CALIFORNIA INSTITUTE OF TECHNOLOGY

Computer Science Department

5113:TR:84

The Wollery

by

Carver Mead

1/31/84

WoL's Eye View of Cell Design

(WoL 6.0 MANUAL)

Carver Mead

3/8/83

WoL is an interactive cell design system primarily aimed at I^2L designs. Bipolar designs look different at the bottom level than MOS designs do because current flows perpendicular to the surface rather than parallel to it. For this reason the inclusion of one region within another has a profound effect on the semantics of the circuit. I have decided to push this idea to its logical conclusion in **WoL**. It occurred to me that the same construction is valuable at higher levels of chip design—at floorplan level the inclusion of several cells within a higher level cell determines the hierarchical organization of the chip. I have plans to make **WoL** into a composition tool as well, using this inclusion metaphor as a primitive. However, at this time **WoL** is definitely just a cell design system.

WoL is written in Pascal and runs on the **hp** chipmunk series (9826/9836), using a **hp** 9111A tablet. A black and white machine needs a color board and a color monitor. A plot package is included for any of the standard **hp-gl** plotters. **WoL** generates a very stylized form of CIF, which it uses for its own files. These files can be used with any of the composition tools around for generating larger designs. Don't, however, try to use a CIF file generated elsewhere as input to **WoL**—it will blow its little mind! **WoL** will also generate files in its own shorthand format, if a `.WOL` extension is specified in a filename. The default extension is `.CIF`.

When the system is started up, **WoL** prompts for an input file—if you just want to start from scratch hit *Return* (marked *Enter*). The system will then be ready

to draw shapes. The only *shape* possible in WoL is a box with its sides parallel to the axes. I^2L designs tend to have a lot of rectangular base regions stacked up like cordwood, and benefits very little from more general geometry. Before drawing a shape, select a color from the menu.

Menu Square	Color	I^2L Layer	CIF Layer
1	RED	BASE	LNP
2	GREEN	COLLECTOR	LND
3	YELLOW	COLLAR	LNI
4	BLUE	METAL	LNM
5	MAGENTA	MBB	—
6	CYAN	—	LMB
7	WHITE	CONTACT	LNC

If you want layer 0 (GLASS cuts for pads), there is no menu square, and you will have to use the *Layer* command in *Command* mode (see below). That layer is displayed in dashed MAGENTA on the screen. Notice that the *n*MOS layers have been used so the resulting CIF will float right through existing Caltech software. The system is set up to be used for *n*MOS and *c*MOS–SOS as well as I^2L.

WoL uses a pen on the tablet rather than a mouse. All the functions work in a very uniform and natural way. When the pen is contacting the tablet surface but is not pressed down, only the cursor is moved. When the pen is depressed, an action is started, and keeps going until the pen is released. WoL comes up in the *Add Box* mode; the depression of the pen creates the corner of a "rubber band box." The pen can be moved around and the other corner of the box follows in real time. Upon release of the pen, The other corner of the box is set, and the shape is added to the data structure of the cell. Enjoy!

In addition to drawing boxes, WoL has a powerful set of editing features. All actions effect the *selected shape*. Just what selection means and how it is done will be discussed later.

Menu Square	Action
8	*Copy* the shape and move the copy
9	*Mirror* the shape in x
10	*Mirror* the shape in y
11	*Rotate* the shape by 90°
12	*Rotate* the shape by 180°
13	*Move* shape with cursor
14	*Edit* one edge of the shape
15	*Delete* shape
16	*Command* mode

In the *Move* mode, a depression of the pen selects a shape and its contents. As the pen is moved around the shape follows in real time. Upon release of the pen, the shape is placed in its new location. In the *Edit* mode, a depression of the pen selects one side of a box. As the pen is moved that edge follows. When the pen is released, the edge stays at its new location.

The *Command* mode prompts for text input from the keyboard—the choices are given as a prompt. To change the display *scale*, use the *S* command and an integer. This operation does not effect the shape, it only changes the scale at which shapes are displayed. A typical scale is 8 (pixels per λ). Any value down to 1 can be used.

The **WoL** data structure contains a tree of *shapes*. Each shape has a *next* shape and a *contents* shape. Selecting any shape also selects all of the contents tree below it. A shape gets to be the contents of another shape by being drawn, moved, or copied into that shape. The opposite is **not** true—don't try to include a shape by drawing a new shape around it.

Selecting a simple shape is done in the obvious way—by putting the pen down anywhere inside it. In *Move* mode, for example, as long as the pen is down the shape will move with the pen. Once the pen is released, only the cursor moves with the pen. All shapes which you create are contained in a "shape at ∞" supplied by the system. For this reason you can scroll around by putting the pen down outside all of your shapes and moving the box at ∞—try it!

For more complex shapes, a slightly more sophistocated selection criterion is used. A shape is contained in another shape if their boundaries are coincident. This construct is necessary for structures like contacts to I^2L collectors, where both the metal square and the collector square are the same size. In *n*MOS, contacts from metal to poly or diffusion often have the same problem. If one shape is apt to grow, as the collector might in the I^2L example, put that shape down first. It will then be the parent of the others which are put down later. In general, the smallest shape will be chosen which surrounds the pen. If the pen is on an edge, it is considered inside the shape. However, if the pen is exactly on a coincident edge, as for example the contact mentioned before, the parent is selected. This algorithm leads to a very natural "feel" in dealing with layout.

I have alluded to being "*exactly* on an edge" of a shape. Yes, you guessed it—the system only allows pointing at discrete points: *The Grid.* Oh well—what can you expect from a totally corrupt believer in integers. The λ grid is maintained at any display scale. I have not found any reason to clutter up the display with a grid when the same effect can be achieved more naturally by quantizing the pointer.

How do you know if a shape is selected? Move it a little bit! Everything which moves is selected. You can leave it exactly where it was thanks to the grid. *Mirror* and *Rotate* operate immediately upon selecting the menu. Don't worry—they are easily reversed if you make a mistake. The *Delete* mode lasts until you either select a color or one of the other modes or operations, so you can delete a bunch of shapes just by putting the cursor on them and pressing the pen. If you accidentally delete a shape, get into *Command* mode and use the *Undelete* command. It restores the

last shape which was deleted. To return to *Add Box* mode, just select a color.

The style I use to design cells is to get something like what you want on the screen and use it like a menu. Copy appropriate pieces over to the design you are doing, modifying them with the *Edit* mode to get them to the right size. An incredible amount of stuff can be generated very quickly in this way. No amount of explanation can really give you a "feel" for the system—you simply have to work with it on some real designs. Be my guest!

So now you have a cell—something you really want to save. For the moment suppose you have deleted all the random garbage off of the screen and what you see is what you want to get. Be careful. Select the box at ∞ by *Moving* it. Hit the *Command* square on the menu. The system prompts for a command. Select *Make Cell File* by typing *m* or *M* and it will prompt for a file name. Caution: **WoL** is really a pretty dumb bird and will happily write over an existing file! Once the menu light on the tablet goes out the cell has been tailor fitted with its very own bounding box and carefully filed away. Whew!

To be sure, move the stuff on the screen out of the way and hit the *Command* square again. This time *Get* the file and it will magically appear on the screen. The bounding box is displayed in purple if the *bb* flag is on. The *B* command toggles this flag, which also controls whether the bounding boxes of subcells are plotted.

Plotted you say? Yes indeed, **WoL** provides its precious users with its very own plot package. No more waiting while the Big Mother is lying wounded on the machine room floor. Plot to your heart's content! Make sure your shape is selected and use the *Plot* command. You will be prompted for either an $8\frac{1}{2} \times 11$ (*Half* size) or 11×17 (*Full* size) plot.

If you wish to file only one of several cells on the screen, simply select the one you want with the *Move* command and then use the *Make Cell File* command as described above. Only the selected shape will be filed away, and the rest unabraded. Since cells filed in this manner will already have a bounding box, **WoL** is smart enough to not create another one.

Once you have a bunch of cell files, you will want to do composition. It is clear that **WoL** can do simple composition, but it won't yet take you very far. It is good for making new cells in the context of the ones with which it has to abut. You can draw a bounding box using color 5, and build the new cell within it. This approach leads to a floorplanning discipline which is really quite healthy. Once they are done, your files can be shipped to 20s or Vaxen with the TRM terminal emulator. There you can use the limitless resources of composition tools developed by casts of thousands over many years. Myself, I have been working on a simple composition system for **WoL** called **WoLcomp**. The manual included below gives details. Maybe there is a chance the old bird will get smarter!

updated 10/27/83 - Josh
updated 1/31/84 - Carver

WoLcomp 2.0 MANUAL

11/28/83

Carver Mead

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 ... ad nauseum ... 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. *Don't ask.*

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. **WoLcomp** 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 **WoLcomp** 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:char);

The character argument is the last character of the layer select. 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 entire mess 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 forgotton 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.

By now, programs in The **WoLery** have been used to do substantial designs in all three technologies. Josh and MaryAnn have upgraded **WoL** to a well-structured program, and there is a version of **WoLcomp** that puts out SOS layer commands. Josh is also hard at work on a new hierarchical version of **WoL**. There are many more sophistocated 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.

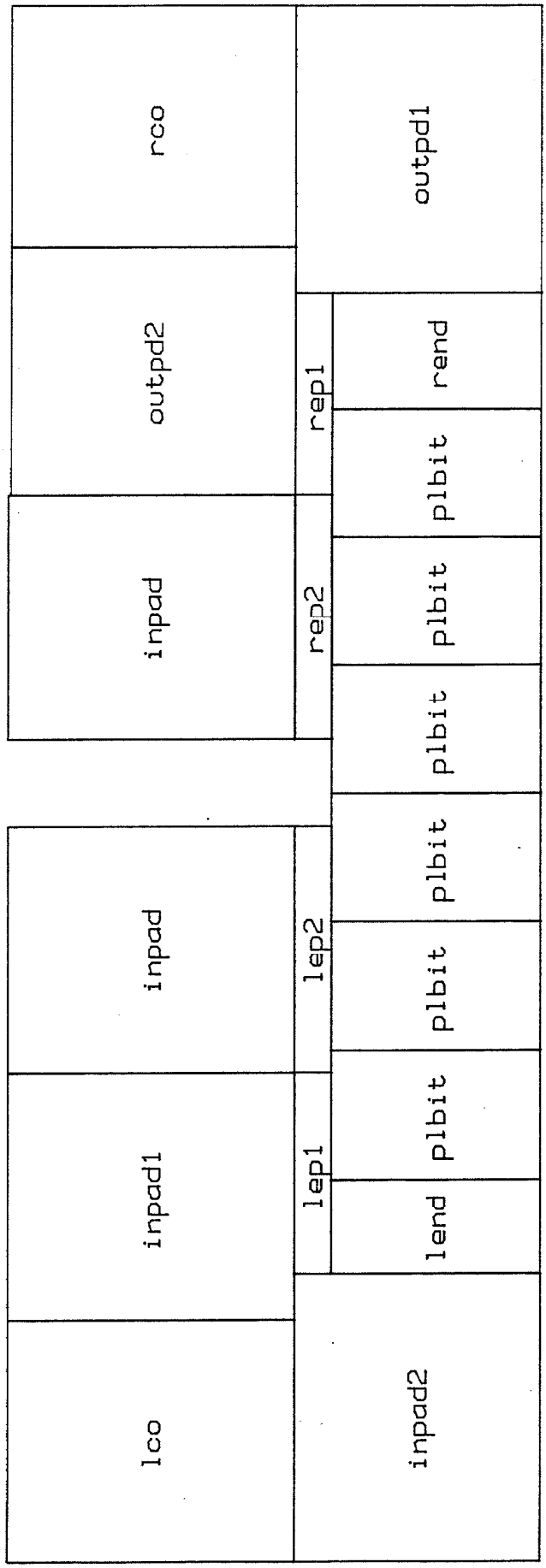
The cells `rend`, `lend`, `lep1`, `lep2`, `rep1` and `rep2` are wiring cells, and contain no active elements. The `lco` and `rco` cells are the VDD and GND pads. The multiplier itself is made out of `plbit` cells, which in turn are made out of subcells. All of the `pd` and `pad` cells are pads with drivers 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.

updated 1/31/84 - Carver

```
begin {main program}
set_up;                              { initialize program state }
   { YOUR DESIGN HERE }
   getfile('lplb');
   getfile('rplb');
   getfile('lplu');
   getfile('rplu');
   getfile('lplo');
   getfile('rplo');
   getfile('leu1');
   getfile('leb1');
   getfile('leo1');
   getfile('reu1');
   getfile('reb1');
   getfile('reo1');
   getfile('inpad');
   getfile('outpad');
   getfile('inpad1');
   getfile('inpad2');
   getfile('lep1');
   getfile('lep2');
   getfile('lep3');
   getfile('lep4');
   getfile('lco');
   getfile('rep1');
   getfile('rep2');
   getfile('rep3');
   getfile('rep4');
   getfile('outpd1');
   getfile('outpd2');
   getfile('rco');
   define('plr');
       cell1:=cell('rplu');
       cell2:=cell('rplb');
       cell3:=cell('rplo');
       draw(cell1); y0:=y1; draw(cell2); y0:=y1; draw(cell3);
   endef;
   define('pll');
       cell1:=cell('lplu');
       cell2:=cell('lplb');
       cell3:=cell('lplo');
       draw(cell1); y0:=y1; draw(cell2); y0:=y1; draw(cell3);
   endef;
   define('plbit');
       cell1:=cell('pll');
       cell2:=cell('plr');
       draw(cell1); x0:=x1; draw(cell2);
   endef;
```

```
    define('lend');
        cell1:=cell('leu1');
        cell2:=cell('leb1');
        cell3:=cell('leo1');
        draw(cell1); y0:=y1; draw(cell2); y0:=y1; draw(cell3);
    endef;
    define('rend');
        cell1:=cell('reu1');
        cell2:=cell('reb1');
        cell3:=cell('reo1');
        draw(cell1); y0:=y1; draw(cell2); y0:=y1; draw(cell3);
    endef;
    define ('plm');
      cell3:=cell('inpad2'); draw(cell3); x0:=x1; x2:=x1; y2:=y1;
      cell3:=cell('lend'); draw(cell3); x0:=x1; y3:=y1;
      cell1:=cell('plbit');
      for i:=0 to 5 do begin
         draw(cell1); x0:=x1;
      end;
      cell3:=cell('rend'); draw(cell3); x0:=x1; x4:=x1;
      cell3:=cell('outpd1'); draw(cell3);
      x0:=x2; y0:=y3;
      draw(cell('lep1')); x0:=x1; draw(cell('lep2')); x0:=x1;
      x0:=0; y0:=y2; draw(cell('lco')); x0:=x1;
      draw(cell('inpad1')); x0:=x1;
      cell1:=cell('inpad'); drawx(cell1,3);
      x3:=x1;
      cell1:=cell('rep1'); x0:=x4-cell1^.lx; y0:=y3; draw(cell1);
      cell1:=cell('rep2'); x0:=x0-cell1^.lx; draw(cell1);
      x5:=x0; y0:=y1; cell1:=cell('inpad'); drawx(cell1,5); x0:=x1;
      draw(cell('outpd2')); x0:=x1;
      draw(cell('rco'));
      x0:=x3; y0:=y2+3; x1:=x5; y1:=y0+16; box('M');
      y0:=y1+4; y1:=y0+16; box('M');
    endef;
    draw(cell('plm'));
shut_down;                          { close output file }
end.
```

plm


rco
outpd1
outpd2
rep1
rend
plbit
plbit
inpad
rep2
plbit
plbit
inpad
lep2
plbit
plbit
inpad1
lep1
plbit
lend
lco
inpad2