CHAPTER 3

GRAPHICS EDITOR

Graphics Editor Reference Manual

3.1 INTRODUCTION

The SCALD Graphics Editor serves as the primary interface between you and the design data base. The Graphics Editor is used to create logic drawings (schematics) and body drawings (shapes of devices) using a high resolution CRT display, alphanumeric keyboard, and graphics tablet (Figure 1). In addition to allowing you to create and modify drawings, the Graphics Editor handles the interaction with the operating system to retrieve and store these drawings.

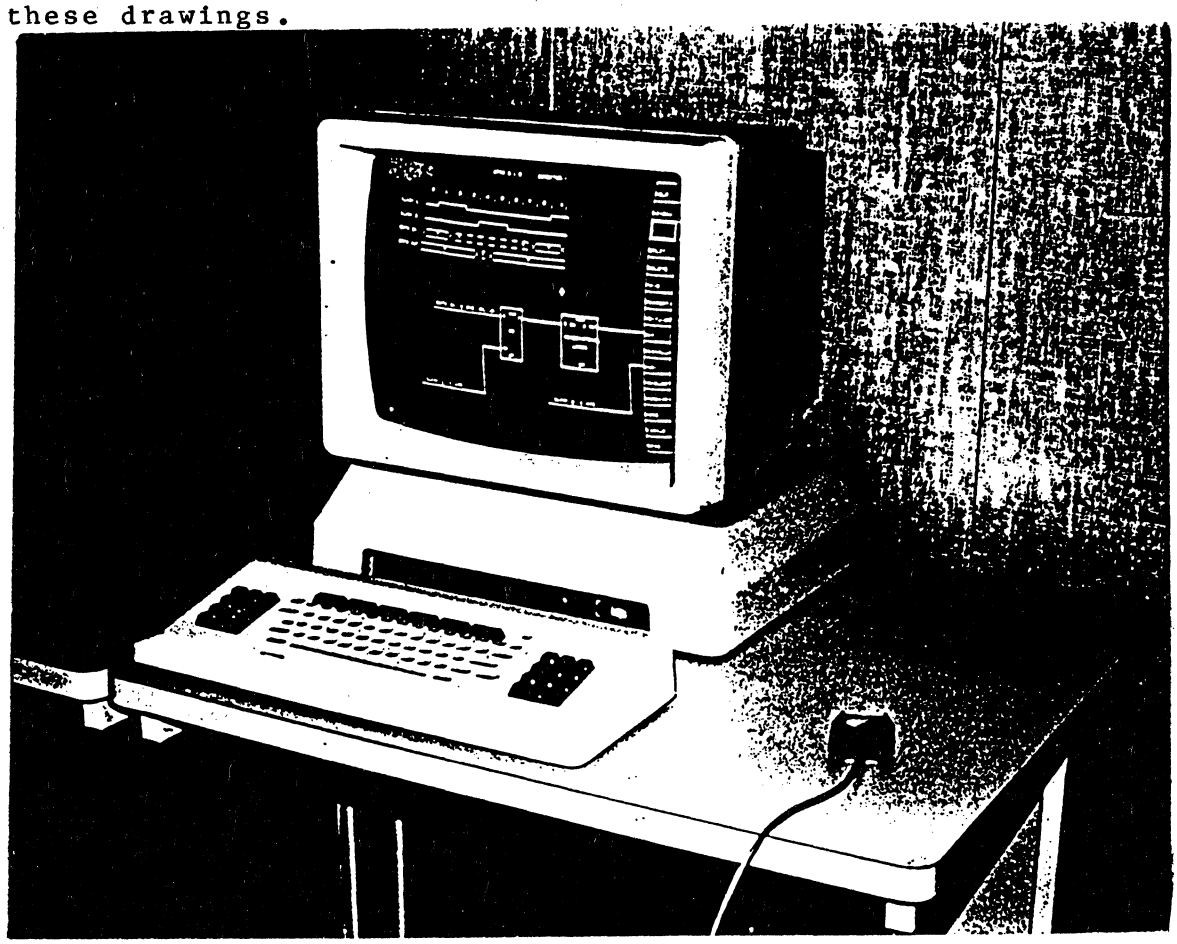


Figure 1. SCALDsystem Design Station

To edit a drawing, you type:

EDIT drawing_name

followed by the "return" key. If the drawing already

exists, then the Graphics Editor will access the appropriate file from the design data base and will display it on the CRT. If the drawing does not exist, that is, it is a new drawing you wish to create, then a blank page will be displayed on the CRT.

In addition to the drawing displayed on the CRT, there is a menu along the right side of the screen. Commands are issued to the Graphics Editor using both the menu and the keyboard. These commands let you place bodies on the drawing, connect pins with wires, add signal names, and other text information, and generally manipulate the information contained in the drawing.

CRT DISPLAY

The CRT display is a 20 inch monochromatic CRT with a resolution of 1024 by 800 pixels. There are four intensity levels per pixel; consequently light, normal, and bright lines may appear on the dark background.

The menu of most frequently used commands is displayed along the right side of the CRT (Figure 2).

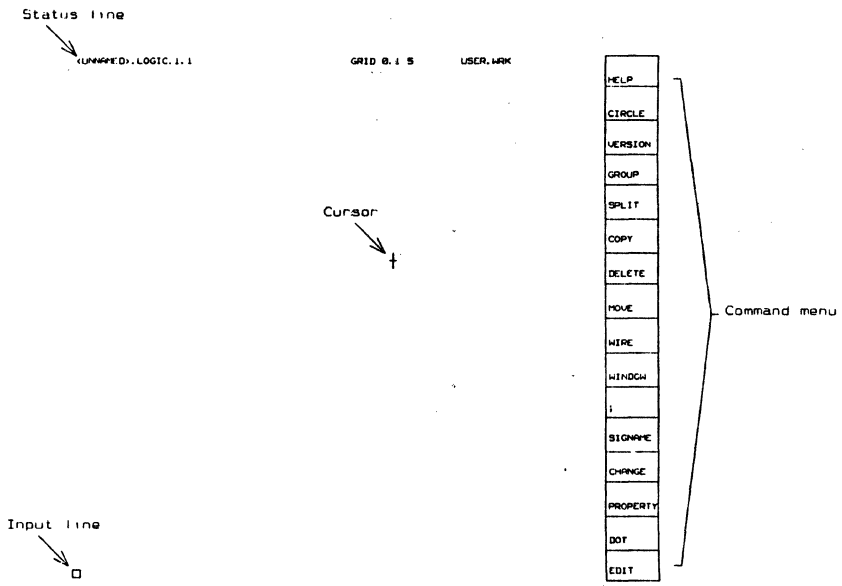


Figure 2. CRT display -- command menu

In addition to the menu, there is a status line displayed along the top of the screen. This status line tells you which file is currently being edited, the status of the grid (to be described later), and the name of the

current working directory.

Along the bottom of the screen is the current text line. As you type on the keyboard, the characters typed will be displayed along this command line. This allows you to make corrections to text prior to its use.

KEYBOARD

The SCALDsystem keyboard is a standard alphanumeric keyboard with programmable function keys along the top and on either side of the main keyboard (Figure 3). The primary purpose of the keyboard is to allow you to type commands, signal names, properties, notes, and other text information required to create a drawing. In addition, the programmable function keys allow you to perform complex operations with a minimum number of key strokes.

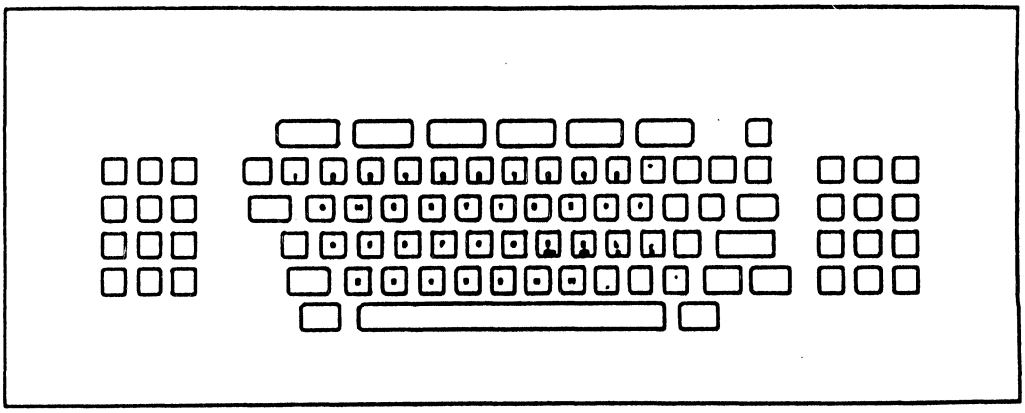


Figure 3. Keyboard

GRAPHICS TABLET

The graphics tablet is built into the table on which the CRT display and keyboard are mounted. You manipulate the puck (Figure 4) in order to move the cursor on the CRT. There are four cursor buttons on the puck that are used to signal to the Graphics Editor that a command is to be selected or executed.

The differences between the four cursor buttons will be described in a later section. For the time being, consider just the use of the yellow button. Throughout this manual, the use of this particular button will be assumed unless explicitly stated to the contrary.

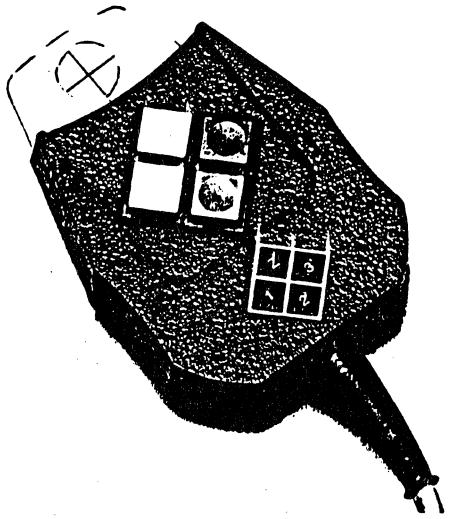


Figure 4. Graphics tablet puck

3.2 INTRODUCTION TO COMMANDS

The form of commands given to the Graphics Editor is

command_name argument_list

The commands may be issued totally from the keyboard, totally from the graphics tablet (using the menu on the display), or using a combination of the tablet and keyboard operations. In general, those commands that require you to type text information, for example, adding a note to a drawing, are issued from the keyboard. On the other hand, those commands that require no text information, for example, the MOVE command, may be executed from the graphics tablet without use of the keyboard. If a command not on the menu is typed from the keyboard, that command is replaces the command in the last menu box. This box can then be selected using the puck just like any other menu command. Initially, the GRID command is in this last menu box.

When the system first starts, the status line reads

UNNAMED.LOGIC.1.1

If one wishes to create a new drawing called "PERIPHERAL INTERFACE", then the command:

EDIT PERIPHERAL INTERFACE<cr>

may be typed. (<cr> indicates that the "RETURN" key should be pressed.) As the command is typed, it will appear along the lower edge of the CRT display (Figure 5). The status line will then indicate the name of the drawing to be created (Figure 6). Since this is a new drawing, the message "New diagram started" will be displayed. (If a drawing called "PERIPHERAL INTERFACE" already existed, then the Graphics Editor would have displayed that drawing.) Note that the last menu box has been changed to read EDIT.

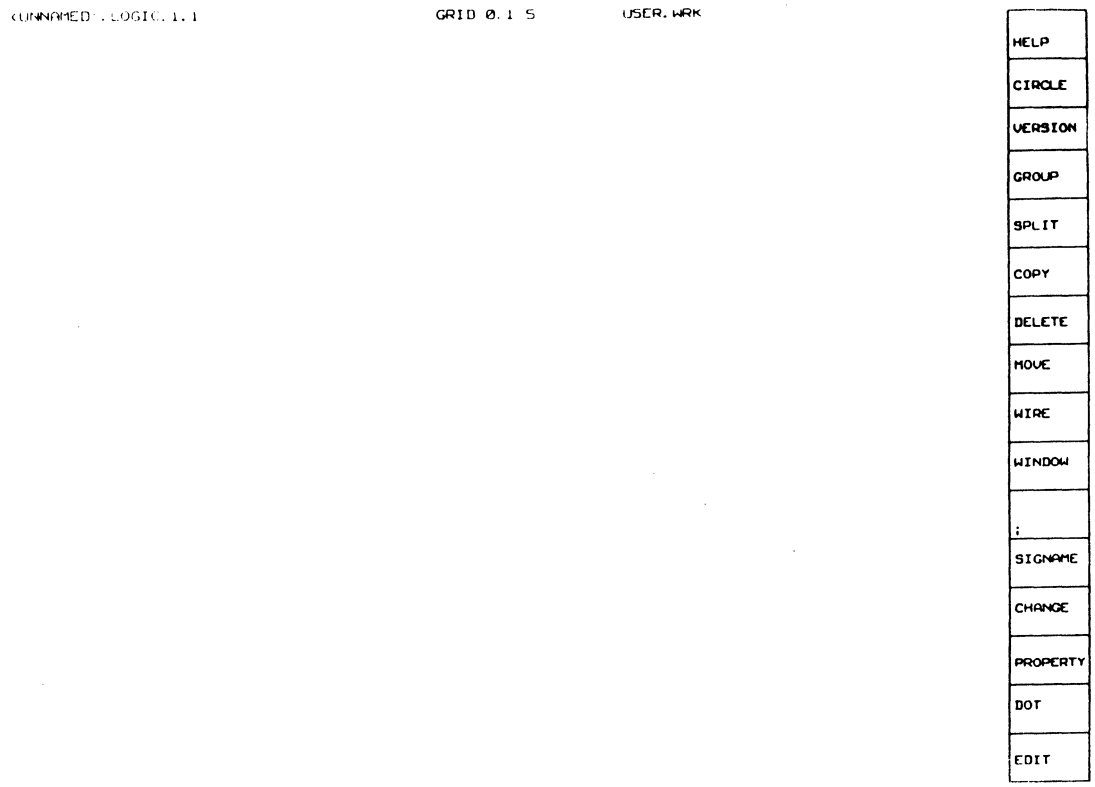


Figure 5. "EDIT PERIPHERAL INTERFACE" Typed and Displayed

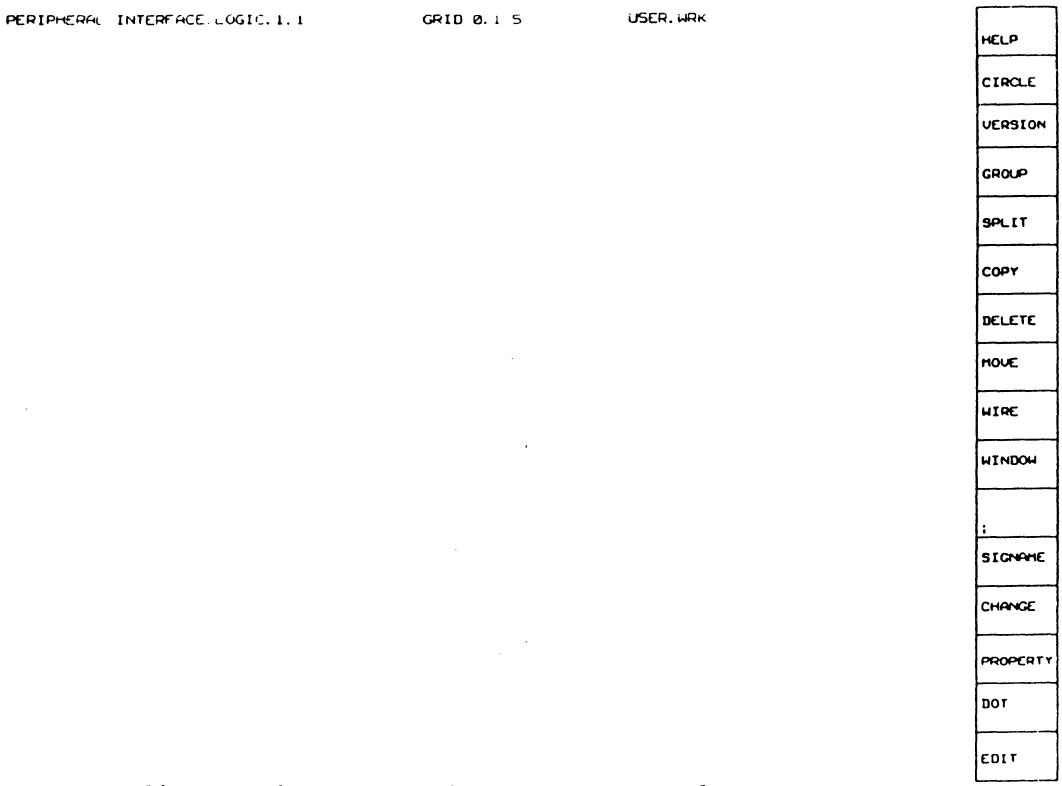


Figure 6. New Diagram Started

For another example of the use of the Graphics Editor commands, consider the addition of bodies to a drawing. Referring to Figure 6, suppose that one wishes to add a NAND gate to the blank page. This is done by typing

ADD NAND<cr>

The Graphics Editor interprets the ADD command and searches through its directories to find the appropriate body. If the body is found an instance of it is attached to the cross hair cursor on the CRT. Next, you must indicate the point on the drawing at which the body should be placed. This is done by manipulating the puck so that the cursor is in the appropriate position on the screen and pressing the yellow button. The command to add a body is then completed; the result is shown in Figure 7.

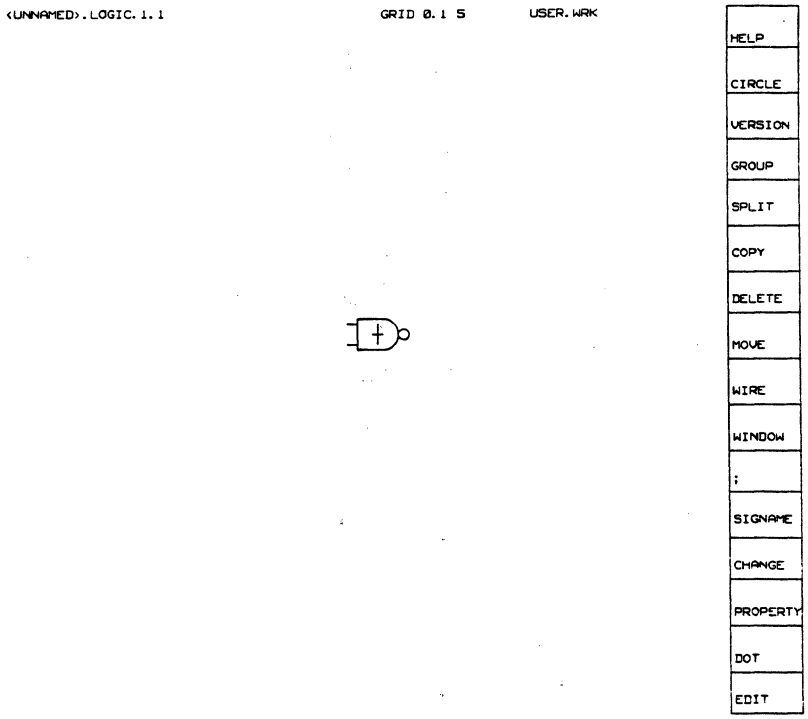


Figure 7. NAND Gate Added

In addition to adding bodies to a drawing, one needs to connect pins on those bodies using wires. For example, Figure 8 shows a drawing with a NAND gate and a flip-flop. To connect the output of the NAND gate to the clock input of the flip-flop, the WIRE command is used. This is done by moving the puck so that the cursor is over the WIRE command on the menu (Figure 9) and pressing the yellow button. The WIRE command is highlighted on the menu so that you know what operation is being performed.

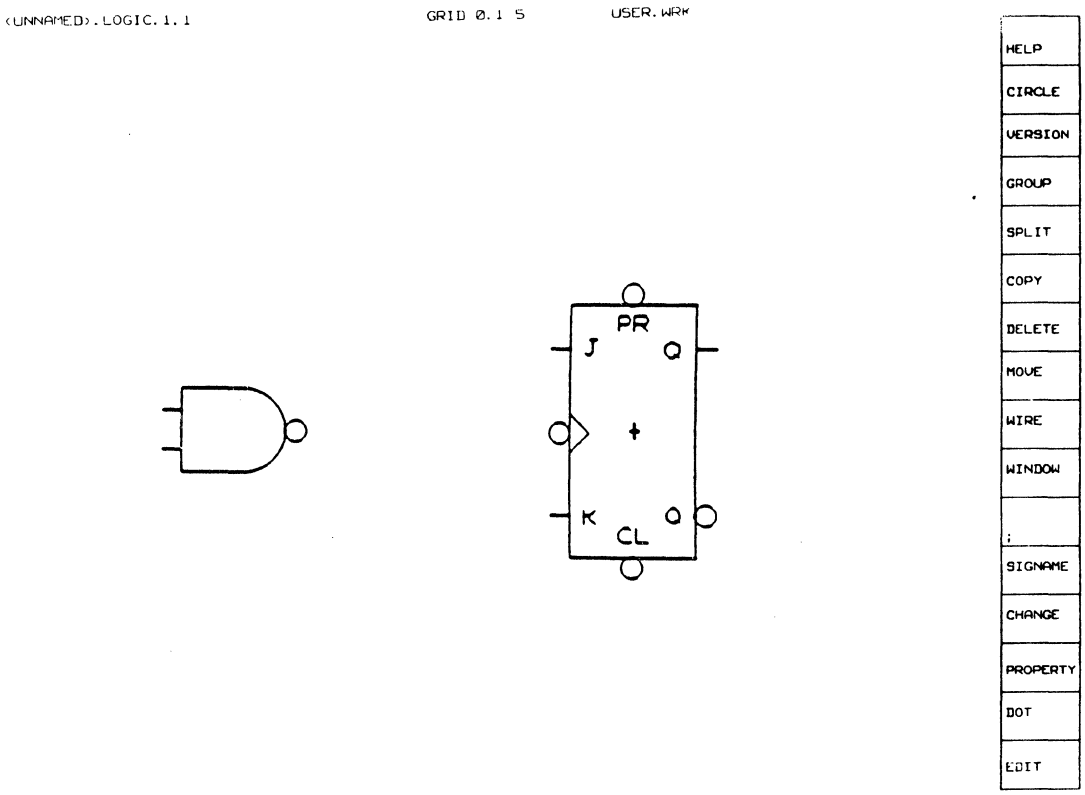


Figure 8. NAND and JK FF bodies added

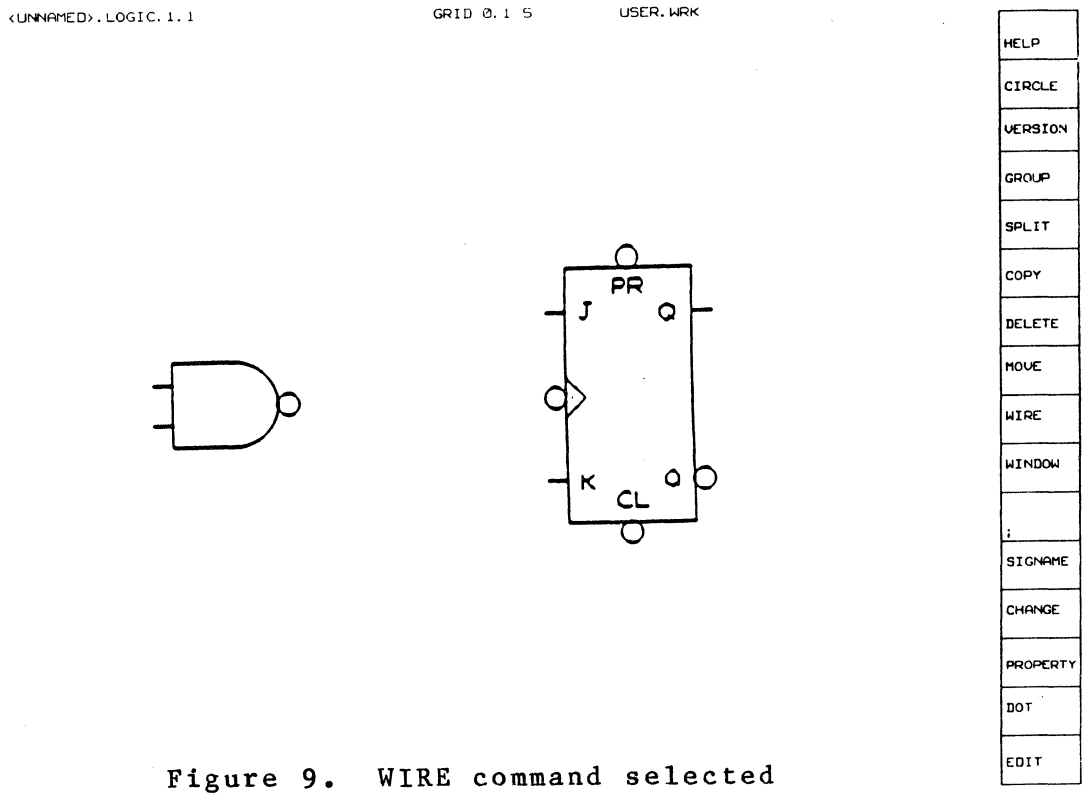


Figure 9. WIRE command selected

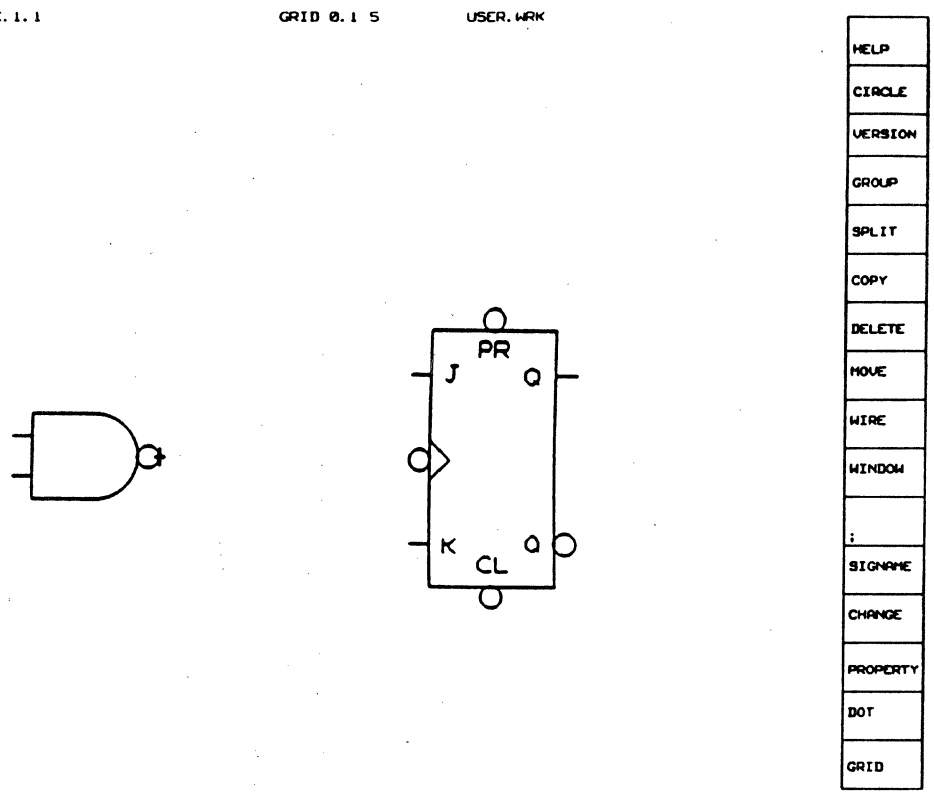


Figure 10. Selecting the starting point for a wire

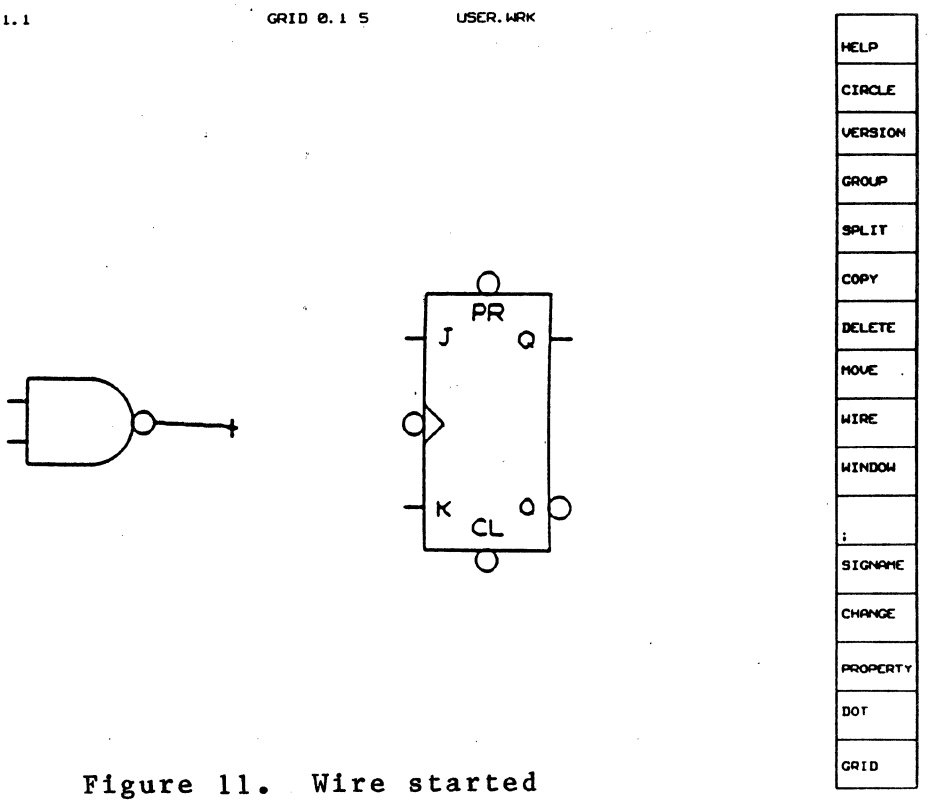


Figure 11. Wire started

In order to draw a wire, you must first specify the starting point for that wire. This is done by moving the puck so the cursor is over the starting point on the drawing (Figure 10) and pressing the yellow button. If the cursor is now moved away from that point, you will see that there is a wire connected from the starting point on the drawing to the current position of the cursor. As the cursor moved about the drawing, the wire will follow the cursor (Figure 11). Note that whatever direction the puck is moved, the wire always remains fixed in the horizontal direction (e.g. is orthogonal or bent in the middle). Changing the direction of the wire is discussed in the section on the uses of the puck buttons below.

To place the other end of the wire on the clock input pin of the flip-flop, simply move the cursor (by manipulating the puck) to that pin and press the yellow button. The connection is made and the wire is terminated. A wire is terminated when connected directly to a pin.

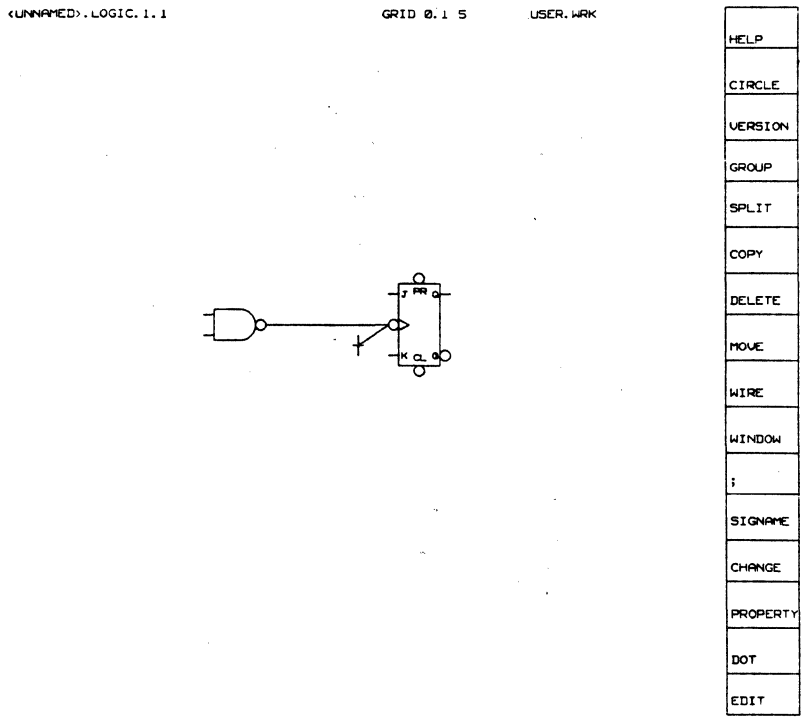


Figure 12. Wire connected to a second pin

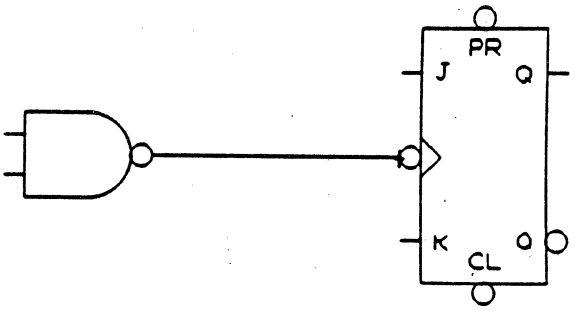


Figure 13. End of wire run

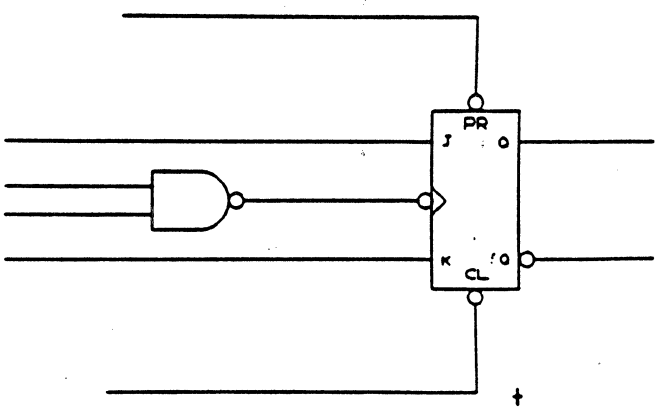


Figure 14. Wires connected to a pin at one end only

Wires do not have to be added from one pin to another. They may, in fact, be added anywhere on a drawing. Usually, however, a wire is connected at one end or the other to a pin on a body. Figure 14 shows several wires that have been added to the drawing shown in Figure 13. Each wire was placed by starting a wire at the appropriate pin, moving the wire to the point at which the wire is to be terminated, and pressing the yellow button twice without moving the cursor from that final position. Whenever a wire is routed to a point in a drawing other than a pin, it is not terminated. Another piece of wire is attached to the cursor so that you can continue the wire to yet another point. In order to terminate a wire that is not ending on a pin, you must press the yellow button twice in the same location.

Now consider the requirement to move an object from one point to another on the drawing. The MOVE command is selected by moving the cursor to the MOVE command on the menu and pressing the yellow button. Next, the cursor is moved so that it is close to the object that is to be moved (it does not have to be exactly on the object) and the yellow button is pressed. This causes the selected object to be highlighted (intensified) and to move wherever the cursor is moved (Figure 15).

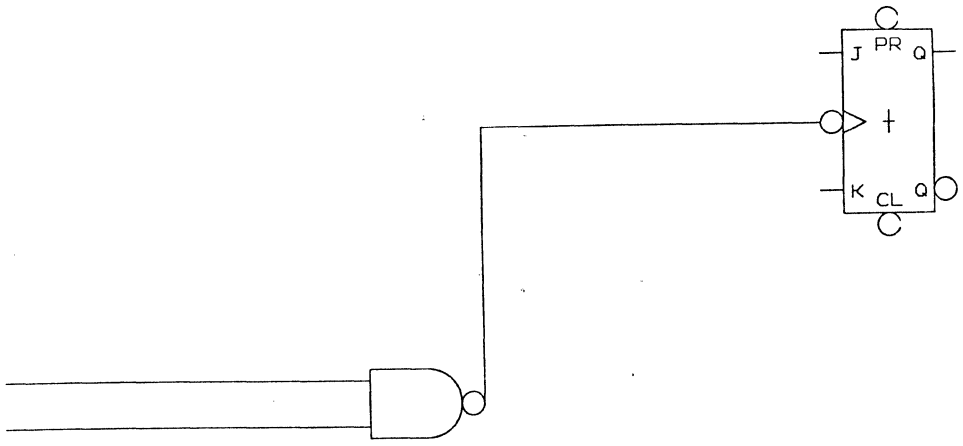


Figure 15. Flip flop moved -- wires remain connected and orthogonal

Note that wires remain connected and orthogonal (bent) as objects are moved about the drawing. This is done so that the user will not have to reconnect or straighten those wires after a body has been moved.

To place the object in its final position, simply move the cursor until the body is positioned correctly and press the yellow button one more time. This deposits the object at the indicated position.

To move the vertical wire on the left to the right, select MOVE from the menu, select the top of the vertical wire with the yellow cursor button and push the cursor to the right. When the wire is in the correct position, place it down by pressing the yellow button again (Figure 16).

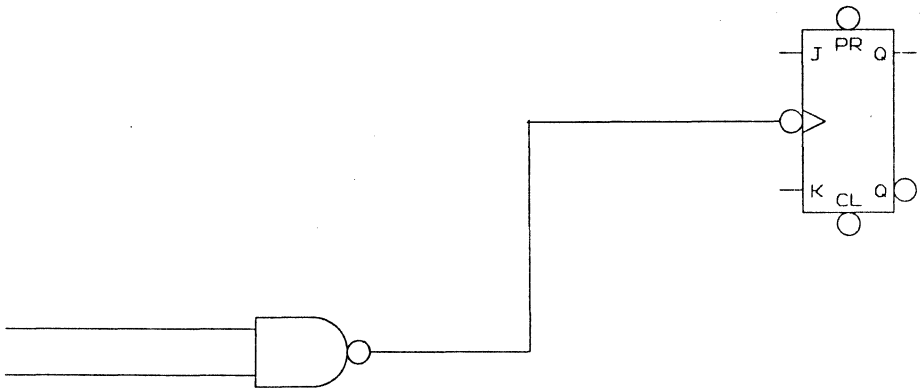


Figure 16. Wires moved

Figure 17 and its related text have been removed from the SCALD Graphics Editor Reference Manual.

Figure 18 and its related text have been removed from the SCALD Graphics Editor Reference Manual.

WINDOWING AND SCALING

The SCALD Graphics Editor manages drawings which, if plotted in a single piece, would be as large as 64 inches on a side. If that much area were to be displayed on the screen, then the objects on the drawing would be so small that they would be very difficult to manipulate. The Graphics Editor lets you view that large drawing area through a "window." By positioning this window and changing the scale at which images are viewed through that window, you are able to display anything from a very small portion of a drawing to the entire drawing on the CRT display.

There are several ways that you may make use of the WINDOW command to change the portion of the drawing that is currently being displayed on the CRT. For example, suppose that the display is as shown in Figure 19. One may pan about the drawing by defining a point on the drawing that should be moved to the center of the screen. This is done by selecting the WINDOW command and establishing a point on the display that is to be moved to the center. An asterisk will appear on the drawing at that point (the S02 gate at the right of the drawing).

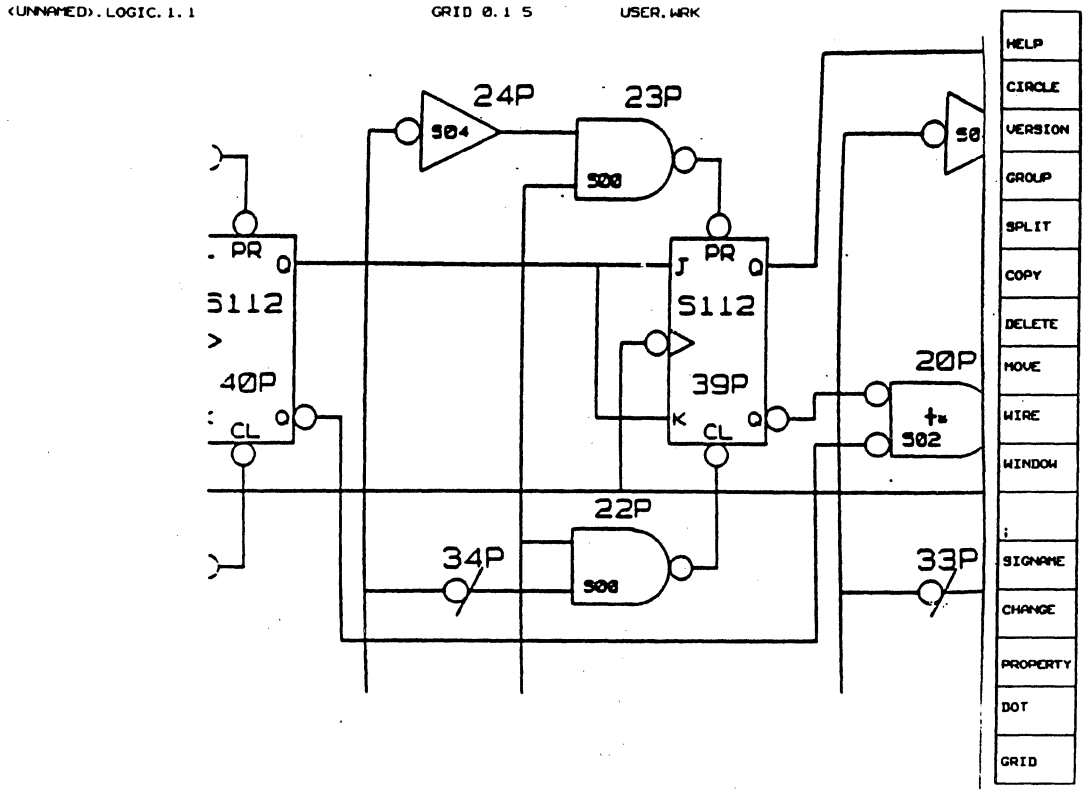


Figure 19. Panning with the WINDOW command

As will be seen later, the WINDOW command may have a variable number of arguments. In the above example, a single argument was used (the point that is to be used as the new center point for the display). Since the WINDOW command may be expecting more arguments, the end of the argument list must be indicated. This is done by selecting the semicolon on the menu. The WINDOW command, having received a single argument, interprets that command and redraws the display as shown in Figure 20.

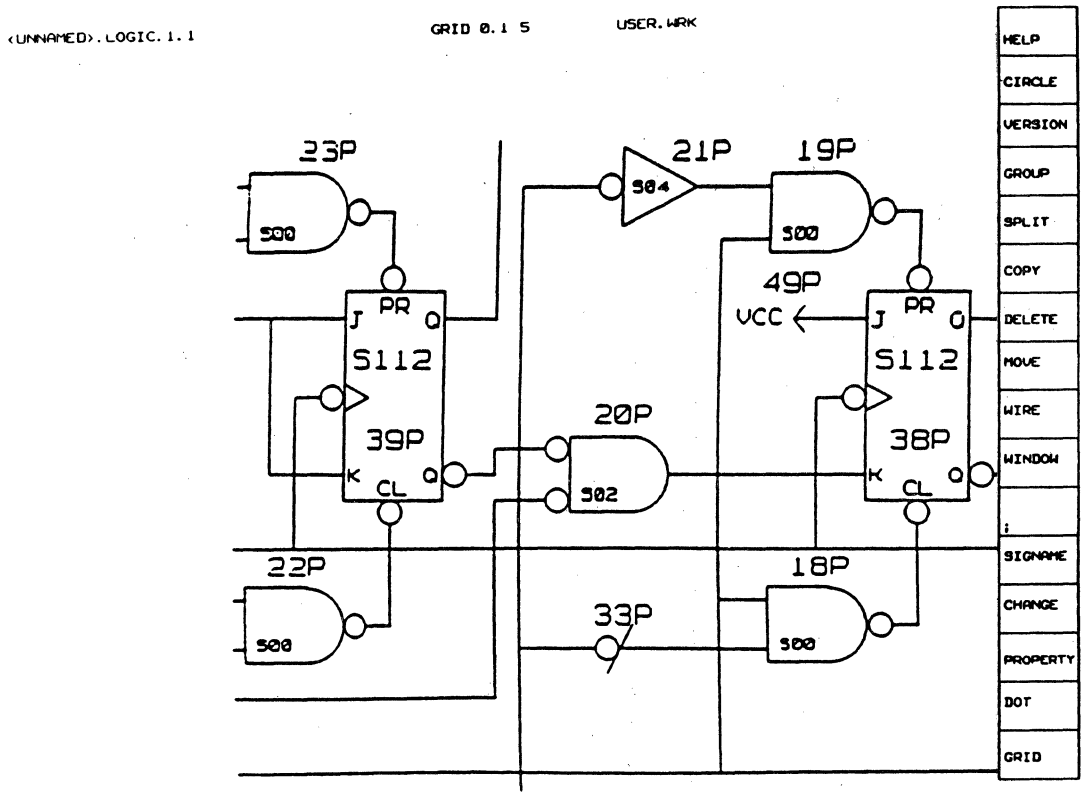


Figure 20. Panning completed

Another variation of the WINDOW command that uses a single argument is the one that fills the screen with all of the information that is contained on that particular drawing. For example, if one is working with a B-size drawing (11" x 17"), then it would be very convenient to display that entire drawing at a somewhat reduced scale on the CRT. This is done by selecting the WINDOW command on the menu (if it has not already been done), typing the letter "F" on the keyboard, and either depressing the return key on the keyboard or selecting the semicolon on the menu. The Graphics Editor then examines the entire page and adjusts both the windowing scale factor and the window position such that all information is displayed on the CRT. The result is as shown in Figure 21.

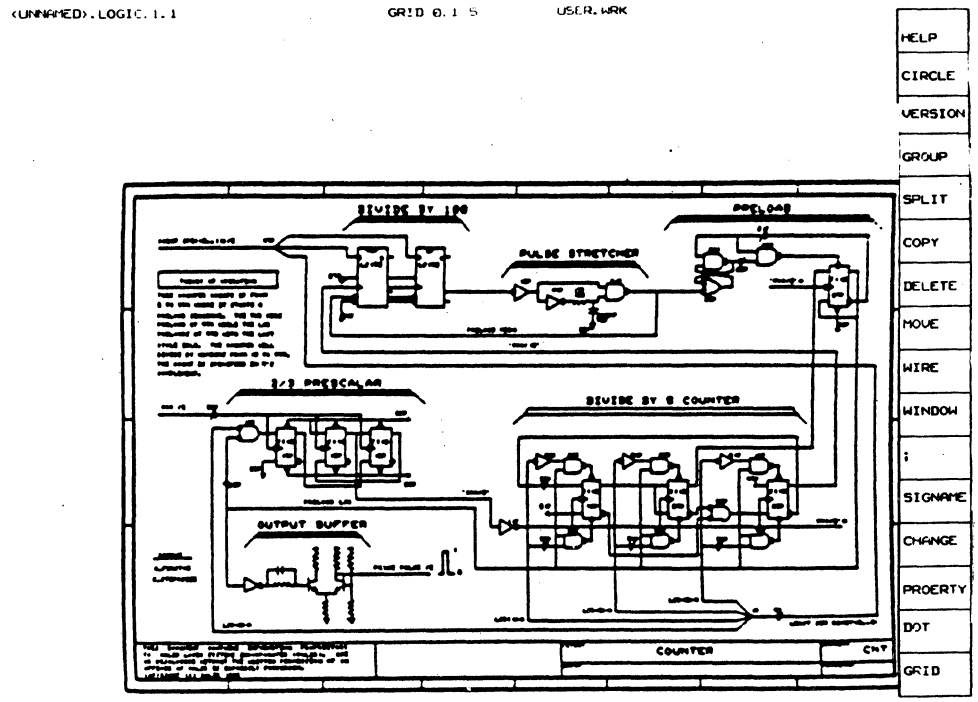


Figure 21. Display filled with a B-size drawing

A third use of the WINDOW command lets you define a rectangular area that he wishes to display on the CRT. The rectangular area that will be enlarged to fill the screen is defined by establishing two points on the diagonal of the rectangle. This is done by selecting the WINDOW command, positioning the cursor at one corner of the rectangular area, pressing the yellow button, moving the cursor to the opposite corner of the rectangular area, and pressing the yellow button again. The result shown in Figure 22 is a pair of asterisks at the appropriate points on the display. The WINDOW command is then terminated by selecting the semicolon on the menu and pressing the yellow button. Figure 23 shows the drawing after it has been redrawn.

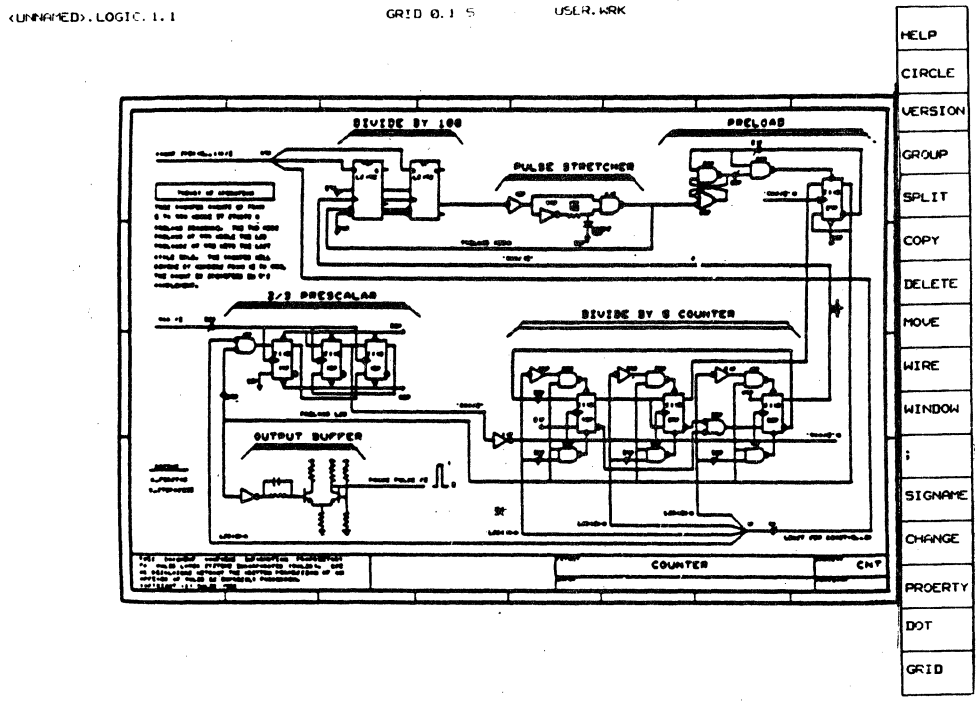


Figure 22. Region to be displayed has been selected

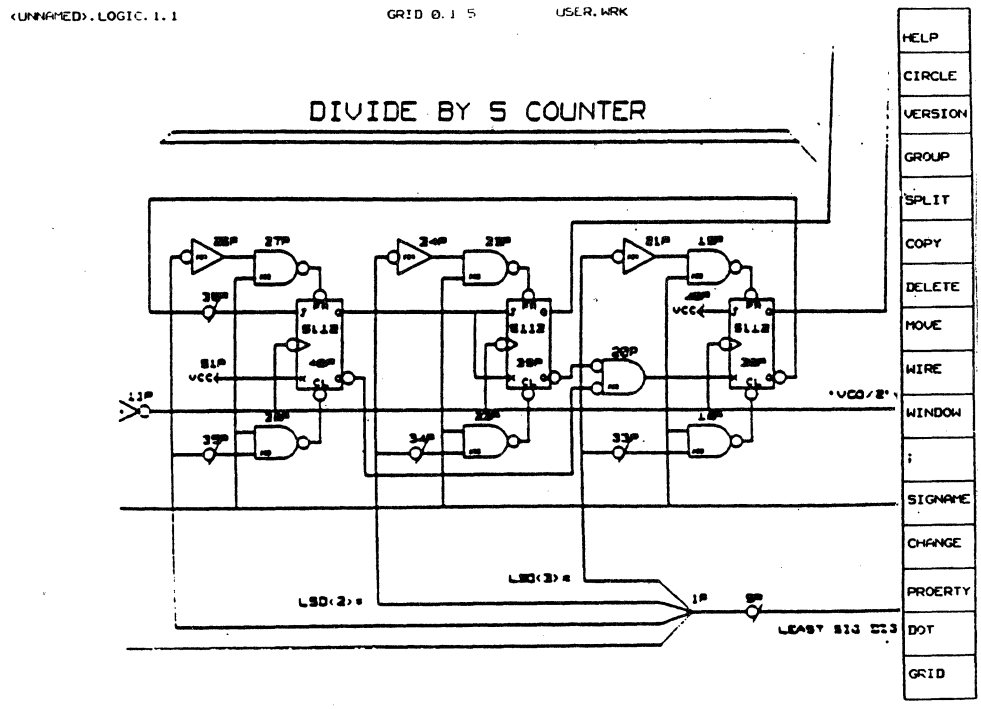


Figure 23. Display re-drawn, showing the desired region

The final use of the WINDOW command allows you to specify a center point and a scaling ratio so that panning and scaling may be done simultaneously. This is done by selecting WINDOW on the menu, selecting a center point (with the cursor and yellow button), selecting a first distance by placing an asterisk the appropriate distance away from the center point in any direction, and finally, by selecting a second distance by placing a third asterisk. Figure 24 shows the distances that were selected in this example.

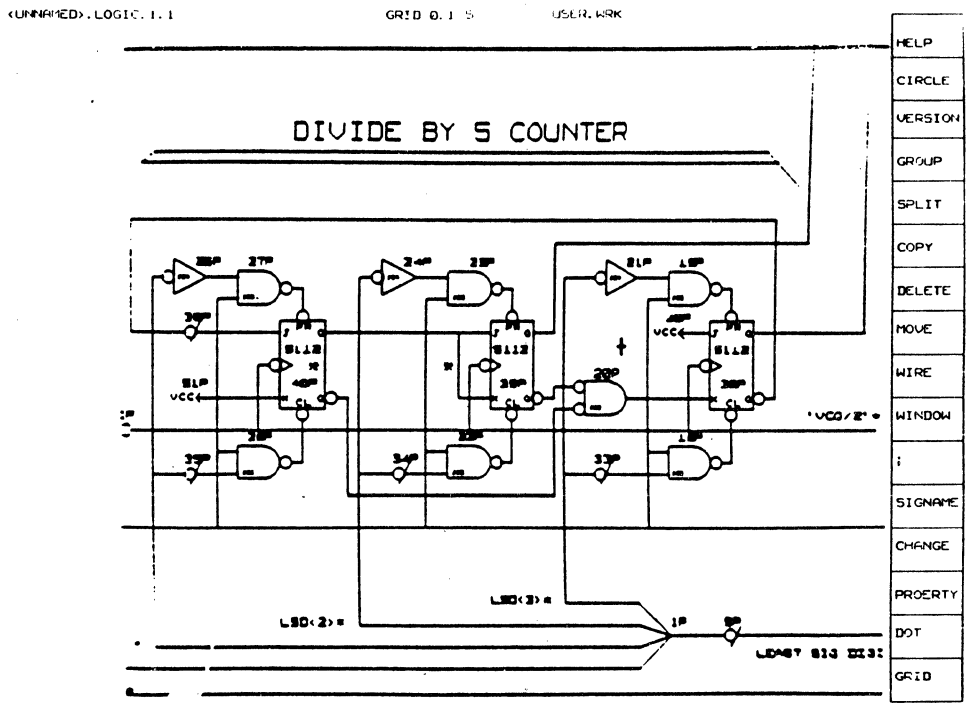


Figure 24. Simultaneous panning and scaling

Since the maximum number of arguments that may be given to the WINDOW command is three, the command is automatically executed upon pressing the yellow button the third time. The result of this operation is shown in Figure 25.

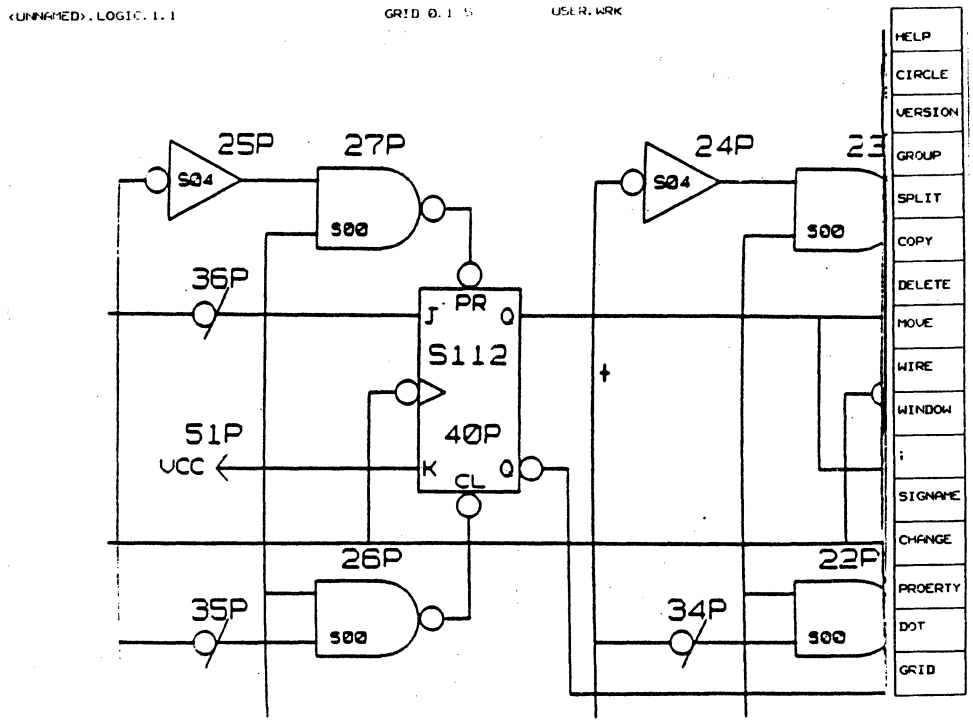


Figure 25. Result of the WINDOW operation

As was the case with the MOVE command described above, it is not necessary to reselect the WINDOW command on the menu if it is already highlighted. This is generally the case with sequences of commands; once the command has been selected, it need not be reselected to execute the command again.

GRID

The Graphics Editor uses a grid to define where an object may be placed. The grid makes it easy to create drawings that are pleasing to the eye. If the grid is too fine (the spacing is very small), then it is difficult to space objects uniformly and to draw wires that are horizontal or vertical. On the other hand, if the grid is too coarse, then it will be impossible to move objects or wires as close to one another as might be desired.

The grid used by the Graphics Editor is specified in terms of inches on the final drawing.

The grid does not refer to size on the CRT display since the you may wish to make the drawing appear larger or smaller on the display. When this is done (using the WINDOW command), the grid scales with the drawing. In other words, things that are placed on a 0.1 inch grid will appear separated by precisely that distance on the final print regardless of the scale that was in effect at the time that drawing was viewed on the CRT.

It is frequently desirable to have the grid displayed on the CRT along with the drawing. If a 1/10 inch grid was being used and if every grid line was displayed, then the drawing would appear as shown in Figure 26. This causes too much clutter on the display; it would be preferable to show, say, every fifth grid line and to leave the remaining lines invisible. The result is shown in Figure 27.

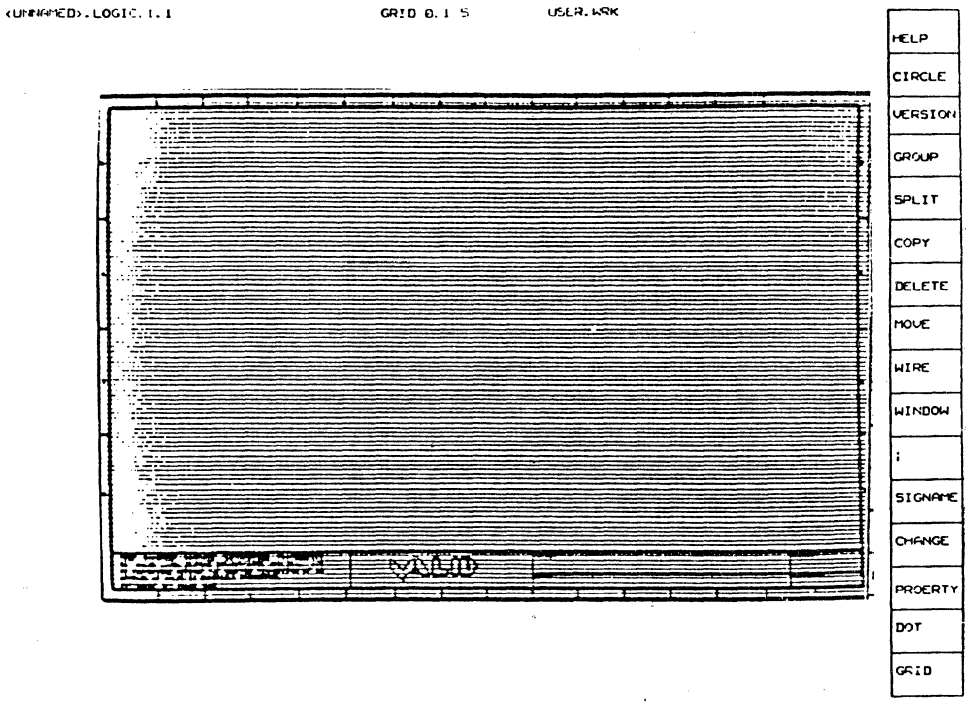


Figure 26. Fine grid displayed

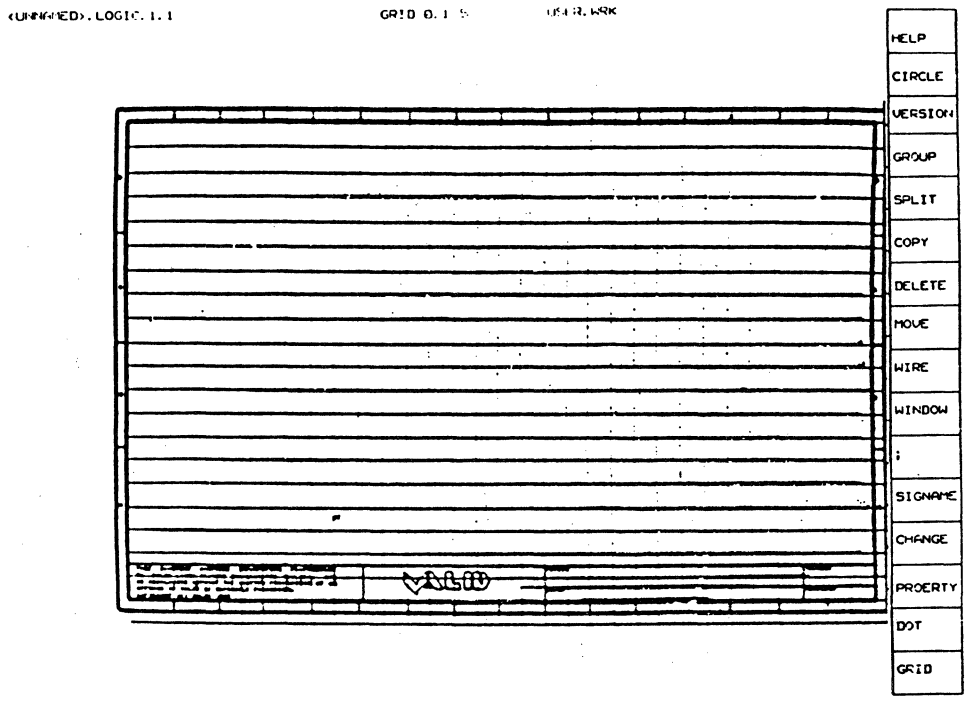


Figure 27. Normal grid for logic drawings

The default grid for logic drawings is a hidden grid line every 0.1 inch and a visible grid line every 0.5 inches. The grid spacing may be changed by typing the command

GRID spacing display ;

For example, if one wishes to change the grid from the default to show every other grid line, one might type on the keyboard:

GRID 0.1 2 ;

This will cause a visible grid line to appear every other hidden grid line or five per inch, as is shown in Figure 28.

We have found that a 0.1 inch grid is the optimal size for logic drawings and a 0.05 inch grid is the optimal size for body drawings where more detail is needed. If the grid size in logic drawings is altered, it is difficult to later connect wires to devices. To see the effects of a coarse grid, we may attempt to place a wire from one point to another with the grid set to 0.5. Note that it is very easy to draw horizontal and vertical lines, but we cannot draw lines that are very close to one another. An attempt to position one end of a line off the grid (Figure 29) will cause it to snap to the nearest grid point (intersection of grid lines) as shown in Figure 30.

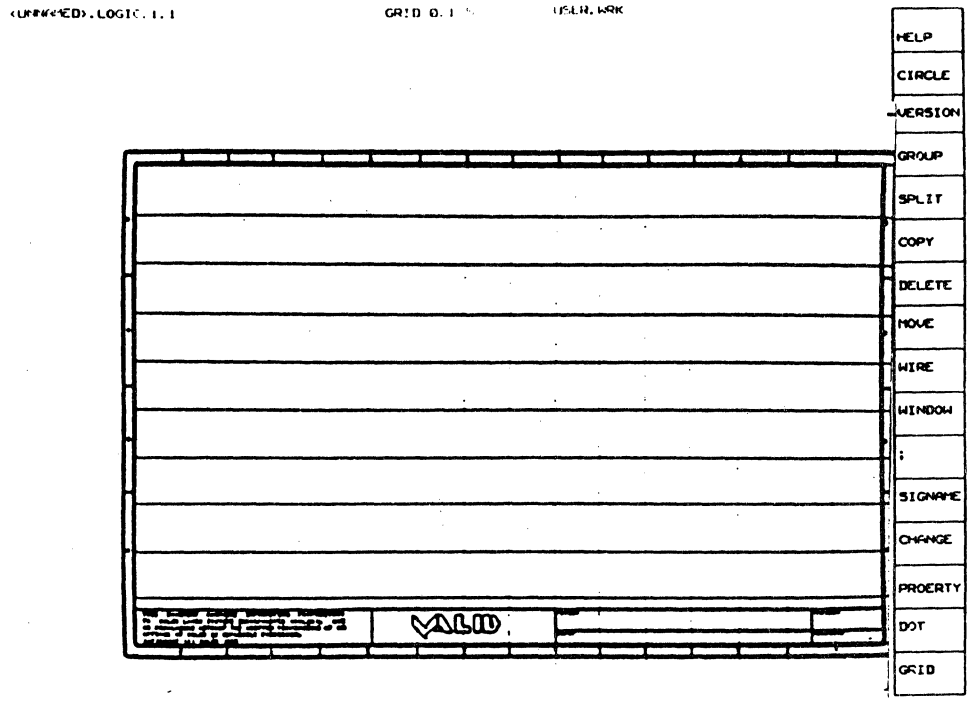


Figure 28. Coarse grid displayed

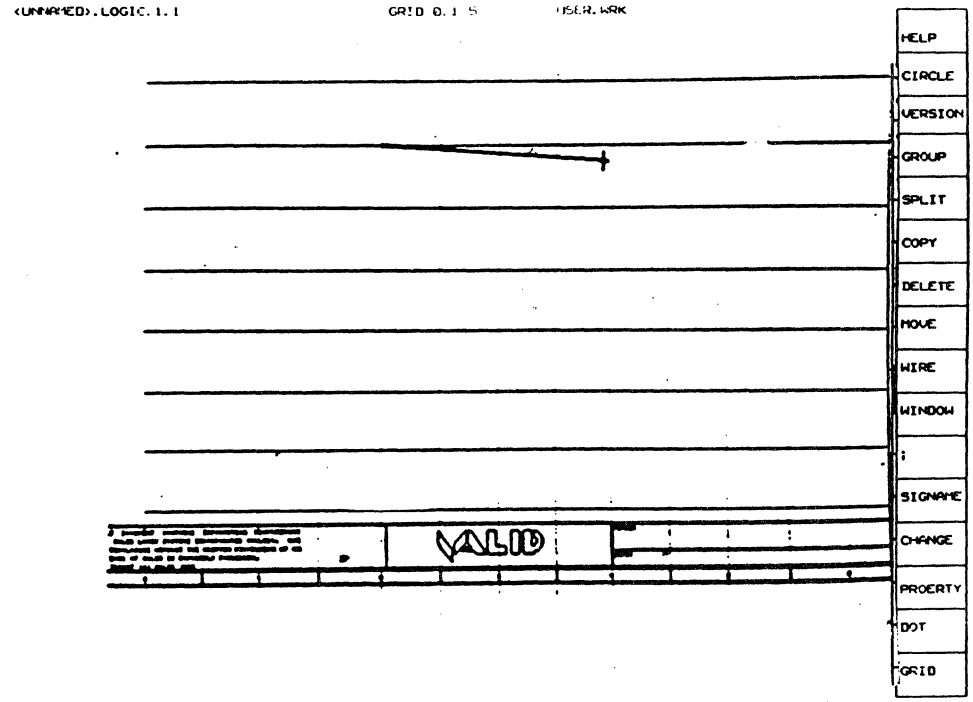


Figure 29. Positioning a wire

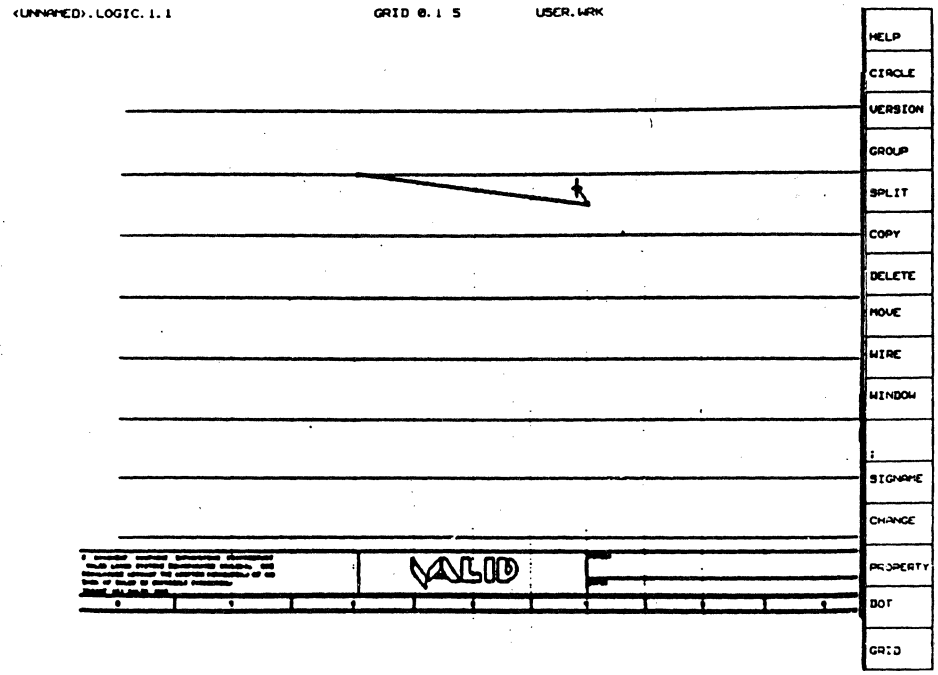


Figure 30. Wire snapped to nearest grid intersection

The use of arbitrary grids should be avoided. It is best to use the default grids exclusively until you are more familiar with the effects of other grid settings.

As a final note on the use of grids, it may be desirable to use a dot grid instead of a line grid. To do this, type:

```
GRID DOTS ON ;
```

3.3 EDITOR PRIMITIVES AND VERTICES

All SCALD drawings are constructed from seven primitives: bodies, wires, signal names, dots, arcs, notes, and properties. Each of these items refers to one or more vertices. They all use one vertex each except for lines which use two and bodies which have one vertex to refer to the body itself and one for each pin. The vertex is normally used to select an object.

Figures 31 through 38 show examples of each of these primitives with the corresponding vertices indicated by arrows. Note that the vertex for any string of text (signal name, note, or property) is at the lower left hand corner of

that string. This is the default position for the vertex of a text string and cannot be changed.

In the discussion of commands above, there was no reference made to vertices; the commands were described in an intuitive manner. In the formal description of each command, the function of those commands will be described in terms of vertices.

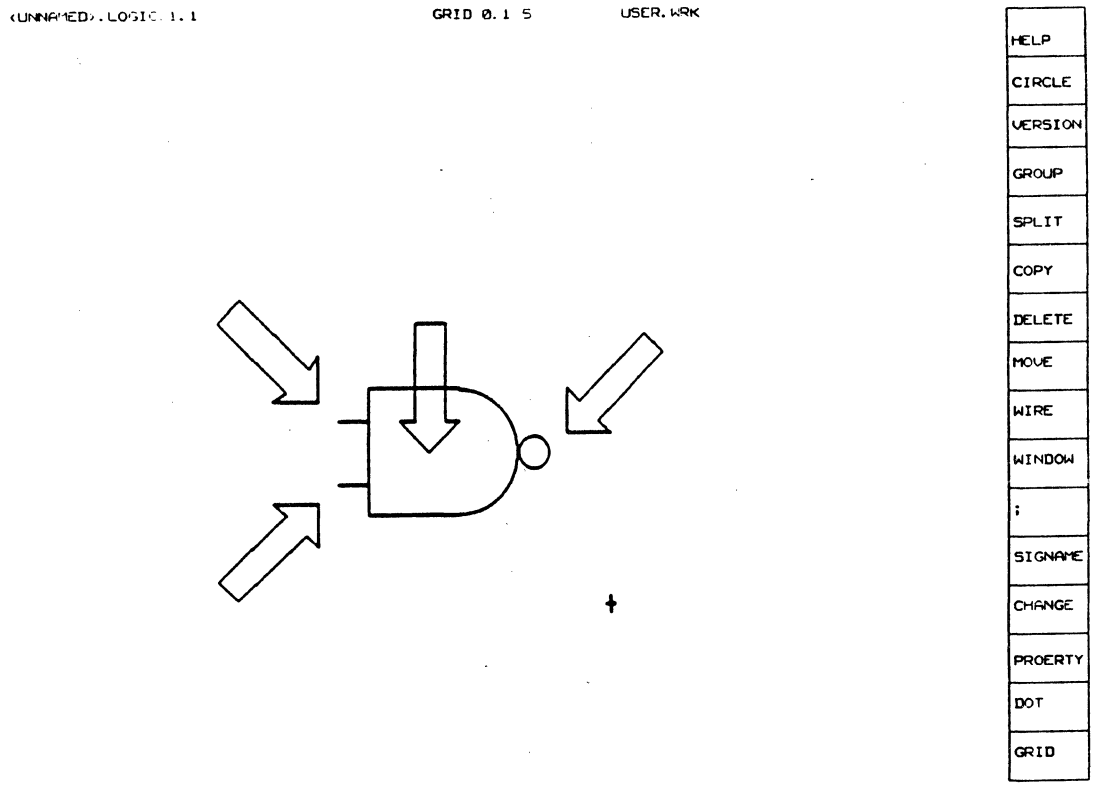


Figure 31. Body -- vertices at center and at pins

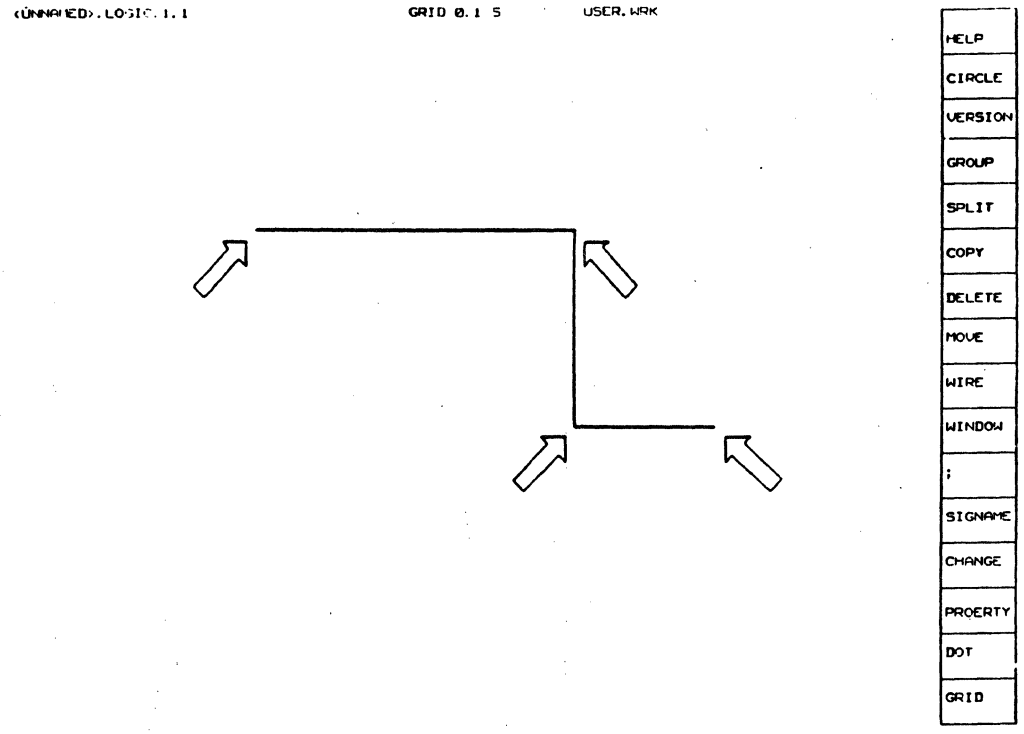


Figure 32. Wire -- vertices at each end and corner

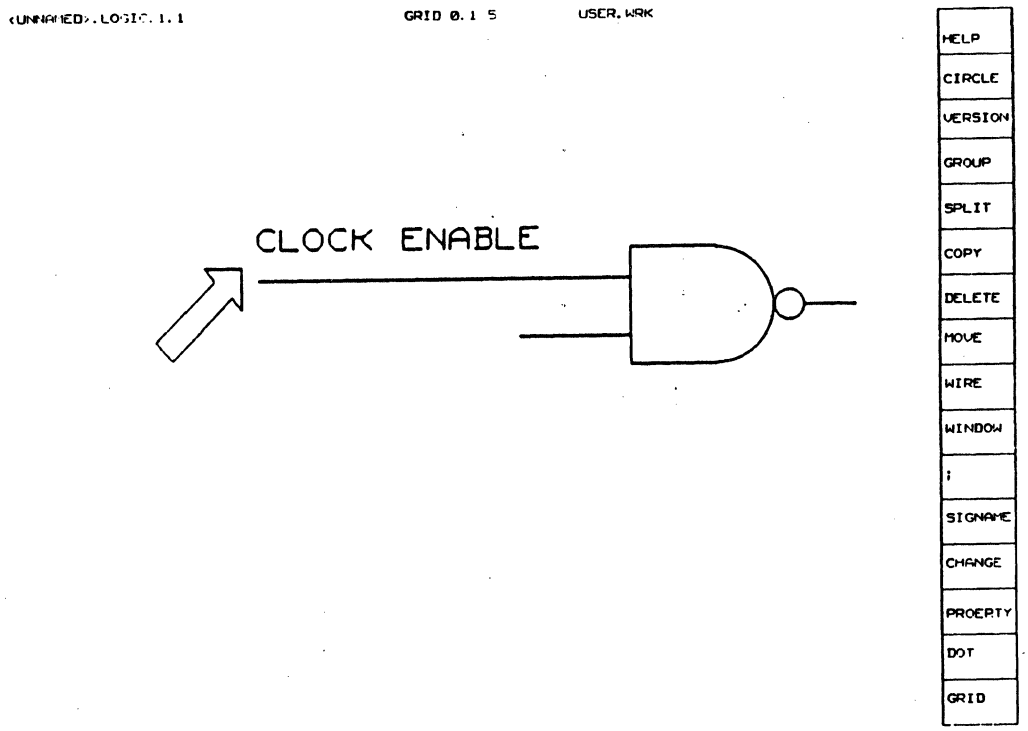


Figure 33. Signal name -- vertex at lower left corner

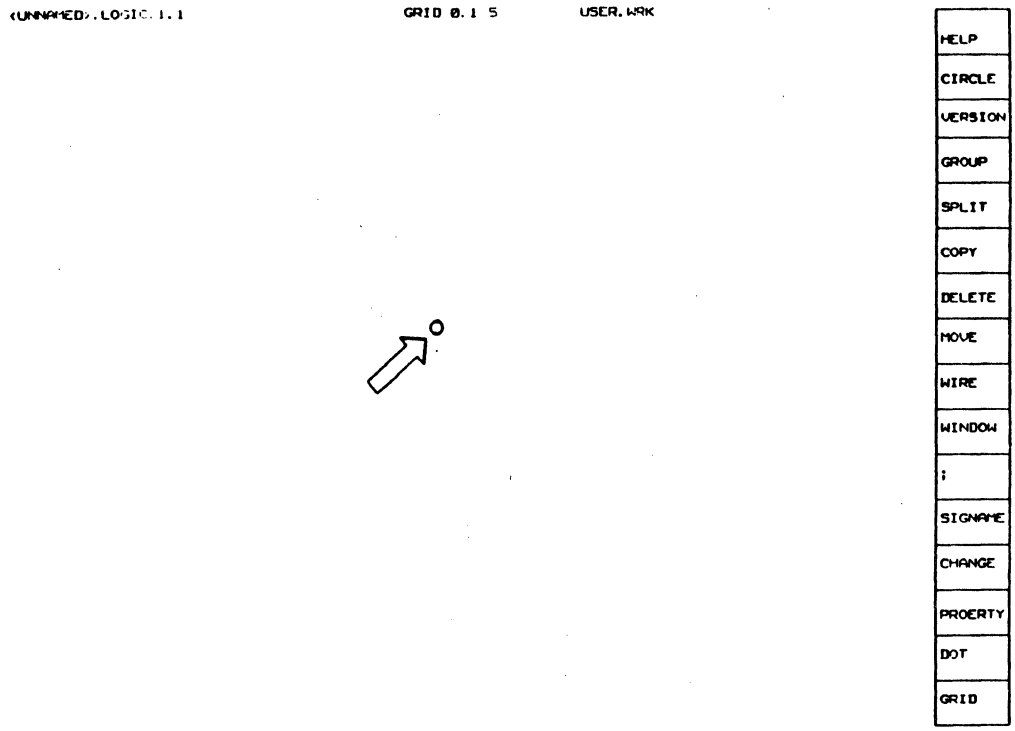


Figure 34. Dot -- vertex at center

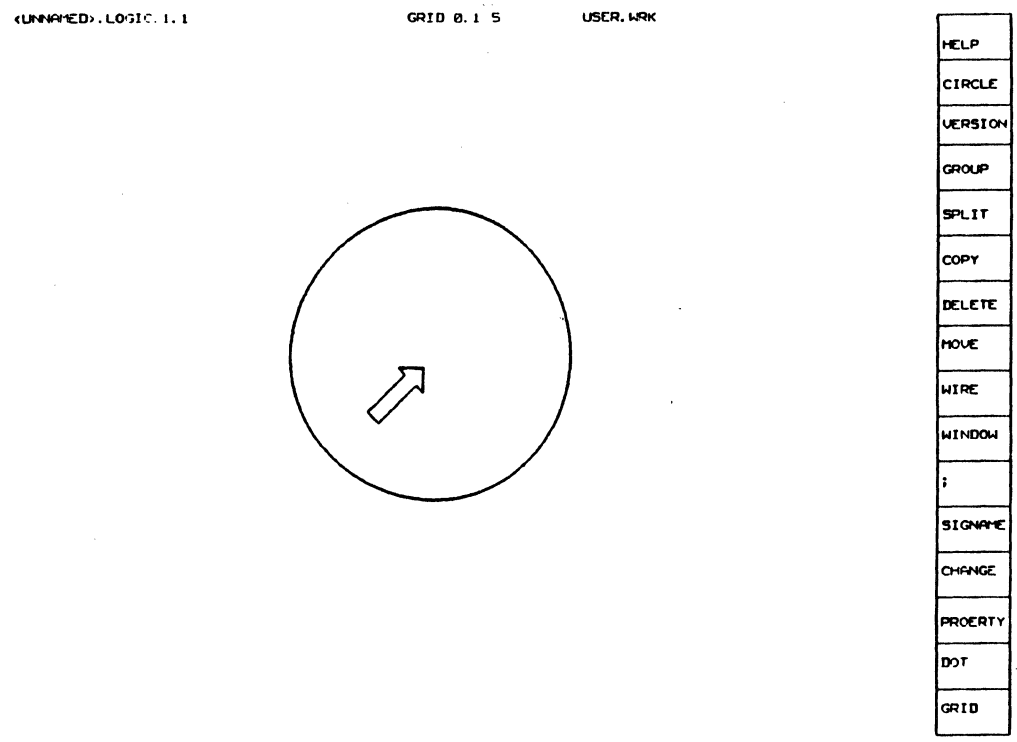


Figure 35. Circle (360 degree arc) -- vertex at center

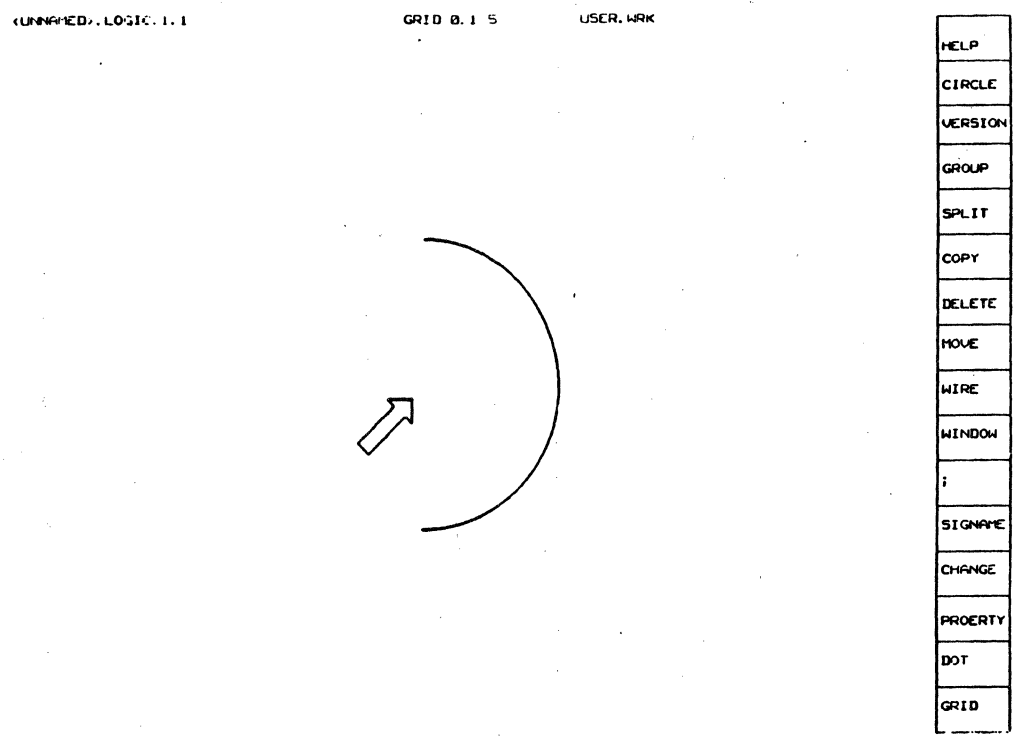


Figure 36. Arc -- vertex at center

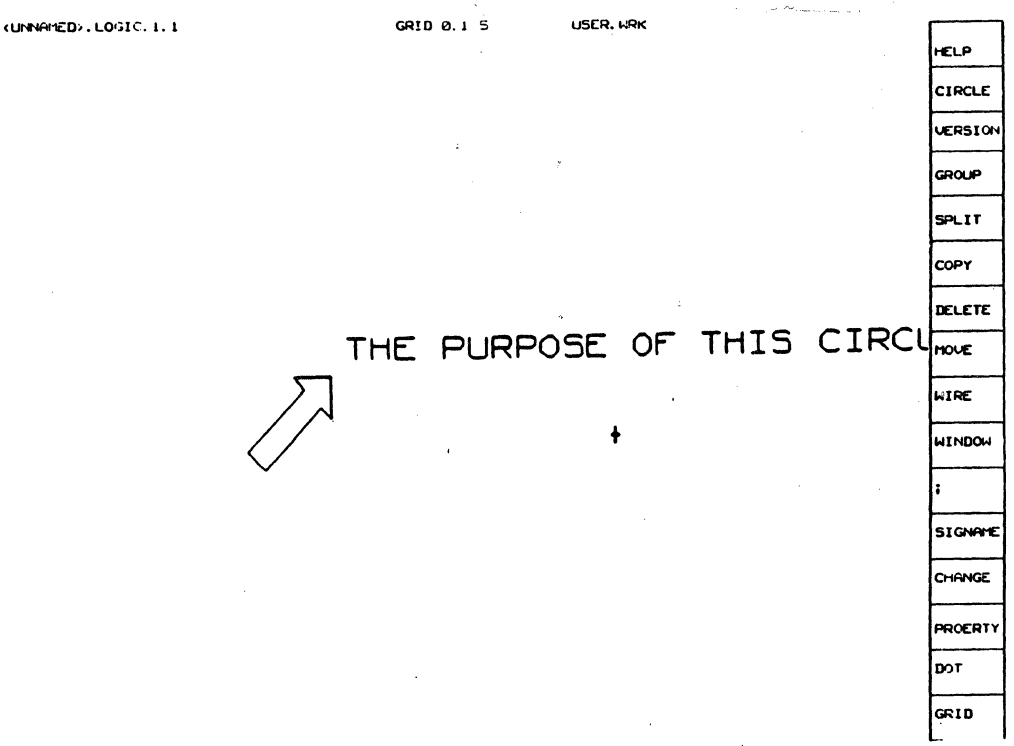


Figure 37. Note -- vertex at lower left corner (default)

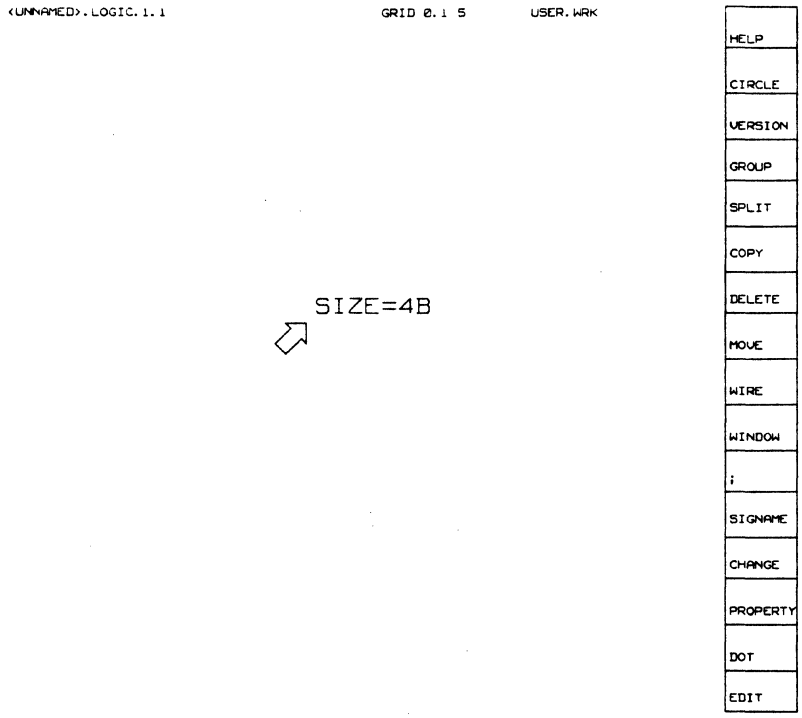


Figure 38. Property -- vertex at lower left corner

3.4 CURSOR AND CURSOR BUTTONS

When one of the cursor buttons on the puck is pressed, the coordinates of a point on the drawing are reported to the Graphics Editor. This is the normal mechanism for indicating "points" to the Graphics Editor commands. The interpretation of the point depends both on the position of the cursor when the button was pressed and the particular button (of the four) that was pressed.

Figure 39 shows how the SCALD Graphics Editor interprets the use of the four different cursor buttons. The yellow and white buttons always use the nearest grid intersection as the point for the operation being performed. The green and blue buttons, on the other hand, always refer to the nearest vertex.

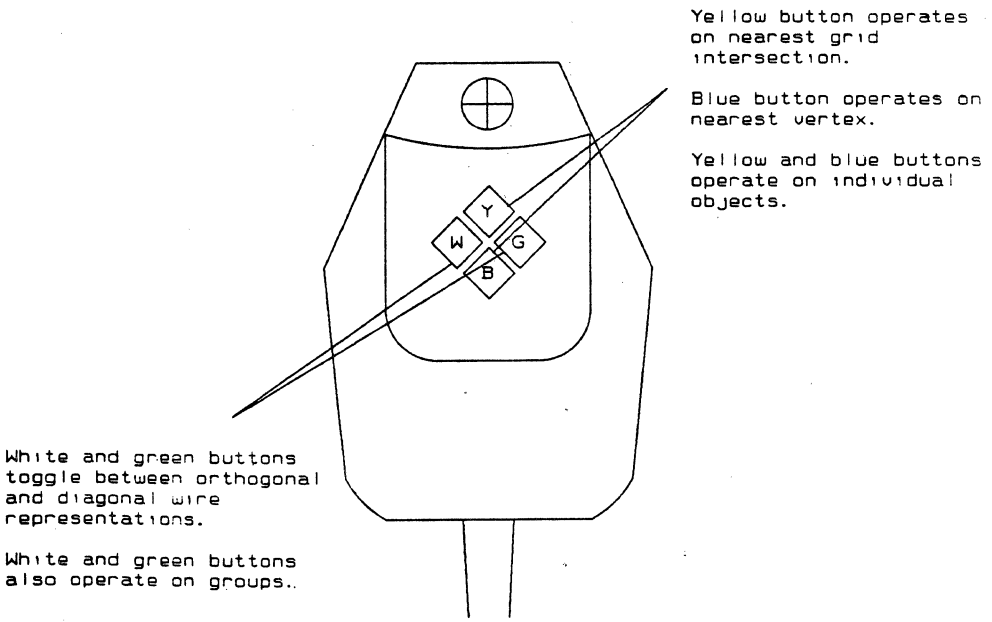


Figure 39. Cursor control buttons

For example, if a wire is being added and the yellow button is used to start that wire, then the Graphics Editor will place the start of that wire at the grid intersection nearest to the cursor (Figure 40).

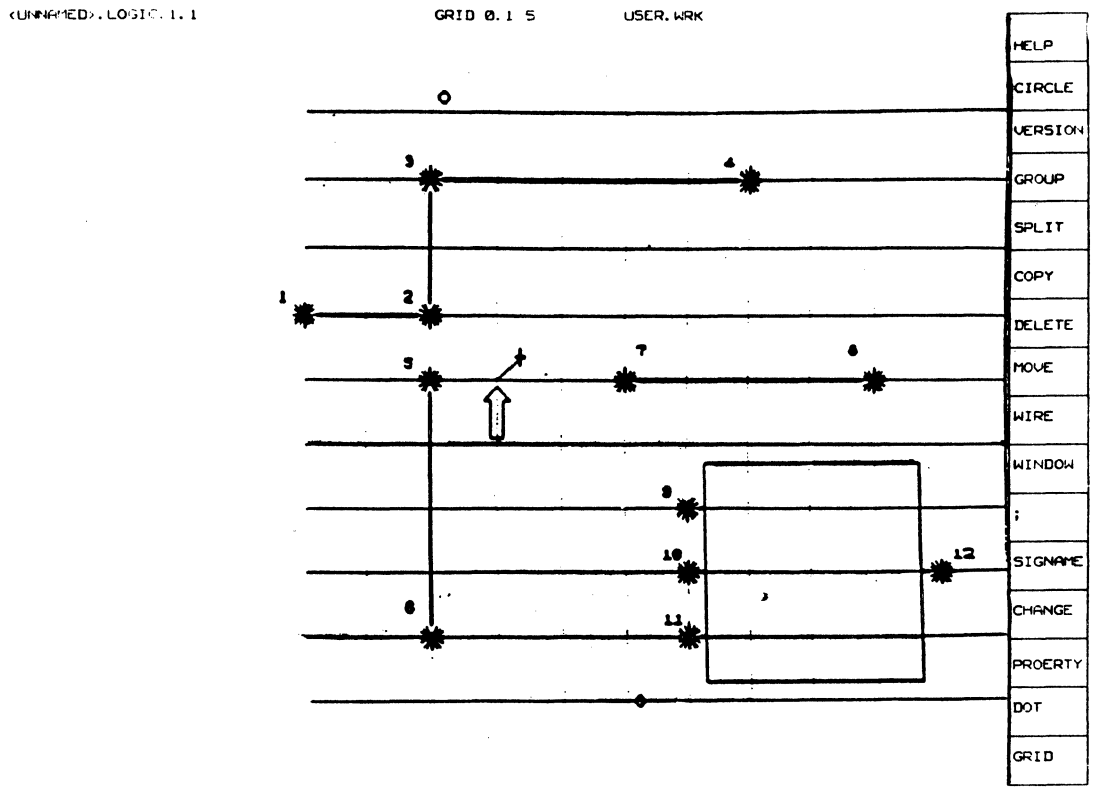


Figure 40. Starting a wire -- nearest grid intersection

Frequently, you wish to start a wire on another vertex. By using the blue button in precisely the same way that the yellow button was used above, the start of that wire will snap to the nearest vertex (Figure 41).

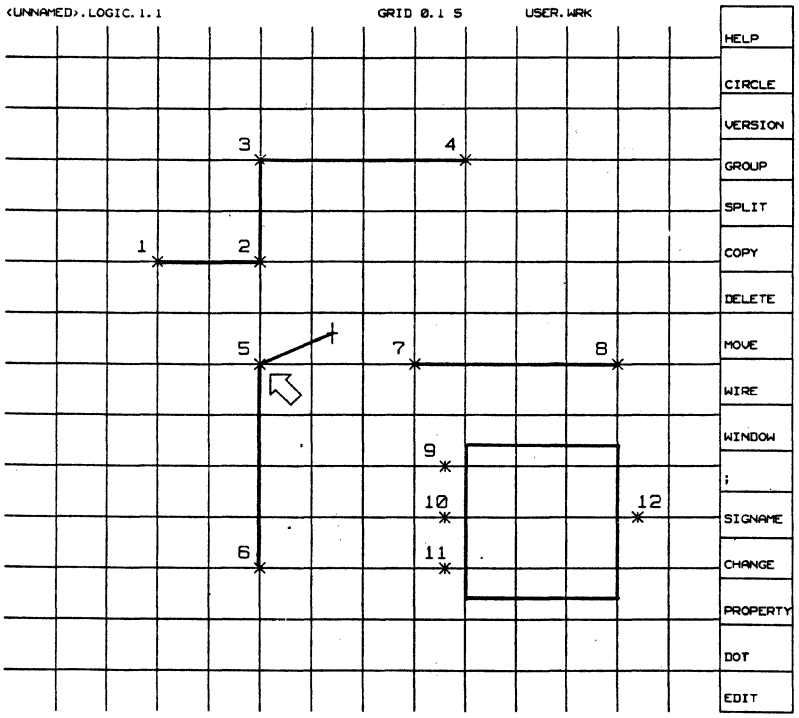


Figure 41. Starting a wire -- nearest vertex

The white and green buttons are used when placing the succeeding points of a wire and are used to fix the direction of wire. Figure 42 shows a NAND gate and a FLIP-FLOP that are to be wired from point A to point B. The wire is started by placing the cross hair cursor at A and pressing the blue button. This attaches the beginning of the wire to the nearest vertex (the output pin of the NAND). The cursor is then moved to point Y. No matter how the puck is moved, diagonally, horizontally or vertically, the resulting wire is always fixed in the horizontal direction. At this point, if the yellow button is pressed, the wire will end at the nearest grid intersection (point Y) and a new segment of the wire can be started. However, Figure 43 shows that if the white or green button is pressed instead of the yellow button, the wire becomes fixed in the vertical direction and the direction of the bend also changes. Again, the wire can be put down at Y by pressing the yellow button. If, instead of putting the wire down, the white button is pressed again, the wire becomes diagonal or mobile in both the horizontal and vertical directions (Figure 44). Figure 45 shows that a fourth press of the white button refixes the wire in the horizontal direction.

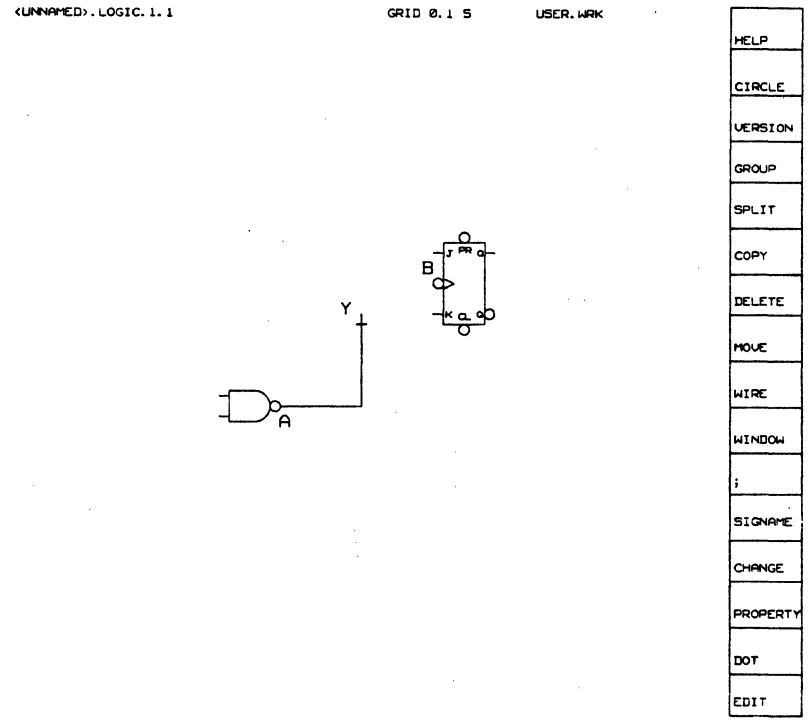


Figure 42. Starting a wire fixed in the horizontal direction

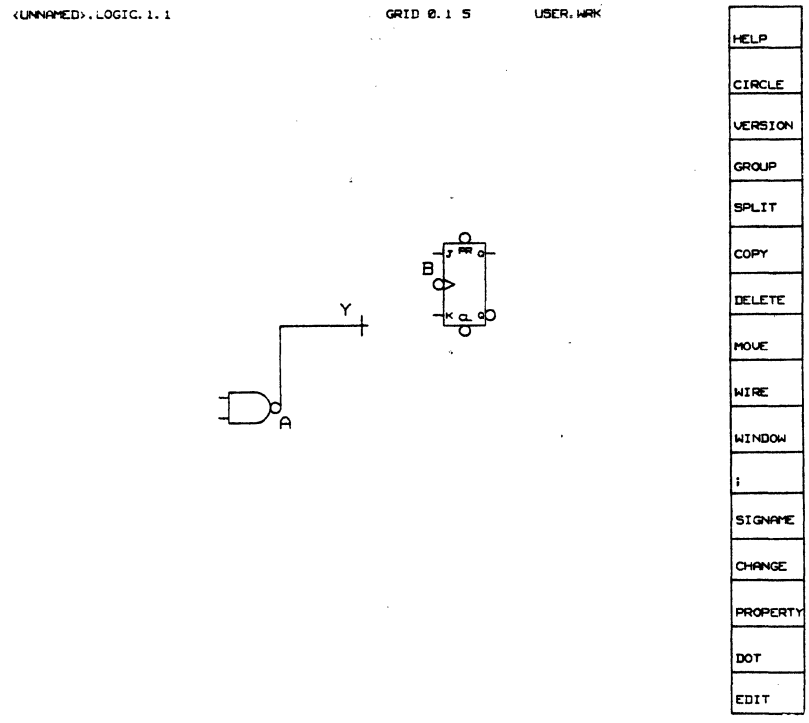


Figure 43. Pressing the white button fixes the wire in the vertical direction

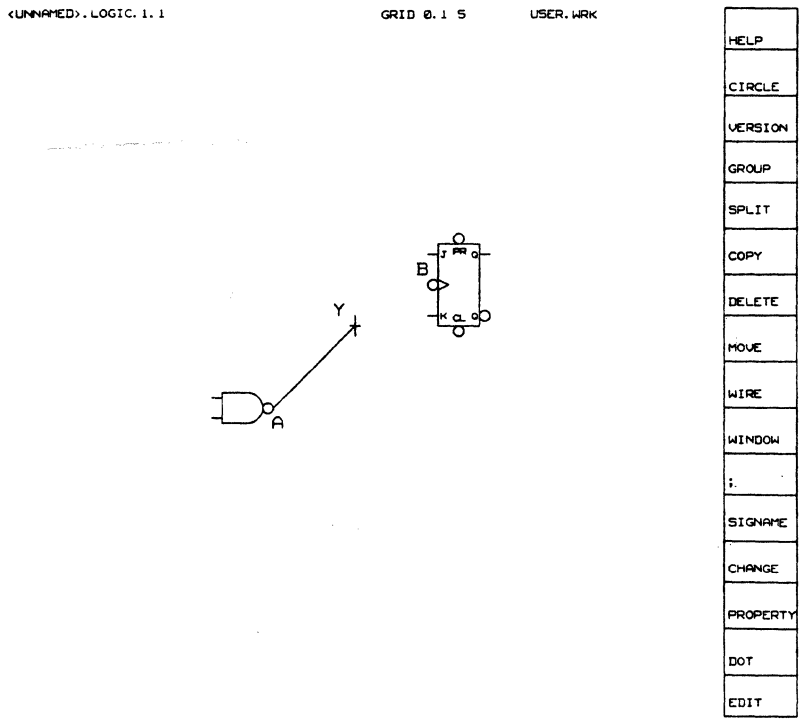


Figure 44. Pressing the white button again makes the wire diagonal

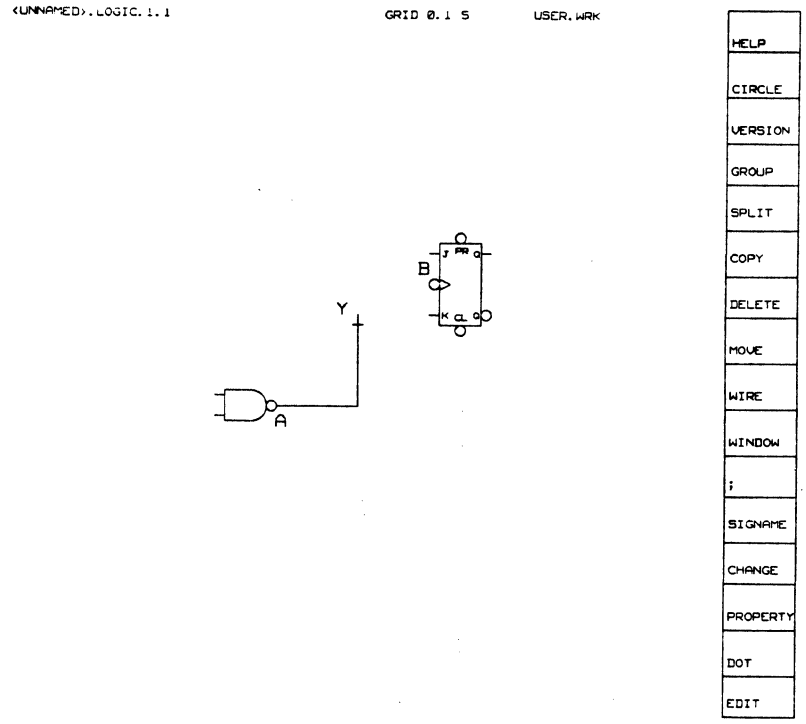


Figure 45. Another press puts the wire back in the horizontal direction

To finish connection point A to point B, place the wire down at Y using the yellow button. Then move the cursor to point B (Figure 46). To change the direction of the wire, press the white button and then snap to the FLIP-FLOP clock pin at B (the nearest vertex) by pressing the blue button. The result is shown in Figure 47.

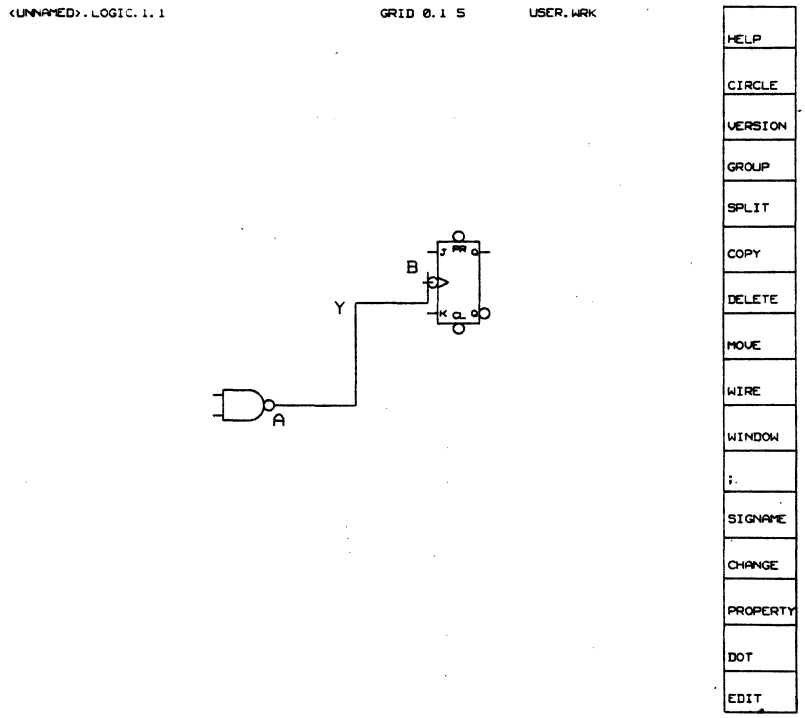


Figure 46. To finish the connection, move the cursor to point B

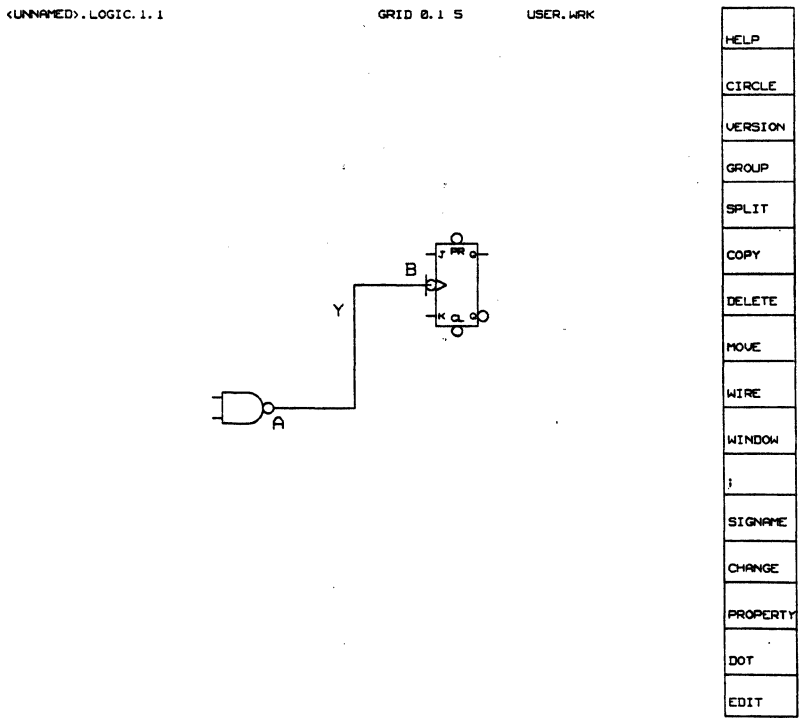


Figure 47. Redirect the wire with the white button and place the wire down with the blue button.

When ending a wire that does not terminate on a pin, the use of the blue button makes it convenient to generate the zero-length segment required to terminate a wire; the cursor need only be moved close to the ending vertex and the blue button pressed.

The remaining distinction between the white and green buttons and their counterparts, yellow and blue, involves operations on groups. If a group has been defined (See GROUP command) and if a group operation is permissible (MOVE, COPY, DELETE), then the white or green button may be used to operate on the group where as the yellow or blue button operates on individual objects or points.

3.5 DRAWING NAMES

The drawing name is used to identify logic drawings and bodies. Drawing names are of the form

```
name.type.version.page
```

The first part of the drawing name is the user-defined identification of the drawing. In general, "name" describes the intended function of the object. Some examples are:

```
ANSI DISK CONTROLLER
32 BIT ALU
LS112
10109
HIGH-SPEED RAM
```

Note that the name is not restricted to short alphabetic identifiers; it may be up to 255 characters long and may contain any printing ASCII character except ".".

The second part of a drawing name, "type", identifies the particular type of drawing. The SCALD III language requires that each piece of a design, that is, each macro, be represented by a body drawing and a logic drawing. The former describes what the item looks like when it appears on a schematic; the latter describes how it is implemented.

Since there are two types of drawings in SCALD III, there are two permissible extensions to the drawing name:

```
BODY
LOGIC
```

If the extension is not specified then the Graphics Editor will use "LOGIC" as the default. Consequently, typing:

```
EDIT 32 BIT ALU
```

would have the same effect as typing:

```
EDIT 32 BIT ALU.LOGIC
```

The next field, "version", is used to identify different versions of a drawing. Generally, you need only concern yourself with drawings that have just one version. There are, however, two cases where versions are of

interest.

In the first case, one may wish to have several different versions of a body. For example, a NAND gate has two representations that the designer may use (Figures 45 and 46). In this example, one of the body drawings is called

LS00.BODY.1

and the other is called

LS00.BODY.2

The use of the different versions of a body will be described below under the "VERSION" command.

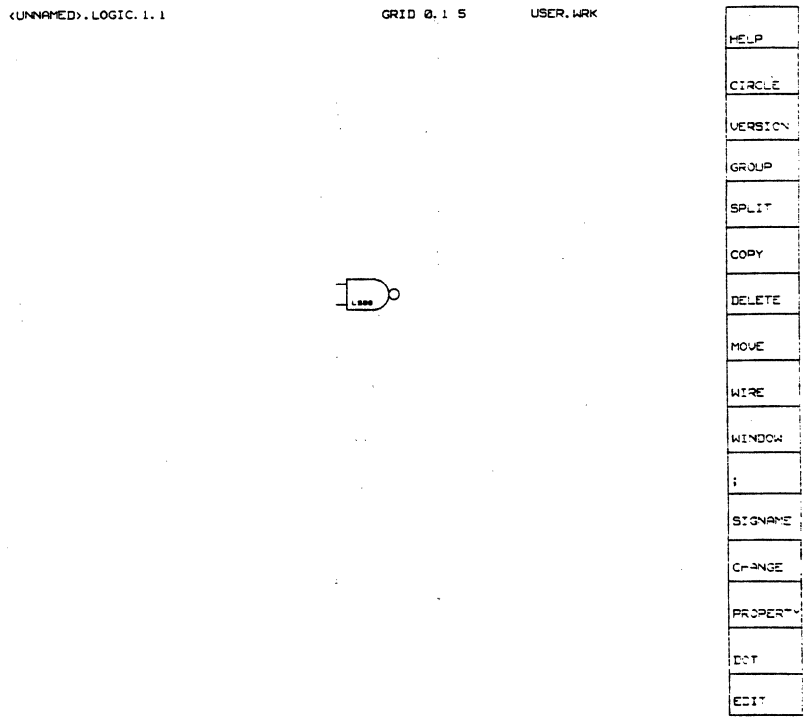


Figure 48. LS00 body -- version 1

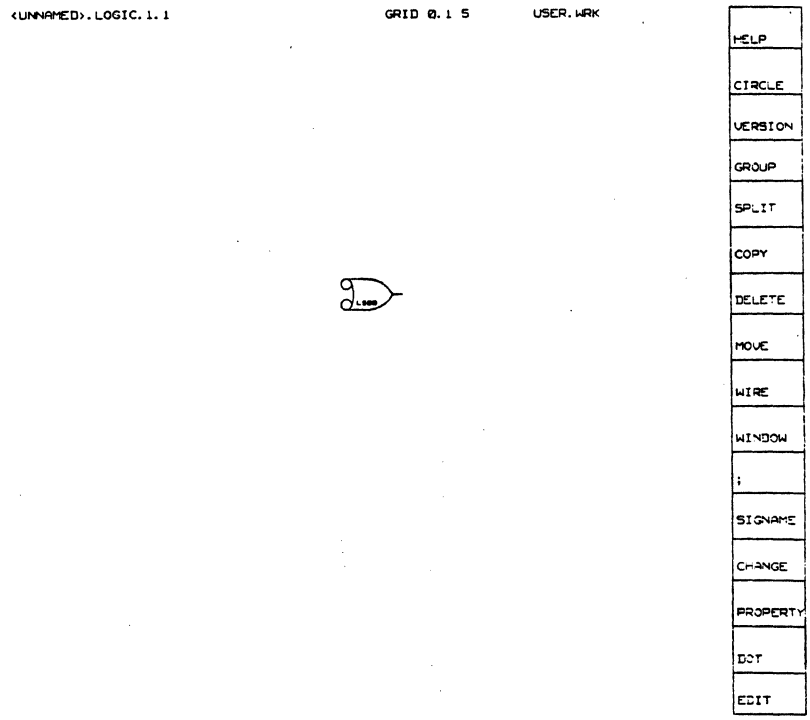


Figure 49. LS00 body -- version 2

The second case where versions of drawings are of interest has to do with the use of select-expressions within a macro (see: SCALD III Language Reference Manual). If the designer wishes to create a macro that is different depending on the value of a parameter that is passed to that macro, then there will be more than one version of that logic drawing.

If the version is not specified, then the Graphics Editor will assume the default version number "1".

The final field, <page>, is used to create drawings that extend over more than one page. This is useful in the case where the amount of logic required to define a particular macro will not fit on a single page.

The default value of <page> is "1".

Table 1 shows some examples of drawing names as typed by you and the fully expanded names assumed by the Graphics Editor.

In the context of EDITing or WRITEing a drawing:	
User types:	**System assumes:**
32 BIT ALU	32 BIT ALU.LOGIC.1.1
NAND.BODY	NAND.BODY.1.1
REGISTER..2	REGISTER.LOGIC.2.1
MUX BOX...4	MUX BOX.LOGIC.1.4
In the context of ADDing a body:	
NAND	NAND.BODY.1.1
LS00..2	LS00.BODY.2

Graphics Editor Command Summary

3.6 INTRODUCTION

Commands are issued to the Graphics Editor either from the keyboard or from the graphics tablet or both. Commands consist of a command_name followed by zero or more operands and zero or more points.

The command syntax may be described by the following:

command_name	[operands...] [points...]

In this representation, lower case words are variables to be replaced by the appropriate value. For example, the command to add a NAND gate to a specific point would be issued by typing

ADD NAND

and indicating the desired point with the puck and cursor control buttons.

The command_name may be specified either by selecting the appropriate command on the menu with the cursor and pressing one of the cursor control buttons, or by typing the command name on the keyboard. Operands, where they are appropriate, are specified from the keyboard. Points may be specified either with the cursor and the cursor control buttons or by typing

(x_integer y_integer)

The range of the x and y integers is -16384 to +16383.

Some commands have a fixed number of arguments (operands and points). For example, the EDIT command has only one operand, the drawing name. More frequently, however, commands may have a variable number of arguments. An example of the latter is the NOTE command; it may have an unlimited number of operand-point pairs as arguments.

In order to represent optional arguments in a command syntax description, the following conventions are used:

1. When fields are enclosed in brackets, "[]", one of the fields may be specified.

2. If fields are enclosed in braces, "{}", one of the fields must be specified.

3. The use of ellipsis, "...", indicates that the preceding may be repeated any number of times.

4. If a sequence of items is enclosed in parentheses, "()", followed by ellipsis, the enclosed sequence may be repeated any number of times.

5. Choices are indicated by stacking:

```
{A}
{B}
```

means that either "A" or "B" may be entered.

6. If a field has a default value, it is shown by being underlined.

For example,

```
command [arg]
```

means that an appropriate argument may be used but is not required. When a choice is available but one must be chosen, then the choices are listed one over the other, enclosed in braces:

```
command {arg 1}
        {arg 2}
        {arg 3}
```

Commands may be classified into several categories, depending on the operation performed:

-File manipulation commands

- DIRECTORY
- DIAGRAM
- EDIT
- FILENOTE
- FORMAT
- GET
- IGNORE
- LIBRARY
- REMOVE
- RETURN
- SCRIPT
- USE
- WRITE

-Commands to add items
- ADD
- AUTO
- CIRCLE
- COPY
- DOT
- NOTE
- PROPERTY
- SIGNAME
- WIRE
- PASTE

-Commands to modify items
- BUBBLE
- CHANGE
- DELETE
- DISPLAY
- MOVE
- REPLACE
- ROTATE
- VERSION
- SPLIT
- SWAP

-Display modification commands
- GRID
- SHOW
- WINDOW

-group manipulation commands
- COPY
- DELETE
- DISPLAY
- GROUP
- MOVE
- CUT
- PASTE

-Miscellaneous commands
- ASSIGN
- BACKANNOTATE
- CHECK
- ERROR
- FILENOTE
- FIND
- FORMAT
- HARDCOPY
- HELP
- MIRROR
- NEXT
- PINSWAP
- QUIT
- REDO

SCALE
SECTION
SHOW
SET
UNDO
VECTORIZE

These commands are described in the following sections.

3.7 COMMANDS

ADd	[<directory>]body_name[.[type][.[version]]] point...
	BODY 1

The ADD command is used to add bodies to logic drawings. The body_name refers to the name of the body drawing which is to be added. The type may be specified, but is not required; since adding a logic drawing to another drawing is not permitted, the drawing type always defaults to BODY. The version defaults to "1", but any version of a body may be added provided, of course, that it exists. If no directory is specified, each SCALD directory in the list is searched until the device with name body_name is found.

If an attempt is made to add a body that does not exist, then the message:
body_name type WITH VERSION version DOES NOT EXIST
will be displayed.

Once the body has been added to the drawing, additional copies can also be added by picking up a copy of the body by pressing the yellow cursor button and then placing it down in the desired place by pressing the button again.

Because the Compiler doesn't allow time or simulator devices in logic drawings, time and sim devices cannot be added to logic drawings. Similarly, sim devices cannot be added to time drawings, etc. The command "DIR <*>" will tell whether any of the SCALD directories in your list are of the wrong type for the currently edited drawing (e.g. using the TIME library when editing a LOGIC drawing). If you try to ADD a body from an illegal library, then the message "Device X is not legal in this drawing" will be displayed.

To substitute one body for another, see the REPLACE command.

ASsign	key quoted-string

The ASSIGN command assigns a text string, contained in quotes, to a function key. The special key must be one of the program function keys (LF2 - LF9, RF1 - RF12 and TF1 - TF6). The text string may be any text of less than 60 characters. For example, to assign LF6 to "DIS 2.0", type:

```
ASSIGN <press LF6 key> "DIS 2.0"
```

After an assign, pressing the function key is identical to typing in the string. A RETURN is always appended to the end of the assigned string. You should not reassign the keys defined for the SCALDsystem (LF1, LF10, LF11 and LF12). The current assignments can be displayed with the SHOW KEYS command.

The default values for the function keys is in /u0/editor/softkeyassign. The entries are similar to using the ASSIGN command in the Graphics Editor, but instead of a key press, the actual value of the key is given. For the example using LF6 above, the entry in the file would be

```
ASSIGN ~@ "DIS 2.0"
```

Keys are assigned these default values when the Graphics Editor is invoked. Users can define their own softkeys by putting ASSIGN statements their startup.ged files. The values for all special keys are given in the table on the next page.

		(SUPER)	(HYPER)	(META)		
	ALONE	LF10	LF11	LF12	CONTROL	SHIFT
LF2	~@!	~D!	~H!	~P!	~B!	~A!
LF3	~@"	~D"	~H"	~P"	~B"	~A"
LF4	~@#	~D#	~H#	~P#	~B#	~A#
LF5	~@$	~D$	~H$	~P$	~B$	~A$
LF6	~@%	~D%	~H%	~P%	~B%	~A%
LF7	~@&	~D&	~H&	~P&	~B&	~A&
LF8	~@'	~D'	~H'	~P'	~B'	~A'
LF9	~@(	~D(	~H(	~P(	~B(	~A(
TF1	~@8	~D8	~H8	~P8	~B8	~A8
TF2	~@9	~D9	~H9	~P9	~B9	~A9
TF3	~@:	~D:	~H:	~P:	~B:	~A:
TF4	~@;	~D;	~H;	~P;	~B;	~A;
TF5	~@<	~D<	~H<	~P<	~B<	~A<
TF6	~@=	~D=	~H=	~P=	~B=	~A=
RF1	~@,	~D,	~H,	~P,	~B,	~A,
RF2	~@-	~D-	~H-	~P-	~B-	~A-
RF3	~@.	~D.	~H.	~P.	~B.	~A.
RF4	~@/	~D/	~H/	~P/	~B/	~A/
RF5	~@0	~D0	~H0	~P0	~B0	~A0
RF6	~@1	~D1	~H1	~P1	~B1	~A1
RF7	~@2	~D2	~H2	~P2	~B2	~A2
RF8	~@3	~D3	~H3	~P3	~B3	~A3
RF9	~@4	~D4	~H4	~P4	~B4	~A4
RF10	~@5	~D5	~H5	~P5	~B5	~A5
RF11	~@6	~D6	~H6	~P6	~B6	~A6
RF12	~@7	~D7	~H7	~P7	~B7	~A7

AUto	{ Path } { Dots }

The AUTO command is used to automatically assign certain objects to drawings. The PATH option assigns a PATH property to each body on a drawing. Each body is examined to make sure that it does not already have a PATH property and that it is not a "special" body. (Special bodies are those bodies that have special meaning to the SCALDsystem.) Each remaining body is then assigned a property "PATH = nP" where n is a unique integer. For each device on the drawing, AUTO PATH places the property directly over any visible body property closest to the body origin.

The use of the PATH option makes it unnecessary for the user to manually assign the PATH property to each body. The use of the PATH property by the SCALDsystem is described in the SCALD III Language Reference Manual.

If a librarian wishes to place a the PATH property on a device when defining the body, the property "PATH = ?P" can be added to the body definition. The "?" acts as a place holder so that when the body is added to a drawing, the AUTOPATH command replaces the "?" with a number.

Whenever a drawing is written, it is automatically auto pathed. When bodies are copied, the path property is not copied.

The DOTS option places a dot at each connection point in the current drawing. The user has the option of having these dots displayed in an open or filled mode. This option is specified with the SET command.

BAckannotate

The BACKANNOTATE command allows you to annotate your designs with information from the Packager. The Graphics Editor reads a schematics annotation file produced by the Packager. To generate a back annotation file for the Graphics Editor, use the following Packager directive when running the Packager:

```
output backannotation;
```

You have the option of backannotating location designators, pin numbers, and physical net names. The backannotation file generated by the Packager, pstback.dat, must be renamed to backann.cmd for the Graphics Editor.

After creating the file backann.cmd, start up the Graphics Editor and type the command BACKANNOTATE. The file backann.cmd must be in the current UNIX directory. The Graphics Editor reads the file, edits each named drawing in turn, adds the appropriate physical information and finally writes the drawing.

The annotated properties added by the Graphics Editor are "soft" properties. This means that the property names begin with $ (e.g. $LOCATION) and are not written into the connectivity file. By not including them in the annotated properties, the Packager can reassign the physical information when it is run again. In addition, soft properties can only be moved or deleted, not changed or added. To make a soft property a hard property, simply add the same property name, minus the $ (e.g. if a component has a $LOCATION property, add a LOCATION property).

The PN (pin number) property is special to the Graphics Editor and cannot be added to any part on a drawing. However, PN placeholders can be incorporated into device definitions just as PATH property placeholders. The PN property with the value '?' can be attached to the pin when defining the body. This placeholder is just a location and size indicator; when the pin on the instance of the device is backannotated, GED looks to see if there is a PN property on the pin and replaces it with the $PN property at the same location. Unlike the PATH placeholder, when the device is added to a drawing, the PN property does not appear (this is true of all pin properties on device definitions). Additionally, if the PN property on the pin has a value of '?', the property is not written into the connectivity file.

BUbble	point...

The BUBBLE command permits you to change the state of a pin from "bubbled" to "unbubbled" and visa-versa, provided that the body has been defined to permit this conversion.

For example, an inverting buffer may be drawn with the bubble either on the input or the output pin (but not both). If, in the definition of the body, the pins were established as part of a BUBBLE_GROUP, then the BUBBLE command may be used to convert the body from one form to another. Instructions for defining bubble groups follows this Command Summary in a section titled, "Defining Bubble Groups in the Graphics Editor."

CHAnge	point...

The CHAnge command invokes an editor to alter text strings. The text strings indicated can be edited using the line editor or the UNIX screen-oriented editor VI. While simple changes can be made with the line editor, VI is preferable for editing many lines of text.

The Line Editor

The text string selected is displayed in the bottom left side of the CRT above the input line. The text strings are chosen with the puck and are altered with control keys. As many strings as necessary can be indicated at one time, and each one is edited in turn.

The line editor uses a vertical-line cursor; the cursor may be moved by typing any of the following control-key combinations:

- control-F to move forward one character,
- control-B to move back one character,
- control-E to move to the end of the line,
- control-A to move to the beginning of the line.

To delete a character, position the vertical-line cursor just to the left of the character and type:

- control-D to delete one character, and
- control-K to delete the remainder of the line.

To search for specific characters in the text, use:

- control-S to search to the right of the cursor's position.

Type the character being searched for and a carriage return immediately after typing control-S.

- control-R to search to the left of the cursor's position.

Again, type the character being searched for and a carriage return immediately after typing control-R. Both of these commands are case sensitive ('n' is not the same as 'N').

To repeat a command a given number of times, type:

- control-U [number] command

For example, typing "control-U 6 control-D" deletes six characters. If no number is given, the default is four.

To insert text to the right of the cursor, simply type the characters to be inserted, followed by the return key. If the character to the right of the cursor is to be deleted at the same time that one or more characters is to be inserted, then the user has the option of typing control-D rather than the return key.

To reposition the text after it has been modified, type:

- control-X to exit from the line editor.

It is necessary to exit from the line editor in order to execute Graphics Editor commands. As explained above, control-X quits the text editor; in addition, there are two other ways to quit the editor:

- Pointing to another piece of text repositions the current text and edit the new piece of text.
- Control-Z aborts any changes to the text currently in the edit line and repositions the original back onto the drawing. The use of control-Z within the line editor overrides its usual function for job control.

The help file for the line editor may be displayed while in line-edit mode by typing:

- control-I for immediate help.

The VIsual Editor

Those who are new to VI may find it helpful to create command files with the line editor; those who have used VI in the C-shell may prefer to use VI because it is a full-screen editor. While in VI, the user will not be able to use the puck or the SCALDsystem softkeys.

To invoke VI, type control-V. Then, using VI commands, edit the strings. Lines can be lengthened, shorted and moved. VI has separate documentation in a chapter in the UNIX Manual. Those who are new to the editor should bear in mind that the program has two modes, one for commands and the other for text entry. A carriage return, for example, does not open a line for new text unless the user is in the insert mode. The insert mode is entered by typing either i for insert, a for append, o for opening a line below the current line, or O for opening a line above the current line.

When the user exits from VI, the changes are put into the drawing. To exit from VI, type either of the following commands:

- :wq
- ZZ

It is possible to exit and abandon all changes by typing :q! while in the command mode.

To protect users from making illegal changes to properties, the strings are labeled. Properties are labeled with !PROP (for user-added properties) and !DEF (for default or body properties). Both the name and value are given for that property string. If a default name is changed, the old name is replaced after exiting VI.

Lines added to the end of the list of strings are made into notes and added, consecutively, in one-grid-space increments after the last item in the list. If lines are deleted while in VI, the last set of notes and properties in the list are not changed.

This page has intentionally been left blank.

CHEck

The CHECK command checks a drawing for several problems which are difficult to see but will result in compile errors. These include:

1. Wires which are connected, but which do not appear to be connected

2. Pins attached to more than 2 wire segments

3. Two identical components in the same place

4. Wires connected to only one pin and not named (NC wires)

5. Nets that are named but not connected to any pins

All of the parts in a drawing are also checked for path properties. If any errors are detected, they are reported and their location is saved. You can step through the errors with the ERROR command.

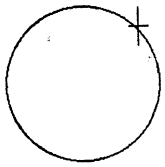
CIrcle	(center_point radius_point [arc_point])...

The CIRCLE command is used to place both circles and arcs. To place a circle on the drawing, the first point is used to define the center and the second point, the radius. As soon as the second has been specified, the circle will appear.

The use of circles and arcs is rarely necessary on logic drawings. Both, however, are commonly used when creating body drawings.

An arc is defined by a center, a radius, and starting and ending angles. To draw an arc, use the CIRCLE command followed by a center point:

and then the radius. Notice that the center star disappears and the completed circle appears as soon as the radius point is specified.



If an arc is desired, the point designating the radius is also the starting angle. The next point given determines the final angle of the arc (counterclockwise) and, of course, must be on the circle. In the figure below, the cursor is moved along the dotted path and the resulting arc is created.

```
COpy|({source_pt} destination_point [propertyreattach_point])
    |({group_name}                                          )...
```

The COPY command may be used to copy any object or group of objects for use elsewhere on the same drawing. The first point identifies the object which is to be copied, and the second identifies the point at which the new copy should be placed.

If the yellow button is used to copy an object, the cursor position serves as a reference point for positioning the copy. When the blue cursor button is used, one vertex of the copy is snapped to the cursor.

The COPY command operates on groups as well as on individual objects. If either the white or the green cursor button is used to pick up an object, the group nearest the current cursor position will be copied. If the green button or the group_name is used, the group will snap to the cursor and only a destination_point needs to be given. If the white cursor button is used to indicate the group to be copied, then the current cursor position is used as a reference point to position the group. Individual properties can be copied by selecting the property to be copied, the new location for the property and the new owner. Once the new location has been specified, a rubber band line is drawn from the cursor to the property so that the new owner can be selected more easily. The new owner can be a device, a pin, a wire or a signal name. Default body properties and those properties generated by the SECTION, PINSWAP and BACKANNOTATE commands cannot be copied.

Default properties and user added body properties are included in copies made of devices. Properties that aren't copied with the body are PATH, those properties generated by the PINSWAP, SECTION and BACKANNOTATE commands, and pin properties. Wire properties are not copied when a wire is copied.

To copy objects or groups from one drawing to another, see the CUT and PASTE commands.

CUt	{ point } { group_name }

The CUT command, in conjunction with the PASTE command, allows objects to be copied from one drawing to another. Objects to be copied are placed in a "cutting" buffer by typing CUT and selecting the object (with the yellow button) or the group (with the white button) to be cut. The cutting buffer can only contain one group or object at a time. The CUT command highlights the object or group selected and also gives the number of objects, wires and notes put into the buffer.

Default body properties and user added body properties are included in copies made of devices. Properties that aren't copied with the body are PATH, those properties generated by the PINSWAP, SECTION, and BACKANNOTATE commands, and pin properties. In addition, wire properties are copied when a wire is cut; usually the cut buffer is transferred to another drawing and the designer wants to copy the signal names to the new drawing.

DElete	({point}) ({group_name})...

The DELETE command is used to remove objects from a drawing. The DELETE command operates on the geometrically closest object; a line or arc may be deleted by pointing to any point along that line or arc.

An entire group may be deleted by selecting either the white or green cursor control button or by identifying the group by name. The group nearest the current cursor position will be deleted.

Default properties on bodies cannot be deleted by the user once an instance of the body has been added to a drawing. In addition, pin number properties (PN) generated by the PINSWAP command cannot be deleted.

DIAgram	[<directory>]drawing_name[.[type][.[version][.[page]]]] LOGIC 1 1

The DIAGRAM command is used to change the name of the current drawing. For example, you may wish to use the drawing SHIFTER.LOGIC as a pattern for a new drawing NEW SHIFTER.LOGIC. You'd type:

```
EDIT SHIFTER
```

and then change the name by typing

```
DIAGRAM NEW SHIFTER
```

The DIAGRAM command also allows you to save a copy of a drawing under a different name before making any significant changes.

The DIAGRAM command does not write the new name into a SCALD directory; you must explicitly do this.

```
DIRectory |[<directory>][drawing_name][.[type][.[version][.[page]]]]|
                                *
```

The DIRECTORY command is used to list the names and contents of the SCALD directories in the current directory list (see USE, LIBRARY and IGNORE). There is no limit to the number of SCALD directories in use at one time. The DIRECTORY command displays the contents in the order that the directories are searched, with the current working directory displayed first.

Wild card characters are permitted in directory names and drawing names. A '*' matches anything, and a '?' matches any single character.

If only the drawing_name is given (e.g. <100K>100102, <100k>1001* or <100k>*), then only the drawing_names will be listed and no drawing_types (e.g. only 100102 and not 100102.LOGIC.1.1, 100102.BODY.1.1, etc). However, if a drawing_type is specified, then only the drawings of that type will be listed. Some examples are:

DIR <*>	lists all directories (but no drawing_names)
DIR *	lists all drawing names in the current directory
DIR <cr>	same as DIR *
DIR ls*	lists all drawing names beginning with LS in the current directory
DIR *.body*	lists all bodies in the current directory
DIR <time>*	lists all drawing_names in directory time
DIR <*>*	lists all drawing_names in all active directories

```
DISplay    | ( { Name        } { point }              )
           | ( { Value       } { "group_name" } ... )
           | ( { Both        }                       )
           | ( { Invisible   }                       )
           |
           | ( { Default     }                       )
           |
           | ( { real_number }                       )
           |
           | ( { Heavy       }                       )
           | ( { Thin        }                       )
           |
           | ( { Pattern number }                    )
           |
           | ( { Filled      }                       )
           | ( { Open        }                       )
           |
           | ( { Left        }                       )
           | ( { Right       }                       )
```

The DISPLAY command is used to change the way objects are displayed on a drawing. DISPLAY can be given a point specifying either a single item or a group or it can be given a group name. If a group name is given, it must be quoted. For instance, the following line makes all properties in group A invisible:

```
DISPLAY INVISIBLE "A"
```

The group can contain any type of item. The DISPLAY command will select the correct type of items to change.

The first four options, Name through Invisible, deal with the way property values are displayed on the drawing.

Properties consist of name-value pairs; one may wish to display the name alone, the value alone, or both, or neither the name nor the value. When a property is added to a drawing, just the value of that property appears. In order to change the display, type the command DISPLAY followed by a space, followed by the letter N (name), V (value), B (both), or I (invisible), and select one or more properties using the cursor. Generally, it is more efficient to change a large number of properties at once as this eliminates the need to type the command over and over again.

The next two options deal with the size of the text on the drawing.

When a text string is added to a drawing, it is defined by a vertex at the lower left corner of the text string. The text, whether in the form of notes, signal names, or properties, is added to a scale of approximately 12 characters per inch. This yields text that is quite legible on the hard copy of the drawing without taking up more space than is absolutely necessary.

To change the size of a string of text, type the command DISPLAY followed by a space and the real_number indicating the factor by which the size of the currently displayed text string should be multiplied. Then, using the cursor, select a string of text to be changed. For example, if one wishes to make the value of the title property appear larger than the remaining text, then the number 2.0 could be typed in. This will double the size of that text. Similarly, one may wish to add "fine print" to a drawing. If the number 0.8 is typed in, then the size of the text will be reduced by that factor.

If the number used with the DISPLAY command is an integer, then it is not necessary to type the ".0" following the integer. If the number is less than 1, then a zero need not be typed before the period.

To return the text to the default scale, use the command DISPLAY D <pt>. The command SHOW SIZE <pt> will print the current size of the indicated text, as compared to the default size.

Once the display form of a property has been changed then that change will remain in force until it is overridden by another use of the DISPLAY command. For example, if a body has been defined and that body has a default property SIZE = 1B to indicate that the default value of SIZE is one bit, then one may wish to suppress the display of the "1B." This is done by using the DISPLAY I command while editing that body. When that body is subsequently added to a logic drawing then the property SIZE = 1B will be hidden from view. If, on that logic drawing, the SIZE property is changed, say, to read SIZE = 32B, then the fact that the original property was invisible will be ignored; the new property will be displayed in the normal manner.

The next three options will change the way an existing wire looks. DISPLAY HEAVY will make the net thicker, to look like a bus. DISPLAY THIN will return a heavy line back to the default wire thickness. DISPLAY PATTERN changes a net to one of six patterned lines. Pattern 1 is a filled line (the default) and patterns 2-6 are a variety of dotted and dashed lines. To get a patterned line type, DISPLAY P pattern_number <pt>.

Again, in a LOGIC drawing, the entire net will be changed whereas in a BODY or DOC drawing, only the wire pointed to will be changed. This is for convenience when drawing pictures or doing documentation.

The next two options allow users to change the display of dots already added to the design. If there are a group of open dots that should be filled, the command DISPLAY FILLED "group_name" will change them all to filled mode. Open dots scale when the WINDOW command is used, filled dots do not.

The last two options allow users to change the justification of text on the screen. By default, all user added text is left justified. To change the justification to the right end of the string, type DISPLAY RIGHT and point to the string. For instance, DISPLAY RIGHT to a left justified string appear as follows:

```
                          THIS IS A PIECE OF TEXT
   THIS IS A PIECE OF TEXT
```

When a right justfied string is moved, the cursor attaches to the end (the final T in the above example). As with other DISPLAY options, justification works on groups as well as single objects. See the SET command to change the default justification.

DOt	point...

The DOT command is used to add dots to drawings. Dots are used in logic drawings to indicate that lines crossing one another are connected. (By default, lines crossing are not connected unless "dotted". Lines joining at a "tee" are connected, even without a dot.) Dots are used in body drawings to indicate pin connection points.

Dots can be filled or open. By default, all added dots are open. See the SET FILLED/OPEN commands to change the default. To fill or open one or a group of dots, see the DISPLAY FILLED/OPEN commands.

To dot all the connections in a logic drawing, see the AUTO DOTS command.

EDit	{[<directory>]drawing_name[.[type][.[version][.[page]]]]	}
	LOGIC 1 1	
	{ point	}

This is the basic command for calling a drawing onto the display. To manipulate the drawing "EXAMPLE", type

EDIT EXAMPLE <cr>

If EXAMPLE exists, it will appear on the screen. Otherwise the message "new diagram" will be displayed. If another drawing was being edited and some changes had been made, EDIT will save the old version in a temporary file before bringing in the new drawing.

If the named drawing was edited during this session and was saved in a temporary file while another drawing was edited, EDIT will use the saved version of the drawing. The SHOW HISTORY command gives a list of all drawings which have been edited during the current session.

The EDIT command also allows you to examine the logic definition of a component. For example, if the body B appears in a drawing then EDIT <pt>, where <pt> specifies the body B, will call in the diagram B.LOGIC, and make it available for editing.

If no SCALD directory is given, each directory in the list is searched until a drawing of the name drawing_name is found. If the drawing_name is not found, the new drawing is assumed to belong to the current SCALD directory. If no drawing name type specification is given, the Graphics Editor assumes the drawing is a logic drawing. To edit another type of drawing, for example the time drawing "size shifter", you must type

EDIT SIZE SHIFTER.TIME

All conventions for sim and time drawings are the same as for logic drawings.

Pages are ignored for body drawings but each body can have multiple versions. Any other drawings (e.g. logic, time or sim) may have multiple versions and/or pages. See Chapter 3.

ERror

The ERROR command steps through the errors found by CHECK. It centers each error on the screen, and draws a star at the location of the error. In addition, the nature of the error is displayed on the screen.

EXIT

The EXIT command is used to leave the Graphics Editor.

FILenote	file_name point

The FILENOTE command adds the named text file to a drawing at the point specified. Each line in the file is converted into a note and can be individually moved, copied, deleted and changed once the file is added.

This page has intentionally been left blank.

FINd	pattern

The FIND command searches through the current drawing for all notes, property names and values, and body names that match the given pattern. The pattern can have wild cards just like the argument for the DIRECTORY command. For instance, typing FIND *P will find all path properties. All such items found are placed on a list which can be stepped though using the NEXT command. Each object found is centered on the display where it can be changed, deleted, etc. FIND and NEXT are much like the CHECK and ERROR commands.

The items found are also put into a group. The one letter name of the group (A-Z) is given along with the number of items found. Below are examples of things that might be done to a found group:

```
HIGHLIGHT ALL PATH PROPERTIES:
     type FIND PATH to get all the properties
     the message displayed is
          Group A
          100 items found
     type SHOW GROUP A

DELETE ALL PATH PROPETIES
     type FIND PATH to get all the properites
     type DELETE A

MAKE ALL PATH PROPERTIES INVISIBLE
     type FIND PATH to get all the properties
     the message displayed is
          Group A
          25 items found
     type DISPLAY I "A" to make them invisible
```

In the last example, the group name (A) must be quoted in the DISPLAY command.

FOrmat	text_file <cr> new_drawing_name

The FORMAT command adds drawings to an existing text file and creates a new Graphics Editor drawing containing both sets of information. It takes a UNIX ASCII text file that has the names of the drawings you want to insert within it, and creates a new_drawing_name.DOC drawing from it. There MUST be a carriage return between the file_name and the new_drawing_name. The UNIX file can be a text file that has been formated by runoff (on the VAX) or nroff (in UNIX). Each page of the text file is turned into a page of a drawing. A page ends with the 60-th line or a user inserted ^L (formfeed).

Each page created by FORMAT is 8-1/2 by 11 inches, with 6 lines per inch. The characters are slightly larger than the default character font (1.29 times the default) for easier readability.

Space must be left in the text file to allow the insertion of drawings into the document. At least two lines are needed for each drawing. The first line must have an '&' in the first column, followed by the name of the drawing you want to insert. The second line must have the number of lines, N, that are allotted for the drawing (6 lines = 1 inch). You must then insert N blank lines. The Graphics Editor reads the named drawing, smashes it and then scales it to fit into the alotted space.

For more information on creating mixed text and graphics documents using GED, see Mixed Text and Graphics Documents later in this chapter.

```
GEt | {[<directory>]drawing_name[.[type][.[version][.[page]]]]  }
                                  LOGIC      1          1
    | { point                                                   }
```

The GET command operates in a manner similar to the EDIT command, but it uses the instance of the drawing stored on disk under the specified drawing name, rather than the most recently modified instance. GET is useful if, while editing a drawing, it becomes necessary to disregard the current work and to revert to an earlier instance.

If no SCALD directory is given, each directory in the list is searched until a drawing of the name drawing_name is found. If the drawing_name is not found, the new drawing is assumed to belong to the current SCALD directory. GET followed by a carriage return will re-edit the current drawing.

```
GRId    [ { ON                                  }          ]
        [ { OFf                                 }          ]
        [ { Dots                                }          ]
        [ { Lines                               }          ]
        [ { grid_size grid_multiple             } ...  ;   ]
```

The GRID command is used to specify the way the grid is displayed. The current values of the grid multiple are displayed on the status line at the top of the CRT. The grid may be turned on and off, that is, made visible or totally invisible, by selecting the ON or OFF options.

The grid multiple (a positive_integer) indicates how the grid should be displayed, if it is to be displayed at all. If the multiple is one, then every grid line may be displayed on the CRT. If it is two, then every other grid line will be displayed, and so on. The default multiple for editing a logic drawing is five.

The grid_size is a value in inches, defining the separation of the grid lines. The size for editing logic, time and sim drawings is 0.1 or 1/10 of an inch. Be careful when changing the grid size because bodies could be placed off grid and if the grid size is again changed, wires may not connect up correctly. The command to change the grid size is:

```
GRID grid_size grid_multiple;
```

The grid_size (a real number) must be a multiple of 0.002 inches -- the smallest possible grid separation.

The GRID command must be terminated by either ';' or any command from the menu. For example:

```
GRID 0.1 2 ON ;
```

However, there is one exception, 'GRID <cr>' will toggle the grid on or off. So, if the grid is on, type 'GRID <cr>' to turn it off. This will not work if GRID is chosen from the menu; in this case, select GRID and then type 'ON ;'.

The grid, by default, is off when a new drawing is edited or retrieved. In order to change the default to always on, use the 'SET GRID_ON' command. The grid_size is, by default, 0.05 on .BODY drawings, 0.166 on .DOC drawings and 0.1 on all other drawing types. To change the default for all types but .BODY and .DOC, use the 'SET DEFAULT_GRID grid_size' command.

The default grid units are grids per inch. The SET DECIMAL/METRIC/FRACTIONAL command allows the grid units to be expressed in different systems.

The Graphics Editor uses 500 internal units per physical inch on the Versatec plotter. The grid multiple displayed on the status line of the display is in grids per inch.

To base plots on the metric system, use the SET METRIC command. The Graphics Editor then uses 512 internal units per physical inch or 20 internal units per physical millimeter. The grid multiple displayed on the status line is expressed in grids per millimeter. Metric users can use standard Valid libraries since pins are on 2.5 mm centers.

With 500 internal units per inch, users cannot use a 1/8 inch grid (the grid can be set to .124 or .126 but not .125). If you use the SET FRACTIONAL command, the Graphics Editor resets the internal units to 400 per inch. 400 was chosen so that the Valid libraries would still be usable. The bodies will appear 25% larger and the pins will be placed on 1/8 inch centers.

GROup	[group_name] point point point...

The GROUP command is used to create a group. The command is issued and a polygon is drawn around a group of objects on the CRT. The polygon may be closed by pressing the blue button when the cursor is near the starting point. All of the vertices that were contained within the polygon will be included in the group just defined. For reference purposes, a one letter group_name is displayed in the upper left corner of the screen. Also, the number of devices, arcs, notes and wires in the group is also listed.

Note that vertices are made members of a group. If a wire is partially in a group and partially out of a group, then only one vertex defining that wire is in the group. The effect of this is described in the use of commands that operate on groups. These commands are MOVE, COPY, CUT, DELETE and DISPLAY.

HArdcopy	[{ A - E } { drawing_name }]	
	[{ real_number } { ; }]	

The HARDCOPY command is used to send drawings to a plotter to produce a hardcopy. Plots can be made on either an electrostatic or pen plotter, including Versatec, Hewlett Packard and Benson models. The SET command specifies what type of device is to be used.

The user has a choice of using default settings or user-specified ones. If no drawing_name is given, the current drawing is assumed. If a page size is given (letters A through E), then the plot is adjusted to that size. If a real_number is entered, the plot is scaled from the normal size.

The drawing_name need not be the drawing currently being edited or in the current working directory. The names should be given in the same format as for the DIRECTORY command (e.g. foo.logic*). If the drawing is from a directory other than the current working directory, the directory name must be given (e.g. <lou.wrk>foo.logic*). If another drawing name is given, a scale factor MUST be given (a real number or A - E). It is necessary to give a ';' after the scale factor to plot the current drawing. For example:

```
HA <cr>              -- plots the current drawing
HA A ;               -- plots the current drawing on an
                        A size page (must end with ';')
HA C *.logic*        -- plots all LOGIC drawings in the
                        current directory on C size pages

HA 1 <100k>100112.body*

                     -- plots the 100112 part from
                        the 100k library with the default
                        drawing size.
HA 1 hyper mux       -- plots all drawing types for the
                        drawing "hyper mux" (e.g. BODY,
                        LOGIC, SIM, etc.)
```

As indicated by the options for the SET command, there are several brands and sizes of plotters that are supported. As of 7.25 software, the plotters supported are the Versatec, Hewlett-Packard, and Benson models.

Versatec electrostatic plots and Hewlett-Packard bed plots can be written to a file and plotted later on a remote or local printer. See the SPOOLED and LOCAL options to the SET command for further information.

There are different procedures for printing files intended for the Versatec and Hewlett Packard plotters while in the SET SPOOLED_PLOT mode. To plot more than one drawing on an electrostatic plotter, issue the GED command HA B *.*. This command writes all drawings into a file called vw.spool. The UNIX command vpl can then be used to plot the drawing or drawings stored there.

Only one drawing can be spooled at a time on the Hewlett Packard plotter. The resultant file is called hpplot.dat. To print a file on a local machine, it is first necessary to return to the UNIX prompt and type

```
/u0/editor/lib/hpfilter <hpplot.dat > /dev/hp
```

These printing instructions differ if the user wishes to plot a drawing on a Hewlett Packard plotter that is attached to another machine on the Ethernet. In this case, return to the UNIX prompt and type

```
/u0/editor/lib/hpfilter <hpplot.dat > /net/machine_name/dev/hp
```

It is possible to vary the resolution on electrostatic output. The Versatec plots two pixels for each one pixel on the screen, a feature that creates darker and more distinct plots. To plot one pixel instead of two, use the SET SINGLE_WIDTH command.

To select the Hewlett-Packard plotter option, use the comment "SET MONO_HPPLOT". "SET HP_PLOT" is no longer in use. The mono specification causes the plotter to use only one pen, as opposed to the color setting, which uses more than one pen. The HP plotter prints filled dots, rotated text, rotated bodies, justified text, and patterned lines.

The HP plotter can plot more than one drawing at a time. To do so, use the metacharacter, as in, for example, HA B FOO*. After each drawing, GED will ask you to type <RETURN> when ready for the next plot, thereby allowing the user to load the paper.

There are two methods for doing hardcopy: HPR (hardcopy print) and VGB (video graphics board). The VGB method, which dates from version 6.0 software, uses the integral graphics board for rasterizing the plot. After version 7.25, it is retained only for compatibility purposes.

The default hardcopy method is HPR. Developments subsequent to 7.25 software, such as new plotters to be supported and color workstations, are enhancements to HPR and not VGB. On the color design station 2310C, for example, only the HPR method is supported. Version 7.25 software supports monochrome plotters.

Compared to the VGB method, HPR will appear both faster and slower. It is faster in that a user will regain the use of a design station much quicker. It is slower in that HPR takes a while to rasterize the plot and get it to the plotter. HPR does the rasterization using the 68010 in the S32. It also queues the plots, allowing more than one design station to plot at the same time.

The VGB method is a compatibility mode. While the Peripheral Interface Board code is much more robust, be warned that this method has certain drawbacks and problems. The VGB method, in contrast to HPR, requires a design station (2310). While plotting with the VGB method, the design station can only be used for rasterization, not for other purposes. Only one design station can plot at a time since the plots are not queued. If the graphics board or peripheral interface board crashes while rasterizing, the design station is not usable. The use of the HPR method avoids all these problems.

The Benson metric 22" (model 9424) plotter is supported beginning with version 7.25 software. It uses the HPR method only. The SET B9424 command is used to initialize the system.

HElp	command_name

This command causes the contents of the specified help file to be displayed. The help file describes the syntax and (briefly) the semantics of the selected command.

HELP HELP or HELP <cr> displays the list of topics on which help is available.

Ignore	[{ SCALD_directory_name }] [{ library_name }]

This command causes the specified SCALD directory or library to be deleted from the active list. The argument specified may have wild cards. If more than one directory matches the pattern, each one is ignored. For example, "IGNORE *" ignores all directories. In addition, IGNORE followed by a carriage-return ignores the current directory. (See the USE command.)

All bodies from the ignored directory or library are made into place-holders. That way a drawing edited in the current session has the body name as a note attached to it and will remind the user to use another library or SCALD directory to replace the place-holder.

If a body is replaced by a place-holder, the other active SCALD directories and libraries are searched for bodies with the same name and version. If one is found, then the place-holder body is replaced by the body from the other directory.

See also the USE and LIBRARY commands.

LED

This command allows a SCALDstar user to compare signal names on the logic design and layout.

To use the command, type LED while in the GED window. Next, use the window manager to go to the LED window and type GED. The two programs will then connect.

Subsequently, whenever the SHOW NET command is used in either the GED or LED window, the net is found and highlighted in both places.

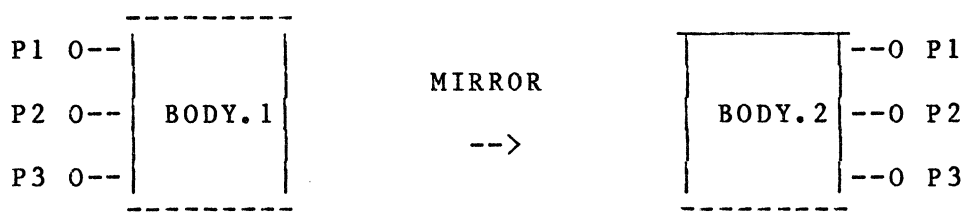
LIbrary	[library_name]

The LIBRARY command adds the specified library to your search list. Library_names are defined by your SCALD library manager. To list the possible names, type "LIBRARY <cr>".

MIrror	point . . .

The MIRROR command creates a mirrored version of a body, as opposed to a rotated version. This command mirrors, about the Y axis, all lines, arcs and text in a body drawing. For the mirrors, only the justification of the text is changed (left --> right, right --> left), and no further rotation is done.

For instance, two versions would resemble the following drawings:



The MIRROR command should be used with caution, especially with bodies with unmarked pins, such as the Valid-supplied merge bodies. Reversing the bits causes subtle, hard-to-find bugs in the design.

The MIRROR command operates differently from the ROTATE command, which allows users to rotate bodies in increments of 90 degrees. Allowing 180 degree rotations of devices will, in some cases, reverse the order of the pins, thereby causing problems in the design. Therefore, a 180 degree rotation of a device will be, in reality, a mirror of a 0 degree rotation (about the Y axis). A 270 degree rotation of a device will be a mirror of a 90 degree rotation (about the X axis).

There are some applications, however, where all four rotations and all four mirrors might be needed. The ROTATE command gives you two rotations (0 and 90) and two mirrors (180 and 270). To get the other two mirrors (0 and 90) and the other two rotations (180 and 270), create another version of the body and use the MIRROR command to mirror it.

Note that MIRROR works differently in body drawings than it does in logic drawings. In a body drawing, the MIRROR command does not require a point. The entire body definition is flipped over, and the procedure is useful for creating other versions of a body.

The SPIN command is used in cases where a true rotation of a body is needed. This command rotates the body 0, 90, 180, 270 degrees without mirroring any of the four representations.

This page has been intentionally left blank.

MOve	({ from_point } to_point)... ({ group_name })

MOVE is the command used to move objects from one position to another on the drawing. To use the MOVE command, you select the command on the menu and then move the cursor to the object that is to be moved. The object is picked up by pressing one of the cursor buttons (usually the yellow button) and moved to a new position by manipulating the cursor. The object may be deposited in its new location by, again, pressing the appropriate cursor button.

If the yellow button is used to select an object, the current cursor position acts as a reference point for placing the object. If the blue button is used, the nearest object or vertex is snapped to the current cursor location.

The MOVE command operates on groups as well as on individual objects. If either the white or the green cursor buttons are used to pick up an object, then the nearest group is selected. If the green cursor button or the group_name is used, the object is snapped to the current cursor location. If the white cursor button is used to select the group, the point serves as reference point for moving the group.

When an object or group of objects is moved and there are electrical connections (wires) leading to that object or group, then the MOVE command will preserve electrical connectivity as well as keep the wires orthogonal.

If users do not want wires to be moved orthogonally, there is a SET option to turn off orthogonal move:

```
SET MOVE_DIRECT
```

To return to orthogonal movement, type

```
SET MOVE_ORTHOG
```

When objects to which properties are attached (including SIG_NAME) are moved, the properties will move with the object. In addition, the properties themselves may be moved independently of the object.

NExt

The NEXT command steps through the items found by FIND. It centers each item on the screen, and draws a star at the location of its vertex. The user can only go through the list once.

NOte	(text_line... point...)...

Notes are text strings that appear on the drawing which in no way affect the evaluation of the drawing by the SCALDsystem. They may be used to document a drawing.

Notes are placed, line by line, using the NOTE command. Once the command has been issued, then any further text will be interpreted as a note. To leave the note command, a semicolon followed by the return key must be pressed. Alternatively, the ";" or any other command on the menu may be selected.

PAste	point ...

The PASTE command, in conjunction with the CUT command, allows objects to be copied from one drawing to another. To copy a group or object that has been CUT, simply type PASTE then select the point where the group or object should be placed.

The command works like the ADD command. When the PASTE command is entered, followed by a carriage return, the cut buffer is attached to the cursor. The group can then be placed down using the yellow or blue cursor button. Additional copies of the cut buffer can be added by pressing the yellow cursor button and then placing the buffer down in the desired place by pressing the button again.

See the CUT command for restrictions on properties that are copied.

PInswap	({ pin_number } point)... ({ point })...

The PINSWAP command can only be used after section assignment has occurred for the part. Also, pin swapping can only occur between pins which have been defined in the library as swappable. Thus it may be legal to swap the two input pins of a NAND gate, but not the input and output pins of the gate.

To swap pins, one can either point at the two pins to be swapped, or one can type in a new pin number for the selected pin. In the latter case, the selected pin will be swapped with the pin with the user specified pin number.

The properties attached by the PINSWAP command cannot be changed, only deleted and moved. They are not written into the connectivity file. Once pins on a part have been swapped, the part cannot be resectioned using the SECTION command.

Only devices in libraries with chips files can be sectioned.

PRoperty | (attach_point (name value text_point)...)...

Properties consist of name-value pairs. A property may be used to specify something about an object on the drawing. For example, if the designer wishes to specify that a particular register is 32 bits wide, then he may attach a property to the register body, the name of which is SIZE, having a value of 32B. For a more thorough discussion of properties see Chapter 3.

A property must be attached to an object; either a body, pin, wire, or signal name. Properties are specified by typing PROPERTY, or selecting the PROPERTY command on the menu, then using the cursor to specify the object (vertex) to which the property should be attached. Next the name and value are typed, separated by a space. To complete the operation, the location on the drawing at which the property text should appear is specified.

Property names may be any string of alphanumeric characters and underscores provided that the first character is an alphabetic character. A property name may not contain any spaces or punctuation (other than the underscore). The property value, on the other hand, may be any string of text. Spaces and punctuation may be included in the property value. As is described in Chapter 3, there are no restrictions on the use, names, or values of properties. Certain kinds of properties, such as SIZE, are known to the SCALDsystem and handled in a consistent manner. Properties that are not known to the SCALDsystem are simply passed through to the output.

The names of properties attached to a given object must be unique; if a newly entered property has the same name as an old property currently attached to that vertex, then the old property value will be replaced by the new property value.

When a property is added to a drawing, only the property value appears. It is possible to temporarily display the names of all the properties on a drawing by using the "SHOW PROPERTIES" command. It is also possible to change the permanent display of the properties such that either the name, value, both or neither are displayed. This is accomplished by using the "DISPLAY" command. Properties can also be manipulated using the commands SWAP, REATTACH, COPY and MOVE.

Quit

This command is used to terminate an editing session. If, after issuing the QUIT command, the system recognizes that drawings have been modified but not written, then the Graphics Editor will so indicate. If you wish to override the warning and terminate the session, then the QUIT command should be issued a second time.

REAttach	(text_point attach_point)...

The REATTACH command is used to move properties from one object to another. If, for example, a property is attached to the input pin of a device and must be moved to the output pin, the REATTACH command is used.

Once the property to be moved is indicated, a line is attached from the current cursor position to the property. You then point to the new attachment point for the property.

Since signal names are simply properties (name = SIG_NAME, value = text string) attached to wires, the REATTACH command may be used to move signal names from one wire to another.

Some properties cannot be reattached, such as default body properties and those produced by the BACKANNOTATE, PINSWAP and SECITON commands. An error message is given when an attempt is made to reattach one of these properties.

REDo

The REDO command redoes the previous operation affecting the screen. A list of operations performed during the current editing session is kept and REDO will repeat events according to this list. The REDO log is reset after each read or write, therefore, REDO can't repeat file operations.

REMove | [<directory>]drawing_name[.[type][.[version][.[page]]]]
.* .* .*

This command deletes a drawing from a SCALD directory. You type REMOVE followed by a drawing name. The editor will echo the names of the files to be deleted, but will not delete the files until another command is selected from the menu. REMOVE can be aborted at any time by typing ABORT, or selecting any command except ";" from the menu. In this case the message 'No changes made' will be displayed. If, after seeing the files to be deleted, you wish to go ahead and delete them, the command is terminated by either a ";" standing alone on a line or selecting the ";" box from the menu. Then the directory entries will be deleted and the files purged.

Wild cards are allowed in the file names specified in the REMOVE command. The wild card '?' will match any character; and '*' will match any number of anything. If just the drawing_name is given (8 BIT MUX) with no wild cards, all types (body, logic, sim, binary, etc) will be deleted. To delete just the binary, for example, type 'REMOVE 8 BIT MUX.LOGIC_BN*'.

If no SCALD directory is given, then the directory list is searched for the first drawing whose name matches the drawing_name. The REMOVE command only takes one argument. In order to delete more than one drawing, type the following:

```
REMOVE drawing_name1
REMOVE drawing_name2
```

When all the drawings have been listed, type ';' alone on a line. Then all the named drawings will be deleted at once.

To delete all drawing types of a specific drawing (SIM, LOGIC, BODY, etc), type

```
REMOVE drawing_name1
```

To delete only the logic type, for instance, type

```
REMOVE drawing_name1.logic.*
```

To delete only the first page of a logic type drawing, for instance, type

```
REMOVE drawing_name1.logic.1.1
```

```
REPlace | [<directory>]body_name[.[type][.[version]]]  point...
                                BODY       1
```

The REPLACE command is used to substitute one device for another. The device to be replaced is selected with a puck point, and the new device is given by name. Any properties attached to the selected device are reattached to the new device. Pin properties are reattached if a pin name on the new device is the same as a pin name on the first device. If no pin name match exists, the pin property becomes a body property. All properties but those generated by the BACKANNOTATE, SECTION and PINSWAP commands are kept. Any wire connections to the original device are only kept if pins are in the same place.

Restore

The Graphics Editor RESTORE command has been deleted. An automatic recovery routine has been implemented and is documented in the Advanced Information section of this chapter of the manual.

However, if you wish to restore a saved temporary ged file from UNIX, type,

```
/u0/editor/ged/gedrestore tempFileName drawing__name directory
```

where

tempFileName is a GED saved file of the form a?aaaaa?.xyz
the drawing__name is in quotes (e.g. "TEMP ONE")
the directory is the directory where the drawing is to be stored.

When the program is done, you can start up ged, edit drawing_name and either delete it or rename it, if you wish.

The Graphics Editor cannot be running when you do a UNIX restore.

RETurn

This command causes the Graphics Editor to return to the drawing previously being edited. If the current drawing was modified but not written, then the system will save a copy of that drawing before returning to the previous drawing. The SHOW HISTORY command lists the drawing that you were previously editing.

ROtate	point....

To rotate a body, type ROTATE and then point to the device or text string to be rotated. Each indication of a device rotates it 90 degrees.

Rotations are in increments of 90 degrees (0, 90, 180 and 270). When a body is rotated, all notes and properties are also rotated. The properties can be rotated or justified independently.

Allowing 180 degree rotatations of devices will, in some cases, reverse the order of the pins (e.g. mergers). This will cause many subtle bugs in users' designs. Therefore, a 180 degree rotation of a device will be, in reality, a mirror of a 0 degree rotation (about the Y axis). A 270 degree rotation of a device will be a mirror of a 90 degree rotation (about the X axis). In the 90 degree rotation, body notes will be rotated 90 degrees and left in their original justification.

For the mirrors, only the justification of the text will be changed (left --> right, right --> left), and no further rotation is done. Text rotations (properties and drawing notes) will be really rotated, not mirrored. To get the other four rotations, create another version of the body using the MIRROR command.

SCAle	(point point) drawing_name

The SCALE command adds the named drawing to the current diagram in the rectangle indicated by the two points. The drawing is smashed (all devices turned into wires, arcs and text). SCALE is most useful for doing documentation drawings (see the FORMAT command).

SCRipt	file_name

The SCRIPT command allows you to specify editor commands in a script file. This is most frequently, but not always, used to initialize the list of working directories. (See also USE and IGNORE.)

A special file, "startup.ged" is expected by the Graphics Editor as an initialization script. If that file does not exist then a warning message will be displayed.

SECtion	[pin_number] point ....

The SECTION command allows the user to step through the different sections of a chip and have the pin numbers of these sections displayed on the drawing. Sectioning a part will automatically auto path the drawing.

The SECTION command works like the VERSION command by pointing to the body or pin of a part. Currently, only parts that can be assigned to a particular section are either SIZE wide parts with a size of 1 or HAS_FIXED_SIZE parts. Assigning sections to a HAS_FIXED_SIZE part is accomplished by pointing to the pin of the section to be assigned. It is an error to point to the body of a HAS_FIXED_SIZE part.

If the part selected can be assigned to a section, the pin numbers for the selected section will be back annotated to the part. If the same part is selected again, the next section will be selected and the new pin numbers will be back annotated to the part. Thus, by pointing to the same part, you can step through all the different possible sections. To assign a specific section directly, first type in a pin number that uniquely defines the section before pointing at the part.

The pin number properties will be schematic annotation properties ($PN), so they cannot be changed, only deleted and moved. They are not written into the connectivity file.

Only devices in libraries with chips files can be sectioned.

See the PINSWAP command for information on swapping the pin assignments once a part has been sectioned. If any pins are swapped, the part can no longer be sectioned.

SET	{ SIZe scale_factor }	
	{ Ascii }	
	{ NOAscii }	
	{ Binary }	
	{ NOBinary }	
	{ Conn }	
	{ NOConn }	
	{ DEpendency }	
	{ NODependency }	
	{ Orthog_wire }	
	{ DIrect_wire }	
	{ STOp_at_pin }	
	{ GO_at_pin }	
	{ DECimal }	
	{ MOVE_Orthog }	
	{ MOVE_Direct }	
	{ DOTS_Open }	
	{ DOTS_Filled }	
	{ STICKY_OFf }	
	{ STICKY_ON }	
	{ CAPSLOCK_OFf }	
	{ CAPSLOCK_ON }	
	{ GRID_OFf }	
	{ GRID_ON }	
	{ MEtric }	
	{ Fractional }	
	{ DEFault_grid number }	
	{ HPr }	
	{ VGb }	
	{ B9424 }	
	{ MONo_HPplotter }	
	{ W11_versatec }	
	{ W22_versatec }	
	{ W36_versatec }	
	{ W42_versatec }	
	{ LOcal_plot }	
	{ SPooled_plot }	

```
|                  { DOUble_width          }                  |
|                  { SINgle_width          }                  |
|                                                             |
|                  { LEFt_justification     }                 |
|                  { RIght_justification    }                 |
|                                                             |
|                  { USER_SIM simulatorfilename  }            |
|                                                             |
|                  { COLOR_Wire color      }                  |
|                  { COLOR_Prop color      }                  |
|                  { COLOR_Dot color       }                  |
|                  { COLOR_Arc color       }                  |
|                  { COLOR_Note color      }                  |
|                  { COLOR_Body color      }                  |
|                                                             |
```

The SET command is used to assign various default parameters. To see how the options are currently set, type SET <cr>.

The size argument sets the default size of entered text. For example, SET SIZE 1.5 makes all text (signal names, notes and properties) 1.5 times the height of the default character set (as if a DISPLAY 1.5 had been done to each piece of text). To return to the default text size, use SET SIZE 1. The default text size will plot 0.082 inch high characters on the electrostatic plotter. If the default size is set to N, then the plotted characters will be N times 0.082 inches high. The maximum height is 2 inches (or 24 times the default size).

The next eight arguments are write options and allow you to specify which types of files the Graphics Editor saves when a write is done. Four different types of files are usually written: ASCII, BINARY (makes for faster reads), CONNECTIVITY (used by the Compiler), and DEPENDENCY (for update procedure). If you are creating a flow chart, for example, you might not want to write the connectivity file. SET NOCONN will cause the write command not to output the connectivity file. To reset the option to write connectivity files, type SET CONN. The default values for the write option are ASCII, CONN, BINARY, DEPENDENCY.

ORTHOG_WIRE/DIRECT_WIRE: These arguments are wire options. ORTHOG_WIRE is the automatic orthogonal wire mode; the white and green cursor buttons are used to change the direction of the wire. DIRECT_WIRE is the non-orthogonal wire mode; the white and green cursor buttons are used to end and create orthogonal wire segments. The default setting is ORTHOG_WIRE. See the description of the WIRE

command for more information.

STOP_AT_PIN/GO_AT_PIN: These arguments affect the WIRE command. To end a wire requires 2 cursor button presses. This is often awkward when you connect a wire to a pin on a body. The STOP_AT_PIN option causes the wire to terminate when a wire reaches a pin. The GO_AT_PIN option does not end the wire. The default value is STOP_AT_PIN.

MOVE_ORTHOG/MOVE_DIRECT: These arguments are move options. When an object is moved, all wires electrically connected to it are also moved. MOVE_ORTHOG is the orthogonal movement mode and preserves the bends in the wires when an object or wire is moved. MOVE_DIRECT is the non-orthogonal movement mode; wires become diagonal when an object or wire is moved and must be reorthogonalized with the SPLIT command. The default setting is MOVE_ORTHOG.

DOTS_OPEN/DOTS_FILLED: The DOTS_OPEN and DOTS_FILLED arguments allow the users to set the default at which DOTS will be displayed on the drawing. To create solder dots type, SET DOTS_FILLED. The default is to leave them OPEN

STICKY_ON/STICKY_OFF: If a property is deleted from a body definition, what happens to the property on all the drawings with an instance of that body. These two arguments determine whether deleted default properties are deleted from instances on logic drawings (STICKY_OFF) or whether they are converted into non-default properties (STICKY_ON). This property deletion or transaction occurs when the logic drawing is read in. The default is to delete them (STICKY_OFF).

CAPSLOCK_ON/CAPSLOCK_OFF: The CAPSLOCK_ON argument creates a software capslock. All keyboard input is upper-cased. If the capslock key is not pressed in, all input will echo as lower case but The Graphics Editor will upper case it internally. The default is CAPSLOCK_OFF.

GRID_ON/GRID_OFF: When a drawing is first edited, the grid is, by default, off. To turn the grid on for that particular drawing, you type GRID <CR>. However, if you want the grid to be on when drawings are first edited, you use the GRID_ON argument. To then turn the grid off for that particular drawing, you type GRID <CR>. The default setting is GRID_OFF.

DECIMAL/METRIC/FRACTIONAL: GED uses 500 internal units per physical inch on the Versatec plotter. The grid multiple displayed on the status line of the display is in grids per inch.

To base plots on the metric system, use the SET METRIC command. GED then uses 512 internal units per physical inch or 20 internal units per physical millimeter. The grid multiple displayed on the status line is expressed in grids per millimeter. Metric users can use standard Valid libraries since pins are on 2.5 mm centers.

With 500 internal units per inch, users cannot use a 1/8 inch grid (the grid can be set to .124 or .126 but not .125). If you use the SET FRACTIONAL command, The Graphics Editor resets the internal units to 400 per inch. 400 was chosen so that the Valid libraries will still be usable. The bodies will appear 25% larger and the pins will be placed on 1/8 inch centers.

DEFAULT_GRID: The grid_size is, by default, 0.05 on .BODY drawings, 0.166 on .DOC drawings and 0.1 on all other drawing types. To change the default for all types but .BODY and .DOC, use the 'SET DEFAULT_GRID grid_size' command.

As indicated in the discussion of the HARDCOPY command, there are two methods for directing files to a printer. The default hardcopy method is HPR, and the compatibility mode is the VGB method.

W11_VERSATEC / W22_VERSATEC / W36_VERSATEC / W42_VERSATEC / HP_PLOTTER: We now support several sizes of Versatec electrostatic plotters. If a user has a 22 inch Versatec and sets the option to 11 inch, the plot will appear normally but won't fill up the paper. If a user has an 11 inch Versatec and sets the option to 22 inch, the plot will be garbage. The default setting is W11_VERSETEC. The user may also switch to an HP plotter by setting the HP_PLOTTER.

LOCAL_PLOT/SPOOLED_PLOT: Output for the Versatec and the Hewlett-Packard electrostatic plotters can now be written to a file to be plotted later or to be plotted on a remote SCALDsystem. The default setting is LOCAL_PLOT. For information on the SPOOLED_PLOT option, see the detailed instruction given in the entry for the HARDCOPY command.

DOUBLE_WIDTH/SINGLE_WIDTH: The electrostatic plotter now prints two pixels where it previously printed one (DOUBLE_WIDTH option). This makes the resulting plots darker and clearer. The DOUBLE_WIDTH option is the default.

LEFT_JUST/RIGHT_JUST: These 2 options change the default justification of text strings (both properties and notes).

USER_SIM: The USER_SIM option allows you to give the UNIX path name of the Simulator to run with the SIMULATE command in GED. The default is /u0/scald/simulator/sim.

```
SHow        { Attach                  }
            { Body_name point         }
            { COOrdinate point...     }
            { CONnections             }
            { Group { group_name }    }
                    { point      }    }
            { History                 }
            { Keys                    }
            { Net   { point      }
                    { net_name   }
            { Origins                 }
            { PIns                    }
            { PRoperties              }
            { Release                 }
            { Size text_point         }
            { Vectors object_point    }
```

The SHOW command is used to make classes of objects visible on the display. The effect of the SHOW command is temporary; objects that are made visible with this command revert to their original state when the drawing is written to the disk file or when the screen is redrawn. The SHOW command may be executed with any of a number of arguments that define which objects are to be shown.

To see a list of all the SHOW options, type "SHOW <cr>".

The ATTACH option is used to display connection between properties and the objects to which they are attached.

The BODY_NAME option will print the name of the indicated device (with its SCALD directory name) on the CRT.

The COORDINATE option is used to display the pixel coordinate of the indicated point.

The CONNECTIONS option is used to display wire connections in the drawing.

The GROUP option causes the specified group to be highlighted. If a point is indicated, the closest group is highlighted and its group_name is given. If a group name is given, that group is highlighted. In addition, the number of devices, notes, arcs and wires in the group is listed.

The HISTORY option lists the drawings you have edited. It shows which edited files have been modified but unwritten. In addition, it lists the drawing the RETURN command will return to.

The KEYS option causes the function key assignments (which function key represents which string) to be displayed.

The NET option lists the name of the indicated net as well as highlighting its segments. The net can be indicated by either typing the name or pointing to a net with the cursor.

The ORIGINS option is used to display the origins of the bodies on the screen.

The PINS option is used to display the pin connection points on bodies.

The PROPERTIES option is used to show all of the properties, both name and value, on the drawing. Since signal names are handled internally as properties attached to the wire, the use of the SHOW PROPERTIES will also cause the text "SIG_NAME =" to appear with each signal name.

The RELEASE option gives the release number of the Graphics Editor.

The SIZE option gives the display size of the characters in the indicated text_string. This size is the multiple of the default text size.

The VECTORS option gives the pin names from the body definition for the indicated part.

SIgname	(point... signal_name...)...

The SIGNAME command allows you to attach signal names to wires or pins. The point specified identifies the location at which the text (signal name) will be placed. The signal name will be attached to the wire or pin that is closest to the specified point.

Signal names are handled internally as properties. For example, attaching a signal called "BUS ENABLE" to a wire is equivalent to attaching a property "SIG_NAME=BUS ENABLE" to that wire.

When editing a body drawing, signal names are called Pin_names and can only be attached to the connection points as identified with dots.

Signal names can be up to 80 characters long.

SMash	point...

The SMASH command breaks a body into pieces. The device is no longer considered one object but individual wires, arcs and notes. Any properties attached to the body are deleted. The SMASH command is useful for creating library body drawings. For example, once a 2 input AND gate is created, N input AND gates can be made by specifying the following:

```
edit N AND.body
add 2 AND <pt>
smash <pt>
```

Now the N inputs can be attached and the drawing written. Because the 2 AND is no longer a body, the graphics editor will not complain when the drawing is written (as it would when a body is added to a body drawing and an attempt is made to write the result).

The SMASH command can only be used on bodies.

SPIn	(point point)...

The SPIN command is used in cases where a true rotation of a body is needed. This command rotates the body 0, 90, 180, 270 degrees without mirroring any of the four representations. See also MIRROR and ROTATE.

SPlit	(point point)...

The SPLIT command can be used to perform two functions. First, to split a single wire into two wires by adding a vertex along that wire. The second use of the SPLIT command allows objects that have been co-located (placed at the same vertex) to be disconnected from one another. For example, if a wire is connected to one pin and the designer wishes to disconnect it and move it to a different pin, then the SPLIT command would be used.

To split a single wire into two wires (adding a vertex along the wire between the original two vertices), select the SPLIT command and identify a point along the wire. Once the new vertex has been added to the wire, the SPLIT command operates much like the MOVE command; the vertex selected may be moved about the drawing until it is in the desired location. The new vertex may be placed by specifying a second point.

In order to disconnect two items that are co-located, the vertex in question is specified. This will cause one of the objects to be disconnected and made mobile on the cursor. If the object that was split off was not the desired one, then simply select the original vertex again and the second object will be pulled off. Continue selecting the vertex until the correct item has been selected. Once an object has been split off, it can be placed at a new location by moving the cursor and pressing the appropriate button. If all the objects have been split off the vertex and one item is not relocated, selecting the vertex one more time will place down the last item. One more selection will begin the cycle again, splitting off each item in turn.

Whenever possible, the SPLIT command attempts to operate on wires.

SWap	{ text_point text_point }...

The SWAP command is used to swap two properties or two notes. For instance, SWAP might be used to change the pin ordering on a part from <0..7> to <7..0>. Only two notes or two properties can be swapped, not a note and a property. Default properties and those generated by the PINSWAP, SECTION, and BACKANNOTATE commands cannot be swapped.

UNdo

The UNDO command undoes the previous operation affecting the screen. A list of operations performed during the current editing session is kept and repeated applications of UNDO will undo events according to this list. Each read or write of a diagram causes the UNDO log to be reset, therefore UNDO cannot undo past file operations.

UPDATE	(a UNIX command)

UPDATE is a UNIX command. It must be run from a design station with no Graphics Editor running. The command cannot be run in the background.

To update the drawings in your SCALD directory, use the command

```
/u0/editor/update |  { <cr>                    }
                  |  { -n                      }
                  |  { -b                     }
                  |  { -a  "drawing name"     }
                  |  { -f  "part_name"        }
```

from UNIX. The parameters are:

<cr> (no parameter): find all drawings in the current directory that need updating and remake them. First deletes the binary and reads in the ASCII version of the drawing so that changes to properties are handled correctly.

-n: find all the drawings in the current directory that need updating and list them.

-b: find all drawings in the current directory that need updating and remake them. Does not delete the binary versions first, so if a binary version exists, property changes are not handled correctly. This option is faster than the first (no parameter) option and is preferred if the user knows that only the body shapes have changed, not the properties.

-a "drawing name": whether or not the named drawing is out of date, remake it. The drawing name should be in quotes and fully specified with no wildcards. For instance,

```
/u0/editor/update -a "SIZE SHIFTER.LOGIC.1.1"
```

-f "part_name": find and list all drawings that use the named part. The part name is quoted and of the same form as the Graphics Editor ADD command. For instance,

```
/u0/editor/update -f "3 MERGE"
```

anything else: lists the parameters for /u0/editor/update and the meaning of each.

The search path used to find the components in the drawings is the path defined in your STARTUP.GED. The drawings updated are those in the current UNIX directory.

USe	directory_name

The USE command allows you to specify the current working directory. There is no limit to the number of directories that can be in use at one time.

The form of "directory" is:

name.extension

where name is an alpha-numeric string beginning with an alphabetic character, not exceeding 8 characters in length. The "extension" is an alpha-numeric string of 1-3 characters. The extension should be either WRK (for a user directory) or LIB (for a part library). In addition, there should only be one SCALD directory per UNIX directory.

To USE a SCALD directory other than one in the current UNIX directory, the UNIX pathname must be given. For instance,

USE /u0/job/common.wrk

See also the IGNORE command.

VECtorize

The VECTORIZE command creates a file called vector.dat that contains the current drawing in vector format. See "Vector Plot Format from GED" for information on the format.

VERsion	point...

Bodies may be created with several different symbolic representations. For example, the NAND gate is equivalent to an INVERT-OR gate by DeMorgan's Theorem. Similarly, a NOR gate is equivalent to an INVERT-AND gate.

In order to step from one representation of a body to another, you select the VERSION command and, using the cursor and cursor buttons, point to the body in question. On pressing the cursor button, the Graphics Editor will determine which version of that body is currently being displayed, and will replace it with the next version in sequence. If the n-th version of a body that only has n versions was being displayed, then the use of the VERSION command will cause version number one to be redisplayed.

The separate versions of a body must all make reference to the same logic drawing. The use of a different version of a body has no influence on the logic drawing defining it.

WINdow	{ point point point } { point point; } { point; } { Fit } { ; } { positive or negative integer } { positive fixed point number }

The WINDOW command was described in detail in the section on "Windowing and Scaling." Basically, the WINDOW command is used to change the view of the drawing on the CRT. This command may be used with zero, one, two, or three arguments. If fewer than three arguments are used, then you must enter a semicolon, either by selecting that character on the menu or by typing a semicolon followed by the return key.

If the WINDOW command is presented with no arguments, just a semicolon, the Graphics Editor will simply redraw the image without changing the center or the scale. This option is useful if the left side of the drawing has several error messages which cover up part of the drawing.

Entering only one point as an argument, followed by a semicolon, causes that point to become the center of the drawing. The scaling is the same as before.

With two arguments (two points followed by a semicolon), the WINDOW command will cause the area, defined by the rectangle having those two points at opposite corners, to be expanded to fill the screen.

If all three points are used with the WINDOW command, then the first point defines the new center of the drawing, the distance between the second point and the first point defines a first distance, and the distance between the third point and the first point defines a second distance. The drawing is redisplayed with the first point at the center of the screen and is scaled by the ratio of the second distance to the first distance. If the second distance is greater than the first distance, then items will appear larger; if the second distance is smaller than the first distance, then items will appear smaller.

The WINDOW command may be used to fit the entire drawing to the screen. This is done by typing WINDOW F<cr> or by selecting WINDOW on the menu and typing F<cr> on the keyboard. It is not necessary to follow the F with a

semicolon.

If an integer or a real number is used as the argument to the window command, then the view of the drawing will be scaled. The center of the window will remain as is. For example:

```
WIN 2
```
will make the drawing appear twice as large

```
WIN -2
```
will reduce the size by a factor of 2

```
WIN 1.5
```
will make the drawing one and a half times larger

```
WIN 0.5
```
does the same as WIN -2

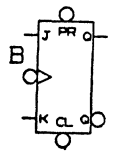
WIRe	(point point...)...

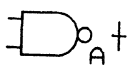
The WIRE command is used to add wires to a drawing. The wire will begin at the first point specified and run to the second. Additional points may be specified in order to draw a wire that consists of one or more segments. To terminate the run, a zero length segment is specified (usually by pressing one of the cursor buttons twice at the final point). If a wire is attached at the pin, it is automatically terminnated without the second press of the cursor button. To change this, use the SET STOP_AT_PIN/GO_AT_PIN command.

The automatic orthogonal wire mode described below is the default wire mode. However, the non-orthogonal wire mode that was available before the 4.2 release is also available. This compatibility mode is described below, after the explanation of the automatic orthogonal wiring.

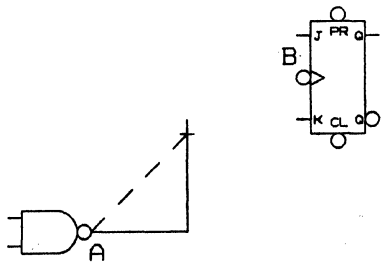
Because schematics almost exclusively use orthogonal wires, wires added to drawings are orthogonalized. After the wire is started, as the puck is moved in any direction -- horizontally, vertically or diagonally -- the attached wire stays bent in the middle. To change the orientation of the bend, press the white (or green) button. If the white (or green) button is pressed a second time, the wire will become diagonal. A third press will put the wire in the first orthogonal position. An example is given below:

To create an orthogonal wire from point A to point B, place the puck at A and press the blue button (to snap to the connection).

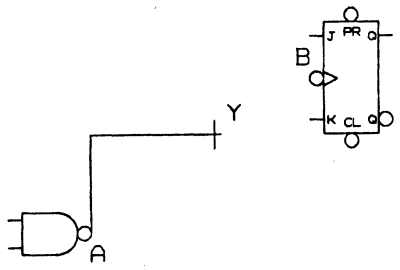




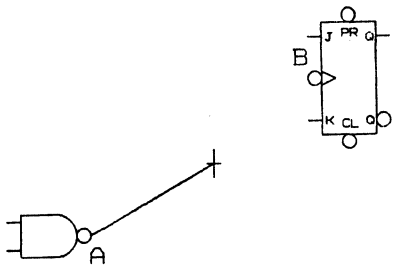
As the puck is moved from A towards B (along the dotted path) an orthogonal wire is created along the solid path.



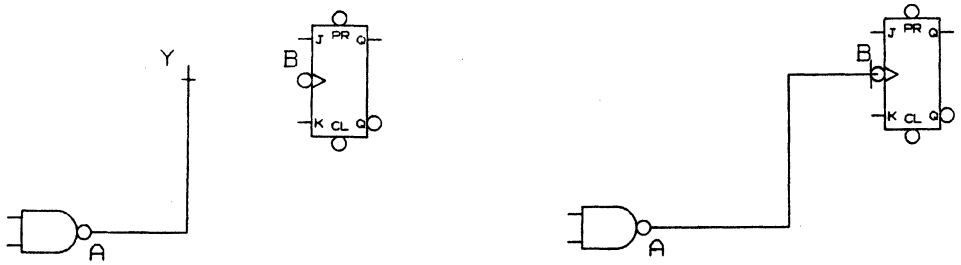
If, you want the wire bent in the other direction, press the white button to get:



Another press of the white button before the wire is put down will un-orthogonalize the wire to look like this:



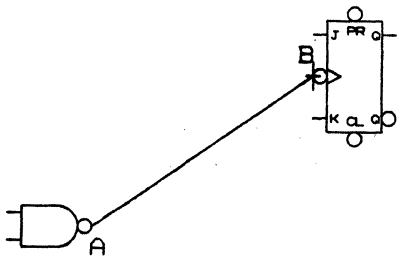
To complete the wire, press the white or green button again, then press the yellow button to bend the wire at Y. Drag the wire towards the clock input and press the blue button twice to snap to the connection and lay the wire down.



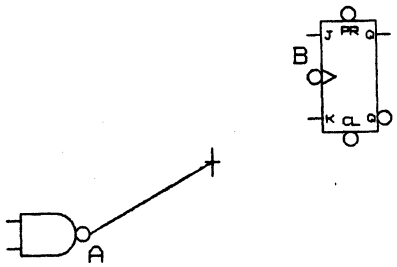
The compatibilty mode for the wire command as it worked before the 4.2 release of the Graphics Editor is available by typing "SET DIRECT_WIRE". The command can be typed at the keyboard or added to the user's startup.ged file. In this mode, all wires are diagonal until they are placed down. Finishing a wire with the yellow or blue button creates a diagonal wire. Ending a wire with the white and green cursor button creates orthogonal wire segments. Pressing the green button to end a wire will create orthogonal wires from the start point to the nearest vertex. Ending the wire with the white button creates orthogonal segments to the nearest grid point.

If in compatibility mode, the automatic orthogonal wiring mode can be entered be typing "SET ORTHOG_WIRE".

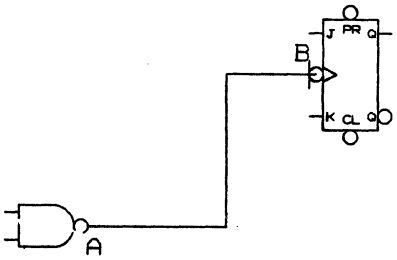
Below, the user wants to create a wire from point A to point B. Placing the puck at A, press the yellow button, pushing the puck to B and pressing either the yellow or blue button results in the diagram below.



To create an orthogonal wire, start the wire at point A by pressing the yellow button.



Then, push the puck to point B and press either the white or green button.



WRite	{[<directory>]drawing_name[.[type][.[version][.[page]]]]}
	{ LOGIC 1 1
	{ <cr> }

This command writes out the current drawing onto disk. If no drawing name is specified, the drawing is written into the drawing name specified on the status line at the top of the display.

If no directory is given, the drawing is written to the SCALD directory it was retrieved from. If the drawing is a newly created drawing and no directory is given, then the drawing is written to the current directory.

If a drawing name other than the one listed in the status line is given and that drawing name is already in a SCALD directory, a warning message is given. You must select WRITE again in order to actually write the drawing. Selecting any other command aborts the write.

Advanced Information on the Graphics Editor

3.8 INTRODUCTION

This section is intended as a technical reference for experienced, not novice, users of the the Graphics Editor. This section describes the files used by the editor, the editor's interaction with other parts of the SCALDsystem, and specifics about Graphics Editor commands.

To learn about building libraries, see "Defining Bubble Groups in the Graphics Editor," in the next section, and "Library User's Guide" and "Valid Library Styles and Standards" in Chapter 11.

3.9 SYSTEM-WIDE FILES

Several files are used by the Graphics Editor for initialization. These are:

```
/u0/editor/startup.ged
/u0/editor/softkeyassign
/u0/lib/ged/config.dat
/u0/lib/master.lib
```

The /u0/lib/master.lib file, which is also used by the SCALD compiler, replaces the former file /u0/lib/ged/master.dir. The file /u0/editor/doc contains the help files. The file /u0/editor/startup.ged is a system-wide initialization file. It defines some of the directories referenced by the Graphics Editor. It is referenced by all users of the system. Currently startup.ged defines the keys on the workstation keyboard. See the file /u0/editor/softkeyassign for the default key assignments. To see how one can assign different values to the keys, type HELP ASSIGN in the Graphics Editor.

The other two initialization files have to do with component library initialization: /u0/lib/ged/config.dat defines the characters that are used for signal name definitions (e.g. -signal_name or signal_name* for a low asserted signal), and /u0/lib/master.lib is a file of name translations for the SCALD component libraries. The file entries contain the short-hand name for the Graphics Editor LIBRARY command and the UNIX pathname to the location of the Library. An example entry in master.lib would appear as follows:

```
'sttl' '/u0/lib/sttl/sttl.lib';
```

Instead of typing USE /u0/lib/sttl/sttl.lib, you can type

LIBRARY sttl.

There are several reasons for including this translation file. First, it is a nuisance to have to remember the long UNIX path names. Second, most of you will not have permission to modify library components; the translation file keeps the libraries somewhat hidden.

The master.lib file is also used by the Compiler.

3.10 USER OWNED FILES

Individual users have several files in their own login directories that are used by the Graphics Editor. The two most important are startup.ged and your SCALD directory.

STARTUP.GED

Startup.ged is used for initialization, as is /u0/editor/startup.ged, discussed above. The file contains standard Graphics Editor commands. The last line should tell the Graphics Editor the name of your SCALD directory. The startup file may also contain library commands. If you don't like any of the Graphics Editor defaults (such as a display grid that is off), any other lines would change these defaults. For example, your startup.ged might be:

```
library 100k
grid dots on ;
use steve.wrk
```

3.11 SCALD DIRECTORIES

The SCALD directory is used as a name translation file between SCALD drawing names and UNIX directory names. The translation file is used because SCALD drawing names can be any length and can have spaces in the name while UNIX directory names are fixed length and one word.

FORMAT

The format of the SCALD directory is important for the Graphics Editor to run correctly. If you ever need to change a SCALD directory manually (without the aid of the Graphics Editor), this format must be followed.

A SCALD drawing name has four parts:
SCALD drawing name.drawing_type.version.page (e.g. SIZE SHIFTER.LOGIC.1.1). There is a UNIX directory for each SCALD drawing name. In the directory is stored all the drawing types (SIM, LOGIC, PRIM, BODY, etc) and all versions and pages. An example of a SCALD directory is:

```
file_type = logic_dir ;
"SIZE SHIFTER" 'sizeshifter';
"LS112" 'ls112';
"LS373" 'ls373';
"SUPER HYPER MUX BOX" 'superhypermuxb';
END.
```

All drawings named SIZE SHIFTER are stored in the UNIX directory /u0/username/sizeshifter. The SCALD drawing name (the name in double quotes) MUST be in uppercase.

CREATING AND USING A SCALD DIRECTORY

When your home UNIX directory is created with the UNIX utility mkusr, a startup.ged file is created with the entry:

```
use user_name.wrk
```

When the Graphics Editor is used for the very first time, the message "user_name.wrk doesn't exist and will be created when you write into it" will appear. Additional SCALD directories can be created at any time with the USE command. The Graphics Editor will create a directory if the name given does not exist.

When specifying SCALD directories in the USE command, the full UNIX path name must be given. For instance, to use another user's directory, you might type:

```
USE /U0/MIKE/MIKE.WRK
```

3.12 OTHER FILES

While using the Graphics Editor, two other files will be created: editor.log? and undo?.log. (? stands for the number of the design station the Graphics Editor was run on. It will be set equal to 0, 1, 2 or 3.) Editor.log? is a history of all messages written on the left hand size of the display.

The file undo?.log is used by the Graphics Editor commands REDO and UNDO. The undo log can be used to restore drawings that were not saved if the SCALDsystem crashed while you were in the Graphics Editor.

These files are deleted when the Graphics Editor terminates normally.

3.13 BODIES AND PROPERTIES KNOWN TO THE GRAPHICS EDITOR

The Graphics Editor, in general, has no knowledge of rules about how the logic design works or how components can be connected together. However, the Graphics Editor does use and know about several bodies and properties. These are discussed below.

SPECIAL BODIES

The Graphics Editor uses three special bodies from the STANDARD library.
1) Origin Body -- used to mark the origin reference point in bodies
2) Drawing Body -- GED sets the last_modified property
3) Pin Names -- shows the pin names of the body for current logic (sim, time) drawings

All other bodies have no special meaning to the Graphics Editor. The Origin Body is only used by the Graphics Editor while the Drawing and Pin Names bodies are used by the Compiler and other programs in the SCALDsystem.

Special Properties

The Graphics Editor currently knows about the properties:

Last_modified (on the drawing body)
Pin_Name
Sig_Name
Properties added by BACKANNOTATE, SECTION and PINSWAP commands

The Graphics Editor knows about the Pin_Name and Sig_Name properties so that they can be attached to components correctly. All other properties are passed to the other programs in the SCALDsystem.

The Graphics Editor also has rules about which commands can be performed on which properties. For instance, default body properties cannot be deleted and their names cannot be changed. Or, properties generated by the PINSWAP, SECTION and BACKANNOTATE commands aren't written into the connectivity file. The Graphics Editor gives an error message if an illegal operation is performed on a property.

The placeholder property '?' can be used in body drawings. For instance, to indicate to the Graphics Editor where the PATH property is to be placed on a body, add the property PATH=? when defining the body. Once the body is added to a drawing, the Graphics Editor checks for a property value of '?' and does not write those properties

into the connectivity file. However, once the property has been given a real value (for instance, the AUTO PATH command was given), the propety is written into the connectivity file.

3.14 RULES ABOUT DRAWINGS

Currently there are few enforced rules about what can be put in drawings. The rules that are enforced are as follows:

1) Users can't write a drawing into a drawing of a different type (e.g. if editing shifter.logic, can't write shifter.sim) Use the DIAGRAM command to change the drawing type of the drawing.

2) Bodies cannot be added into other body drawings and saved. Although other bodies can be added to other bodies for comparison purposes, the Graphics Editor will complain if the body drawing is written out. If you want to use another device as a model, add the device to the body drawing and use the SMASH command.

3) Users cannot add incompatible bodies to diagrams. For instance, sim devices are illegal in time diagrams. Both the Graphics Editor and the Compiler will complain about illegal bodies in drawings. If a directory is illegal for the drawing currently being edited, the DIRECTORY command listing will state it is illegal.

3.15 DRAWING FORMATS

The Graphics Editor stores drawings in four formats: binary, ASCII, connectivity and dependency. See the Editor File Formats section later in this chapter for an exact description of the syntax of these files. The binary, ASCII and dependency representations are used exclusively by the Graphics Editor. The ASCII file can be printed, is easy to understand, but is slow for the Graphics Editor to read. The binary file can't be printed, isn't easily understood, but is quicker to read. If a binary file exists, the Graphics Editor reads it instead of the ASCII file. The connectivity file is used by the Compiler.

When a drawing is written, all four formats are created. The files are stored in the same UNIX directory using the SCALD drawing name as the UNIX directory name. If a drawing is named SIZE SHIFTER.LOGIC, then the three formats will be:

```
sizeshifter/logic.1.1 -- ASCII
sizeshifter/logic_bn.1.1 -- binary
```

```
sizeshifter/logic_cn.1.1 -- connectivity
sizeshifter/logic_dp.1.1 -- dependency
```

For sim and time drawings, substitute sim or time for logic, above. Body drawings are only stored in a binary representation. SIZE SHIFTER.BODY.1.1 would be stored as:

```
sizeshifter/body.1.1
```

When writing a DOC drawing (generated by the FORMAT command), only the binary and ASCII representation is written. DOC drawings are assumed to be drawings, rather than schematics, so the connectivity representation isn't necessary.

The SET command allows users to not write either the binary, ASCII, connectivity, or dependency files.

3.16 TEMPORARY FILES AND RECOVERY FROM CRASH

When you edit a second drawing without writing out the first one, the Graphics Editor saves a copy of the first drawing. The saved file is in binary only and is named aTaaaaa?.xyz where ? is the number of the Nth temporary file stored in your UNIX directory and T is the number of your design station. Temporary files are not written into your SCALD directory. Saving only a binary version makes it faster to read in a re-edited drawing.

These temporary files are deleted from the UNIX directory if the Graphics Editor terminates normally. However, if the Graphics Editor or the cluster crashes, all drawings but the last one can be restored via the temporary files.

If GED or UNIX crashes, it is possible to recover the drawings that were being edited while GED was running. This recovery feature was altered in version 7.25 software to make the procedure simpler. In the event of a crash, a user can recover files by simply answering "yes" to the query about recovering files. Every time GED is called to the screen, this query appears as one of the first messages from the editor. For normal operations it can be ignored; in the event of a crash, it is easy to recover files.

If a user elects to recover drawings, they are all placed in a SCALD directory called restore.wrk. The recovered drawings are called RESTORED1, RESTORED2, ... If restore.wrk exists, it will be overwritten. A warning message is printed about this, and it is possible to elect not to recover. The user must type USE RESTORE.WRK to get the recovered drawings.

3.17 UPDATING OUT-OF-DATE DRAWINGS

If library parts change, it is often difficult and time consuming to look through a SCALD directory to determine if any drawings are effected. An update facility is provided with the Graphics Editor to make this process easier. This update facility should perform several functions. First, it should allow you to ask which drawings are out of date and then remake them, using the new parts. This can be done from UNIX and in a 'batch' mode. Second, when editing a drawing, the Graphics Editor should inform you when parts are out of date and, if a new library or directory is used with parts already in the current drawing, you should have the option of replacing the parts. Currently, the first function is implemented.

In order to update a drawing, several things are needed. First, a list of all the parts used by a drawing must be compiled. This list can then be used to determine whether any of the parts are newer than the drawing. Second, changed properties on parts must be handled correctly. For instance, if a property on the part is added or deleted, that property must also be added or deleted on the drawing. In addition, if you have modified a part property value, that value should over-ride any default value.

DEPENDENCY FILES

In addition to writing ASCII, binary and connectivity files for drawings, the Graphics Editor writes a dependency file for each drawing. This file lists the bodies used by a drawing as well as which UNIX directory the parts came from. When running the update facility, the date on the file containing each part is compared to the date of the last write for the drawing. If any of the parts are newer than the drawing, the drawing needs to be updated.

UPDATING A DRAWING

To update the drawings in your directory, use the command

```
/u0/editor/update |  { <cr>                    }
                  |  { -n                      }
                  |  { -b                     }
                  |  { -a  "drawing name"     }
                  |  { -f  "part__name"        }
```

from UNIX. The parameters are:

<cr> (no parameter): find all drawings in the current

directory that need updating and remake them. First deletes the binary and reads in the ASCII version of the drawing so that changes to properties are handled correctly.

-n: find all the drawings in the current directory that need updating and list them.

-b: find all drawings in the current directory that need updating and remake them. Does not delete the binary versions first, so if a binary version exists, property changes are not handled correctly. This option is faster than the first (no parameter) option and is preferred if you know that only the body shapes have changed, not the properties.

-a "drawing name": whether or not the named drawing is out of date, remake it. The drawing name should be in quotes and fully specified with no wildcards. For instance,

```
/u0/editor/update -a "SIZE SHIFTER.LOGIC.1.1"
```

-f "part_name": find and list all drawings that use the named part. The part name is quoted and of the same form as the Graphics Editor ADD command. For instance,

```
/u0/editor/update -f "3 MERGE"
```

anything else: lists the parameters for /u0/editor/update and the meaning of each.

The search path in your STARTUP.GED file is used to determine which libraries to use and in what order to use them. UPDATE will work only for drawings in the current UNIX directory. Be sure your present working directory is the same directory used when running GED when you use the UPDATE command.

The UPDATE command is run from UNIX. It must be run on a design station without a Graphics Editor running. The command cannot be run as a background process.

3.18 ADVANCED GRAPHICS EDITOR COMMANDS

This section is divided into two parts. The first is a detailed description of several Graphics Editor commands. The second section is a list and short description of all Graphics Editor commands. The second section is available on-line with the Graphics Editor HELP command.

FEATURES AND COMMANDS UNIQUE TO THE GRAPHICS EDITOR

Several features of the Graphics Editor are unique to it and ignored by other programs in the SCALDsystem. Most of these features have to do with how the drawings are

displayed. For example, left justifying a property has no effect on the drawing, only the way it is displayed. Notes are also ignored by other parts of the system.

HARDCOPY

To produce a permanent copy of a drawing, the Graphics Editor provides the HARDCOPY command. Drawings can be plotted on A through E size paper (sizes greater than B are printed in strips) and between 0.4 and 2.5 times the nominal size. Various sizes of Versatec plotters are supported -- 11, 22, 36, and 42 inches.

In addition, model 7580A and 7580B HP Pen Plotters are supported.

SCRIPTS

Although you can't capture your keystrokes while in the editor, prewritten files of the Graphics Editor commands can be used in batch mode. For example, you may want to plot all the logic drawings in several directories. You can create a file called printAll containing the following:

```
use user1.wrk
use user2.wrk
use user3.wrk
use user4.wrk
ha A <*>*.logic.*
forcequit
```

(The forcequit command is described below.) You would change your startup.ged to contain only:

```
script printAll
```

and start the Graphics Editor.

Several Graphics Editor commands have been added for use in scripts. FORCENOTE and FORCESIG must be used in place of NOTE and SIGNAME, respectively. The CHANGE command (text editor) is not allowed in scripts. The commands are:

1) FORCEADD component_name
point ;

The ADD command complains if a component is not found in the current working directories and libraries and does not add the component. Forceadd, however, creates a place holder for the component in the drawing. If a library used later contains that component, it is added to the correct places in the drawing.

2) FORCENOTE note
point ;

The interactive note command takes combinations of notes and points until you enter another Graphics Editor command. This does not work in a batch mode and the note command is not allowed in a script. To add notes, use the forcenote command which takes one note and the point where it should be placed.

3) FORCEQUIT

The quit command, like the note command, is only for interactive use. To exit from a script, use the forcequit command.

4) FORCESIG signal_name
point ;

The forcesig command adds a signal name to the specified point. The signame command is not allowed in scripts, only the forcesig command.

Section and Pin Assignments

3.19 INTRODUCTION

Users want to be able to do section and pin assignments in their drawings and have the Packager perform these assignments for the actual physical design. To provide these capabilities, the Graphics Editor has the commands SECTION and PINSWAP which place special properties in the drawings that are understood by the Packager.

3.20 USER INTERFACE

```
SECTION    |   ( [ pin_number ] point )...
```

The SECTION command works like the VERSION command. Currently, only parts that can be assigned to a particular section are either SIZE wide parts with a size of 1 or HAS_FIXED_SIZE parts. Assigning sections to a HAS_FIXED_SIZE part is accomplished by pointing to the pin of the section to be assigned. It is an error to point to the body of a HAS_FIXED_SIZE part.

If the part selected can be assigned to a section, the pin numbers for the selected section will be back annotated to the part. If the same part is selected again, the next section will be selected, and the new pin numbers will be back annotated to the part. Thus, by pointing to the same part, you can step through all the different possible sections. To assign a specific section directly, first type in a pin number that uniquely defines the section before pointing at the part.

If you delete the pin numbers back annotated to a SECTIONed part, the section assignment is still there. The only way to de-assign a section is to REPLACE the part with a new copy of itself.

```
PINSWAP    |   ( { pin_number } point )...
           |   ( { point      }       )...
```

The PINSWAP command can only be used after section assignment has occurred for the part. Also, pin swapping can only occur between pins which have been defined in the library as swappable. Thus, it may be legal to swap the two input pins of a NAND gate, but not the input and output pins of the gate.

To swap pins, you can either point at the two pins to be swapped, or you can type in a new pin number for the selected pin. In the latter case, the selected pin will be

swapped with the pin with the user specified pin number.

Once pin swaps have been performed on a part, further section assignments are no longer allowed for the part. This means that if you wish to assign a part to a different section after performing pin swaps, the part must first be de-assigned by using the REPLACE command. You can then assign the new part to the desired section.

3.21 MAKING INDIVIDUAL CHIP FILES

Only devices in libraries with chips files can be sectioned and pin swapped. Only chips files created with the 6.0 or later release of the compiler can be used. If VALID-provided libraries are used, devices from the 4.1 or later release must also be used.

Release 4.1 or later version of the libraries provided by Valid are already set up so that the SECTION and PINSWAP commands will work. User-defined libraries must have the library's chips file divided and distributed to each part in the library.

To create a chips file for a user-defined library, you must create a library drawing containing one each of all devices. For the Valid supplied 100K Library, this drawing is called 100K LIBRARY. Then physical information such as pin numbers, input load, output load, etc., is attached to each device on the drawing. The drawing is then compiled for LOGIC and output with the directive OUTPUT CHIPS. The Compiler then produces a file called chips.dat. This file is renamed to library_name.prt for Valid libraries, e.g. 100k.prt.

The chips file contains the physical information for all the library devices. In order for the SECTION and PINSWAP commands to work, the chips file must be divided into a separate file for each part in the library. To divide a chips file, use the command:

```
% /usr/bin/makechipsfiles chips_file library_name
```

For example, to break up the 100K library,

```
% cd /u0/lib/100k
% /usr/bin/makechipsfiles 100k.prt 100k.lib
```

The new files are stored in the subdirectory for each part. For example,

100171/chips_prt

is the individual chips file for the device 100171.

If the chips file for the library is not separated, the SECTION and PINSWAP commands will not work.

To use the SECTION and PINSWAP commands, installations must have the program:

/usr/bin/section/section

This is the program that figures out the section and pin assignments for the various parts. Installations must also include the following files:

/usr/bin/secassign
/usr/bin/makechipsfiles
/usr/bin/makedrawingnames
/usr/bin/maketextfile
/usr/bin/makewritefiles

Mixed Text and Graphics Documents

3.22 INTRODUCTION

You can create mixed text and graphics documents interactively using the Graphics Editor's set of graphics tools. You can also add graphics to existing text. The need to physically cut and paste drawings into text is eliminated.

To avoid confusion, the term "document" means a mixed text and graphics paper like this one. A drawing is a schematic created using the Graphics Editor. A text file is an ASCII file that is part of the UNIX file system. Text files are created using a text editor such as VI (in UNIX) or emacs (on a VAX).

This document is divided into several sections. The first explains how to add drawings to an existing text file, thus creating a document. The next two sections describe how to create a document interactively with the Graphics Editor and how to edit an existing document. These first two sections will allow you to begin creating documents. The final sections are more advanced and give specific warnings and conventions.

3.23 ADDING DRAWINGS TO EXISTING TEXT FILES

The FORMAT command is used to add drawings to an existing text file. It is useful when documentation is already written and there are figures that must be cut and pasted into the document. By adding the drawing names to the existing text files and formatting them using the Graphics Editor, no cutting and pasting need be done.

FORMAT | UNIX_file_name <CR> SCALD_drawing_name

The FORMAT command takes a UNIX ASCII text file, followed by a carriage return, and the name of the drawing the document is to be called. It then creates a SCALD .DOC document from it called SCALD_drawing_name.DOC. The UNIX file can be a text file that has been formatted by runoff (on the VAX) or nroff (under UNIX). Each page of the text file is turned into a page in a SCALD drawing. A page ends with the 60th line or a user inserted ^L (formfeed).

Each page created by FORMAT is 8 1/2 by 11 inches, with 6 lines per inch. The characters are slightly larger than the default character font (1.29 times the default) for easier readability.

Space must be left to allow the insertion of drawings into the document. At least two lines are needed for each inserted drawing. The first line must have an "&" in the first column, followed by the name of the SCALD drawing you want to insert. The second line must have the number of lines, N, that are allotted for the drawing (6 lines = 1 inch). You must then insert N blank lines. The Graphics Editor reads the named drawing, smashes it and then scales it to fit into the stated space. For instance:

```
& AN EXAMPLE.LOGIC.1.1
3
```

The following examples, the first in runoff and the second in nroff, will produce Figure 1.

In runoff Format:

```
.p
The 2 to 1 MUX.  IF S is high, the output, Y, is
I1.  If S is low, the output is I0.
.sk 2
.br;_&MUX.BODY
.br;4
.sk 4
```

In nroff format:

```
.PP
The 2 to 1 MUX.  IF S is high, the output, Y, is
I1.  If S is low, the output is I0.
.sp 2
.br
&MUX.BODY
.br
4
.sp 4
```

The 2 to 1 MUX. IF S is high, the output, Y, is I1. If S is low, the output is I0.

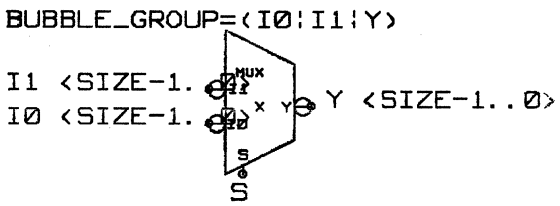


Figure 1: Using the FORMAT command

The FORMAT command adds tick marks on each corner of the document so that you can cut the Versatec output to the correct size.

3.24 CREATING DOCUMENTS INTERACTIVELY

Creating a document while in the Graphics Editor requires being able to add both text and drawings. To add text lines, use either the NOTE command or create a file of text using another editor and then add it to the document using the FILENOTE command. To add a figure, you create the drawing with the Graphics Editor and add it to the document using the SCALE command.

As a illustration, to create Figure 1, make a file with the text ("The 2 to 1 MUX...") using VI or ed and then create the MUX using the Graphics Editor. Now, from the Graphics Editor, edit the drawing EXAMPLE.DOC. Type FILENOTE UNIX_file and point to the spot where the note should go. The results of this operation are in Figure 2.

Next, use the SCALE command to add the drawing. Type "SCALE MUX.BODY" and point to the corners of the rectangle where the figure should go. The results are shown in Figure 3. Note that using the SCALE command causes all bodies to be "smashed" into their primitive pieces. The BODY definitions are not maintained.

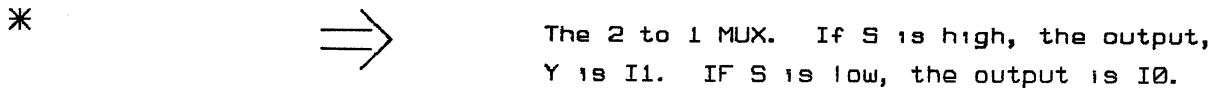


Figure 2: Using the FILENOTE command

The 2 to 1 MUX. If S is high, the output, Y is I1. IF S is low, the output is I0.

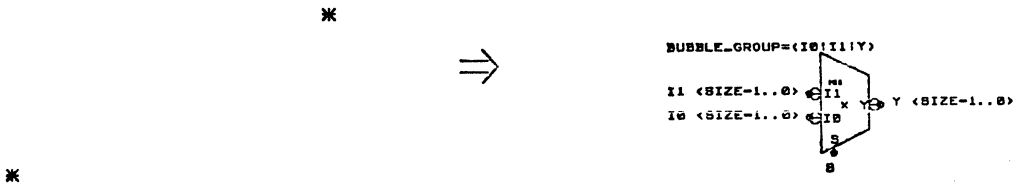


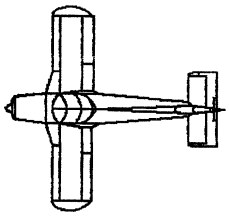
Figure 3: Using the SCALE command

3.25 EDITING A DOCUMENT IN THE GRAPHICS EDITOR - WARNINGS AND CONVENTIONS

CHANGING AN EXISTING DOCUMENT

Once a document is created, either using the FORMAT command or interactively with the FILENOTE and SCALE commands, it can be edited using the Graphics Editor. You may want to, for instance, rescale figures or make simple changes to lines of text. Modifications can be made using regular Graphics Editor commands such as MOVE, COPY, CHANGE, WIRE and GROUP.

As an example, Figure 4, is an existing document. Several changes need to be made. First, the word 'First' is misspelled in the second line. This can be corrected using the CHANGE command. Second, to add emphasis, the words Data, Wing Span and Accommodation might be underlined using the WIRE command. Finally, the scale of the drawing is too small. Delete the drawing by defining a group and then doing a group delete, and then read it at a different scale using the SCALE command. The results are in Figure 5.



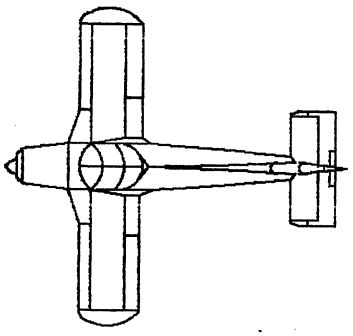
Aerotec 122 Uirapuru (Brazil)

Frst Flight 1965

Data: Standard T-23 military version
Cuising speed (Model A-122A) is 155mph
Powered by 108hp Lycoming Engine
Wing Span: 27 ft 10 3/4 in (8.5m)

Accommodation: Two seats side by side, with dual controls. Baggage compartment, capacity 66 lb (33 kg), aft of seats.

Figure 4: An Existing Document to Edit



Aerotec 122 Uirapuru (Brazil)

First Flight 1985

Data: Standard T-23 military version
Cuising speed (Model A-122A) is 155mph
Powered by 108hp Lycoming Engine
Wing Span: 27 ft 10 3/4 in (8.5m)

Accommodation: Two seats side by side, with dual controls. Baggage compartment, capacity 66 lb (33 kg), aft of seats.

Figure 5: The Existing Document Changed

THE .DOC DRAWING TYPE IN THE GRAPHICS EDITOR

Documents made using the FORMAT command are .DOC drawings in the Graphics Editor. Editing a .DOC drawing is different from editing a regular schematic (.LOGIC, .TIME, .BODY, etc.). First, the grid is set up so that there are 6 grid spaces per inch on the final plot (the grid is set to 0.166) In addition, when a DOC drawing is written, only the ASCII and binary representations are saved. There is no need to create a connectivity representation because documents are not read by the SCALD Compiler.

PRINTING A DOCUMENT WITH THE GRAPHICS EDITOR

When a document is created using the FORMAT command, tick marks are placed at the corners of the page. These tick marks serve 2 purposes. First, they allow you to hardcopy at the default scale (HA 1 ;) and get an 8 1/2 by 11 inch page. Second, the tick marks are used to cut the plotter paper to the correct size. Therefore, you should be sure not to delete the tick marks on the sides of the document or to palce text outside the tick marks, otherwise the page will not hardcopy correctly. If the marks are deleted or the page is created manually, type the following to create the tick marks:

```
DOT (410,4864) ;
DOT (410,-50) ;
DOT (4648,4864) ;
DOT (4648,-50) ;
```

Make sure that all the text and graphics lie within the box created by the dots, and the page will always be plotted correctly.

3.26 USING RUNOFF ON THE VAX TO CREATE A TEXT FILE

SETTING UP THE PAGE MARGINS

Several rules must be followed when setting up the page margins for a runoff text file. The default page length works fine (59 - 60 lines per page) so this doesn't need to be altered. However, you need to be aware of the margins that are used. The FORMAT command centers documents created with the default margins; if the margins are changed, the page may not be centered.

INCLUDING REFERENCES TO DRAWINGS

When including GED drawings, you must know where the first column is on the page. For instance the runoff example above will work correctly with the default margin. If, for example, a drawing is included in a list, the runoff input might look like this:

```
.ls
.le
Administation:

.i-9;_&ADMIN ORG CHART.LOGIC.1.1
.i-9;9
.sk 9
```

COPYING THE FORMATTED TEXT TO THE CLUSTER

When coying a document that has been formatted from the VAX to the SCALD station, use the cpfromvax utility. Do not use cp because runoff puts 2 carriage returns at the end of every line. The cpfromvax utility deletes the extra carriage return. (Don't use cpfromvax to copy format ASCII documents because the lines will be all strung together). If you don't use cpfromvax on your runoff files, your documents will be double spaced when run through the Graphics Editor.

ITALICS, BOLDFACE AND UNDERLINING

The Graphics Editor only supports one text font (although it can vary in height). Therefore, the Graphics Editor ignores any italcize and boldface commands.

3.27 USING NROFF IN UNIX TO CREATE A TEXT FILE

Several rules must be followed when using nroff. The -ms macros should be used to create the document. For more information on these macros, see "Typing Documents on the

UNIX System: Using the -ms Macros with Troff and Nroff" by M.E. Lesk.

SETTING UP THE PAGE MARGINS

The default page length is too long for the font created by the Graphics Editor. To create pages with 59 lines, use the command

```
.pl -7
```

at the beginning of the file. The default page margins center correctly on the Graphics Editor drawing page.

ITALICS, BOLDFACE AND UNDERLINING

Because the Graphics Editor only supports one text font (although it can vary in height), italicize and boldface commands are ignored.

Defining Bubble Groups in the Graphics Editor

3.28 INTRODUCTION

This section describes how bodies with bubble groups are created. It is assumed that you have an understanding of how bodies are created.

3.29 CREATING BODIES WITH BUBBLE GROUPS

For a pin to be bubbleable, it must have two characteristics : The correct physical construction and the correct bubble_group properties. The correct physical properties are needed since the pin can be drawn in two different physical states, and the bubble groups are needed to indicate which pins bubble simultaneously.

The physical requirements for a bubbleable pin are simple: The connection must be directly on the circle, and there must be a line from the pin that goes across the diameter of the circle. When the body is displayed, either the circle or the line (but not both) will be displayed. If the circle is displayed, the pin is in the BUBBLED state, if the line is displayed the pin is in the NON-BUBBLED state.

Bubble_groups define which pins change state when a given pin is bubbled. For example, if pins A, B and C are in one bubble group, then if any of them is bubbled they all will be bubbled.

Bubble_groups are indicated by properties with the name BUBBLE_GROUP attached to the origin of the body. Each BUBBLE_GROUP property defines one bubble group. The syntax is:

<bubble_group_name>(<abbrev>|<abbrev>|<abbrev>|<abbrev>...)

Where <abbrev> is a non-ambiguous abbreviation for a pin name (of a bubbleable pin). The character '|' is the vertical bar character; it should not appear in your pins names.

The bubble_group name is a single letter (case not relevant). All bubble_group properties of the same name together define one large bubble group. It is not necessary to name bubble groups unless you wish to define one that cannot fit on a single line in the editor. (This limit is 80 characters.)

There is an additional feature called an ASYMMETRICAL bubble group. If pins A and B are members of such a group, then bubbling A will bubble B, but bubbling B will have no effect on A. This is only useful for a small number of bodies, of which the most prominent example is the XOR gate (or any parity generator). If an XOR gate has inputs A and B, and output Y, then the bubble behavior should be:

If you bubble	Then should also bubble
A	Y
B	Y
Y	A (but not B)

There is no way to express this using conventional bubble groups, but it can be expressed as follows:

(A^Y)
(B^Y)
(Y^A)

An asymetrical bubble group has the syntax

(<pin1>^<pin2>|<pin3>|...)

Which means that if the pin <pin1> is bubbled, all the other pins are bubbled, but if any of the other pins are bubbled, there is no effect.

Once the groups have been defined, the next step is to tell which pins start in the bubbled state, and which start in the non-bubbled state. The BUBBLED property, also attached to the origin of the body, contains this information. The syntax of the bubbled property is:

(<abbrev>|<abbrev>|<abbrev>....)

Where the <abbrev>s are the abbreviations for the pins that are default bubbled.

Editor File Formats

3.30 INTRODUCTION

This document describes the format of files written and read by the Graphics Editor. These formats are subject to change without notice; they have been changed several times in the past and will continue to be changed in the future.

3.31 TYPES OF FILES

The editor writes out five types of files:

- ASCII files: These files are script files that can be used to generate any drawing except for BODY drawings. Consisting of a set of editor commands, they are sufficient to recreate the drawing from point zero. They work as if the commands were typed at the keyboard. In releases before 7.25 these ASCII files were called LOGIC files.

- BINARY files: These files contain the same information as the ASCII file described above, but in a binary format that is quicker to read and write. This format is strictly internal and not described in this document.

- BODY files: These files contain descriptions of bodies in ASCII format. They include line segments, arcs, pin names, bubble groups, connection points, and default properties.

- CONNECTIVITY files: These files describe all the bodies on a drawing. The information includes the names of the bodies, the names of the signals tied to their pins (with bubble state), and the properties that belong to the body. Connectivity files, which are in ASCII format, are the only files used by the Compiler.

- DEPENDENCY files: These files list the UNIX directories that were the source for all bodies added to a drawing. These files are used by the update procedure, which allows drawings to be updated if any bodies are out of date. There are DEPENDENCY files for all drawings except BODY drawings.

3.32 ASCII LOGIC FILE FORMAT

These ASCII logic files are a specific type of text files that consist of commands to add each part in a drawing. In this sense they are "logic" files. The file is kept in no particular order; reading and writing a drawing reverses the order of items in a file.

Points are represented in text files by their coordinates, enclosed in parenthesis. Thus the point x=100, y=200 is represented by (100 200). In the rest of this section, a point is represented by <pt>.

Angles are represented by a number 0..7 where
0: 0 degrees
1: 90 degrees
2: mirror of 0 degrees
3: mirror of 90 degrees
4: 180 degrees
5: 270 degrees
6: mirror of 180 degrees
7: mirror of 270 degrees

ASCII LOGIC FILE IDENTIFICATION

Each ASCII logic file starts with the line:

```
FILE_TYPE = MACRO_DRAWING;
```

This line identifies the type of file to the system.

ASCII LOGIC FILE CONTENTS

BODIES can use as many as four lines in the ASCII logic files. The description of the body in the ASCII logic file follows the form listed below:

```
FORCEADD <name>
[R angle]
<pt> ;
```

The name includes the version. The angle is optional. FORCEADD is used so that a placeholder is created if the body is not found. If the body has a color, then the next line is

```
PAINT color <pt>
```

Descriptions of wires in the ASCII logic files consist of a single line that follows the form:

```
WIRE line_type pattern <pt> <pt> ;
```

The linetype includes both the color information and whether the wire is thin or heavy. If the number is converted to binary, the least significant bit is the thin/heavy bit (0 = thin, 1 = heavy). The remaining seven bits pertain to the color.

-16384 <= pattern < 16384. If the pattern = -1, the line is filled. Using the "DISPLAY PATTERN" command, one finds that there are six defined patterns in GED:

1. -1
2. 273
3. 682
4. 2175
5. 3135
6. 4383

DOTS are written out as:

```
DOT type <pt> ;
```

where type = 0 if the dot is open and 1 if it is filled. (If the type is not 0 or 1, the dot is assumed to be open.) If the dot is colored, it is followed by a PAINT command.

CIRCLES and ARCS are written out as:

```
CIRCLE <pt1> <pt2> ;
or
CIRCLE <pt1> <pt2> <pt3> ;
```

where they are specified in the same way as described in GED manual. If the circle is colored, it is followed by a PAINT command.

NOTES are written out on three or four lines. The first two are:

```
FORCENOTE
<contents>
<pt> angle;
```

The forcenote command is similar to the note command in the editor except that the forcenote command terminates after

reading one note. This limitation is necessary because the normal note command continues to interpret the rest of the file as notes. If the note is not the default size, there is a fourth line:

```
DISPLAY <size> <pt> ;
```

This procedure makes the text the correct size. If the note is colored, it is followed by a PAINT command.

PROPERTIES are written out as two to four lines. They occur directly after the object they are attached to. The format of the first two lines is:

```
FORCEPROP <default_status> LAST <name> <value>
[R angle]
J justification_type
<pt> ;
```

The forceprop command is similar to the property command in the editor except it takes a <default_status> flag. This flag is necessary for correctly handling changes to properties on library bodies. The <default_status> flag can have three values: two if the property is known to be non-default (i.e. one that the user added to the ASCII logic drawing); one if the property is known to be default (i.e. one that comes from the body definition); or 0 if the status of the property is unknown (i.e. an undefined variable whose status is determined when the body definition is searched). LAST is a keyword indicating the property is to be attached to the last object or wire entered.

The angle is optional.

The next line describes the text justification. A value of 0 means that the text is left justified, and a value of 2 means it is right justified. If no justification is given, the property is created with the current default justification. If an illegal justification is given, the system assumes that the user wishes to have left justification.

If the property does not have the standard visibility, it is followed by a DISPLAY command to set the visibility of the name and the value. If the property is a PIN property, then the keyword LAST is replaced by the keyword LASTPIN followed by a <pt> describing the location of the pin in absolute coordinates. If the property is attached to another property, then the keyword LAST is replaced by the keyword LASTPROP.

If the property is colored, it is followed by a PAINT command.

Bubbled pins for an object are written out using the format:

FORCEBUBBLE <pt> ....

All pins that are not in their default bubbled state are listed.

END OF THE ASCII LOGIC FILE

The file ends with a line simply containing:

QUIT

3.33 FORMAT OF BODY FILES

Body files are written out in an abbreviated format. They are not read with the main editor input parser, so they are not tolerant of errors. Bodies are composed of seven elements: Lines, Arcs, Text, Connections, Bubble groups, body properties, and pin properties. As in the ASCII logic files, all coordinates are in 0.002 inch units.

Lines require 1 line each in the body file. A thin line has the format:

L x1 y1 x2 y2 pattern

A thick line has the format

M x1 y1 x2 y2 pattern

The pattern is optional and -16384 <= pattern < 16384. If the pattern is -1, the line is filled.

Arcs require one line each in the body file. The line has the format:

A Xcenter Ycenter Radius Start_angle Stop_angle

The center and radius are in integer units, and the start and stop angles are measured in degrees counterclockwise from the X axis. They are in floating point.

Text strings require 2 lines each in the body file. The first line gives the specification of the text; the second gives the contents. The first line has the format:

```
T x y angle slant size over inv just font Nch color

x,y     integer reference point for text
angle   real    0.00, 90.00, 180.00, 270.00
slant   real    (not implemented)
size    integer height of characters
over    0-1     (not implemented)
inv     0-1     (not implemented)
just    0-2     0=left justified, 2=right justified
font    0-4     (not implemented)
Nch     integer number of characters
```

The next line consists only of the NCH characters of the text string.

Connections require one line each in the body file. The contents of the line depend on whether the pin is bubbleable of not. The format is:

```
C x y Name bubbleable (default_state x2 y2 x3 y3) f size angle just
```

In this command line, the portion in parentheses is present only if the pin is bubbleable. The term name is a quoted string containing the name. The terms bubbleable and default_state are both integers: 1 if TRUE and 0 if FALSE. The points represented by X2,y2 and x3,y3 are those for the bubbleable pins. The letter f is 1 if the dot on the connection is filled, and 0 if open. Size is the size of the pin name string, and the GED default is 41. The word angle in the line above is the angle of the pin name string attached to the connection (0 = 0 degrees, 1 = 90 degrees, 2 = 180 degrees, 3 = 270 degrees). The abbreviation just shows the justification of the string (R = right, L = left).

For example, an entry for a bubblable pin could read as:

```
C 50 50 "Y<0>" 1 0 100 50 50 50 0 41 0 R
```

For a non-bubblable pin, the entry might be:

```
C 50 50 "Y<0>" 0 0 41 2 L
```

Body properties require one line each in the body file. The format is

```
P name value x y angle slant size over inv just font NV VV IP

name    quoted string   name of property
value   quoted string   default value of property
NV      0-1             name is visible by default
```

VV	0-1	value is visible by default
IP	0-1	1 if property is a parameter

The other numbers describe the text in the same manner as the T command.

Pin properties require one line each. The start with an X, rather than a P, and occur directly after the connection they are associated with.

Bubble groups require several lines apiece in the body file. They start with a line beginning with B and end with a line containing only the word END. Each bubble group is on a line by itself, with the format:

(name1, name2, name3, ...)

where all the names are quoted strings. If the bubble group is asymmetrical, the first comma is replaced by a colon.

3.34 FORMAT OF CONNECTIVITY FILES

Connectivity files describe the components in a drawing, how they are interconnected, and the names of the signals that connect them. Connectivity files are the only files read by the compiler. This file format was changed in the 7.0 release of GED. First the new format is described and then the old one. The SCALD Compiler will continue to read the old format but post-6.0 Compiler features will not be supported won't be supported in the old format.

Connectivity Format

There are only four types of items in a connectivity file: the header, the NET section, INVOKE commands, and comments.

Each connectivity file has the form:

```
FILE_TYPE = CONNECTIVITY;
{GED_Release: date and number}
[<expr property>]
[<nets>]
[<invokes>]
END.
```

where the second line is a comment, the third is the EXPR property from the drawing body, the fourth is the net section, the fifth is the invoke commands, and finally an END. The EXPR, net, and invoke sections are optional. The

continuation character for lines in a connectivity file is '~'. This character can occur anywhere in the line, even in the middle of words, but must be followed by <CR><LF>. GED limits line length to 80 characters, so it puts out a continuation character for lines longer that 80 characters.

Comments

Comments begin with an open brace "{" and end with a close brace "}". They may appear anywhere in a connectivity file except in the middle of identifiers or quoted strings and may cross lines.

Expr Property on Drawing Body

<expr property> ::= EXPR=<expression string>;

The expression string is the expression property value from the drawing body. For example

EXPR="SIZE=10";

Nets

Each time GED writes a connectivity file, it numbers the all nets. The NC net is always net 0 and unnamed signals are also numbered. These numbers will not be the same each time the connectivity file is written.

<nets> ::= <constant> <net name string> [<property list>] ;

The constant is the net number. The net name is either the signal name for the net or the unnamed string created by GED; the net names string is always quoted. The property list is optional.

<property list> ::= { <identifier> <string> }

The identifier is the property name; it can only contain letters, digits, and underbar ('_') and must begin with a letter. There are two reserved identifiers -- FILE_TYPE and END. The string is quoted. An example of several net entrys is:

```
2"UN$1$2P$A";
3"A\NWC"LOAD"37"CONNECTED_TO"PAGE 4";
```

Invocation of Components

Each component in the drawing is described as follows in the connectivity file:

```
<invokes> ::=
%<invoke name string>
<version str>,<xy str>,<rotation>,<directory str>,<path str>;
[ <parameter property list> ] ;
[ <property list> ] ;
{ <pin name string> [<property list>] <constant>; }
```

The invoke name string is the name of the component and is quoted. The next line contains body properties that are always output -- the body version number, in quotes, the (x,y) coordinate of the body on the page, the rotation of the body, in quotes, the name of the directory where the body came from (not rooted, so /u0/lib/lsttl/lsttl.lib is shortened to lsttl.lib), and the path property. If any of these properties doesn't exist, the null string ("") is used. The rotation string is :

```
0: 0 degree rotation
1: 90 degree rotation
2: mirror of 0  degrees
3: mirror of 90 degrees
4: 180 degree rotation
5: 270 degree rotation
```

The property list is optional but the semicolon is not. The parameter property list is the PARAMETER...END_PARAMETER block in the old format. It is optional and is the same format as a property list. The pin name string is quoted and the constant is the number of the net attached to it. The net numbers are assigned in the net section. The <pin name string>... line takes the place of PIN and BINDING section in the old connectivity format. An example of an invoke:

```
%"LS00"                              {body name}
"1","(100,345)","0","lsttl.lib","2P"; {body information}
SIZE"SIZE";                          {parameter property list}
COLOR"RED"SECTION"U32";              {body property list}
"A"23;                                {pin names}
"B"5;
"Y"OUTPUT_LOAD"(50.0,-50.0)"3;
```

An example with no path property string and no body property list:

```
%"LS02"                              {body name}
```

```
"2","(500,1234)","3","lsttl.lib","";      {body information}
;                                        {parameter property list}
COLOR"RED";                              {body property list}
"A"23;                                    {pin names}
"B"5;
"Y"OUTPUT_LOAD"(50.0,-50.0)"3;
```

The Old Format (pre-7.0 release)

There are only three types of items in a connectivity file: the header, INVOKE commands, and comments.

Connctivity File Identification

Each connectivity file begins with the 3 lines:

```
FILE_TYPE = MACRO_DEFINITION;
      {GED_Release: date and number}
      MACRO
```

where the second line is a comment and ends with the line:

```
END_MACRO.
```

Comments

Comments begin with an open brace "{" and end with a close brace "}". They may appear anywhere in a connectivity file except in the middle of identifiers or quoted strings.

Invocation of Components

Each component in the drawing is described as follows in the connectivity file:

```
INVOKE <name>
   PROPERTY
   <property list>
   END_PROPERTY;
BINDINGS
<bindings list>
END_BINDING;
END_INVOKE;
```

Both the properties section and the binding section are optional. The <property list> describes the properties of

the component, both pin properties and body properties. The <bindings list> describes the pins of each component, their bubble state, and the signals to which they are connected.

Format of the Property List

The property list consists of PIN properties (those belonging to a specific pin of a component) and BODY properties, which belong to the component as a whole.

A body property is represented as:

```
BODY
name = value;
END_BODY;
```

The name is an identifier and is not quoted. The value is an arbitrary string and is quoted. All bodys have an XY property that gives the location of the body origin.

Pin properties are represented as:

```
PIN pin_name :
name = value;
END_PIN;
```

The pin name is quoted.

Bindings

Bindings are represented as follows:

```
BINDINGS
pin_name = signal_name ;
pin_name = signal_name
etc.
END_BINDINGS;
```

Both the pin name and the signal name are quoted. The pin name has a "\B" appended to it if the pin is bubbled. Unnamed signals are named according to the standard convention. All named signals are given a unique number and that unique number is output as the NN property.

Properties On Properties

Properties are written as

```
prop_name = prop_value : p1="v1",
                          ...
                         pN="vN";
```

with the property's properties listed after a colon and separated by commas.

3.35 DEPENDENCY FILE FORMAT

The first line of a dependency file is a logic file name, followed by ':', followed by a blank separated list of body file names. The names are all UNIX file names with paths. For example, for the logic drawing 'MY EXAMPLE.LOGIC.1.1', the dependency file might be:

```
myexample/logic.1.1 : \<cr>
        /u0/lib/standard/1s00/body.1.1 \<cr>
        /u0/lib/standard/1s03/body.1.1 \<cr>
        adder/body.1.1 \<cr>
        shifter/body.1.1
```

'<cr>' is used to continue across lines. Lines continued in this way must begin with <tab>. It is assumed that files are referenced from the UNIx directory with the SCALDdirectory that contains the logic drawing. From the directory /u0/class, it is only necessary to say shifter/logic.1.1, not /u0/class/shifter/logic.1.1. However, parts that are added from scalddirectories not in the current UNIX directory must be given a full path, as in, for example

```
/u0/lib/standard/1s00/body.1.1).
```

The entire path name must be written out, and no wild cards are allowed.

The last line in the dependency file is

```
/u0/editor/MakeAddToList "drawing_name.extension.version.page"
```

The drawing name is quoted, and all four parts of the name must be given.

This page has been intentionally left blank.

Back Annotation File Format

3.36 INTRODUCTION

This document gives the format for the file read by the Graphics Editor BACKANNOTATE command. If users do not use the backannotation file generated by the Packager, there is no guarantee that such information is consistent with the physical design.

3.37 WHAT THE GRAPHICS EDITOR EXPECTS

The back annotation file contains physical information grouped by drawing. The file should be called backann.cmd. The first line is

```
FILE_TYPE = BACK_ANNOTATION;
```

The information in the file includes:

1. The name of the drawing. The line should look like:

   ```
   DRAWING = "SCALD drawing name";
   ```

   The drawing name must be quoted.

2. The name of the body within the drawing. This is specified by giving the body's name and path property and any information to be attached to the body. If there is no information to be attached, the line should be:

   ```
   BODY = "name","path_property";
   ```

   If properties are to be attached, the above ends with a colon and is followed by property name/value pairs, separated by commas. For instance:

   ```
   BODY = "name","path_property":
          prop1 = "value1",
                ...
          propN = "valueN";
   ```

   Property names must be 15 characters or less. Property values are quoted, but not property names. There MUST be spaces around any equals sign (=). The only property that should be attached to a body is the LOCATION designator.

3. The name of a pin on the body, as well as any information to be attached to the pin. Vectored pins cannot be annotated. The pin name should be quoted. If a pin does not have any properties, the pin should not be listed. For instance:

```
"pin name": prop1 = "value1";
```

   Property names must be 15 characters or less. Property values are quoted, but not property names. There MUST be spaces around any equals sign (=). The only information given should be the pin number (PN property).

4. The name of a net, in user syntax form, and any information attached to the net. Only scalar nets can be annotated. The form is:

```
NET = "net name": prop1 = "value1",
                    .....
                  propN = "valueN";
```

The last line in the file should be

```
END.
```

The back annotation file should not contain information for bodies with SIZE and/or TIMES properties except as follows:

1. A LOCATION property for the body should be output only if ALL SIZE replicated logical sections of the body are allocated to the same physical part.

2. Pin numbers for pins of SIZE replicated components should be output only if the pin is common to all sections and appears on the same pin for all.

There should be no information for a drawing that is used more than once in the design.

3.38 AN EXAMPLE

```
FILE_TYPE = BACK_ANNOTATION;
DRAWING = "C C.LOGIC.1.1";
BODY = "LS74","6P":
LOCATION = "U32";
"CLOCK*": PN = "1";
"D": PN = "2";
BODY = "LS08","5P":
```

```
LOCATION = "U34";
"Y<0>": PN = "1";
NET = "XOUT":
FOO = "BAR";
DRAWING = "C C 2.LOGIC.1.1";
BODY = "LS74","6P":
LOCATION = "U34";
"CLOCK*": PN = "3";
"D": PN = "2";
BODY = "LS08","5P":
LOCATION = "U32";
"Y<0>": PN = "7";
NET = "XOUT":
FOO = "MUMBLE";
END.
```

3.39 WHAT THE GRAPHICS EDITOR DOES WITH THE INFORMATION

The Graphics Editor reads the file containing the information to be added to drawings. The file contains the following items:

1. The name of the drawing.

2. The object to which the information is to be attached. The object may be a body, a pin or a signal.

3. The information to be added in the form of a property name/value pair.

The Graphics Editor decides where the information to be added is to be placed. The Editor may use some heuristics or may use manually entered placeholders. The information added is in the form of properties.

Information added to a drawing through back annotation is interpreted differently than properties added manually to a drawing. The Editor considers back annotated information to be "comments" and they are not output to the connectivty file. That is, information back annotated does NOT appear in the SCALD system data bases; it only resides in the drawings. Properties added by the BACKANNOTATE command begin with the character $. For instance, LOCATION becomes $LOCATION.

Back annotated information will replace values that are already present in the drawing (a new $LOCATION value will replace an old $LOCATION value). However, a hard property value CANNOT be replaced by a back annotated value (a $LOCATION value will not replace a LOCATION value). Furthermore, back annotated information cannot be manually changed or reattached, but may be moved or deleted.

Vector Plot Format From GED

3.40 INTRODUCTION

This section describes the vector plot format of the plot file produced with the Graphics Editor's VECtorize command. Execution of this command produces an ASCII plot file that can be used for transmission to other machines or that can be used to drive a pen plotter (with the aid of a format conversion program).

3.41 THE FORMAT

The vector output file represents a plot of the entire circuit as maintained in the display list (i.e., the file contains the entire drawing, not just the portion shown on the screen).

All drawing are defined by three types of primitives: LINES, ARCS, and TEXT_STRINGS. Each primitive begins on a separate line of the file. The first character of the line specifies the primitive type. Individual parameters within the line are separated by spaces. All units are nominally 0.002 inches.

Line Primitive

A line primitive is identified by an "L" as the first character in the line. The line primitive is defined by the six integers that follow. Note that each integer is separated by a space.

```
L  X1  Y1  X2  Y2  pattern  line_type
```

The first four integers (X1, Y1, X2, Y2) are the line's endpoint coordinates. Pattern is an integer from -16384 to 16383 with -1 representing a solid line. The line_type describes both the color and thickness of the line; if line_type is converted to a binary value, bit 0 defines the thickness (0=thin) and the seven most-significant bits define the color.

Arc Primitive

An arc primitive is identified by an "A" as the first character in the line. The arc primitive is defined by the five numbers that follow.

```
A  X  Y  radius  start_angle  stop_angle
```

The angles, which are in degrees, are in floating point; all other numbers are integers. X and Y are the coordinates of the center of the arc or circle. The angles are measured counter-clockwise from the X axis. If radius is negative, the circle is filled; otherwise it is open.

Text String Primitive

A text string primitive is identified by a "T" as the first character in the line. Each text string primitive consists of the following four lines; each line is terminated by a line feed character.

```
  T  X  Y
->angle  slant  size  overbar  inverse_video
  justification  font
  string
```

The individual parameters within each line are:

X, Y: origin point of text string
angle: 0, 1, 2, 3 (for 0, 90, 180 and 270 degrees respectively)
slant: not implemented
size: integer (height)
overbar: not implemented
inverse_video: not implemented
justification: 0 = left, 2 = right
font: not implemented
string: the text string (not quoted)

For an example of how to convert the Valid Vector Plot Format to the HPGL (hewlett packard graphics language) format for display on an hp pen plotter, see the source in /u0/editor/lib/hpfilter.pas.