SIL, ANALYZE, GOBBLE, BUILD Reference Manual

by C.P. Thacker, R.F. Sproull, R.D. Bates

February 20, 1981

This manual describes the Palo Alto Design Automation system, which includes SIL, an interactive illustrator; ANALYZE, a postprocessor for SIL output; GOBBLE or ROUTE, wirelisting programs; BUILD, a data-management aid, and several printing utilities.

XEROX
Electronic Operations Department
Los Angeles / Palo Alto

PALO ALTO RESEARCH CENTER
3333 Coyote Hill Road / Palo Alto / California 94304

Introduction

This document is a description of a group of programs intended to aid in the documentation and construction of digital logic systems. The primary goal in the creation of these programs was to automate a number of system-building tasks which have previously been difficult to do well, particularly in an environment which has no support staff for design documentation. In addition, the system provides a *de facto* documentation standard, since its capabilities are limited and specialized. The complete logic design system consists of: Sil, an interactive drawing program used to produce logic diagrams; Analyze, which creates a file of node interconnection and component information from the graphic representation created by the designer with Sil; Gobble or Route, wirelisting programs which merges a number of files generated by Analyze into a complete wirelist for a circuit board; and an additional program available to operate the semi-automatic stitchwelding facility at Palo Alto, given the Gobble or Route wirelist as input.

Making simple illustrations

Although the system described here was built primarily to offer help in design documentation and design automation, portions of it are useful for making simple illustrations for documents. The reader interested only in such facilities should read about Sil only.

Technology and construction process

The logic design system is specialized for a hardware construction method in which integrated circuits are assembled on standard cards, which are in turn assembled into a standard backplane. Both the boards and backplane are interconnected using Wire-Wrap, Stitchweld, or a similar interconnect scheme. The primary documentation for such a system consists of logic diagrams for the cards and an indication of the arrangement of the cards in the backplane.

Secondary documentation, which is derivable from the primary documentation, includes wirelists for the cards and backpanels, IC loading charts for the cards, and a number of summaries to aid in debugging, such as pin lists and signal name lists.

Using the design-automation system

Because most hardware-construction tasks are joint efforts, it is important that the suite of programs described in this manual are used in similar ways by all designers on the team. A number of **conventions** for using the system have been developed over time, and are offered in this manual as a help in organizing a large design. Many of the conventions have resulted from the tremendously difficult problem of managing different versions of drawings, numerous Alto files produced or required by the system, and the like.

Files you need and where to find them

This section summarizes the files you will need on your Alto in order to run the various systems described in this manual. All files are on the MAXC2 file system under the SIL directory.

SIL -- for illustration use only
- Sil.Run
- Helvetica10.Al
- Helvetica7.Al
- Template64.Al (use to draw circles and diagonal lines)
- Template64.lb5 (use to draw bold circles and diagonal lines)
- +changes to your User.cm; see UserCmSlice

SIL -- for use in logic design
- Sil.Run
- Helvetica10.Al
- Helvetica7.Al
- Gates32.Al
- Sil.Lb5 to Sil.Lb8 "loaded" from SilLibraries.dm
- LogicBlock.Sil (this is a prototype title block)

Documentation on current libraries can be found on ECLDataSheets.Press and TTLDataSheets.Press.

ANALYZE
- Analyze.Run
- TtlDict.Analyze
- EclDict.Analyze (if you are using ECL)

GOBBLE/ROUTE/BUILD
- Gobble.Run (obsolete) or Route.Run
- RouteAub.br
- Build.Run

THIS MANUAL
- SilManual.Press

SIL

Sil is a SImple Illustrator designed for (but not limited to) the creation of logic diagrams. It allows the user to create pictures composed of variable width horizontal or vertical lines, and text strings in one of three user-specified fonts. Text fonts may be specified with bold face and/or italic face, and Sil will modify the screen font to so indicate the face. In addition to the three text fonts, a special font is available which includes most of the symbols on a standard logic designer's drafting template. Sil also allows the creation of *display macros*, which are composed of a number of objects that the user prefers to name and manipulate as a single unit. In addition to user-specified macros, Sil can use symbols from any of five *Macro Libraries*, which are used to hold commonly-used symbols (e.g. logic symbols). Drawings created with Sil may be formatted and printed on Press printers using the Sil/H to create a press file "Sil.press" and send it to a printer by invoking Empress.run. Sil/P will also create the "Sil.press" file but will not invoke Empress.run.

Although Sil is (was) simple, it is certainly appropriate for new users to read this description completely at least once before using Sil, as many of the commands and features are not obvious.

Starting SIL

The first time you use Sil, or after you have changed font names in user.cm or changed libraries, you must run Sil/I. This will cause Sil to read your user.cm entries and build up two scratch files (Sil.fps and Sil.fonts) for future fast access.

Sil is subsequently started by typing "Sil(CR)" or "Sil filename(CR)" to the Executive. After a few seconds, the screen will be cleared, and a single status line of text will appear at the top. If a filename is specified then Sil will immediatly Input that file.

A typical status line is:

GLMF: 4111b TF0N Space: 12000 Selections: 0 X: 0 Y: 0

These quantities have the following significance:

GLMF: 4111b

G: is the current *Grid*. The cursor and all objects are constrained to lie on points which are multiples of the grid spacing. The grid may be set to 2**n (n=0-7) by typing ^G (Control-G)n. The standard grid for design-automation work is 4, i.e, ^G2.

L: is the current width for any lines added to the picture. The line width is set to n by typing ^Wn (n=1-9).

M: is the current magnification. The magnification is changed using the ^E command.

F: is the current *font*. Sil has ten fonts (0-9), the first four of which are text fonts which may be set by the user (see "Fonts"). Fonts four through nine are macro names (see "Macros"). For fonts, an optional "b" and/or "i" may be displayed to indicate that text entries will be added in bold or italics face as appropriate.

TF0N

The "**T**", is a flag to indicate that macro expansion will go one level deep only (*see "Macros" under ^H and ^L*).

The "**F**" is a flag to indicate what vertical offset is to apply when a carriage return is typed (*see ylock flag under ^Y*).

The **"0"** indicates the state of the storage compacter (an internal Sil function), which when non-zero indicates that Sil has not yet completed updates from that last edit. You do not have to wait for one edit to be completed before initiating the next.

The **"N"** indacates the default color (Neutral in this case), and is changed by the ^F command.

Space: 12000

Indicates the number of words of space remaining in the storage pool for objects. A line requires five words for its description, a string requires five words plus one word for every two characters in the string.

Selections: 0

Indicates how many objects are currently *selected*. Selected objects are displayed gray (halftone) rather than black. There are three types of objects in Sil, text strings, vertical or horizontal lines, and backgrounds. Strings may contain normal characters or macro names, but an entire string must be in a single font (no ^F during typein), and when complete becomes a single object. Most Sil commands which affect the picture change only the selected objects.

X: 0 Y: 0

These are the coordinates of the cursor at the time of the last depression of the left (or top) mouse button. X=0, Y=0 corresponds to the upper left corner of the screen; Y increases downward, X increases to the right. If any mouse button is kept depressed while moving the mouse, the frequency of refresh of the status line is increased and the coordinates of the *cursor* are displayed. This is provided to aid in debugging drawings prepared for Analyze, which reports errors in terms of screen coordinates.

In addition to these quantities, Sil displays messages and interacts with the user via information which is placed at the end of the status line when necessary.

Mice, Marks, and Modes

Mice have three buttons, and come in two types. We will refer to depressions of the left or top (closest to the cord) button as **mark**, depressions of the middle button as **draw**, and depressions of the right or bottom button as **select**. Actions occur when the button is depressed, as opposed to when it is released.

There are three special objects whose positions are controlled (usually) by the mouse:

The *cursor* is a small arrow, and moves as the mouse is moved. When the cursor is used to point at objects, the tip of the arrow should be positioned over the object.

The *mark* appears as a short vertical bar. On every **mark** (depression of the left or top mouse button), the mark is moved so that its upper left corner is coincident with the cursor.

The *origin* is a small horizontal bar. The origin provides a reference point when moving or copying selected objects.

Both the origin and the mark blink approximately twice per second.

Sil has three modes: *Normal mode*, in which lines are added to the picture or objects are selected for further action, *magnified mode*, described later, and *add text* mode, in which text strings are added to the picture. In the last mode, the message ADD TEXT is displayed at the end of the status line.

In normal mode, depressions of the mouse buttons, sometimes in conjunction with depressions of the control and/or left-hand shift keys, have the following meanings:

mark: Move the mark to the current cursor position.

shift-mark: Moves the origin to the current position.

control-mark: Moves the mark to the current cursor position, then moves all selected objects so that their origin is at the mark, selects the objects which were moved, and deselects any previously selected objects. This command is equivalent to **mark**-^X (see below).

control/shift-mark: Same as **control-mark** except that any attached lines are not stretched. See description of **mark**-^X below.

draw: Draw a line between the mark and the current cursor position. The line will be horizontal if the difference between the y-coordinates is less than that of the x-coordinates, otherwise it will be vertical. The mark is moved to the end of the new line closest to the cursor, the line becomes selected, any previously selected objects are deselected, and the origin is moved to the upper left corner of the new line.

control-draw: Moves the mark to the current cursor position, then copies all selected objects such that their origin is at the mark. It then selects all copied items, and deselects any previously selected objects. This command is equivalent to **mark-**^C (see below).

shift-draw: Deletes just the item pointed to, not the selected items.

control/shift-draw: Undeletes and selects the set of objects last deleted with **shift-draw** or **^D** (see below).

select: The object at which the cursor is pointing is selected, any previously selected objects are deselected, and the origin is moved to the upper left corner of the selected object. When objects overlap, the object with the smallest parameter is selected. If there are no objects of any type under the cursor, nothing is selected, and the origin is moved to be coincident with the cursor.

control-select: Same as select, except that previously selected objects are not deselected.

shift-select: Selects all objects which lie fully within the rectangular area bounded by the current cursor position and the mark. The origin is moved to the upper left corner of the selected area, and all previously selected objects are deselected.

control/shift-select: Deselects the item pointed to while leaving the origin at its current location.

Keyboard

In normal mode, Sil is awaiting commands. The initial character of every command is a control character. If a character is typed which is not a control character or space, ADD TEXT mode is entered. In this mode, character strings in the current font (and face) are put into the picture at the mark. Strings are terminated by typing carriage return or ESC, or by depressing any mouse button. Remember that each string is a separate object.

The following control characters are available during input to facilitate editing:

BS	Backspace one character
^A	Backspace one character
^W	Backspace one word
^Q	Backspace the entire string but don’t leave the mode
^S	Convert the last character typed into a control character (i.e. "A^S >^A)
DEL	which clears anything typed and returns to normal mode without modifying the picture.

Once a string has been entered, it can be modified via the **BS** command (see Commands below).

Commands

Sil commands consist of single control characters, control characters followed by single digits or characters, or more complex commands which carry on brief interactions with the user via messages in the status line. Only the first character of multi-character commands uses the CTRL key. Subsequent characters, if any, are typed normally. In a couple of instances, special meaning is attributed if the Control **and** Shift keys are held down simultaneously when the command key is typed. The notation used for this will be ^shA ('control shift A'). The commands are:

^**A**: Display text from an "Alternate text file". The first time ^A is typed, the user is asked to supply a file name, that file is opened, and the first text line is displayed in place of the status line. For subsequent occurrences of ^A, if the Status line still displays text from the file, then the next line of the file is displayed, otherwise the current line is redisplayed.

This feature is intended primarily for viewing Analyze error files while in Sil. When reading a line of text, Sil looks for possible x,y coordinates, as generated by Analyze and, if found, moves the Mark to that location. The text file is closed when the last line is read or after two successive ^K commands (see below). A ^shA will backup one line. This will go back only one line.

^**B**: Draws a box. The mark and the origin define the corners of the box. The box is selected, previously selected objects are deselected, and the origin is moved to the upper left corner of the box.

^sh**B**: Draws a rectangular background. The mark and the origin define the corners of the area. This command is useful in creating colored drawings. Text and lines drawn over a background will "work" correctly for color printers, but will disappear on a black printer (backgrounds may be disabled during press formatting with the use of "Sil/p **0/b** foo.sil" in the command line). Backgrounds over backgrounds will print such that the last background **created** with Sil will appear over previously created backgrounds.

^**C**: Copies all selected items and positions them away from the original items by the offset of the *mark* from the *origin*. The copy is selected, and previously selected items are deselected. The origin is moved to the mark, so that ^C can be done repeatedly.

^**D**: Deletes all selected items from the picture. Objects are not immediately lost, but are marked as deleted 'level 1' and no longer displayed. All objects already marked deleted have their level incremented by 1. When objects exceed level 5 they are discarded and their space reclaimed.

^**E**: Expands the rectangular area defined by the two preceding marks so that it fills the screen (as nearly as possible). The magnification factor (2-9) is shown in the status line. A second ^E exits this mode.

^**F**n: Sets the current font, face, or color according to the following table

n=0-9:	Font = n.
n=b/i	set font face to bold or italic respectively
n=B/I	set font face to not bold or not italic respectively
n=D,R,O,Y,L,G,T,C,A,V,U,M,P,S,N,W	set color to **D**arkBrown, **R**ed, **O**range, **Y**ellow, **L**ime, **G**reen, **T**urquoise, **C**yan, **A**qua, **V**iolet, **U**ltaviolet, **M**agenta, **P**ink, **S**moke, **N**eutral (black), or **W**hite respectively.

All new items are created according to these parameters as appropriate.
See Printing below, for more information about the use of colors.

^**G**n: Sets the current grid to 2**n (n=0-9).

^**H**c: Expands the macro c and puts its upper left corner at the mark. Macros within the macro c will not be expanded if the *onelevel* flag (the first "T" in the status line) is true. The ^N command complements this flag (i.e. makes it *false*, displaying "F").

^**I** Input: When ^I is done, the message "Input From: " is displayed, and the user is expected to enter a filename. If any previous input or output has been done, the previous filename is displayed. If it is the desired file, a carriage return, ESC, or mouse button depression confirms it. If not, simply type the new filename or *edit* the existing filename with ^Q, BS, or DEL (to abort input).

^**J**: "Jam" new text font, face, or color into all selected items. This allows the user to change all selected items to a new font, face, or color. Font and Face apply only to fonts 0, 1, 2, or 3.

^**K**: This command ("kill") clears the screen and reclaims all storage so you may start on a new drawing without leaving Sil. You will be asked to confirm with a carriage return if you made any changes since your last Output command. Filenames are remembered for use with subsequent ^I commands if desired and the ’Alternate text file’ may remain open. Any ’Alternate text file’ that may be open is closed if a ^K is executed when Sil is already in the initial state (i.e. 2 sucessive ^K commands).

^**L**c: Defines the macro c (in font 4) to be the collection of currently selected items (if there are no selections, or if c is a control character, the command is aborted). The reference point for the macro is its upper left corner, except that macro definitions are forced to fall on a grid of 4 screen units.

First, the definition is checked to ensure that none of the objects are the macro c itself. Such a definition would destroy Sil when an attempt was made to display the macro - the message "Bad Macro Definition" is output and the command is aborted. If all is well, the message "Confirm with CR" or "Confirm with CR to Overwrite" is output. If confirmation is given, the macro is defined, and the original set of objects is replaced by a (selected) single character string which is an instance of the macro. The origin is moved to the upper left corner of this string.

^**M** (CR): The mark is moved down by an amount determined by the ^Y command.

^**N**: Complement the *onelevel* flag, which controls the depth of macro expansion.

^**O** Output: A filename is requested as for ^I, and the macro definitions and picture are written on the file. Sil-format files normally have names with extension ".Sil".

^**P**: Writes a snapshot of the current state on the file "Sil.temp" (not including deleted items). It is a good idea to do this occasionally to avoid losing work if Sil or your Alto breaks.

^**Q**: This command exits Sil. It requires confirmation if the picture has been changed since it was last written out.

^**R**: This command requests confirmation and then deletes all macro definitions. It is used to edit macro libraries.

^**S**: This command ("Show") copies the area of the picture around the origin into the cursor. Since the cursor is only 16 screen units square (about .25 inch), only a small part of the object will be shown. If there are no selected items, or if the area around the origin is blank, the command is aborted. The idea of this command is that the origin will be positioned (automatically or manually) to an interesting area of a selected object, which may then be moved or copied with great accuracy using **control-mark** or **control-draw**. Doing anything other than a move or copy operation resets the cursor to the original arrow.

^**T**: Turns on (or off) an array of single point "ticks", which may be used as a positioning aid. The ticks are on a sixteen screen unit grid.

^**U**: Undeletes and selects the set of objects last deleted with ^D or shift-draw, after deselecting any previously selected objects. This works for up to five levels, at which point there are no more objects to Undelete.

^**Vn:** View has two distinct modes as follows:
For n= 4-9, the status line is replaced with a line of text enumerating the macros currently defined in that font. Typing any character returns to normal mode.
For n= 0-3, b,i,B,I, or color, all items with match the indicated parameter are selected (thus making it easy to jam a new parameter).

^sh**Vn:** only has meaning for n= 0-3, b,i,B,I, or color. In this case, items left selected are a subset of those selected before the command is issued. As an example, ^V0 ^shVR will select only those items which are font 0 and Red.

^**W**n: Sets the line width (n=1 to 9).

^**X** Move (translate): All selected objects are moved so that the origin is at the mark, then the positions of the origin and the mark are interchanged (so that another ^X puts things back the way they were).

As a feature, if the selected objects are thought of as defining a rectangular window, and if they are moved in X or Y only (not diagonally), the endpoints of any lines which cross the window boundary are moved (by shortening or lengthening the line) so that the endpoints have the same relation to the selected objects as in the original view. If the line would be shortened to zero or a negative length, or if it is fully in the window but not selected, it is not modified. This feature is handy for moving rows of components in a logic diagram or adjusting the boundaries of a form. This feature is turned off by ^shX or if a ^S is in force (i.e. if there is something in the cursor other than the normal arrow).

^**Y**: This command sets (or clears, if it is set) the *ylock* flag. At the time *ylock* is set (changed from "F" to "T" by the ^Y command), the difference between the Y coordinates of the mark and the origin is saved as *yinc*.

When subsequent carriage returns are typed, the mark is moved down by a distance determined by the height of the last object put into the picture (if *ylock* is false) or by the distance *yinc* (if *ylock* is true). *ylock* is initialized to false, and its status is shown in the status line (it is the second letter of the "TF0N" section).

^**Z**: This command complements "Hardcopy" mode, erases the screen and re-draws the entire picture. Sil normally spaces characters in text strings according to the widths of the Alto font. This makes them easy to read on the Alto screen, but means that the end of the string on your screen will not accurately indicate where the string will end on the printed page. To correct this, hardcopy mode positions each character according to the printing width information found in the Fonts.Widths file during initialization.

BS: This command (the BackSpace key) allows you to modify an existing text string. You must have exactly one item selected on the screen that is in fonts 0 through 9. This item will then be "opened up" the first time you hit the BS key. You then modify the contents of the item by appending characters, backspacing characters (the **BS** key with its normal meaning), backspacing words (**^W**), or by clearing the line (**^Q**) and starting again. Leaving this mode with the DEL key will return the original string. Besides the obvious advantages for editing the end of the string, you can use this command to replace an item with a new one having the same font, face, and color.

^**_** Moves the STATUS display to the x,y location of the last mouse action. This is useful in full drawings when items in the drawing must be placed over the normal location for the STATUS display. Note the the initial location can be specified in the User.cm file (see Fonts below).

Macros

Sil allows an arbitrary collection of objects to be defined as a *macro*, which is given a single character name, and thereafter behaves exactly as if it were a normal character. Control characters, space, and DEL are not allowed as macro names; there may thus be up to 94 macro names per font.

Font 4 is reserved for user-defined macros which are associated with a single drawing. The macro definitions in font 4 are saved with the drawing when an output file is generated with the ^O command.

Fonts five through nine are used for *Library Macros*, which are symbols which are used over a number of pictures. Libraries will be constructed for commonly used integrated circuit symbols, for example. Definitions for library macros are *not* saved with the file; instead, when a picture is read into Sil, any macros it requires from fonts 5-9 are read from the five files Sil.lb5 - Sil.lb9 (or whatever file name is given in user.cm). This also happens the first time a macro name in these fonts is used in ADD TEXT mode. If you intend to use these macro libraries, be sure the files are on your disk before you attempt a use. (Because users who desire only to make simple illustrations will not customarily use the libraries, their presence on the disk is not mandatory.)

A set of standard libraries is released with Sil. Sil.lb5 and Sil.lb6 contain component definitions for TTL-family circuits; Sil.lb7 and Sil.lb8 contain ECL-family definitions. These libraries change relatively slowly over time. However, when you save Sil drawings in a permanent way, it is prudent to save the libraries as well, to guard against incompatible future changes. You do not need to have all of these libraries on your disk if you do not make any reference to one or more of them.

Libraries are created in font 4. The only thing special about a library is its filename. When macro definitions are read from a library file, they are changed to the appropriate font. A library may contain a picture which describes its contents if desired. This portion of the file will be ignored when the library is used.

Two caveats about libraries: When one is created, it should not make use of any fonts other than 0-4 (i.e. it should not use any libraries other that itself). Also, if a library macro has other macros within it, they should have names which are greater (i.e. have greater ASCII codes) than the containing macro's name. If this is not done, things will still work, but library access will be much slower, since Sil will have to make multiple passes over the library file when reading it.

Fonts

When Sil/I is specified, the file User.cm is read to determine which fonts to use for fonts 0-3, where to find them on your disk, and where to find your library files on your disk. The file pointers for Alto fonts and Sil libraries are stored in "Sil.fps", and the font names, faces, and printing widths are stored on "Sil.fonts". A possible entry in a users User.cm would be:

```
[SIL]
0:  Helvetica10
1:  Helvetica7
2:  Template64
3:  Gates32
9:  Foo.lb5
Y:  712 (optional)  (X: val is also recognised but not very useful)
A:  TtlDict.Analyze (optional)
```

This is an example of a straightworward User.cm specifying fonts etc. suitable for logic drawings. A more completcated User.cm might be as follows:

```
[SIL]
0:  Helvetica10B Helvetica10N
1:  Helvetica7B
2:  Helvetica7BI Helvetica7B
3:  Template64
5:  Template64.lb5
Y:  712 (optional)  (X: val is also recognised but not very useful)
A:  TtlDict.Analyze (optional)
```

The above contains examples of many different ways of using your user.cm to control Sil fonts. The first name following the font number will be taken as the name of the printing font to use. If a bold or italic face is specified, then that face is the "default face", and can be overridden by the ^F and ^J commands described above. The second name (or first if a second is missing), with ".al" appended, is the name of the Alto font to use for display. If a face is specified for this font, then Sil will not further indicate this face. That is Sil will not italicize an italics .al font or bold-face a bold .al font. Numbers 5 through 9 may be used to specify a SIL file for use as a Library file. A "Y", or "X" entry is used to specify the position of the STATUS line on your display.

To expand on the above example, font 0 will print in Helvetica10 with Bold as the normal face, and will be displayed from Helvetica10N.al on the screen. Font 1 will print as Helvetica7 with bold as the normal, and will be displayed from Helvetica7B.al on the screen. Note that the font will not be displayed bold on your screen wheather specified bold or not. Font 2 will be the same as font 1 except that the default face will be bold and italics. This font will be made italics on the screen. Font 3 will be the special Template font used to draw arcs, circles, and diagonal lines. The other option for font 3 would be Gates32 (if you are doing logic drawings). Font 5 will be displayed according to the macros defined in the sil file Template64.lb5. The STATUS LINE will be displayed on the screen with its upper left hand corner at coordinates Y=712 (this is just above the title block of standard logic drawings). Finally, the entry A: is ignored by Sil, but is there for use by the Analyze program (see below).

If a font 0 through 3 is unspecified, it is defaulted to font 0, and Sil will complain bitterly if no font 0 is specified. If Sil has any problem reading the fonts, it calls Swat with an explanatory error message. If a Library file 5 through 9 is unspecified then "SIL.lb5...9" is substituted and looked up on your disk. No complaint is made if the Library name is not found.

Three caveats: (1) You should not use a large font for font 0, or the status line will overflow the right edge of the screen. Fonts up to about 10 points work well. (2) If you are working on exceptionally large drawings and you run out of SPACE for new items, you can Output your file and re-enter with "Sil/n(CR)" (n=0,1, or 2). In this case Sil will only read in the font definition for font n, and use that definition for displaying items in fonts 0,1 and 2. This will recover approximately 2000 words of storage.

Miscellaneous Information

Sil rebuilds the screen automatically whenever the picture is changed. This takes place incrementally, and if the picture is complex, it can take a while. The rebuilder is operating if the number in the "TF**0**N" part of the status line is non-zero. If you are confused about the state of the picture, wait for the rebuilder to stop.

Sil correctly windows objects so that things which overlap the screen boundaries are only partially displayed. It is thus possible to type a string which extends beyond the screen boundary, then move it back onto the screen. If, however, an object is moved completely off the screen and then becomes deselected, it will be destroyed, and the space it occupied reclaimed. If an item is moved such that any part of it is above or to the left of the screen boundary, that item is immediately lost and connot be reclaimed with Undelete.

Printing

Sil/P (alias nPPR.run) is an alternate way of invoking Sil which will cause it to enter a completely different mode of operation. In this mode, Sil converts Sil format files into one Press format file called "Sil.Press". Sil/H may be used to start up Sil, in which case the same "Sil.press" file will be generated, but in addition, Empress.run will be invoked to send the press file to the appropriate printer for making hardcopy. Sil will use the font related information saved in "Sil.fonts" (during Sil/I) for printing.

```
Sil/H File.press/F {color switches} File01 File02... Printer/H n/C
```

In the above example, Press formated output will be written in "File.press" which will then be sent to Host "Printer", and it will ask for n copies to be printed. If the /H or /C commands are omitted Empress will look in your User.cm under [HardCopy] for your default Host printer, and will print 1 copy. If the /F switch is not omitted the output will be written on "Sil.Press". Note that the /F switch, if used, must come before any input files.

A whole slew of additional commands have been implemented for press file creation to control the color in drawings to be sent to a color printer. In essence, when one specifies a color in the Sil drawing, you are only setting a "pointer" to a table for the printing phase of Sil. This table is initialized to values which will print the colors which correspond to the name given that color.

Each color has associated with it a Hue, Brightness, and Saturation. There is one table for text/lines and another for Backgrounds. The default values are the same for both tables according to the following table.

Color	Hue	Brightness	Saturation	Comments
Neutral(black)	000	000	000	3 primary colors
White	000	255	000	
Smoke	000	192	000	requires half-tone process
DarkBrown	005	090	255	requires half-tone process
Red	000	255	255	2 Primary colors
Orange	020	255	255	requires half-tone process
Yellow	040	255	255	Primary color
Lime	060	255	255	requires half-tone process
Green	080	255	255	2 Primary colors
Turquise	100	255	255	requires half-tone process
Cyan	120	255	255	Primary color
Aqua	140	255	255	requires half-tone process
Violet	160	255	255	2 Primary colors
Ultaviolet	180	255	255	requires half-tone process
Magenta	200	255	255	Primary color
Pink	220	255	128	requires half-tone process

Notice that some colors are primary colors, some are combinations of primary colors, and others require half-tone dot screens to be applied by the press printer. You should keep in mind that the half-tone process will introduce some degradation of character and line edge sharpness.

All of the above parameters may be changed with the following switches. Whenever an entry of this type is encountered, it will affect the printing of subsequent files on the command line:

C/o	following switches will effect text and lines with color C.
C/b	following switches will effect backgrounds with color C.
N/b	set the Brightness of the appropriate entry to N.

If no color has been specified, then the brightness of ALL backgrounds will be set to this value. (white will get 255-N)

N/s	set the Saturation of the appropriate entry to N. If no color has been specified, then the saturation of ALL backgrounds will be set to this value. (neutral & white will get 255-N)
N/h	If N is a number then set the Hue of the appropriate entry to N. If N is a name then send Sil.press to printing host N.
/i	will re-initialize all entries
0/b	Disable backgrounds so that the press file may be printed on a black printer. Actually, 0/*, where * is any character, seems to do this.

In order to properly print colored strings and lines over colored backgrounds, it is necessary to specify software scan conversion in the press file. This takes a great deal of time for the host printer to perform , so if you determine that this is not necessary in your case you may invoke Sil with Sil/PF, where the F (fast) switch inhibits software scan conversion.

Finally there are three other switches which effect Sil hardcopy. Note: If N=0 these switches should be omitted; otherwise they would disable backgrounds (see 0/b above).

N/x	Shifts the entire drawing N Alto screen units right (left if negative).
N/y	Shifts the entire drawing N Alto screen units down (up if negative). [Known bug, found by Dick Lyon: N acutally has some units other than screen units, about 11/16 inch].
N.NN/z	Scale the entire drawing by the indicated amount. The number may be less than or greater than one. The point size of the printing fonts requested will be scaled, and it is left up to the printing server to select the best available printing font to match the point size requested. This works well for Helvetica, but not at all for Gates or Template fonts. If both scaling and shifts are specified, the shifts are performed before the scaling.

A word of caution. The last set of switches were added quickly and are not done carefuly. In particular, objects are not clipped properly at the page boundary, and the press printer will "wrap-around" if you move or expand something very far off the page.

ANALYZE

Analyze is a program that transforms logic diagrams produced using Sil into a file which can be input directly to the Gobble wirelister. In addition, Analyze produces a file in Sil format which contains the IC pin numbers which were not assigned by the user in the original drawing. Analyze assigns such pins automatically, and the resulting file may be merged with the original drawing to obtain a complete drawing, including all IC pin numbers.

Analyze is invoked with:

```
Analyze(/D) file01.sil file02.sil file03.sil ...
```

where file01.sil file02.sil file03.sil ... are the names of Sil output files. The program produces two output files for each input file: If ’filename.anything’ is the name of the input file, ’filename.NL’ is a text file that contains a listing of all components and nets in the drawing, and ’filename.PN’ is a Sil format file which contains the IC pins which were assigned by Analyze (if no new pin numbers are assigned, the file is deleted before Analyze finishes). Any errors detected during operation are recorded on ’filename1.er’, a text listing of error messages; it will be deleted if no substantive errors were detected. If the /D switch is used, the program writes (a great deal of) internal information onto the error file. During operation, Analyze turns off the display, and does not signal the occurrence of errors in any way. After completion, Analyze leaves a brief message "[n] = worst Error Severity" on the screen. N=0 means no problems were encountered, n=1 thens possible problems (warnings) were encountered, and if n=2 then errors were found and the ".pn" and ".nl" files are not valid. Warning and error messages are left in the ".er" file. Analyze requires one or more dictionary files, which are text files containing the correspondence between IC pin numbers and IC pin names for all components in the drawing. These file will be maintained concurrently with the Sil macro libraries.

Input File Format

The drawings produced by Sil contain four types of objects: The first are macros in fonts 5-9 (see below for exceptions) which are the graphic description of *components*; the second are user defined macros in font 4 (see below for exceptions) which consist of collections of components, lines, and strings. The remainder of the file contains lines and strings (including single character font 4-9 strings corresponding to instances of user macros and components) which constitute the body of the drawing. These objects must conform to a number of syntactic rules, which will be described in detail.

components

Components are macro definitions which describe the component function to the user, and the component pin assignments to Analyze. A reference on the drawing is signified to be a component if it is a macro in fonts 5-9 ***or*** font 4, character codes 0 through 9. It is intended that all common macros be available in the public library files as font 5-9.

As a mechanism for accomodating unusual or seldom used macros, Analyze will also treat font 4 character codes 0 through 9 as component macros. This will alow you to maintain additional Sil files containing component macros. When needed, one essentially copies some macro difinition from one of these library files, re-defines it under a new character code, and merges in into ones drawing. The proceedure is as follows: Read the Sil macro file in, delete the entire picture, expand the one macro (with ^H) of interest, type ^R to delete all macro definitions, select the expanded items of the new component, and define them (with ^L) to a macro character 0 through 9. Finally read in the file you are working on and use the newly defined font 4 macro.

Dictionary File Format

Analyze looks at one or more "dictionary" files to find the correspondence betwen component names the the pin assignments. Analyze first looks in user.cm to find an entry "A: someDict.analyze" for the first dictionary file to open. If not found, Analyze will look for "Dict.analyze". Each dictionary may "get" another dictionary (as EclDict.analyze gets TtlDict.analyze), and Analyze will look down as many dictionaries as necessary to complete a drawing. There are currently two standard dictionary files maintained, TtlDict.analyze and EclDict.analyze. You may get either or both as required. In general, if you have some component not already in one of these dictionaries, you should arrange to get it entered. If you wish, however, you make a dictionary of your own (examine the standard dictionaries for the format) , place its name in user.cm and call the standard dictionaries from within your own. You may have one dictionary which calls another which is not present on your disk. If all components are defined in the existing dictionary(s) then no error or warning is generated.

Lines

Analyze recognizes lines of width 1 as signal paths. Lines of other widths are ignored.

Names

Analyze recognizes three types of names: signal names, component type names, and component pinnames. All names must be in font 1 and not italic; strings in fonts 0, 2, and 1-italic are ignored (exception: see title syntax, below). Names must be completely unique (i.e. it is illegal to have a signal named "IN", since this is a component pin name).

Component type names are recognized as such because of their inclusion in the component dictionary files (TtlDict.Analyze & EclDict.Analyze). A component type name may appear either in the macro definition corresponding to the component (i.e. in a library) or in the body of the drawing. The former situation will occur when the definition macro corresponds to a single component type, the latter is used if the macro is used to represent more than one type of component.

Component pin names may occur only in component definition macros (i.e. only in macro libraries and in Dict.Analyze). These names are associated with *connection points* in the definition.

Signal names may occur in the body of the drawing, or in macros which are for the purpose of replicating a common structure (i.e. font 4 macros). Signal names are always associated with horizontal lines, and must be positioned in the drawing such that the center of the string is above the line and closer to it than 30 screen units.

Component Definitions

Component definitions are Sil macros in fonts 5-9 which contain connection point characters at the top level of their definition. These definitions are taken from the libraries Sil.lb5-Sil.lb9 as required. Connection points are specified with the font 3 characters Q,R,S, and T. They indicate the locations at which Analyze expects to find interconnecting lines (signal paths). The four connection point characters correspond to connections made at the left (R), right (T), top (Q), and bottom (S) of the component symbol. In the font file ’GATES32.AL’, the connection point characters are small T’s in four orientations, and are short lines in the printing font. Associated with each connection point is an IC pin name string in font1. If Analyze cannot locate an IC pin name string near a connection point in the definition, it defaults the name to ’IN’ for left side connection points, to ’OUT’ for right side points, or indicates an error for top and bottom points.

Note that component definition macros may *not* include IC pin numbers. This information is supplied by Dict.Analyze, which contains the correspondence between pin names and pin numbers. If it is desired to preassign IC pins to control the placement of interconnects on a card, the pin numbers must be put in the body of the drawing, rather than in the component definition. Any pin numbers which are included must be numbers in the range 1-63, and be in font 1 (or font 3).

Components correspond either to a subsection of an IC package (e.g. a 2 input gate which occupies one-fourth of a package), or to an entire package (e.g. a shift register). A single macro definition may be used for many IC types which are drawn identically, or a single IC type may be drawn with more than one macro. An example of the latter situation is a NAND gate, which may be shown as an AND gate with inversion at the output, or as an OR gate with inversion at the input.

The correspondence between component types, component connection points, and their associated pin numbers is made by the component dictionary files. Each component in the drawing must have associated with it a board location and group number. This descriptor is a font 1 string of the form <letter number letter >, with number <64, and a ≤letter ≤y. The first letter and number represent the coordinate of the component on the board, the second letter represents the component's group within the IC package. The group letter may be omitted if the package contains only a single group, in which case Analyze will default the group to 'a'.

Analyze locates the name string for each connection point character in a macro definition on the basis of its proximity to the connection point, then uses the type name, group letter, and connection point name to look up a list of valid pin numbers in the dictionary file(s). If pin numbers were predefined in the input file, their validity is checked. If some or all pin numbers are not assigned in the input file, they are assigned automatically.

To summarize, the precise graphic content of a component definition macro is unimportant. The important information contained in a definition macro consists of:

1) The relative coordinates of connection point characters
2) The pin name strings associated with each connection point
3) Optionally, an IC type name string

User defined macros

Macros in font 4 may be used to replicate common structures in a drawing. These macros are expanded by Analyze when instances are encountered in the body of a drawing. Such macros may contain lines, character strings, components, and other macros, nested to any depth.

Drawing body

The main body of the input file contains the information which the user explicitly added to the drawing using Sil. Legal objects for Analyze are a subset of the possible objects which may be drawn with Sil. The interpretation of all objects which may be created is as follows:

1) All items: All items (text, lines, components etc) that are assigned the color Majenta are ignored. *Note: Analyze will remind you (with a count) of Majenta colored items if it finds level 2 errors on a drawing.*

2) Lines: If they have width = 1, they are interpreted as interconnects. If not, they are ignored.

3) Font 0 strings, Font 2 strings, and Font 1 italics face: These are ignored. It is thus possible to include comments in drawings which will be ignored by Analyze.

4) Font 1 strings (normal and bold face): If they can be interpreted as <letter number letter>, they are assumed to be a board location/group number.

If the string can be interpreted as a number, it is assumed to be an IC or edge pin number.

If the string corresponds to a component type defined in a dictionary file, it is assumed to be a component type name. If these interpretations fail, the string is assumed to be a signal name.

Recall that all names must be unique. In general, it is best to begin all signal and component type names with a capital letter. A component type name must be associated with each component in a drawing, although if there is more than one component group at a particular board location, only one of the groups must have a type name string - the others are inferred.

5) Font 3 strings: If the string consists of the single character P or p, it is interpreted as an edge or cable pin, respectively. If it is the single character ’n’, it is interpreted as a connection to ground, and any signal path associated with it will be given the name GND. If it is the single character ’x’, it will be assumed to be a connection to a pseudo-net; the pseudo-net has the signal name ’+’. If the string can be interpreted as a number, it is assumed to be an edge pin, cable pin, or IC pin number. If it is the character ’.’, it is a blob, which is placed on a connection point to inform Analyze not to expect to find a line connected to the point. If all of these interpretations fail, an error message results.

Pin number strings are associated with edge pins, cable pins, or component connection points on the basis of proximity. For cable and edge pins, the string will be associated with the pin if it is placed within the rectangular edge/cable pin character. For IC pin numbers, the string must be positioned relative to the component connection point as follows:

> Left side connection points: above and to the left
> Right side: above and to the right
> Top: above and to the left
> Bottom: below and to the left

When Analyze automatically assigns omitted pin numbers, it outputs them in Sil format to the pin number file with this orientation.

6) Font 4 strings: These strings must be exactly one character long, or an error message results. The macro corresponding to the font 4 character is expanded.

Search Rules

Analyze uses a number of rules to determine if two objects are associated. In general, if a drawing is prepared using Sil’s grid 4 (^G2), all errors detected by Analyze will be legitimate. Using the default font set and libraries, it should not be necessary to use a grid size other than 4, except to add single font 3 special characters. This is best done using Sil’s ^S feature, and should be quite precise.

The search rules are:

> 1) Two line endpoints which form an "L" must be within 2 screen units to be considered connected.
>
> 2) Two lines which form a "T" must be within 3 units to be considered connected. Lines which cross each other are NOT considered connected.
>
> 3) Edge and cable pins must touch the horizontal line to which they are connected.
>
> 4) Edge and cable pin numbers must be inside the box which constitutes the pin.
>
> 5) When strings are matched with anything, the coordinates used are those of the center of the string.
>
> 6) Signal name strings must be less than 30 screen units above a horizontal line. Signals are not associated with vertical lines.

7) Type name strings and board coordinate strings must be within a bounding box which is the height of the associated component, starts at its left edge, and extends to the right until another component is reached.

8) Lines and blobs associated with connection points on components must be within 2 screen units of the connection point.

In practice, these rules mean that things which touch are associated, things which do not are not. Although Analyze can detect a number of errors made by the user, sloppiness can cause problems which Analyze cannot detect. NEATNESS COUNTS!

Parsing the title block

Several facilities in Analyze encourage you to place near the bottom of each drawing a "title block" that describes the drawing. A prototype for an acceptable title block can be found in the file LogicBlock.Sil. Analyze will look in this area (y>720) for strings in fonts 0, 2, or 1-italic that describes the drawing.

In particular, Analyze finds strings below and to the right of four keywords: File, Rev, Date, and Page, and associates the strings with the corresponding properties. It passes this information along in the .NL file as an aid in identifying the origin of the file. Gobble will copy this information into the final wirelist, thus providing you with a reasonably good description of the drawings that were used to make the wirelist. If a string "Reference" or "Documentation" is found anywhere within the title block area, then Analyze will generate a nul net list for compatibility and ignore ALL text and lines on that drawing. This feature allows reference drawings to be included in a consistently names set of drawings that Build will maintain for you. The title parsing is also intended to work with the Build subsystem.

The facility is useful only if you keep the title information accurate or use Build to keep it accurate for you. Build will process the Sil drawings, assigning consecutive page numbers, setting the file name to be the name of the file, setting the date (if the drawing is not marked build), and setting the rev level of all files.

If a title block can be successfully parsed for a file name, Analyze will use the file name to generate unique names for un-named nets in the drawing. It does so by appending the character ’+’ and a number to the file name. Thus if the file name mentioned in the title block is "Or01.Sil", a net might have the name "Or01.Sil+4". This convention will allow you to relate entries in a final wirelist back to appropriate drawings. If there is no title block, or if a file name cannot be found, Analyze will emit no names for un-named nets, and Gobble will generate names of the form "XXX", followed by a unique number.

Prescan

If Analyze is invoked by typing Analyze/P file01.sil file02.sil file03.sil ..., it will produce a file file01.ps that contains the page numbers (the i-th file processed is page i or the page number in the title block) on which signal names and IC type names appear. The file file01.pe contains any error indications. This feature is used to provide a quick check on signal names and component count, without having to assign the IC’s to board positions and run Gobble for the entire board. No node list or pin number files are produced, and almost all error checking features are turned off (so the process is quite fast).

Error Messages

The following list summarizes all the error messages which may produced by Analyze, and a statement of the rule which was broken to cause the error if it is nonobvious. In some cases, a single error will generate more than one message, since a number of cross-checks on a drawing’s validity are done.

In the messages, "*" indicates an x,y screen coordinate which may be used with Sil to find the source of error in the drawing.

***Font 4 string with length #1**
Strings which are instances of macros must be one character long.

***Font 4 macro has no definition**
This should never happen, since it is a fatal error in Sil, but its consequences are sufficiently horrible that it is checked.

***Malformed font 3 string**
The string cannot be interpreted as a single special character (p, P, n, or x), nor can it be parsed as a number.

***Multiple definition for symbol**
All names must be unique, and one isn't.

***Can't find line for pin**
***Can't find number for pin**
***Can't find line for ground**

***Ground appears connected to top of line?**
The ground symbol must attach to the bottom of a vertical line.

***Can't find line for pseudonet point**
***Can't find line for signal name**

***Component more than 1 character long**

***Component has no definition**
The macro library for this component doesn't include it. This is a fatal error for Sil, and should never happen.

***Can't assign all t/b conpoints to line**
***Can't assign all l/r conpoints to lines**
One of the connection points on the component pointed to by the * has one or more unused connection points. This message may be suppressed using 'blobs', as described earlier.

***Multiple type names possible for component**
***Multiple board locations possible for component**
The bounding box for type name searches contains more than one type name or board location string.

***Can't find bloc/group for component**
The bounding box contains no board location.

***Coalesced overlapping vertical lines**
***Coalesced overlapping horizontal lines**
This is a warning. Analyze has found two lines which lie on top of one another in the drawing, and has merged them into one line.

***Line with no associated component or edge pin**
All lines must be connected to either a component or an edge pin

***Line with no name has only one pin**

***Unused Epin**
***Unused Cpin**
***Unused ground**
***Unused pseudonet point**
***Unused blob**
These messages are part of a final check done to ensure that Analyze was able to make use of all the objects in the drawing.

***Can’t find or default type name for component**
The component at the specified screen position has no type name, nor does any other component at the same board location.

***Net has multiple names**
A single line has more than one name string associated with it. This is an error even if the names are the same.

***Pin name is not in dictionary**
***Preassigned pin is not a valid choice**
***Can’t find pin to assign**
For these messages, the coordinates point to the ic pin with which Analyze is having trouble.

***Unused pin in section near this point, named ...**
The dictionary definition of the IC section you are using contains pin definitions that you have not wired up. This is simply a warning.

***Group shown for this component is not in dictionary**
The coordinates point at the component.

***Multiple type names for component at this board location**
The component pointed to and another component at the same board location were assigned two different ic typenames. This may be an error in the name, or an error in the board location.

The following messages may be generated when parsing the title block in the drawing. They are only warnings:

Unable to find title entry for File
Unable to find title entry for Rev
Unable to find title entry for Date
Unable to find title entry for Page
File name cited in Sil title region does not match true filename

The following messages are generated as the libraries are read. They should not occur unless the standard libraries have been modified. The "%" indicates the font and character in which the error occurred:

Component real name conflicts with other symbol type: %
Strange entry in dictionary header: %
Name in compdef has wrong symbol type: %
Compdef contains overlapping connection points: %
Compdef has more than one type name: %
Compdef has more pin names than connection points: %
Compdef has no connection points: %
Can’t find name for conpoint in compdef: %
Can’t assign string in compdef: %

The following messages are generated as the dictionary files are read. Only component definitions for types used in the drawing are examined. The $ indicates the type name in the messsage.

Analyze does not tell you which dictionary, but the type shuld be sufficient information for you to figure that out.

Dictionary has groupname #a-y in type $
Dictionary has strange entry in type $
Dictionary has strange pinname in type $
Dictionary has bad pinnumber in type $
Dictionary has nonnumeric pin in type $
Dictionary has pinnumber >63 in type $

GOBBLE

Gobble is a program that merges a number of node list files (.NL) created by Analyze to produce a wirelist for a single board. It also does automatic terminator assignment for ECL signals, and routes each net to achieve a minimum wire length. Gobble does not do automatic component placement: this information must be supplied by the designer in the original Sil drawing.

Gobble is invoked by typing:

```
Gobble/x file01.nl file02.nl file03.nl ...
```

to the Executive. The switch is a single letter (A to Z) that tells Gobble the *board type* to use for the wirelist. Brief descriptions of boards are given below.

Gobble creates several output files, overwriting any previous contents. If file01.nl is the *first* filename in the list of .NL files given to Gobble, the wirelist file will be written on file01.WL, the backpanel pin list will be written on file01.BP, and errors will be written on file01.GE.

While it is running, Gobble complements the cursor each time it begins processing a new net. This will start a few tens of seconds after the program is started (after all input files are read), and will continue until all processing is done. A few more seconds will elapse as the output files are written, and Gobble will finish. The amount of time spent processing a net depends on its length, and may be up to a few tens of seconds for nets that include many nodes.

The routing algorithms used by Gobble can be controlled by optional entries in the command line appearing just before the first filename:

number/H	Specifies amount of work the heuristic router will do (default=20; 100 is "a lot")
number/E	Specifies the size of the net above which the heuristic router is used and below which the exhaustive router is used (default=7)
name/M	Specifies the metric to use in computing net length (name="Manhattan" or "Euclidean"; default is Manhattan)

ECL terminator assignment is normally performed on any net that has one or more outputs from ECL-family components, and does not visit any edge or cable pins. In addition, any net with a name that ends in the character '!' will not have terminators assigned automatically. These conventions will require you to draw explicit terminators for any net that either (1) visits an edge pin; (2) visits a cable pin; or (3) has a name that ends in !.

Gobble recognizes certain reserved net names, usually used for ground and various sorts of power. The reserved names are listed with the board type, below.

Reworking a board

When a board has already been constructed, and a rather small set of changes is required, Gobble is willing to generate a file of "adds and deletes" that will alter the already-constructed board to agree with the new drawings. For this purpose, invoke Gobble with \ rather than / preceding the board letter. Gobble will read all the .NL files *and the .WL file that corresponds precisely to the present board*, and will create two new files: file01.AD, a list of adds and deletes to make to the current board, and file01.wlNew, a revised complete wirelist. *This wirelist will represent "truth" only after the add/delete modifications have been accomplished on the board.*

The Gobble re-work option requires rather careful attention to data-management to avoid careless errors while modifying a board several times. Although the Build subsystem will help with this chore, it is wise to understand the underlying principles. The most important point is that *it is essential to keep a .WL file that accurately represents the current state of the board,* for it is from this file that Gobble determines what must be changed when updates are required. This .WL file cannot, in general, be reconstructed from the Sil drawings that agree with it, because automatic terminator assignment may place terminators differently.

GOBBLE board definitions

Information about geometric and electrical properties of various boards is "assembled into" Gobble, and can currently only be modified by modifying the program. The following list cites boards known to Gobble, and gives a brief description of their properties. All ranges are *inclusive.*

B. Augat 8136 stitch-weld board.

C. Board specially designed to fit in Xerox 9200 engine control.

L. D1 main logic board. Board location letters range from a to l (ell); numbers range from 1 to 24 for 16-pin dips, 41 to 52 for SIPS, and 60 to 63 for 24-pin packages. Cable and edge pin numbers range from 1 to 188. There are five reserved net names, corresponding to various power supply voltages: GND (0), VCC (+5), VTT (-2), VEE (-5.2), and VDD (+12).

M. D1 memory storage board.

X. Wild card board -- allows all board positions (a-y, 01-63), and does no routing.

Care

The judicious user of Gobble will always be skeptical, and will examine carefully the output before stitch-welding. Special inspection should be applied to add/delete lists.

Error Messages

...to come...

BUILD

Build is a subsystem that helps with the data-management aspects of building boards and keeping the design-automation data files current. Suppose a board is currently "revision C" (or Rev C), two drawings are updated, and it is now time to undertake to get everything (drawings, wire lists, and board) updated to the new Rev D. Good practice will encourage us to change *all* the drawings to show Rev D as the current version; but sheer tedium and errors will soon let chaos creep in. Build should help.

Let us first define a "built drawing" for a certain revision level as the drawing that accurately represents the particular rev level. Built drawings are distinguished by a series of broad horizontal lines at the very bottom of the page: these lines mean that the drawing, as you see it, was used to build the board whose revision level is cited in the drawing. *Sil will remove these markers whenever you change the drawing in any way.* Thus the veracity of the "Rev x" label in the drawing is determined by the obvious markers.

Build is invoked with a command line that lists *all* the files for a particular board. For example, "Build D/R \GL aabb*.Sil" will build the D revision of the board aabb. Build undertakes several steps:

1. All .Sil files are "edited" by Build, and changes are made in the title area of each drawing. The revision level is updated to xx if the phrase xx/R is present in the command line. The file name cited in the title area is changed to match the true file name. The page number is updated to match the ordinal position of the filename in the list of files provided to Build. And finally, unless the .Sil file was already built (i.e., unmodified since last Build), the date is updated. The file is marked "built," and the special marker lines are added to it. The first file specified always has the date updated. The specification on page numbers assigned may be modified by placing a num/p switch before some page, where num may be 0,1,2 etc or .+2, or .+0 etc..

2. Now Analyze is run on all the files that were not marked "built" at the start of step 1 above. The error messages will, as usual, be collected on one file aabb01.er, where aabb01.Sil was the first name passed to BUILD.

3. Any .PN files produced by Analyze are merged into the corresponding .Sil files unless the Tentative switch has been set. Build will terminate before appending pin numbers if Analyze reported any serious errors.

4. Gobble or Route is invoked on all the .NL files produced by Analyze. The "board letter" switch for Gobble is extracted from the /Gy switch in the command line (or \Gy for rework). Any other items in the command line that are of the form "xxx/Gyy" are passed on to Gobble as "xxx/yy". If you are using BUIILD for re-work, you must of course have aabb01.wl on your disk (this is the current wirelist file, for rev C in our example). In rework mode, you may explicidly give the name of the .wl file to be reworked by placing oldname.wl/CG in the command file.

5. If re-work has been specified and an old .wl file has not been given explicidly, aabb01.wl is copied into aabb01.wlOld (backup); then aabb01.wlNew is copied into aabb01.wl (critical step!!), and finally aabb01.wlNew is deleted. The critical step will be crash-protected with a restart procedure that copies aabb01.wlOld back into aabb01.wl so that no harm is done.

6. This step executes any number of commands according to the contents of a special file called BuildBackupTemplate.cm. If no such file exists, then Build will default to dumping all files it believes are critical to the build run on both IVY and MAXC. Build essentially reads in BuildbackupTemplate.cm expands some special escape sequence as defined below, and passes the results on to the Alto operating system in Rem.cm for further system expansion.

If a "$Z" is found in the template, then the template is expanded as follows:

N:	name of the first file given to build with the extension stripped off. $ZN.ad >> Foo01.ad
B:	name of the first file given to build with the extension and numbers stripped off. $ZBC.ad >> Foo.ad (only if running "change" mode - see below)
A:	names of the files given to build. $ZB.sil >> Foo01.sil Foo02.sil . . .
F:	same as $ZB plus the string "-Rev-" and revision appended. $ZF.dm >> Foo-Rev-Aa.dm

In addition, there are a number of characters that will modify whether an expansion is to take place attall.

C:	Only if running in change mode
G:	Only if using the Gobble program
R:	Only if using the Route program
M:	Only if using the Multiwire option of Route

Build can be invoked "tentatively" by saying Build/T ... In this case, only steps 2 and 4 are executed. This allows you to look at the .er, .ge and .ad files before committing to the revisions. If you are re-working, the new wirelist will of course be on aabb01.wlNew, i.e., no copying will have been done.

Build is designed to permit restarting at any time without damage to vital files. Thus, if the Alto disk fills up during one of the operations, you can delete some old files and retry the Build command. You specify the step at which Build is started by add the number to the global switches to build, thus "Build/6 . . ." will start Build up at level 6.

Build uses a rather general mechanism for passing phrases from its command line on to the subsystems it invokes (Analyze, Gobble, Sil/p, and Ftp). Phrases of the form xxx/qyy will be passed on as xxx/yy to the subsystem whose initial letter is q. Thus conn/fc is passed to Ftp as conn/c; maxc/f is passed to Ftp as maxc; 4/ge is passed to Gobble as 4/e. If the xxx phrase is empty, the switch phrase is passed as a global switch. Thus \gy is passed to Gobble as the global switch \y. So a *real* command to Build that might update a real board to rev F is:

```
Build \gl 6/ge f/r maxc/f conn/fc d1/f password/f d1pa*.sil
```

CONVENTIONS FOR USING THE DESIGN AUTOMATION SYSTEM

This section recommends several conventions for using the design-automation system. Although not every design group will want to follow them all, they have aided a number of large projects. Note that this information is supplemental to a great deal of detailed conventions mentioned elsewhere in this document (e.g., the rules that must be followed to please Analyze).

Drawing files have the extension .Sil, and are given reasonably short names of the form xxyydd.Sil, where xx is a code for the project, yy is a code for a specific logic board, and dd is a two-digit number that indexes the drawings for the board. The two-digit number three is 03 -- this convention assures that the Alto Executive * feature in filename recognition will always process drawings in numerical order. Thus "Sil/P D1CB*.Sil" will print the drawings consecutively.

In Sil, do all drawings on a grid of 4 units (^G2). Macros should be defined so that all connection points lie on grid points. Use a "title area" at the bottom of each drawing -- the file LogicBlock.Sil provides a template.

Component names should be all upper case. For example, MC101, N123 or H04 are good component names. Note that H04 is a component name and h04 is a board location.

Signal names should be mnemonic, pronounceable and meaningful. Names should begin with a capital letter, use upper case to separate words, and should contain no spaces, {, }, <, >, ;, or * characters (some of these confuse the stitch-welding program). Examples of beautiful signal names are:

```
MemoryDataReady   BufferEnable   CompareError
```

In order to name individual lines of buses and registers, follow the signal name with .dd, where . is the period character, and dd is a one or two digit number. Include the leading zero for buses wider than 10 bits to make name sorting convenient. Thus:

```
Video.0         BMux.00
Video.1         BMux.01
Video.2         ...
Video.3         BMux.15
```

The active low version of a signal is denoted by appending the character ’ (single quote) to the name. Ready inverted is Ready’.

Signals that are driven differentially (often cable and edge signals) have + as the final character of the differential positive signal and - as the final character of the differential negative signal. For example, LineSync+ and LineSync- would be received differentially and produce LineSync as output.

Perhaps the largest single problem in managing a large design project is data management: keeping all the drawings, wire lists, macros, etc. together, especially as changes are made. The Build subsystem, described above, is an attempt to impose some order on the chaos. Even if you object to the specific actions Build takes, you should enforce some conventions for preventing disaster. The nexus of the problem concerns Gobble re-work: it is essential to have a wirelist that accurately reflects the current state of your board. As revisions fly thick and fast, it is surprisingly easy to get confused.

FILE FORMATS

SIL drawing format

The format of Sil files is considered "private" to the suite of design-automation programs.

Text file conventions

Most of the files used in the design-automation system are text files, for convenience in fixing problems, editing, seeing what you are doing, and the like. Several conventions apply to all such files:

Comments are embedded in files by putting semi-colon (;) as the first character in a comment line; the remainder of the line is ignored.

Dictionary format

The file Dict.Analyze is a text file which contains definitions for all components which may be used in a drawing. The file contains two sections, a header, which contains the names for all components in the balance of the file, and a body, which contains the association between groups, pin names, and pin numbers for each component. A (small) dictionary might have the form:

```
MC100=MC10100/16/E
MC136=MC10136/16/E
@
MC100
a,IN,4,5
a,c,9
a,OUT,2
b,IN,6,7
b,OUT,3
c,IN,10,11
c,OUT,14
d,IN,12,13
d,OUT,15
@
MC136
a,CC,13
a,D0,12,
a,D1,11
a,D2,6
a,D3,5
a,S1,9
a,S2,7
a,CI,10
a,C0,4
a,Q0,14
a,Q1,15
a,Q2,2
a,Q3,3
@
```

The first two lines are the header. The first string is the *print name* of a component (MC100), which is the name Analyze expects to find in the drawing. Usually, these names are short versions of the *real name*, which is the second item. The real name is the manufacturer's name for the component, and might be used for automatic preparation of purchase orders, for example. The "/16/E" following the real name is the number of pins on the IC and the *family type*. This information about the IC is required by the Gobble wirelister, so that it can do terminator assignment (ECL), power assignment, and routing properly. The family types currently supported by Gobble are (the term "PackPin" is the number of pins in the package, either 14 or 16):

E: MECL 10K (Gnd on pins 1,16; Vee on pin 8)
N,S,H: Normal, Schottky, and H TTL (Vcc on PackPin; Gnd on PackPin/2)
T: 8-pin SIP terminators (Vee on pin 1)
P: 16-pin pullup resistor newtorks (Vcc on pin 16)
M: MOS (Gnd on pin 16; Vcc on pin 9; Vdd on pin 8; Vee on pin 1)

The header is terminated with @.

The balance of the file consists of blocks describing each component. Each block begins with the print name of the component, and is terminated with an @. The first field in each line is the group id for the information on the balance of the line, the second entry is the pinname, and the third (and subsequent) entries are a list of the ic pins which can have the given group id and pin name. There may be multiple pins with the same group id and pin name if the pins are logically identical (i.e. if Analyze is allowed to permute them during the pin assignment process). The MC100, for example, contains four groups, one for each of the four gates in the package. The groups are lettered a-d. The IN signals in each group may be permuted. The MC136 has only one group, group a, and its pins may be assigned in only one way.

If a chip in a particular family does not obey exactly the power/ground rules for the family (stated above), it is possible to include information in the component blocks that describes the power requirements. This is accomplished with a line following the print name that begins with the word POWER, and cites pin numbers and power names. For example, the following line would describe an MC100, although it is identical to the normal rules:

```
POWER GND,1,VEE,8,GND,16
```

And the following line would describe an MC210:

```
POWER GND,1,VEE,8,GND,15,GND,16
```

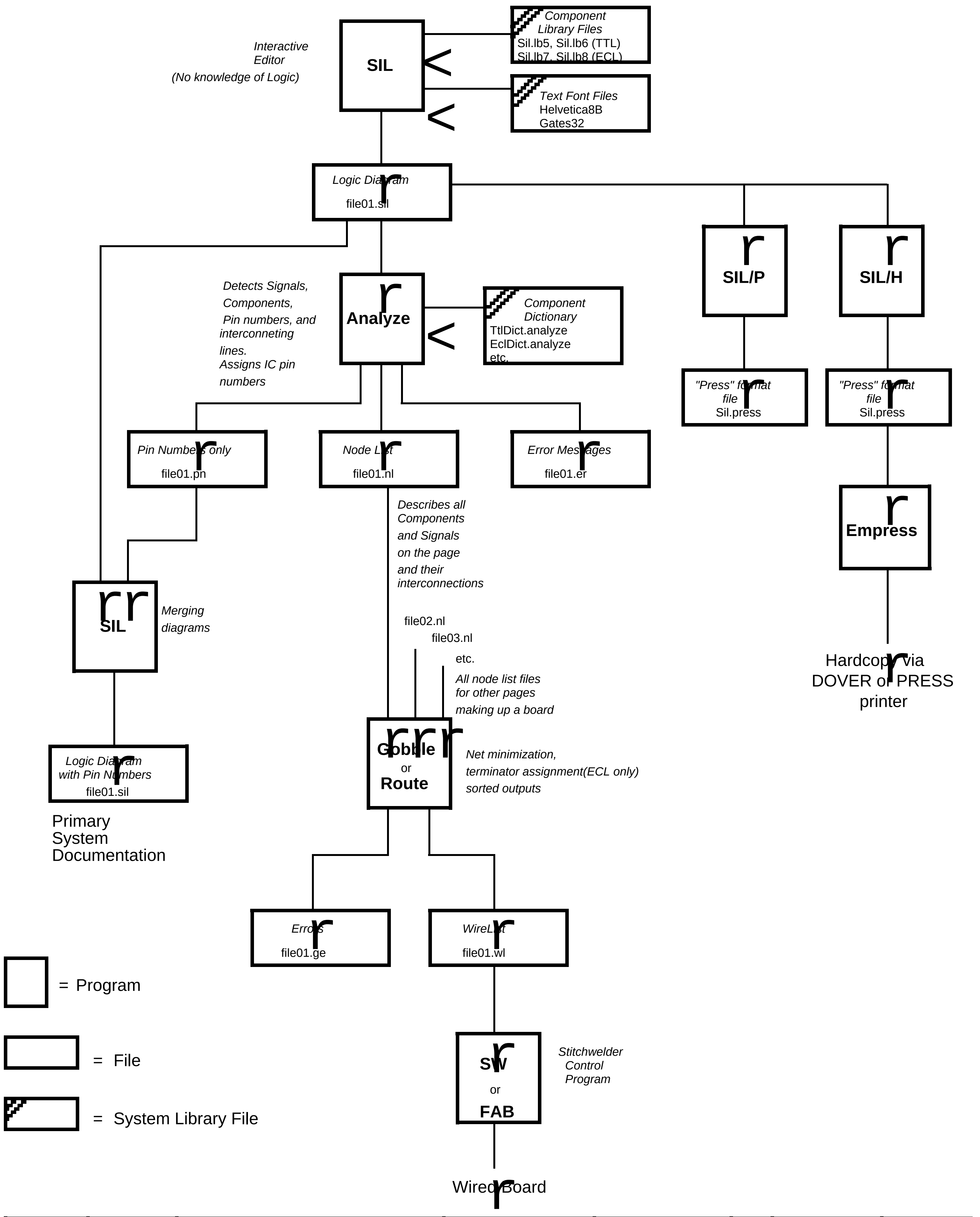
The pin numbers should be in ascending order.

WARNINGS, UNIMPLEMENTED THINGS, ETC.

As of 25 July 1977, the following cautions apply:

1. Gobble makes no use of the special POWER entries in the dictionary, and will only perform with the "standard" powering rules. Gobble really understands very little about power: it is willing to wire power to N,S,H, and P components. Others are assumed to have power already wired. This is being fixed.

2. By the same token, Gobble will not issue instructions for "cutting" already-wired pins away from committed power buses. This too will be fixed.



XEROX *PARC*	*Project* S I L	*Reference* Design Automation Flow	*File* SilManual01.sil	*Designer* R Bates	*Rev* A	*Date* 7/12/79	*Page* 30

SIL'S FONT 3 CHARACTER SET

a:	a	A:	A	
b:	b	B:	B	
c:	c	C:	C	
d:	d	D:	D	
e:	e	E:	E	
f:	f	F:	F	
g:	g	G:	G	
h:	h	H:	H	
i:	i	I:	I	
j:	j	P:	P	
k:	k	Q:	Q	*top*
l:	l	R:	R	*left*
m:	m	S:	S	*bottom*
n:	n	T:	T	*right*
p:	p	W:	W	
q:	q	X:	X	
r:	r	Y:	Y	
<:	<	Z:	Z	
>:	>	y:	y	
x:	x	z:	z	
		.:	.	'unconnected' pin

These characters define connection points in component definitions

Pieces of 16 pin packages

Small numbers as themselves:

1234567890

Larger numbers (5x7) as 'capital numbers:
i.e. '!' through ')', respectively

!@#$%~&*()

8/09/78

This figure shows the available characters in "Gates32.al"
These characters are normally read in by SIL as FONT 3

(this picture is available as <SIL>GATES32.press)

XEROX *PARC*	*Project* S I L	*Reference* Gates Symbols	*File* Gates32.sil	*Designer* R Bates	*Rev* A	*Date* 7/12/79	*Page* 31