4.7 SIL—A SIMPLE ILLUSTRATOR FOR CAD

Charles P. Thacker

This section describes SIL, a Simple Ilustrator implemented on a minicomputer equipped with a raster-scanned display. SIL was originally built to produce high-quality logic diagrams for digital systems, as well as other specialized illustrations. Although it has a number of limitations, SIL has been applied to a wider variety of applications than originally anticipated.

Rather than simply being used to produce documentation, SIL is now used to generate input for a group of programs that aid in the implementation of digital hardware. The logic diagrams produced by SIL are the only input required by these programs; the SIL files are interpreted to provide the information required by the rest of the design system. SIL has also been used for a wide variety of general-purpose illustration tasks.

Section 4.7.1 discusses the hardware environment in which SIL was built and the main objectives of the program. Section 4.7.2 describes the user interface and editing functions provided by SIL, and Sec. 4.7.3 provides an overview of the implementation. In Sec. 4.7.4, the major applications of SIL are described.

4.7.1 Environment and Characteristics

The primary goal in the design of SIL was to replace manual drafting in the preparation of hardware logic diagrams. The intended user community consisted of approximately 20 engineers and scientists engaged in the design of experimental digital hardware at the Xerox Palo Alto Research Center.

SIL was designed to run on the Alto (Fig. 1), a personal computer described in detail in Ref. 22. The Alto is a small machine with 64K 16-bit words of main storage, and a microprogrammed CPU capable of executing a typical register load or add instruction in approximately 2 μs. Local storage on the Alto is provided by a 2.5-Mbyte cartridge disk drive.

The principal output device of the Alto is an 875-line monochrome television monitor, oriented with the long dimension of the tube vertical. The monitor shows a bilevel black-and-white image 600 pixels wide by 800 pixels high, refreshed at 30 Hz. The image is represented in an integral frame buffer that

consumes slightly less than half of the Alto's main storage. The display hardware also supports a *cursor*, which is a 16 by 16 pixel bit map taken from a fixed area in memory. The cursor position can be set by a program, and the contents of the bit map are merged with the normal video at the indicated position. This implementation allows the contents and position of a small area of the screen to be ORed with the bit map rapidly without changing the contents of the main frame buffer.

Input is provided by a keyboard and a mouse, a pointing device that provides relative position information when rolled over a surface. In most Alto applications, including SIL, the mouse coordinates are clipped so that they lie within the boundaries of the display and are then used as the coordinates of the display cursor. The mouse also provides three buttons that allow the user to supply a small amount of information to a program without the necessity of moving his or her hand from the mouse to the keyboard.

A number of Altos are interconnected by an Ethernet network [14], a 3-Mbit/s packet-switched network that also provides access to remote file storage and printing services. The printer used for SIL output is a raster-scanned xerographic unit with a resolution of 384 lines/in., which produces 8.5 by 11 in. sheets.

Because of the characteristics of the printer, the contents of the SIL display correspond to a single hardcopy page. There are no provisions in SIL for using the display as a window onto a larger document, a feature often found in graphics systems. For applications involving figures for reports

Fig. 1 Alto personal computer display, keyboard and mouse.

and the like, this is not a limitation, since the normal format for these documents is an 8.5 by 11 in. page. Initially, there was concern that this format would be inconvenient for logic diagrams, since these diagrams are customarily prepared on 11 by 17 or 17 by 22 in. sheets. Experience has shown that this is not a problem, since a single page will usually contain a complete subsection of a design and fits conveniently in standard notebooks. A printed-circuit board containing 150 medium-scale integrated circuits, for example, can be shown in 15 to 20 SIL pages.

Objects. At its lowest level, SIL manipulates only two types of objects, *rectangles* and *text strings*. Rectangles are used for horizontal and vertical lines and for *backgrounds*, which are large areas that may be overlaid with other objects and printed in color on printers with color capability. Text strings may be specified in one of a number of fonts (typefaces) and may also be displayed in boldface or italic.

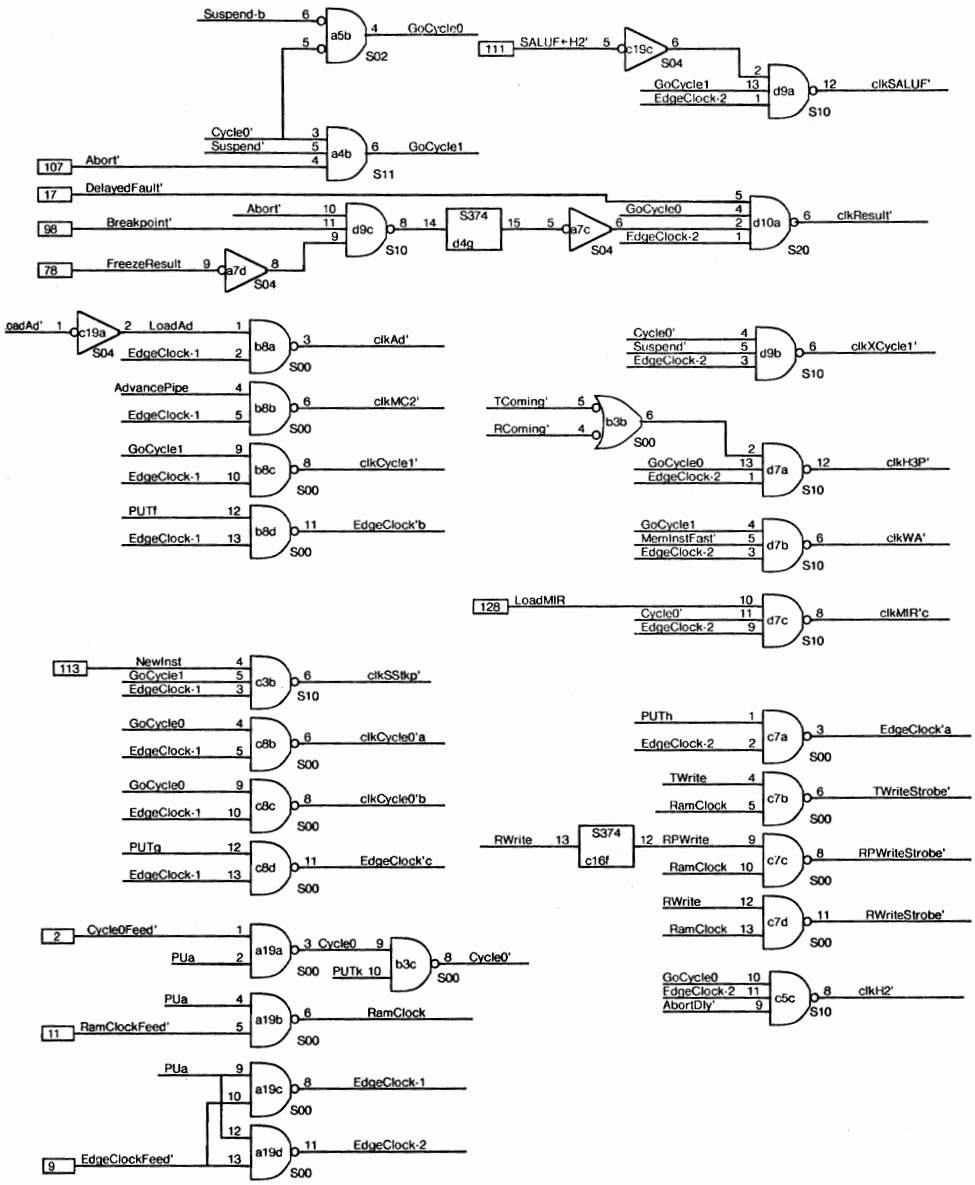

Fig. 2 Typical SIL logic diagram.

SIL also provides *display macros*, which allow an arbitrary collection of objects, including other macros, to be packaged together and subsequently referred to as a single character in one of five *macro fonts*.

Using only these primitives, a surprising range of illustrations may be created. For the logic drawing application, a font was designed that contains about 50 characters corresponding to the symbols on a logic designer's drafting template, and from this symbol set, several hundred display macros representing digital integrated circuits have been constructed. These macros are distributed as library files, and are used by all designers. Figure 2 shows a typical logic drawing produced with SIL using some of these macros.

One of the major apparent drawbacks of SIL is its inability to produce lines of arbitrary orientation. This disadvantage has been partially overcome by providing a font with characters consisting of line segments at various angles, and arcs of circles of various sizes. Using an operation that allows text to be placed at an arbitrary location with high precision, it is possible to produce complex figures using this font (see Fig. 3).

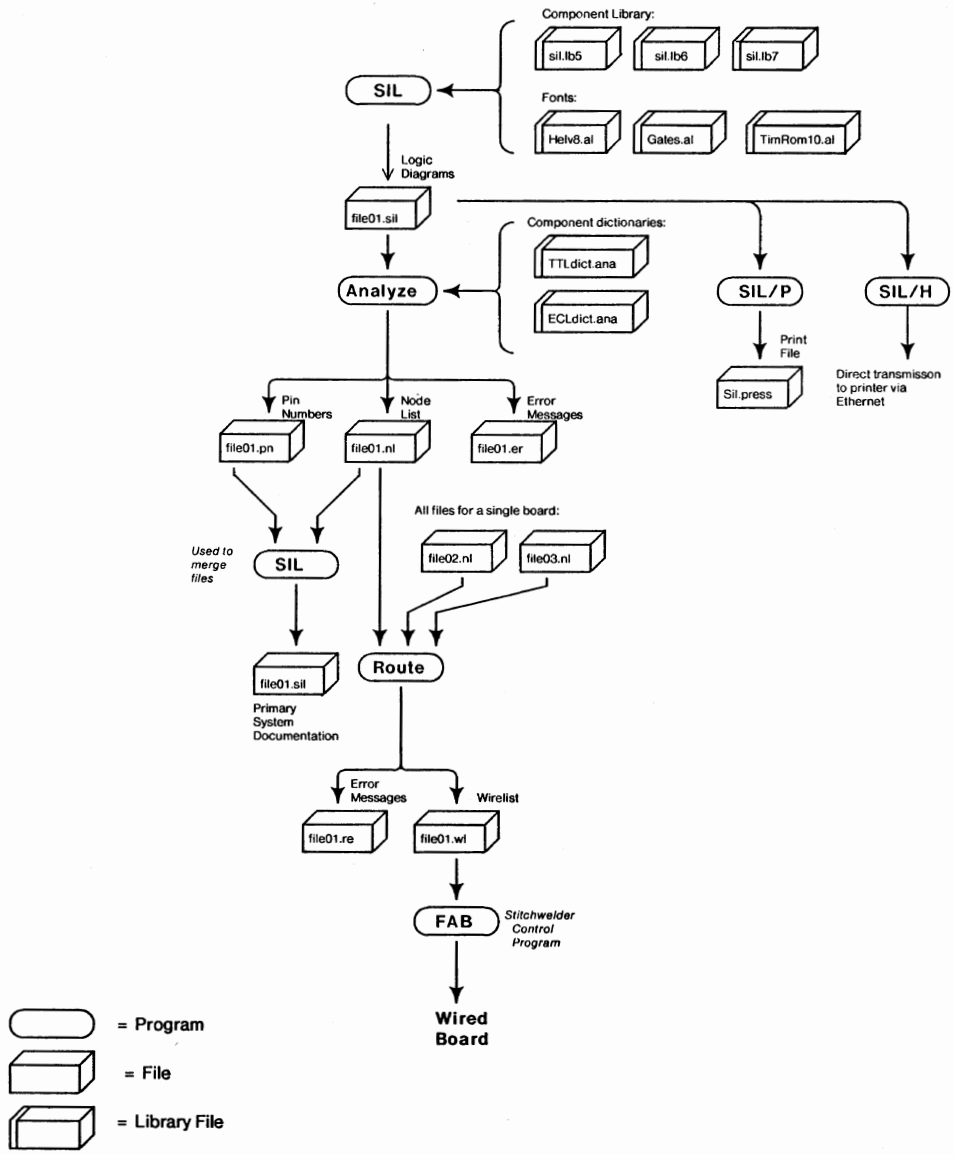

Fig. 3 SIL CAD system flow. This illustration uses a "template" font to achieve slanted lines.

SIL fonts are provided in two resolutions, a 75-pixel/in. version used with SIL itself, and a higher-resolution version suitable for the printer used for hardcopy output. SIL files contain the screen coordinates of the objects to be displayed, which are scaled appropriately for the particular printer used. The final output is thus a faithful rendition of the displayed image, but at high resolution. This makes the output much more readable than simply scaling the bit map to the resolution of the printer.

4.7.2 User Interface and Functions

The choice of a user interface is perhaps the most difficult part of the design of an interactive system, for once the functions the system is to provide have been selected, the user interface determines whether these functions can be accessed easily and naturally. The SIL user interface was designed with several primary ideas in mind:

1. The user should be able to develop a simple (although not necessarily correct) model of the operation of the program, and should be able to rapidly predict the result of a command, so that an appropriate operation can be selected easily.
2. The rate of interaction with the user should be high. Thus SIL commands are usually specified by a single keystroke or mouse button depression. This was done at the expense of a more easily learned command structure such as menu selection, since it was judged that speed of operation was more important than ease of learning. The office typewriter provided a precedent for this decision.
3. The amount of context that the user is required to remember should be small. The effects of a command should not depend on "modes" previously set by the user.
4. The user should not have to request status information from the program. Any status information required should be continually displayed, and the amount of such status should be minimal.

In this section, the functions provided by SIL and the commands used to evoke these functions are discussed. Also, the status display, which provides the user with information about the current state of SIL, is described.

Status Display. When SIL is started, the display screen is cleared with the exception of a single line of text, the *status line*, and three special indicators, the *cursor*, the *mark*, and the *origin*. Figure 4 shows the initial SIL screen. The cursor, the mark, and the origin serve to identify special locations on the screen. The cursor is a small arrow, displayed with the Alto's display cursor. Its position on the screen is changed by rolling the mouse over the work surface. The mark and the origin are short horizontal and vertical bars, respectively. They provide reference points for the commands described later, and are positioned using the mouse. To emphasize their positions, they blink approximately twice per second.

The status line displays the current values of a number of internal parameters, some of which may be changed by the user. It is updated by SIL whenever any of these quantities change. The significance of the entries in the status line shown in Fig. 4 is as follows:

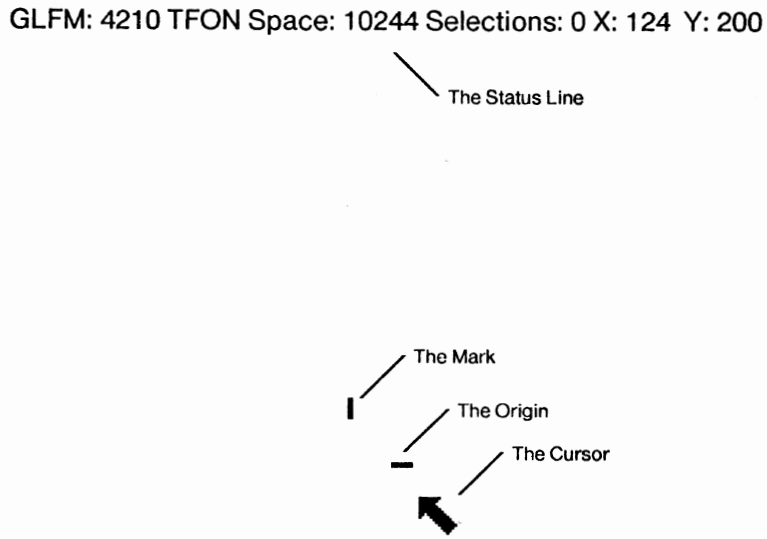

Fig. 4 Initial SIL screen.

GLMF: 4210b

G(4) is the current *grid*. The cursor and all objects added to the display will be constrained to start on points that are multiples of the grid spacing. The grid may be set by the user to any power-of-2 screen units (1,2,4,...). Use of a grid to constrain the placement of objects allows collections of objects with repetitive structure to be drawn easily.

L(2) is the current *line width*. The line width may be set by the user to any value between 1 and 9 screen units, and affects the width of any lines added to the display.

M(1) is the current *magnification*. SIL provides a facility (described later) to "zoom" a subsection of an image to allow precise placement of objects.

F(0b) is the current *font*. Any text characters added to the image will be displayed in this font. Four text fonts, one user macro font, and five library macro fonts are available. Boldface (**b**) and italic (**i**) attributes may be specified for text fonts. The correspondence between SIL font numbers and font file names is made by a group of entries in a *user profile* file. SIL and other applications programs read this file to obtain information user-specific parameters.

TF0N The first three of these indicators display internal state information that is usually useful only to SIL maintainers rather than to users. The fourth indicates the current color, Neutral (black) in this case. Any objects added to the picture will be printed in this color on printers with color capability.

Space: 10244 This entry indicates the number of unused words remaining in the storage pool for objects (i.e., how many more objects may be added to the display). A rectangle requires five words of storage; a string requires five words plus one word for every two characters in the string.

Selections: 0 This entry indicates the number of *selected* objects. Selection is a central idea in SIL, as most commands apply only to selected objects.

X: 124 Y: 200 These entries give the current coordinates of the mark. If a mouse button is depressed while the mouse is moved, the coordinates reported are those of the cursor. This facility allows the screen coordinates of objects to be determined rapidly by holding down a mouse button and pointing to the object.

In addition to the information above, the status line is used to display text used to prompt the user and to echo user input for some commands. For example, the names of input and output files are supplied by the user in this way.

Commands. The user provides commands to SIL using the keyboard and the mouse buttons. Keyboard commands are single *control characters*, generated by depressing the CONTROL key, then depressing the desired character key.

The most frequently used commands are specified with the three mouse buttons, sometimes modified by the CONTROL and SHIFT keyboard keys. The three mouse buttons are named MARK, DRAW, and SELECT, corresponding to their primary functions.

SIL commands use postfix syntax, in that the user specifies a location on the screen or an object or group of objects to be operated upon, then issues the command. All commands are carried as soon as the mouse button or key is depressed, and the only feedback usually given to the user is the command's effect on the contents of the display.

Selection. For commands that apply to existing objects, only the *selected* objects are affected. Selection of a single object is accomplished by moving the mouse so that the cursor lies over some portion of the object, and depressing SELECT. The selected object is displayed in gray (by changing alternate pixels in its bit-map representation white), and if any objects were previously selected, they are deselected and redrawn with solid lines and characters. CONTROL-SELECT is also used to select objects, but in this case, previously selected objects are *not* deselected. This allows the user to select a group of objects as the target of a command. It is also possible to select all objects within a rectangular area. One corner of the area is marked by moving the cursor to its location and depresssing MARK. The opposite corner is then located with the cursor and SHIFT-SELECT is depressed. SIL locates all objects lying completely within the area and selects them.

If an object is inadvertently selected, it may be deselected by pointing at it with the cursor and depressing CONTROL-SHIFT-SELECT. In all operations involving selection or deselection, when objects overlap, SIL chooses the object with the smallest perimeter as the target. When an object or group of objects is selected, the mark and the origin are moved to the upper left corner of the object, for use by later commands.

Adding Objects to the Display. Lines are added to the display by identifying the locations of the endpoints. First, the cursor is placed at one endpoint, and MARK is depressed. This places the mark at the indicated position. Then the cursor is moved to the other endpoint, and DRAW is depressed, drawing the line. Lines will be vertical if the difference of the endpoint x coordinates is less than the difference of the y coordinates, and horizontal otherwise. Lines are drawn at the current line width

shown in the status line. The user may change the line width at any time by typing CONTROL-W followed by a single digit.

When a line is drawn, it becomes the selected object, so that it may be the target of a *move* or *copy* command. Also, when a line is drawn, the mark is moved to the endpoint last indicated by the user. This allows a path to be drawn by marking the starting point and depressing DRAW once for each line segment comprising the path.

Text strings are added to the display by positioning the mark (with MARK) and typing the string, terminated by RETURN, ESC, or by depressing any mouse button. Control characters are provided during text entry to backspace over characters, words, or the entire string. Text is added in the current font shown in the status line, and this font can be changed by the user by typing CONTROL-F followed by a digit or a single letter. A digit indicates the font; a letter indicates an attribute (bold or italic) or one of 16 colors indicated by a single letter.

When the string is terminated, the mark is moved down by the height of the particular font used. This amount may be changed to a constant value, allowing text strings to be placed with precise vertical spacing.

Since commands are control characters, the first character of a string can be identified as such. When the first character of a string is typed, SIL enters *Add Text Mode*, and the message "Add Text" is placed in the status line. While the string is being entered, SIL will accept only commands that terminate the string, so it is not possible to change the font during the input. Note that Add Text mode does not violate the principle of modelessness described earlier, since the user does not need to take explicit action to enter the mode.

Changing the Display. Experience with SIL has shown that users spend considerably more time modifying existing drawings than creating new ones. To facilitate rapid changes, SIL provides a number of commands that allow a group of selected objects to be moved, copied, or deleted.

With a frame buffer, moving a large area of the screen is a time-consuming operation, since a large number of words in storage must be moved, and the source and destination are not generally word-aligned. For this reason, SIL does not provide the ability to "drag" collections of objects. Instead, a group of objects are selected, causing the origin to be placed at the upper left corner of the rectangle that bounds the objects; then the user specifies the new position that the origin should occupy by pointing to it and depressing MARK; and finally, the *move* command (CONTROL-X) moves the objects so that their origin is coincident with the new position of the mark. The origin and the mark are then interchanged, so that if the user is not satisfied with the new position, an additional CONTROL-X restores the situation to its original state. This requires three keystrokes per move: one to select the object, one to indicate its destination, and one to do the move. Because moving is a frequent operation, an idiom has been supplied to optimize it. Once the desired selection has been made, the cursor is pointed at the destination, and CONTROL-MARK is depressed. This is equivalent to MARK followed by CONTROL-X. The only reason for retaining the CONTROL-X command is to provide the "undo" feature.

The use of a frame buffer for display refresh introduces a problem when it is necessary to move an object that overlaps another stationary object. The desired effect is that the object should disappear from its original position and reappear at its new position, and that any overlapping objects should not be modified during this process. Although it is possible in principle to achieve this effect, it is computationally expensive, since when the object is being removed from the display, the program must decide for each of its pixels whether the final value will be white or black, depending on whether it is also part of an overlapping stationary object. The strategy used in SIL is considerably simpler and less expensive: When an object is deleted, an area of the display corresponding to the object's bounding rectangle is cleared, deleting the object. The coordinates of the bounding rectangle are passed to a background process whose job is to rebuild the screen by redrawing any objects that lie within the rectangle. The object is then redrawn at its new position. Since the background process cannot run until the object has been redrawn, the user will see the object at its new position, and then the area around the old position of the object will be filled in as the rebuilding process runs.

When a group of objects is moved, each member of the group is deleted individually, and the bounding rectangle passed to the rebuilding process is incrementally expanded to cover the entire set of objects. When all objects have been removed from the display, they are redrawn at their new position and the rebuilder runs as described above. This strategy allows the user to see the objects at their new position rapidly, and keeps the screen consistent. Unless the total area of the object that is moved is large and there are a large number of overlapping objects, the rebuilding process is usually not noticeable.

In addition to moving the selected object, SIL will also stretch or shorten any horizontal or vertical lines attached to the object, provided that the object is moved only in x or y, respectively. This feature was provided to stretch and shorten lines connected to components in logic diagrams, and may be disabled using CONTROL-SHIFT-MARK rather than CONTROL-MARK to move the object.

Copying is similar to moving. The user selects one or more objects, then indicates the new position of the origin using MARK. The *copy* command (CONTROL-C) then draws the selected objects at the

new position. An optimization, CONTROL-DRAW, is equivalent to MARK followed by CONTROL-C. After the copies are drawn, the original objects are deselected, the copy is selected, and the origin is moved to the upper left corner of the copy. This allows multiple copies to be made with one depression of CONTROL-DRAW per copy.

Since deletion is a common operation, two commands are provided for it. The first uses a keyboard command (CONTROL-D) to delete all selected objects. The second, SHIFT-DRAW, deletes the object at which the cursor is pointing, and is an abbreviation for SELECT followed by CONTROL-D. When an object is deleted, the origin is moved to the place it occupied, so that it may be replaced by another object using SELECT followed by CONTROL-X or CONTROL-C.

When objects are deleted, they are erased from the display but are not lost irrecoverably. Each deletion causes the selected objects to be placed on a last-in first-out stack, and this stack may be popped (and the objects restored to the display) by the CONTROL-U (undelete) command. The stack holds up to five groups of deleted objects; as more objects are deleted, the earlier ones are lost, and the storage space used by their representations is reclaimed.

Precise Positioning. SIL provides two commands that allow objects to be placed on the display with high precision. The first provides a "zoom" capability, and is invoked by identifying two opposite corners of a rectangular area with two successive marks, then typing CONTROL-E (expand). SIL executes this command by clearing the screen, calculating the integral magnification that will cause the indicated area to most nearly fill the display, and redrawing the portions of the display that will fit at this magnification. The magnification chosen is reflected in the status line. A second CONTROL-E restores the original display. While magnification is in effect, all other commands continue to operate normally.

The second positioning command provides a limited form of "dragging." It is invoked by selecting an object, indicating a reference point within the object by moving the origin to the point and depressing SHIFT-MARK, then typing CONTROL-S. SIL responds by copying the area around the reference point into the bit map for the hardware cursor, replacing the normal arrow. This patch of the selected object can then be moved about the display rapidly, and if a move (CONTROL-MARK) or copy (CONTROL-DRAW) command is issued, the selected object will be moved or copied such that the reference area in the object exactly overlays the portion of the object in the cursor. The normal arrow is restored by invoking any command other than move or copy.

Macros. The SIL macro facility allows a number of objects in a drawing to be encapsulated and subsequently treated as a single character. Macros may be user-defined and specific to a particular drawing, or they may be library macros. The definitions for user-defined macros are saved as part of the output file created for the drawing, but since the library macros are used for a number of drawings, their definitions are stored separately on specially named disk files. Font 4 is reserved for user-defined macros, and fonts 5 through 9 contain the definitions for library macros. When a particular library macro character is typed by the user for the first time in Add Text mode, the appropriate library file is read, and the definition is extracted from the file. Subsequent uses of the definition do not incur the disk delay.

To define a group of objects as the macro "M," the user selects the objects that are to make up the macro, and types "CONTROL-LM." SIL responds "Confirm with RETURN," or "Confirm with RETURN to overwrite" if there is already a macro with name "M." If the user issues the confirmation, the definition is created, and the original objects are replaced by an instance of the macro.

A macro definition may be modified by breaking an instance of it into its component objects, or *expanding* it, using the CONTROL-H command. The objects may then be modified individually, and the collection may then be redefined as the original macro. The CONTROL-V (view macro definitions) command puts the names of all existing font 4 macros into the status line, so that the user can determine which names have been used.

The main use of the SIL library facility is to provide a number of macros for digital integrated circuits. Library files are ordinary SIL files, distinguished only by their filenames, and the macro definitions they contain may be modified with SIL in the usual ways. To allow users to determine the contents of the libraries, a catalog is maintained with each library. This catalog consists of a number of SIL files showing the macros used for particular logic components, and the characteristics of the component. Designers keep a hardcopy version of the catalog at hand when using SIL for logic design.

4.7.3 Implementation

SIL is a *simple* illustrator primarily because of the simplicity of its implementation. Four aspects of SIL contributed to the simplicity of the implementation:

1. Since a SIL drawing represents a single 8.5 by 11 in. page, the number of objects will be small, usually between a few hundred and a few thousand. This means that the descriptions of all objects can be kept in main storage, and that simple linear search can be used to locate a particular object given its coordinates.

2. Since all SIL objects are rectangular, the routines to locate, clip, and draw objects on the screen are not complex. In particular, none of these routines require multiplication or division and are therefore quite fast.

3. By treating all complex objects as characters, it is possible to make use of an Alto machine instruction, *Convert*, that ORs the bit-map representation of a character into the display bit map. This increases the speed of these operations considerably over an implementation using load and store instructions.

4. Since SIL is interactive and runs on a personal machine as opposed to a shared system, there is a great deal of idle time available. This background idle time is used to simplify some of the operations required. Two examples are the reconstruction of the user display and reclamation and compaction of storage. Both of these operations are done in the background, and their implementation is simplified by not requiring that they be done rapidly.

In the balance of this section, the important features of the SIL implementation are discussed.

Representation of Objects. Each rectangle, background, and character string, whether a visible part of a drawing or part of a macro definition, is represented in memory by an *object descriptor.* The format of an object descriptor is shown in Fig. 5. It contains the coordinates of the object, its *type*, color, current *state*, and, if the object is a text string, the string itself. This information is sufficient to completely define each object, and is the only source of information about the object used by SIL. The bit map is reconstructed from the object descriptors when necessary.

The type field of an object descriptor determines whether an object is a rectangle or a string, and if it is a string, the font and type face to be used to display it. The I(italic) attribute in the object descriptor is applicable only to text fonts. The state field of the descriptor determines how the object is to be displayed. Objects with State = 0 are displayed normally. Selected objects have State = 1, and are shown in gray. Objects with $1 < \text{State} < 7$ are *deleted*. Deleted objects are not displayed, and each time a delete command is issued by the user, the state of all deleted objects is incremented. When the state becomes 7, the object becomes *dead*, and the storage it occupies will be reclaimed. When the user issues an undelete command, the state fields of all deleted objects are decremented, causing the most recently deleted set of objects to become selected. These objects are then displayed.

Each visible object has a single descriptor, and these descriptors are chained together in a singly linked list. A new object descriptor is created, added to the head of the list, and displayed each time the user issues a draw or copy command, or terminates a string in Add Text mode. In addition to descriptors for visible objects, there are also descriptors for components of macro definitions. Each macro definition is a separate list, the head of which is an element of an array (the *Macro Table*) indexed by the font and character that is the macro name. When a macro is defined by the user, the descriptors making up the macro are removed from the list of visible items, their coordinates are made relative to the origin of the macro, and they are chained together starting at the Macro Table entry corresponding to the font character specified by the user. A single descriptor for the macro is then added to the visible object list.

A single routine is used to scan convert an object into the bit map. This routine draws rectangles by ORing into the bit map at the locations determined by the coordinates of the object, converts strings using the machine's Convert instruction, and converts macros by calling itself recursively for each of the objects making up the macro definition.

The files produced by SIL are similar in format to the main storage representation. A SIL file consists of a one-word *password* that identifies the file as a SIL file, followed by a number of records, each one representing a single object. The format of the records is the same as that of an object descriptor, with the exception that the state field is always zero and the link field contains -1 if the record describes a visible object, and contains the (one byte) macro name if the record is part of a macro definition.

Rebuilding the Screen. When an object is initially added to the display, it is necessary only to scan convert it at its proper location, but when objects are moved, the situation is more complex. In this case, it is necessary to redraw any objects that lie within the bounding rectangle of the moved object, since this area is cleared as part of the move. This is done by a *rebuilder* routine that is invoked whenever there are no commands to be processed. The rebuilder is passed the bounding rectangle of the area to be redrawn, and it traverses the entire chain of visible objects, scan converting any objects that lie within this area. Each time an object is inspected, the rebuilder checks for new commands, and processes any that arrive. If a new command modifies the screen before the rebuilder is finished, the rebuilder is restarted at the head of the visible object list, and the size of the bounding rectangle is increased to be the smallest area that covers both the original and the new area. This simple technique works well only because the total number of objects is small and there are excess cycles available. If this were not the case, a more complex data structure that could rapidly locate all objects lying within a given area would be necessary.

Similar considerations apply to the problem of determining which objects are desired when the user makes a selection. SIL must examine the entire visible object list to find the one at the indicated

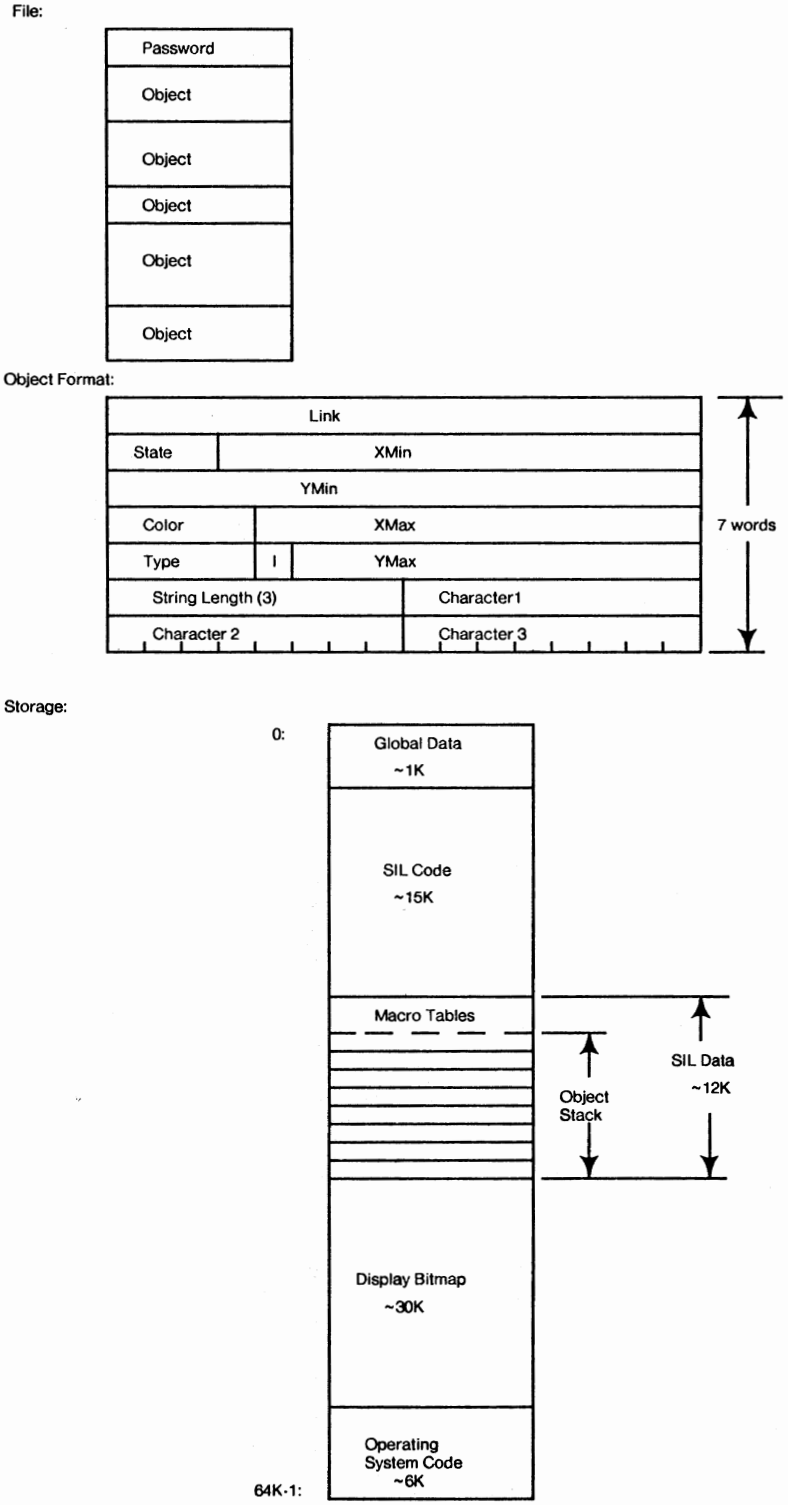

Fig. 5 SIL files, objects, and storage layout.

coordinates with the smallest perimeter (if a single item is selected), or to find the set of objects lying in a specified area (in the case of area selection). For a typical display, the time to do a selection is much less than a second, but this is only because the total number of objects is small.

Storage Management. The amount of storage required for object descriptors expands and contracts as objects are added to and deleted from the display. Because of the small amount of main storage in the Alto, it is important that available space be used efficiently. SIL uses a simple technique that capitalizes on excess cycles to manage storage.

Storage for object descriptors is allocated from a stack. As objects are deleted, "holes" appear in the occupied region of the stack. A compacting routine, called only if there are no commands to be processed and if no screen rebuilding is required, is responsible for eliminating the holes by compacting storage. Like the screen rebuilder, the storage compactor is incremental. It continually traverses the occupied region of storage looking for dead descriptors. When it finds one (or more than one), it copies the first "live" descriptor it finds beyond the dead region down in memory, and readjusts the link field of the descriptor that points to the one that was moved. This causes the dead blocks to "bubble" to the top of the stack. Each time a descriptor is examined, a check for a new command is made so the process is invisible to the user.

4.7.4 SIL APPLICATIONS

The major application of SIL is to produce input for a digital logic design system. Figure 3 shows the information flow in this system, which is composed of several programs in addition to SIL.

The initial input to the system is a set of SIL drawings showing the desired logic, its interconnections, the types of all components, and their locations on a standard Stitchweld board. Each SIL file is processed by *Analyze*, a program that determines the interconnection of the components by interpreting the SIL files. Analyze produces three output files: The first is a "node list" file in text form containing a list of the integrated circuits used in the drawing and the pin numbers connected together by each signal path. The second is a SIL file containing any pin numbers not preassigned by the user. If the user assigns pin numbers in the original drawing, Analyze will check them for validity; otherwise, it will assign them from information in the *component library* file. The SIL file containing pin numbers is merged with the original drawing using SIL to produce the primary system documentation. The third file produced by Analyze is a text file describing any syntactic errors discovered during processing.

A set of node list files describing an entire board are then processed by the *Route* program. Route does further checking for unused or multiply used signal names and pins, and produces a wiring list for the board. Route uses a *board description* file that specifies the locations of all possible components on the type of board being used, and minimizes the overall length of the wiring. The wire list file generated by Route is used to drive a semiautomatic Stitchweld machine that does the actual wiring.

The major limitation of this system is that it does not do automatic component placement. It does eliminate most of the clerical work associated with logic design, and allows designers to produce all the documentation for a system with no support staff. Using this system, it has been possible to design, build, and test prototype boards containing approximately 100 integrated circuits in as little as 2 weeks.

The design system described here was built *after* SIL, when it was realized that the information in a SIL file could be interpreted to generate the interconnection information required by a routing program. SIL itself has no special knowledge of logic design, in contrast to the graphics components of most design automation systems.

SIL has also been used for a number of general-purpose illustration tasks, such as the preparation of the illustrations for this section. Many standard office forms have been prepared as SIL files, and are filled out by individuals using SIL. SIL is also used for preparing charts and tables for reports, and figures for technical publications. A program that merges several text files into a single file suitable for printing has been augmented to allow SIL drawings to be placed at arbitrary locations in these documents, further increasing SIL's utility.

Acknowledgments

A number of individuals have contributed to the success of SIL and the CAD system of which it is a part. Roger Bates implemented the color printing facility, as well as a number of extensions to the original SIL. Ron Pellar developed the font shapes used with SIL. Ed McCreight designed and implemented Route, and Butler Lampson made a number of helpful suggestions for improvements to the user interface.