# CIFPLOT: PLOTTING SOFTWARE FOR IC LAYOUTS

by

Daniel T. Fitzpatrick

Memorandum No. UCB/ERL M81/96

29 December 1981

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

# CIFPLOT: Plotting Software for IC Layouts

*Daniel T. Fitzpatrick*

Computer Science Division
Department of Electrical Engineering and Computer Science
University of California, Berkeley
Berkeley, California 94720

*ABSTRACT*

This report describes the usage and implementation of CIFPLOT, a program to plot integrated circuit designs described in CIF 2.0, the Caltech Intermediate Form.

December 29, 1981

Sponsored by Defense Advance Research Projects Agency (DoD) ARPA Order No. 3803 Monitored by Naval Electronic System Command under Contract No. N00039-78-G-0013-0004

# CIFPLOT: Plotting Software for IC Layouts

*Daniel T. Fitzpatrick*

Computer Science Division
Department of Electrical Engineering and Computer Science
University of California, Berkeley
Berkeley, California 94720

## Introduction

In recent years there has been much interest among the academic and research communities in Very Large Scale Integration(VLSI). It is presently possible to build entire processors on a single integrated circuit chip. There are tremendous benefits to being able to design entire systems on a chip. In order to build these systems, though, designers need the help of several computer tools. This report describes one such tool, CIFPLOT, a program that reads the layout description of a circuit and produces a plot on one of many different output devices. In addition to being able to produce checkplots, CIFPLOT can reformat the layout description and is used as a front end to a program that extracts the underlying circuit from the layout.

CIFPLOT accepts layout descriptions in the Caltech Intermediate Form(CIF). CIF has become the accepted standard interchange language of most universities and several companies. Designs specified in CIF are regularly fabricated in fast turnaround facilities as Multi-Project Chips(MPC).

CIF is a general geometric language especially well suited for describing integrated circuits. Symbols can be created from primitive shapes, such as polygons and rectangles, and from other symbols. This allows for a simple yet powerful calling hierarchy.*

The generality of CIF, though, makes it hard to implement. Often the complexities of CIF are ignored and left unimplemented. Vlsi design involves a huge number of shapes, making the designs very complex. Simple approaches to deal with these designs often fail because of the enormous amount of data that must

* See [Hon & Séquin] for a complete description of CIF.

be manipulated. Programs to manipulate this data can become intolerably slow.

CIFPLOT was designed to be a useful tool for a large class of users. As such, many of the problems faced by VLSI tools had to be addressed. In the design of CIFPLOT several goals were established. The first goal was to accept any valid CIF description. This includes the ability to interpret the full CIF language, and to handle general geometry, including non-orthogonal features and self-intersecting polygons. This allows it to accept designs from any system that produces CIF. Thus, CIFPLOT is not locked into any particular design style or system.

The second goal was to check the CIF description carefully for syntactic or semantic errors. Designers have missed important MPC deadlines due to syntactic errors in their CIF that should have been caught by their design tools. By careful checking, this problem can be avoided. Of course, this checking is not much good unless it provides the user with help in fixing the errors. A system that, after reading a 10,000 line CIF file, prints "syntax error" and nothing more, is not much help. When an error is discovered, CIFPLOT should assist the user in locating the error.

In order to meet the needs of a many users, the third goal was to make the system flexible. Different users have different ideas about how a plot should look. The system should provide large number of user options, which specify what should and should not be plotted, set the size of the plot, select the area of interest, and specify the plotting device. Also since not all designs are done in NMOS, the system should be extensible to other technologies.

The final goal was efficiency. It was recognized that the goal of efficiency might conflict with the other goals of generality and flexibility. As such, efficiency was considered of secondary importance to the other goals. Yet, due to the large number of objects that must be handled, and the continuing trend to larger and more complex designs, efficiency could not be ignored.

Throughout the design of CIFPLOT these goals have been kept in mind. The interpreter and plotter have been written with a minimum amount of dependence on each other. As a result the interpreter can be used by other programs that need to read CIF. The plotter can be used by interpreters for other languages. The plotting routines have been used by STIFPLOT[Kohn], a program

that interprets STIF[Séquin].

This report is divided into three parts. The first part describes the use and operation of CIFPLOT. It is intended to serve as the user's manual for CIFPLOT. Part two describes CIFPLOT's implementation. The final part discusses experience with the program and discusses some possible improvements to CIFPLOT.

## Part I: Usage

### 1. Introduction

CIFPLOT is a program that interprets Caltech Intermediate Form (CIF), and produces a plot. CIFPLOT also provides a front end to a circuit extraction program, and the CIFPLOT interpreter is capable of reformatting CIF files. CIF is a low-level graphics language designed for describing integrated circuit layouts. Although CIF is suitable for other graphics applications we will assume throughout this report that it is being used for integrated circuit design. In addition to interpreting the full CIF 2.0 language, CIFPLOT recognizes a number of local extensions. Local extensions can sometimes cause incompatibility with other programs that interpret CIF. CIFPLOT, though, can be made to produce a file of standard CIF that is equivalent to the source file with local extensions. So if you use local extensions you still will be able to transport your circuit description to other installations by having CIFPLOT produce a standard CIF file.

There are several potential sources of CIF files. CIF files can be generated by a graphics editor, such as CAESAR[Ousterhout] or KIC[Keller]. A CIF file may be the interpreted form of a higher level descriptive language. CIF files can also be created directly, either using a program such as MKCIF[Krause] or straight from a text editor. Once you have a CIF file, plotting it is a straightforward process.

The calling convention of CIFPLOT is similar to that of many other UNIX programs. CIFPLOT is called from the command line with the name of the CIF file to be interpreted. If more than one file is used, a list of files may be given on the command line. By convention each CIF file should end with '.cif', but this convention is not enforced. As an example, suppose 'lib.cif' contains CIF descriptions of commonly used cells such as I/O pads, super buffers, clock drivers, etc. and that 'sorter.cif' contains a CIF description of a sorter using symbols found in 'lib.cif'. The command line to plot this circuit looks like this:

**cifplot lib.cif sorter.cif**

This causes CIFPLOT to first read the files 'lib.cif' and then 'sorter.cif'. The CIF *End* command should be only in 'sorter.cif' since CIFPLOT ignores all text following the *End* command. If there are no errors, CIFPLOT will print out the window size, and an estimate of the size of the plot. It will then stop and ask you if you

want a plot. You can then check to see that the size of the plot seems reasonable. If it is, type 'y'. It will then proceed to produce a plot on the default output device (Currently the Varian-Benson plotter). A typical session might look like this:

```
% cifplot lib.cif sorter.cif
Window -5700 174000 -76500 168900
Scale: 1 micron is 0.004075 inches
The plot will be 0.610833 feet
Do you want a plot? y
```

CIFPLOT recognizes a number of command line options that allow you to change the default settings. These options need to be specified before the list of CIF files. For instance, to send output to the Versatec instead of the Varian use the -W option. The command line should then look like this:

```
cifplot -W lib.cif sorter.cif
```

Other options that affect the scale or looks of the plot may also be specified on the command line. These are discussed in more detail later.

CIFPLOT does extensive error checking on the input CIF file. This helps verify that the CIF files are syntactically correct and will be accepted by the fabrication site without problems. When an error is found, CIFPLOT tries to pinpoint the problem so the user can quickly fix the mistake. As an example consider the following CIF file.

```
(sample CIF file)
L ND; B 10 20 0 0;
L NP; W 2 0 0 0 10 20 10;
C 15 MX;
L CB; R 20 10 10;
```

This file has several syntactic and semantic errors. For instance, the comment on line 1 does not end with a semi-colon, and on line 4 there is a call to symbol 15 but symbol 15 is never defined. Running CIFPLOT on this file causes the following messages to be printed:

```
% cifplot errors.cif
   1  (sample CIF file.)
   1 ------------------^----error: Comments must end with a semi-colon
   4  C 15 MX;
   4----------^-------Error: Symbol 15 Undefined
   5  L CB; R 20 10 10;
   5 ------^---------Error: Unknown Layer
Line 6  error: No End Statement
Plotting Suppressed
```

Error messages are discussed in more detail later.

CIFPLOT recognizes the full CIF language. This includes even CIF constructs such as *Delete Definition* and symbol renaming. (Note: although these constructs are recognized, it must be emphasized that they are not recommended and should be avoided. These constructs are a frequent source of errors and are usually not implemented by other programs that accept CIF.) Care has been taken to avoid placing restrictions on the circuits CIFPLOT can handle. Geometric shapes can be placed at arbitrary angles. Polygons may be defined with an arbitrary number of vertices. Polygons may even be self-intersecting, thus allowing polygons with holes in their center. CIF files may be of arbitrary length, with no practical limits on the number of primitives, symbol definitions, or the depth of the calling hierarchy. The only limiting factor is the virtual memory size of the VAX. This has caused no practical limitation on circuits accepted by CIFPLOT.

## 2. Command Line Options

CIFPLOT attempts to be as flexible as possible in letting you choose how your plot should appear. Most often the default settings should be adequate but occasionally you will want to call upon different options. These options allow you to choose which device you want to plot on, the layers you want plotted, the window size for your plot, and various other things. You may specify as many options on the command line as you like. All option specifiers are preceded by a dash('-'). When conflicting options are specified the last option is always used. Consider the following command line.

```
cifplot -W -b "Sample circuit" -V -s 500x circuit.cif
```

The -W option, which selects the Versatec plotter, is followed by the -V option, which selects the Varian plotter, therefore the plot is sent to the Varian. The -W option is ignored. This is useful since the user may want to use the 'alias'

feature of the CSH[Unix] to set default options for CIFPLOT. These default options can be overridden by adding the correct options to the end of the line. The following sections explain the command line options of CIFPLOT.

## 2.1. Selecting Devices

Normally, plots are sent to the Varian. CIFPLOT can be made to send the plot to another device. The **-W** option sends output to the Versatec. Rather than sending the plot directly to the Versatec, CIFPLOT stores the bit pattern in a temporary file. Once all the bits have been computed, this temporary file is sent to the plotter. This results in a nice crisp plot. It also avoids tying up the plotter while computing what the plot looks like. Unfortunately, storing the bit pattern in a file uses up a lot of disk space. Plotting on the 3 foot wide Versatec requires approximately 2 megabytes per foot of plot. No compaction is done on this dump file. Before starting CIFPLOT on a large plot, you should make sure there is enough disk space for your plot. (It is possible to break big plots into smaller pieces by using the windowing command discussed later.)

CIFPLOT also knows about a few other display devices. The **-Ga** option displays the circuit on an AED 512 graphics terminal. The **-Gh** option displays the circuit on a HP 2648 graphics terminal. For both these options, **-Ga** and **-Gh**, CIFPLOT must be run on a terminal of the specified type. The **-K** option produces a file in UNIX plot(5) format. This file can be used for making pen plots. CIFPLOT also has the ability to make crude plots on a standard alphanumeric CRT screen. This is the **-T** option. Of course, this can only be used for small plots.

## 2.2. Windowing

Often a picture of the whole circuit is not necessary or even desirable. For instance, suppose that you have a CIF description of a chip, and you wish to see only part of the chip. If you are not interested in much of the chip, such as the I/O pads, and other peripheral circuitry, you can specify a window of that part of the chip that interests you, and thus not waste time or paper generating a plot of the other circuitry.

This can be done by setting a window of the plot. The window is that part of the plot you wish to see. By default the window is the entire plot. The **-w** option is used to set the window. The form of this option is

**cifplot -w *xmin xmax ymin ymax* file.cif**

where values of *xmin*, *xmax*, *ymin*, and *ymax* are in user defined units. (Default is CIF units. See sections 2.4 and 5.1.)

For historic compatibility CIFPLOT plots with the Y-axis running across the page and the X-axis advances in the direction that the paper advances. Knowing this is important when trying to set the desired window.

## 2.3. Scaling

By default the scale is set such that the window of the plot fills the whole page. Thus the bigger the circuit, the smaller the scale. Sometimes, even on a three foot wide Versatec, the details of the circuit are too small to be easily seen. The **-s** option can be used to set the scale to any desired value and make the details of the circuit more readable. The **-s** option must be followed by a floating point number that is the desired scale in inches per micron. For example, the following command line sets the scale so that 1 micron is one-twentieth of an inch.

**cifplot -s 0.05 circuit.cif**

If the scale is set such that the plot will no longer fit on a singe page, several pages will be made. These pages can be laid side-by-side to achieve the effect of a larger plot.

The scale can also be specified in millimeters per micron or in terms of magnification. To set the scale in millimeters per micron, follow the scale number immediately by an 'm'. To set the scale in terms of magnification, follow the number immediately with a 'x'. The following command line sets the scale to a magnification of 250.

**cifplot -s 250x circuit.cif**

Care is required when setting the scale since a large value could produce a ridiculously large plot.

## 2.4. Units

By default CIFPLOT uses CIF units for coordinates and measures of size. Often, the user would prefer to think in another system of units, such as microns, mils, or lambda. CIFPLOT allows the user to specify a basic unit of measurement. By following the **-u** option with a number, the basic unit becomes that many CIF units. For instance, to make the basic unit 1 micron the command line looks like this:

**cifplot -u 100 circuit.cif**

The window will now be reported in microns. Also you can specify the window size and grid size on the command line in microns. (See sections 2.2 and 2.9.) The grid will be drawn in micron coordinates. Following the standard command line convention, the units option affects only those options that follow it. (Also see section 5.1.)

## 2.5. Making Layers Invisible

Often it is desirable to make a plot without all of the layers displayed. For instance, if a designer wants to check that each contact in his design is covered by metal, he could just plot the metal and contact layers but not the poly or diffusion layers. The **-l** option lets you specify a list of layers you want made invisible. Each layer is specified by its CIF name. For the above situation the command line looks as follows:

**cifplot -l NP,ND,NI,NG,NB circuit.cif**

This command makes invisible the poly, diffusion, implant, overglass, and buried contact layer. Note that there are no spaces in the list of layers and layer names are separated by commas.

In addition to making layers invisible, this command can be used to make other features invisible. Along with layer names, this command recognizes several other names: **bbox, alltext, text, symbolname, pointname** and **outline.** Since each of these names contain only lower case characters there is no problem with a conflict with a layer CIF name. (Note that CIFPLOT relies on the distinction between upper and lower case for the layer command. The string 'nm' will not match the layer name 'NM', and the string 'Text' will not match the keyword

'text'.) The keyword **bbox** makes the bounding boxes of symbols invisible. The keyword **alltext** makes all text invisible. The keyword **text** causes text that comes from CIF text extension command '2' not to be drawn. (Text extension command '2' is discussed in section 3.5.) The keyword **symbolname** suppresses putting the symbol name in the upper left hand corner of the bounding box of each instance of a symbol. The keyword **pointname** suppresses point names from being drawn. Point names come from the CIF extension command 94 discussed in section 3.6. The keyword **outline** suppresses the thin outline drawn around each layer. The keywords **alltext**, **symbolname**, and **pointname** can be abbreviated by **at**, **sn**, and **pn**, respectively.

### 2.6. Multiple Copies

Large plots can take a lot of time to produce. Once all the work to make a single plot has been done it is not much extra work to make several copies of the plot. This can be done with the **-c** option. This option should be followed by the number of copies wanted. For instance, the following command line makes 5 copies.

**cifplot -c 5 circuit.cif**

### 2.7. Rotate

As mentioned above, plots are made with the Y-axis running across the page and the X-axis advancing as the paper advances. This is counter to the way that many people think. To make the plot so that the X-axis runs across the page and Y decreases as the paper advances use the **-r** option. This essentially rotates the plot 90 degrees counter clockwise.

### 2.8. Depth

Too much detail can often hide important features in a circuit. If, for instance, your circuit is a CPU, then a picture of the placement of registers may convey more information than a picture of all the transistors. The **-d** option tells CIFPLOT to only plot the circuit down *n* levels of call, where *n* is an integer that is specified on the command line. To plot with only the top two levels of calls plotted, the command line looks as follows:

**cifplot -d 2 circuit.cif**

The calls not plotted are replaced with their bounding box and the name of the symbol.

## 2.9. Grid

Even with a checkplot in hand it is often hard to determine the coordinates of certain features. The **-g** option draws a grid over the plot with the coordinates displayed near the edges. The **-g** option must always be followed by an integer that sets the spacing between grid lines in user defined units. (Default is CIF units. See sections 2.4 and 5.1.) To draw a grid line every 10000 CIF units (100 microns) the command line looks as follows:

**cifplot -g 10000 circuit.cif**

If you follow the **-g** option with too small a value there is the possibility that the plot will be completely covered with grid lines.

## 2.10. Approximations

By default, roundflashes are approximated by octagons. Many times the octagon approximation will be inadequate. The **-a** option allows you to specify how to approximate roundflashes. This option is followed by an integer that specifies the number of sides to give a roundflash.

## 2.11. Banner

On top of each plot CIFPLOT places the user's login name, the date and time the plot was created, the bounding box of the plot in CIF units, and the scale of the plot. Often it is useful to have other information. The **-b** option lets you specify what else should appear on the banner. A quoted string follows the **-b** and this string is placed at the top of the plot. For example, suppose the file 'circuit.cif' was a CIF description of a four by four multiplier. The following command line places the string '4 by 4 multiplier' at the top of the plot.

**cifplot -b "4 by 4 multiplier" circuit.cif**

## 2.12. Listings

The -L option makes CIFPLOT print the CIF file on the terminal screen as it reads it. This option is helpful for debugging syntax errors in hand coded CIF, but it is usually unnecessary.

## 2.13. Comments

CIFPLOT is quite fussy about comments. They must be syntactically correct, or an error message will be issued. Many other programs that read CIF files are not as finicky about comments as CIFPLOT. As a result many CIF files that are accepted without complaint by other CIF parsers are syntactically incorrect due to errors caused by comments. It is, of course, possible to change these CIF files so that they are syntactically correct, but this is a tedious job. The -C option will cause CIFPLOT to treat comments as it they were blanks, removing most comment related complaints.

Your circuit should be developed without this option because other programs may in fact be as fussy as CIFPLOT about comments. If you have been lax about the syntax of comments in your CIF description, it may happen that when it is sent for mask fabrication your file may be rejected for comment related errors. Therefore this option should be used only as a stop-gap measure.

## 2.14. Non-Interactive

After reading the CIF files, CIFPLOT will stop and display an estimate of the size of the plot and ask for confirmation before proceeding to produce a plot. This prevents paper and compute cycles from being wasted by mistakenly specified plot parameters. However, you can suppress this feature with the -I option. Calling CIFPLOT with this option causes it to make a plot without asking for confirmation. The window and plot size information will still be printed but it will immediately start the plotting phase. This is mostly useful for running CIFPLOT in the background.

## 2.15. Standard CIF

The -e option causes CIFPLOT to accept only legal CIF 2.0 files. User extensions produce warnings and are ignored. This is useful when sending plots to other places that may not accept the local extensions that CIFPLOT does. Also it is useful in receiving CIF files that use extension commands in a different manner than CIFPLOT.

## 2.16. Defining New Layers and Stipple Patterns

The definition of CIF 2.0 allows for more than the standard layers. For technologies other than NMOS new layer names should be used. It is, of course, impossible to foresee all the possible new layers. CIFPLOT, therefore, allows you to set up your own layer and corresponding stipple patterns.

To define your own layers, it is necessary to create a file with the layer names and stipple patterns. Layer names are a sequence of up to 4 uppercase letters and digits. Each layer specifier consists of the layer name in double quotes followed by 8 integers. Each integer specifies 32 bits. Hence, the 8 integers specify a 32 by 8 bit pattern. Ones are black, zeroes are white. The integers may be decimal, octal, or hex. Hex numbers start with '0x', octal numbers start with just '0'. You may have as many specifiers as you wish in a file. The following command line makes CIFPLOT read the stipple file 'new.pat'.

**cifplot -P new.pat circuit.cif**

This mechanizism also allows you to change the default stipple pattern of predefined layers. If you redefine the stipple patterns for a layer that CIFPLOT already knows about, your stipple patterns are used. The following is an example of what your stipple file might look like:

```
"56",0x00000000, 0x03030303, 0x48484848, 0x03030303,
     0x00000000, 0x30303030, 0x84848484, 0x30303030,
"POLY",0x08080808, 0x04040404, 0x02020202, 0x01010101,
     0x08080808, 0x04040404, 0x02020202, 0x01010101,
"NM",0x22222222, 0x00000000, 0x88888888, 0x00000000,
     0x22222222, 0x00000000, 0x88888888, 0x00000000,
"DARK",0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF,
     0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF
```

## 2.17. Structured Output Files

After parsing a CIF file, CIFPLOT can be made to output an equivalent CIF file that is easy to parse and completely complies with standard CIF 2.0. Important local extensions have been converted into legal CIF 2.0. *Include* and *Array* commands have been expanded into equivalent CIF constructs.* This allows easy transportation to other CIF programs that do not recognize the same extensions. The command line looks something like this:

**cifplot -O outfile circuit.cif**

The file 'circuit.cif' may contain any legal CIF description including the local extensions discussed in section 3. It is parsed and a file 'outfile' is created. The file 'outfile' will contain a CIF description in a highly structured form. This description does not use many of the features of CIF, especially those features that make files difficult to parse.

## 2.18. Text Fonts

By default all text is printed using 6 point Roman print. You may, however, specify any text font by including on the command line a **-F** followed by the file containing the font description. (See vfont(5) in the UNIX programmer's manual for the form of font files). To change the text font to 10 point italics the command line looks as follows:

**cifplot -F I.10 circuit.cif**

The specified file must be in the directory '/usr/lib/vfont'. (See section 5.1 to specify a different font directory.)

## 2.19. Circuit Extraction

CIFPLOT serves as a front end to a circuit extraction program. A circuit extractor reads a layout description and produces a description of the circuit in terms of transistors. To call the circuit extractor type:

---

* Text and naming commands do NOT comply with standard CIF and are left as extensions. These commands, however, are not necessary for the correct fabrication of a chip and no harm is done by leaving these commands as extensions.

**cifplot** -X *out file.cif*

The circuit extractor will create two files, *out*.**sim** and *out*.**nodes**. The file *out*.**nodes** is a list of nodes in the circuit. A node is any electrically connected part of the circuit. See section 6.3 on how to get a plot with node numbers. The file *out*.**sim** is a list of the transistors in the circuit.

## 2.20. Debugging Options

There are two options that are useful mostly for debugging CIFPLOT. The **-n** option causes CIFPLOT to carry out the computations necessary to produce a plot, but no plot is actually produced. The **-D** option causes CIFPLOT to dump a core image whenever it terminates abnormally. This option may be followed by a number between 0 and 9, which causes various information about internal structures to be printed out. The higher the number, the greater the amount of information printed.

# 3. Local Extensions to CIF

In the definition of CIF 2.0 provisions were made for local extensions. This section describes the extensions recognized by CIFPLOT. Although these extension numbers have been chosen to conform with recommended extension numbers it is important to realize that this section discusses extensions only meaningful to CIFPLOT. Other programs that recognize CIF may not recognize these extensions or may even interpret them differently. Use of the extensions therefore make CIF files non-transportable.*

In standard CIF all lower case characters are treated as blanks. In the extension commands, however, lower case characters are not treated like blanks. Therefore, you may not use lower case letters indiscriminately in extension commands. The following few sections describe the extensions recognized by CIFPLOT.

* There is an option to CIFPLOT that will produce a transportable file. See section 2.17 for details.

### 3.1. Arrays

Circuits are often made up of a cell repeated several times and located with regular spacing to form a rectangular array. A common example of this is a memory circuit. A single memory cell is designed and then laid out several times. In standard CIF the simplest way to do this is to make several calls to the memory cell, with each call appropriately transformed. CIFPLOT recognizes an array command, which can build an array of cells. The array command has the form:

**0A** *SymbolNumber x-count y-count x-displacement y-displacement*

This specifies an *x-count* by *y-count* array of symbol *SymbolNumber* with *x-displacement* and *y-displacement* specifying the offsets in the x and y direction, respectively. This construct only allows rectangular arrays, but generally these are all that is desired. Most regular patterns in integrated circuits are rectangular. If the array is enclosed in a definition it can then be translated, mirrored, and rotated just like any other symbol.

### 3.2. Include Files

Most designers rely on a common set of standard cells, such as I/O pads, clock drivers, etc. If these cells are kept in a library file then you can use the include command which will cause CIFPLOT to read the library file as though it was actually part of the designer's original CIF file. The form of the include command is:

**0I** *filename;*

Also the form

**0** *filename;*

is permitted to maintain consistency with other CIF programs. Once this command is encountered, CIFPLOT starts reading from the specified file, interpreting the commands as though they were in the original file. Upon reaching the end of the file, CIFPLOT resumes reading from the original file. Include files may be nested. Nesting, however, is limited to a depth of six.

### 3.3. Vectors

Often it is useful to place line drawings in a plot. CIFPLOT recognizes a vector command which causes a line to be drawn between specified points. The form of the command is:

**0V** *x0 y0 x1 y1 ... xN yN;*

A thin line is drawn between the points.

### 3.4. Printing Messages

This command causes a message to be printed on the terminal (or wherever error messages are sent.) The command has the form:

**1** *message;*

This is useful in giving you an idea of how far CIFPLOT has parsed.

### 3.5. Text on Plot

Text can be placed on the plot with the following command:

**2** *"text" transform;*

The text is then printed out using the specified transformation to specify where the lower left hand corner of the text is to appear. The command

**2C** *"text" transform;*

will cause the text to appear with its bounding box centered about the transformation point. The permitted transformations are all those allowed for symbol calls (rotate, translate, mirror). As in calls, the order is important. Almost always you will want to do the rotating, and mirroring before translating. Actually, rotating and mirroring are fairly useless since text always appears horizontally; translations only apply to the point around which the text is to appear.

### 3.6. Names

Often it is desirable to attach names to CIF objects. Analysis tools such as circuit extractors and design rule checkers can use the names to relate information back to the designer in terms that the designer is familiar with, rather than as meaningless numbers. Names differ from text in that they are, in a sense, part of the circuit. When plotting, however, names can be treated the same as text.

A symbol may be named by including in the symbol definition the command:

**9** *name;*

This is useful if CIFPLOT is not to draw all the details of a plot but just the bounding box of symbols. In this case the symbol name appears in the center of the bounding box. Giving an already named symbol another name produces a warning. The names will appear concatenated. Normally the symbol name appears at the top of the symbol.

You can also name a point anywhere on your circuit. The command to name a point can be in either one of the following forms:

**94** *name x y;*
**94** *name x y layer;*

On the plot the name will appear below and to the right of the point. Names may not contain spaces and should not be numbers. The second form of this command allows a name to be attached to a point on a particular layer. *Layer* must be a CIF layer name. The layer name for plots makes no difference except that if a layer is made invisible so will be the corresponding point names.

## 4. Error Messages

There are several different types of errors that can occur when running CIFPLOT. One type concerns specification errors (i.e. command line errors). This occurs when you call CIFPLOT incorrectly, for instance, by referring to a non-existent file as the CIF source file, or by using a non-existent flag. When this happens, a message is printed out and the program terminates.

Another type of error is CIF syntax errors and questionable semantic constructs found in your CIF file. This type of error can fall into three different classes: warnings, recoverable errors, and fatal errors. When possible, CIFPLOT will print out the line on which the error was discovered and an indication of where on the line the error occurred. The error message is preceded by the word 'warning', 'error' or 'Error'. Messages preceded by 'error' are recoverable errors which, though incorrect, CIFPLOT can deal with. Messages preceded by 'Error' are fatal errors which cause CIFPLOT to quit without plotting anything after it has finished reading your file. Warnings indicate constructions that, while legal in CIF, are almost always errors or simply bad practice (such as symbol renaming).

Another type of error that might occur is a runtime or internal error. Internal errors indicate bugs in CIFPLOT and should be brought to the attention of the program maintainer. Runtime errors occur when something that CIFPLOT depends upon does not work properly. For instance, if CIFPLOT is plotting and someone turns off the plotter then a runtime error will occur.

## 5. Miscellany

This section goes over details which, though important, do not belong in any particular section.

### 5.1. CADRC Files

As part of its initialization procedure, CIFPLOT reads two files, if they exist. These files are '~cad/.cadrc' and the '.cadrc' file in the users home directory. From these files CIFPLOT can set default values for several parameters. The '.cadrc' files are ASCII text files that can contain several command lines. Each command line begins with a keyword that identifies the command type. If CIFPLOT does not recognize the keyword, the command line is ignored. By convention, there is no distinction made between upper and lower case letters in the keyword. The next few paragraphs discuss some features of the '.cadrc' file.

You may find that there are certain options in CIFPLOT that you always specify. For example, you may always suppress symbol names and always plot on the Versatec. Before parsing the command line CIFPLOT looks for a line in the '.cadrc' file with the keyword 'cifplot'. When seeing this, it will parse the rest of

the line for command line options. For the above example the '.cadrc' line looks as follows:

**cifplot -l symbolname -W**

Do not try to put CIF files on this line, since they will be ignored.

As discussed in section 2.4 it is possible to set the default unit size on the command line. This may also be set in the '.cadrc' file by including a line of the form:

**unit** *number*

The default unit is set to *number*. Setting units on the command line overrides the units set in the '.cadrc' file. This is useful for lambda based designs in that it allows the user to specify a default size for lambda, but if he gets a CIF file where lambda is a different size, he can switch the units on the command line.

By default, the maximum length of a plot is 8 feet. This may changed by including in the '.cadrc' file a line of the form:

**maxlength** *length*

*length* is the new maximum length in feet.

Normally, when looking for font files (these are the files that specify how text is to look) CIFPLOT looks in the directory '/usr/lib/vfont'. If you wish CIFPLOT to look elsewhere for font files you must include in the '.cadrc' file a line of the form:

**fontdir** *dirname*

*dirname* is the name of the directory to look for font files.

CIFPLOT creates its temporary files in '/usr/tmp'. You may want it to create the temporary files elsewhere. To do this enter a line in the '.cadrc' file of the form:

**tmpdir** *dirname*

CIFPLOT will then create its temporary files in *dirname*.

CIFPLOT has been set up so that it knows about the Varian and Versatec plotters. If another device becomes available or if one of these plotters is replaced, it may be necessary to change the specification of the devices. Entries in the '.cadrc' file to do this have the form:

**Device** *DevCh xmax ymax resolution DumpProg*

*DevCh* specifies the device. 'V' for Varian, 'W' for Versatec, and 'U' for user device. The command line options for these are **-V**, **-W**, and **-U**, respectively. *xmax* and *ymax* are the maximum number of dots in the x and y directions. The field *resolution* is the resolution in dots per inch. *DumpProg* is the program that will dump the file onto the device.

## 5.2. Installing CIFPLOT on Other Machines

CIFPLOT depends on the availability of a number of files and auxiliary programs in order to run. For instance, it requires a font file in order to be able to plot text, it requires a program to dump its raster file onto the plotter. These dependencies can make the program hard to move from one machine to another. It is desirable to be able to transport just the program's binary and not the sources, in order to save disk space and not have to worry about non-compatible sources. To achieve this machine independence, CIFPLOT requires that there exist a pseudo-user called 'cad'. This allows CIFPLOT to refer to files and programs in '~cad'. Within the '~cad' directory CIFPLOT expects to find the file '.cadrc' for initialization. It also expects to find the subdirectories 'lib' and 'misc'. In '~cad/lib' are auxiliary programs such as the programs to dump the raster file to the screen, or to drive graphics terminals. In '~cad/misc' are miscellaneous files, such as the log file discussed in the next section. Experience has shown that after setting up the initial '.cadrc' file, the program's binary is easily transportable to other systems.

### 5.3. Log Files

Each time CIFPLOT is run, an entry in the file '~cad/misc/log' is made. This file is used to gather statistics about CIFPLOT and is useful for finding bugs. This file can also be used to find out the status of jobs you have recently run. Often it is necessary to run big jobs overnight. When you come back in the morning, you can tell by looking at this file if your job ran to completion or for some reason failed. Since over time this file can get rather large you probably do not want to look at the entire file. The UNIX command 'tail' is useful here because it only prints out the last few lines.

## 6. Examples of Use

This section gives some hints for using CIFPLOT in circuit layout. Often you will want to call CIFPLOT with a fairly long list of options. A good way to do this is to call CIFPLOT through a shell script. Suppose you usually do not want bounding boxes about symbols, or symbol names at the top of each symbol but you do want a grid every 10000 CIF units. You can create a file called 'Plot' that contains the following:

```
# shell script to call cifplot with desired options
cifplot -l bbox,symbolName -g 10000 ($argv)
```

This script can be used to call CIFPLOT with the desired options. Any other options can be placed on the command line to 'Plot' and these will be interpreted by CIFPLOT. If you want to send output to the Versatec the command line would now look like this:

**Plot -W circuit.cif**

Setting the scale of all your plots to some fixed value is a good idea since it helps give an idea about the sizes of the actual circuit. For lambda based design rules as described in [Mead & Conway] it is recommended that the scale be set so that it is at least 0.04 inches per lambda. So for a 2 micron lambda the scale should be set to at least 0.02, or 500x. At this scale all features and overlaps are clearly visible.

## 6.1. Visual Design Rule Checking

In order for a chip to have any reasonable chance of being correctly fabricated it must satisfy certain design rules. If there is no program to automatically check the design rules then this must be done by visual inspection. The job of design rule checking can be made easier if checkplots are made with only a few layers visible and the stipple patterns are set so the design flaws are easy to spot. To check minimum width and minimum separation rules, the layer being checked should be plotted by itself, with the layer plotted in solid black. To do this you can define a file called 'black.stp' that contains the following stipple definition:

```
"NM" 0xffffffff 0xffffffff 0xffffffff 0xffffffff
    0xffffffff 0xffffffff 0xffffffff 0xffffffff
"NP" 0xffffffff 0xffffffff 0xffffffff 0xffffffff
    0xffffffff 0xffffffff 0xffffffff 0xffffffff
"ND" 0xffffffff 0xffffffff 0xffffffff 0xffffffff
    0xffffffff 0xffffffff 0xffffffff 0xffffffff
```

To check the metal layer call CIFPLOT as follows:

**cifplot -P black.stp -l ND,NP,NC,NB,NI,NG,alltext -s 500x outreg.cif**

The resulting plot is shown in figure I-1. It is important that the -l option follows the -P option since the -P option will redefine the layer names and make them visible. (This is consistent with the feature that allows you to override previously set command line options.)

It is useful to check interactions between layers. For instance, cuts must be covered by metal and either poly or diffusion. This can be checked by making a plot with only metal and cut visible, and making another with just poly, diffusion, and cut visible. To get these plots type

**cifplot -l NB,NI,NG,NP,ND -s 0.04 -b "Metal and Cut" file.cif**
**cifplot -l NB,NI,NG,NM -s 0.04 -b "Poly, Diffusion, and Cut" file.cif**

Figure I-2 is a plot with just the poly, diffusion, and cut layers shown. With a little practice it becomes easy to spot design rule violations in plots.

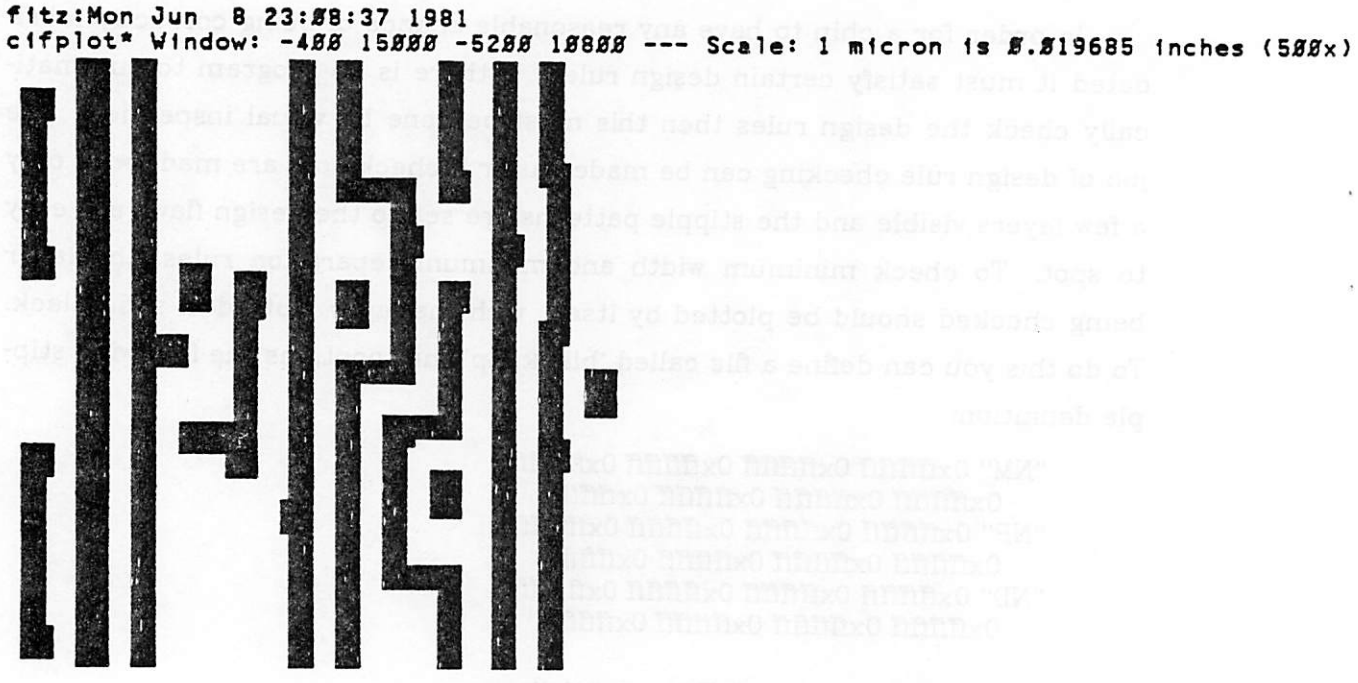


**Fig. I-1.** Plot with just metal layer. (Stipples set to black.)

### 6.2. Designing for other Technologies

By default CIFPLOT recognizes layer names for only NMOS technology. It is possible to adapt CIFPLOT to other technologies beside NMOS. It is necessary to define new layer names and design your own stipple patterns for each new layer. A general set of mask level names and of more abstract conceptual CIF layers used by the designer to specify the desired features in silicon rather than artifacts (mask geometries) necessary to produce them has been proposed in [Séquin 81]. A corresponding set of stipple patterns can be found in the file '~cad/misc/cmos.stp'.

### 6.3. Plotting with Node Numbers

The circuit extractor (i.e. CIFPLOT invoked with the **-X** option) creates a file of node numbers in CIF format. A plot with these node numbers is useful in simulating the circuit. The command to make such a plot is the following:

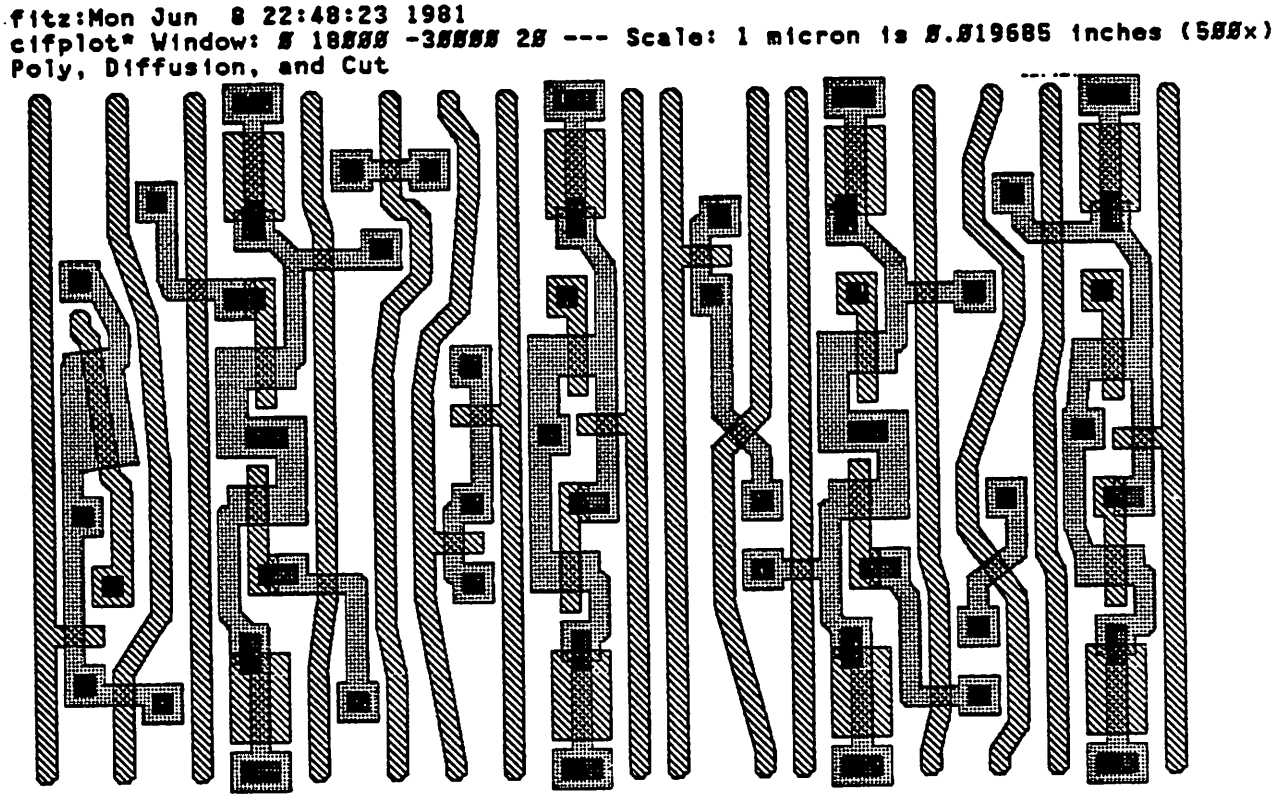


**Fig. I-2.** FIFO cell with just poly, diffusion, and cut plotted.

**cifplot circuit.nodes circuit.cif**

The scale should be set to at least 250x so that node numbers don't start writing over each other. Figure I-3 shows a plot with node numbers.

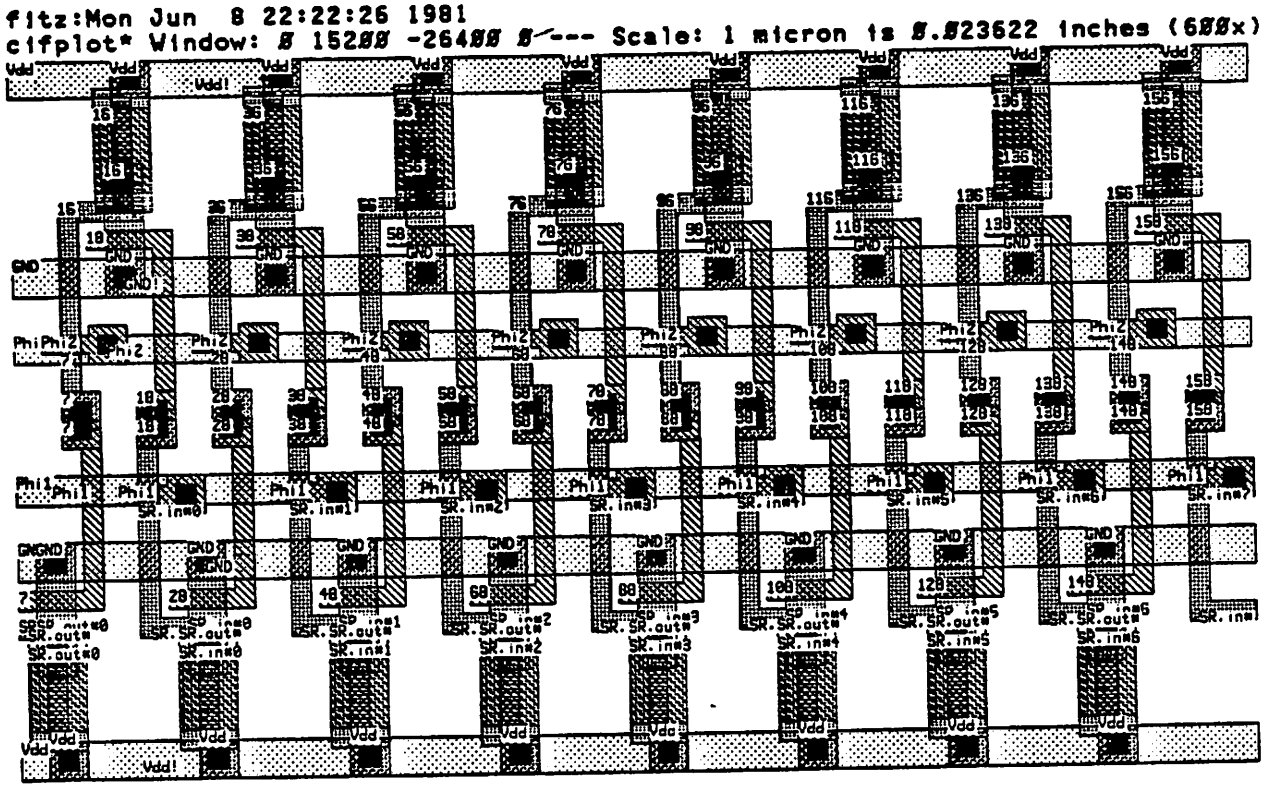


**Fig. I-3.** Eight bit shift register with node numbers.

## Part II: Implementation

### 1. Introduction

CIFPLOT can be divided into five major parts; the controller, the parser, the interpreter, the plotting routines, and the utility routines. The utility routines include storage allocation, error reporting, and so on. These routines are fairly simple and will be discussed only briefly.

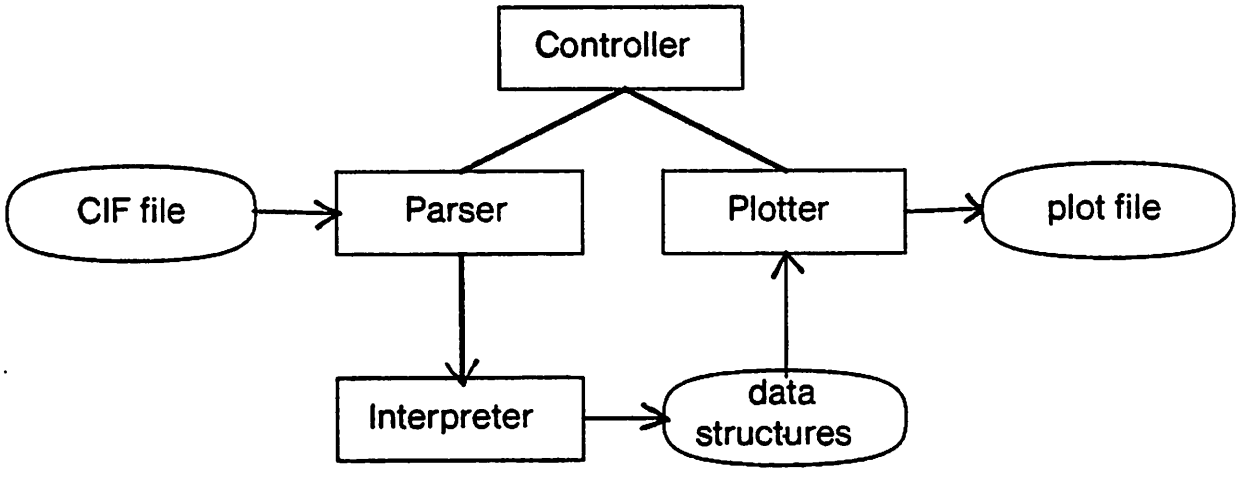


**Fig II-1.** Flow of Control

Figure II-1 is a diagram of the flow of control of CIFPLOT. The *controller* is the main guide of the program. The controller calls the parser and the plotting routines. Included in the controller are the initialization routines and command line interpreter.

The *parser* includes the routines that read and scan the input. It parses the input, reporting any syntax errors, and calls upon the appropriate interpreter routines for each CIF command. It also handles the context switches necessary for the include command and multiple file input. The actual parser is written in YACC[Johnson], a general purpose parser generator.

The *interpreter* is responsible for setting up data structures, computing the bounding boxes of each object, and reporting semantic errors (recursive calls, zero width wires, etc.). The bulk of the code is in the interpreter since it must

set up the data structures in a highly efficient form for plotting.

The responsibilities of the plotting routines include maintaining object lists in sorted order, breaking objects into more primitive objects, and finally the translation from geometric objects to bit maps. The next few sections examine each of these modules in detail.

It is useful at this point to review the structure of CIF. CIF 2.0 is a simple graphics language. Geometric primitives include boxes, roundflashes, wires, and polygons. Very few restrictions are put on these geometric primitives; boxes can be specified at arbitrary angles of rotation, polygons can have an arbitrary number of sides, and can even be self-intersecting. Each primitive element must have an associated layer. The layer command sets the current layer. Defining primitive elements before any layer has been set is an error. These geometric primitives may be used to make up CIF symbols. Each symbol definition specifies a symbol number, which is used as the symbol's name. A symbol call specifies a symbol number and a transformation to apply to the symbol. The transformation can include translation, mirroring about the X or Y-axis, and a rotation through an arbitrary angle. Symbols may contain calls to other symbols. However, a symbol definition can not be contained in another symbol (i.e. Symbol definitions do not nest). Geometric primitives and calls not included in a symbol definition are called *top level elements*.

Within a symbol there may be calls to symbols not yet defined. This is called forward referencing. Once a symbol is called at the top level, however, all forward referencing must have been resolved within that symbol, and any symbol in the calling chain. In order for a symbol to become instantiated, it must be called, either directly or indirectly, from the top level. Symbols may not be called recursively.

In order to avoid symbol numbering conflicts when combining CIF files together, there is a 'Delete Definition' command. The 'Delete Definition' command specifies a number that causes symbols with that number or greater to become undefined. This command allows a CIF description to define symbols and make top level calls to the symbols, and then delete all the previously defined symbols. Now symbols can again be defined without worrying about the old symbol numbers. This command may not appear in a symbol definition.

It is also considered legal, though bad practice, to redefine symbols. If symbol *n* has been defined and later another symbol definition uses *n*, any call now made to *n* refers to the new symbol definition.

CIF has a free format syntax. In general, as long as there is enough syntax to disambiguate a command it is legal. Lower case letters are treated as blanks. Any upper case letter may occur almost anywhere within a command.* For example, the command for a box located at the origin with length 10 and width 4 is the following:

```
B 10 4 0 0;
```

But this can also be expressed as any of the following:

```
Box with Length = 10 Width = 4, Located @ 0,0;
BOX 10.4,0.0;
BIRD10CAT4X0ZZZ0;
the quick brown fox etc ... B10+4&0/0;
```

In addition to standard CIF a number of local extension commands are recognized. These were added to increase the usefulness of CIF as an IC design language. These extension commands include text on plot commands, symbolic names, and arrays.

One of the major goals of CIFPLOT was to have it recognize full CIF 2.0. Great care was taken to implement full CIF, even when some of the CIF features did not seem to be worth the implementation effort. By implementing full CIF, an estimate of the cost of the various features was obtained. We are now better equipped to examine which features are truly useful, and which are unnecessary. This may prove valuable in the design of the next version of CIF and other IC specification languages.

## 2. The Parser

The parser is divided into three conceptual levels. The bottom layer, called the *reader*, reads characters from the CIF file and stores them in a line buffer. This layer gets the next CIF file upon reaching the end of the current file. It also can be called upon to process the include files, where it does the necessary

*This is a generalization. See [Hon & Séquin] for the exact syntax.

stacking operations. Upon reaching the end of the included file, reading begins in the interrupted file where it left off. Above this layer, all the other routines simply see a stream of characters, totally unaware of what files they came from.

The next layer is the *scanner*. The scanner accepts the stream of characters from the input layer, and looks for lexical tokens to send to the parser. Unfortunately, because of the definition of CIF, there are not many lexical tokens it can look for. When several white space characters or lower case letters appear together, the scanner can compress these to a single blank. When it finds a left parenthesis, it looks for the matching right parenthesis, counting the nesting levels as it goes. It then sends a single comment token to the parser. Other than these two compressions there is not much more it can do. Parsing time could be reduced if the scanner could recognize integers. CIF, however, does not distinguish between identifiers and integers. A layer name can be any sequence of up to four uppercase letters or digits. The strings "POLY", "15", and "56G" are all valid layer names. The scanner, therefore, leaves the job of converting strings of digits to integers up to the parser.

The final layer is the actual parser. The parser was written with YACC[Johnson], a parser generator. Much of the parser was a straightforward translation from the formal syntax of CIF 2.0 given in [Hon & Séquin] to the YACC description. The CIF definition given in [Hon & Séquin] was extended to include extension commands. (See appendix B for the formal definition of the extension commands.) In standard CIF, lower case letters are equivalent to blank spaces. In the extension commands, however, lower case letters can not be treated as blanks. In order for the include command to work properly, lower case letters must be processed since the UNIX operating system distinguishes between case in file names. Further, names become more readable if upper and lower case letters are permitted.

Passing all lower case letters to the parser could potentially slow it down when reading standard CIF commands since it would have to interpret each lower case letter encountered. Instead, lower case letters are usually treated as blanks in the scanner. Whenever the parser enters an extension command, it sets a flag. When this flag is set the scanner sends all lower case characters to the parser.

The YACC[Johnson] parser generator is used for parsing the CIF file. The translation from the formal definition of CIF syntax to the YACC description was a straightforward process. The parser had to be augmented with error reporting and recovery routines. By far the most time spent in writting the YACC parser was spent writing the error handling code.

## 3. The Interpreter

The main responsibility of the interpreter is to set up the data structures needed by the plotting routines. It is called by the parser to interpret the CIF being read. The interpreter, in addition to storing the data for later use, must do several consistency checks on the data such as checking for recursive symbol calls, and checking that symbols are defined before they are used. After all the CIF has been read the interpreter computes the bounding box of the circuit.

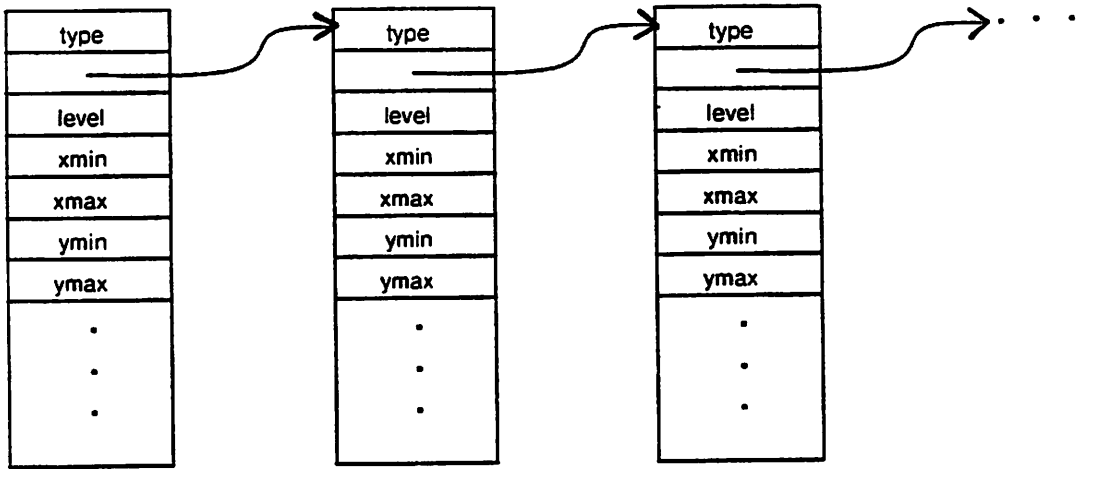


**Fig. II-2.** Command Data Structures

### 3.1. Data Structures

Each CIF object is translated into a structure called a 'Command'. This structure varies according to the type of object it holds. The beginning of the structure is always the same, however. The first field is an integer that indicates the type of command (Definition, Polygon, Wire, etc.) The next field is a pointer to another structure of type 'Command'. This allows us to string Commands

together in a list. The next field is called 'level'. It indicated what layer the geometric primitive is to be plotted on. The next four fields specify the bounding box of the command. The bounding box indicates the minimum and maximum range of the geometric primitives in the 'Command'.

After these fields the remainder of the structure depends on its type. A polygon, for instance, just has a pointer to a list of points that are the vertices of the polygon. A wire has both a list of points and the wire's width. Most of the geometric primitives contain just the information necessary to describe themselves.

Two of the more important types are 'calls' and 'symbols'. A call contains the call's transform, the identifying number of the symbol it calls, and a pointer to that symbol. A symbol contains a pointer to a list of 'Commands' that make up the symbol, a pointer to a list containing the numbers of the symbol that call it, the symbol's status, and the symbol's name.

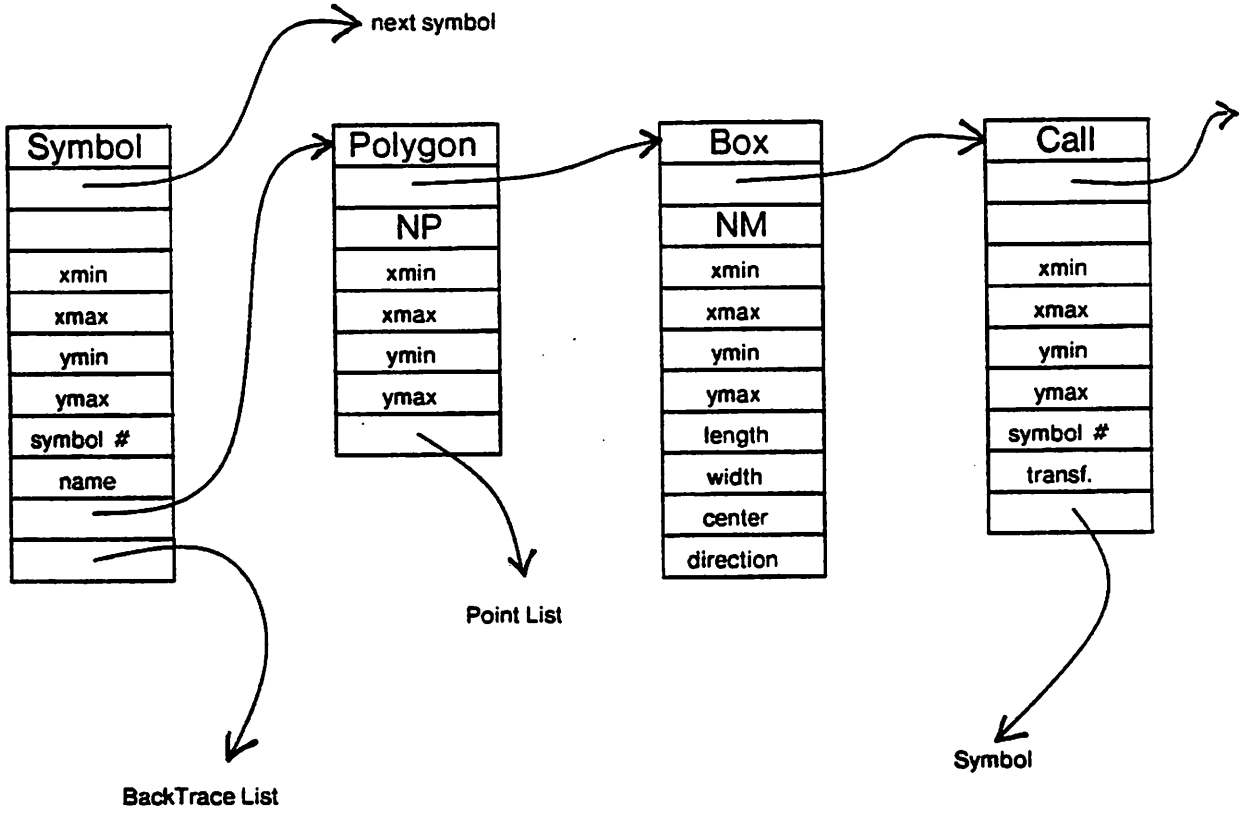


**Fig. II-3.** Data Structure for Symbol Command

Each CIF object may or may not be part of a symbol definition. If it is not then it is considered called at the top level. For uniformity, though, CIFPLOT treats these objects as though they were part of a special symbol. This allows all objects to be placed into symbols. Whenever a 'Definition Start' command is encountered a new symbol 'Command' is created and placed into the symbol table. All further commands are put onto the list pointed to by the new symbol 'Command'. This continues until a 'Definition Finish' command is encountered. This causes further commands to be placed on the special list of top level symbol.

### 3.1.1. Symbol Table

Once a symbol has been defined it is necessary to store the symbol in the symbol table. The symbol table is a hash table of pointers to a list of Symbol-Headers. Access to the symbol table is controlled by two functions, *StoreSymbol(x)* and *FindSymbol(n)*. *StoreSymbol(x)* places the symbol header $x$ into the proper symbol list in the hash table. *FindSymbol(n)* returns a pointer to the symbol header for symbol $n$.

### 3.1.2. Call Transformations

With each CIF 'Call' command there is an optional transformation field. The transformation field can specify rotations, mirrors, and translations to be performed on the called symbol. These transformations can all be represented as a single 3 by 3 matrix. (See [Newman & Sproull] for details.) Each call therefore maintains a pointer to a 3 by 3 matrix that represents the transformation to be applied to the symbol.

### 3.1.3. Layers

CIFPLOT recognizes the standard NMOS layers defined in [Mead & Conway]. In addition CIFPLOT lets the user define his own layers. Layers are stored in a structure called an 'LCell'. A 'LCell' contains a link to the next 'LCell', the layer name (such as 'NM' or 'NP'), the stipple pattern to use for this layer, the layer number, and a flag to indicate whether or not the layer is visible. For each CIF layer, all the standard layers and any user specified layer, a new 'LCell' is added to the layer list. To find if a layer is defined this list must be searched. This list

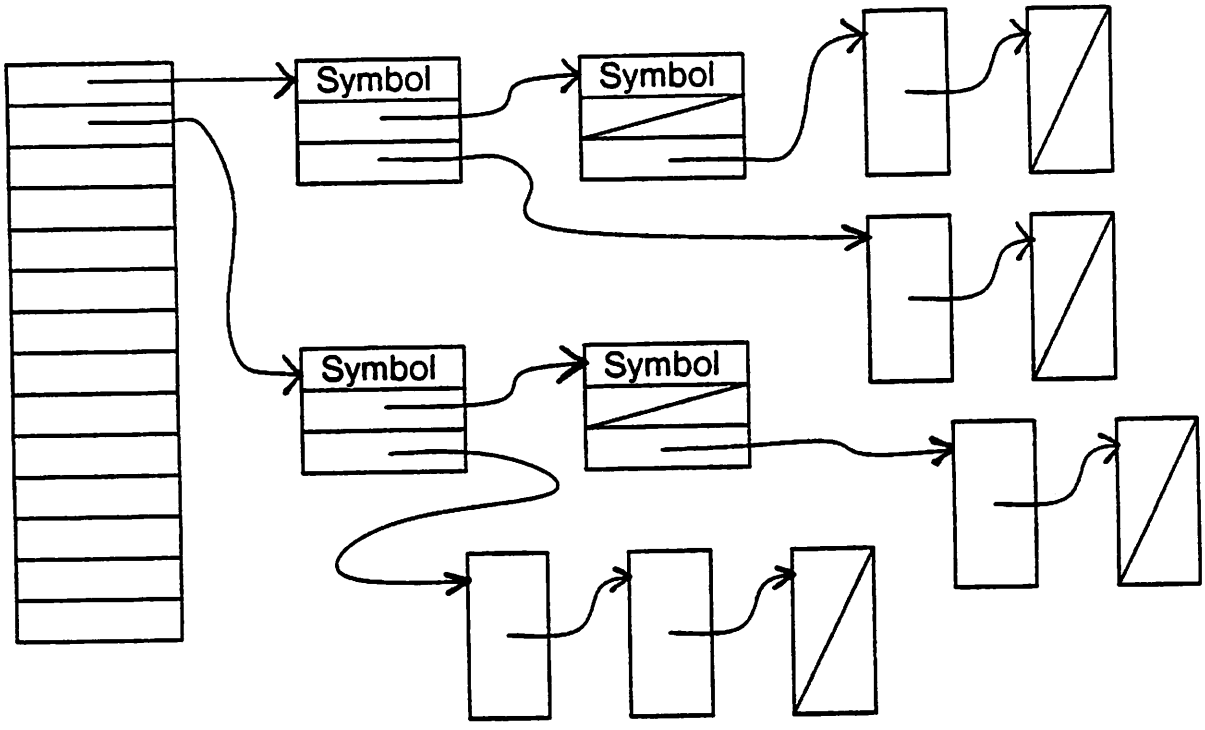


**Fig. II-4.** Symbol Table

never gets very long and experience has shown that not much time is spent searching it.

### 3.2. Forward References and Recursion Detection

CIF requires that symbols be defined before they are used. (That is, they must be defined before there is a direct or indirect call to them from the top level.) The order in which the symbols are defined is unimportant. In order to deal with forward references CIFPLOT does not try to establish links for call commands until a symbol is used. Checking for recursion, which is illegal in CIF, is also not done until a symbol becomes used. If a symbol is recursive but never used, no error is detected.

When a symbol is defined, it is given the status 'UNUSED'. Whenever a symbol call occurs at the top level the procedure *Examine* is called with the symbol that is referenced as a parameter. If the symbol is 'UNUSED' then its status is

changed to 'ACTIVE', and *Examine* is called on every command in that symbol. Thus if there is a call in the symbol, the symbol referenced in that call is examined. When *Examine* returns the symbol's status is changed to 'USED'. If *Examine* is called on a symbol that has status 'ACTIVE', then that symbol must be called recursively. In this case *Examine* issues an error message and returns. If the symbol does not exist an error message is issued to that effect.

*Examine*, after it has checked the status and taken the appropriate action, sets a link from the call command to the symbol, then adds that call command to the symbol's 'BackTrace' list. *Examine* is the only procedure that sets these links and it is only called when the symbols are used. At this point, to be legal CIF, all the symbols must be defined. Therefore, there is no problem with forward references. After all this has been done *Examine* computes the bounding boxes and returns.

### 3.2.1. Symbol Redefinition and 'Delete Definition' Commands

In CIF it is legal, although not considered good practice, to redefine symbol names. Redefinition can cause a number of problems. The effect of redefining one symbol name can ripple through the CIF program, changing the meaning of several symbols, Consider the following CIF command sequence.

```
DS 1;
(CIF text goes here);
DF;
DS 2;
C 1;
(Symbol #2 calls symbol #1);
DF;
C 2; (Symbol #2 is called which calls the above Symbol #1);
DS 1;
(Symbol #1 is redefined);
DF;
C 2; (Call symbol #2 which calls the new symbol #1);
E
```

Since symbol #2 calls symbol #1, and since symbol #1 has been redefined, the second call to symbol #2 produces a different plot than the first call. For this reason redefining a symbol causes all symbols that call it to change. The 'Delete Definition' command can cause the similar problems whenever it leaves dangling references.

In order to cope with these problems, CIFPLOT must keep track of all the references to a symbol. Whenever symbol *x* is redefined, all symbols that previously referred to symbol *x* must be changed. Further, all those symbols that were changed cause all symbols that refer to them to be changed, and so on.

It is necessary for every symbol to have associated with it a list of symbols that call it. Whenever a symbol is redeclared the list associated with the old symbol is checked to see if it is empty. If not, an advisory message is issued and the procedure *CopyDelete* is called on each symbol that references the redefined symbol. *CopyDelete* makes a copy of the symbol that was passed to it and marks the old symbol deleted. It then calls itself recursively on all the symbols that reference the just deleted symbol. This causes it to trace through all the references, direct and indirect, to the redefined symbol.

The strategy for 'Delete Definition' is similar. Whenever a 'Delete Definition' command is encountered all symbols to be deleted are removed from the symbol table, and placed on a list called 'RemovedSymbols'. All these symbols are then marked deleted. *CopyDelete* is called on all the symbols that reference any of the symbols in 'RemovedSymbols'. If *CopyDelete* encounters any symbol that is not marked deleted, then there are dangling references and an advisory message is issued.

### 4. Plotting Routines

CIFPLOT tries to be as device independent as possible. Early versions of CIFPLOT directly controlled the plotting devices. The protocols for opening, closing, and controlling the plotter were coded into the program. This device dependence limited CIFPLOT's usefulness.

The current implementation of CIFPLOT knows about three types of devices: raster plotters, vector plotters, and trapezoid plotters. The Versatec and Varian plotters are examples of raster plotters. For these devices CIFPLOT must specify for every pixel whether it is on or off. A pen plotter is an example of a vector plotter. For these devices CIFPLOT must specify the endpoints of each line and say what layer the line is on. Trapezoid plotters are devices that can display trapezoids, such as a raster graphics terminal. For these devices CIFPLOT must specify the vertices of the trapezoid and say what layer the trapezoid is on.

CIFPLOT needs to know five things about the plotter: the plotter type (raster, vector, or trapezoid), the number of pixels in the horizontal direction, the number of pixels in the vertical direction, the resolution in pixels per inch, and the program that can display the plot file on the plotter. The program that displays the plot file on the plotter is called the *display program*. With this information, CIFPLOT creates a plot file for the output device and then calls the display program. The display program is usually quite simple and frees CIFPLOT from having to know about the details needed to drive the display device.

The next few sections will concentrate on plotting for raster plotters. Plotting for the other two types of devices is quite similar.

## 4.1. Instantiating Symbols

Initially each command called at the top level is placed on the 'UnActiveList'. (The 'UnActiveList' is actually implemented as a pseudo-hash table, but it is easier to discuss it as a list.) The commands are sorted by minimum x value of their bounding box in device coordinates. The first element on the 'UnActiveList' is taken off and is passed to the function *Activate*. The basic job of *Activate* is to break objects down into more primitive objects.

If the element was a 'Call', then the called symbol and the call's transformation are passed to the function *Instantiate*. *Instantiate* applies the specified transformation to each object in the symbol. Each transformed object is sent to the selector routine. The selector routine checks to see if the object is visible. Usually all this involves is checking that the object is within the clipping window. If it is visible, then it is sorted into the 'UnActiveList'. (See figure II-5.) The selector routine is described later. This method prevents the 'UnActiveList' from becoming too large by not instantiating symbols until they are actually needed by the plotting routines. If symbols were kept fully instantiated, the number of objects to keep track of could become overwhelming.

If *Activate* is called with an object that is a geometric primitive, such as a box or polygon, it is broken down into edges. The edges are sent to the selector routine to see if they are visible. For edges, the selector sees if they are horizontal. If they are, they are thrown away. If not, they are sorted into the 'UnActiveList'.

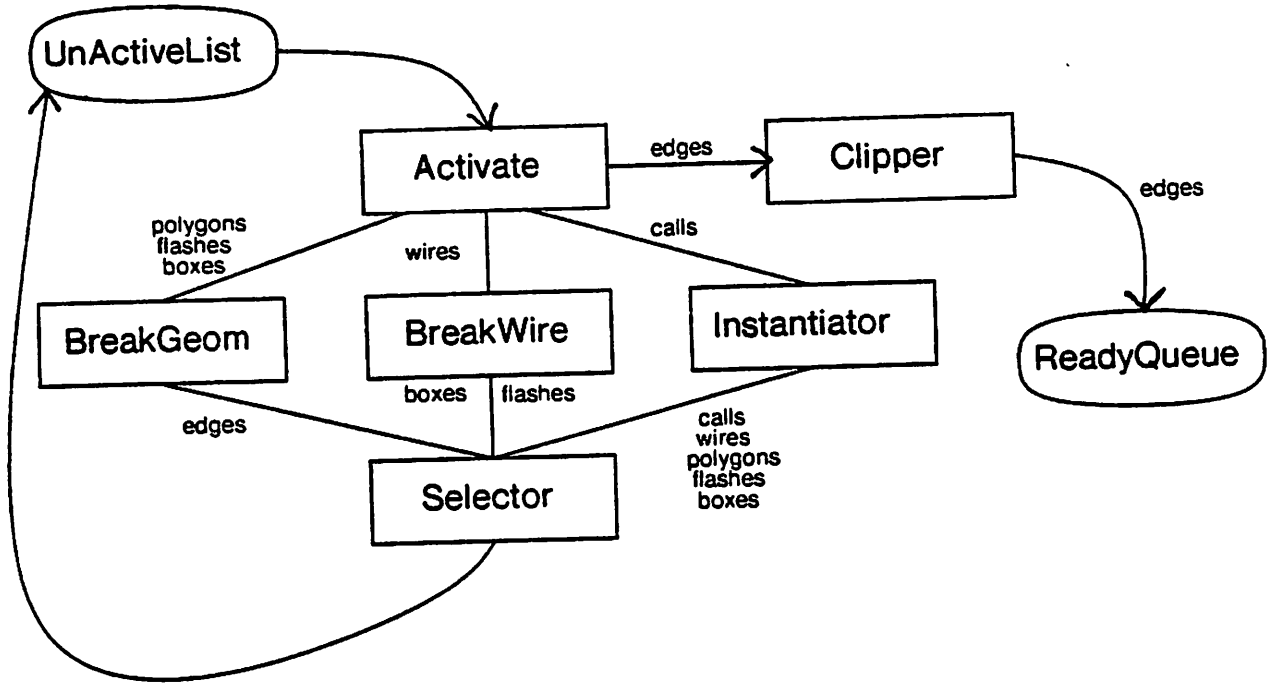


**Fig. II-5.** Activation of Geometric Objects

Finally *Activate* may be called with an edge. An edge cannot be broken down further. The edge is sent to the clipping procedure. It is clipped so that all that remains is what belongs on the display. After being clipped the edge is sent to the 'ReadyQueue' to be put on the 'ActiveList'. Actually, one 'ReadyQueue' and one 'ActiveList' is kept for each layer.

This process of taking the first object off the 'UnActiveList' is repeated until the 'UnActiveList' is empty. Once the 'UnActiveList' is empty, the CIF file has been exhausted.

### 4.2. Selection

The selector routine determines where to send elements. If the element is text then it is sent to the text clipping routine which puts it on a text list. If the element is an edge then it is checked to see that it is horizontal. If it is it is rejected, otherwise, it is sorted into the 'UnActiveList'.

Other objects, such as polygons, wires, or calls, are checked to see whether any part of the bounding box lies within the display area. If not, it is rejected. The object's layer is also checked to see if it is to be plotted. Again, if not, the object is rejected. Finally, it is checked to make sure that its depth in the calling hierarchy is not greater than the user specified limit. If the element passes all these tests, it is then sorted back into the 'UnActiveList'.

### 4.3. Sorting

As mentioned earlier, the 'UnActiveList' is not actually a list. The 'UnActiveList' is a table, where each element of the table is a pointer to a list. To enter an element into the 'UnActiveList', its minimum x value in device coordinates is computed. This number taken modulo the table size to find the appropriate list in the table. The element is then sorted onto this list. The element should be placed at the front of the list since this method of instantiating symbols tends to give us objects near the current scan line, and objects that the scan line has passed are not on the list.

### 4.4. Ready Queue

The 'Ready Queue' provides a buffer for edges between the 'UnActiveList' and the 'ActiveList'. The 'Ready Queue's main function is to provide a clean interface between the object instantiator and the scan line plotter. These two functions could be run as separate processes communicating solely through this queue.

This buffer is also useful in avoiding floating point round-off errors that would normally causes edges to enter the 'ActiveList' out of sort. Because an objects bounding box is computed in CIF coordinates and the list is sorted in device coordinates, a problem arises when sorting edges onto the 'ActiveList'. The transformation converting a symbol's bounding box to device coordinates may indicate that the minimum x value is $n$, yet the transformation for an edge in that symbol may indicate that its minimum x value is $n$-1. This type of error occurs infrequently, but can cause problems if it is ignored. The 'Ready Queue' provides a simple solution to this problem. Each edge is checked to see if its minimum x is greater than or equal to the minimum x of the last element of the queue. If it is then the new edge is added to the end of the queue. If not, the

edge moves down the queue till it finds an edge with a minimum x less than or equal to its own.

### 4.5. Maintaining the Active List

If an edge is to be added to the 'ActiveList' for a particular layer, it is taken from the queue and first placed into a separate list. All edges to be added to the 'ActiveList' are sorted into this special list. This list is then merged with the 'ActiveList' for that layer. This minimizes the number of scans of the 'ActiveList' needed to insert new elements.

As new elements are entered, the point at which the next edge in the ActiveList ends is recorded. When plotting reaches this point, the 'ActiveList' is scanned and any edges that are no longer visible are removed.

Edges are entered into the list sorted, and removed without disturbing the ordering of the other edges. The only reason that the edges in the list could get out of order is if edges were to cross. If this happens, the 'ActiveList' must be resorted. It is necessary to predict when edges will cross. Every time the 'ActiveList' is changed the point of the next intersection is calculated. A simple bubble sort is used to reorder the edges. Since usually only one or two of the elements are out of order a bubble sort is adequate.

### 4.6. Taking Advantage of Slow Scan Coherence in Drawing

It is not necessary to rescan the 'ActiveList' for each new line that is drawn. It is only necessary when a new edge is entered into the 'ActiveList', or an edge has been removed, or edges cross. Otherwise, information from the previous line is used to determine how the next line will look. CIFPLOT takes advantage of this coherence from one scan line to the next to avoid having to rescan the 'ActiveList', thereby speeding up the program.

Whenever a change is made to the 'ActiveList', it must be rescanned. This scan finds the beginning and end points of each figure. It also finds the incremental change of the beginning and end points in the y-direction for each scan line. Thus to draw the following scan lines it is only necessary to increment the beginning and end points by the appropriate amount, and place them on the next line. This can be done until the next change in the 'ActiveList'.

Information telling about the next change is available at this point. It is the minimum of the following three events: the next time an edge enters the 'ActiveList', the next time an edge is removed from the 'ActiveList', and the next time two edges intersect.

**Part III: Experience**

CIFPLOT has been running for 18 months now. It is widely used by circuit designers in many types of technologies. In general these designers tend to be pleased with the program. The quality of its plots is quite high. The major problem is its speed. CIFPLOT tends to require a lot of CPU time. Below is a table comparing the time and space requirements of CIFPLOT with VCIF, CIFPLOT's predecessor.

**performance of cifplot vs. vcif**

| file name | cifplot | | | vcif | | |
|---|---|---|---|---|---|---|
| | user time | system time | memory | user time | system time | memory |
| sfifo | 334.6 | 8.4 | 53+147k | 1352.1 | 17.7 | 26+134k |
| xfifo | 64.3 | 2.1 | 52+127k | 303.4 | 4.2 | 26+133k |
| cherry | 206.1 | 5.2 | 52+205k | 712.4 | 7.1 | 26+196k |
| rcherry | 856.8 | 16.5 | 53+214k | 1034.5 | 7.1 | 26+256k |
| becker | 136.3 | 10.6 | 51+138k | 552.6 | 9.3 | 26+109k |
| logic | 392.9 | 16.4 | 52+282k | 1125.8 | 27.9 | 26+277k |

Although CIFPLOT is about a factor of three faster than VCIF, it is slower than we would like. As circuits get more complicated, the turnaround time gets worse. Large plots can take more than two hours of CPU time.

There are a number of approaches to cutting down on the processing time used. One way is to restrict the designs to a subset of CIF, such as allowing only manhattan features. This allows several shortcuts to be taken in the program and can dramatically improve the run time performance. Another approach would allow lower quality plots. CIFPLOT tries to represent the circuit as well as it can, outlining the border of each layer. While working on a circuit, a number of scratch plots are made. These plots give the designer a feel for how the circuit is progressing, and helps him make changes to the circuit. It is more important to get these plots out quickly than to get a high quality plot. Higher quality plots could be made at less frequent intervals when the looks of the plot outweigh the turnaround time.

Almost all CIF is now generated by programs. This reduces the need for a full CIF parser on CIFPLOT. Parsing full CIF is expensive in processor time and program size. Although it is important to have a program that does parse full CIF and checks for errors, it is doubtful that it should be done every time a file is

plotted. A program that just parses full CIF and does extensive error checking, and then reformats the file in a simpler to parse CIF format would be quite valuable.

The error reporting facilities of CIFPLOT are quite good. It is usually able to pinpoint the problem very accurately. Since most CIF is now machine generated, the error reporting facilities are somewhat overkill. A simpler scheme, which just reports the offending line with an appropriate message, would be adequate.

## Conclusion

Over the last 18 months CIFPLOT has become an essential tool for IC designers at Berkeley. By accepting general geometry specification and by allowing the user to select among many options, CIFPLOT has become accepted by a large and varied design community. Several designers have customized CIFPLOT to their individual needs by setting many of the options in their command file. New processes can be accommodated by CIFPLOT, by setting up the appropriate layer table. This has allowed CMOS designers to use CIFPLOT even though there is no internal knowledge of any CMOS processing layers. The generality of CIFPLOT, though, has caused it to use more processor time than we would like.

## Acknowledgements

I would like to thank Carlo Séquin for the original idea, and for direction and guidance throughout this project. I would like to thank Martin Newell of Xerox PARC for many ideas used in the implementation. I would like to thank Barbara Wood for carefully proofreading several drafts of this report. I would also like to thank the many users of CIFPLOT for their valuable comments and suggestions. Figure I-1 is part of an ALU designed by Korbin Van Dyke and myself, figure II-2 is a FIFO cell designed by Howard Landman, and figure II-3 is a shift register designed by Jim Cherry of MIT. This work was funded by ARPA, whose support is gratefully acknowledged.

**References**

[Hon & Séquin]
R. Hon, and C. H. Séquin, "A Guide to LSI Implementation" Xerox PARC, 1980

[Johnson]
S. T. Johnson, "YACC: Yet Another Compiler Compiler" Bell Laboratories, Murry Hill, New Jersey July 1978

[Keller]
K. Keller "Tutorial for KIC 2 - A Graphics Editor for Integrated Circuits", Available from author, March 1981.

[Kohn]
J. Kohn, "Implementation of STIF 1.1 and Interfacing to Plotting Routines", Master's Report, U.C. Berkeley Dec 3, 1980

[Krause]
J. Krause, "CIF Interface", ERL Report, Dept EECS, U.C. Berkeley 1979

[Lesk & Schmidt]
M. E. Lesk and E. Schmidt, "Lex - A Lexical Analyzer Generator" Bell Laboratories, Murry Hill, New Jersey 1978

[Mead & Conway]
C. Mead, and L. Conway, *Introduction to VLSI Systems*, Addison-Wesley, 1980

[Newell & Séquin]
M. E. Newell, and C. H. Séquin, "Inside Story on Self-Intersecting Polygons" *LAMBDA* Spring 1980

[Newman & Sproul]
W. M. Newman, and R. F. Sproul, *Principles of Interactive Computer Graphics, Second Edition*, McGraw-Hill 1979

[Ousterhout]
J. Ousterhout "Editing VLSI Circuits with CAESAR", Available from author, September 1981.

[Séquin]
C. H. Séquin, "Standard Interchange Formats for Integrated Circuit Design", Available from author, 1981.

[Séquin]
C. H. Séquin, "Generalized IC Layout Rules and Layout Representations", in Digest of VLSI 81, Edinburgh, August 1981.

[Séquin & Newell]
C. H. Séquin, and M. E. Newell, "Cutting Corners when constructing CIF wires and Offset Boundaries Around Polygons", Available from author, Spring 1980