolpla(input,output);

import
   $search '/LIB/WOLLIB/WOLCOMP/WOLCOMP'$  wolcomplib,
   $search '/LIB/WOLLIB/WOLCOMP/WOLCOMPII'$  wolcompii,
   $search '/LIB/WOLLIB/PLAGEN/PLAGEN'$  plagen;

var
   tech_name,      { technology name for PLA }
   file_name,      { PLA personality file }
   cell_name       { cell name of generated PLA }
         : string[255];
   pla_lambda : integer;  { lambda in 1/100 micron }

begin
   writeln ('WOLPLA Version 1.00');  writeln;
   write ('Enter technology name:  '); readln (tech_name);
   write ('Enter PLA personality file name:  '); readln (file_name);
   write ('Enter cell name of generated PLA:  '); readln (cell_name);
   write ('Enter lambda in 1/100 of microns:  '); readln (pla_lambda);

   wolcomp_init (pla_lambda);
   load_placells (tech_name);
   make_pla (cell_name, file_name);

end.
```

8.4 Building a PLA cell library

The user can supply their own cell library to PLAGEN for composition. The `load_placells(tech);` procedure searches in the directory `/LIB/WOLLIB/PLAGEN/TECH` for the following files (each file contains a single cell of the same name as the file):

p_and	if a literal appears uncomplemented in an implicant
p_nand	if a literal appears complemented in an implicant
p_null	if a literal does not appear in an implicant
p_andend	left-hand side of AND-plane
p_buffer	input buffers on bottom of AND-plane
p_buss	power jumper between bottom of AND- and OR-planes
p_andcorner	lower-left hand corner cell
p_or	an implicant does appear in an output
p_ornull	an implicant does not appear in an output
p_orcorner	lower-right hand corner cell
p_orend	right-hand side of OR-plane
p_pcinv	cell between AND- and OR-planes
p_outinv	output buffer along bottom of OR-plane

8.5 The WOL Finite State Machine Compiler

A powerful tool for compiling finite state machines into PLAs has been developed, and is available. Talk to Mass Sivilotti for more info.

9 The WOL Decoder Generator

A WOLCOMP module exists for generating decoders. The user must specify the number of decoded outputs (which must be even), the CIF layer they should be on, and the pitch between them (minimum 36λ). The decoder generator is accessed by "perm"ing /LIB/WOLLIB/DECODER/DECODER, and including the following import text in the WOLCOMP program composing the user's chip:

```
import
$search '/LIB/WOLLIB/WOLCOMP/WOLCOMP'$     wolcomplib,
$search '/LIB/WOLLIB/WOLCOMP/WOLCOMPII'$ wolcompII,
$search '/LIB/WOLLIB/DECODER/DECODER'$ decoder;
```

The cell library of decoder cells is loaded by a call to

PROCEDURE load_decodercells;

The actual decoder is generated by

PROCEDURE make_decoder (name : strg; rows, a, b : integer; ciflayer : strg);

where `name` is the name of the cell that is to be the decoder, `a` and `b` encode the pitch of the decoder outputs (i.e. outputs alternate between being `a` apart and `b` apart), and `ciflayer` is the layer of the outputs (valid choices are PG, MF, and MS, for poly, metal1, and metal2).

A multiplexer generator is also available, with similar syntax:

PROCEDURE make_mux (name : strg; rows, a, b : integer; ciflayer : strg);

10 The WOLCOMP composition system

This section is taken almost verbatim from a document prepared by Carver Mead. For more information, please refer to the **WoLery**. Changes and additions are indicated by horizontal bars.

WoLcomp is a very simple composition system embedded in Pascal. It takes cells from only those CIF files prepared on **WoL**, and composes them into larger cells which can be further composed. Being an embedded language, the user builds a chip by writing a program. There are a number of built-in functions and procedures to make all that a lot easier. To compose a design, one must first get cells—each of which resides as an individual file on your file system—into **WoLcomp**. The procedure that brings in a cell is

procedure getfile(name:strng10);

To get a cell from file `XOR.CIF` the call is **getfile**('`XOR`'); Note that the file must have a `.CIF` extension and you must be prefixed to the directory where the file resides. If you need to access a cell in a different directory, set the global string variable **prefix** to the appropriate directory (don't forget to terminate it with a "/").

EDITOR'S NOTE: It is no longer required that all geometry cells be in separate files. The `load_library` *command described below can accomodate multiple cells per file (as created by NEWOL, particularly by the DUMP command). Since it can also accomodate one cell per file,* `load_library` *is a superset of* `getfile`*, and can be used in its place everywhere!*

WoLcomp builds a symbol table and places the cells in it so they can be accessed by name. A pointer to an entry in the symbol table is of type 'cptr'. To get a pointer to a cell use

function cell(name:strng10):cptr;

To instantiate a copy of the cell, there are two commands:

procedure draw(c:cptr);

places an untransformed instance of the cell pointed to by the argument with its lower left corner at the position given by the global coordinates x_0, y_0.

For situations where cells need to be mirrored or rotated

procedure drawx(c:cptr,tform:integer);

places an instance with its lower left corner at x_0, y_0 and transformed by one of the 8 transforms in the dihedral group defined by mirrors in x and y and rotations by integral multiples of 90°.

The transforms are defined as follows:
0–3 unmirrored cell rotated by the number times 90°.
4–7 cell first mirrored in x, then rotated as above.

When a cell is originally drawn, all of the shapes within it are defined with respect to some *origin* local to the cell. Many embedded systems use the CIF convention that positions an instance of the cell by placing the origin at some x, y location. Since the instance may be transformed by rotation or mirror operators, the placement of the origin depends upon the transformation applied. This approach places the burden of keeping track of these coordinate offsets on the user. **Wol_comp** automatically applies the proper offsets such that all cells are placed with their lower left corner at x_0, y_0 rather than with their origin at that point. In this way, the user need never be aware of the cell origin. Cell instances are treated as tiles, and composition is the proper tiling of the plane. Both procedures **draw** and **drawx** leave coordinates of the the upper right corner of the instance in global variables x_1, y_1.

The main program must start with a call to **start_up** and end with a call to **shut_down**. You may of course, add procedures, variables, etc., above the main program to make your life easier. Since you are thus mixing your own stuff with the **Wol_comp** primitives, your file will be unique to a chip design. I suggest that you name it after the chip.

A large design is always built up by defining a cell as a composition of already defined cells. To define a new cell use

procedure define(name:strng10);

Then put the required **draw** or **drawx** commands for the subcells and end with

procedure endef;

which buttons the whole thing up and adds it to the symbol table.
Note—definitions cannot be nested !

The methodology I have used to build complex chips size is to define all the "real" cells (those with transistors in them) in the context of the others to which they must abut. The odd corners are then filled up with *wiring cells*, to connect the whole thing together. I keep a floorplan with cell names as I go along. The cell names are also the file names. To date, **WoL** is not smart enough to do this floorplanning job for us, but it does give a nicely annotated floorplan plot as it is run. At the end, I write a little program in **WoLcomp** to put the whole thing together. A small example follows to illustrate the technique. Note that there is not a single absolute coordinate in the entire text. All of the placement coordinates are done by the draw commands interacting with global variables. In a pinch, there are times when you will really want to stick a long wire down, and don't want to make an enormous wiring cell in **WoL** just for that wire. For those occasions there is

procedure box(c:strg);

The string argument is the CIF layer name of the box. This procedure creates a box of the appropriate layer with one corner at x_0, y_0 and the other at x_1, y_1. You can make a whole pile of boxes (even a whole design) by using this primitive and the composition functions described above. **WoLcomp** is thus a complete embedded layout language. *However, it was never intended to be used that way*—**WoL** is really a much easier and more idiot-proof way to design cells.

When your design is complete, compile the program and run it! You will be prompted for the name of an output file. Some generally helpful information is displayed on the screen as things progress. If you have forgotten to define a cell, or to get it off the disk, **WoLcomp** will tell you that the cell is not loaded. Once all is well, you will be asked if you want a floorplan plot. *Say yes!* You will find it much easier to debug your composition this way before you start staring at all that layout. **WoLcomp** can plot any cell you have defined, so be sure the entire project is defined as a cell. Floorplans are only plotted to one level—if you want to see the subcells of one of the subcells, you should plot it as a separate cell *right?—O.K.—I'm sorry I asked!* The name of the cell whose floorplan is plotted is shown in the upper right corner of the plot. The names of the subcells are centered in their bounding boxes. Of course, the output CIF generated by **WoLcomp** can be read into NEWOL, and viewed or modified there.

By now, programs in The **WoLery** have been used to do substantial designs in all three technologies. There are many more sophisticated members of the **PooH** family nearly ready to use. Meanwhile, the simple programs described in this document are a good example of which portions of a design are naturally done in an interactive way, and which are more appropriate to do as a language. It sounds silly, but I know of no other system that has got that issue right. I hope the listeners in our far-flung radio audience are enjoying the latest episode in the life of my favorite stuffed animals. *They are so much more interesting than stuffed people!*

A Small Example

The following example—a 6 bit pipeline multiplier—illustrates many of the composition principles used in this style of design. The variables `cell1`, `cell2`, etc., are pointers to cells; `x0`, `y0`, `x1`, `y1`, etc., are integers. All of these variables were declared as globals in the program.

WoLcomp programs all follow the pattern shown. First, all of the needed cells are brought in. Next, cells are defined in terms of smaller cells. Finally, the entire design is defined as a cell, and brought into being with the one naked *draw* command at the end. This arrangement produces CIF conforming to the "one call convention" described in certain obscure books on the subject.

The cells `rend`, `lend`, `lep1`, `lep2`, `rep1` and `rep2` are wiring cells, and contain no active elements. The `lco` and `rco` cells are the VDD and GND pads. The multiplier itself is made out of `plbit` cells, which in turn are made out of subcells. All of the `pd` and `pad` cells are pads with drivers or static protection. The **box** procedure is used to create two horizontal metal wires that tie the VDD and GND busses in the pad ring between the two center `inpad` cells. The number `3` is the wire coordinate relative to the cell bounding box. The `16` is the width of the wire. The `4` is the spacing between the two wires. The original design was 32 bits long, and was obtained by changing the 5 in the FOR loop to a 31. The floorplan generated for this example by **WoLcomp** is shown in the figure.

10.1 WOLCOMP Procedure Calls

In order to link in the **WoLcomp** module, the following file should be searched, linked, and perm'ed:

```
import
   $search '/LIB/WOLLIB/WOLCOMP/WOLCOMP'$  wolcomplib;
```

The following is a brief synopsis of the calling sequence for all the routines in the current **WoLcomp** (version #1.00).

procedure getfile(filename:strg);

Load a single cell out of a .CIF file created with either WOL or NEWOL. The file must contain one geometry cell. "filename" refers to the name of the file that contains the cell. If "prefix" has been specified, that string will be pre-appended to "filename", thus providing a facility to specify a search directory. The cell will be added to the internal symbol table. If the CIF file contains a cell name (i.e. user extension 9), this name will be associated with the cell; otherwise, a cell name will be generated from the supplied filename.

procedure getcompfile(filename:strg);

Load a file containing many cells. These cells will be renumbered to guarantee uniqueness and consistency within each top-level cell, and the top-level cell will be added to the internal symbol table. This is intended for use as a MPC composition facility.

procedure draw(c:cptr);

Draw the cell "c" at the position currently specified by x0, y0. The origin of the cell (usually the lower left corner) will be positioned at x0, y0. Upon return, the global variables x1, y1 will point to the upper right corner of the cell that was just placed.

procedure ndraw(c:cptr; drawit:boolean);

If "drawit" is TRUE, then this is the same as a call to "draw"; otherwise, the pointers x0,y0,x1,y1 will be updated, but no geometry will be drawn. This is very useful for testing compositions of large, repetitive structures.

procedure drawx(c:cptr; tform:integer);

Draw the cell "c" at (x0,y0), but apply a transformation "tform" to it. Possible transformations are:

0 - no transformation;
1 - rotate CCW by 90;
2 - rotate CCW by 180;
3 - rotate CCW by 270;
4 - mirror in the y-axis;
5 - mirror in the y-axis, then rotate CCW by 90;
6 - mirror in the y-axis, then rotate CCW by 180;
7 - mirror in the y-axis, then rotate CCW by 270;

procedure ndrawx(c:cptr; tform:integer; drawit:boolean);

Same as "ndraw", but draws a transformed cell.

procedure plot(cell:cptr);

Plot out a cell, either to screen or to pen plotter.

function cell(name:strg):cptr;

Lookup cell named "name" in symbol table, and return a pointer which can then be passed to "draw", "drawx", etc.

procedure define(name:strg);

Start the definition of a composition cell.

procedure endef;

Close the definition of a composition cell.

procedure box(c:strg);

Draw a box on layer “c” from (x0,y0) to (x1,y1).

procedure set_up;

Should be called first to set up all **WoLcomp** internals. This sets a default “prefix” (null), and “lambda” (1.5 microns). If a different lambda is desired, it should be changed (by assignment to the exported variable **lambda**) <u>after</u> **set_up** is called. Remember: **lambda** is in hundreths of microns. Similarly, **prefix** should be set after **set_up** is called.

procedure shut_down;

Closes the output file, and optionally plots a floorplan for a generated cell. The floorplan can be drawn on either the screen or a pen plotter. Also, the internal symbol table can be displayed (useful for seeing how large cells are).

procedure dumplist;

Prints out the internal symbol table. This is primarily a debugging aid for **WoLcomp** maintainers.

procedure set_mbb(llx,lly,urx,ury:integer);

Forces the MBB of a composition to be set to the specified values. This is primarily useful for handling non-WOL generated CIF, in conjunction with the **cif_type** attribute.

procedure load_library(filename:strg);

Reads a file containing a composition cell, and inserts all contained geometry cells into a “library table”. If the global variable **search_library** is **true**, any subsequent calls to cells with names that match that of a cell in the library will cause a call to the library cell to be generated, instead of the supplied cell. This is most useful for commonly maintained cells such as pad libraries, etc.

function lam(x:integer):integer;

Returns the value of **x** lambda in **WoLcomp** internal units. This is the recommended way to incorporate fixed (i.e. non-relative) offsets. (e.g. x0 := x1 + lam(4); would cause the two cells to be separated by 4 lambda).

function fix_name(fname:strg):strg;

Fix up file names. You should never have to use this.

Routing wire primitives

procedure width (x : integer);
procedure wire (layer : strg; startx, starty : integer);
procedure x (delta : integer);
procedure y (delta : integer);
procedure dx (delta : integer);
procedure dy (delta : integer);
procedure xy (x, y : integer);
procedure dxdy (deltax, deltay : integer);
procedure endwire;

These routines allow routing wires to be included in WOLCOMP-generated cells. All dimensions are specified in WOLCOMP internal coordinates (basically, $1\lambda = 2$ units, and the cell origin is in the lower left corner). The width of the wire is set by **width**; the wire is started by a call to **wire** with the layer (pass the CIF layer name–for example, metal in the SCG technology is CMF–and the start coordinates of the wire). The procedures **x**, **y**, and **xy** set the next corner of the wire to have that x,y and (x,y) coordinate, respectively. The procedures **dx**, **dy**, and **dxdy** are analogous, but are passed the *relative* displacement of the next vertex from the current point. To terminate the wire, call **endwire**.

Exported Variables

```
EXPORT
TYPE
   strg = string[80];
   strng10 = string[10];
   cptr = ^cel;
   sptr = ^subcel;
```

```
   cel = record
          name : strg;
          lx, ly, cifnum  :integer;
          next : cptr;
          contents : sptr;
       end;

   subcel = record
              cell : cptr;
              x, y, xfrm : integer;
              next : sptr;
            end;

   cif_types = (LYON, TELLE, WOL);

VAR
        { cell position, last cell position   }
   x0, y0, x1, y1 : integer;

        { value of lambda in 1/100 of microns }
   lambda : integer;

        { type of CIF being parsed            }
   cif_type : cif_types;

        { should the library be searched for each subcell? }
   search_library : boolean;

        { if true, print cell list on line printer }
   print_cells : boolean;

        { filename prefix to specify directories }
   prefix : strg;
```

11 System Maintainer's Guide

NEWOL reads three files upon commencing execution: (i) a configuration file that sets the run-time user preferences (display style, refresh mode, cell/file menu environment, etc.), (ii) a technology file that binds menu layer names, cif layer names, and display styles to menu squares in the LAYER menu, and (iii) a message-of-the-day file for posting news to users. In addition, other files used by WOL include: (i) a DRC rules file, (ii) a log file for monitoring DRC use, (iii) a log file for monitoring extractor use, (iv) a gripe file to report bugs, (v) an improvements file to report suggestions.

11.1 Configuration files

NEWOL searches for a configuration file `INFO.WOL` in the current directory, then if none is found, in the user's home directory, and lastly in the /LIB/WOLLIB directory. The form of the configuration file is:

```
TECHNOLOGY:  SCGE
LAMBDA:  1.50
DISPLAY STYLE:  1
CELL NAME PROMPTING:  OFF
FILE NAME PROMPTING:  ON
EDGE-PRIORITY SELECTION:  OFF
AUTO-REFRESH:  ON
DISPLAY BOUNDING BOXES:  ON
DISPLAY PORTS:  OFF
MANHATTAN ROUTING:  ON
FLESHED-OUT ROUTING:  ON
BACKGROUND COLOR:  0
MEMORY THRESHOLD:  300000
DISABLE HEIRARCHY:  OFF
POINT-TO-POINT MOVE:  OFF

Written by WOL 1.08  May 13 1987  4:04 pm
User:  CARVER
```

These files are created by the CNFG command, and are order- and case-insensitive.

11.2 NE$_{\text{W}}$OL Technology Files

The technology file binds CIF layer names to internal layer numbers, and these numbers to menu squares, display styles, etc. This file is named `<technology>.WOL`, and is searched for in the current directory, then in the system library.

A sample technology file is contained in /LIB/WOLLIB/SCG.WOL, and is reproduced below:

```
#
#      WOL TECHNOLOGY FILE FOR MOSIS GENERIC GROUNDED SUBSTRATE
#             CMOS BULK PROCESS.   DEC. 7, 1987  MAS
#
#
# File format:
#   layer:  NUMBER  MENU-NAME  CIF-NAME
#   display: DISP-NUMBER  MENU-NAME:  DEFERRED(for cuts) BOUNDARY-COLOR
#            BOUNDARY-STYLE  FILL-COLOR  FILL-STYLE  MENU-SQUARE
#   plotter: MENU-NAME:  BOUNDARY-COLOR  BOUNDARY-STYLE  FILL-COLOR  FILL-STYLE
#   cmap:    COLOR  RED-VALUE  BLUE-VALUE  GREEN-VALUE
#   pattern: PAT-NUMBER ROWS&COLS VALUES...
#
#   Magic pattern numbers:
#        0 - solid fill with fill color
#       -1 - no fill
#       -2 - draw 'X' in box
#       -3 - draw 'X' in box, but black out box first
#    1..?? - user-specified bit patterns (ignores fill-color)
#
layer:  1   POLY   LCPG
  display: 0 POLY FALSE 1  0  0 -1  1
  display: 1 POLY FALSE 1  0  1  0  1
  display: 2 POLY FALSE 1  0  1  0  1
  display: 3 POLY FALSE 1  0  1  0  1
  plotter: POLY 1  0  0 -1

layer:  2   DIFF  LCAA
  display: 0 DIFF FALSE 2  0  0 -1  2
  display: 1 DIFF FALSE 2  0  2  0  2
  display: 2 DIFF FALSE 2  0  2  0  2
  display: 3 DIFF FALSE 2  0  2  0  2
  plotter: DIFF 2  0  0 -1

layer:  3   METAL   LCMF
  display: 0 METAL FALSE 4  0  0 -1  3
  display: 1 METAL FALSE 4  0  4  0  3
  display: 2 METAL FALSE 4  0  4  0  3
  display: 3 METAL FALSE 4  0  4  0  3
  plotter: METAL 3  0  0 -1

layer:  4   WELL   LCWG
```

```
  display: 0 WELL FALSE 3  0  0  -1  4
  display: 1 WELL FALSE 3  0  0   1  4
  display: 2 WELL FALSE 3  0  0   1  4
  display: 3 WELL FALSE 3  0  0  -1  4
  plotter: WELL 6 0 0 -1
  pattern: 1 8 3 T T T T T T T

layer:  5   CUTP   LCCP
  display: 0   CUTP FALSE 7 0 1 -2 5
  display: 1   CUTP TRUE  7 0 1 -3 5
  display: 2   CUTP TRUE  7 0 1  0 5
  display: 3   CUTP TRUE  7 0 1 -3 5
  pattern: 2 4 7070 0707 7070 0707
  plotter:  CUTP 4 0 0 -1

layer:  6   CUTA   LCCA
  display: 0   CUTA FALSE 7 0 2 -2 6
  display: 1   CUTA TRUE  7 0 2 -3 6
  display: 2   CUTA TRUE  7 0 2  0 6
  display: 3   CUTA TRUE  7 0 2 -3 6
  pattern: 2 4 7070 0707 7070 0707
  plotter:  CUTA 4 0 0 -1

 layer:  7   GLASS   LCOG
   display: 0   GLASS FALSE 6 2 6 -1 7
   display: 1   GLASS FALSE 6 2 6 -2 7
   display: 2   GLASS FALSE 6 2 6 -2 7
   display: 3   GLASS FALSE 6 2 6 -1 7
   plotter:  GLASS 8 0 0 -1

layer:  8   MET2   LCMS
  display: 0   MET2 FALSE 15 2 0 -1 8
  display: 1   MET2 FALSE 15 0 8  0 8
  display: 2   MET2 FALSE 15 0 8  0 8
  display: 3   MET2 FALSE 15 0 8  0 8
  plotter: MET2 4 0 0 -1

layer:  9   SEL   LCSG
  display: 0   SEL FALSE  6 0 0 -1 9
  display: 1   SEL FALSE  6 0 0  3 9
  display: 2   SEL FALSE  6 0 0  3 9
  display: 3   SEL FALSE  6 0 0 -1 9
  pattern: 3 8 T T T T TTTT6 T T T
  plotter:  SEL 8 0 0 -1

layer: 10  VIA  LCVA
  display: 0  VIA FALSE 7 2 0 -1 10
  display: 1  VIA FALSE 7 0 7  0 10
  display: 2  VIA TRUE  7 0 0  4 10
  display: 3  VIA TRUE  7 0 7  0 10
  pattern: 4 4 FTFT TFTF FTFT TFTF
  plotter: VIA 4 0 0 -1

layer:  13   CUT   LCCC
  display: 0   CUT FALSE 7 0 7 -2 0
```

```
  display: 1   CUT TRUE  7 0 7 -3 0
  display: 2   CUT TRUE  7 0 7  0 0
  display: 3   CUT TRUE  7 0 7 -3 0
  pattern: 2 4 7070 0707 7070 0707
  plotter:  CUT 4 0 0 -1


layer: 14  ERROR LERR
  display: 0  ERROR FALSE 7 0 7  0 0
  display: 1  ERROR FALSE 7 0 7  0 0
  display: 2  ERROR FALSE 7 0 7  0 0
  display: 3  ERROR FALSE 7 0 7  0 0
  plotter: ERROR 7 0 0 -1

layer: 15  BOUND MBB
  display: 0  BOUND FALSE 5 0 0 -1 0
  display: 1  BOUND FALSE 5 0 0 -1 0
  display: 2  BOUND FALSE 5 0 0 -1 0
  display: 3  BOUND FALSE 5 0 0 -1 0
  plotter: BOUND 7 0 0 -1

layer: 16  COMP LCX
  display: 0  COMP FALSE 7 0 0 -1 0
  display: 1  COMP FALSE 7 0 0 -1 0
  display: 2  COMP FALSE 7 0 0 -1 0
  display: 3  COMP FALSE 7 0 0 -1 0
  plotter: COMP 8 0 0 -1

layer: 17  SELKT SEL
  display: 0  SELKT FALSE 15 0 0 -1 0
  display: 1  SELKT FALSE 15 0 0 -1 0
  display: 2  SELKT FALSE 15 0 0 -1 0
  display: 3  SELKT FALSE 15 0 0 -1 0
  plotter: SELKT 5 0 0 -1

cmap:  0  3  3  3
cmap:  1 15  0  0
cmap:  2  0 15  0
cmap:  3 15 15  0
cmap:  4  0  0 15
cmap:  5 15  0 15
cmap:  6  0 15 15
cmap:  7 15 15 15
cmap:  8  0  0  0
cmap:  9  8  0  0
cmap: 10  0  8  0
cmap: 11  8  8  0
cmap: 12  0  0  8
cmap: 13  8  0  8
cmap: 14  0  8  8
cmap: 15  8  8  8
```

Note that all layers with non-zero menu square numbers will have squares allocated in the various LAYERS menus. The other layers are internal layers, which the user is not permitted to draw in. The permissible range of layer numbers in NEWOL is currently from 1 to 18.

11.3 Message-of-the-day file

The message-of-the-day file is printed to the screen when NEWOL is first started, unless the "-q" flag was specified on the command-line. This file is called `MOTD.WOL` and is searched for in the current directory and the system library directory. It provides a convenient mechanism to inform users of new changes, etc. By convention, the old MOTD files have been amalgamated into `/LIB/WOLLIB/MOTD.OLD`.

11.4 DRC rules file

Upon every invocation, the DRC reads a configuration file of rules. This file is called `<technology>.drc`, and is searched for in (i) the current directory, (ii) the user's home directory, and, failing these, (iii) in the system library directory (/LIB/WOLLIB). The syntax of the configuration file is the following:

`width A dist`	`layer A must be` *dist* `wide`
`separation A B dist`	`A and B must be` *dist* `apart`
`separation2 A B C dist`	`A within B must be` *dist* `from C`
`overlap A B dist`	`A must surround B by` *dist*
`overlap2 A B C dist`	`A within B must be surrounded by C by` *dist*

11.5 Performance log files

Two files, `XTR.LOG` and `DRC.LOG` are maintained in the system library directory. These are intended as aids to monitoring the performance of the system with regards to several parameters under the control of the program maintainer (see the source of the extractor and design rule checker for more information). These files should be truncated (NOT deleted) periodically, to conserve disk space.

11.6 User-feedback files

Two files, `GRIPES.WOL` and `IDEAS.WOL` are maintained in the system library directory. The GRIPES file contains bug reports contributed by users, and the status of the bugs as reported by the system maintainer. The IDEAS file contains user suggestions regarding improvements for NEWOL. These are accessed by the user by typing $\hat{G}$ and $\hat{W}$, respectively.

11.7 Other files

The composition buffer PLOT command reads a file LABELS.TEXT in the current directory, which contains the (x,y) position (in lambda, with a space separating them) of a string to plot.

The PORTS command in the OPTS menu creates creates a file PORTS.WOL that contains the (heirarchical) portnames of all instanced cells, as well as their absolute (x,y) coordinates (in lambda, with (0,0) in the lower left corner of the chip). This is primarily useful for interfacing with an external router.

11.8 NEWOL's CIF dialect

NEWOL's CIF parser is capable of interpreting only a highly stylized version of CIF. Unfortunately, when something goes wrong, it is not very good at reporting what the problem *really is*.

11.8.1 Whitespace in CIF

The CIF standard specifies that only uppercase letters, numbers, parentheses, minus signs and semicolons are significant; all other characters are to be ignored. In particular, commas between pairs of numbers should be treated as spaces. NEWOL's parser requires blanks in these positions.

11.8.2 Encoding λ in cell definitions

The form of a cell definition is `DS 1 150 2 ;` the "150" is the size of lambda in $1/100^{th}$s of microns. NEWOL *assumes* the last number to be a 2.

11.8.3 Non-freeform input

NEWOL requires each CIF command to start on a new line, in column 1.

11.8.4 LAYER commands

NEWOL's parser treats the layer command ("L") and the subsequent layer name as a SINGLE token. Therefore, there can be no space between the L and the name.

11.8.5 Transformations

NEWOL only parses certain transformations correctly. The list of acceptable transformations, and their form, is:

```
C 1 T 0 56;                 -  untransformed
C 1 MX T 36 0;              -  mirrored in X
C 1 MX R 0 1 T 86 42;       -  mirrored in the line Y=-X
C 1 R 0 1 T 86 62;          -  rotated by 90
C 1 R -1 0 T 124 104;       -  rotated by 180
C 1 R 0 -1 T 126 98;        -  rotated by 270
C 1 MX R -1 0 T 88 48;      -  mirrored in Y
C 1 MX R 0 -1 T 126 6;      -  mirrored in the line Y=X
```

Contents

Index