SIL, ANALYZE, GOBBLE, BUILD Reference Manual

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

July 12, 1979

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 MAXC1 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 (or Helvetica10N.al)
Helvetica7.Al (or Helvetica7B.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.Ears and TTLDataSheets.Ears.

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

GOBBLE/ROUTE
Gobble.Run
or
Route.Run

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 numb r 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.

↑shB: 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.

↑Fn: 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 DarkBrown, Red, Orange, Yellow, Lime, Green, Turquoise, Cyan, Aqua, Violet, Ultaviolet, Magenta, Pink, Smoke, Neutral (black), or White respectively.
All new items are created according to these parameters as appropriate.
See Printing below, for more information about the use of colors.

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

↑Hc: 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 or 2.

↑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).

↑Lc: 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).

↑shVn: 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.

↑Wn: 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 fot 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 "TF0N" part of the status line is non-zc 'o. 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 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:

C/i 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

Whenever an entry of this type is encountered, it will effect the printing of all following files.

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.
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).
N.NN/b 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.

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 $\leq$letter $\leq$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

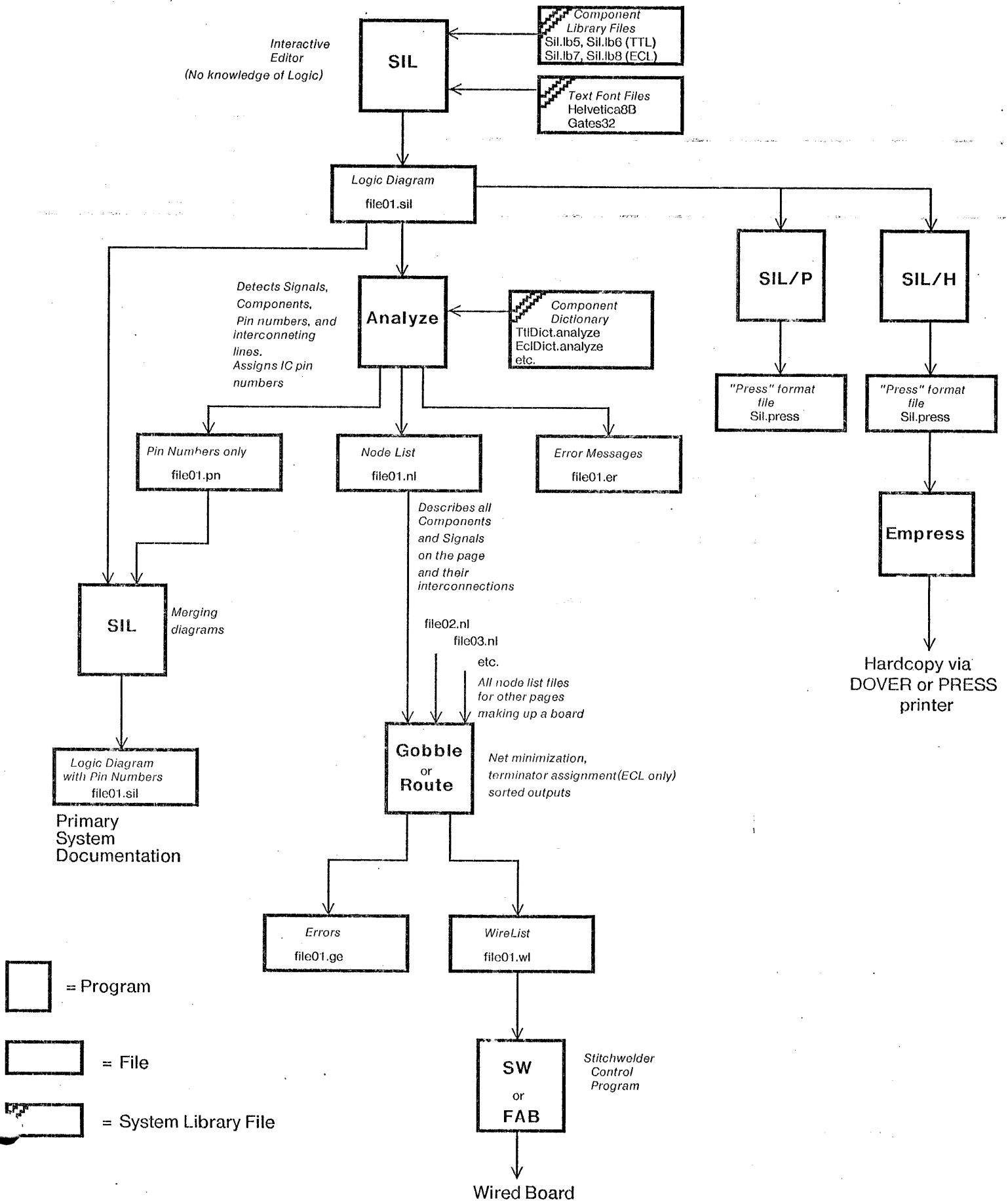
The pin numbers should be in ascending order.

WARNINGS, UNIMPLEMENTED THINGS, ETC.

As of 25 July 1977, the following cautions apply:

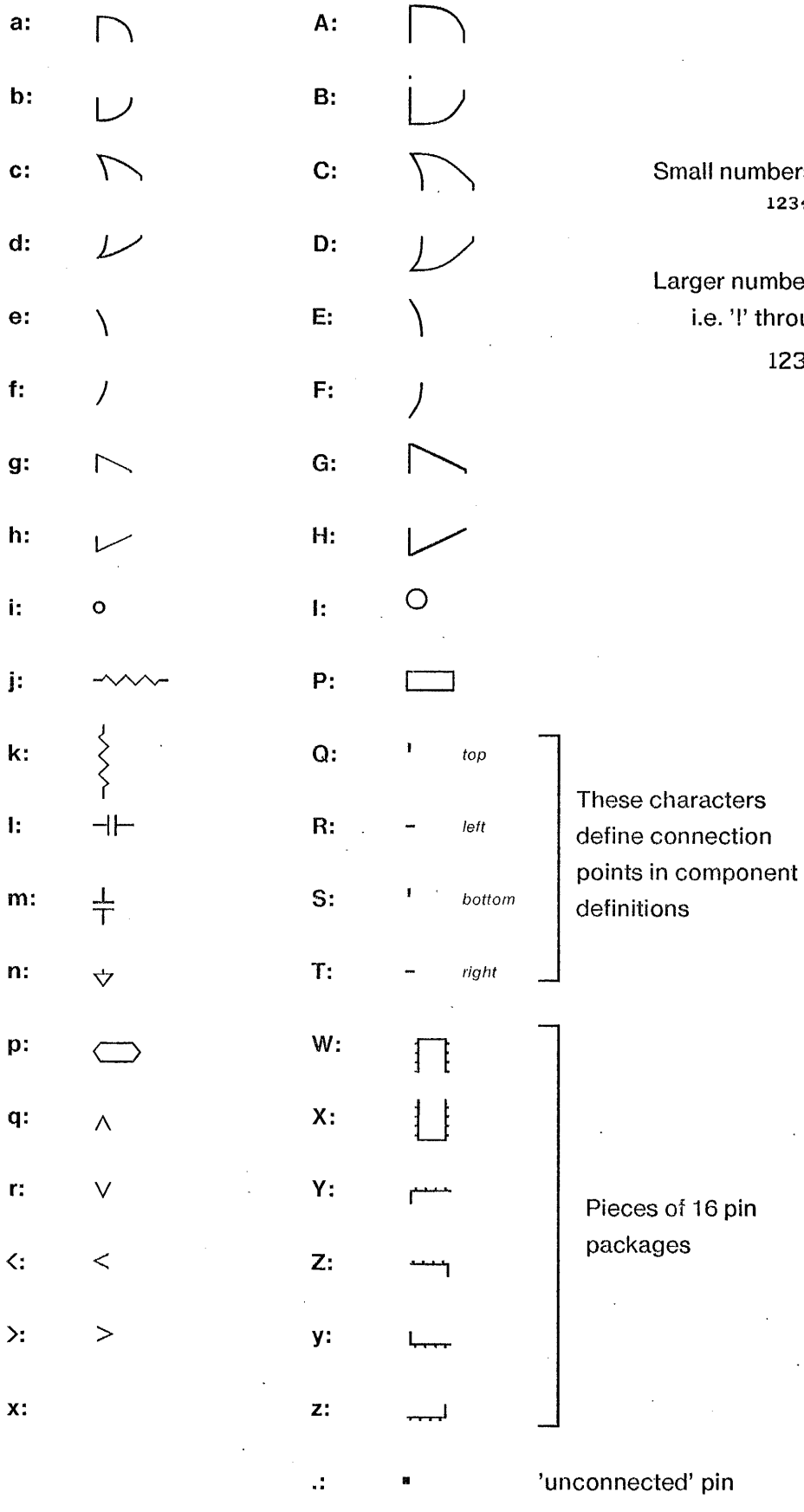
1. Gobble makes no use of the special POWER entries in the dictionary, and will only perform with the "standard" powering rules. Gobble really understands very little about power: it is willing to wire power to N,S,H, and P components. Others are assumed to have power already wired. This is being fixed.

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	Reference	File	Designer	Rev	Date	Page
	SIL	Design Automation Flow	SilManual01.sil	R Bates	A	7/12/79	30

SIL'S FONT 3 CHARACTER SET



Small numbers as themselves:

1234567890

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

1234567890

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)

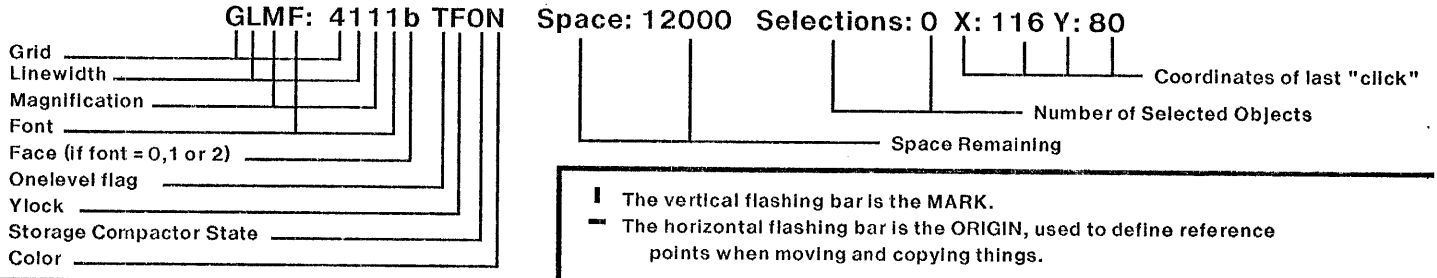
XEROX PARC	Project SIL	Reference Gates Symbols	File Gates32.sil	Designer R Bates	Rev A	Date 7/12/79	Page 31

SIL COMMAND SUMMARY

(available as <SIL>SilSummary.press)
(Print on Color printer if you can!)

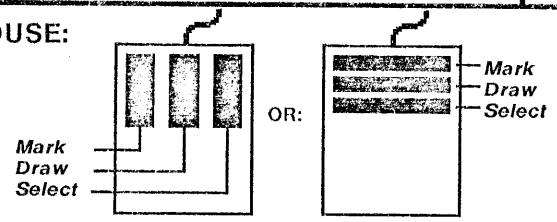
STATUS DISPLAY:

AT THE TOP, YOU ARE SHOWN THE STATUS OF SIL IN THE FOLLOWING FORMAT:



I The vertical flashing bar is the MARK.
— The horizontal flashing bar is the ORIGIN, used to define reference points when moving and copying things.

THE MOUSE:



BASIC MOUSE BUTTONS:

Mark: Remember this point, but take no action.
Draw: Draw a line from here to the last mark.
Select: Select the object pointed to (shown halftone).

MOUSE BUTTONS PLUS SHIFT AND/OR CONTROL KEYS:

Shift Mark: Move the origin (horizontal flashing bar) to here.
Control Mark: Move selected object to here (same as mark, control-X).
Cont/Shift Mark: Same as Control Mark except attached lines are not pulled.
Control Draw: Copy selected object here (same as mark, control-C).
Control Select: Select this object, but don't deselect previous one.
Shift Select: Select everything in the area between here and the last mark.
Cont/Shift Sel: Deselects the object pointed to.
Shift Draw: Delete just the object pointed to.
Cont/Shift Draw: Undelete (same as control-U).

When you select something, the orign goes at its upper left corner (upper left corner of the area if you use shift-select). You can also put the origin where you want it, using shift-mark.

COMMANDS (Control Characters):

↑A: Display text from an 'Alternate text file' one line at a time (text appears in place of STATUS).
↑B: Draw a box. The origin and the mark define the corners. (with shift down, draws a background)
↑C: Copy. Everything selected is copied so that the origin is at the cursor.
↑D: Delete all selected items.
↑E: Magnify display. The last two marks define the corners of the area to be magnified.
↑Fn: Change to new Font, Face, or Color.
↑Gn: Set Grid to 2**n
↑Hc: Expand the (font 4) macro c at the mark.
↑I: Input a file (name is requested). Edit with BS, ↑Q, confirm with CR.
↑Jn: Jam a new text Font, Face, or Color into selected item. (n = 0-2, b/i, or color)
↑K: Reinitialize SIL (requires confirmation if picture changed).
↑Lc: Define the group of selected objects to be the (font 4) macro c.
↑N: Complement the onelevel flag.
↑O: Output to a file (name is requested).
↑P: Write the current picture ("put it") on the file SIL.TEMP.
↑Q: Quit (requires confirmation if picture changed).
↑R: Delete all font 4 macro definitions (as well as any uses of them in the picture).
↑S: Show. Copy the area around the origin into the cursor. Allows precise copying or moving.
↑T: Turn "ticks" on and off.
↑U: Undelete the last group of things deleted with control-D or 'Shift Draw' (up to 5 levels).
↑Vn: View all items of a given text Font, Face, or Color, or macro definitions of fonts 4-9.
↑Wn: Set the width of lines to n (0-9).
↑X: Xlate (move) all selected objects so that the origin is at the mark.
↑Y: If Ylock is false, make it true and set YINC to the distance (in y) between the origin and the mark. Subsequent CR's will move the mark down by this amount. Used for precise leading.
↑Z: Hardcopy mode switch on/off. In 'Hardcopy' mode, text characters are spaced according to printing widths.
BS: Edit the 1 selected string. You are restricted to BS, ↑W, ↑Q, and append to the END of the string (see Add Text).
↑←: Moves the STATUS DISPLAY to the screen location of the last mouse button action.

Add Text: (non-control characters)

If you start typing, text goes into the picture at the mark. You may edit with BS (backspace character), ↑W (backspace word), and ↑Q (backspace line). Terminate with ESC, CR, or by pushing any mouse button (DEL to abort).

COLOR

The three commands to use are:
↑J: to Jam a new color
↑F: to select a new color
↑V: to select all items of a color

The available default colors are:

D for DarkBrown
R for Red
O for Orange
Y for Yellow
L for Lime
G for Green
T for Turquoise
C for Cyan
A for Aqua
V for Violet
U for UltraViolet
M for Magenta
P for Pink
S for Smoke
N for Neutral
W for White

Fonts: Printing-font Alto-font *(standard for logic design)*

Font 0: Helvetica10B Helvetica10N
Font1: Helvetica7B Helvetica7B
Font2: Helvetica7BI Helvetica7B
Font3: Gates32 (for logic) or Template64 (for diagrams)
Font 4 is user-defined macros (saved with picture on file)
Font 5-9: Library macros for logic design

To print:

Sil/P file1 file2 . . . Makes Sil.press
Sil/H file1 file2 . . . Makes Sil.press and invokes Empress.run for printing
Sil/H file1 . . Host/H 4/C sends sil.press to printer host to make 4 copies
Sil/H Out.press/F file1 . . Makes Out.press from files etc.
Sil/I Initialize sil. Must be used when fonts and library files are changed.
Other commands are available for controlling color content of press files

Last updated July 11, 1979

Inter-Office Memorandum

To	Sil Users	Date	August 1, 1979
From	Roger Bates	Location	Palo Alto
Subject	Sil file definition	Organization	CSL

XEROX

Filed on: [MAXC]<Sil>SilFileDescription.press

This is a description of the SIL file structure. The definition of the Sil file format was made solely for the convenience of the Sil program, and is simply a copy of the internal data structure that Sil maintains in memory during operation.

The first word in the file is a check code, that can be used to see that the file is actually a Sil format file. This word has two possible values, octal 34562 and 34563. The second value has been adopted as part of the design automation process to indicate the a Sil schematic file has been "build" into a wire-list file via the Build process.

Following this word are a secession of "item blocks" that describe the contents of the Sil drawing. An Item block has the following form:

word 0: macro-code

If this word equals a character code, then this item is to be added to the list of items for the macro Font 4 character. All items which are part of macro definitions will appear first in the file.

If this code is -1, then this is a normal item (string, line etc) and is to be displayed once as indicated by the following words.

word 1: state and X-min

bits 0,,3 are used internally to indicate the current state of an item - normal, selected, or deleted.

bits 4,,15 are the minimum X coordinate bit position of the bounding box that the item occupies.

word 2: Y-min

bits 0,,15 are the minimum Y coordinate bit position of the bounding box that the item occupies.

word 3: color and X-max

bits 0,,3 are used to indicate the color associated with this item according to the following table:

00: Neutral	08: Brown
01: Red	09: Orange
02: Yellow	10: Lime
03: Green	11: Turquise
04: Cyan	12: Aqua
05: Violet	13: UltraViolet
06: Magenta	14: Pink

07: White 15: Smoke'

bits 4,,15 are the maximum X coordinate bit position of the bounding box that the item occupies.

word 4: font, face, and Y-max

bits 0,,3 are used to indicate the "font" associated with this item. Font values are interpreted according to the following table:

```
00: Font 0
01: Font 0 with "boldness" opposite to default value in user.cm
02: Font 1
03: Font 1 with "boldness" opposite to default value in user.cm
04: Font 2
05: Font 2 with "boldness" opposite to default value in user.cm
06: Font 3
07: Font 3 with "boldness" opposite to default value in user.cm
08: Font 4 macro
09: Font 5 macro
10: Font 6 macro
11: Font 7 macro
12: Font 8 macro
13: Font 9 macro
14: A line with lingth and width indicated by preceeding coordinates
15: A background with area indicated by preceeding coordinates
```

bit 4 is used to indicate the "italics" opposite to default value in user.cm. This bit only has significance if the font value is 0 through 7.

bits 5,,15 are the maximum Y coordinate bit position of the bounding box that the item occupies.

word 5 etc: optional string

If the preceeding word indicated an item of fonts 0 through 13, then additional words are required to define the characters in the item string. This is done with a BCPL string definition, that is, the left byte of word 5 is the number of characters in the string, the right byte is the first character. Additional words are appended as needed to contain the specified number of character bytes.

All coordinate value are expressed in "Alto Screen" units and are refernece to the upper left hand corner of the screen. When making a press file, Sil scales all coordinate values by 17780/500.

Notice that descriptions of library fonts 5 through 9 characters are not defined within this Sil file. A table is filled in by Sil during input, and later the library files indicated in user.cm are opened, and the front of the file is scanned for items defining the required character codes for each of the library fonts characters.

Editorial comment

Notice that in the preceeding definition, there are several instances of references to the contents of the user.cm to get font family name, default faces, and library file names. This means that a signifacant part of a Sil picture is not defined by the Sil file at all. Users of Sil may come back to an old file that they carefully saved away only to find that they really "lost" part of the drawing because they change some of the parameters in their user.cm file for new applications.

This has even more impact for people interested in processing Sil files by other programs, since they must either work around the parameters in user.cm, or deal with user.cm

When one runs Sil/I, Sil reads user.cm for font information, and file pointers, and saves this information in two files called Sil.fonts, and Sil.fps (file pointers). The proceedures for reading these two scratch files could easily be lifted from the Sil sources [MAXC]<SIL>SilSources.dm.

Programs should in fact only use user.cm to fill in default values when first needed. All information should be explicidly saved as part of the output file, so that no future refernences should be dependant of the contents of user.cm

ROUTE Program Logic Manual

E. McCreight
Parc/CSL

Draft of June 2, 1978 11:17 AM

ROUTE is a Bcpl program that is part of the Parc/SDD/EOD electronic design automation system. Its function is to combine the net lists describing a number of logic drawings that together describe an entire logic board, and to generate a set of wiring orders sufficient to produce the board automatically.

It has been said that a university is an otherwise unrelated set of colleges sharing a common heating system. So it is with ROUTE. ROUTE is the current repository for a fairly large collection of unrelated functions sharing a common net list input format and wire list output format. These functions are such diverse things as automatic terminator assignment, wire length minimization, and wiring order determination.

ROUTE is further complicated by the necessity to carry out revisions incrementally. This means, among other things, that nets must be recognized as "the same" even if net names change, and that terminator assignment must change as little as possible.

0. Normal Operation

ROUTE is normally invoked from the Alto Executive with a command line like this:

> **Route**[/*switches*] *board*/**B** [*metric*/**M**] [*exhaustive*/**E**] [*heuristic*/**H**] [*boardlocation*/**L**] *file1 file2 file3* ...

ROUTE normally reads a set of .nl-format files produced by the ANALYZE program and produces several output files. If the first .nl file in the input list (*file1*) had the name **AARGH.nl**, then the following output files would be produced

> * **AARGH.wl**, a file containing wiring orders for the stitchwelding program FAB to fabricate the board from scratch,
>
> * **AARGH.re**, a file containing error messages,
>
> * **AARGH.bp**, a file describing the external (backpanel, usually) connections of the board, and
>
> * **AARGH-*x*.nl**, a file for each each external connector type *x* describing the connections through that connector. These files are intended to be used as input for automatic backpanel routing.

If correction (revision) is specified, in addition ROUTE will read **AARGH.wl** and will produce two other output files:

> * **AARGH.wlnew**, instead of AARGH.wl. That's so that if something goes awry, the original AARGH.wl is unchanged and the correction can be re-run. The other output file is

* **AARGH.ad**, a file containing FAB wiring orders to implement the revision on the pre-existing board.

The following switches have the indicated effect:

c: This specifies that correction is to be done.
b: This causes two .wl-format files to be produced, one containing all pins that are initially floating, and one containing all pins trace-wired to a power plane. This is useful for automated incoming board inspection.
m: This requests Multiwire-format wiring output, suitable for sending to Photocircuits, Inc.
h: This causes a list of hole positions to be produced in Multiwire-format. Photocircuits needs to know where not__ to put the wires, too.
t: This causes a check list of trace-wired pins to be produced (not normally useful except for Board debugging (see below)).

The file *board* contains a concatenation of .BR files that collectively define all the Board routines (see below). The parameters *exhaustive* and *heuristic* together control the wire-routing part of ROUTE; their normal settings are 7 and 20. Better routings of long nets can be gotten at the expense of longer running times by increasing *heuristic*. The *metric* is either **Manhattan** (rectilinear) or **Euclidean** (as the crow flies); the default is **Manhattan**, which is faster. The *boardlocation* is a string that will be substituted for the connector name *x* in the **AARGH-*x*.nl** file mentioned above.

1. File Formats.

1.1 .NL format.

An .nl-format file is an ASCII text file that looks like this:

<comment line>
<IC line>
...
<IC line>
@
<net line>
...
<net line>

The <comment line> is ASCII text preceded by a ";" and terminated by a carriage return. ANALYZE generates the comment from a piece of "boiler plate" in the drawing. The standard boiler plate template is known as LogicBlock.sil.

The <IC line> has the following format:

<IC position>: <Short IC name>(<long IC name>/<npins>/<IC family>); <used group string>

For example,

h24: S04 (SN74S04/14/S) ; badfe

The <IC position> is a string in one of two forms: either a lower-case letter followed by a decimal number, or a string preceded by "#". In the former case, the name is normalized by suppressing leading 0's and then forcing the decimal number to be two digits long or longer. Thus **a004** and **a4** would both be normalized to **a04**, and **c04000** would be normalized to **c4000**. Interpretation of this normalized IC position string is strictly up to the board routines, but some guidelines have developed among board designers:

* It is pretty clear what ought to happen when the board socket and the IC are congruent. It is less clear what ought to happen when a Sip is plugged into a Dip socket, or an 8-pin Dip is plugged into a 16-pin socket, etc. Usually, something like **a41** means that pin 1 of the IC is supposed to go into pin 1 of the board's socket **a41**, and the rest of the pins of the IC plug into other pins of the board in an obvious manner defined by the board routines.

* For some boards, **a41** means that the IC is to be inserted in some board-standard part of **a41**. For example, D0 boards are covered with 20-pin sockets whose pin 10's are trace-wired to GND. The D0 board routines interpret the number **a41** on a 14-pin TTL IC as meaning that pin 1 of the IC goes in pin 4 of the socket, so that pin 7 of the IC goes in pin 10 of the socket, the GND pin.

* Most boards have adopted an offset convention. In this convention, **#3a41** means that pin 1 of the IC is to be offset 0.3" in the direction of increasing socket pin numbers from pin 1 of the socket. For the D0 board above, **a41** and **#3a41** would specify the same position for a 14-pin TTL IC. If you wanted to put an 8-pin Sip in pins 12-19 of a 20-pin 300-mil-wide Dip socket, you would say **#1←3a41**, signifying a "sideways" shift of 300 mils and a "vertical" shift of 100 mils.

* More positioning conventions will likely evolve. The idea is for any reasonable positioning to be possible, and for common ones to be easily specified, preferably to happen by default.

The <IC family> is a string from which ROUTE infers a number of characteristics of the IC, such as

a) to what pins (if any) automatic terminator assignment applies,
b) what fixed voltages are applied to what pins,
c) how to compute the co-ordinates of pin *i* given the co-ordinates of pin 1.

The <net line> has the following format:

<net name>: <pin>, <pin>, ..., <pin>

For example,

Sin.15: j26.13o, E146
stor08.sil+8: g22.2i, k25.12o, j23.11i
WEo': g22.3i

A <net name> is an alphanumeric string beginning with an alphabetic character and optionally ending with the character "!". Two net names differing only in the final "!" are considered to be the same for matching purposes, and the "!" affects only automatic terminator assignment.

In general, a <pin> may be one of the following:

<Connector string><decimal number><**i/o/p/nothing**>
<IC position string>.<decimal number><**i/o/p/nothing**>

A connector string is something like **E** or **C**. It may not end with a period or digit. An IC position string is something like **a41**. It can contain periods but may not end with one. Interpretation of the connector strings and IC position strings rests with Board routines to be described later. The final letter **i** or **o** or **p** indicates whether ANALYZE believes that the pin is an input pin, an output pin, or a power pin. These beliefs form the basis of some of ROUTE's warning messages.

1.2 .wl *format.*

An .wl-format file is an ASCII text file that looks like this:

<board type line>
<comment line>
...
<comment line>
<IC line>
...
<IC line>
@
<basic wiring command>
...
<basic wiring command>

The <board type line> consists of a string followed by a carriage return. The string uniquely encodes the board type. The <comment line>'s are just the comment lines collected from the various .nl files. An <IC line> now has the following form:

<IC position>: **(**<long IC name>/<npins>/<IC family>**)**; <pin number>,...,<pin number>

where the <pin number>'s are pins unused by any of the input .nl files. For example,

i26: S04 (SN74S04/14/S); 8,13,14,15,16

A <basic wiring command> looks like one of the following:

<CR>**CALIBRATE:** <<decimal command number>>; <operator instructions><CR>
<four blanks><pin> <co-ordinate> ... <CR>
<four blanks><pin> <co-ordinate> ... <CR>
...

or

<CR>**DISCONNECT:** <<decimal command number>><CR>
<four blanks><pin> <co-ordinate> ... <CR>
<four blanks><pin> <co-ordinate> ... <CR>
...

or

<CR><net name>: <<decimal command number>> (<decimal wire length>)<CR>
<four blanks><pin> <co-ordinate> ... <CR>
<four blanks><pin> <co-ordinate> ... <CR>
...

A <co-ordinate> is a string of the form:

{<decimal number>,<decimal number>}

where the two decimal numbers are interpreted as distances in the x and y axes, measured in units of .025 inch, from the "origin" of the board (an arbitrary fixed position).

For example,

```
Sout.00: <8> (7)
      b01.05i {052,007}      C176 {052, 000}
```

CALIBRATE is a command that causes FAB to solicit operator assistance in attaching the board to the x-y table, attaching a working tool, and locating four calibrating points on the board, which must form the corners of a rectangle. After that FAB can find every point by itself until the board is removed from the table.

DISCONNECT is a command to operate a milling tool to isolate a stitchweld pin from the trace-wired net to which it was originally connected during board manufacture. This facilitates installing TTL IC's in sockets intended for ECL IC's, for example.

<net name> is an implicit command to wire up the named net in the same order as the pins are mentioned.

1.3 .ad *format.*

An .ad-format file is almost a superset of a .wl-format file. There are two differences. First, an .ad-format file does not have the <board type line>. Second, the .ad-format file allows several additional wiring commands:

<CR>**UNPLUG:** <<decimal command number>><CR>
<four blanks><pin> <co-ordinate> ... <CR>
<four blanks><pin> <co-ordinate> ... <CR>
...

or

<CR>**DISCARD:** <<decimal command number>><CR>
<four blanks><pin> <co-ordinate> ... <CR>
<four blanks><pin> <co-ordinate> ... <CR>
...

or

<CR>**RECONNECT:** <<decimal command number>><CR>
<four blanks><pin> <co-ordinate> ... <CR>
<four blanks><pin> <co-ordinate> ... <CR>

...

or

<CR>DELETE: <<decimal command number>>; <net name><CR>
<four blanks><pin> <co-ordinate> ... <CR>
<four blanks><pin> <co-ordinate> ... <CR>
...

UNPLUG and **DISCARD** are commands designed to give the stitchwelding machine unrestricted access to stitchweld pins on which it will need to work. The difference is that **DISCARD** means that the IC should not be replaced after the update, while **UNPLUG** means that the IC will be re-plugged afterwards. **UNPLUG** is actually a list of positions that will be re-plugged after the change, so it may contain board positions that originally were empty. Sorry for the confusion.

RECONNECT is a command to restore the connection from trace to pin that was earlier destroyed by a **DISCONNECT** command. This would normally be done by soldering. The Board routines specify under what conditions the operator can be asked to do this.

DELETE is a command to remove a net that was wired onto the board in an earlier revision.

1.4 Multiwire Net format

1.5 Multiwire Hole format

2. MetaProgram.

First, let us imagine that ROUTE is being invoked to do its simplest task: collect together several .nl files and produce a .wl file, a .bp file, some -*x*.nl files, and a .re file. Processing proceeds as follows:

1. The command line is processed to extract parameters. These include:

    a. the names of the .nl files,
    b. the names of the .wl, .bp, and .re files,
    c. the name of a .br-format file (or concatenation of .br-format files) containing compiled Bcpl Board routines,
    d. whether or not re-work is desired (/R), and
    e. parameters to control wire length minimization.

2. Read in each .nl file. For each <net name>, accumulate a list of all <pin>'s contained in that net.

3. See whether any of the pin assignments in (2) above conflict with so-called trace-wired nets; that is, board-defined nets that are wired by PC-board traces. If so, and if this is permissible for the board type, disconnect the offending pins from their trace-wired nets.

4. All pins connected to trace-wired nets are partitioned into clusters according to the closest pins that are still trace-wired. These clusters are then reassigned to new

nets with names like **VCC1, VDD35,** etc.

5. All nets are routed if this has been requested. For each net this involves a permutation of the net order so as to minimize the total wire length. Additional termination pins may be added to nets in this step.

6. The nets are then sorted into an order intended to optimize the wiring process. For stitchwelding this is currently believed to be:

a) nets with very short arcs first, so other wires will not interfere with them,

b) if two nets have shortest arcs of the same length, wire the shorter net first.

7. All the output files are created from the data structures built up in steps 1-6.

2.1 Automatic terminator assignment.

The ECL logic family does not work properly unless each net contains at least one terminating resistor. Assignment of terminating resistors to nets by hand is a tiresome task, and it is reasonable that the DA system should do it. Particularly for long wires, the driver should be on one end of a wire and a terminating resistor be on the other, or else the driver should be in the middle and terminators on both ends. ROUTE is the only program with enough information about board position and wire routing to be able to do rational terminator assignment.

First, let us expand on step 5 above to explain how termination comes about:

5a. A permutation of the net is chosen to minimize the wire length, subject to the following constraints:

i) the first edge (or cable) pin is constrained to lie at the end of the permutation, or

ii) if there is any ECL output pin in the net, and if exactly one pin in the net is marked as an output pin, and if the net contains no edge or cable pins, then if the resulting net would be no longer than 1.2 times the length of the constrained net, the output pin is constrained to lie at the end of the permutation.

5b. If no instance of the net name ended in the character "!" (signifying that termination is to be ignored) and there is an ECL output pin in the net, and there are no terminating resistors explicitly included in the net, then either one or two terminating resistors will be assigned to the net. Two resistors will be assigned if the net is longer than 4 inches, or if the net contains more than one output pin, or if any output pin is not at the end of the net. One resistor will be assigned otherwise.

5c. If any terminating resistors are to be assigned, they are assigned either at the ends of the nets or between the next-to-end and end pins in the net so as to minimize the increase in wire length (except if an unterminated stub longer than 3 inches would result). Terminators are chosen as close as possible to the end pins. If only one terminator is assigned, it is assigned on the opposite end of the net from the output or edge pin.

2.2 Correction.

Another practical consideration is correction (revision). It should be possible to correct a set of drawings and have the size of the resulting wiring change relate to the size of the change in the drawings. This is done by reading the previous .wl file and only changing the wiring of a net if it differs between the old .wl file and the new one. Unfortunately, one of the least significant physical features of a net is its name, so ROUTE must be able to recognize identical nets as identical even if their names change. If re-work is requested, a new step 4.5 is inserted after the nets are completely specified but before they are routed:

> 4.5 For each net in the old .wl file, determine what new net it is the same as, except for terminating resistors. If it is not the same as any new net, then mark it for deletion in the .ad file. If it is the same as some new net, and if the termination for the old net makes sense for the new net, and if the new net has no explicit termination of its own, then route and terminate the new net exactly as the old one was, and mark it as routed and terminated so that it will not be worked on in step 5 nor output to the .ad file.

3. Internal Data Structures.

3.1 Names and Namees.

The basic organizing concept in ROUTE is the "name". Nets have names, IC types have names, board positions have names, etc. ROUTE maintains a single name data structure, and attached to each name is a list of named objects with that name. The name data structure is a hash table where each bucket is a pointer to a list of **name** blocks that all hash to that bucket. A **name** block looks like this:

```
structure name:
  [ next word     // next name block in bucket list, or nil
  mark word // =-1. Namee list is circular and ends here.
  nameString @string
  ]
```

The **name** block is immediately followed by a namee block, several types of which are described below.

3.1.1 Nets.

One namee is a net. A **net** block looks like this:

```
sturcture net:
  [ next word // to next namee with this name
  flags bit 4
  unused bit 8
  type bit 4  // =net
  pinList word . // pointer to first pin of pin list
  shortestarc word = netnum word = minSperge word
  netlength word
  ]
```

3.1.2 IC instances.

Another namee is an IC instance. The name denotes the board position. An **icinst** block looks like this:

```
sturcture icinst:
  [ next word // to next namee with this name
  type word  // =icinst
  ictype word     // pointer to ictype block for this IC
  pinattribute word
  pin↑1,npins     // each one links to the next pin in
     // the pin list of its net, or nil
  ]
```

Note that one can chain one's way into an **icinst** block at some undetermined index in its pin vector, and then get properly aligned with the **icinst** block by scanning backward until detecting type=icinst, which is an illegal value for pin, pinattribute, and ictype words.

3.1.3 IC types.

Another namee is the IC type. An **ictype** block looks like this:

```
sturcture ictype:
  [ next word // to next namee with this name
  npins bit 12 // same as npins in icinst
  type bit 4  // =ictype
  icclass word     // pointer to IC class containing this type
  outpins↑1,npins bit 1    // one bit per pin, true if pin
     // ever used in any IC instance as output
  ]
```

3.1.4 IC classes.

The final namee is the IC class. An **icclass** block looks like this:

```
structure icclass:
  [ next word // to next namee with this name
  isTraceWired bit 1      // needs termination?
  isConnector bit 1 // is this a terminator IC?
  printUsedList bit 1
  unused bit 9
  type bit 4  // =icclass
  PinOffset word
     // PinOffset(npins, pinNo, lv XOffset, lv YOffset) is
     // a routine that computes the X- and Y- offsets
     // of the given pin from pin 1, in 25-mil units.
     // For a standard 14-pin DIP, pin 1 would result in
     // {0,0}, pin 2 would result in {4,0}, and pin 14 would
     // result in {12,0}.
  PinAttributes word
     // PinAttributes(icinst, pinNo) = attributes, such as
     // isEcl or isTerminator
  ImplicitICNets word
```

```
    // ImplicitICNets(npins, icInstNameString) is
    // a routine that writes
    // nets for such things as IC power and ground onto
    // a file called "implicit.nl"
npins word
    // # pins, overridden by ictype block
... and other fairly esoteric stuff for terminators and trace-wired nets
]
```

3.2 Boards.

A board is a set of Bcpl subroutines dynamically loaded into ROUTE. It is represented as one or more concatenated .BR files. These subroutines are:

FindIndexFromCoord(xPos, yPos, picclass, pPinNo) = index

This routine finds an integer from 1 to maxPins that uniquely represents the board pin at co-ordinates xPos, yPos. It returns 0 if there is no board pin at co-ordinate xPos, yPos. If xPos = yPos = -1, then it returns maxPins+1. In addition, if the board pin is initially connected to some trace-wired net, @picclass is set to the icclass describing that trace-wired net and @pPinNo is set to the number of the pin within the trace-wired net.

DeclareInitialNets(TWBuild, TermBuild, ConnBuild)

This routine declares all the trace-wired nets, terminator sets, and connectors that the board has.

ZeroTablePoint(point) = string

If point=0 then the string name of the board is returned. If point is from 1 to 4 the string name of the proper calibration point is returned. The calibration points should form a rectangle on the board.

LevelTransform(level, x, y, px, py, pName, pPull, pWire) = exists

A given pin is described by different co-ordinates according to whether a board is positioned wiring-side up or component-side up. For each value of level, this subroutine implements a transformation from internal co-ordinates x,y to printing co-ordinates @px,@py. @pName is set to a string giving the name of this level. @pPull is set according to whether components can be removed at this level. @pWire is set according to whether wiring can be done at this level.

FindCoordFromString(s, px, py, vop1, hop1) = {absolute, relative, illegal}

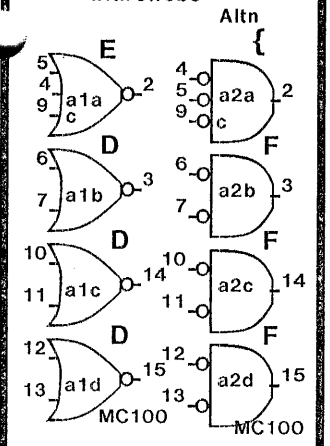
This routine takes the string name of a board location and computes the internal co-ordinates x,y of that position. It then adds vop1 and hop1 as the offsets vertically (the long way) and horizontally (the short way) of some desired pin assuming that pin 1 is installed at x,y. The result is placed in @px,@py. The result is absolute if the pin is legal, and illegal otherwise.

BoardPinCoord(s, icinst, pinNo, px, py) = {true, false}

This routine is normally defined by ROUTE itself (using the board's FindCoordFromString), but the board module can override ROUTE's definition. This routine takes the string name of a board location, a pointer to an IC instance, and a pin number, and computes the internal co-ordinates x,y of that pin. The result is true if the pin position is legal, and false otherwise. This routine would be defined instead of FindCoordFromString if you want to do screwball ball things like put TTL IC's upside-down in Ecl-wired sockets.

MC100

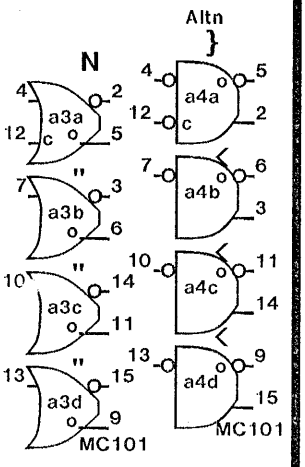
Quad 2-In NOR with strobe



Tpd = 3.3 est

MC101

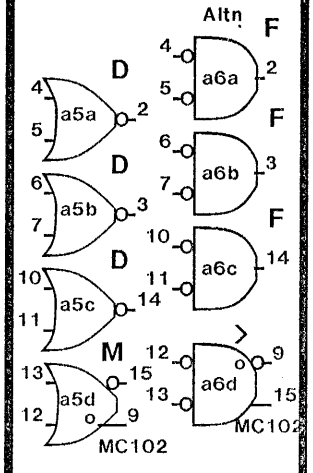
Quad OR/NOR



Tpd = 3.3

MC102

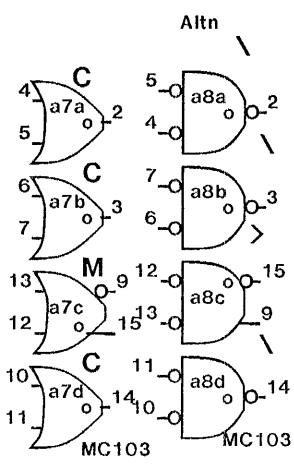
Quad 2-In NOR



Tpd = 3.3

MC103

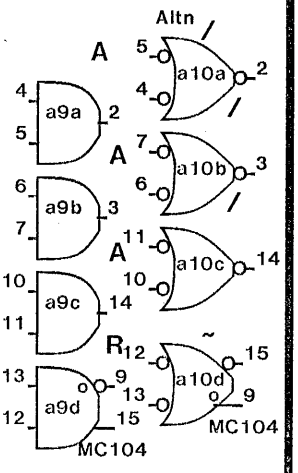
Quad 2-In OR



Tpd = 3.3 est

MC104

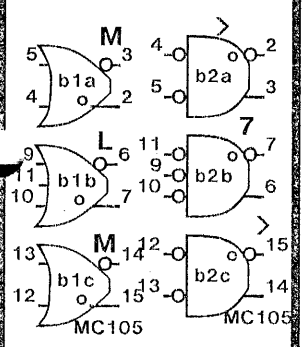
Quad 2-In AND



Tpd = 4.2

MC105

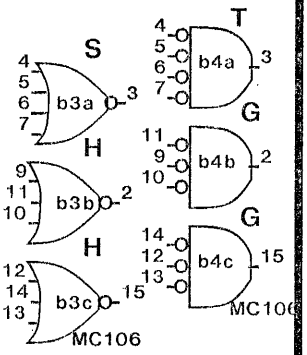
Trip 2-3-2 In OR/NOR



Tpd = 3.3

MC106

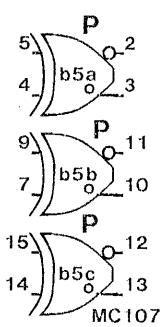
Trip 4-3-3 In NOR



Tpd = 3.3

MC107

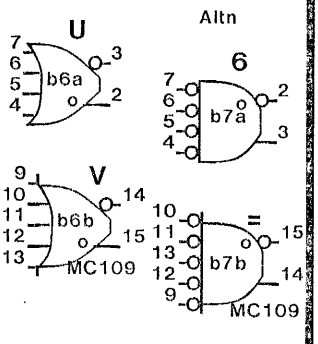
Trip 2-In Ex-OR Ex-NOR



Tpd = 4.0

MC109

Dual 4-5 In OR/NOR

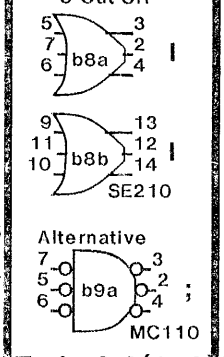


Tpd = 3.3

MC110 MC210 SE210

(Selective MC210)

Dual 3-In 3-Out OR

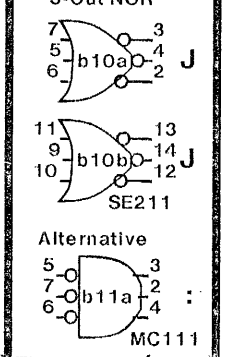


Tpd = 3.8(110)
Tpd = 2.8(210)
Tpd = 1.5(SE) est

MC111 MC211 SE211

(Selective MC211)

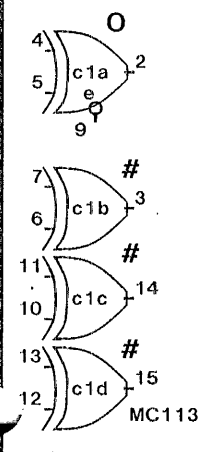
Dual 3-In 3-Out NOR



Tpd = 3.8(111)
Tpd = 2.8(211)
Tpd = 1.5(SE) est

MC113

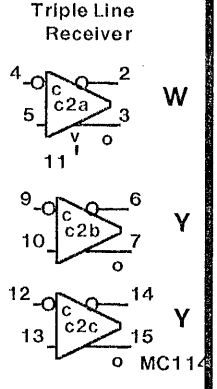
Quad Ex-OR



T(2←4) = 4.5
T(2←9) = 5.1
both EST.

MC114 MC116

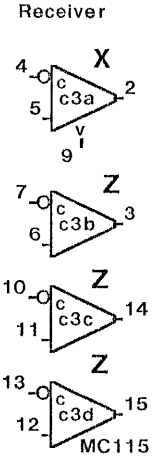
Triple Line Receiver



Tpd = 4.3(114)
Tpd = 3.6(114)
(Diff. Input)
Tpd = 3.3(116)

MC115

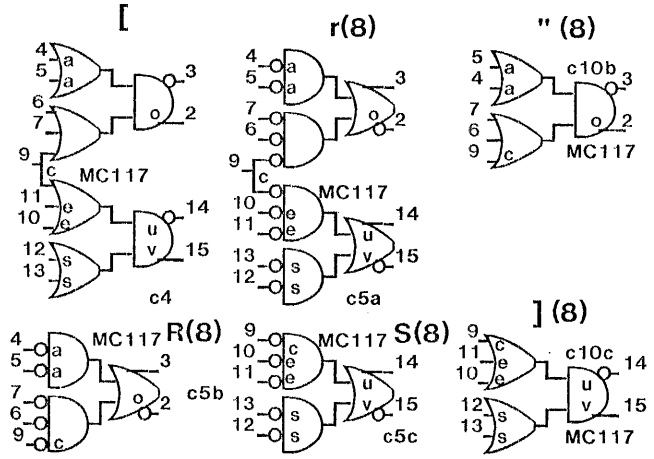
Quad Line Receiver



Tpd = 3.3

MC117

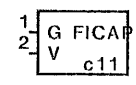
Dual 2-3 In OR-AND/ OR-AND-INVERT



Tpd = 3.8

Connect only one c pin on bitslice, blob the other c pin

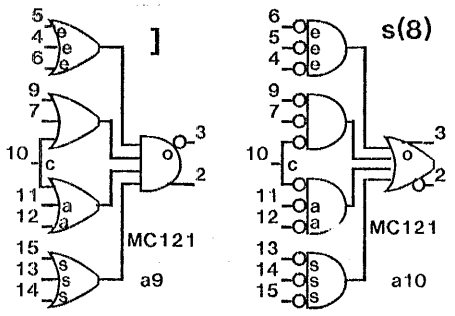
FICAP [(8)



XEROX SDD	Project	ECL SSI Datasheet	File	Designer	Rev	Date	Page
			MC-1.sil	S. T. Chang	B	9/12/78	1

MC121

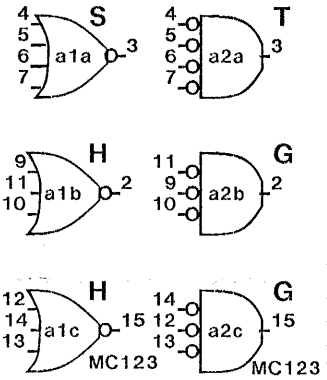
4-wide OR-AND/
OR-AND INVERT



Tpd = 3.8

MC123

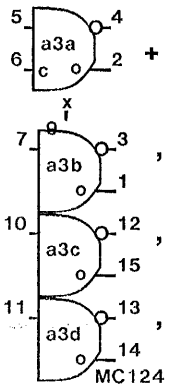
Triple 4-3-3
Bus Driver



Tpd = 3.8

MC124

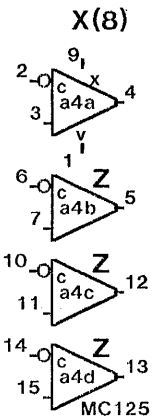
Quad Transl
TTL to ECL



In	Out	Tpd
6 +	1 +	6
7 +	3-	6
6-	1-	6.8
7-	3 +	6.8

MC125

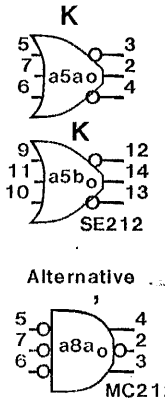
Quad Transl
ECL to TTL



Tpd = 6

MC212
SE212
(Selective MC212)

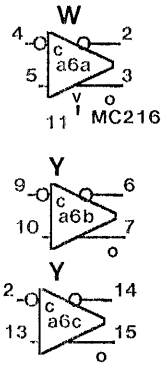
High speed
Dual OR/NOR



Tpd = 2.8(MC212)
Tpd = 1.5(SE212 (est.)

MC216

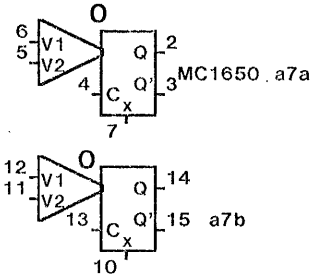
High speed
Line Receiver



Tpd = 2.7(est.)
(Diff. Input)
Tpd = 2.3(est.)

MC1650, MC1651

Dual A/D Comparator



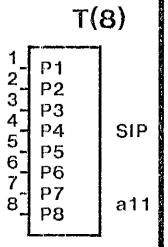
1650 (high impedance inputs)
1651 (low impedance inputs)

C	V1,V2	Qtn+1	Q'tn+1
H	V1>V2	H	L
H	V1<V2	L	H
L	Don'tCare	Qtn	Q'tn

T(2←6) = 5.7 T(2←4) = 5.2

SIP

SIP package



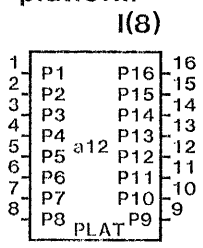
From b to i
for pin # 1 to 8

TERM

automatic assignment terminator

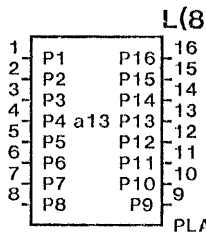
PLAT

Augat platform



From b to q
for pin # 1 to 16

PLAT's

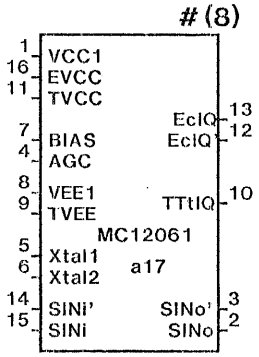


From b to q
for pin # 1 to 16

Platform for trace Cutting

Component name	CUT traces on pins
PLAT1	1
PLAT8	8
PLAT16	16
PLAT18	18
PLAT116	116
PLAT816	816
PLAT1816	1816

MC12061

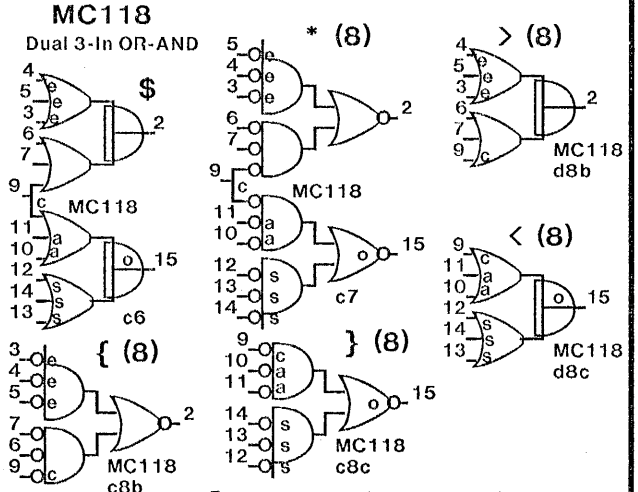


Crystal Oscillator

See page 6-42 of Motorola Semiconductor Library Volumn 4

MC118

Dual 3-In OR-AND

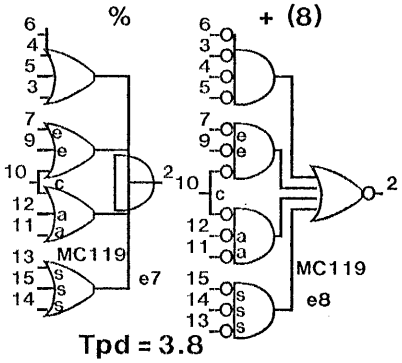


Tpd = 3.8

Connect only one c pin on bitslice, blob the other c pin

MC119

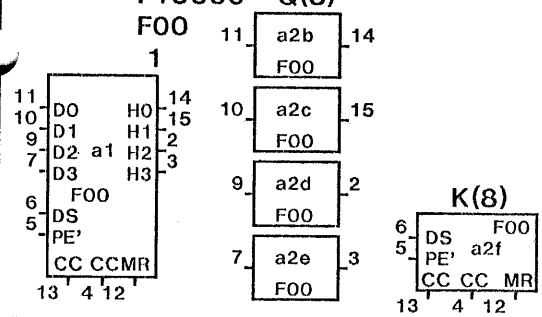
4-Wide 3-3-3



Tpd = 3.8

XEROX SDD	Project	ECL SSI Datasheet	File: MC-2.sil	Designer: S. T. Chang	Rev: C	Date: 9/7/78	Page: 2

F10000 F00



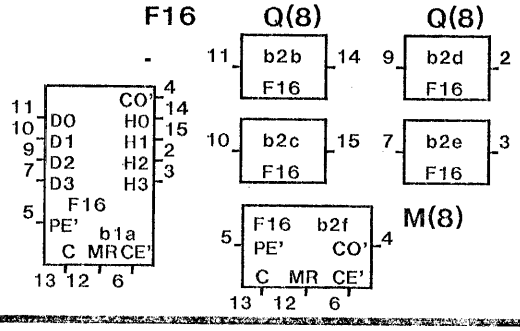
4 bit Shift Register

MR	PE'	CC	CC MODE
H	X	X	X reset - all outputs low
L	L	L	↑↑ parallel load
L	L	↑	L parallel load
L	H	↑↑	L shift left
L	H	L	↑↑ shift left
L	X	⎍	X hold
L	X	X	H hold

Tpd = 5.5
Tw' = 3.8 (clock pulse width)
Ts = 1.7 (data setup)
Th = 1.1 (data hold)
Tse = 6.0 (PE' setup)

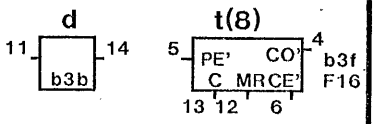
(all times estimates)

F10016 F16



4 bit Binary Counter

CE	PE'	MR	CP	FUNCTION
L	L	L	↑↑	parallel load
H	L	L	↑↑	parallel load
L	H	L	↑↑	count
H	H	L	↑↑	hold
X	X	L	⎍	masters open, slaves hold
X	X	H	X	reset

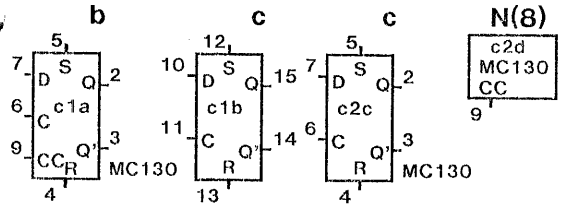


Tpd = 5.5
Tw = 3.5 (clock pulse width)
Ts = 2.2 (data setup)
Th = 1.1 (data hold)
Tse = 2.8 (PE' and CE' setup)

MC130

Dual Latch

obsolete: A(8) c3d CC MC130



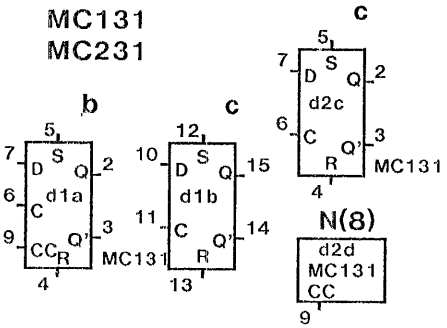
D	CC	C	S	R	Qn+1	FUNCTION
L	↑	↑	X	X	L	Q follows D
H	↑	↑	X	X	H	Q follows D
X	H	X	L	L	Qn	hold
X	X	H	L	L	Qn	hold
X	H	X	H	L	H	set
X	X	H	H	L	H	set
X	H	X	L	H	L	reset
X	X	H	L	H	L	reset
X	H	X	H	H		indeterminate

Tpd = 4.1
Ts = 2.8 est
Th = 1.7 est

SE231 (Selective MC231) MC131 MC231

Dual D type Master-Slave Flip-flop

X X X H H indeterminate



D	CC	C	S	R	Qn+1	FUNCTION
X	↓	L	L	L	Qn	hold
L	↑	↑	L	L	L	memory
L	↑↑	L	L	L	L	memory
H	↑	↑	L	L	H	memory
H	↑↑	L	L	L	H	memory
X	X	X	H	L	H	set
X	X	X	L	H	L	reset
X	X	X	H	H		indeterminate
X	H	X	H	H		indeterminate

obsolete: A(8) d3d CC MC131

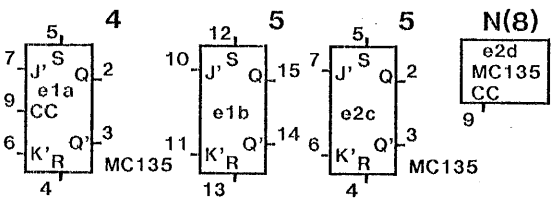
MC131	MC231	SE231
Tpd = 5.0	Tpd = 3.7	Tpd = 2.5
Ts = 2.8	Ts = 1.1	Ts = 1.1
Th = 1.7	Th = 0.8	Th = 0.8

(all times estimates)

MC135

Dual J-K Master Slave Flip-Flop

obsolete: A(8) e3d CC MC135



RS Truth Table

R	S	Qn+1
L	L	Qn
L	H	H
H	L	L
H	H	indet

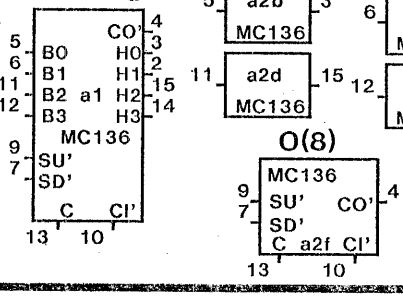
Clocked JK Truth Table

J'	K'	Qn+1
L	L	Qn'
H	L	L
L	H	H
H	H	Qn

clock = ↑↑ on CC

Tpd = 4.6
Trs = 5.2 (R, S to Q)
Ts = 2.8 est
Th = 1.7 est

MC136



Hexadecimal Counter

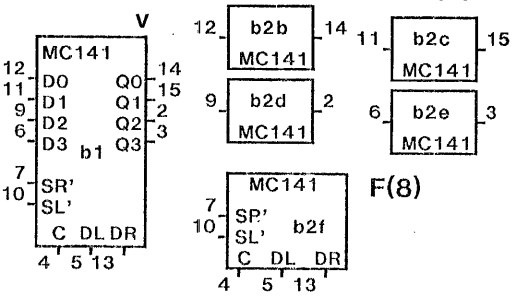
SU'	SD'	B0	B1	B2	B3	CI'	C	H0	H1	H2	H3	CO'	
L	L	H	H	L	L	X	↑↑	H	H	L	L	L	Load
L	H	X	X	X	X	L	↑↑	H	H	L	H	H	Count Up
L	H	X	X	X	X	L	↑↑	H	H	H	L	H	
L	H	X	X	X	X	L	↑↑	H	H	H	H	L	
L	H	X	X	X	X	H	L	H	H	H	H	H	Hold
L	H	X	X	X	X	H	↑↑	H	H	H	H	H	
H	H	X	X	X	X	X	↑↑	H	H	H	H	H	
L	L	L	L	H	H	X	↑↑	L	L	H	H	L	Load
H	L	X	X	X	X	L	↑↑	L	L	H	L	H	
H	L	X	X	X	X	L	↑↑	L	L	L	H	H	Count Down
H	L	X	X	X	X	L	↑↑	L	L	L	L	L	
H	L	X	X	X	X	L	↑↑	H	H	H	H	H	

Inputs asumed to vary In sequence from top to bottom

Tpd = 11.5
Tsd = 3.9(data setup)
Tss = 8.3(select setup)
Tsc = 4.1 (CI' setup)
Thc = 3.4 (CI' ld)

(all times estimates)

MC141



4 bit Shift Register

SR'	SL'	MODE	Qi(n+1)
L	L	parallel load	Di(n)
L	H	shift right	Qi-1(n)
H	L	shift left	Qi+1(n)
H	H	hold	Qi(n)

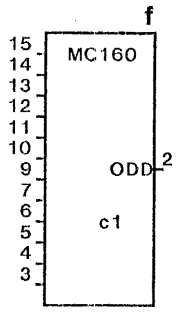
ALL OUTPUTS EXIST AFTER CLOCK

clock = ↑↑ on C

Tpd = 4.2
Tsd = 2.8(data setup)
Tss = 5.5(select setup)
Thd = 1.7(data hold)
Ths = 1.1 (select hold)

(all times estimates)

MC160



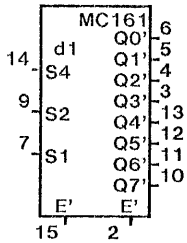
12-bit Parity Generator/Checker

Sum of high inputs	=	Output
Even		Low
Odd		High

Tpd = 8.3 est

MC161

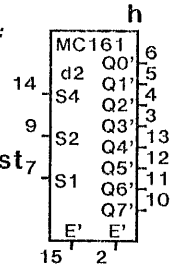
8(8)



Binary 1 Of 8 decoder - low true outputs

E'	E'	S1	S2	S4	Q0'	Q1'	Q2'	Q3'	Q4'	Q5'	Q6'	Q7'
L	L	L	L	L	L	H	H	H	H	H	H	H
L	L	H	L	L	H	L	H	H	H	H	H	H
L	L	L	H	L	H	H	L	H	H	H	H	H
L	L	H	H	L	H	H	H	L	H	H	H	H
L	L	L	L	H	H	H	H	H	L	H	H	H
L	L	H	L	H	H	H	H	H	H	L	H	H
L	L	L	H	H	H	H	H	H	H	H	L	H
L	L	H	H	H	H	H	H	H	H	H	H	L
H	X	X	X	X	H	H	H	H	H	H	H	H
X	H	X	X	X	H	H	H	H	H	H	H	H

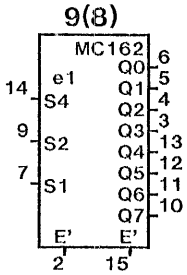
obsolete:



Tpd = 6.6 est

MC162

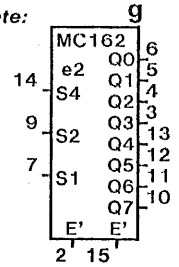
9(8)



Binary 1 Of 8 decoder - high true outputs

E'	E'	S1	S2	S4	Q0	Q1	Q2	Q3	Q4	Q5	Q6	Q7
L	L	L	L	L	H	L	L	L	L	L	L	L
L	L	H	L	L	L	H	L	L	L	L	L	L
L	L	L	H	L	L	L	H	L	L	L	L	L
L	L	H	H	L	L	L	L	H	L	L	L	L
L	L	L	L	H	L	L	L	L	H	L	L	L
L	L	H	L	H	L	L	L	L	L	H	L	L
L	L	L	H	H	L	L	L	L	L	L	H	L
L	L	H	H	H	L	L	L	L	L	L	L	H
H	X	X	X	X	L	L	L	L	L	L	L	L
X	H	X	X	X	L	L	L	L	L	L	L	L

obsolete:

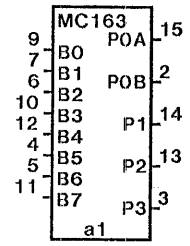


Tpd = 6.4

XEROX SDD	Project	File	Designer	Rev	Date	Page
	Datasheet ECL MSI datasheet	MC-4.sil	K. Pier	B	2/3/77	4

MC163

n



Error Detection/Correction Circuit

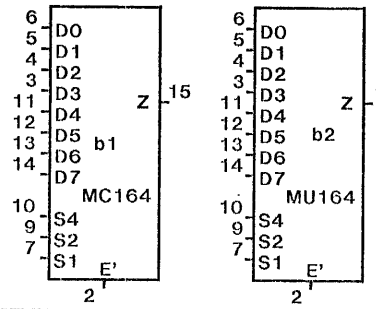
P0A = B1, B2, B4, B7
P0B = B0, B3, B5, B6
P1 = B1, B3, B5, B7
P2 = B2, B3, B6, B7
P3 = B4, B5, B6, B7

(, denotes EXOR)

Tpd = 7.5 est

MC164 / MU164

l



@(8)

Midas multiplexer ONLY

8 line Multiplexer

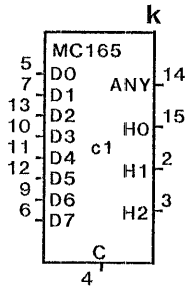
E'	S1	S2	S4	Z
L	L	L	L	D0
L	H	L	L	D1
L	L	H	L	D2
L	H	H	L	D3
L	L	L	H	D4
L	H	L	H	D5
L	L	H	H	D6
L	H	H	H	D7
H	X	X	X	L

Tpdd = 4.8 (data)
Tpds = 6.5 (select)
Tpde = 3.1 (enable)

(all estimates)

MC165

k



8 input Priority Encoder with latch

D0	D1	D2	D3	D4	D5	D6	D7	ANY	H0	H1	H2
H	X	X	X	X	X	X	X	H	L	L	L
L	H	X	X	X	X	X	X	H	L	L	H
L	L	H	X	X	X	X	X	H	L	H	L
L	L	L	H	X	X	X	X	H	L	H	H
L	L	L	L	H	X	X	X	H	H	L	L
L	L	L	L	L	H	X	X	H	H	L	H
L	L	L	L	L	L	H	X	H	H	H	L
L	L	L	L	L	L	L	H	H	H	H	H
L	L	L	L	L	L	L	L	L	L	L	L

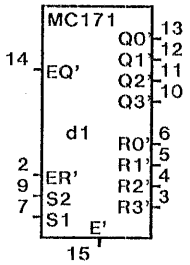
Outputs held when C is high

Tpdd = 16.5 (data)
Tpdc = 5.3 (clock)
Ts = 5.1
Th = 0.0

(all estimates)

MC171

j



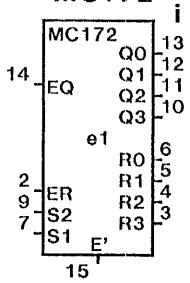
Dual Binary to 1-of-4 Decoder - low true outputs

E'	EQ'	ER'	S1	S2	Q0'	Q1'	Q2'	Q3'	R0'	R1'	R2'	R3'
L	L	L	L	L	L	H	H	H	L	H	H	H
L	L	L	H	L	H	L	H	H	H	L	H	H
L	L	L	L	H	H	H	L	H	H	H	L	H
L	L	L	H	H	H	H	H	L	H	H	H	L
L	L	H	L	L	L	H	H	H	H	H	H	H
L	H	L	L	L	H	H	H	H	L	H	H	H
H	X	X	X	X	H	H	H	H	H	H	H	H

Tpd = 6.4

MC172

i



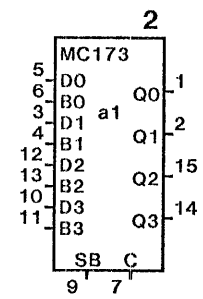
Dual Binary to 1-Of-4 Decoder - high true outputs

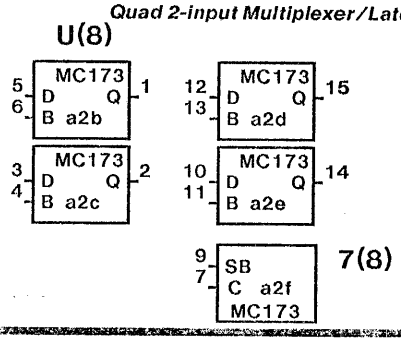
E'	EQ	ER	S1	S2	Q0	Q1	Q2	Q3	R0	R1	R2	R3
L	H	H	L	L	H	L	L	L	H	L	L	L
L	H	H	H	L	L	H	L	L	L	H	L	L
L	H	H	L	H	L	L	H	L	L	L	H	L
L	H	H	H	H	L	L	L	H	L	L	L	H
L	L	H	L	L	L	L	L	L	H	L	L	L
L	H	L	L	L	H	L	L	L	L	L	L	L
H	X	X	X	X	L	L	L	L	L	L	L	L

Tpd = 6.4

MC173

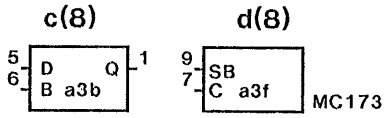
Quad 2-input Multiplexer/Latch





SB	C	Qi(n+1)
H	L	Bi(n)
L	L	Di(n)
X	H	Qi(n)

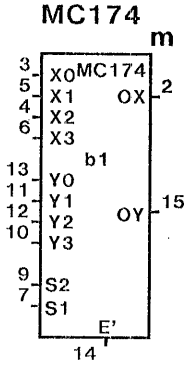
obsolete:

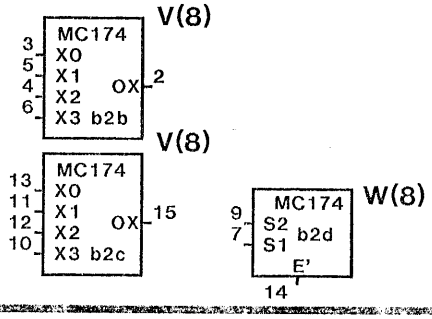


Tpdd = 5.3 (data)
Tpdc = 6.8 (clock)
Tpds = 6.7 (select)
Thd = 2.8 (hold data)
Ths = 1.7 (hold select)

MC174

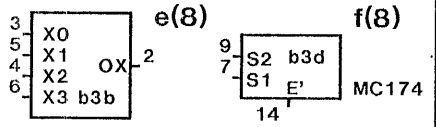
Dual 4 to 1 Multiplexer





E'	S1	S2	OX	OY
H	X	X	L	L
L	L	L	X0	Y0
L	H	L	X1	Y1
L	L	H	X2	Y2
L	H	H	X3	Y3

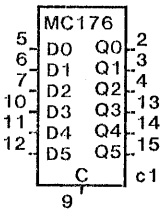
obsolete:

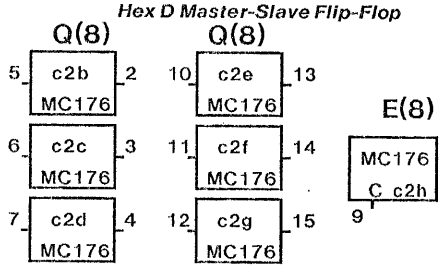


Tpdd = 4.8(data)
Tpde = 3.2(enable)
Tpds = 6.4(select)

MC176

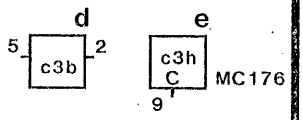
Hex D Master-Slave Flip-Flop





C	D	Qi(n+1)
L	X	Qi(n)
↑	L	L
↑	H	H

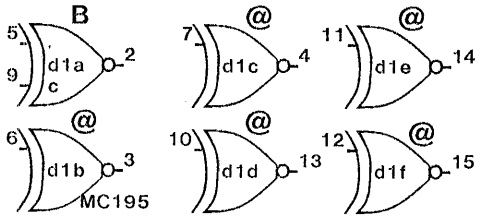
obsolete:

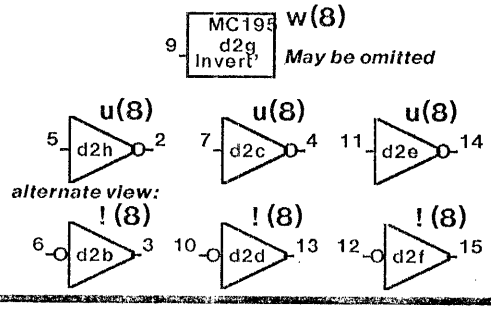


Tpd = 5.0
Ts . = 2.8 est
Th = 1.7 est

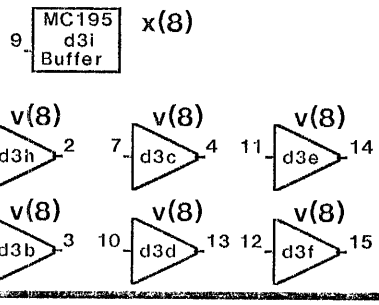
MC195

Hex Inverter/Buffer



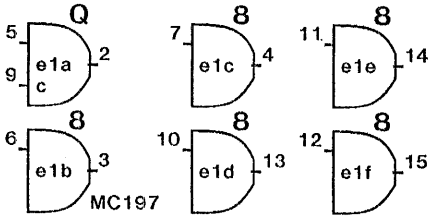


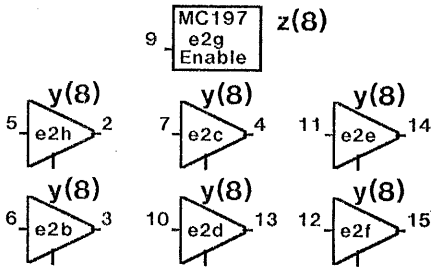
Tpd = 4.2 est



MC197

Hex AND gate

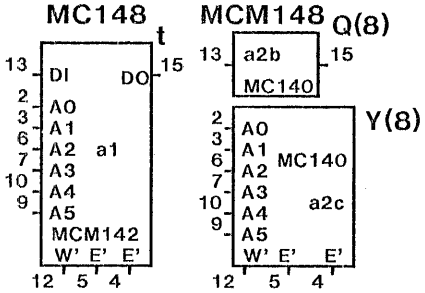




Tpdd = 4.2 (data) est
Tpde = 5.3 (enable) est

MC140 MCM140
MC142 MCM142
MC148 MCM148

64 Bit Random Access Memory



Tace = 14(E' access)
Taad = 11(Ai access, MCM142)
Taad = 17(Ai access, MCM140,148)
Tw' = 11 (write pulse width)
Te' = 15 (enable pulse width)

(MCM140 drives 90 ohm loads)

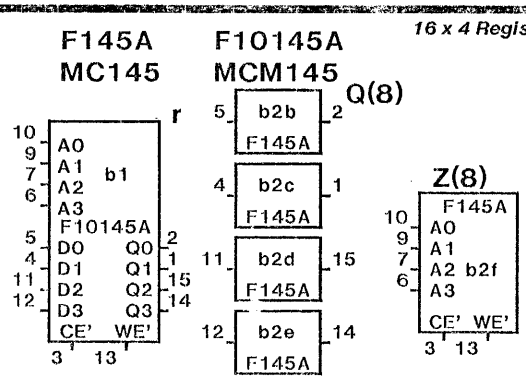
Tsd = 0 (setup times)
Tse = 3.5
Tsa = 5.5

Thd = 3.5 (hold times)
The = 0
Tha = 3.5

(all times estimates)

F145A F10145A
MC145 MCM145

16 x 4 Register File



F10145:
Tace = 6.6(E' access)
Taad = 9.9(Ai access)
Tw' = 4.4(write pulse width)
Te' = 4.4 (enable pulse width)
Twr. = 6.6 (ce,we output recover)

Tsd = 5.0 Thd = -1.0
Tse = 5.0 The = 0.5
Tsa = 3.9 Tha = 1.0

obsolete: a(8) b(8) b3b b3f MCM145

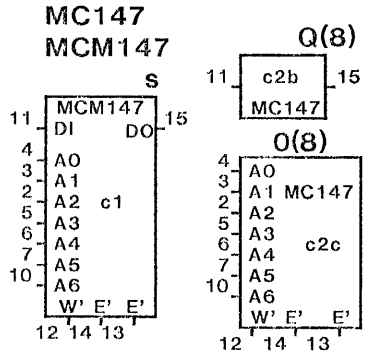
MCM145:
Tace = 10(E' access)
Taad = 15(Ai access)
Tw' = 11 (write pulse width)
Te' = 11 (enable pulse width)

Tsd, Tse relative to end of WE'

Tsd = 0 Thd = 5.0
Tse = 5.0 The = 5.0 all est
Tsa = 5.0 Tha = 5.0
Twr = 11.0

MC147
MCM147

128 x 1 bit Random Access Memory



Tace = 8.8(E' access)
Taad = 14(Ai access)
Tw' = 8.8(write pulse width)
Te' = 11 (enable pulse width)

ALL UNUSED INPUTS MUST BE TIED TO VEE

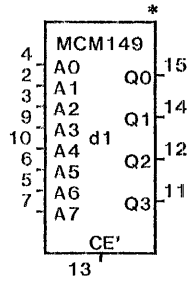
Tsd = 1.0 (setup times)
Tse = 1.0
Tsa = 4.0

Thd = 1.0 (hold times)
The = 1.0
Tha = 3.0

(all times estimates)

MC149 MCM149
MC150 MCM150

256 x 4 bit Programmable ROM

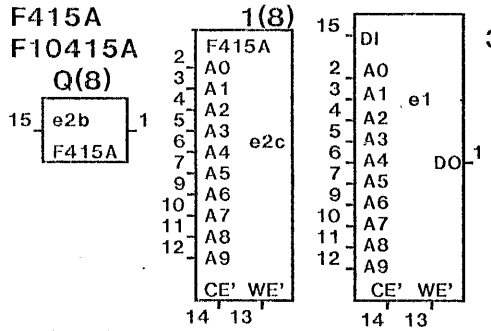


MCM149	MCM150
Tace = 13	Tace = 10.5
Taad = 32	Taad = 30

(all times estimates)

F415A
F10415A

1024 x 1 bit Random Access Memory



Tace = 10(E' access)
Taad = 35(Ai access) *
Tw' = 25 (write pulse width)
Te' = 35 (enable pulse width)

* a 25ns Taad, 20 ns. Tw' part is available

(all times estimates)

Tsd = 5 (setup times)
Tse = 5
Tsa = 8

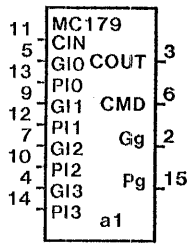
Thd = 5 (hold times)
The = 5
Tha = 2

XEROX SDD	Project: Datasheet	ECL MSI datasheet	File: MC-7.sil	Designer: K. Pier	Rev: B	Date: 2/14/77	Page: 7

MC179

Look Ahead Carry Block

o



Tpd = 6 est

$Pg = P0 + P1 + P2 + P3$

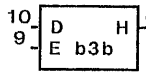
$Gg = (G3 + P2 + P1 + P0)(G2 + P1 + P0)(G1 + P0)G0$

$CMD = (CIN + P3 + P2)(G3 + P2)G2$

$COUT = (CIN + P0 + P1 + P2 + P3)(G3 + P2 + P1 + P0)(G2 + P1 + P0)(G1 + P0)G0$

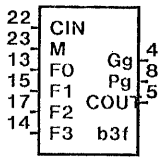
obsolete:

g(8)



MC181

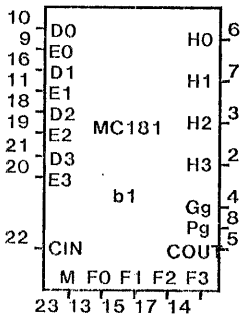
h(8)



4 bit ALU/Function Generator

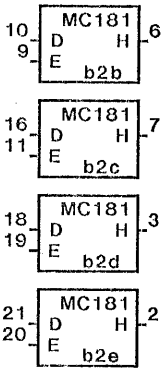
MC181

p

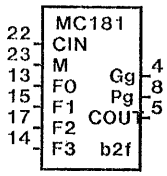


				LOGIC FUNCTIONS M is HIGH	*ARITHMETIC FUNCTIONS M is LOW CIN is LOW*
F0	*F1*	*F2*	*F3*	*H*	*H*
L	L	L	L	D'	D PLUS 0
L	L	L	H	D' + E'	D PLUS (D AND E')
L	L	H	L	D' + E	D PLUS (D AND E)
L	L	H	H	all 1	D TIMES 2
L	H	L	L	D' AND E'	(D + E) PLUS 0
L	H	L	H	E'	(D + E) PLUS (D AND E')
L	H	H	L	D XNOR E	D PLUS E
L	H	H	H	D + E'	D PLUS (D + E)
H	L	L	L	D' AND E	(D + E') PLUS 0
H	L	L	H	D XOR E	D MINUS E MINUS 1
H	L	H	L	E	(D + E') PLUS (D AND E)
H	L	H	H	D + E	D PLUS (D + E')
H	H	L	L	all 0	MINUS 1 (2s complement)
H	H	L	H	D AND E'	(D AND E') MINUS 1
H	H	H	L	D AND E	(D AND E) MINUS 1
H	H	H	H	D	D MINUS 1

2(8)



3(8)

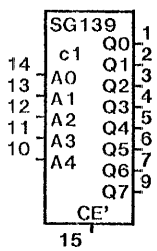


Tpd = 10.8 (max, all functions and carries)

SG10139

?

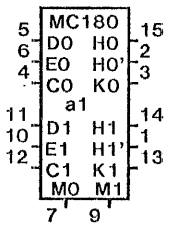
32 x 8 Programmable ROM



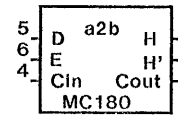
Tace = 17.5

Taad = 25 est

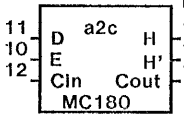
MC180



I(8)



I(8)



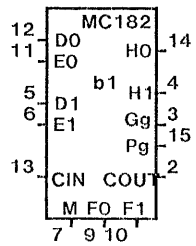
G(8)

Dual 1 bit Adder/Subtractor

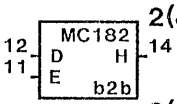
M0	M1	H
H	H	D PLUS E PLUS CARRY
H	L	D MINUS E MINUS CARRY
L	H	E MINUS D MINUS CARRY
L	L	0 MINUS D MINUS E MINUS CARRY

Tpd = 5.8 (all functions)

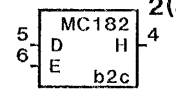
MC182



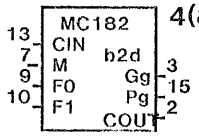
2(8)



2(8)



4(8)



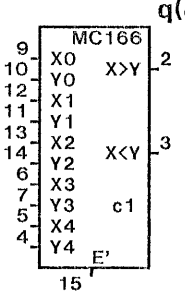
2 bit ALU/Function Generator

F0	F1	Logic Function M is HIGH H	Arithmetic Function M is LOW H
L	L	D XNOR E	D PLUS E PLUS CARRY
L	H	D XOR E	D' PLUS E PLUS CARRY
H	L	D AND E	D PLUS E' PLUS CARRY
H	H	D OR E	D TIMES 2

Tpd = 10.5 est

MC166

q(8)



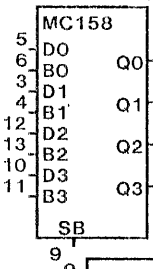
5 bit magnitude comparator

E'	X	Y	X<Y	X>Y
H	x	x	L	L
L	X = Y		L	L
L	X > Y		L	H
L	X < Y		H	L

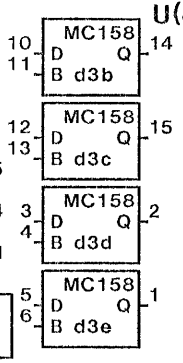
Tpdd = 9.0 est
Tpde = 3.8 est

MC158

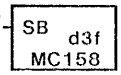
B(8)



U(8)



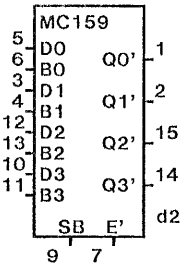
H(8)



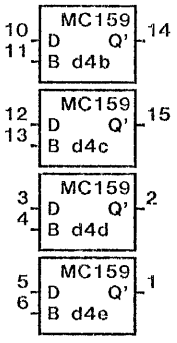
MC159

Quad 2-to-1 multiplexers

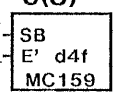
C(8)



6(8)



5(8)



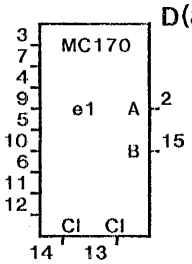
E'	Di	Bi	SB	QI	QI'
H	x	x	x		L
L	L	X	L	L	H
L	H	X	L	H	L
L	X	L	H	L	H
L	X	H	H	H	L
159 only				158 only	159 only

MC158	MC159
Tpdd = 3.6	Tpdd = 3.6
Tpds = 5.0	Tpds = 5.0
	Tpde = 5.0

all times est

MC170

D(8)



9 bit parity circuit with
2 carry inputs

SUM OF PINS 3-12	PIN 2	PIN 15
EVEN	L	?
ODD	H	?
SUM OF ALL INPUTS		
EVEN	?	L
ODD	?	H

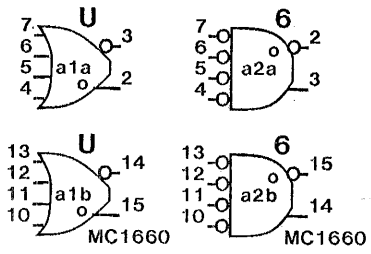
TpdA = 6.6
TpdB = 9.9
Tpdcarry = 3.3

all times est

XEROX SDD	Project	File	Designer	Rev	Date	Page
	Datasheet ECL MSI datasheet	MC-9.sil	K. Pier	E	2/3/77	9

MC1660

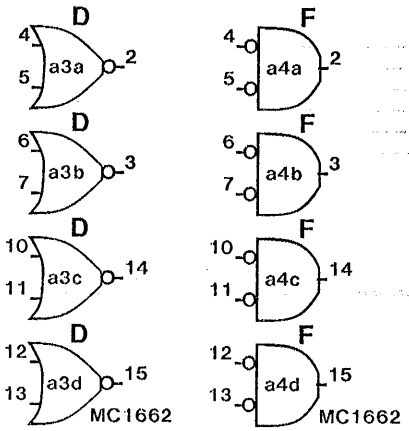
Dual 4 input OR/NOR gate



Tpd = 1.9

MC1662

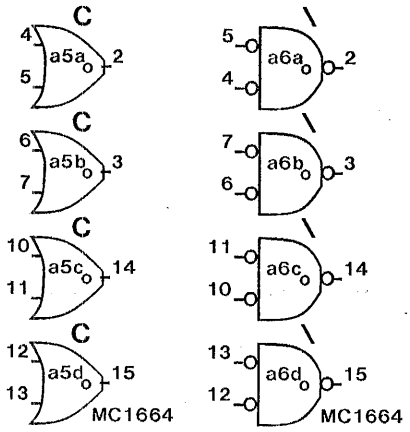
Quad 2 input NOR gate



Tpd = 1.9

MC1664

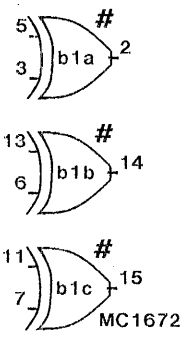
Quad 2 input OR gate



Tpd = 1.9

MC1672

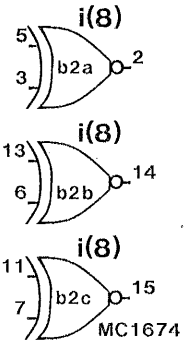
Triple 2 input EXOR gate



Tpd = 2.8

MC1674

Triple 2 inpuEXNOR gate

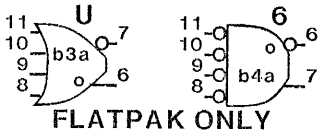


Tpd = 2.8

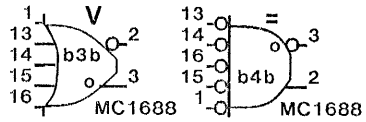
MC1688

Dual 4/5 input OR/NOR gate

FLATPAK ONLY



FLATPAK ONLY

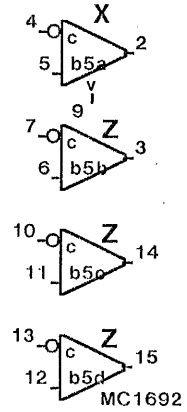


FLATPAK ONLY

Tpd = 1.2 est

MC1692

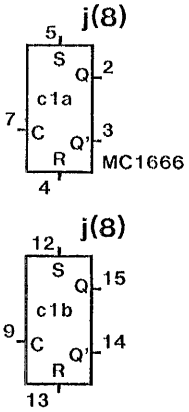
Quad Line Receiver



Tpd = 1.9

MC1666

Dual Clocked R-S Flip-Flop

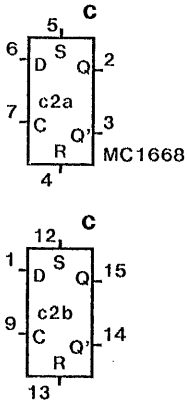


Tpd = 2.8

MC1668

Dual Clocked Latch

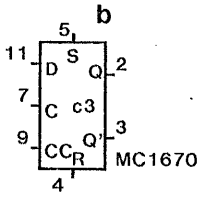
(Q follows D when C is HIGH)



Tpd = 2.8

MC1670

Master-Slave D type Flip-Flop



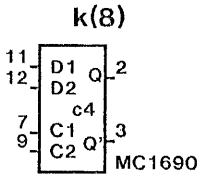
Tpd = 2.9

Tsetup = 0.5 est

Thold = 0.5 est

MC1690

Very Fast D type Flip-Flop



Tpd = 2.3 est

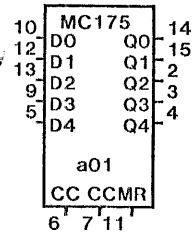
Tsetup = 0.5 est

Thold = 0.5 est

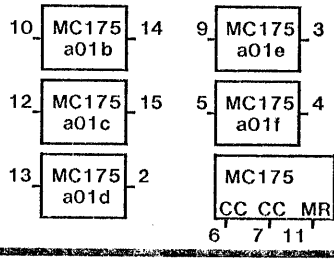
XEROX SDD	Project	File	Designer	Rev	Date	Page
	Datasheet MECL III DATASHEET	MC-10.sil	K. Pier	B	9/10/76	10 end

MC175 *Quint Latch*

((8)



Q (8)

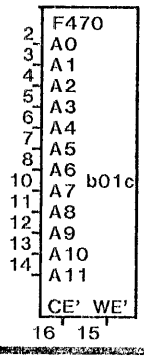


D	CC	CC	MR	Q(n+1)
L	L	L	L	L
H	L	L	L	H
X	H	X	L	Q(n)
X	X	H	L	Q(n)
X	H	X	H	L
X	X	H	H	L

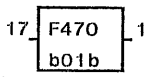
Tpdd	= 3.6	(data)
Tpdc	= 4.4	(clock)
Tpdr	= 4.2	(reset)
Ts	= 3.7	(setup)
Th	= 2.2	(hold)

F470 *4K x 1 random access memory*

$ (8)



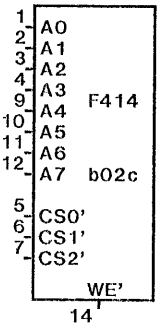
Q (8)



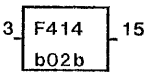
Taad	= 35	A access
Tace	= 15	CE access
Tw'	= 25	write pulse
Tsd	= 5	setup data
Thd	= 5	hold data
Tsa	= 10	setup addr
Tha	= 5	hold addr
Tsce	= 5	setup CE
Thce	= 5	hold CE
Twd	= 15	write disable
Twr	= 20	write recover
Tcer	= 15	CE recover

F414 *256 x 1 random access memory*

~ (8)



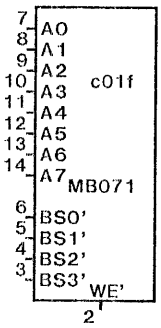
Q (8)



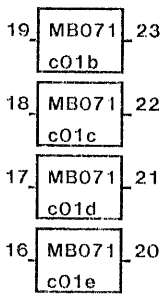
Taad	= 10	A access
Tace	= 6	CE access
Tw'	= 7	write pulse
Tsd	= 1	setup data
Thd	= 2	hold data
Tsa	= 1	setup addr
Tha	= 2	hold addr
Tsce	= 1	setup CE
Thce	= 2	hold CE
Twd	= 8	write disable
Twr	= 10	write recover
Tcer	= 6	CE recover

MB071 *256 x 4 random access memory* *QIT package + platform*

% (8)



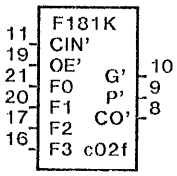
Q (8)



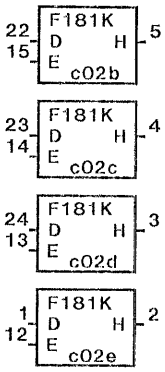
Taad	= 10.0	A access
Tabs	= 4.5	BS access
Tw'	= 8.0	write pulse
Tsd	= 2.0	setup data
Thd	= 2.0	hold data
Tsa	= 2.0	setup addr
Tha	= 2.0	hold addr
Tsbs	= 2.0	setup BS
Thbs	= 2.0	hold BS
Twd	= 5.0	write disable
Twr	= 9.0	write recover
Tbsr	= 4.5	BS recover

F181K *100K series 4 bit ALU 0.4 wide DIP package*

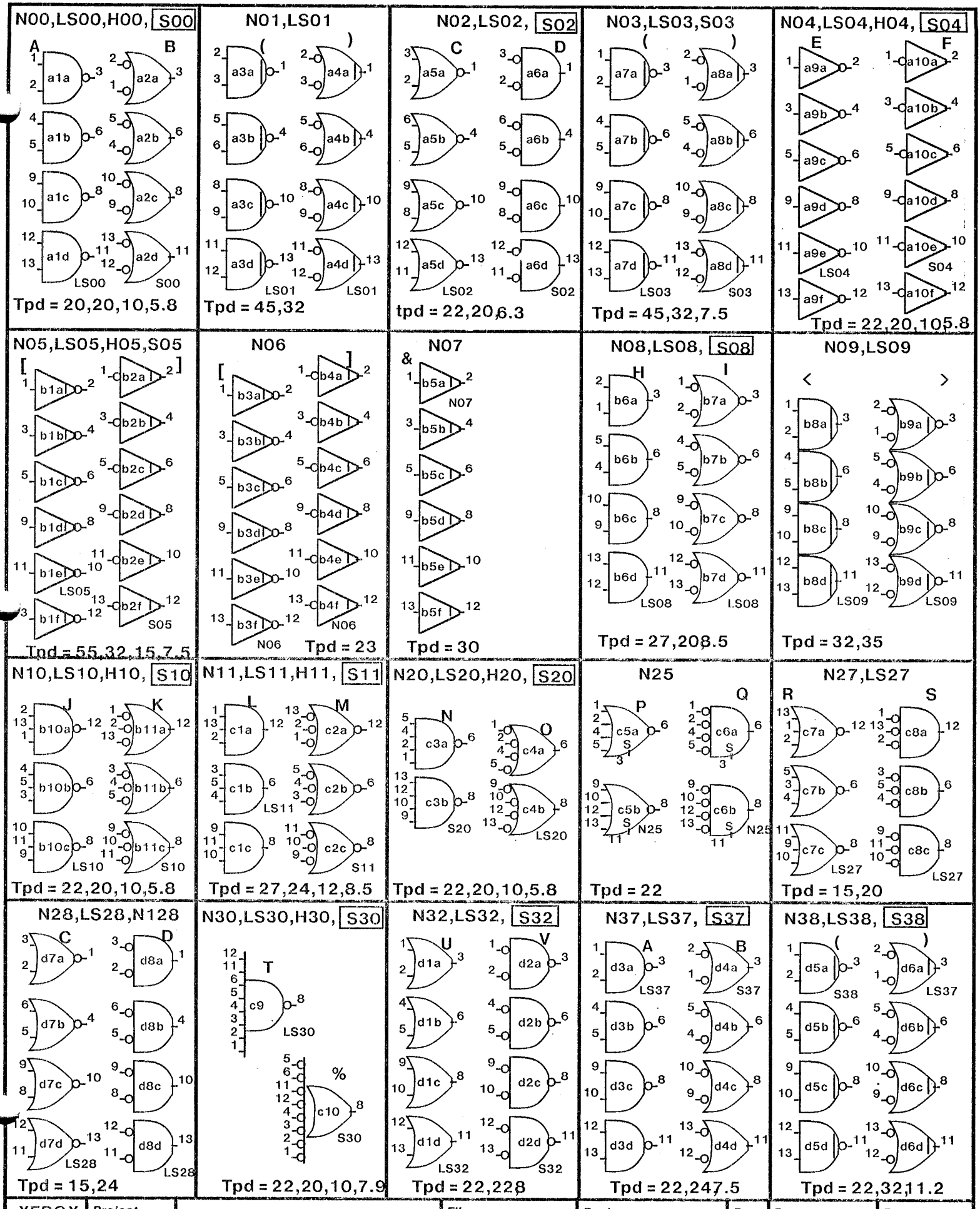
& (8)



2 (8)



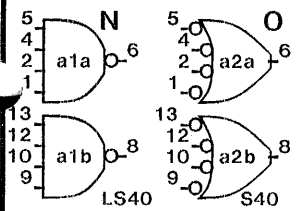
Tdh	= 6.5	D to H
Teh	= 6.5	E to H
Tfh	= 6.5	F to H
Tcinh	= 4.7	CIN to H
Tcio	= 3.6	CIN to CO
Tdpg	= 4.1	D to P,G
Tepg	= 4.1	E to P,G
Tfpg	= 4.1	F to P,G
Tdco	= 5.4	D to CO
Teco	= 5.4	E to CO
Tfco	= 5.4	F to CO
Toeh	= 2.1	OE to H



XEROX SDD	Project NSIL	TTL SSI Datasheet	File TTL1.sil	Designer Rosen	Rev B	Date 7/29/79	Page 1

N40,LS40,H40, S40

NS140

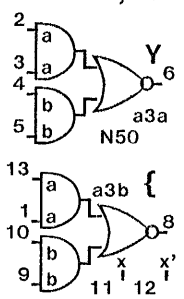


S140 drives 50 ohm lines

Tpd = 22,24,12,7.5

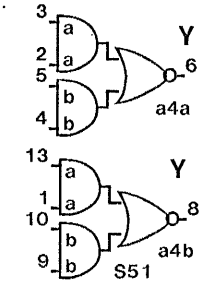
S140 Tpd = 6.5

N50,H50



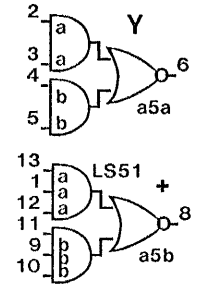
Tpd = 30,15

N51,H51, S51



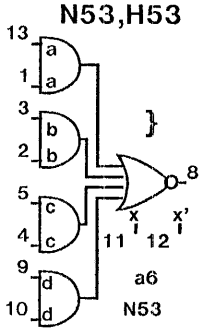
Tpd = 22,11,6.3

LS51



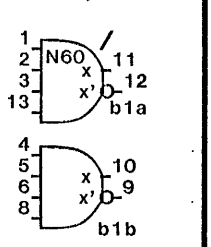
Tpd = 20

N53,H53

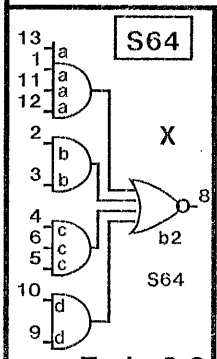


Tpd = 30,15

N60,H60

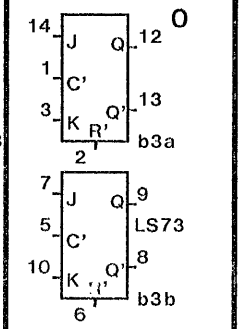


S64



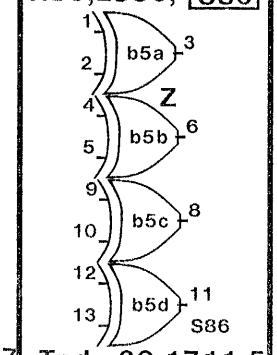
Tpd = 6.3

N73,LS73,H73



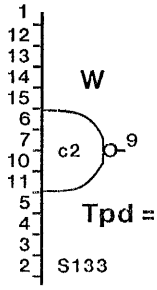
Tpd = 40,30,27

N86,LS86, S86



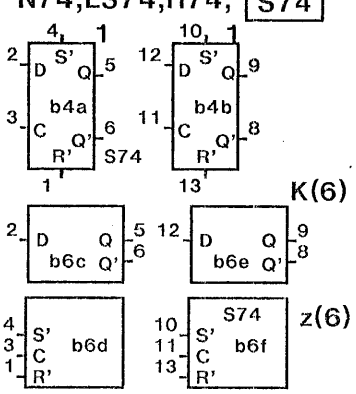
Tpd = 30,17,11.5

S133



Tpd = 8

N74,LS74,H74, S74

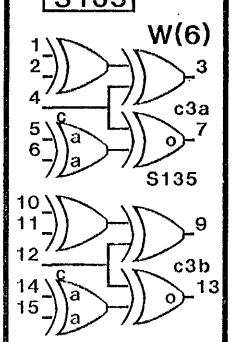


Tpd = 40,40,30,15

S74:T(5←3) = 10 T(5←4) = 9

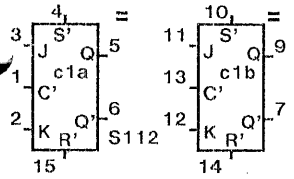
T(5←1) = 15 DataSetup = 4

S135



Tpd = 17

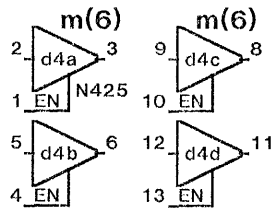
S112



T(5/6←1/4/15) =8

DataSetup = 4

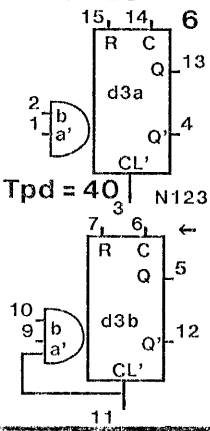
N125, N425



T(3←2) = 18

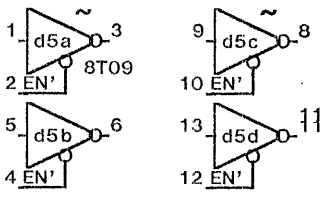
T(3←1) = 25

N123



Tpd = 40

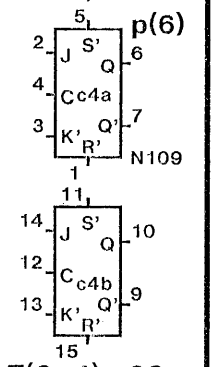
8T09



T(3←1) = 12

T(3←2) = 16.5

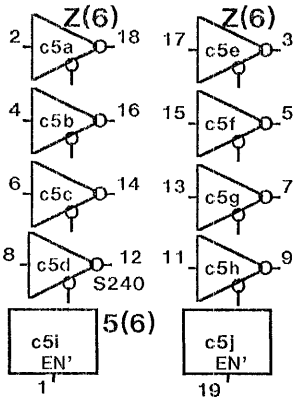
N109,LS109



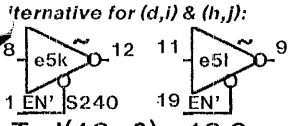
T(6←4) = 28

T(7←5) = 35

LS240, S240



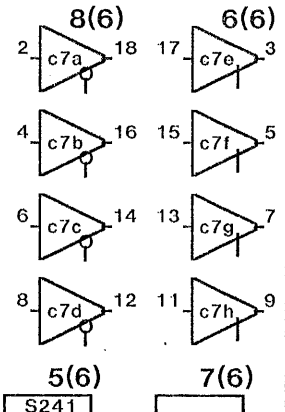
Alternative for (d,i) & (h,j):



Tpd(18←2) = 18,8

Tpd(18←1) = 30,17

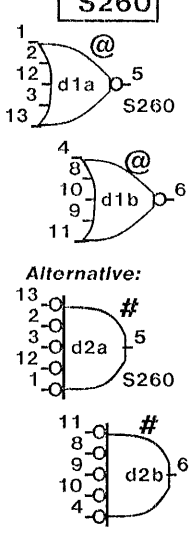
S241



Tpd(18←2) = 10.5

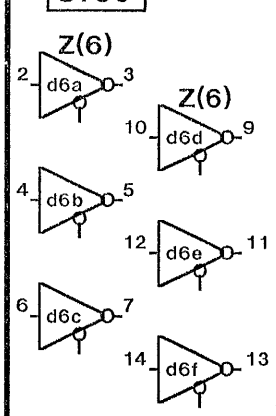
Tpd(18←1) = 17

S260



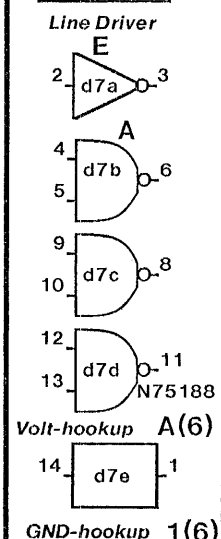
Tpd = 6.8

8T96



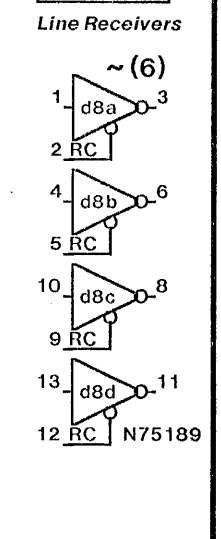
N75188

Line Driver



N75189

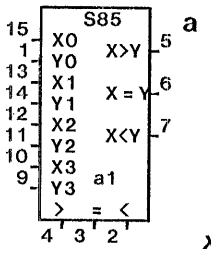
Line Receivers



XEROX SDD	Project NSIL	TTL SSI Datasheet	File TTL2.sil	Designer Chang	Rev A	Date 7/29/79	Page 2

N85, S85

4 Bits Comparator



X Y	>	=	<	X>Y	X = Y	X<Y
X>Y	X	X	X	H	L	L
X<Y	X	X	X	L	L	H
X = Y	L	L	H	L	L	H
X = Y	L	H	L	L	H	L
X = Y	H	L	L	H	L	L

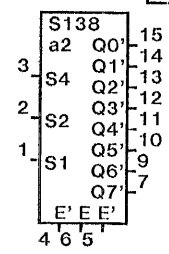
For other states, see catalog

X0/Y0 is the MostSignificant bit
X3/Y3 is the LeastSignificant bit

Tpd		N85	S85
X,Y	X>Y	30	18
X,Y	X<Y	30	18
X,Y	X = Y	35	20
<	X>Y	17	9.5
=	X>Y	17	9.5
=	X = Y	20	12
<	X<Y	17	9.5
=	X<Y	17	9.5

LS138, S138

Binary 1 of 8 Decoder



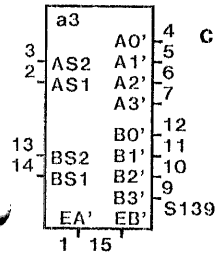
E'	E	E'	S4	S2	S1	Q0'	Q1'	Q2'	Q3'	Q4'	Q5'	Q6'	Q7'
L	H	L	L	L	L	L	H	H	H	H	H	H	H
L	H	L	L	L	H	H	L	H	H	H	H	H	H
L	H	L	L	H	L	H	H	L	H	H	H	H	H
L	H	L	L	H	H	H	H	H	L	H	H	H	H
L	H	L	H	L	L	H	H	H	H	L	H	H	H
L	H	L	H	L	H	H	H	H	H	H	L	H	H
L	H	L	H	H	L	H	H	H	H	H	H	L	H
L	H	L	H	H	H	H	H	H	H	H	H	H	L
H	X	X	X	X	X	H	H	H	H	H	H	H	H
X	X	H	X	X	X	H	H	H	H	H	H	H	H
X	L	X	X	X	X	H	H	H	H	H	H	H	H

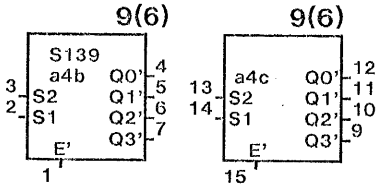
Tpd	LS138	S138
S to Out	41	13.2
Enable	38	12

Q0' is the MostSignificant bit
Q7' is the LeastSignificant bit

LS139, S139

Dual Binary 1 of 4 decoder





S2	S1	E'	0'	1'	2'	3'
X	X	H	H	H	H	H
L	L	L	L	H	H	H
L	H	L	H	L	H	H
H	L	L	H	H	L	H
H	H	L	H	H	H	L

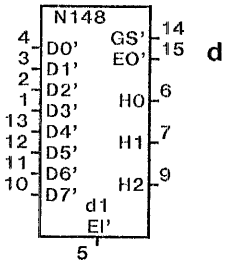
Each Section

A0'/B0' is the MostSignificant bit
A3'/B3' is the LeastSignificant bit

Tpd	LS139	S139
S to Out	38	13.2
Enable	32	11

N148

8 Line Priority Encoder



EI' = Enable In (cascade)
EO' = Enable Out
GS = Group Select (EOvEI')
DO' is higest priority with 000 on outputs when asserted

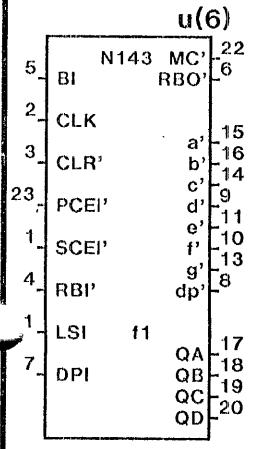
Tpd D to H =	21
Tpd D to EO =	27.5
Tpd D to GS =	33
Tpd EI to H/GS =	17
Tpd EI to EO =	33

DO' is the MostSignificant bit
D7' is the LeastSignificant bit

H0 is the MostSignificant bit
H2 is the LeastSignificant bit

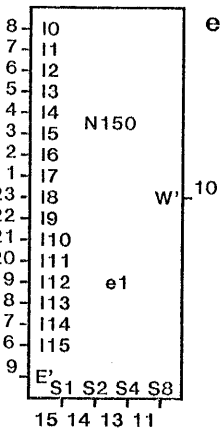
N143

4-bits Count/Latch/LED Drivers



N150

1 of 16 multiplexor



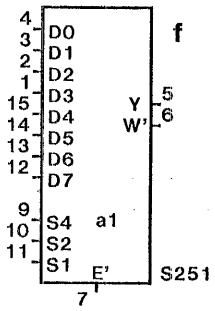
S8,S4,S2,S1 = 0000 selects I0
S8,S4,S2,S1 = 0001 selects I1
S8,S4,S2,S1 = 0010 selects I2
S8,S4,S2,S1 = 0100 selects I4
S8,S4,S2,S1 = 1000 selects I8
S8,S4,S2,S1 = 1111 selects I15

Tpd select to out 35
tpd input to out 20
Tpd Enable to out 30

I0 is the MostSignificant bit
I15 is the LeastSignificant bit

N151,LS151, S151 N251,LS251, S251

1 of 8 Multiplexor



S4,S2,S1 = 000 selects D0
S4,S2,S1 = 001 selects D1
S4,S2,S1 = 010 selects D2
S4,S2,S1 = 100 selects D4
S4,S2,S1 = 111 selects D7

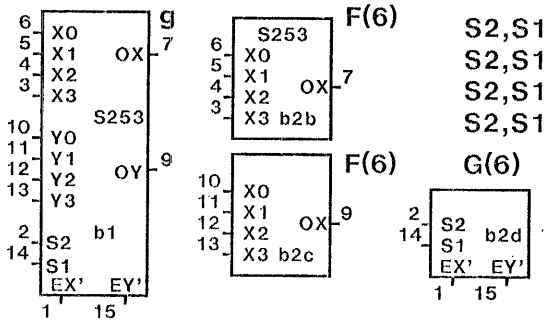
	151	251
Tpd S to Y	38,50, 20	45,45, 21.5
Tpd S to W'	30,39, 16.5	33,33, 17
Tpd D to Y	27,32, 13.5	28,28, 13.5
Tpd D to W'	14,21,8	15,15,8
Tpd E to Y	33,42, 20	40,40, 23
Tpd E to W'	23,31, 14.5	40,40, 23

251 has Tri State Outputs

D0 is the MostSignificant bit, D7 is the LeastSignificant bit

N153,LS153, LS253(AM74LS253), S153 , S253(AM74S253)

Dual 1 of 4 multiplexors



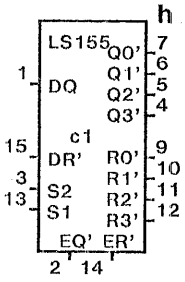
S2,S1 = 00 selects X0(Y0)
S2,S1 = 01 selects X1(Y1)
S2,S1 = 10 selects X2(Y2)
S2,S1 = 11 selects X3(Y3)

Tpd S to O = 34,38,30, 20,20
Tpd Data to O = 23,26,15, 10,10.5
Tpd E to O = 30,32,25, 16.5,23

X0/Y0 is the MostSignificant bit, X3/Y3 is the LeastSignificant bit

N155,LS155

Dual 2 to 4 decoders



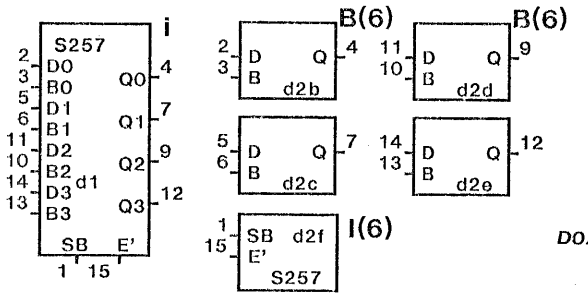
S2,S1 = 00 selects Q0'(R0')
S2,S1 = 01 selects Q1'(R1')
S2,S1 = 10 selects Q2'(R2')
S2,S1 = 11 selects Q3'(R3')

Tpd S to Q(R) = 32,30
Tpd DQ' to Q = 30,27
Tpd DR to R = 32,30
Tpd E to Q(R) = 27,30

Q0'/R0' is the MostSignificant bit, Q3'/R3' is the LeastSignificant bit

N157,LS157, S157 , S257

Quad 2 to 1 multiplexors (non inverting outputs)



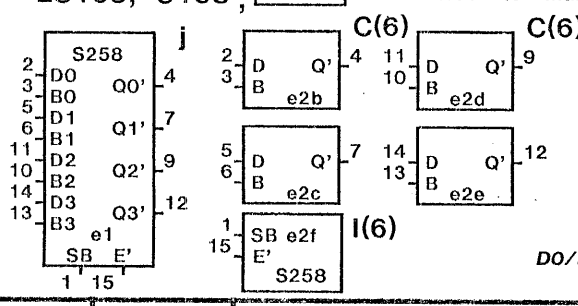
SB = 0 selects D
SB = 1 selects B

Tpd SB to Q = 27,27, 16.5,16.5
Tpd E' to Q = 21,21, 14,23
Tpd D/B to Q = 14,14, 8.5,8.5

D0/B0/Q0 is the MostSignificant bit, D3/B3/Q3 is the LeastSignificant bit

LS158, S158 , S258

Quad 2 to 1 multiplexors (inverting outputs)

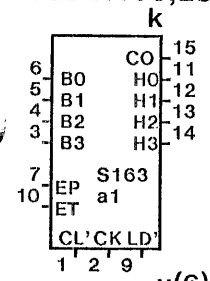


SB = 0 selects D
SB = 1 selects B

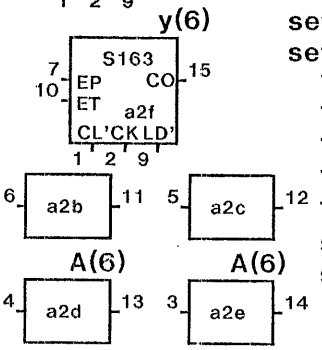
Tpd SB to Q' = 24, 13.2
Tpd E' to Q' = 18, 13.2
Tpd D to Q' = 12, 7

D0/B0/Q0' is the MostSignificant bit, D3/B3/Q3' is the LeastSignificant bit

N160-N163,LS160-LS163, S162-S163 *Four bit counters*

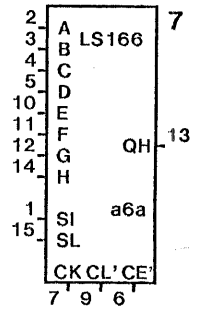


160/161 are decade counters
160/162 has direct clear
161/163 are binary counters
162/163 has synchronous clear
EP is parralell enable (doesn't affect CO)
ET is ripple carry enable (affects CO)
min clock pulse width = 25,25,11
min clear pulse width = 20,20,11
setup load = 25,20,16 hold load = 0,0,0
setup EP = 20,20,14 hold EP = 0,05
Tpd clock to Q = 23,27, 17 (count)
Tpd clock to Q = 29,29, 17 (load)
Tpd clock to CO = 35,35, 28
Tpd ET to CO = 16,23, 17
Tpd CL' to Q = 38,28, ? (direct clear only)
setup CL' = 20,20,16 hold CL' = 0,0,0
setup data = 20,20,5 hold data = 0,0,4

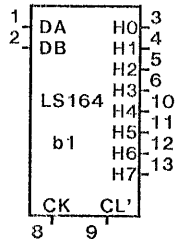


B0/H0 is the MostSignificant bit
B3/H3 is the LeastSignificant bit

LS166, N166 *8 bits P/S shifter*

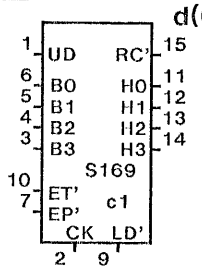


N164,LS164 *8 bit parallel-out serial shift register*



DA and DB must both be high for a 1 to get shifted in (they are Anded together)
Asynchronous clear
width of clock or clear min = 20
Tpd 37,32 hold = 5,5
tpd clear = 42,36
Data setup = 15,15

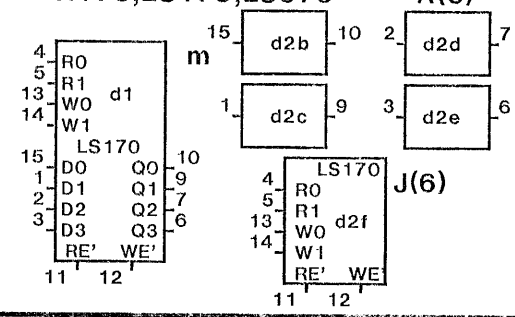
S169 LS169 *4-bits Up/Down Binary Counters*



Tpd CK to H's = 17
Tpd CK to RC' = 30
Tpd UD to RC' = 24
Tpd ET' to RC' = 27
min clock pulse = 12
setup data = 5 setup load = 7
setup EP'/ET' = 6 setup UD = 22

B0/H0 is the MostSignificant bit
B3/H3 is the LeastSignificant bit

N170,LS170,LS670 *4 by 4 Register Files*



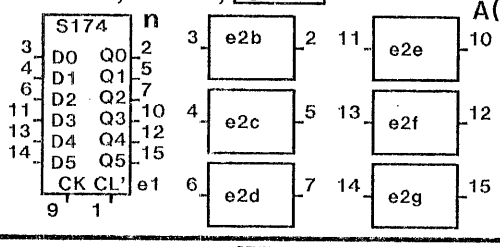
R0 and R1 select read address
Wo and W1 select write address
RE' is output enable
WE' is write enable

170s have OC outputs
LS670 has tristate outputs

Tpd R0/R1 to Q = 40,40,45
Tpd RE to Q = 30,30,35-50
Tpd WE to Q = 45,45,50
Tpd D to Q = 45,45,45
Data setup = 10 hold = 15
min width of write pulse = 25

R0/W0 is the MostSignificant bit
R1/W1 is the LeastSignificant bit

N174,LS174, S174 *Hex D Registers*



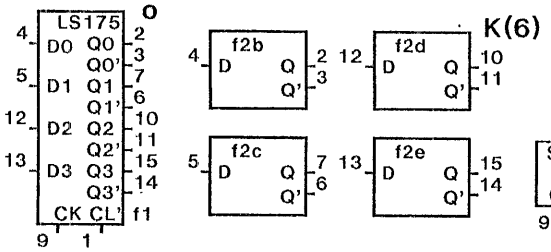
Clear is asynchronous

clock pulse width min = 20,20,9
clear pulse width min = 20,20,14
clear inactive to clock min = 25,25,

Tpd CK to Q = 30,30, 19
Tpd CL' to Q = 35,35, 25

Data setup = 20,20,6
Data hold = 5,5,4

N175,LS175, S175 *Quad D registers with Q and Q' outputs*



Clear is asynchronous

clock pulse width min = 20,20,9
clear pulse width min = 20,20,14

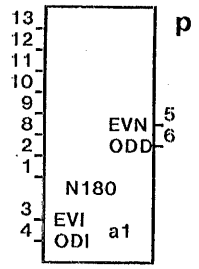
Tpd CK to Q = 30,30, 19
Tpd CL' to Q = 35,35, 25

Data setup = 20,20,6
Data hold = 5,5,4

XEROX SDD	Project NSIL	TTL MSI DataSheet	File TTL5.sil	Designer Rosen	Rev A	Date 4/4/80	Page 5

N180

9 bit ODD/EVEN parity generator/checker

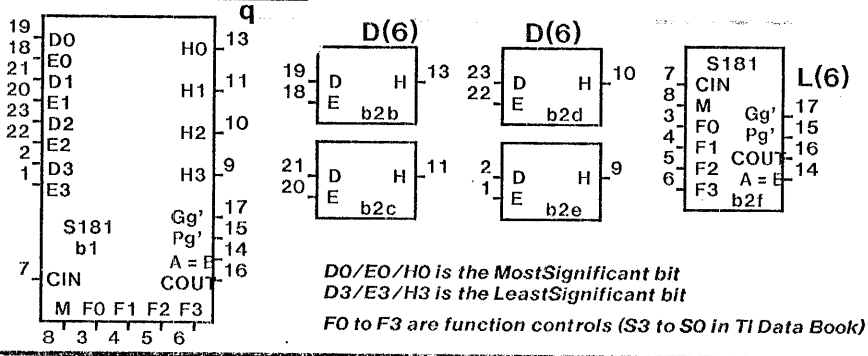


EVI and OVI are cascading inputs

Tpd IN to EVN = 68
(OVI grounded)
Tpd IN to EVN = 48
(EVI grounded)
Tpd IN to ODD = 48 (OVI gnded)
Tpd IN to ODD = 68 (EVI gnded)
EVI/OVI to EVN/ODD = 20

N181,LS181, S181

Arithmetic Logic Unit



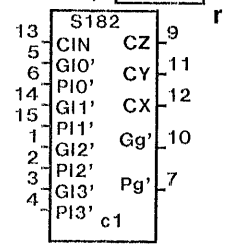
D0/E0/H0 is the MostSignificant bit
D3/E3/H3 is the LeastSignificant bit

F0 to F3 are function controls (S3 to S0 in TI Data Book)

Tpd CIN to COUT = 19,27, 12
Tpd D/E to COUT(Sum) = 43,38, 21
Tpd D/E to COUT(Diff) = 50,41, 26.5
Tpd CIN to H = 19,26, 14
Tpd D/E to Pg/Gg(Sum) = 19,30,14
Tpd D/E to Pg/Gg(Diff) = 25,33, 17
Tpd D/E to H (Sum) = 42,32, 19
Tpd D/E to H (Diff) = 48,32, 25
Tpd D/E to H (Logic) = 48,38, 25
Tpd D/E to "A = B" = 50,6235

N182, S182

Lookahead carry generator



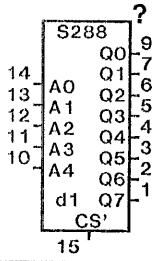
GI0'/PI0' is the MostSignificant bit
GI3'/PI3' is the LeastSignificant bit

CZ is the MostSignificant bit
CX is the LeastSignificant bit

Tpd G/P to C = 22, 8
Tpd G/P to Pg = 22, 11
Tpd G/P to Gg = 22, 11.5
Tpd CIN to CZ = ?, 11.5

N188,N288,S188,S288

32 by 8 PROM

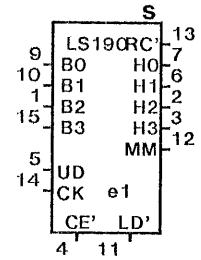


188 is OC outputs
288 is Tri-state

Address Access = 50,50,30,30
Enable Access = 50,50

N190,N191,LS190,LS191

Synchronous Up/Down Counters with Mode Control



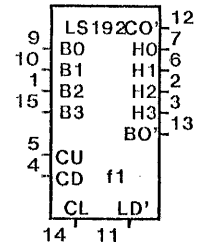
190s are decade counters
191s are binary counters
UD = 0 for up count, 1 for down count
Change CE only when CK is high
LD is asynchronous RC is ripple carry
MM is max/min output (no carry in)
Min width CK = 25, min width LD = 35
Data setup = 25, Data hold = 0

Tpd CK to H = 36
Tpd LD to Q = 50
Tpd B to H = 50
Tpd CK to MM = 52
Tpd CK to RC = 24
Tpd UD to RC = 45
Tpd UD to MM = 33

B0/H0 is the MostSignificant bit
B3/H3 is the LeastSignificant bit

N192,N193,LS192,LS193

Synchronous Up/Down Counters with Dual clock



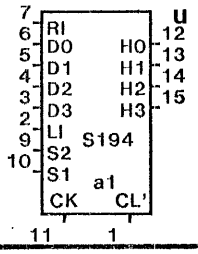
192s are decade counters
193s are binary counters
Hold one clock input high when pulsing other clock
LD is asynchronous Clear is asynchronous
BO is fed forward from CD CO is fed forward from CU
Min width CK,LD or CL = 20, data setup = 20, hold = 0

B0/H0 is the MostSignificant bit
B3/H3 is the LeastSignificant bit

Tpd CK to H = 47
Tpd LD to H = 40
Tpd CL to H = 35
Tpd CU to CO = 26
Tpd CD to BO = 24

N194,LS194, S194

4 bit Bidirectional shift register



S2	S1	operation
H	H	load
L	H	shift right (H1 gets H0)
H	L	shift left (H0 gets H1)
L	L	Do nothing

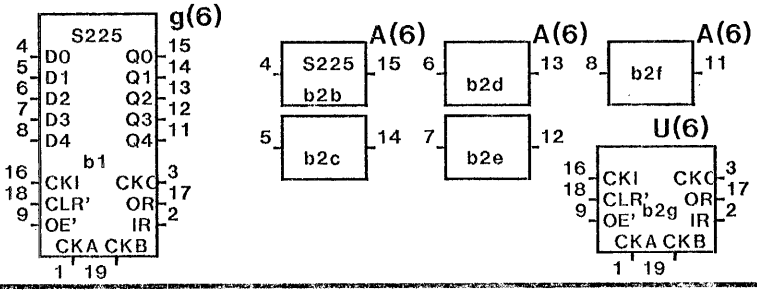
Clear is asynchronous

D0/H0 is the MostSignificant bit
D3/H3 is the LeastSignificant bit

Tpd CK to H = 26,47, 18.5
Tpd CL' to H = 30,54, 20.5
Data setup = 20,20,6
Mode setup = 30,30,12
All holds = 0,0,4
min width Clock = 20,20,9
min width Clear = 20,20,14
min Clear Inactive = 20,20,10

S225

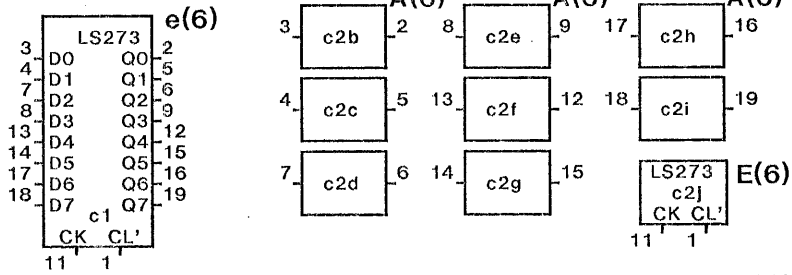
16 by 5 FIFO Memory



See catalog for timing

LS273

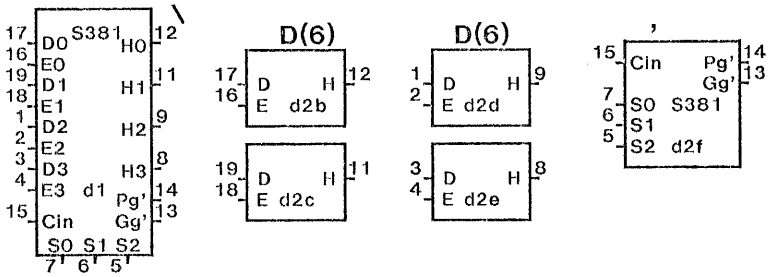
Octal D-Flip-Flop



Tpd CK to Q = 30
Tpd CL' to Q = 30

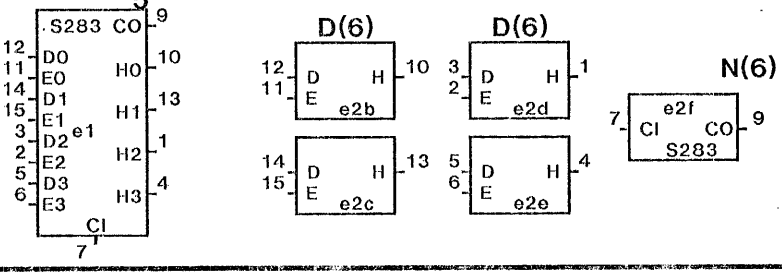
LS381, S381

4 bits Arithmetic Logic Unit



N283,LS283, S283

Four Bit adder

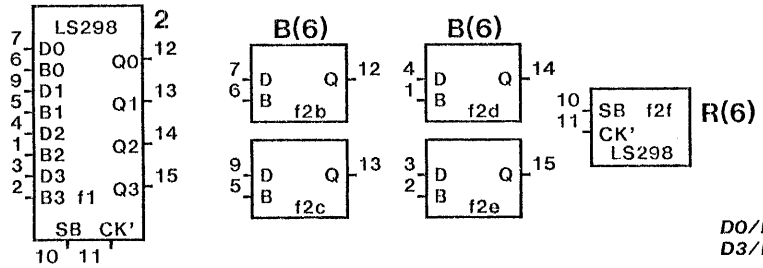


Tpd D/E to H = 24,24, 20
Tpd CI to H = 21,24, 20
Tpd CI to CO = 16,17, 12.5
Tpd D/E to CO = 16,17, 13.5

D0/E0/H0 is the MostSignificant bit
D3/E3/H3 is the LeastSignificant bit
CI is the Carry Input
CO is the Carry Output

N298,LS298

Quad 2 input multiplexor/register



SB = 0 selects D inputs
SB = 1 selects B inputs

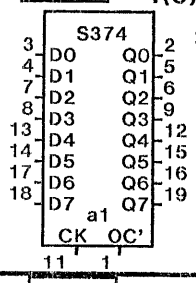
Clock on negative transition

Tpd CK' to Q = 32
Min width CK' = 20
Data Set-up = 15
Data Hold = 5
SB setup = 25
SB Hold = 0

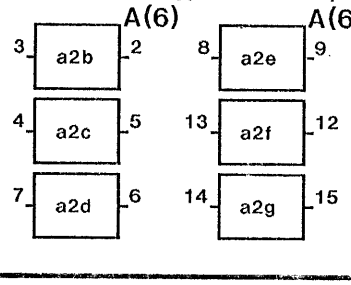
D0/B0/Q0 si the MostSignificant bit
D3/B3/Q3 is the LeastSignificant bit

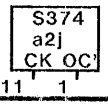
S374 f(6)

Octal D-type Tri-state FlipFlop with Enable



S374 A(6)





T(6) S374

For S373 use \(6) instead of f(6)
and use /(6) instead of T(6)
at pin 11 is EN instead of CK

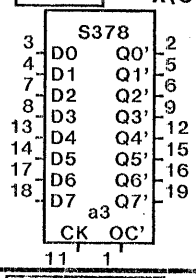
OC'	CK	D	Qn
L	↑↑	H	H
L	↑↑	L	L
L	L	X	Q_{old}
H	X	X	Off

Tpd CK to Q = 19
Tpd OC' to Q = 20
min width clock =9.5
setup data = 6
hold data =3

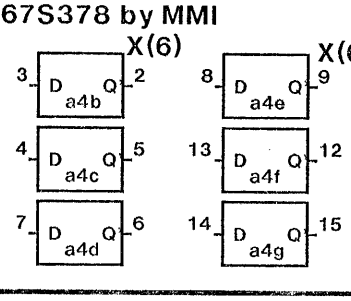
S378 x(6)

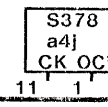
67S378 by MMI

Octal D-type Tri-state FlipFlop with Enable and Inverted Outputs



X(6)





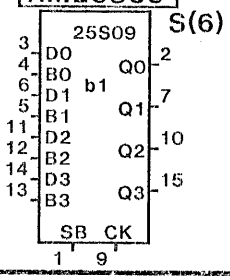
T(6)

OC'	CK	D	Qn'
L	↑↑	H	L
L	↑↑	L	H
L	L	X	Q'_{old}
H	X	X	Off

Tpd CK to Q = 19
Tpd OC' to Q = 20
min width clock =9.5
setup data = 6
hold data =3

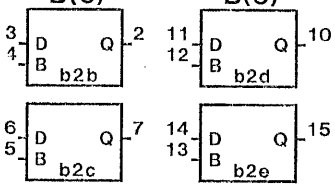
AM25S09

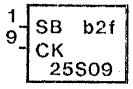
Quad Two Input High Speed Register



S(6)

B(6)



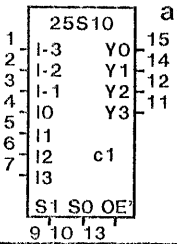


M(6)

Tpd CK to Q = 19
min clock width =9
set data = 7 hold data =4
set SB = 12 hold SB =4
SB = 0 for D inputs
not compatible with 298

AM25S10 a(6)

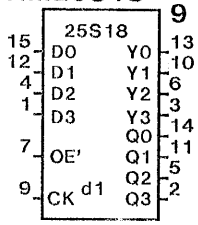
4 bits four-ways Shifter with Tristate Outputs



Tpd I to Y = 12
Tpd S to Y = 20
Tpd OE to Y = 21

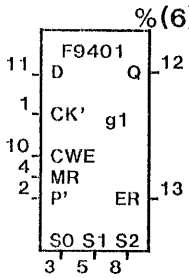
AM25S18 9

Quad Register with both totem pole and Tristate Outputs



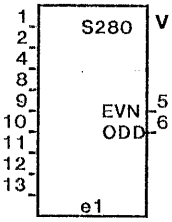
F9401 %(6)

CRC Generator/Checker



S280

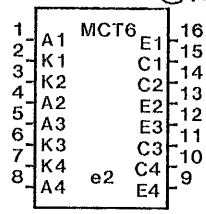
9 bit Even/Odd parity generator/checker



Tpd IN to EVN/OD = 23

MCT6 @(6)

Dual Opto-Isolator (2 8 pins packages)

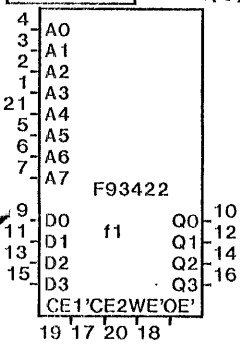


Ouput is isolated from Input and Power supply

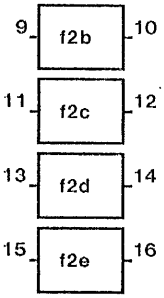
Made by Monsanto

F93422 h(6)

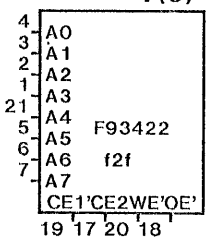
Fairchild 256 by 4 RAM



A(6)



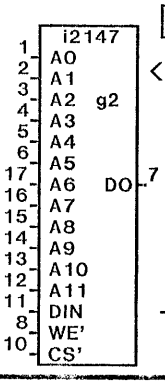
V(6)



Tpd CE' to Q = 30
Tpd A to Q = 45
min write pulse = 30
setup address =10
setup data = 5
setup CE = 5
hold address =5
hold data =5
hold CE = 5
write recovery =40

i2147

1024 by 1 RAM



<(6)

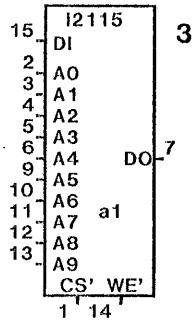
To replace AM9135J

XEROX SDD	Project NSIL	TTL MSI DataSheet	File TTL8.sil	Designer Chang	Rev A	Date 3/6/80	Page 8

i2115,i2125,F93415A, F93425A

1K by 1 RAM

intel,fairchild



2115 and 93415 are OC output

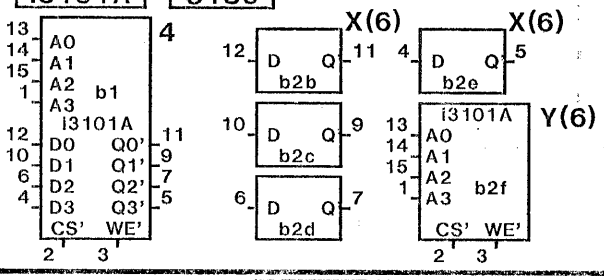
2125 and 93425 are tristate outputs

Address Access = 70,45
Chip Select = 40,30
Chip Select recovery = 40,30
Write pulse min = 35,50
Data Setup = 5,5 Hold = 5,5
Address setup = 15,5 Hold = 5,5

See data sheets for more information on write cycles

i3101A S189

16 by 4 RAM

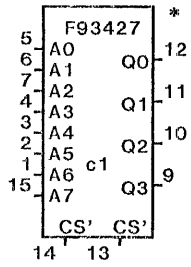


Address Access = 35
Chip Select = 17
Sense Amp Recovery = 35
Write pulse min = 25
Address setup/hold = 0

Data stable prior to l to h transistion on WRITE = 25

i3601, 3621, F93427

256 by 4 PROM

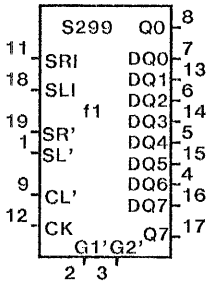


Address Access = 50,40
Enable Access = 25,25

A0 is the MostSignificant bit
A7 is the LeastSignificant bit
Q0 is the MostSignificant bit
Q3 is the LeastSignificant bit

S299

8-bits Universal Shift/Storage Register



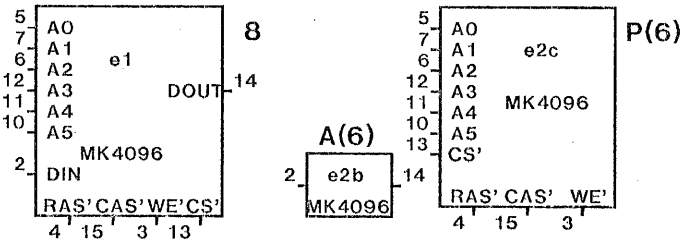
See TI Databook Page 7-437

Tpd CK to DQ = 23
Tpd CL to DQ = 24
Tpd G's to DQ = 20
min width clock = 12
setup data = 8
hold data = 5

Q0 is the MostSignificant bit
Q7 is the LeastSignificant bit

MK4096

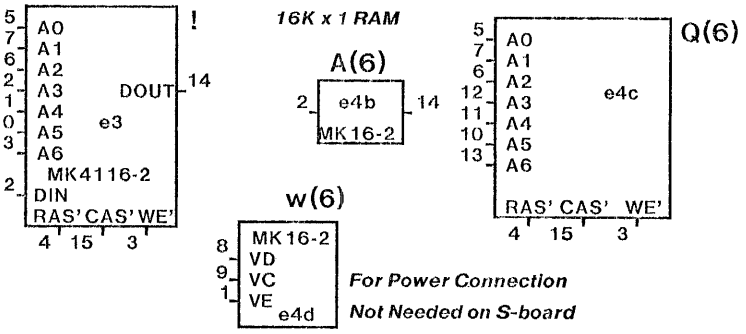
4K by 1 RAM



See Data Sheet for timing specs

MK16-2,MK16-3 MK4116-2, MK4116-3

16K x 1 RAM

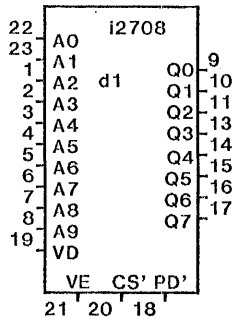


For Power Connection
Not Needed on S-board

See Data Sheet for timing specs

i2708

1K by 8 EPROM



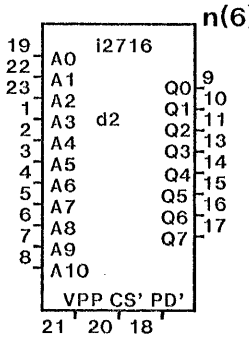
T(Q←A) = 450

See Data Sheet

Note pin numbers are not all same as in catalogue

i2716 i2758

2Kx8 EPROM (2716), 1Kx8 EPROM (2758)



T(Q←A) = 450

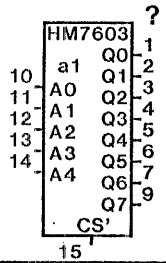
See Data Sheet

Note pin numbers are not all same as in catalogue

XEROX SDD	Project NSIL	TTL MSI DataSheet	File TTL9.sil	Designer Chang	Rev A	Date 7/29/79	Page 9

HM7603

Harris 32X8 PROM



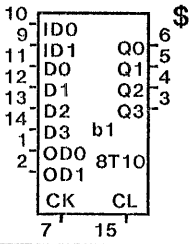
Tpd A's to Q = 40
Tpd CS' to Q = 30

A0 is the MostSignificant bit
A4 is the LeastSignificant bit

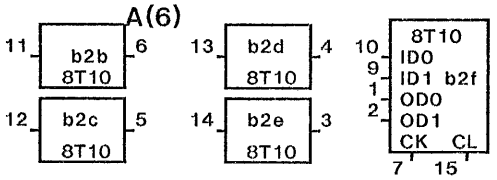
Q0 is the MostSignificant bit
Q7 is the LeastSignificant bit

8T10

Quad register with Tristate outputs



IDO and ID1 are NORed together
ODO and OD1 are NORed together

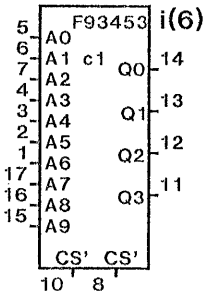


Tpd CK to Q = 28
Tpd OD to Q = 33
Tpd CL to Q = 24
Data setup =6
Data hold =6
min clock width =14
min clear width =17

DO/QO is the MostSignificant bit
D3/Q3 is the LeastSignificant bit

i3625, F93453

1K by 4 PROM



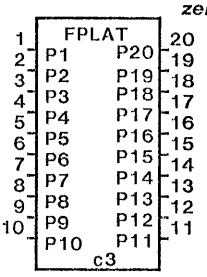
Tpd A's to Q = 50
Tpd CS' to Q = 30

A9 is the LeastSignificant bit
Q3 is the LeastSignificant bit

FPLAT

O(6)
zero

20 pins PlatForm

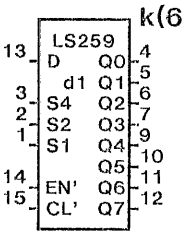


pin # 10 is grounded

pin # 20 is VCC (+5v)

LS259

8-bits Addressable Latches



CL'	EN'	Addressed OutPut	Other OutPut
L	L	D	L
L	H	L	L
H	L	D	Q_{old}
H	H	Q_{old}	Q_{old}

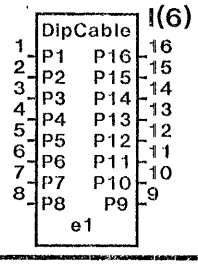
QO is the MostSignificant bit
Q7 is the LeastSignificant bit

S4	S2	S1	Latch
L	L	L	Q0
L	L	H	Q1
L	H	L	Q2
L	H	H	Q3
H	L	L	Q4
H	L	H	Q5
H	H	L	Q6
H	H	H	Q7

T(Q←CL') = 27
T(Q←D) = 32
T(Q←S's) = 38
T(Q←EN') = 35
T_{setup} = 15

DipCable

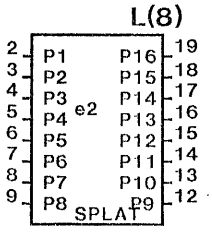
Dip for 16 lines Cable



Note Pin # 8 is grounded

SPLAT

16 pins PlatForm



Note it is staffed in the middle of a 20 pins position on S-board so to avoid GND and VCC

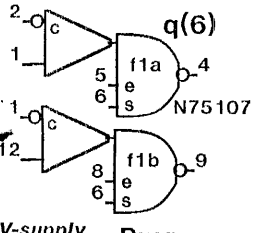
Added the following subgroups in your drawings if you do not want the trace-wire of GND & VCC to be CUT:

GND 10 — 1(6) — e2b

VCC 20 — 1(6) — e2c

N75107

Dual Line Receivers

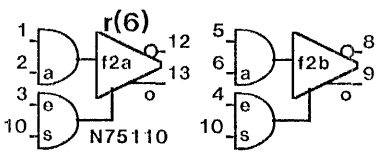


Differential Inputs	Strobe e	Strobe s	OUTPUT
Vge 25mV.	L/H	L/H	H
V<abs(25mV)	L/H	L	H
	L	L/H	H
	H	H	UnKnown
Vle -25mV	L/H	L	H
	L	L/H	H
	H	H	L

V-supply hookup — -Pwr 11 — f2c

N75110

Dual Line Drivers

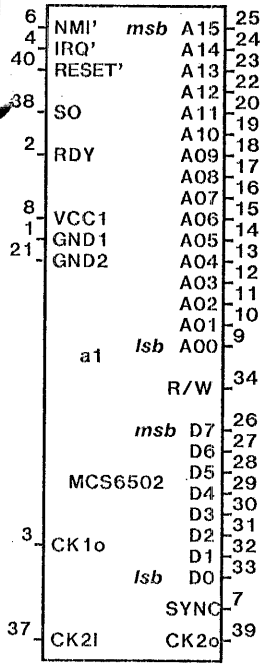


Logic Inputs IN	Logic Inputs a	Strobe e	Strobe s	o	OUT
L/H	L/H	L	L/H	H	H
L/H	L/H	L/H	L	H	H
L	L/H	H	H	L	H
L/H	L	H	H	L	H
H	H	H	H	H	L

V-supply hookup — -Pwr 11 — f2c

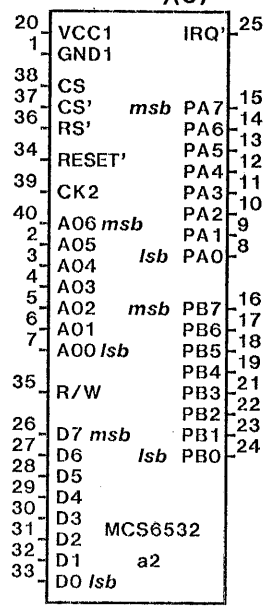
MCS6502

((6)



MCS6532

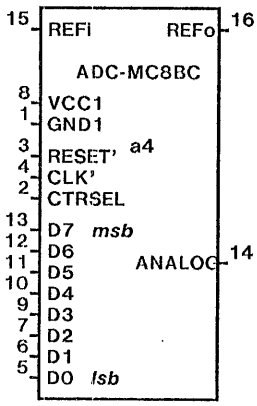
)(6)



ADC-MC8BC

A/D-D/A Converter

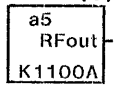
$(6)



K11xxA

Crystal Oscillator

+(6)



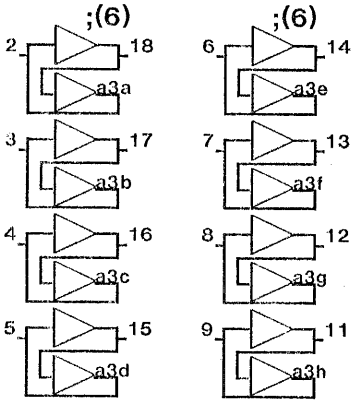
available parts

K1100A K1144A
K1114A K1145A
K1115A
K1116A

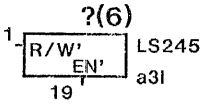
LS245

Octal Bus Transceivers

;(6) ;(6)



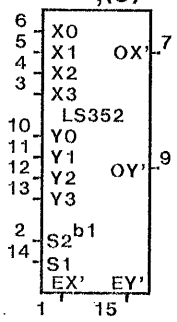
?(6)



LS352

Dual 4x1 Multiplexers with Inverted Outputs

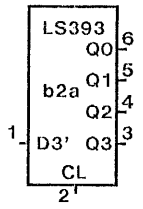
;(5)



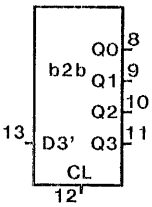
LS393

Dual 4-bits Binary Counters

"(5)



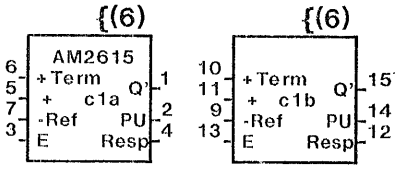
"(5)



AM2615

Differential Receivers

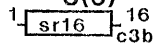
{(6) {(6)



sr16

Serial Resistor

3(6)

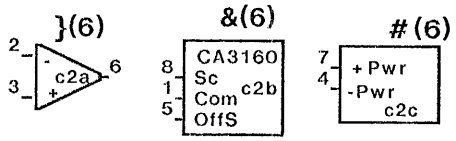


from c to i
for pin-pairs from
2-15 to 8-9

CA3140, CA3160

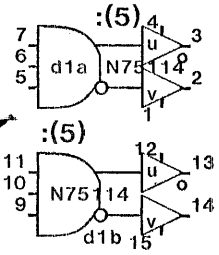
Voltage Comparator

}(6) &(6) #(6)



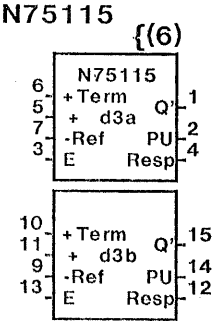
N75114

:(5)



N75115

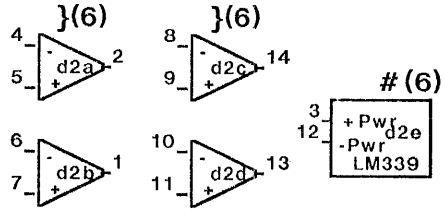
{(6)



LM339

Quad Voltage Comparator

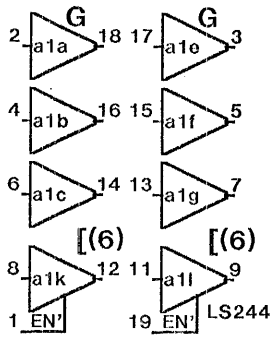
}(6) }(6) #(6)



XEROX EOD	Project D0	TTL Data Sheets	File TTL11.sil	Designer S. Tom Chang	Rev A	Date 7/29/79	Page 11

LS244

Octal Buffer/ Drive-receivers



i2114, i2148

1Kx4 Ram