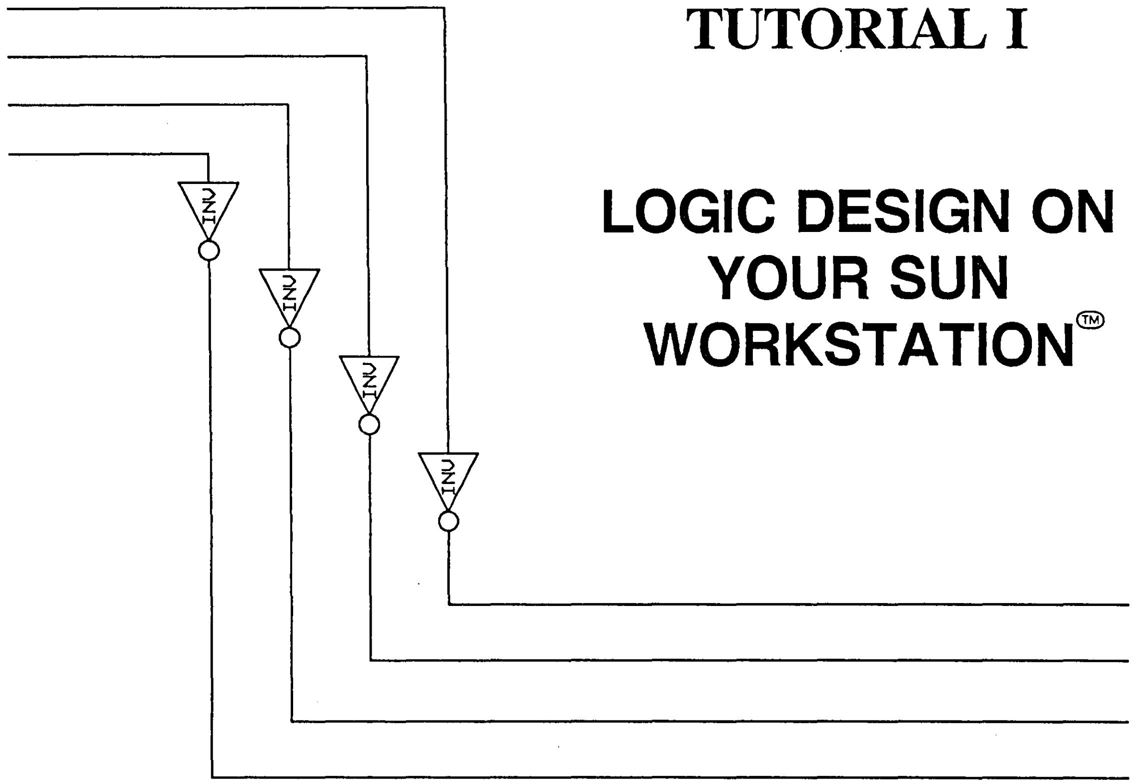
TUTORIAL I

LOGIC DESIGN ON YOUR SUN WORKSTATION™



January 15, 1989

Valid Logic Systems, Incorporated
2820 Orchard Parkway
San Jose, California 95134
(408) 432-9400 Telex 371 9004
FAX (408) 432-9430

PREFACE

This tutorial describes schematic entry with Valid's Graphics Editor (ValidGED) and the preparation of the design for a physical design system with the ValidCOMPILER and ValidPACKAGER analysis tools.

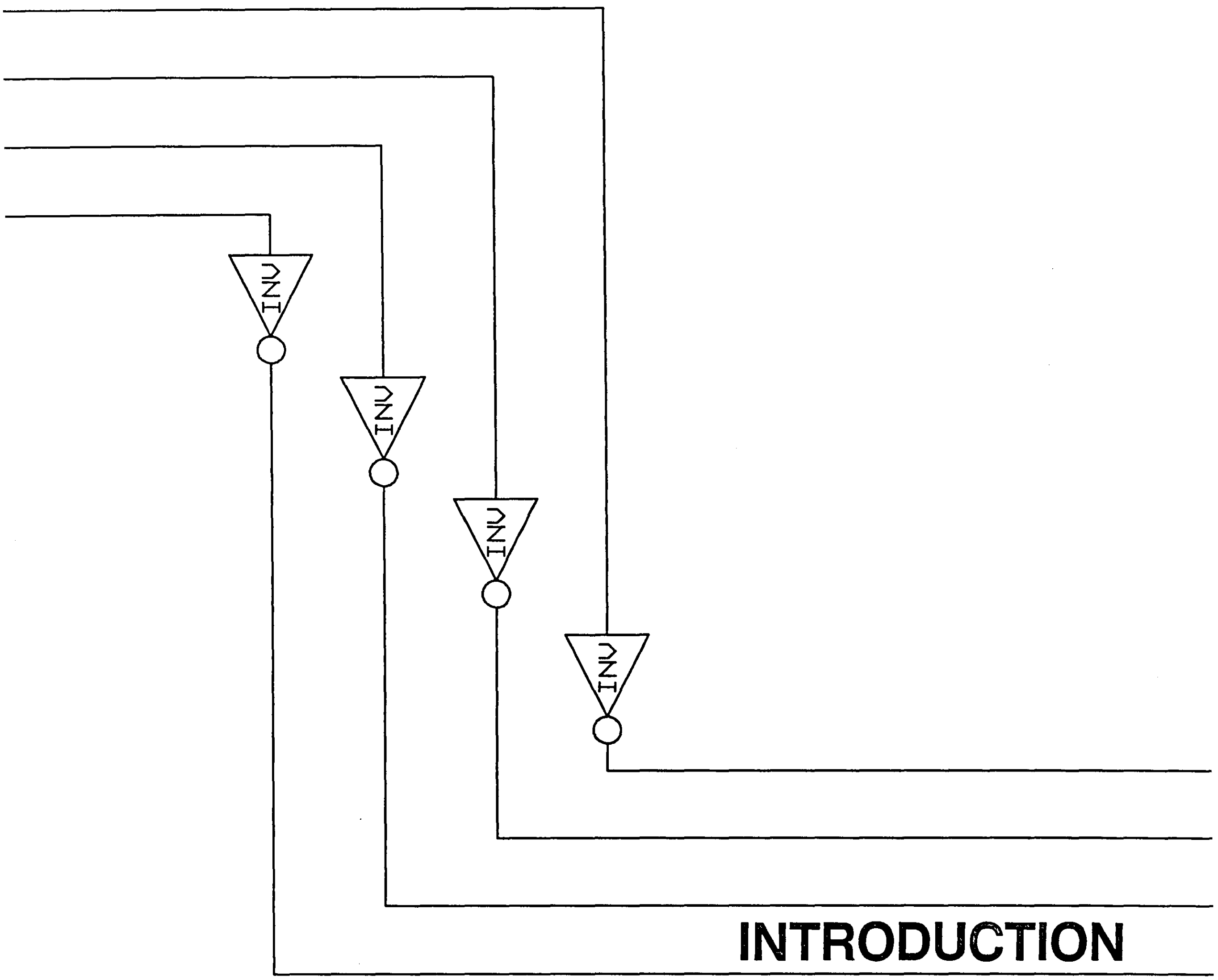
The tutorial applies to a Logic Design System on a Sun Workstation. If you purchased an Entry Design System (schematic capture only), lessons 8 through 11 do not apply to your system configuration. If you purchased a Validation Design System (simulation and timing verification), you should first go through all of the lessons in this tutorial and then complete the lessons in tutorial II.

This tutorial supports the following software releases:

• ValidGED	Release 9.0
• ValidCOMPILER	Release 1.4 (or later)
• ValidPACKAGER	Release 2.0 (or later)

If you use this tutorial with an earlier release of any of the above application programs, the exact operation of these programs is different from their descriptions in the tutorial. This tutorial is most effective when used with the above releases.

Table of Contents



INTRODUCTION

Welcome to Tutorial I for your Sun Workstation. This tutorial introduces you to your new workstation and software design tools. The tutorial's approach is hands-on and achievement-focused. Since your first priority is to get your work done, you start out right away with schematic entry. By the end of Lesson 2, you are already wiring a circuit. Using a subtractor circuit as a model, you proceed step-by-step through schematic entry. Along the way, you learn how to check for errors and you read special tips on how to get the most out of your system. The next three lessons explain several important concepts that give you the knowledge and power to best implement your Valid design tools to achieve the solutions you want.

Your system includes a group of software tools that you use to design systems interactively. This tutorial introduces you to the following programs:

ValidGED (Graphics Editor)
ValidCOMPILER
ValidPACKAGER
Interface Program (GSCALD)

and teaches you enough about UNIX to effectively use these programs.

Valid's CAE Software

Your software design tools enable you to create, modify, and manage logic designs. The interrelationship of the entire family of Valid's software design tools for the Sun workstation is shown in Figure 1.

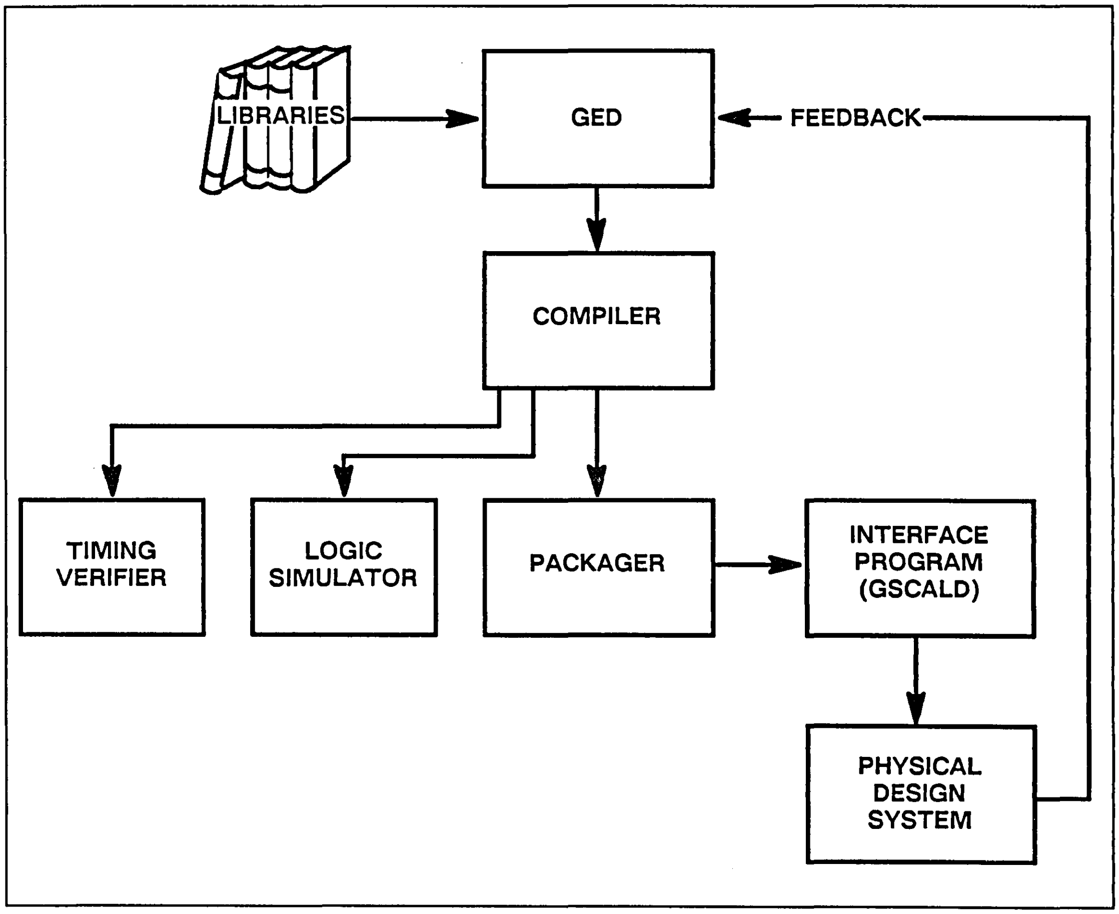


Figure 1. Software Design Tools

ValidGED: The Graphics Editor

You use the Graphics Editor (GED) to quickly enter and modify your design. Using a convenient menu of commands, you call up bodies (the graphical representations of library parts) from a full spectrum of Valid-supplied, user-created, or possibly third-party libraries. Then you wire these bodies together rapidly by drawing lines with the mouse. All or part of a design may be moved or copied very quickly.

The Compiler

ValidCOMPILER prepares your design for the Packager (ValidPACKAGER), the Timing Verifier (ValidTIME), or the Logic Simulator (ValidSIM). The design you create in GED is run through the Compiler to check for errors and to convert the design into a form that can be interpreted by one of the Valid analysis programs. ValidCOMPILER is called automatically whenever you run the Packager, the Timing Verifier, or the Logic Simulator.

The Packager

ValidPACKAGER reads the Compiler output, tests your design for loading and fan-out violations, and assigns physical part designators to the logical parts in your design. The Packager output files supply all of the basic information needed by a physical design system.

Interface Programs

Interface programs read the Packager output and produce files in the specific format required by a particular physical design system. You use the same procedure to run all interface programs. The Standard Interface Program, available with your Logic Design Tools (GED–Compiler–Packager), includes GSCALD, a general-purpose interface program. The GSCALD interface produces net lists and parts lists. By using GSCALD, you learn how to use the other interface programs.

Additional Analysis Tools

The basic group of design tools just listed forms the core of the Valid schematic capture and design verification system. GED, ValidCOMPILER, ValidPACKAGER, and the GSCALD interface are all covered in detail in this tutorial.

Two additional design verification tools, ValidTIME and ValidSIM, are optionally available. These tools and hierarchical design techniques are described in *Tutorial II.*

The Timing Verifier

ValidTIME uses timing models from the libraries to analyze partial or complete designs for timing violations and to produce a detailed report of signal behavior. An additional program, Plottime, produces detailed waveform diagrams of the timing behavior of each signal. ValidTIME can be used throughout the design cycle to eliminate hard–to–find timing errors.

The Logic Simulator

ValidSIM uses simulation models from the libraries to perform detailed logic simulation of a design at the component level. This powerful interactive program gives you complete control over the simulation and allows you to display signal behavior as waveforms or instantaneous values.

Tutorial Overview

This tutorial is made up of 15 lessons that divide naturally into three sections. The first section, Lessons 1 through 6, teaches you the basic GED commands by stepping you through an entire design entry cycle.

The second section, Lessons 7 through 11, teaches you how to use the Compiler, the Packager, and the GSCALD interface and how to perform back annotation of your drawings. You also learn a few UNIX commands and the basic commands of the screen editor, *vi*. There is just enough UNIX and *vi* for you to get your job done.

The third section of the tutorial, Lessons 12 through 15, explains the basic architecture of UNIX and your system, where drawings and libraries are stored, and how to manipulate your drawings and files to best use the power and capabilities of your system. The tutorial concludes with a brief procedure to get you started on a project of your own.

Tutorial Library Installation

A tutorial library is provided with your system. This is a library of components for use with the tutorial. Make sure that you have installed the library before you use this tutorial.

Signal Syntax

Signal names use a compact syntax to convey design information. All signal names in this tutorial use Signal Syntax 1. If your site uses a different signal syntax, you will have to make minor modifications to the signal names used in Lessons 6 through 11. For more information on signal syntax, consult the *SCALD Language Reference Manual.*

Command Syntax

GED commands can be entered in either uppercase or lowercase. The tutorial makes no attempt to describe all the possible syntax options of commands. For additional information on specific commands, see the appropriate reference manual.

Documentation Conventions

- Commands that you enter are displayed in `typewriter font`. *Italics* are used for the parts of a command or a character string that are variable, as in:

  Your working directory defaults to *susan*.wrk where *susan* is your login name.

- System responses are displayed in `typewriter font.`

- When embedded within the text, GED commands appear in **lower case bold** and UNIX commands appear in *lower case italics.*

- A function name enclosed in a double box ▭ means press the key named in the box.

- Steps you are to perform are numbered with large, bold numerals (**1**, **2**, etc).

- Special helps and tips for your information are marked by a small, bold checkmark (✓).

- The term *click* is used exclusively for the left mouse button. Click means place the cursor on an item, press the left mouse button, then release the button.

  The symbol ✓ illustrates the click action.

The Lesson Plan

To let you quickly find a particular topic or lesson, here is a summary of the contents of each lesson in the tutorial.

Lessons 1 - 6: GED

The first six lessons of the tutorial introduce the Graphics Editor (GED) and explain how to use it to create and modify drawings. A summary of GED commands is provided in Appendix A.

Lesson 1: GED - Getting Started

When you have finished with this lesson, you will know how to:

- Log on and off the system
- Access the Graphics Editor (GED)
- Recognize the GED screen and menu
- Use the mouse to select commands
- Type in commands at the command prompt
- Tell GED what libraries of components you want to use
- Check the name of the drawing you are editing
- Exit from GED

Lesson 2: Drawing

When you have finished with this lesson, you will know how to:

- Place library parts (bodies) on the screen
- Wire bodies together
- Move bodies and wires
- Correct wiring errors
- Refresh the screen
- Name signals
- Modify your drawing
- Save a drawing
- Get a hardcopy of a drawing

Lesson 3: Finding and Retrieving Drawings

When you have finished with this lesson, you will know how to:

- Get a list of your drawings by name
- Edit an existing drawing
- Edit a new drawing
- Understand drawing name conventions

Lesson 4: Looking

When you have finished with this lesson, you will know how to use the **zoom** command to:

- Center a drawing on the screen
- Enlarge a portion of a drawing to fill the screen
- Zoom in to enlarge a view of a drawing
- Zoom out to reduce a view of a drawing

Lesson 5: Time Savers

When you have finished with this lesson, you will know how to:

- Use the short form of GED commands
- Use softkeys to enter GED commands
- Get help on GED commands

Lesson 6: Subtractor

When you have finished with this lesson, you will know how to:

- Draw a more complex circuit
- Select versions of a library part
- Rotate library parts
- Wire parts together automatically
- Separate overlapping objects
- Check wire connections
- Check signal name assignments
- Attach properties
- Check for logical errors

Lessons 7 - 11: Completing the Design Cycle

The next five lessons of the tutorial teach you how to use the Compiler, the Packager, an interface program (GSCALD), and the back annotation feature. This section also introduces enough UNIX and *vi* for you to use your design tools effectively. A summary of UNIX and *vi* commands is provided in Appendix B. A brief description of each default command file and your other default files is provided in Appendix C.

Lesson 7: UNIX and Vi

When you have finished with this lesson, you will know how to:

- List files and directories to the screen
- Edit text files with the screen editor *vi*
- Edit your *startup.ged* file

Lesson 8: The Compiler

When you have finished with this lesson, you will know how to:

- Edit the Compiler directives file
- Specify drawing names and locations
- Specify libraries used in the design

Lesson 9: The Packager

When you have finished with this lesson, you will know how to:

- Edit the Packager directives file
- Run the Packager
- Consult the Packager listing file *(pstlst.dat)* for error information
- Correct typical Packager errors
- Consult the Compiler listing file *(cmplst.dat)* for error information
- Correct typical Compiler errors
- Attach physical properties to nets

You will also understand the flow of data through the Packager and the use of State files and Feedback files.

Lesson 10: Back Annotation

When you have finished with this lesson, you will know how to:

- Back annotate your schematic with physical reference assignments made by the Packager
- Manually assign physical reference designators (U-numbers)
- Manually assign physical sections

Lesson 11: The GSCALD Interface Program

When you have finished with this lesson, you will know how to:

- Create an interface command file
- Use the Packager output to run GSCALD
- Run other interface programs

Lessons 12–15: Using Your Design Workstation Effectively

The last section of the tutorial explains SCALD Directories, the UNIX file system, and the structure of Valid-created libraries so that you can more effectively use your design tools.

Lesson 12: SCALD Directories

When you have finished with this lesson, you will know how to:

- Recognize SCALD Directories
- Use multiple SCALD Directories
- Get a list of your current directories
- Get a list of the parts in one of your libraries
- Change the search order of your directory stack
- Borrow a drawing from another user

Lesson 13: Understanding UNIX

When you have finished with this lesson, you will know how to:

- Identify your current UNIX directory
- Move around in the UNIX file system
- Use full UNIX pathnames
- Print UNIX files to the screen

You will also understand where your GED drawings are stored in the UNIX file system.

Lesson 14: Understanding Libraries

When you have finished with this lesson, you will:

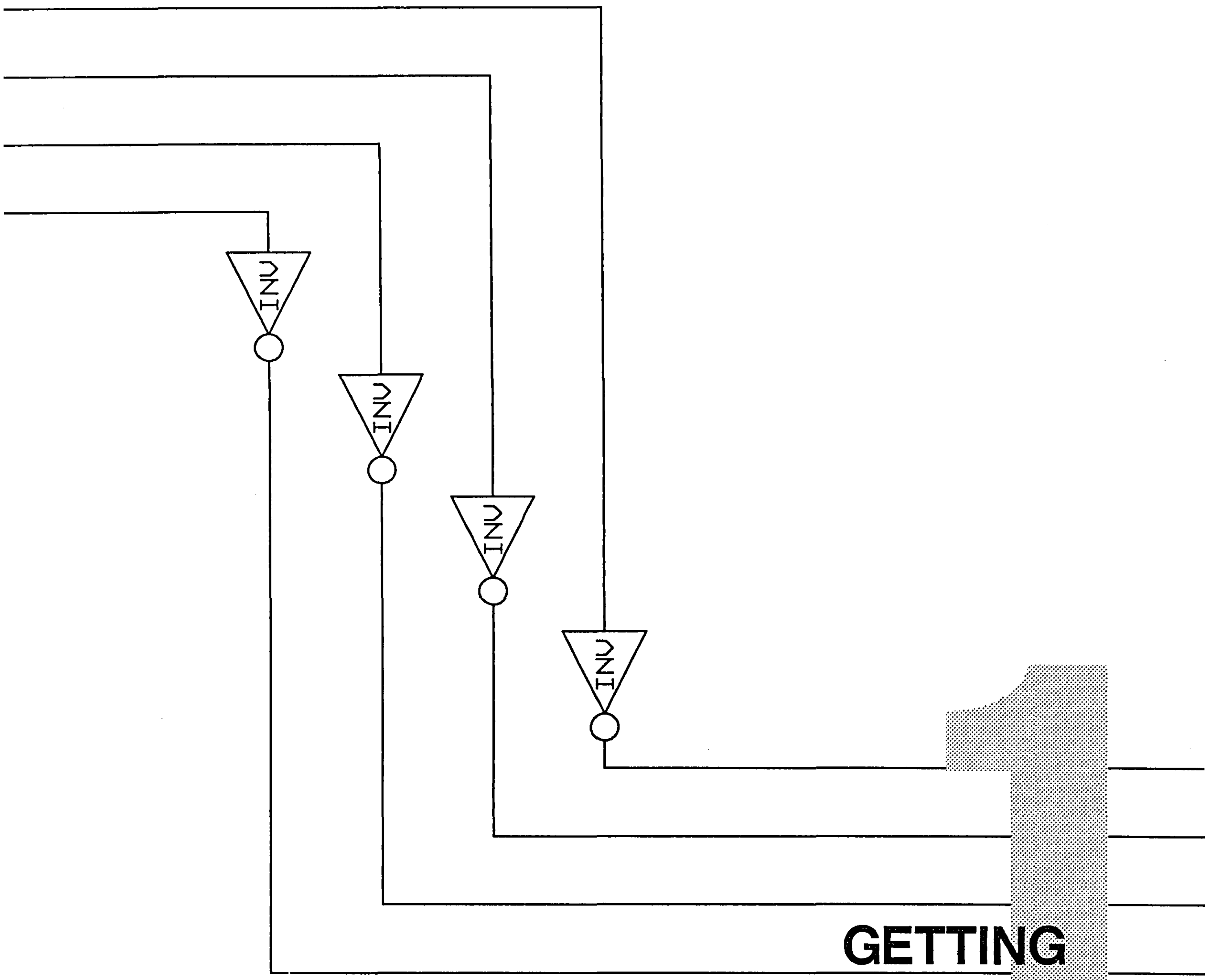
- Know where libraries are stored
- Understand the structure of Valid-created libraries
- Be able to locate and use the *master.lib* file
- Recognize the purpose of each drawing and file in a Valid library
- Understand how alternate models for compilation are stored for each library part
- Be able to use libraries effectively

Lesson 15: Starting a New Project

When you have finished with this lesson, you will know how to:

- Make a new directory in which to store a new design project
- Copy the necessary files into this new directory
- Edit your *startup.ged* and *compiler.cmd* files for a new project

LESSON 1



GETTING STARTED

In this lesson, you learn how to:

- Log on and off the workstation
- Access and exit GED
- Use the GED menu and the mouse

Before you can begin using GED, you need:

- A Sun workstation with UNIX and SCALD application software installed and running.
- The tutorial library installed on the system.
- A SCALD user account created by the system manager.

Logging On

The following procedure assumes that the workstation is running and that a full screen with the login: prompt is displayed. To log on:

1. Enter your user name at the prompt and press Return.

2. Enter your password at the `password:` prompt and press Return.

   If your account does not have a password, you are not asked for a password. If this is your first time logging in and you don't know your password, see your system manager.

When you have successfully logged in, the UNIX system prompt appears. The percent symbol (%) is the default prompt; you may have a different prompt.

Using the Sun Window System

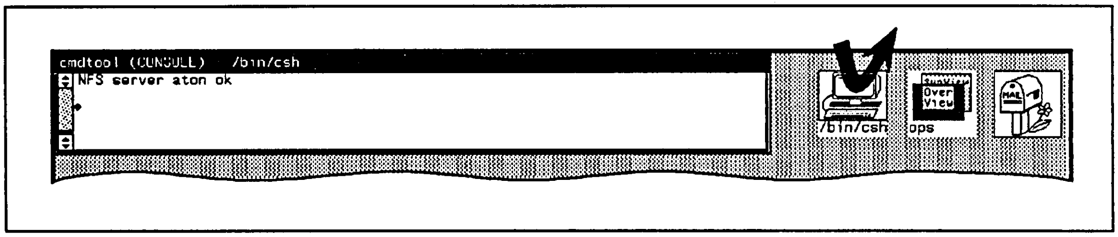
GED runs in a window created with the Sun Window System. To create a GED window:

1 Type

```
suntools
```

and press `Return`. A set of icons appears on the screen.

2 Select the /bin/csh icon by using the mouse to position the cursor over the icon and clicking the left mouse button.



```
cmdtool (CONSOLE)   /bin/csh
NFS server aton ok
```

The Shelltool – /bin/csh window appears.

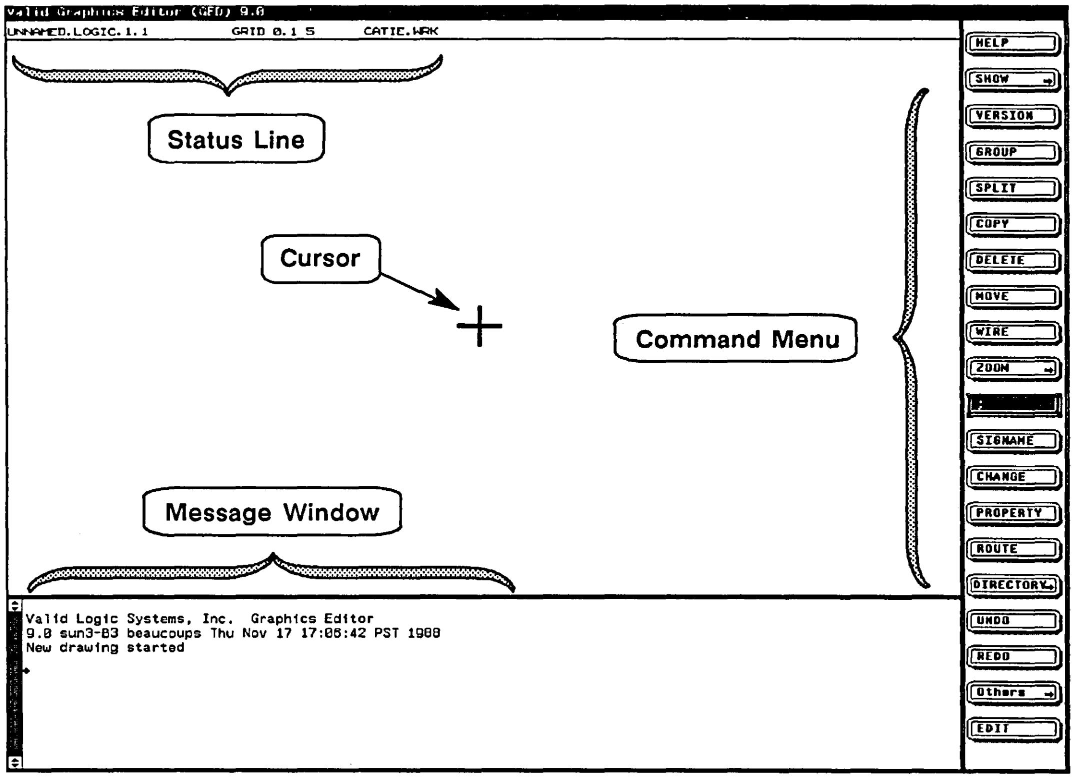
3 Move the cursor inside the window and type:

```
ged
```
`Return`

A new window is created and the Graphics Editor screen shown in Figure 1–1 is displayed.

The GED Screen

When you enter GED, the GED window is automatically created.



```
Valid Graphics Editor (GED) 9.0
UNNAMED.LOGIC.1.1          GRID 0.1 5        CATIE.WRK
```

```
Valid Logic Systems, Inc.  Graphics Editor
9.0 sun3-83 beaucoups Thu Nov 17 17:06:42 PST 1988
New drawing started
```

Figure 1–1. Typical GED Screen

The GED window includes:

- The cursor
- The command menu
- The message window
- The status line

The Cursor

The cursor moves around the screen as you move the mouse. You can choose a command from the menu by pointing to it with the cursor and clicking one of the buttons on the mouse. Later in the tutorial, you learn how to use the cursor and mouse to draw lines, position library parts, and change the size or view of your drawing.

The Command Menu

The command menu is the boxed list on the right side of the screen. It contains many of the GED commands. To select one of these commands, move the mouse so that the cursor is positioned within the desired box and click any button on the mouse. That box is then highlighted to remind you which command you are using.

Take a minute to look at the commands listed in the menu. Most of them are self-explanatory, and you learn later how each one of them works. One command on the menu is not self-explanatory and is very important – this is the semicolon (;). The semicolon indicates the end of a command. It tells GED that you are finished using one command and are ready to use another. You don't need to use the semicolon after every command, just at certain times. It's very useful when you choose the wrong command and want to go back and choose another.

When you use the keyboard to enter a GED command that is not on the menu, the command appears highlighted in the bottom box of the menu. This way you always know which command you are currently using, and you can execute it again without retyping it.

The Message Window

If you make a mistake during command entry (before pressing Return*), you can correct it by using the backspace key and retyping the command correctly.*

The message window is at the bottom of your screen. This is where you type in commands to GED and any text you want to appear on your drawing.

There are more GED commands than appear on the menu. To use a command that is not on the menu, you type in the command at the keyboard and then press Return.

GED output that requires more than one message window page pauses after each page and inquires:

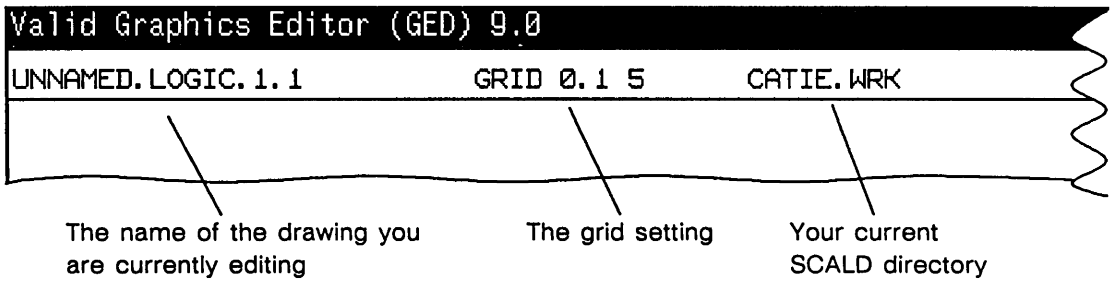
```
More? [ync]
```

The possible responses are:

y or Return	Yes. Present more information.
n or *q*	No. Do not print any more output.
c	Continue. Print the entire message output without pausing for page prompts.

The Status Line

The status line contains three items:


Valid Graphics Editor (GED) 9.0
UNNAMED.LOGIC.1.1 GRID 0.1 5 CATIE.WRK

The name of the drawing you are currently editing
The grid setting
Your current SCALD directory

When you access GED, `UNNAMED.LOGIC.1.1` is listed as the name of the drawing. You learn later how to change the name of a drawing, and how to save and retrieve drawings. For now, just remember that this is where you can check the name of the drawing you are currently editing.

The grid is off by default because you usually don't need to see it.

The grid setting has a default value of **0.1 5**. This means that there is a grid point every 0.1 inches, and that every fifth grid point is visible when you turn the grid on.

For more information on grids, see the ValidGED User's Manual.

We strongly recommend that you do not change the grid setting. In GED, components and wires can only be added and connected at grid points. The Valid component libraries depend on the default grid setting to function properly.

The current SCALD directory is where your present drawing can be found, and where it is stored (unless you specify otherwise). When you enter GED, your working directory defaults to:

usr/*catie*.wrk

The word *catie* is replaced by your actual login name. Your default working directory is determined by the system manager when your account is created. You learn later (in Lesson 12) how to specify other directories and more about directory structures.

Specifying Libraries

Before you begin to draw, you must tell GED which libraries of components you require. Type:

```
library tutorial [Return]
```

This command lets you use the library designed for use with this tutorial.

Exiting GED

The **quit** *command is the same as the* **exit** *command.*

To exit GED and return to the system prompt, type:

```
exit [Return]
```

This command does not save your current drawing. If you issue this command when your drawing has been modified but not written, you get a warning message. To save the drawing, use the **write** command, then the **exit** command. To exit without saving the modifications, issue the **exit** command twice.

Exiting the Sun Window System

When you exit GED, you are automatically returned to the window from where you entered GED. To exit the *suntools* window:

1 Move the cursor outside the GED window and press and hold the right mouse button. The following menu is displayed:

```
ShellTool
CommandTool
MailTool
TextEditor
DefaultsEditor
IconEditor
DbxTool
PrefMeter
GraphicsTool
Console
Lock Screen
Redisplay All
Exit Suntools
```

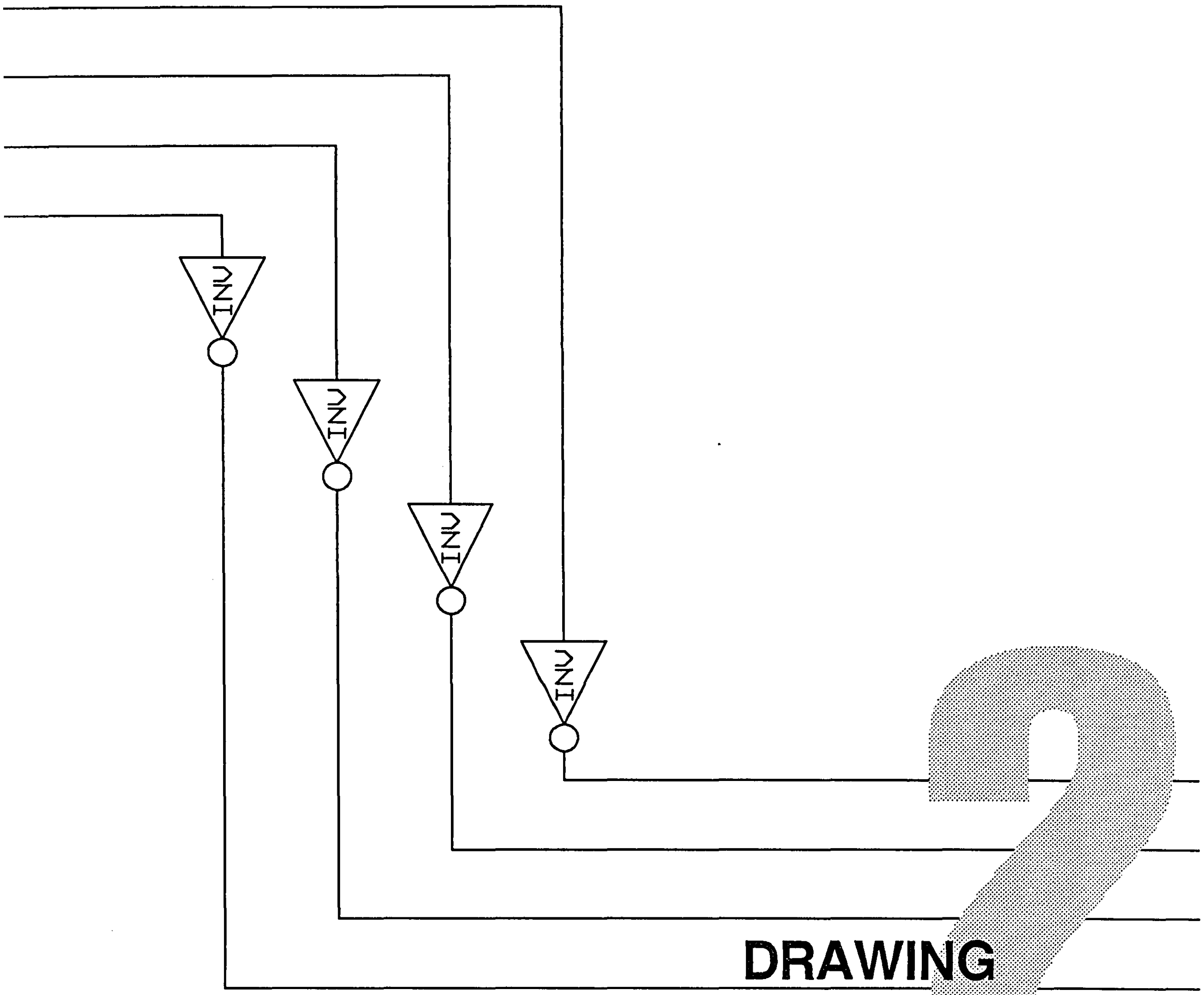
A menu box is "selected" when it is highlighted in reverse video.

2 While still holding the right mouse button, slide the cursor down to the **Exit Suntools** menu box. When the box is selected, release the right mouse button. When you release the button, the message appears:

```
Press the left mouse button to confirm
Exit. To cancel, press the right mouse
button now.
```

3 Press the left mouse button. The UNIX prompt appears in the full screen. You can now log off the system by typing **exit** and pressing Return.

.ESSON 2



DRAWING

2

This lesson introduces the most basic GED commands by showing you how to draw a simple circuit and how to correct your mistakes as you go.

New Commands

add	**connect**	**copy**
delete	**hardcopy**	**move**
select	**show**	**signame**
undo	**wire**	**write**
zoom; (Refresh)		

The circuit used for this lesson is a **full adder circuit**, as shown in Figure 2–1.

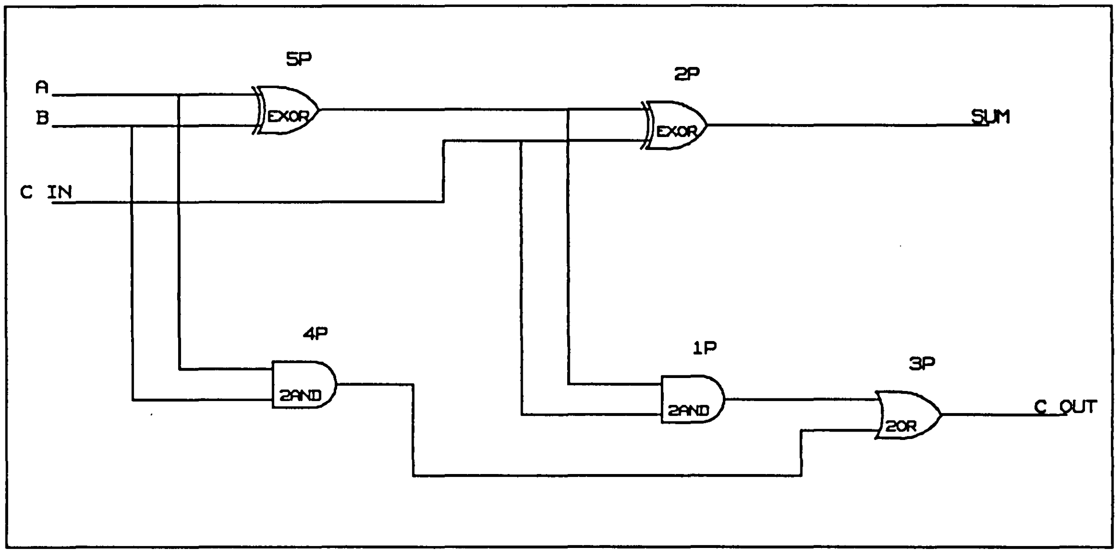


Figure 2–1. Full–Adder Circuit

If you exited GED in Lesson 1, type **ged** *to redisplay the GED screen.*

To begin creating the full–adder circuit, you first need to add some library parts from an installed library onto the screen. Library parts are also called *bodies*. To bring a library part, or body, onto the screen, you use the **add** command.

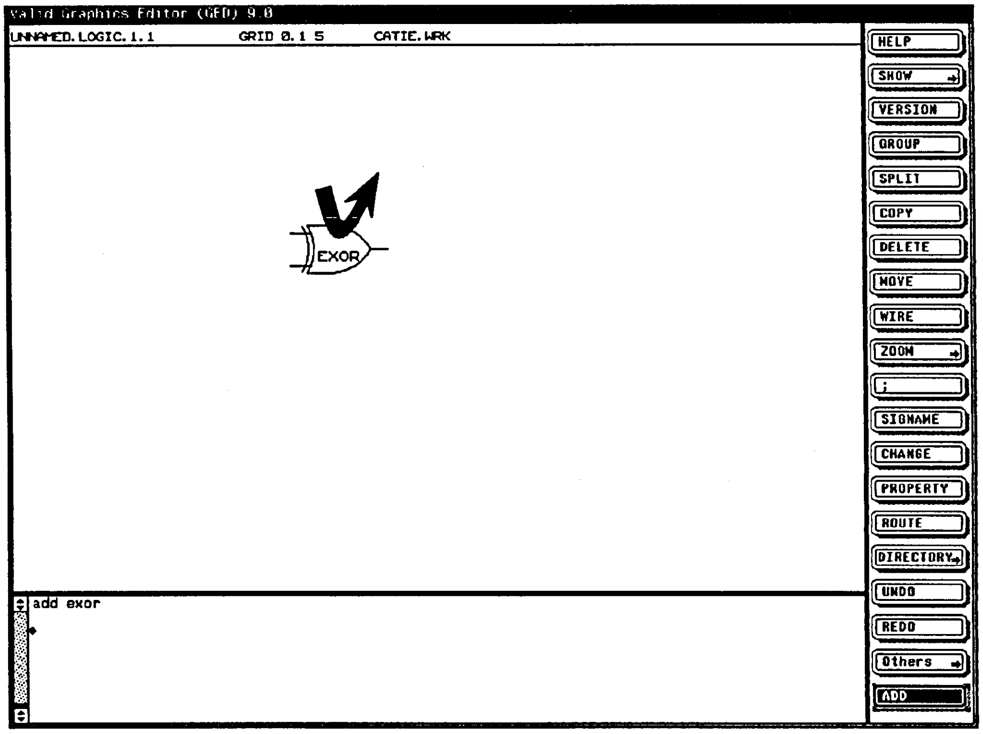
The Add Command

To add an EXOR part to the drawing, type:

```
add exor  Return
```

Move the mouse and see how the EXOR body follows the cursor around the screen.

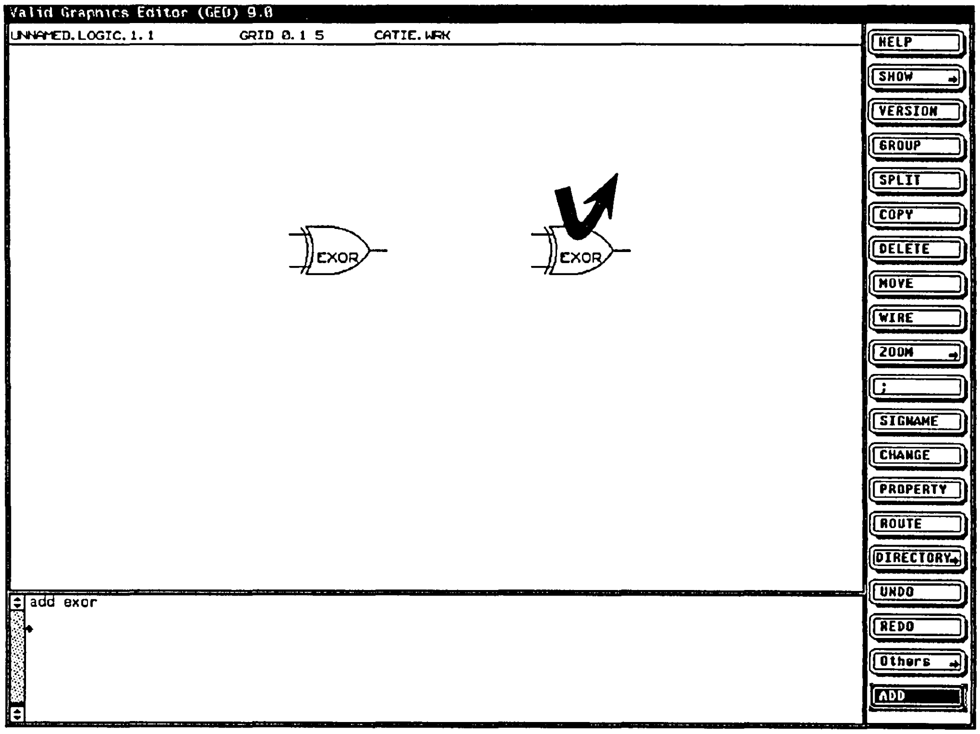
To release the EXOR and place it on the screen, click the left button on the mouse.



```
Valid Graphics Editor (GED) 9.0
UNNAMED.LOGIC.1.1          GRID 0.1 5      CATIE.WRK
```

```
add exor
```

Move the mouse away and click the left button again; another EXOR is now following the cursor. To place this EXOR on the screen, click the left button again.

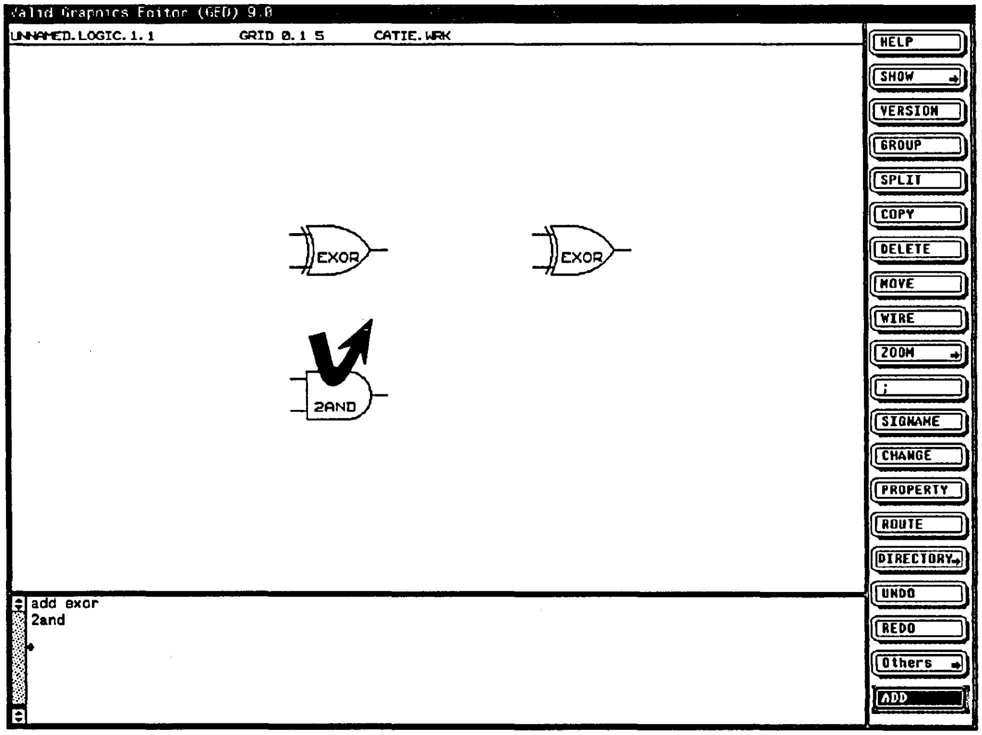


add exor

To stop adding EXORs and add another part, type the name of another part. This time type:

`2and` Return

Now place the 2AND on the screen.



```
add exor
2and
```

The Delete Command

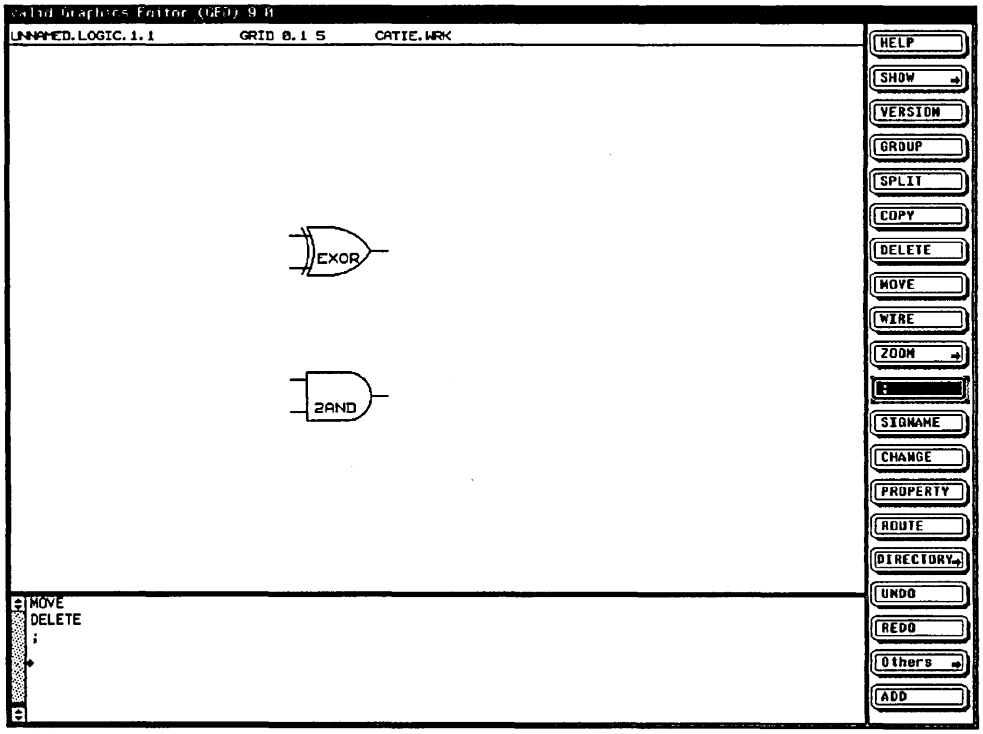
To delete a body you don't want, just choose the **delete** command from the GED menu, move the cursor to the center of the body, and click the left button on the mouse. You can delete bodies, wires, or any other object on the screen.

The Move Command

To move a body after you have added it to the drawing:

1. Choose the **move** command from the GED menu.
2. Move the cursor to the center of the body you want to move, click the left button, and move the mouse. You see the selected body move with the cursor.
3. Move the body to a new location and click the left button to release the body.

Now use the **delete** and **move** commands to make your screen look like Figure 2–2.



```
MOVE
DELETE
;
```

Figure 2–2. Half-Adder

Tips on Drawing

✓ *Adding*

If when you try to add a part you get the message "`Could not find device of name:` *2AND*," the problem may be:

- You mistyped the name of the part.
- The library to which that part belongs has not yet been made accessible to GED.
- The library has not been installed on your system.

To use this tutorial, you need the library named "tutorial." If you did not do so already, type:

```
library tutorial [Return]
```

This command makes the tutorial library available for use by GED. If you still cannot add an EXOR, make sure that the tutorial library is installed on your system.

✓ *Deleting and Moving*

To delete or move a body, move the cursor to the center of the body and click the left button on the mouse. With the **move** and **delete** commands, the left button selects the origin of the body nearest the cursor (the origin of a body is usually at its center). If you are too far from the desired origin, the command may affect an object other than the one you want.

The Undo Command

The **undo** command is very simple. When you make a mistake, it undoes the last command you gave. If you select **undo** again, it undoes the next previous command. You can back up one step or many with the **undo** command. You cannot back up beyond:

If you back up too far, there is also a **redo** *command. See the* ValidGED Command Reference Manual *for more details.*

- The last **write** command
- The last **edit** command
- The beginning of the editing session

Now you can learn how to wire the bodies together.

The Wire Command

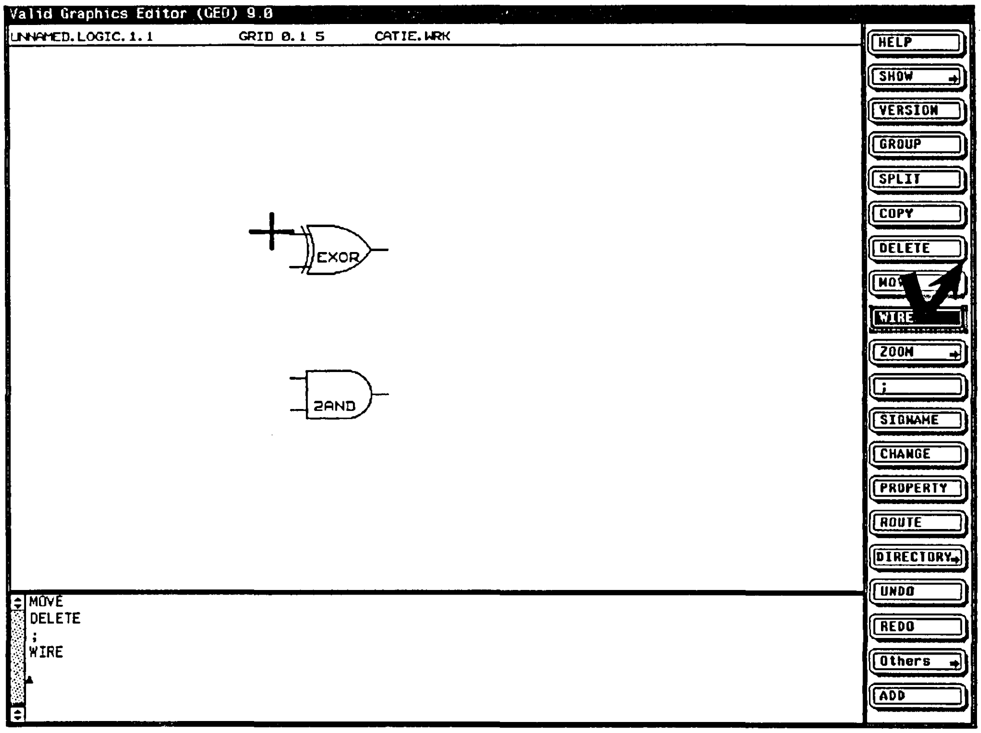
The **wire** command uses the different buttons on the mouse. Up to now, you have used only the left button on the mouse. With the **wire** command, you use the right and left buttons. Here are the rules:

- The end of the wire is at the center of the crosshair.
- Use the right-hand button once to start or end a wire on a pin of a body.
- Use the left button once to start or end a wire on another wire or on the open screen.
- Use the left button to put an additional 90 degree bend in a wire (one click).

Follow these steps to wire the bodies of Figure 2–2 together:

1 Select the **wire** command from the menu.

2 Move the mouse so that the cursor is on the upper input pin of the EXOR.

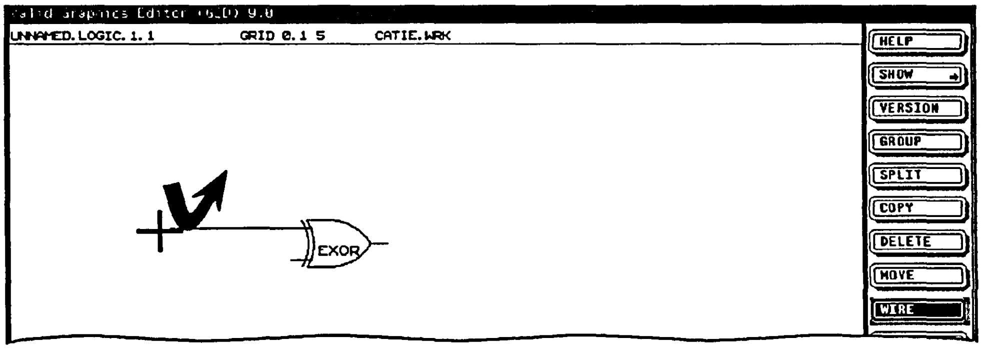


MOVE
DELETE
;
WIRE

3 Click the right-hand button on the mouse.

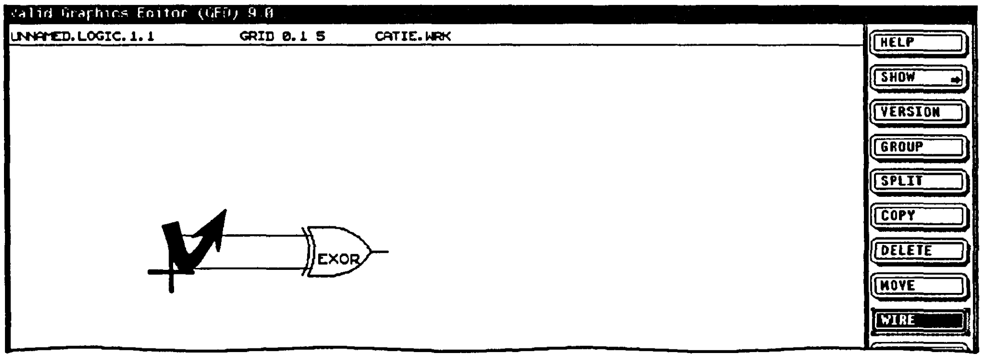
4 Move the mouse to the left and draw the wire.

5 To end the wire, click the left button twice.



6 Move the mouse to the lower pin of the EXOR.

7 Repeat steps 3 through 5.

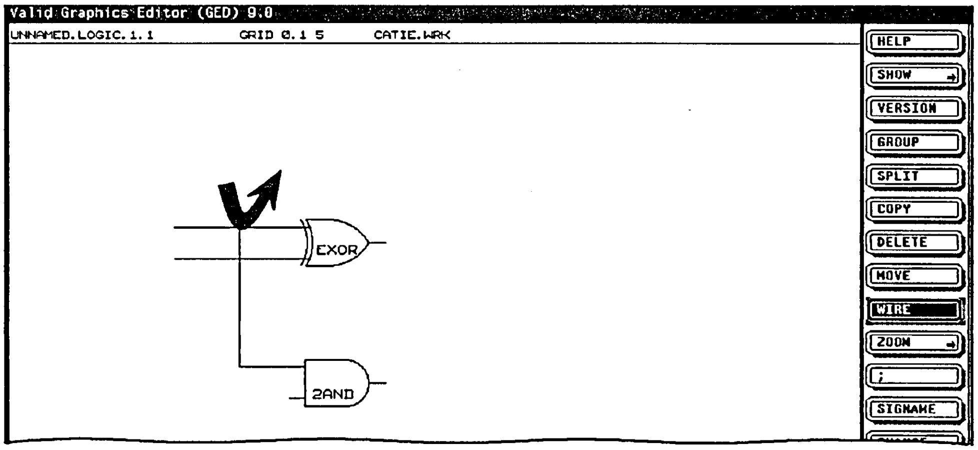


8 Move the mouse to the upper input pin of the 2AND and click the right-hand button.

9 Move the mouse to the left about an inch and up toward the EXOR. The wire now has a bend in it.

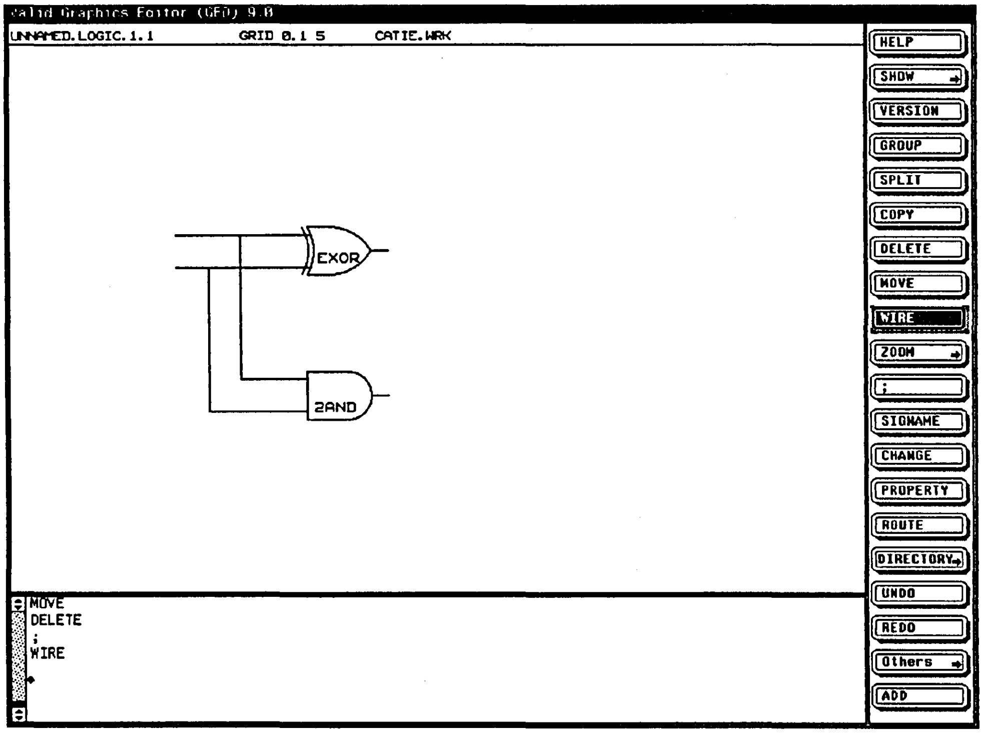
10 Continue the wire upward until it intersects the wire from the upper input pin of the EXOR.

11 To attach the wire at the intersection, click the left button once.



12 Repeat steps 8 – 11 to wire the lower input of the 2AND to the lower input of the EXOR.

Now you should have a half-adder on your screen, as shown in Figure 2-3.



```
UNNAMED.LOGIC.1.1          GRID 0.1 5        CATIE.WRK
```

```
MOVE
DELETE
;
WIRE
```

Figure 2-3. Half-Adder

Tips on Wiring

Wiring problems are very common. When you first use the **wire** command, your drawing may not look as neat as you would like. Here are some tips:

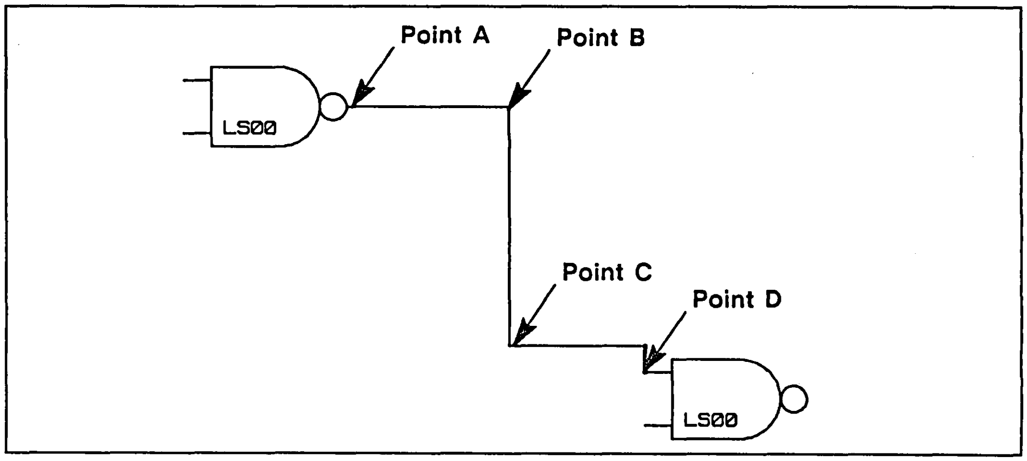
✓ *Ending a wire*

When you want to end a wire in free space, be sure to click the left button twice before you move the mouse again. If you click only once and move the mouse, you get a bend in the wire and it continues to follow the mouse.

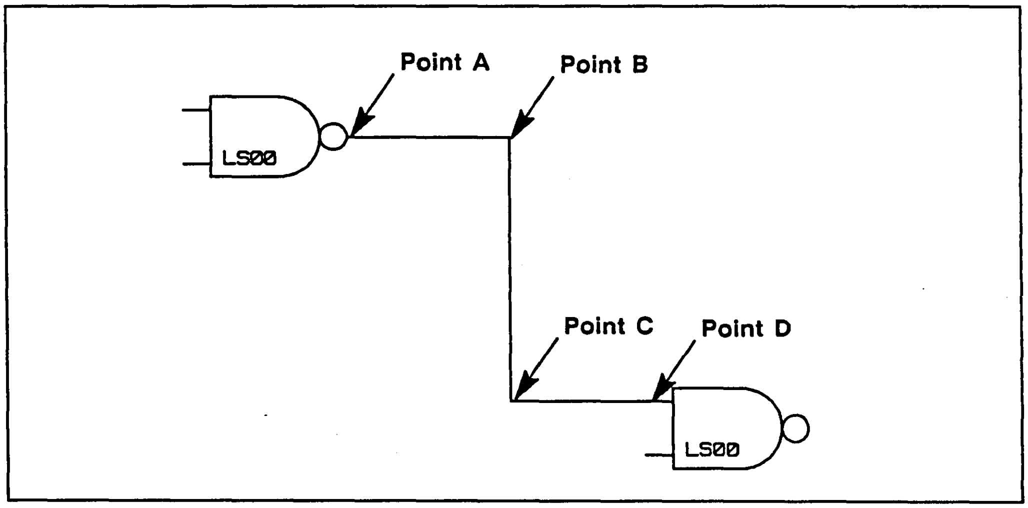
✓ *Aligning wires and pins*

When you attach wires, sometimes it is difficult to align the last wire with the attachment point. If you "fix" the second-to-last bend of a wire, rather than the last bend, you retain the flexibility of movement you need to align the last wire and the attachment point.

For example, connect two parts, wiring from the left part to the right part. If you start the wire at Point A and click the left button at Point C (the last bend you want to place in the wire), you must estimate the placement of Point C so that the wire aligns with the pin at Point D. If Point C is not even with Point D, you may end up with an unwanted jog in the wire (shown as a thick line in the figure below).



To avoid the alignment problem, click the left button at Point B (the second-to-last bend in the wire) instead of Point C. Then use the middle button to change the direction of the bend in the wire. Now Point B is fixed on the screen and the wire flexibility extends from Point C. You can simply extend the wire from Point C to Point D and press the right button to connect the two points.



✓ *Realigning parts and wires*

Use the **move** command to change the shape of wires and the alignment of bodies. When you move a body that has wires attached to it, the wires move too. To move a wire and leave the attached bodies in place:

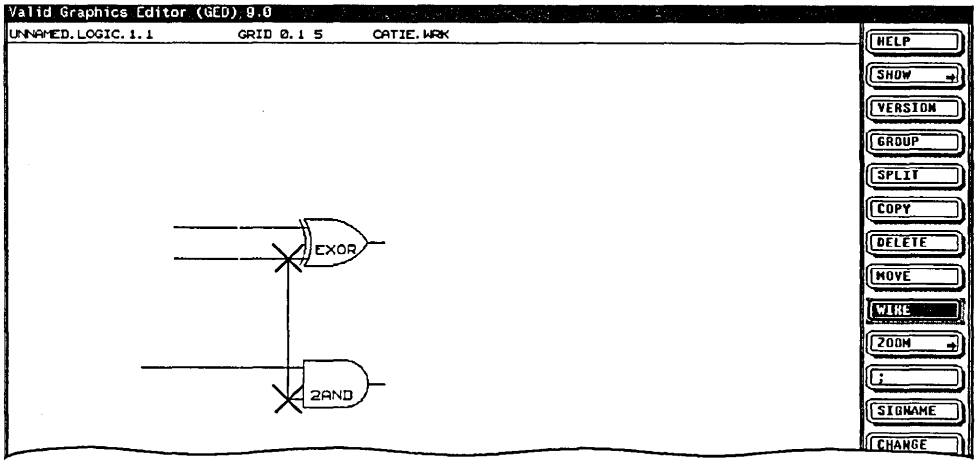
1. Select the **move** command from the menu.

2. Point to the wire, click the left button on the mouse, and then move the mouse.

3. When the wire is the shape you want, click the left button to release it from the mouse.

Wiring pins together

A common wiring problem is illustrated in the following figure.



Here, the vertical wire from the 2AND passes so close to the body that it attaches to both input pins instead of just one. To find out where the vertical wire is attached, type:

```
show connect [Return]
```

GED then indicates (with an asterisk) where it recognizes pin connections.

To correct the error, delete the vertical wire segments and rewire. Type

```
zoom ; [Return]
```

to refresh the screen. Then, use **show connect** again to make sure that there are pin connections only where you want them.

✓ *Removing extra wire segments*

Use the **delete** command to remove unwanted wire segments.

The Refresh Command

The command you used in the previous paragraph is called the "refresh" command. This command redraws the screen without changing your drawing.

To use this command, type:

```
zoom ; [Return]
```

The refresh command is used any time you want to redraw the screen.

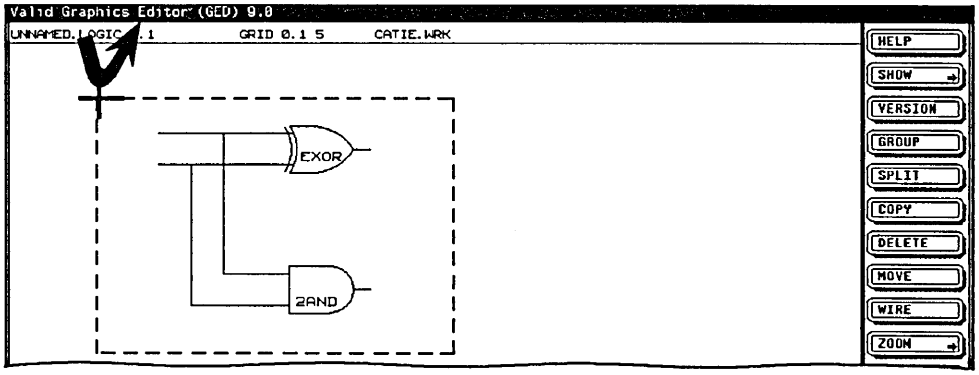
The Select Command

The fastest way to make your half adder into a full adder is to use the **select** command to define the half adder as a group, and then to use the **copy** command on the entire group. Follow these steps:

1 Type:

```
select [Return]
```

2 Click the left button at any corner of the half adder. A stretchable rectangle is attached to the cursor.



3 Move the cursor to position the rectangle around the circuit and click the left button to complete the selection. You have now defined the half adder as a group. The rectangle disappears from the screen, the group is highlighted, and the following message appears:

You can define up to 26 different groups on a single page of a drawing.

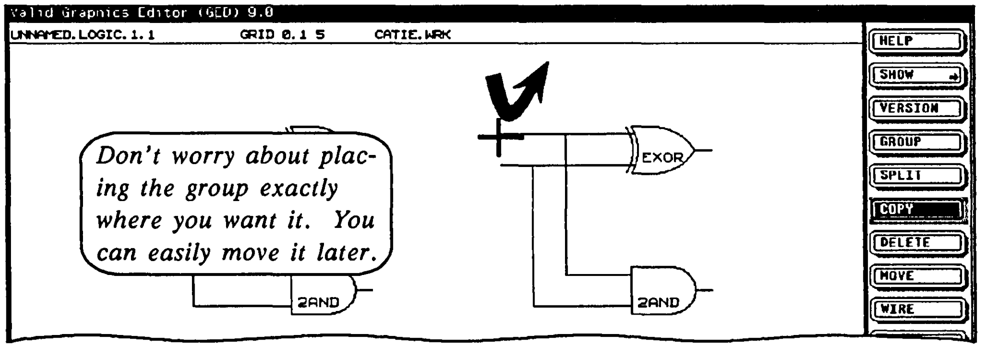
```
Using group A
Group contains:
   2 bodies
   8 wires
```

4 Next, select the **copy** command from the menu.

If you click the left button, you select only the object (body, note, or wire) that is closest to the cursor.

5 Move the cursor within the defined group and click the center button to select the entire group.

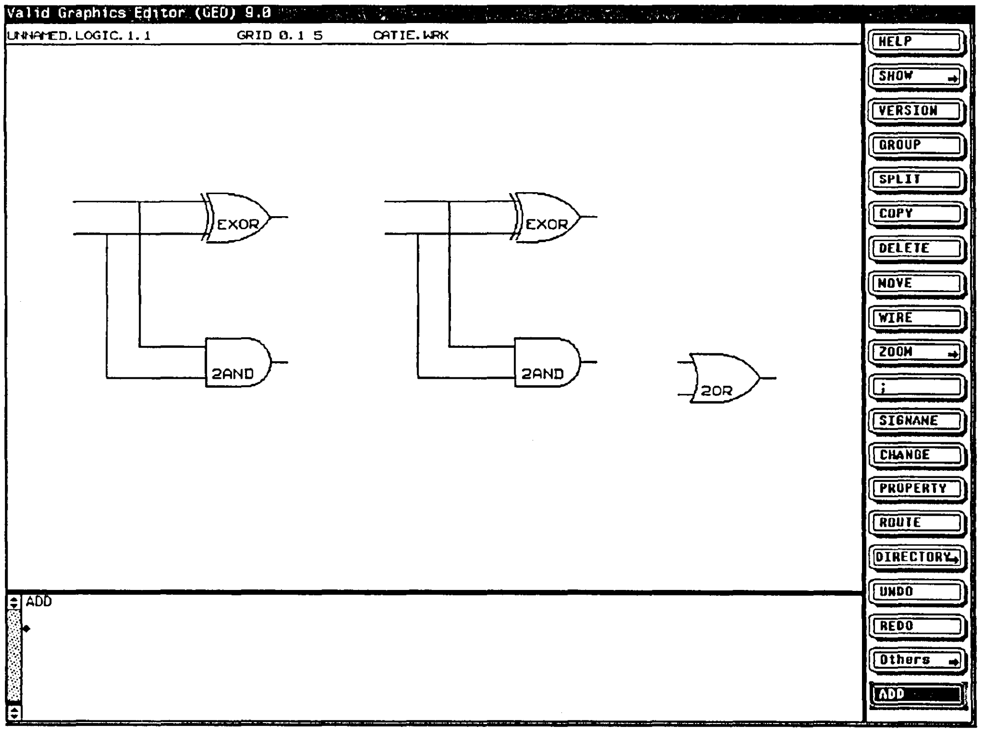
6 Now move the cursor to the right and you see a copy of the entire group following the cursor. To place the group on the screen, click the left button.



Don't worry about placing the group exactly where you want it. You can easily move it later.

To complete the full adder shown in Figure 2–1 at the beginning of this lesson, you just need to add six more wires and a body. Follow these steps:

1 Add a 2OR to the screen at the bottom right.



This is pin–to–pin wiring. You start the wire with one press of the right–hand button, and end the wire with one press of the right–hand button.

2 Wire the output from each 2AND to an input of the 2OR:

1) Select the **wire** command.

2) Move the cursor to the lower input pin of the 2OR.

3) Press the right–hand button once.

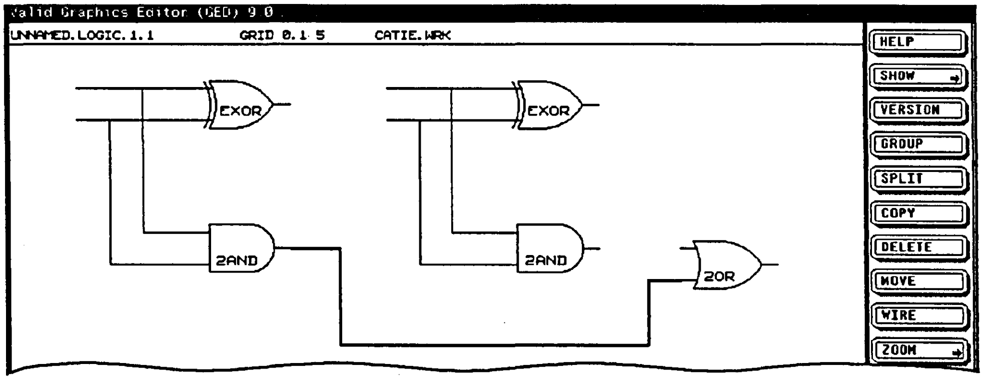
4) Move the cursor left and down, and watch the wire follow.

5) Click the left button once to bend the wire again.

6) Move the cursor to the left and upward, until it is level with the output pin of the second 2AND.

7) Click the left button once to bend the wire again.

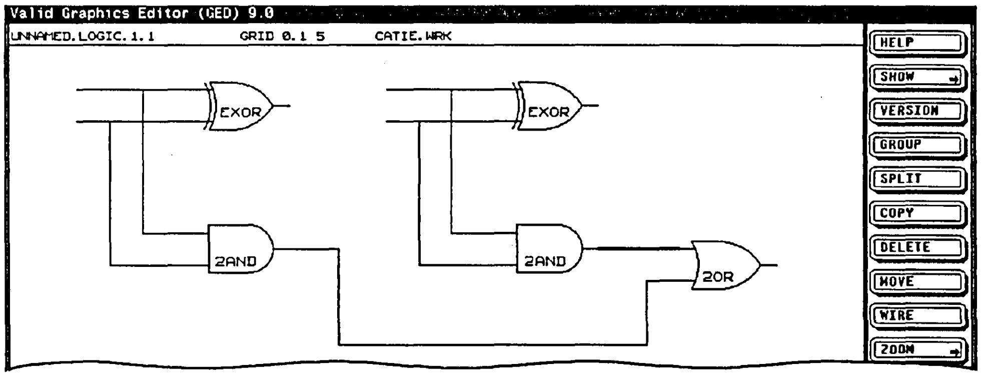
8) Press the right–hand button once to end the wire.



Valid Graphics Editor (GED) 9.0

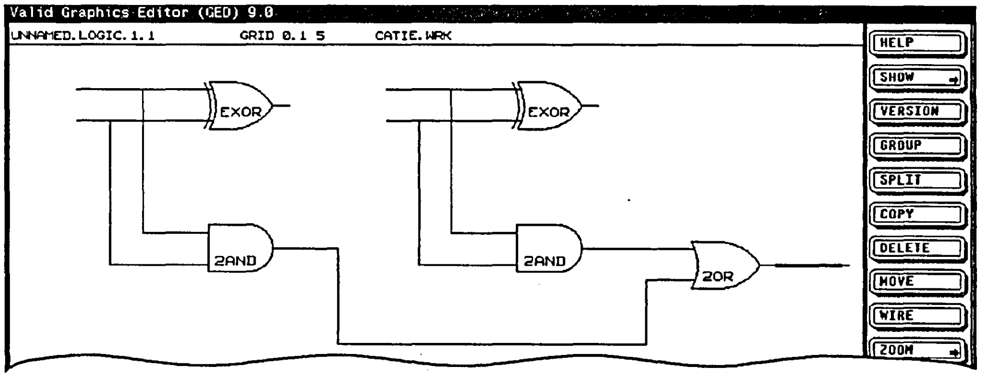
UNNAMED.LOGIC.1.1 GRID 0.1 5 CATIE.WRK

3 Now wire the upper input of the 2OR to the output pin of the other 2AND.


Valid Graphics Editor (GED) 9.0

UNNAMED.LOGIC.1.1 GRID 0.1 5 CATIE.WRK

4 Continue the output wire of the 2OR to the right. Use the right-hand button to start the wire and two clicks of the left button to end the wire.


Valid Graphics Editor (GED) 9.0

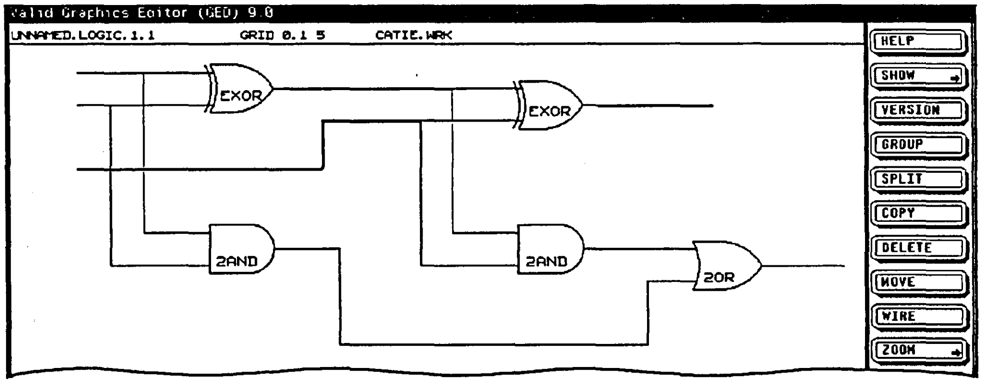
UNNAMED.LOGIC.1.1 GRID 0.1 5 CATIE.WRK

5 Next wire the right-hand EXOR:

1) Continue the output pin of the EXOR to the right. Use the right-hand button to start the wire and two clicks of the left button to end the wire.

2) Now wire the upper input pin to the output pin of the other EXOR (pin-to-pin wiring). Use the right-hand button to start and end the wire.

3) Continue the lower input pin out to the left. Use the right-hand button to start the wire. Click the left button once to put a second bend in the wire. Click the left button twice to end the wire.



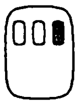
6 Finally, use the **move** command if you want to realign the bodies.

Now the wiring of your full adder is complete.

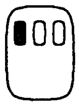
Wiring Concepts

Now that you've done some basic wiring, you probably want to know more about the mouse buttons and why you use different ones at different times.

Remember:



✓ The right-hand button is used to wire pin-to-pin



✓ The left button is used to start and stop wires on the open screen and at other wires

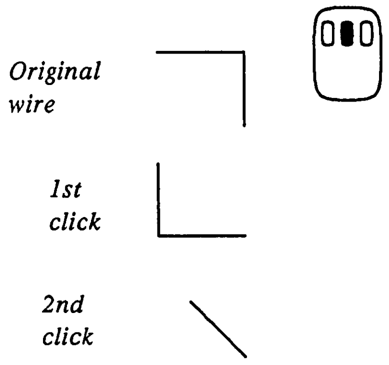
This is because the right-hand button causes the wire to attach to the nearest *vertex* of a pin or wire, and the left button causes the wire to attach to the nearest *grid point.*

A wiring *vertex*, by definition, is found at each of these locations:

- A pin on a body
- The end of a wire
- A bend in a wire

If you want to start a wire on the open screen and you press the right-hand button by mistake, you'll notice that a long wire appears. By pressing the right-hand button, you have selected the nearest vertex, even if that vertex is halfway across the screen.

It is also important to get in the habit of using the right-hand button to wire pin-to-pin. If you move the cursor right to the pin and press the left button, the wire will probably start where you want it to. But since the left button selects the nearest grid point, if you are not exactly on the pin, the wire may start from a grid point that is not a vertex. To make sure you wire correctly, get in the habit of pressing the right-hand button when you want to start or end a wire on a pin. By using the right hand button, you select the nearest vertex, and you are sure your wire starts on the pin you want it to.



The middle button is used during the **wire** command to change the shape of the wire. Draw a right angle wire on the screen and, before you set the end of the wire down, press the middle button several times. The wire cycles through three different shapes (two angled and one diagonal). After you use the middle button to change the shape of the wire, the selected shape remains in use until you select another shape.

Figure 2–4 is a wiring reference chart that shows the most common wire shapes and the buttons used to draw them.

The following conventions are used in the chart:

- Button click points are shown as a filled dot.
- Press all buttons once unless indicated otherwise (2).
- All wiring shown is left-to-right.

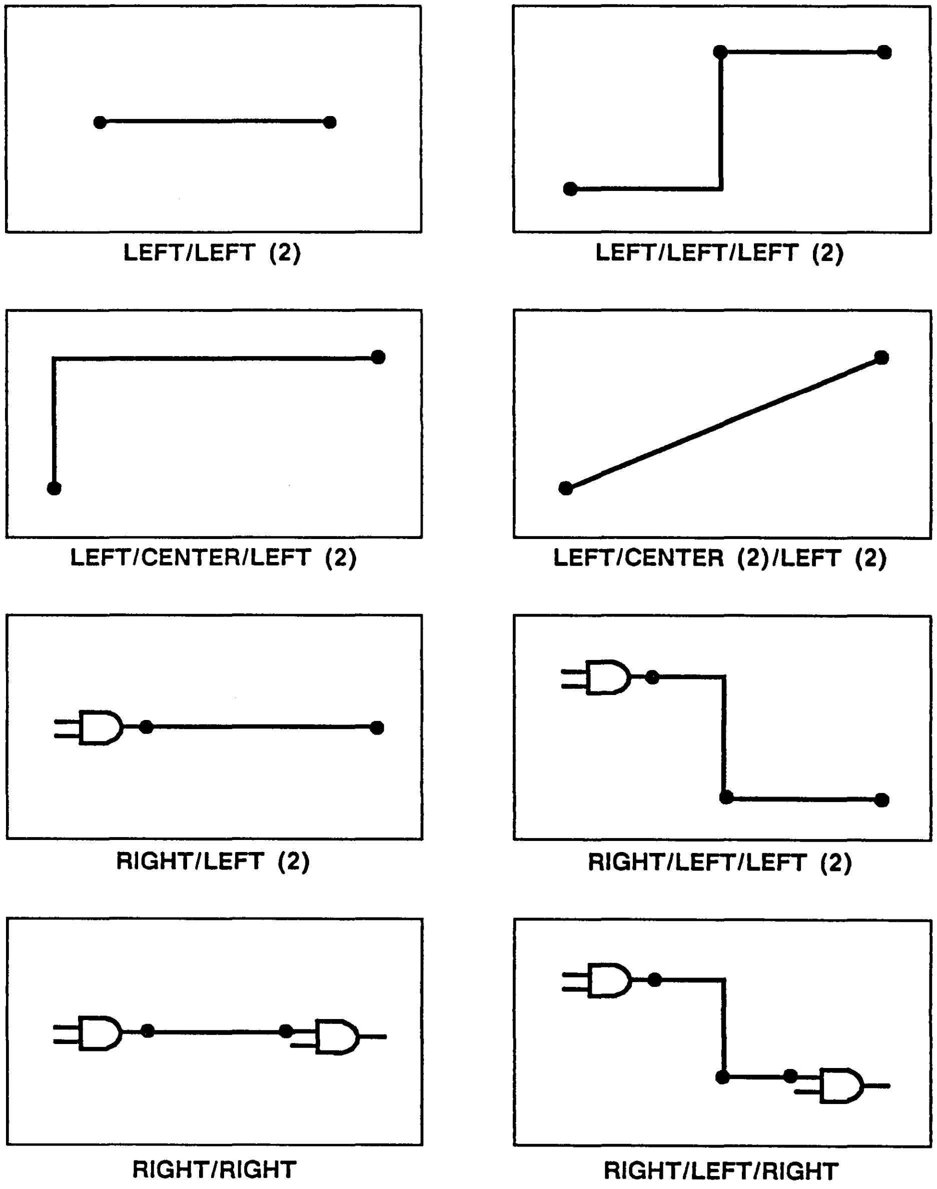


Figure 2–4. Wiring Reference Chart

Finishing a Schematic

To make your full adder circuit into a finished schematic, you must:

- Put signal names on the input and output signals
- Save the drawing
- Print a hardcopy

This completes your first basic drawing.

The Signal Name Command

To name the signals on a schematic, you use the command **signame**. With this command, you not only add a signal name onto the drawing so you can refer to it, but you also associate this name with a particular signal (wire). After the name is associated with the correct signal, the name that appears on the drawing (the text string) remains associated with the correct signal even if you move the signal name. Because of this association, the **signame** command involves two steps:

1. Type in the name of a signal.
2. Point to the wire where you want the name attached.

If you have many signals to name, it is awkward to type in a name and then point to a signal, and then type in another name and point to another signal. So the **signame** command lets you type in all the names and then point to each signal in turn. To name the signals on the adder circuit, follow these steps:

1 Select **signame** from the GED menu.

2 Type:

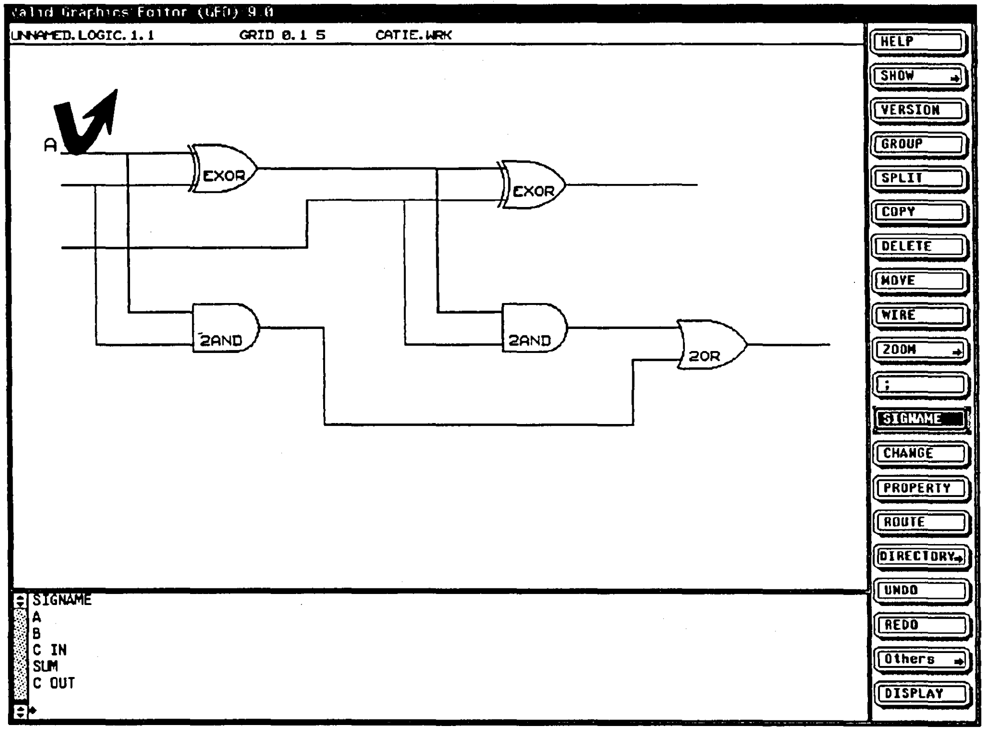
    A [Return]

   The 'A' is attached to the cursor.

3 Type the following signal names. Be sure to press [Return] after each name:

```
B
C IN
SUM
C OUT
```

4 Now move the mouse so that the cursor points to the wire for signal A and click the left button once. You'll see the name A appear near the wire.



```
Valid Graphics Editor (GED) 9.0
UNNAMED.LOGIC.1.1          GRID 0.1 5        CATIE.WRK
```

```
SIGNAME
A
B
C IN
SUM
C OUT
```

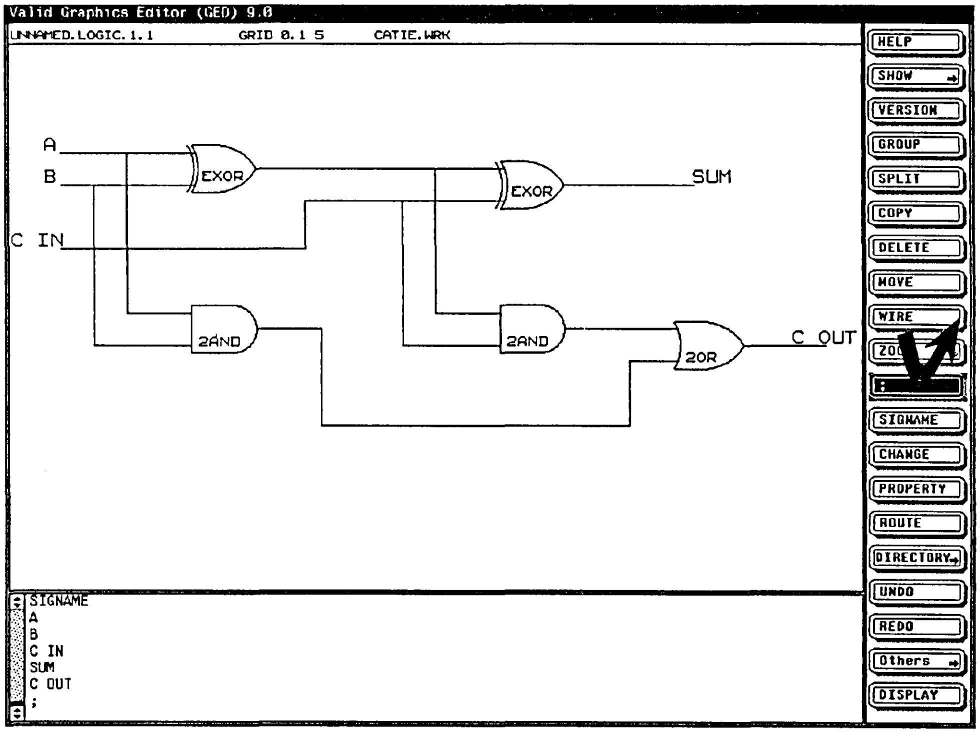
5 Next move the cursor to signal B and click the left button.

6 Move to C IN and click.

7 Move to SUM and click.

8 Move to C OUT and click.

9 Select ; (semicolon) from the menu to end the **signame** command.



```
SIGNAME
A
B
C IN
SUM
C OUT
;
```

All of the primary inputs and outputs of the adder circuit now have signal names. The drawing is now complete.

The Write Command

Now that your drawing is complete, you should save it. To do so, use the **write** command. Type:

```
write adderckt [Return]
```

This command tells GED to save the current drawing in the current working directory under the name "adderckt."

After you give this command, two types of information appear on the screen: path properties (or P–numbers) and status messages.

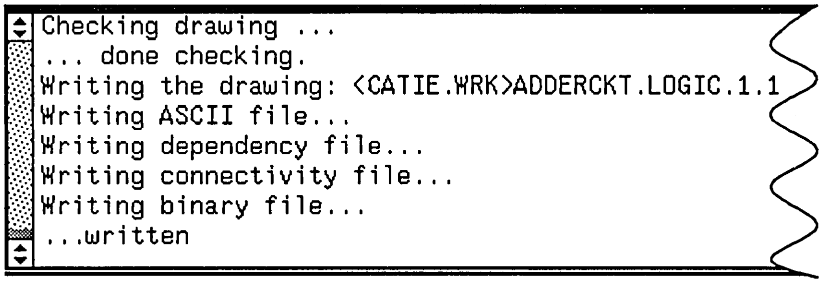
Path Properties

When you **write** a drawing, GED assigns each library part on the drawing a reference number. This number is always an integer followed by the letter "P." These P–numbers designate the locations of each of your logical parts. They are logical location designators. The P–numbers are used by the Compiler, the Packager, and the other verification programs. The P–number is also called a *path property,* because it uniquely identifies the path to a particular library part on your drawing.

The first time you save a design, GED assigns P–numbers to all of the library parts. These P–numbers become part of your drawing. The next time you save the design, GED keeps any previous P–number assignments and assigns new P–numbers only to newly–added library parts.

Status Messages

When you give a **write** command in GED, status messages in the message window tell you that GED is writing your drawing. They look like this:


```
Checking drawing ...
... done checking.
Writing the drawing: <CATIE.WRK>ADDERCKT.LOGIC.1.1
Writing ASCII file...
Writing dependency file...
Writing connectivity file...
Writing binary file...
...written
```

The last line of this message tells you that your drawing has been saved.

Errors are discussed in Lesson 6.

The **write** command automatically checks the drawing for certain types of errors. If any errors are reported in the status messages, disregard them for now.

Tips on Saving a Drawing

✔ If you type:

write Return

without typing a drawing name, GED saves the drawing on the screen under the drawing name currently showing in the left field of the status line.

See Lesson 3 for more on saving and retrieving drawings.

If you have not given your drawing a name, `unnamed.logic` is the default drawing name used. If you try to write an unnamed drawing, GED reminds you that the drawing is unnamed and asks you to reenter the command with a drawing name or to confirm that you really want a drawing named `unnamed.logic`.

The Hardcopy Command

Now that your drawing has been completed and saved, you probably would like to get a hard copy of your finished schematic. To do so, use the **hardcopy** command. Type:

```
hardcopy [Return]
```

If your workstation is connected to a plotter, a copy of your drawing is printed on the plotter.

Since **hardcopy** prints the drawing currently displayed on the screen, always save your drawing with the **write** command before you use the **hardcopy** command. By writing your drawing immediately before making a hardcopy, the drawing printed will accurately reflect your saved drawing.

If you're ready to take a break, and you have used the **write** command to save your most recent drawing, type

```
exit [Return]
```

to exit GED and return to the system prompt.

If you have made changes to your drawing since your last **write** command and you try to exit, you see a warning message. Either use the **write** command or, if you *do not* want to save the most recent changes, type

```
    exit Return
```

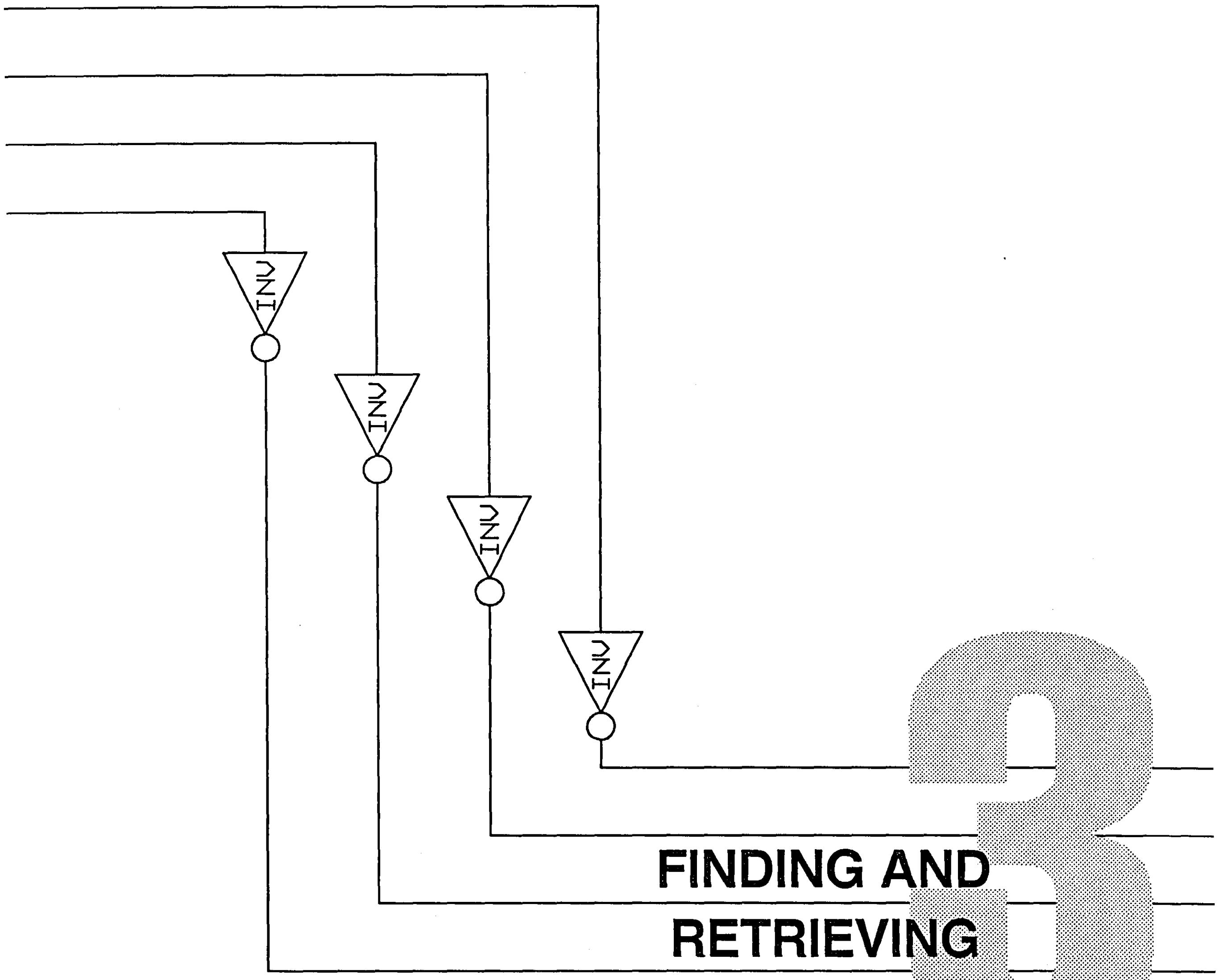
again to exit GED. You may also use the **quit** command to exit from GED. In GED, **exit** and **quit** are identical commands.

Now you're ready to go on to Lesson 3, *Finding and Retrieving Drawings.*

Note

In the remaining lessons, the tutorial does not always explicitly instruct you to press Return after typing a command. You still need to press Return after GED and UNIX commands, but the tutorial assumes that you no longer need to be reminded of this.

LESSON 3



FINDING AND RETRIEVING DRAWINGS

The **write** command was introduced at the end of the last lesson. In this lesson you learn how to find and retrieve drawings.

New Commands

directory	**edit**

Drawing names are also discussed.

In the previous lesson, when you used the command

```
write adderckt
```

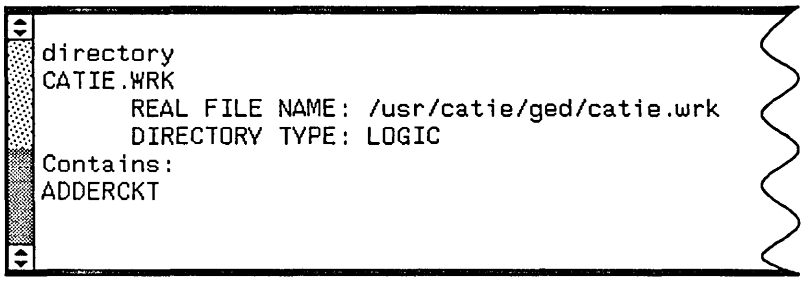
GED saved the drawing you were working on under the name `adderckt` in your current working directory. If you used the **write** command with no name (no argument), GED asked you if you really wanted to save your drawing under the name "unnamed."

The Directory Command

To see what drawings you have saved, use the **directory** command. While running GED, type:

```
directory
```

The message window looks something like this:



```
directory
CATIE.WRK
     REAL FILE NAME: /usr/catie/ged/catie.wrk
     DIRECTORY TYPE: LOGIC
Contains:
ADDERCKT
```

You learn more about the **directory** *command in Lesson 12.*

The first three lines tell you that your current directory is *catie*.wrk (*catie* is assumed to be your login name). The remaining lines list your drawings by name.

The Edit Command

When you enter GED and you want to call a drawing up on the screen, you use the **edit** command. But first, type

```
library tutorial
```

You learn how to avoid typing **library** *commands when you learn about the* startup.ged *file in Lesson 7.*

so that your drawing will be usable and complete.

After you know the names of your drawings, you can call one back on the screen. To edit the drawing named `adderckt`, type:

```
edit adderckt
```

Now that your drawing has a name and is no longer called “unnamed,” you can issue a **write** command without giving an argument. Just type

```
write
```

The current drawing name is the name that shows in the left-hand field of the status line.

and GED saves your current drawing under the current drawing name.

To finish with `adderckt` and start another drawing called “test,” first save the `adderckt` drawing (with the **write** command) if you have not done so. Then type:

```
edit test
```

You'll see the name on the status line change to

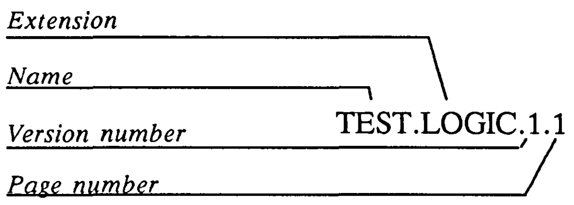
```
TEST.LOGIC.1.1
```

and the screen blanks, except for the GED menu, the status line, and the message “`New drawing started.`”

Drawing Names

When you used the **edit** command before, you asked to edit the drawing `TEST` and GED displayed the drawing `TEST.LOGIC.1.1`.

A *drawing name* is made up of four fields.



To clear the directory information from the screen and return to the drawing you were editing, select any command from the GED command menu.

Drawing extensions are explained in the next section.

- The first field is for the name that you assign to the drawing.
- The second field contains one of several extensions that indicate the kind of drawing.
- The third field shows the version number of the drawing. This field is used primarily in library development. It provides a way of storing different versions of library parts. It is not suitable for storing different drafts or revisions of logic drawings.
- The last field indicates the page number within a drawing. This field allows you to draw a schematic that is several pages long. If you are editing `TEST.LOGIC.1.1` and

you want to continue to a second page, you only need to type:

```
edit ...2
```

The three dots (...) hold the place of the first three fields, one for each field.

This tells GED to edit page two of the current drawing.

Drawing Name Extensions

The extensions are used by the design tools to distinguish regular schematics from documentation drawings and component models (in our libraries), and for hierarchical design. This tutorial covers only drawings with the default extension .LOGIC, which is the kind of drawing used for building logic designs.

The extension .BODY is also used in hierarchical design.

Another extension is .BODY. Each body that you bring up on the screen with the **add** command is stored in the system as a drawing with the extension .BODY, for example, EXOR.BODY or LS08.BODY.

Entering a Drawing Name on the Command Line

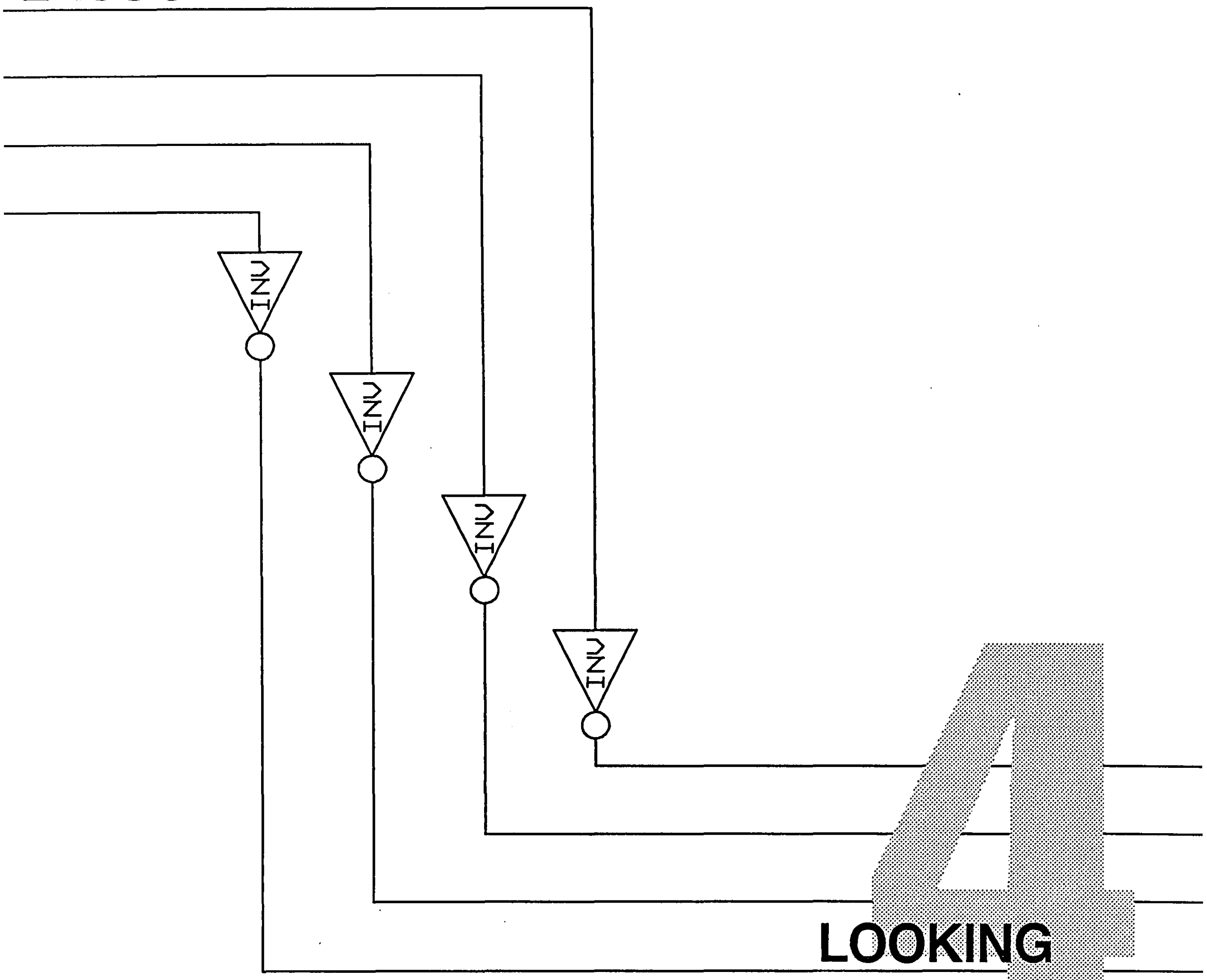
To save steps, you can enter the name of the drawing you want to edit when you enter GED with the *ged* command. If you are in a Suntools window and want to edit the full adder drawing, type:

```
ged adderckt
```

You can also use a new drawing name if you want to start a new drawing without using the **edit** command. If your drawing name contains spaces and/or special characters (e.g., !, >, &, etc.), you must put quotes around the name. For example:

```
ged "mem unit"
```

LESSON 4



LOOKING

The **zoom** command lets you:

- Redraw the screen
- Fit the complete drawing on the screen
- Center the screen around a specified point
- Move a drawing around on the screen
- Enlarge a drawing
- Reduce a drawing

You have already learned how to use the simplest form of the **zoom** command:

```
zoom ;
```

When you pressed Return, this command refreshed the screen and redrew your drawing in its current size and position.

Another useful form of the **zoom** command is:

```
zoom fit
```

When you edit an existing drawing, it is automatically fit to the full screen.

This command fits the entire drawing to the full screen.

Centering

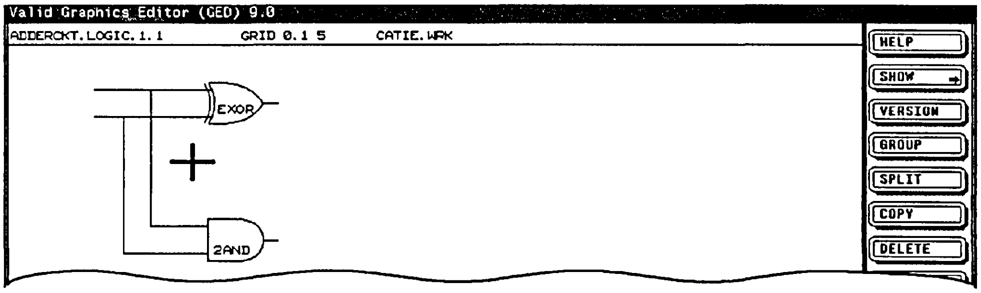
The **zoom** command also lets you pick a point on the screen as the new center of the drawing. This lets you view portions of your drawing that are currently off-screen. When centering a drawing, the size or scale of the drawing is not changed.

To center a drawing:

1 Type:

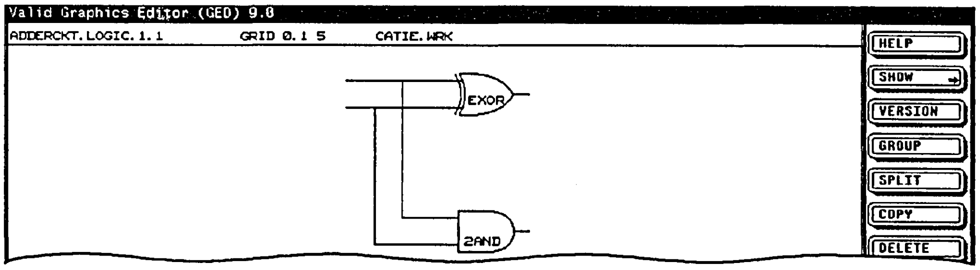
```
zoom
```

2 Move the cursor to the point on your drawing that you want centered on the screen.



3 Click the right mouse button.

The screen is redrawn with the point on the drawing at the center of the screen.



Moving a Drawing on the Screen

The **zoom** command has several options that allow you to move a drawing around on the screen. These options are shown in Table 4–1.

Table 4–1. Screen Placement Commands

COMMAND	DESCRIPTION
zoom in	Enlarge the size of the drawing on the screen
zoom out	Reduce the size of the drawing on the screen
zoom up	Reposition the center of the screen up above the drawing (move the drawing down on the screen)
zoom down	Reposition the center of the screen down below the drawing (move the drawing down on the screen)
zoom right	Reposition the center of the screen to the right of the drawing (move the drawing left on the screen)
zoom left	Reposition the center of the screen to the left of the drawing (move the drawing right on the screen)

Enlarging an Area to Fill the Screen

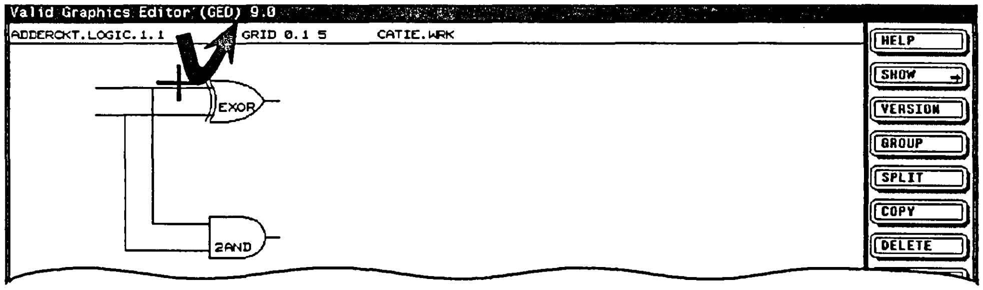
The **zoom** command lets you zoom in on a portion of your drawing to see more detail. This form of the command is very useful when you have a large design, particularly when you need to wire and check connections.

To enlarge a portion of a drawing:

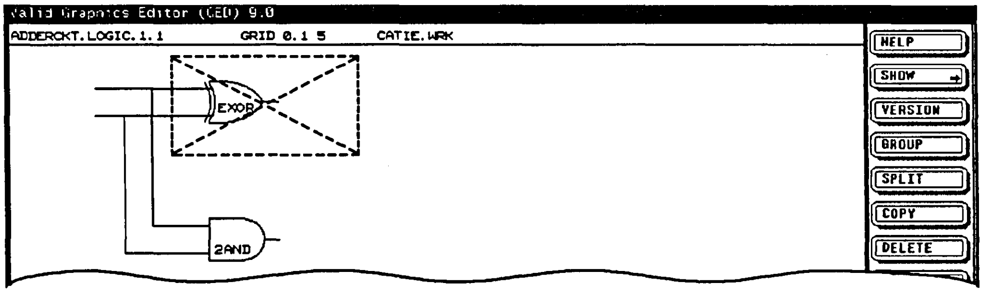
1 Type:

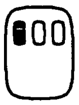
```
zoom
```

2 Move the cursor to one corner of the area you want to enlarge and click the left button.

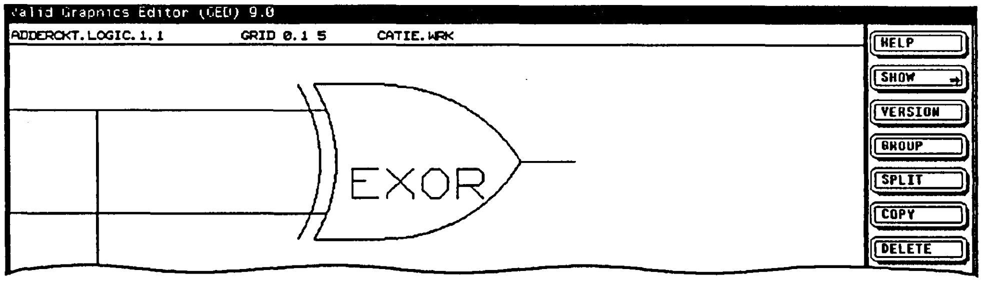


3 Move the cursor diagonally to the opposite corner of the area you want to enlarge. A stretchable rectangle defines the area to be enlarged.





4 Click the left button again. When the button is pressed, the area within the rectangle is enlarged to fill the screen.



Reducing a Drawing

The **zoom** command also lets you reduce the size of a drawing (zoom out). When reducing a drawing, you define a rectangle that determines both the size and position of the drawing on the screen.

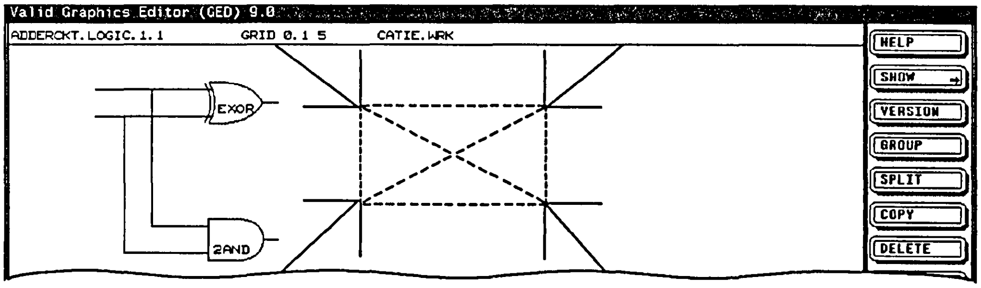
To reduce a drawing:

1 Type:

```
zoom
```

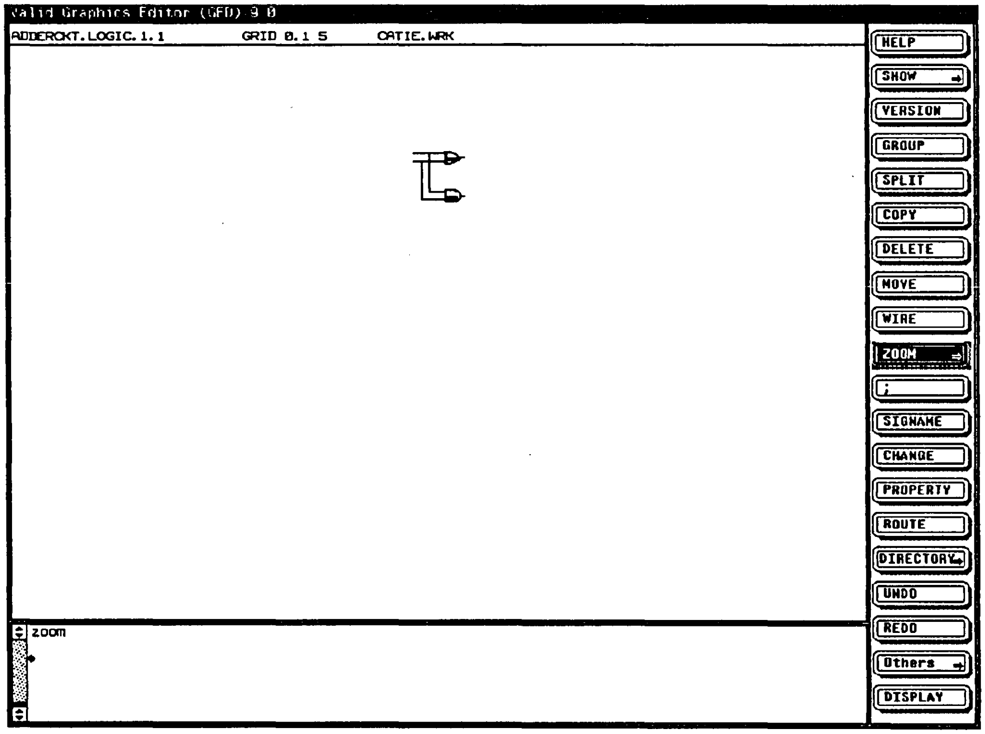
2 Use the cursor and left mouse button to define the first corner of the rectangle.

3 Move the cursor to the opposite corner and click the *middle* button. Arrows from the screen edges point to a stretchable rectangle.



4 Move the mouse to size the stretchable rectangle.

5 When the rectangle is the desired size, click the left button. When the left button is pressed, the area of the drawing displayed on the screen is reduced to fit into the rectangle.



```
ADDERCKT.LOGIC.1.1          GRID 0.1 5       CATIE.WRK
```

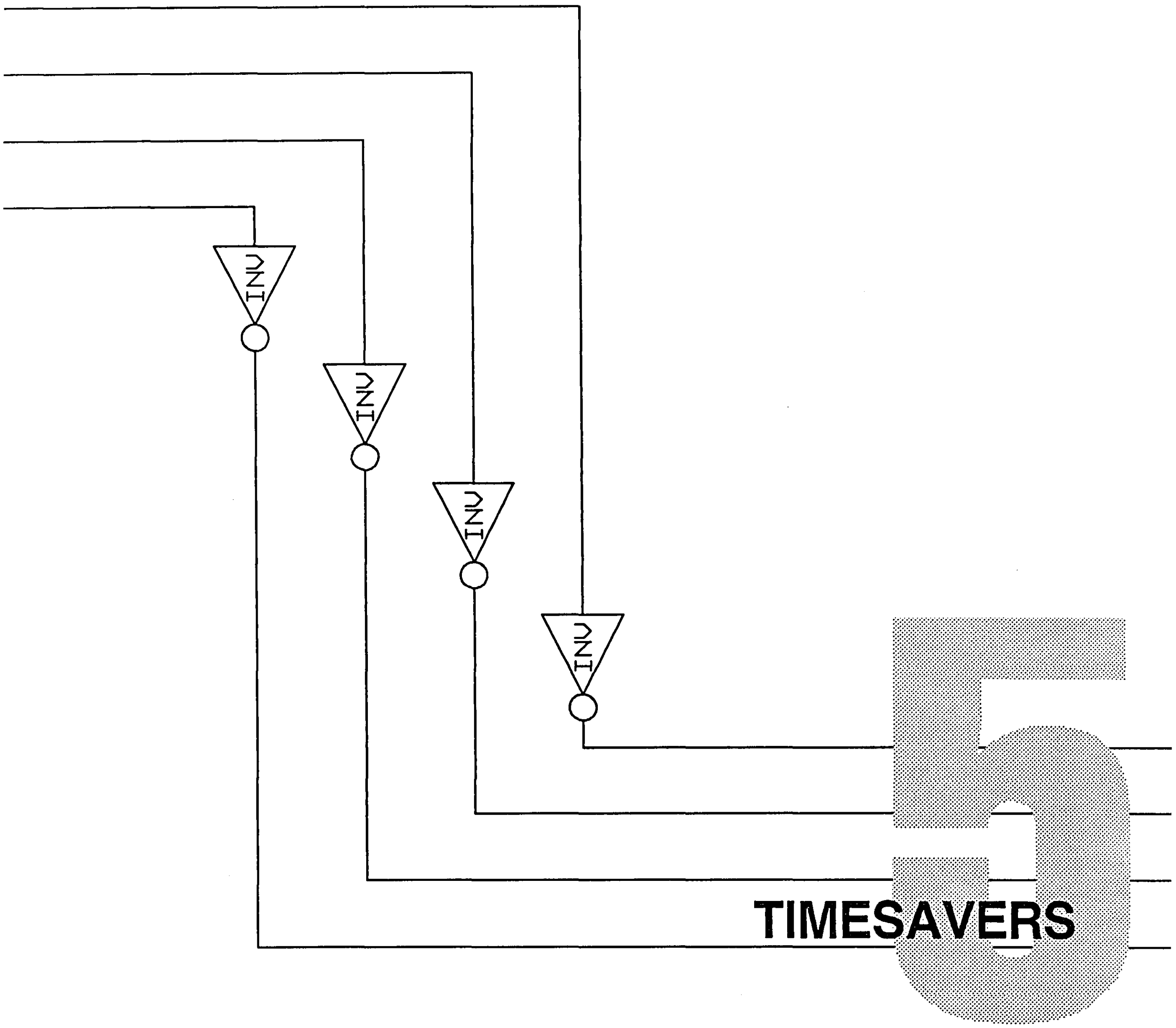
```
zoom
```

Tips on Using Zoom

The **window** *command is similar to the* **zoom** *command. Refer to the* ValidGED Command Reference Manual *for more information.*

- ✓ When sizing a drawing, GED uses the longest side of the stretchable rectangle as its reference and does not distort the drawing shape.
- ✓ If you enlarge or reduce a drawing too much or if you incorrectly position a drawing, use the **zoom fit** command to start over with the full drawing displayed on the screen.
- ✓ When reducing a drawing, you may see rectangular boxes at various places on the drawing. These boxes represent text that is too small to display on the screen. When you enlarge the drawing, the boxes are replaced with their actual text.
- ✓ The **zoom previous** command switches from the current zoom scale/position to the previous zoom scale/position.

LESSON 5



TIMESAVERS

Your workstation has several features that help you work more efficiently. Two features let you enter GED commands rapidly:

- Command abbreviations
- Predefined softkeys

Two more features are available but are not discussed in this tutorial:

- The *message window popup menu,* which provides shortcuts to some GED operations, such as **edit** and **get**.
- The **loadmenu** command, which lets you customize your on-screen GED menu with commands of your choice.

Refer to the manual, *Using ValidGED on Your Sun Workstation* for details about the message window popup menu and the **loadmenu** command.

A **help** command gives you help on each GED command. This lesson covers the following topics:

- Short Forms of Commands
- Using Softkeys
- On-line Help

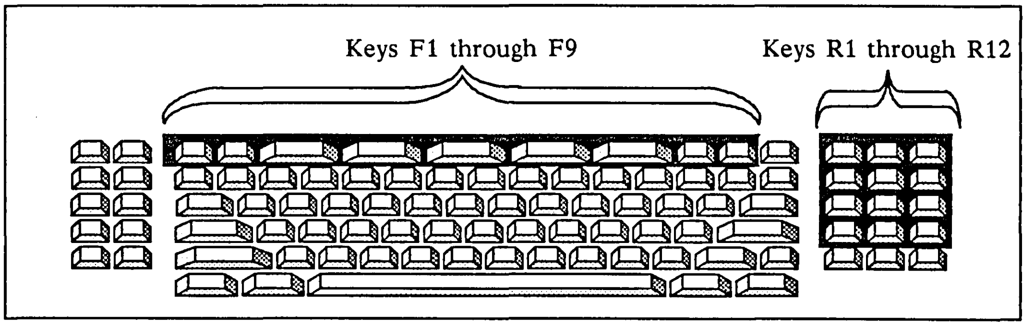
Short Forms of GED Commands

Appendix A lists the abbreviations for each command introduced in this tutorial.

Now that you're familiar with GED commands, you can save some time by using abbreviations for each command. You can type in just the first two or three letters of the command instead of the whole word. As long as what you type does not designate more than one command, GED accepts the short form of the command. The **add** command can be entered as **ad**, the **delete** command can be entered as **del**, and so on.

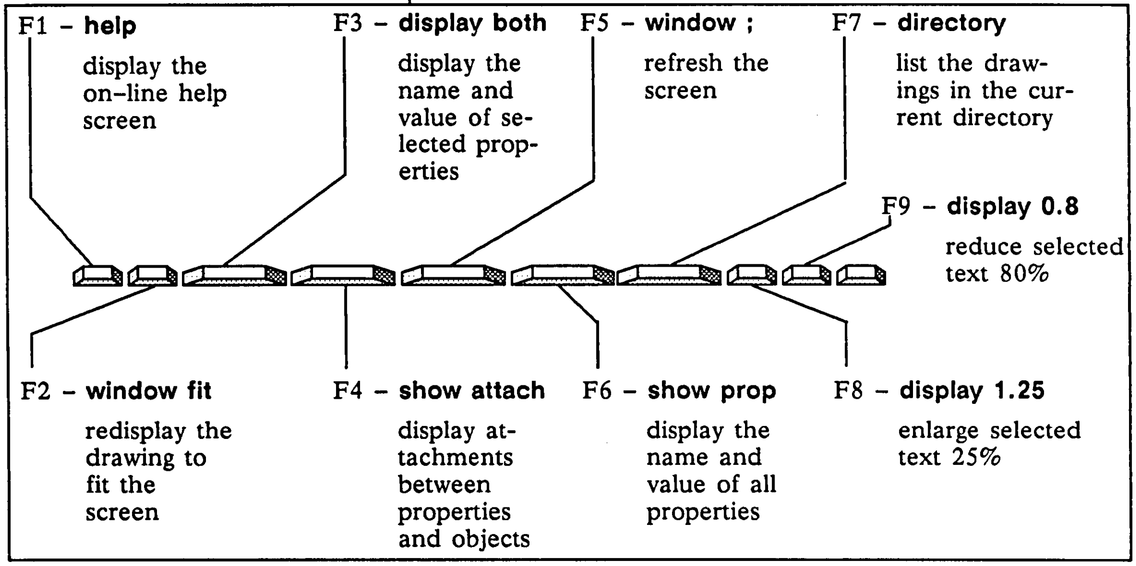
Using Softkeys

The other way to work more efficiently is by using function keys that are preset with frequently-used commands. The nine keys across the top of the keyboard (F1 through F9) and the top twelve keys on the numeric keypad (R1 through R12) are predefined keys.



To learn how to reassign any of the predefined keys, see the **assign** *command in the* ValidGED Command Reference Manual.

Figure 5–1 shows the default function key assignments. Some of these commands, such as **directory**, are already familiar to you. Others you learn about in later lessons. The **window** command is similar to the **zoom** command.


F1 - **help**
display the on-line help screen
F3 - **display both**
display the name and value of selected properties
F5 - **window ;**
refresh the screen
F7 - **directory**
list the drawings in the current directory
F9 - **display 0.8**
reduce selected text 80%
F2 - **window fit**
redisplay the drawing to fit the screen
F4 - **show attach**
display attachments between properties and objects
F6 - **show prop**
display the name and value of all properties
F8 - **display 1.25**
enlarge selected text 25%

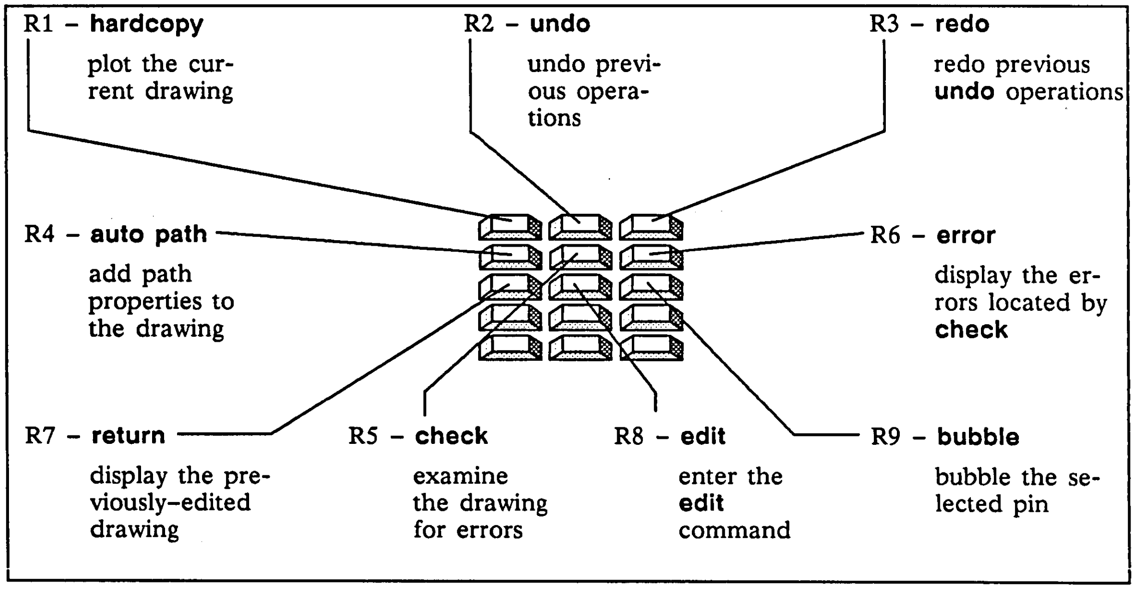


Figure 5-1. Default Function Key Assignments

On the right-hand keypad, keys R4 through R12 perform different functions when pressed with the Shift key. These shifted function keys are shown in Figure 5-2.

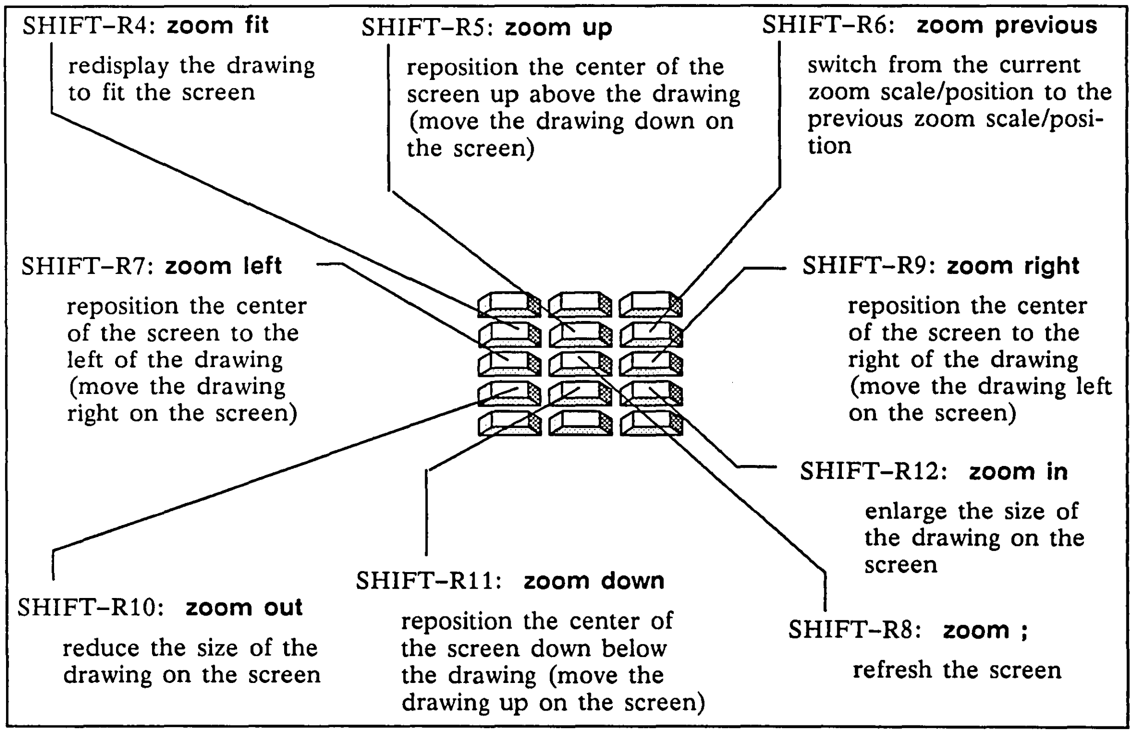

SHIFT-R4: **zoom fit**
redisplay the drawing to fit the screen
SHIFT-R5: **zoom up**
reposition the center of the screen up above the drawing (move the drawing down on the screen)
SHIFT-R6: **zoom previous**
switch from the current zoom scale/position to the previous zoom scale/position
SHIFT-R7: **zoom left**
reposition the center of the screen to the left of the drawing (move the drawing right on the screen)
SHIFT-R9: **zoom right**
reposition the center of the screen to the right of the drawing (move the drawing left on the screen)
SHIFT-R12: **zoom in**
enlarge the size of the drawing on the screen
SHIFT-R10: **zoom out**
reduce the size of the drawing on the screen
SHIFT-R11: **zoom down**
reposition the center of the screen down below the drawing (move the drawing up on the screen)
SHIFT-R8: **zoom ;**
refresh the screen

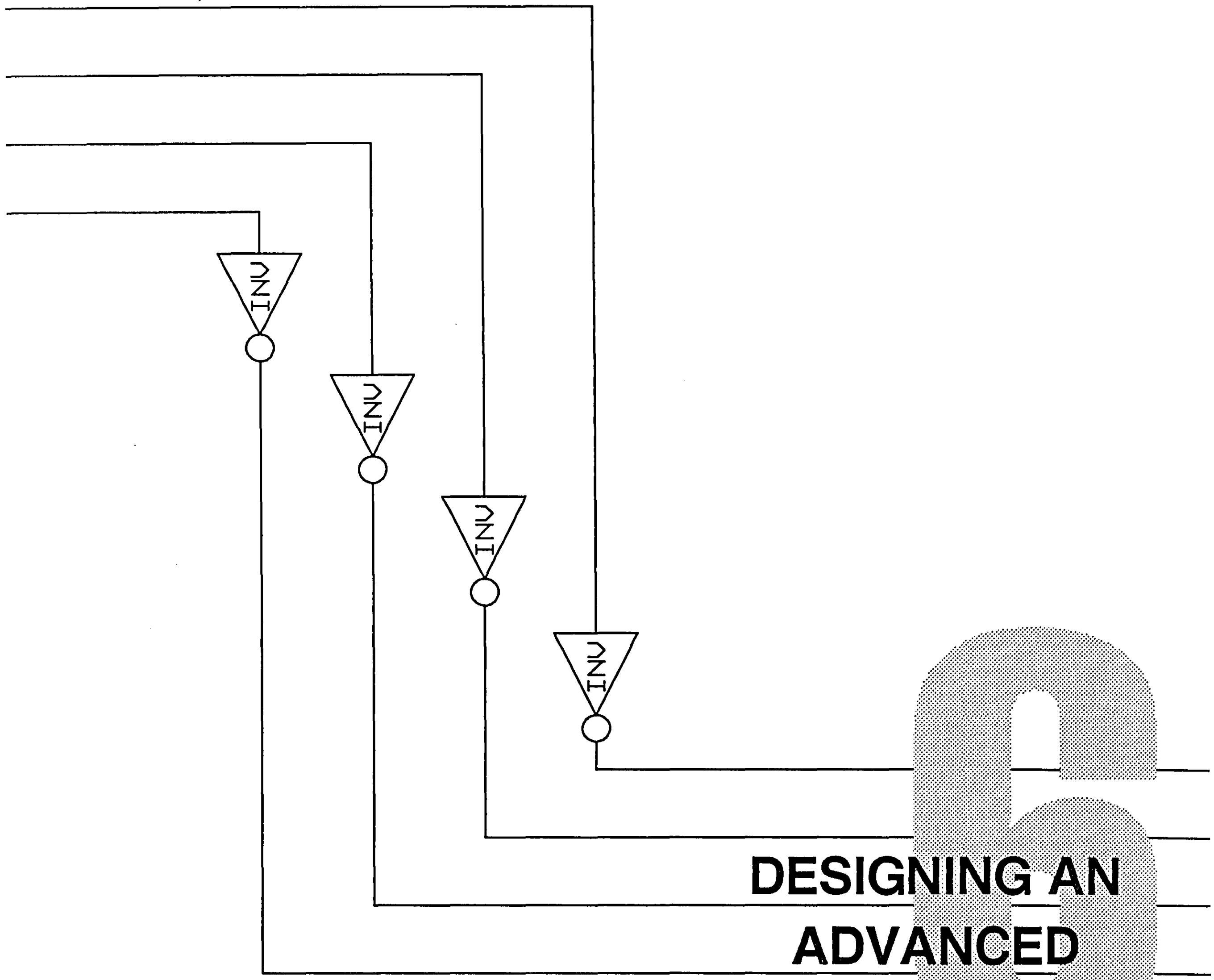
Figure 5-2. Shifted Function Key Assignments

On–Line Help

To help you use GED more efficiently, on–line help for each GED command is available. To get help on a GED command, select **help** from the menu, type **help** from the keyboard, or press the F1 key and then either select the command you want help on from the menu or type its name at the keyboard.

For a complete list of GED commands for which help is available, select **help** twice from the menu or type **help help**. To exit from the **help** command, type or select any GED command except **window** or **zoom**.

LESSON 6



DESIGNING AN ADVANCED CIRCUIT

In this lesson you draw a more advanced circuit, the *subtractor* circuit, to learn how to:

- Select different part versions
- Rotate parts in the drawing
- Mark wire intersections individually or automatically
- Separate dots from wires or wires from pins, or split one wire into two
- Show signal name and wire attachments
- Reattach incorrectly connected signal names and wires
- Add an abbreviation to the drawing
- Attach properties to bodies, signal names, and wires
- Check a drawing for errors
- Display error messages for any errors found on the drawing

New Commands

add drawing	**error**	**route**
autodot	**property**	**show attach**
check	**reattach**	**split**
dot	**rotate**	**version**

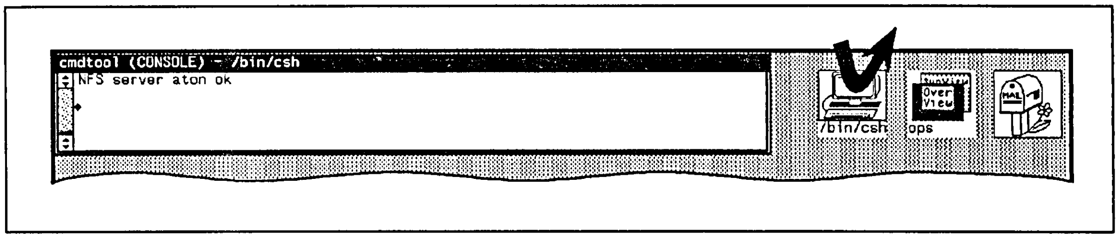
Logging On

If you are not currently logged on, follow these steps to start your drawing named subtractor:

1 Type:

```
suntools
```

2 Select a /bin/csh icon.



```
NFS server aton ok
```

3 Move the cursor into the Shelltool – /bin/csh window that opens.

4 Type:

```
ged subtractor
```

Starts the graphics editor program and edits the drawing named **subtractor**.

5 Type:

```
library tutorial
```

Lets you access the tutorial library.

Starting a Drawing

If your full-adder circuit (or any other drawing) is currently on your screen, follow these steps to start editing a new drawing named subtractor:

Saves your full-adder drawing.

1 Type

```
write adderckt
```

2 Type:

```
edit subtractor
```

The screen clears and the drawing name

```
SUBTRACTOR.LOGIC.1.1
```

appears in the status line.

If you forget how to use a command introduced earlier, go back to the appropriate lesson for detailed instructions or use the **help** *facility.*

You are now ready to draw the subtractor circuit shown in Figure 6-1.

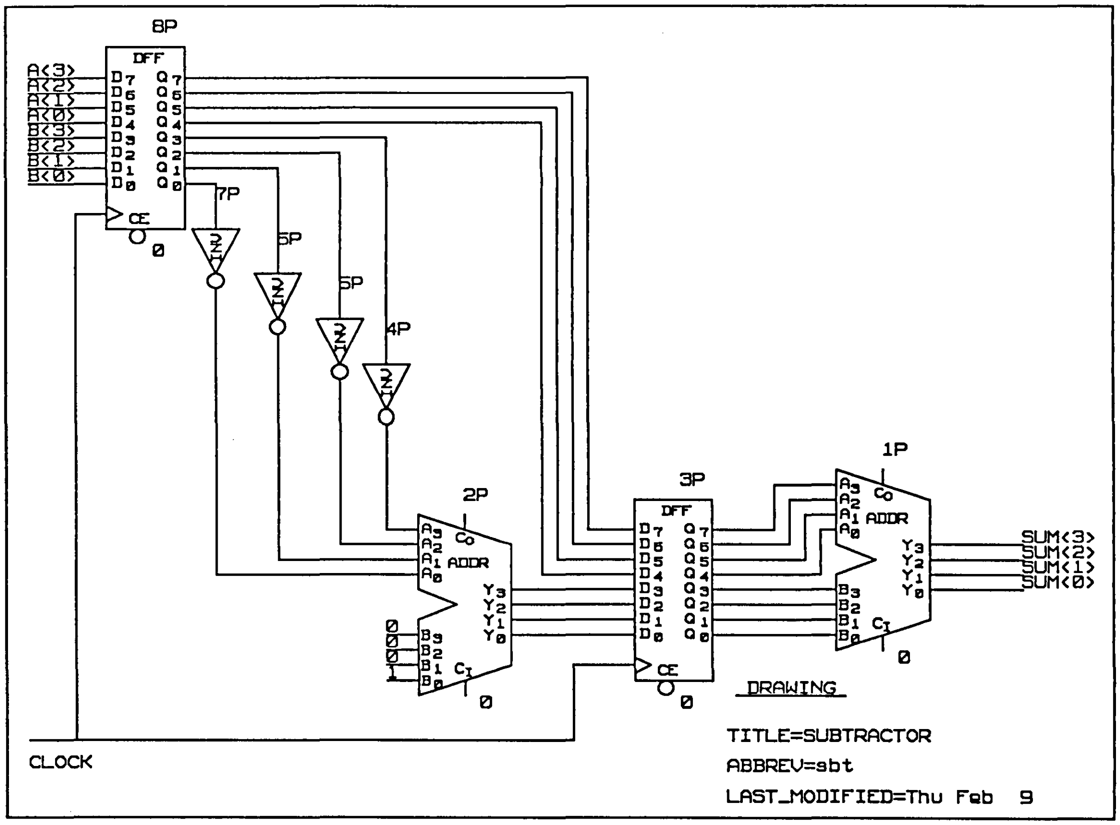


Figure 6–1. Subtractor Circuit

Adding Parts to the Drawing

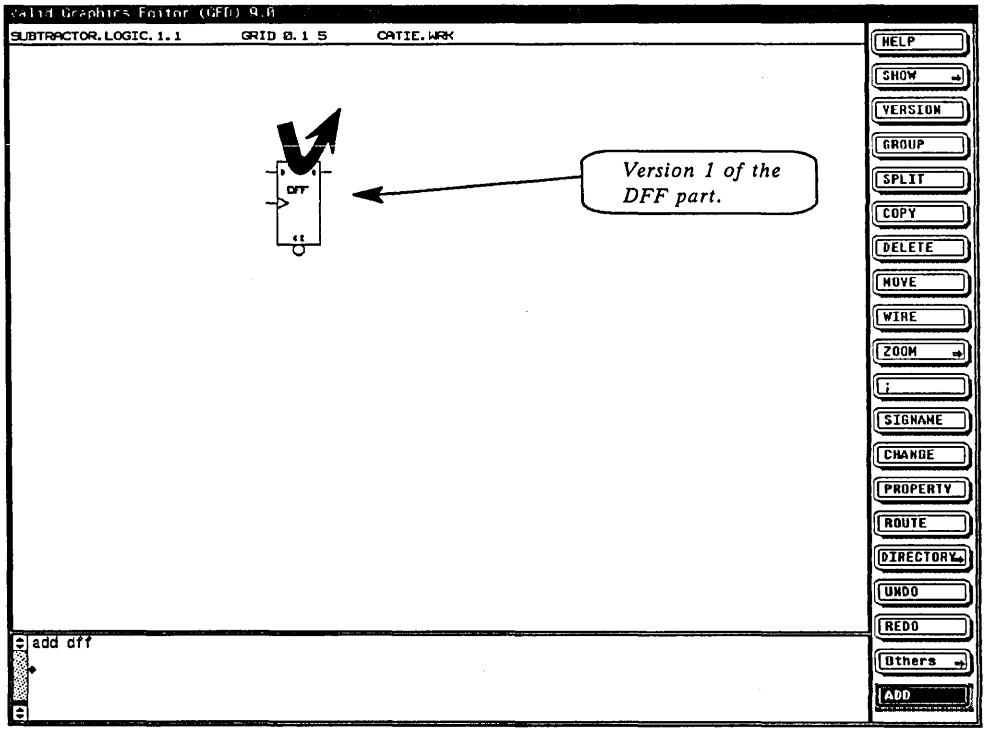
Use the **add** command to add the four main bodies in their approximate locations.

1 Type:

```
add dff
```

A DFF library part appears at the cross-hair cursor.

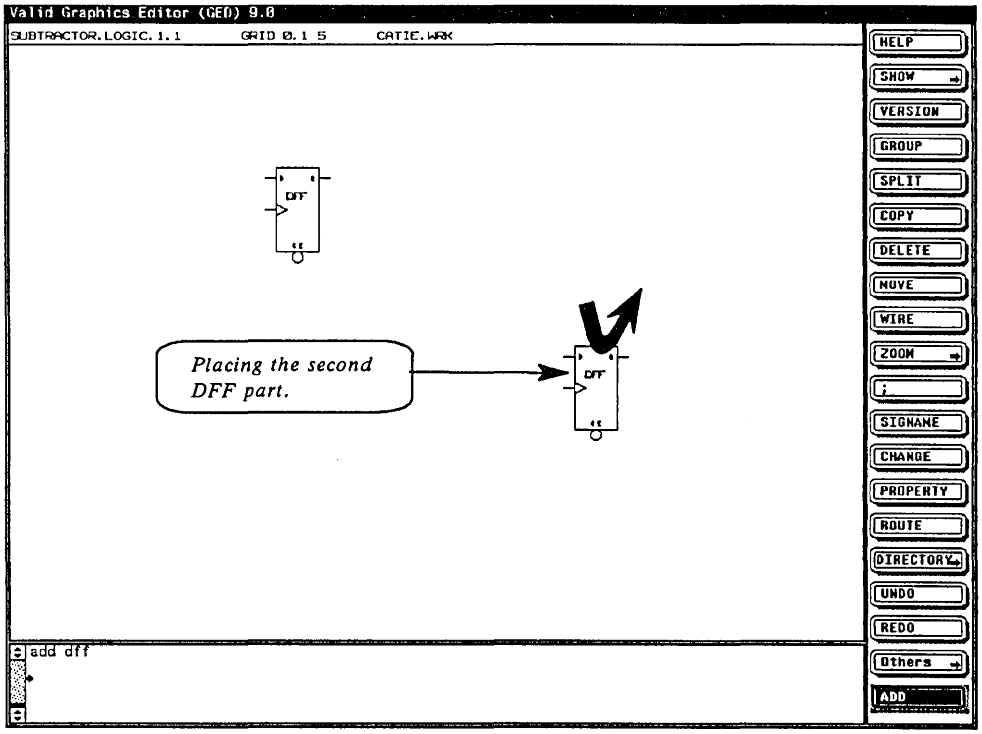
2 Click the left button to place the library part on the screen.



```
Valid Graphics Editor (GED) 9.0
SUBTRACTOR.LOGIC.1.1        GRID 0.1 5        CATIE.WRK
```

```
add dff
```

3 Click the left button again to bring another DFF onto the screen, then place it down.

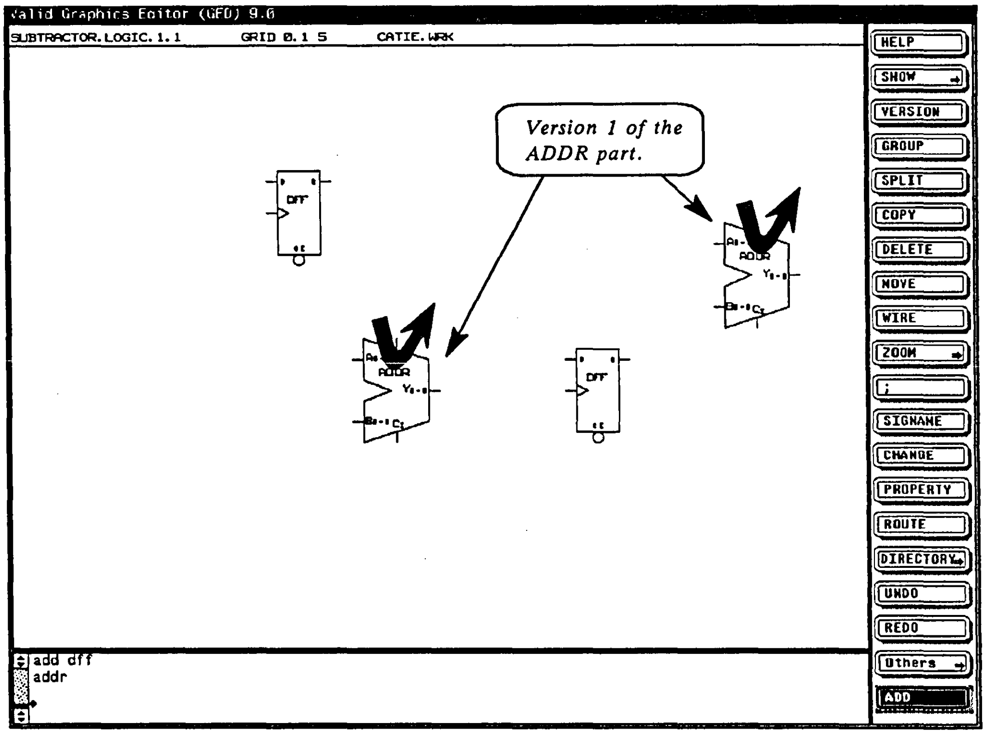


```
add dff
```

4 While still in the **add** command, type:

```
addr
```

5 Place one ADDR on the screen, and then another.



```
add dff
addr
```

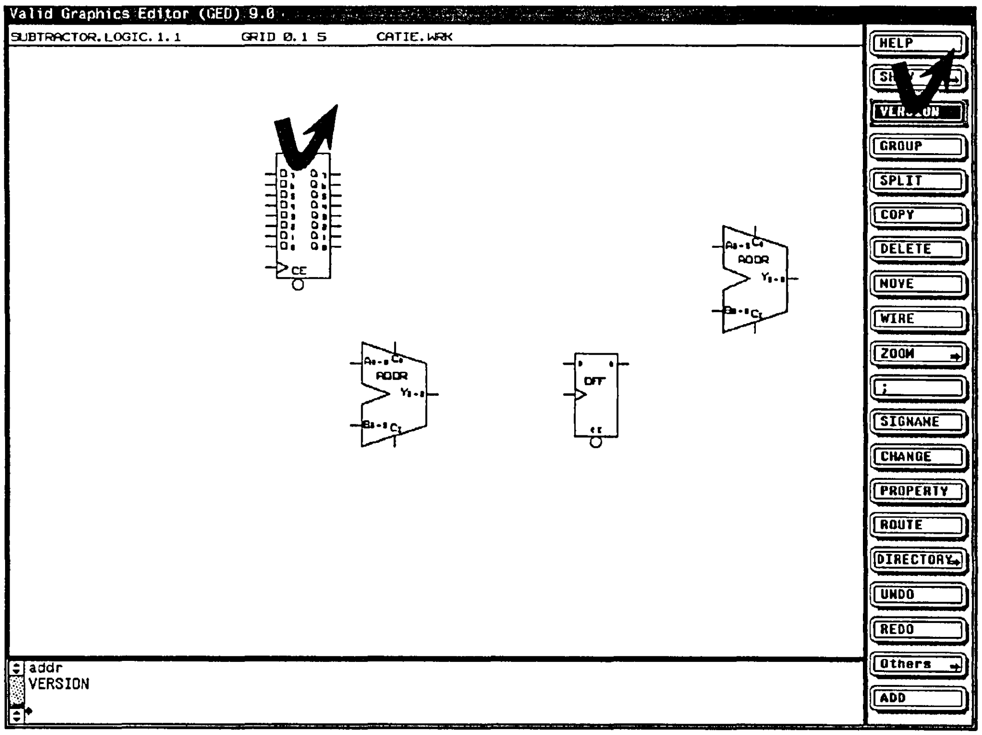
The Version Command

Note that the bodies you placed do *not* look like those in the subtractor circuit in Figure 6–1. This is because there are multiple versions of each of these library parts.

To see the other versions of a part and to select the one you want to use, follow these steps:

1. Select the **version** command from the menu.
2. Move the cursor to the DFF part and click the left button.

   Version 2 of the part appears on the screen.


```
Valid Graphics Editor (GED) 9.0
SUBTRACTOR.LOGIC.1.1        GRID 0.1 5        CATIE.WRK
```
```
addr
VERSION
```

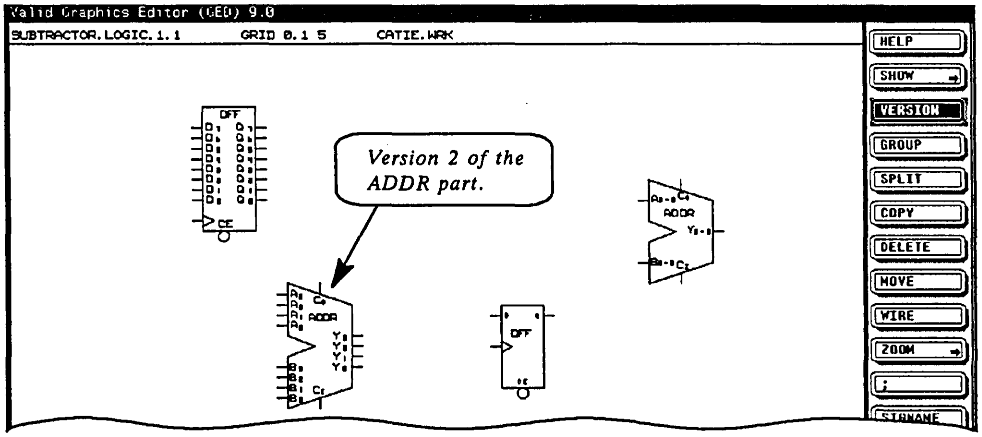
3 To see if there are more versions of this part, click the left button again.

Version 1 reappears on the screen. This means that there are only two versions of this part.

4 Click once more to go back to version 2.

5 Move the cursor to the ADDR part.

6 Click the left button several times to cycle through the available versions (there are only two versions of this part) and select version 2.



```
Valid Graphics Editor (GED) 9.0
SUBTRACTOR.LOGIC.1.1      GRID 0.1 5      CATIE.WRK
```

7 Use the **version** command on the other DFF and the other ADDR.

Tips on Versioning

✓ If you know that you want version 2 of a part, you can add it directly by specifying the version number of the part when you enter the **add** command.

EXAMPLE

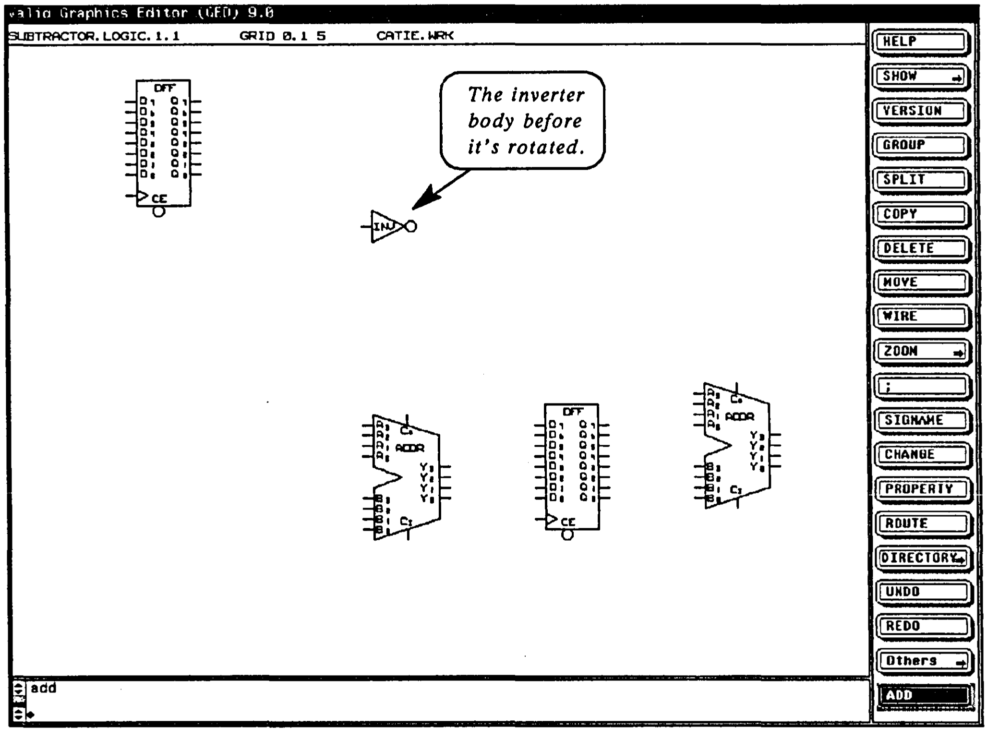
```
add addr..2
```

Remember from Lesson 3 that the third field of the drawing name is for the version number. This further illustrates that the version field of the drawing name is used for versions of library parts, not versions of your own schematics.

The Rotate Command

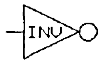
Now add the inverters to your drawing. Type:

```
add inv
```

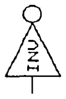


The inverter that appears on the screen isn't turned the right way. To rotate the part, click the middle button three times *before* you place the inverter on the screen with the left button.

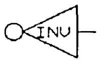
Original inverter



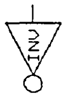
First rotation



Second rotation



Third rotation



It's faster to copy the inverter than to add three inverters and then rotate each one.

If you placed the inverter on the screen without rotating it first, you can use the **rotate** command. Type

```
rotate
```

and then move the cursor to the inverter and click the left button until the part is turned the way you want it.

Now copy the rotated inverter three times:

1 Select the **copy** command.

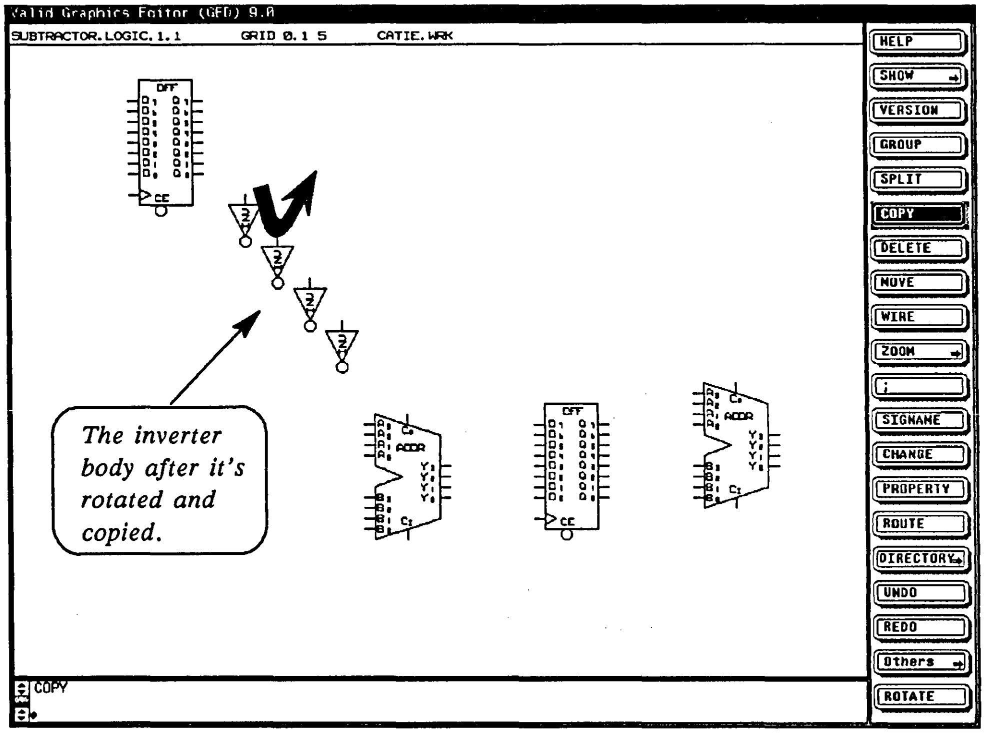
2 Move the cursor to the inverter and click the left button. A second inverter appears on the screen.

3 Type:

   3 Return

   (This is the number of copies you want.)

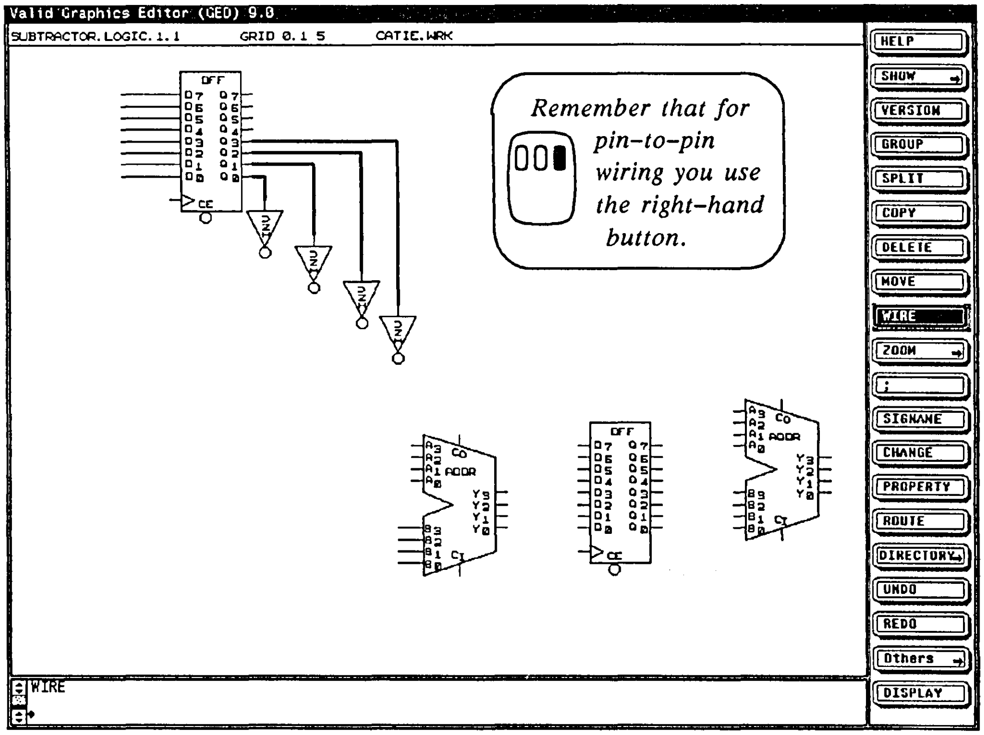
4 Move the cursor to the location where you want the first copy to be positioned, and click the left button. Three copies appear in succession.



Wiring the Drawing

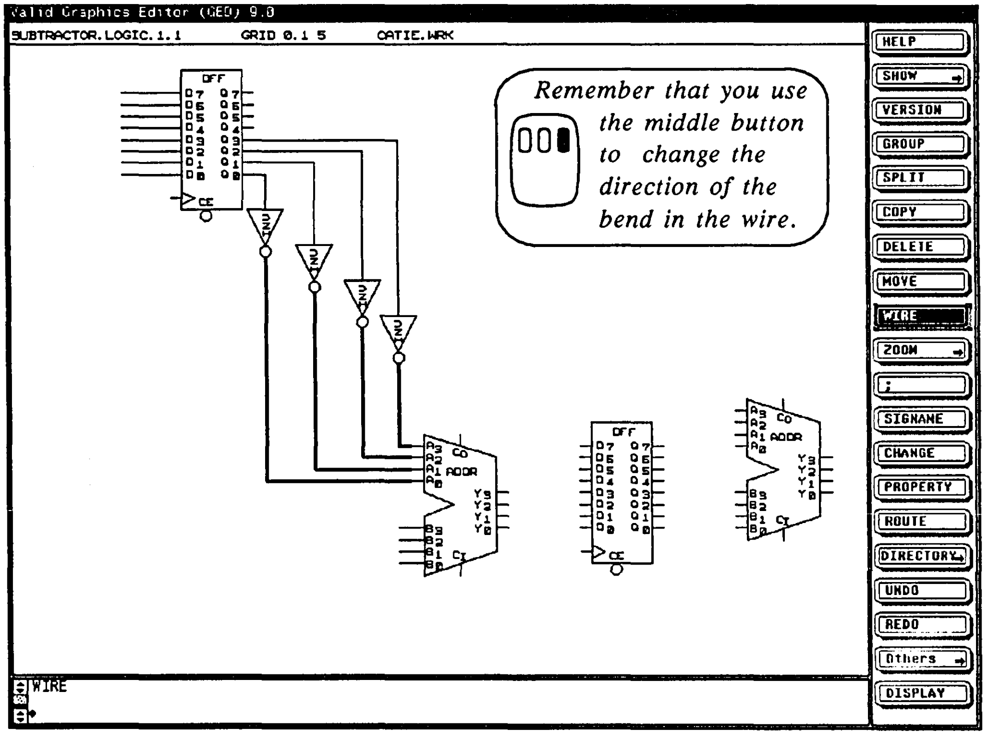
Now that you have added, versioned, and positioned the parts for your subtractor circuit, use the **wire** command to interconnect the parts.

1 Wire from the left-hand DFF outputs to each of the four INVs starting with the Q0 output.

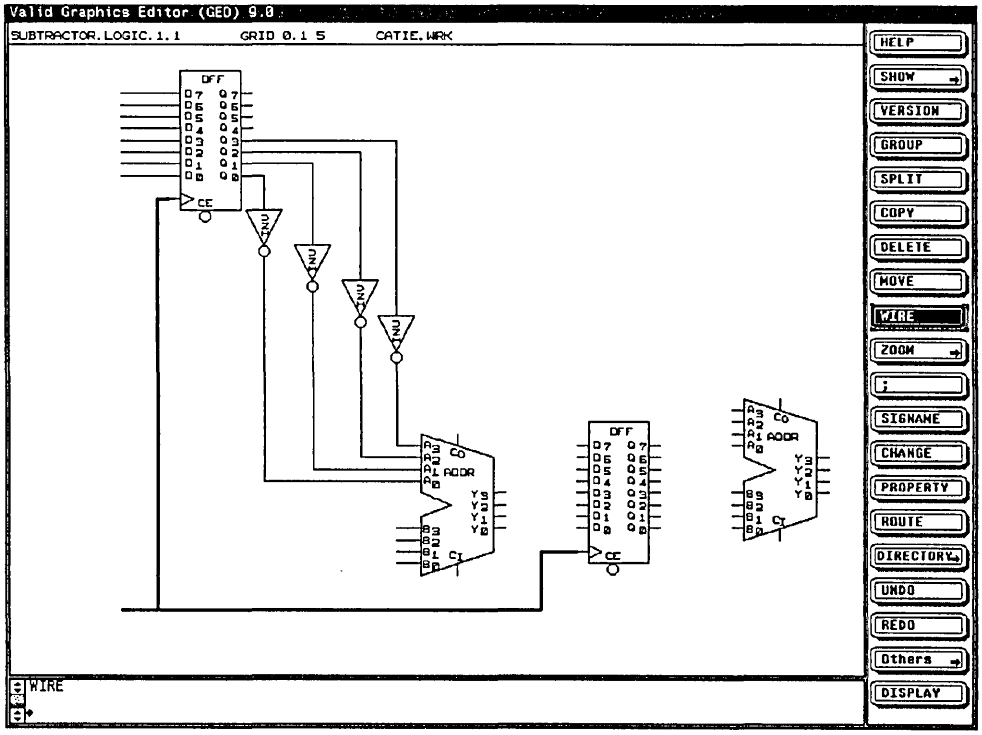


If necessary, use the **move** command on each inverter to align them. Notice how the wires move with the inverters.

2 Wire from each of the four INVs to the ADDR, starting with the closest one.



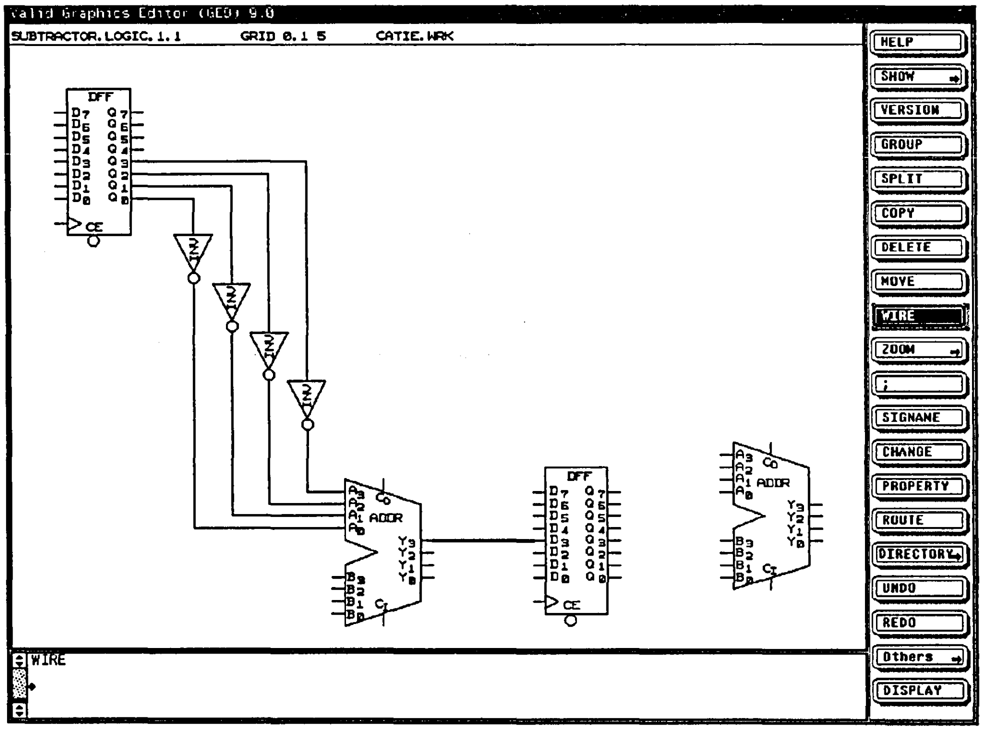
3 Wire the two clock signals.



Next you learn a quick way to wire the ADDR outputs to the DFF inputs.

1 If necessary, move the right-hand DFF so that you can wire in a straight line from the ADDR outputs to the D0 to D3 inputs of the DFF.

2 Wire pin-to-pin (right-hand button) from the Y3 output on the ADDR to the D3 input on the DFF.



```
Valid Graphics Editor (GED) 9.0
SUBTRACTOR.LOGIC.1.1        GRID 0.1 5        CATIE.WRK
```

```
WIRE
```

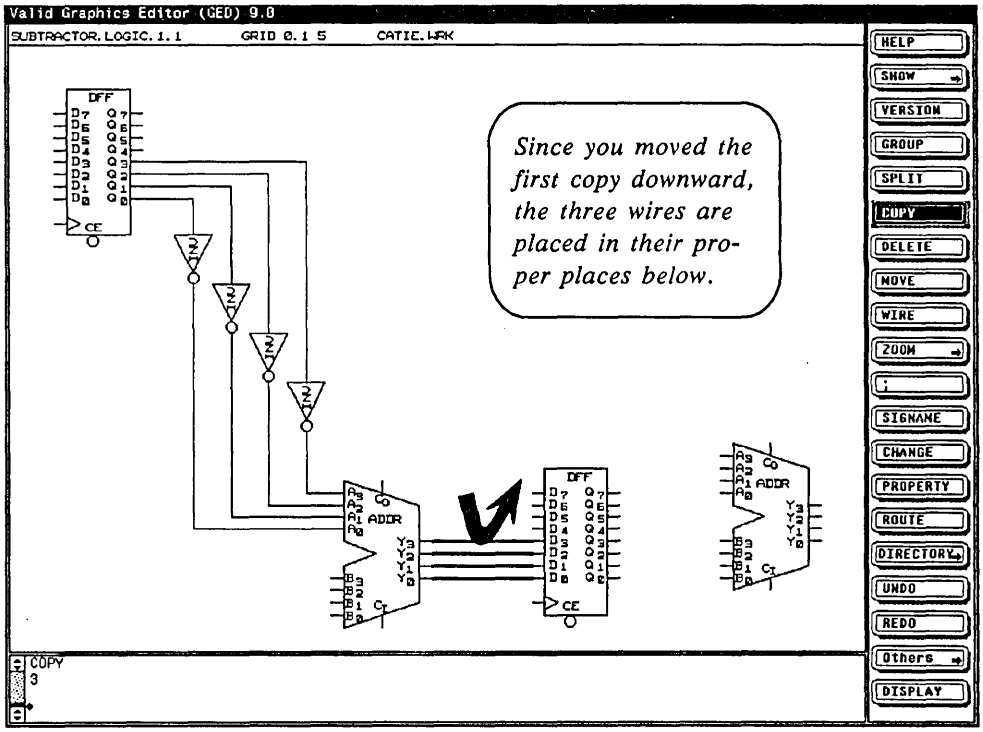
3 Select **copy** from the menu.

4 Move the cursor to the wire you just created and click the left button.

5 Type:

3 Return

6 Move the cursor down to align the wire with Y2 and click the left button again.


Valid Graphics Editor (GED) 9.0

SUBTRACTOR.LOGIC.1.1 GRID 0.1 5 CATIE.WRK

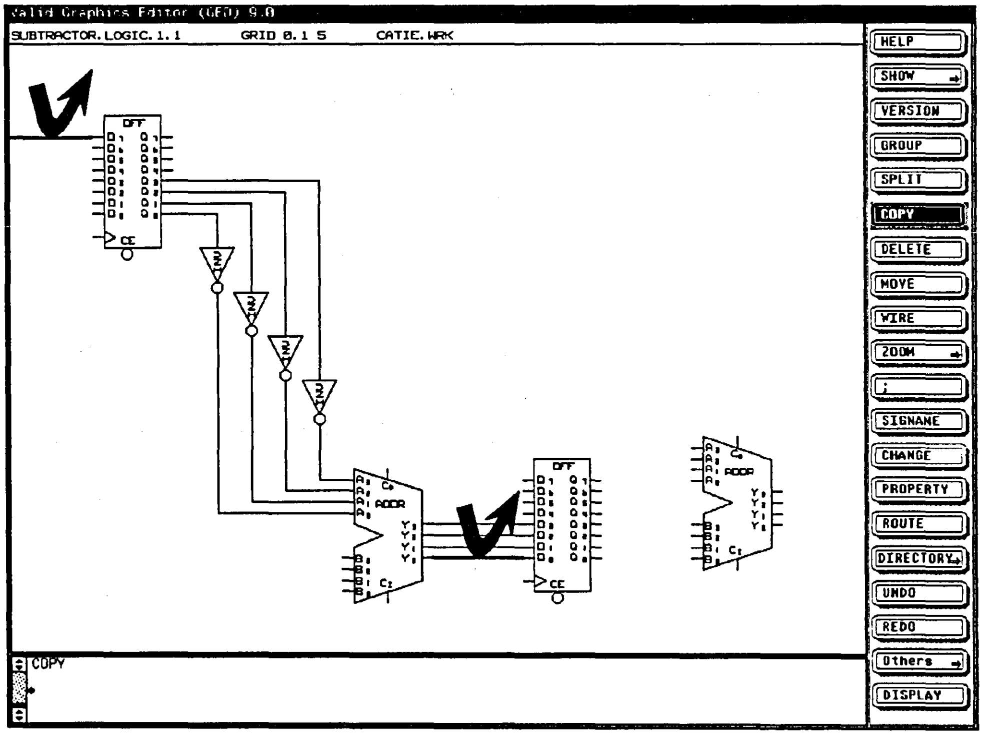
Since you moved the first copy downward, the three wires are placed in their proper places below.

COPY
3

Next, wire the right-hand DFF to the nearest ADDR.

1 Move the cursor to the wire aligned with Y0 and click the left button to copy it.

2 Move the cursor up to the inputs of the first DFF, align the wire with D7, and click the left button to place the first copy.

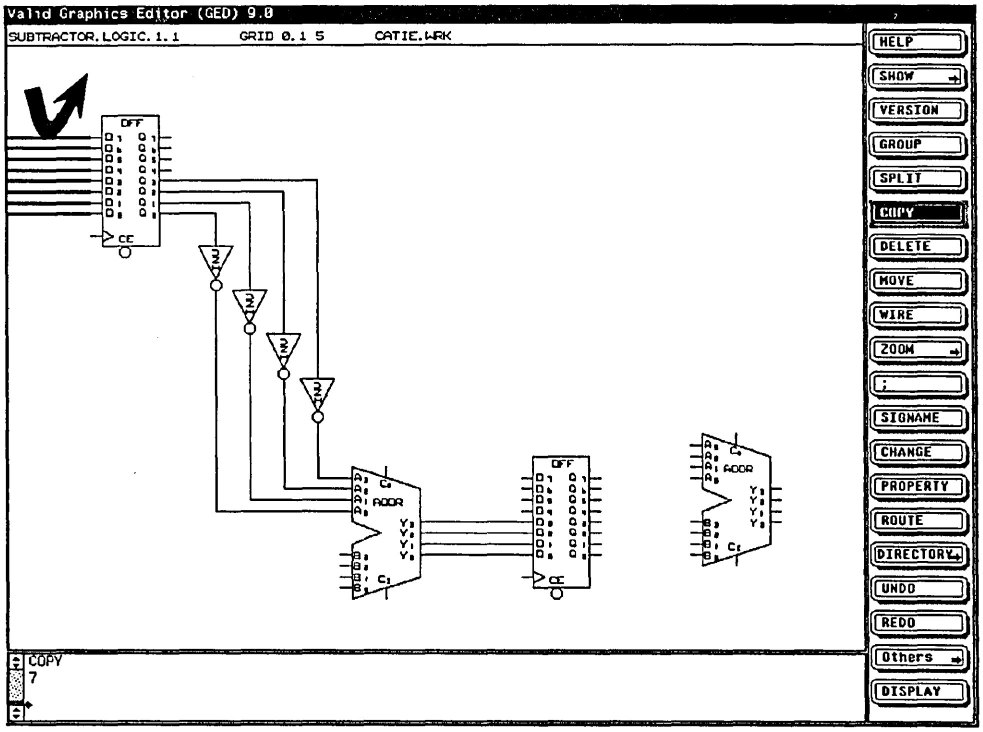


3 Now click the left button again on the D7 wire to copy it.

4 To place the same wire on the other seven inputs of the first DFF, type:

7 Return

5 Align the wire with D6 and click the left button.



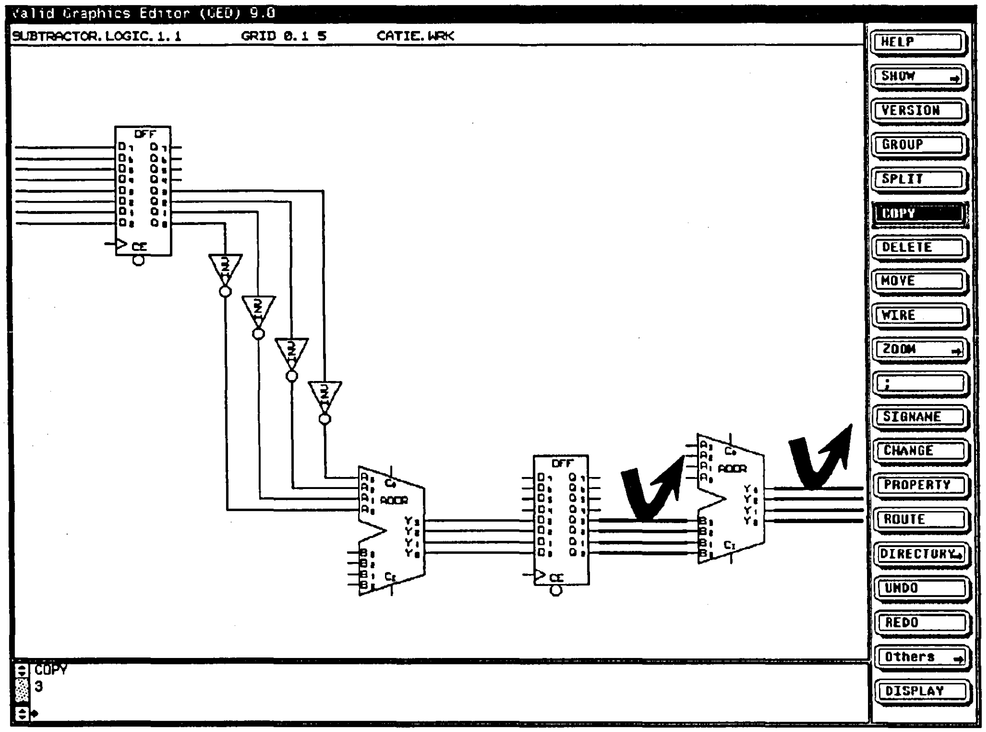
```
Valid Graphics Editor (GED) 9.0
SUBTRACTOR.LOGIC.1.1          GRID 0.1 5          CATIE.WRK
```

```
COPY
7
```

For more information on the **copy** *command, see the* ValidGED Command Reference Manual.

Using the **copy** command to copy multiple wires saves time and produces input wires that are all the same length and that are evenly spaced.

Use the same combination of the **wire** and **copy** commands to wire the second DFF to the second ADDR and to wire the outputs of the ADDR.



```
COPY
3
```

The Route Command

Now you need to wire the outputs of the left-hand DFF to the inputs of the other DFF. A simple way to do this is with the **route** command. The **route** command wires components together by drawing a line between two points.

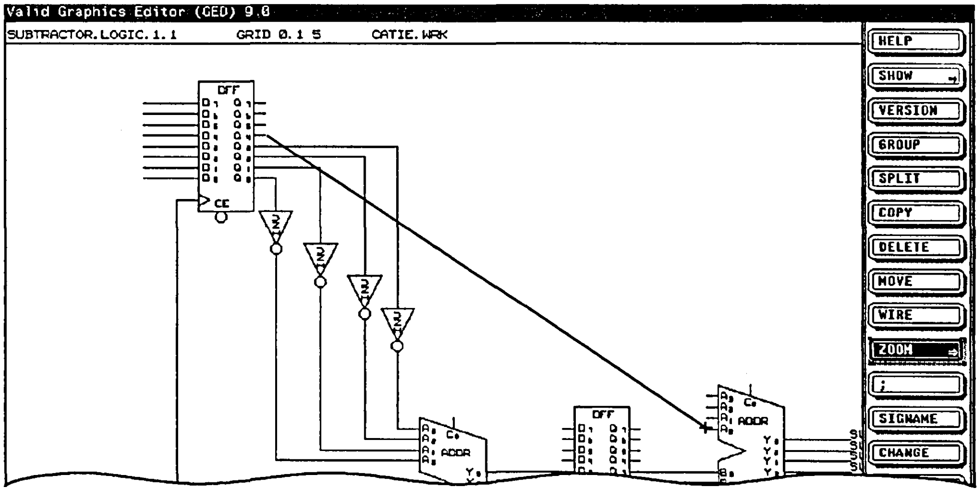
Follow these steps to wire the DFF parts:

1 Type:

```
route
```

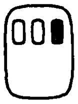
2 Move the cursor to the Q4 wire of the left-hand DFF and click the right button to select the first point. A flexible line is attached to the end of the Q4 wire.

3 Now move the cursor to the D4 wire.

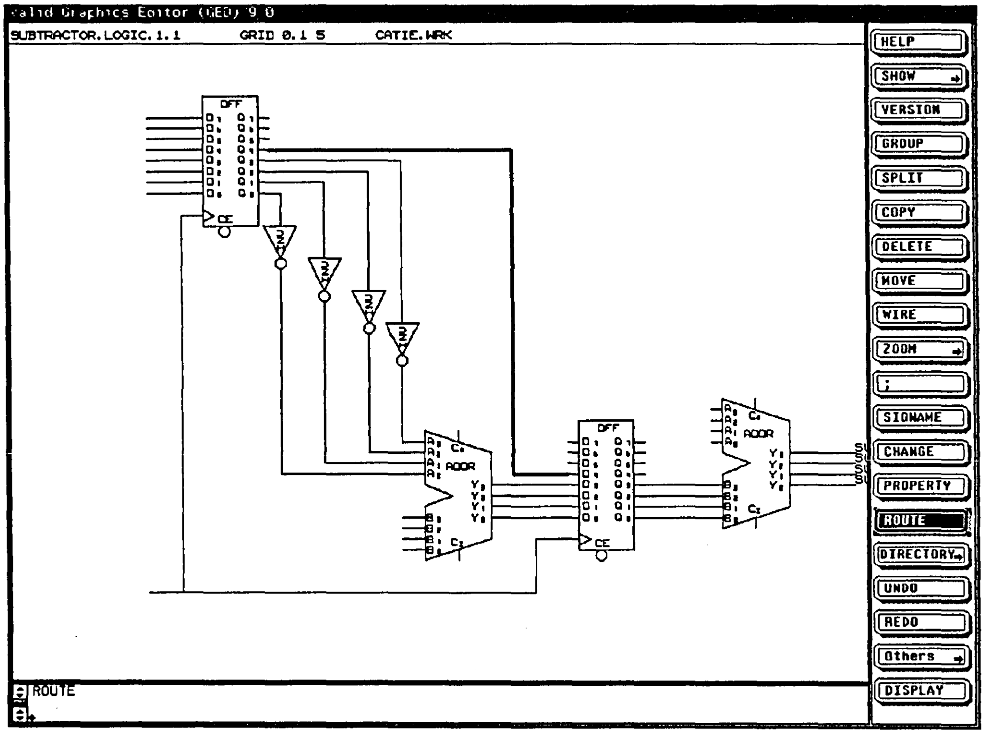


Valid Graphics Editor (GED) 9.0

SUBTRACTOR.LOGIC.1.1 GRID 0.1 5 CATIE.WRK

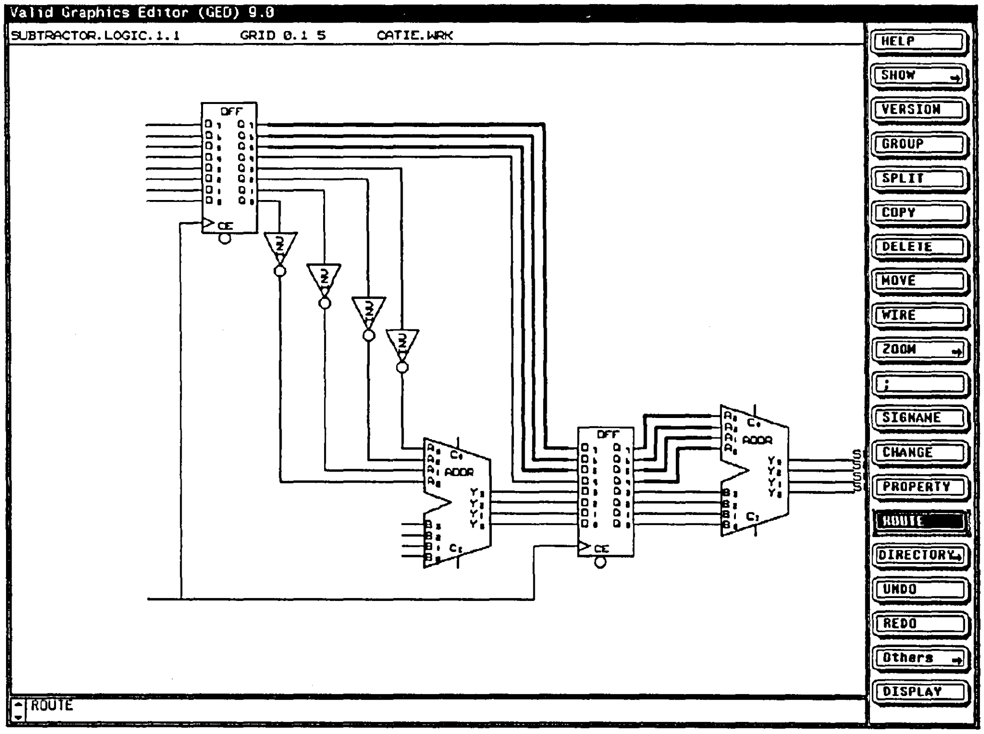


4 Click the right button to attach the flexible line to the second point. **route** connects the two points with a wire.



ROUTE

5 Use the same technique to wire the rest of the DFF outputs and to wire the right-hand DFF to the ADDR.



Tips on Using the Route Command

- Use the blue button to select the nearest pin or wire vertex for a **route** point. Use any other button to select the nearest grid point.
- If it cannot draw a horizontal or vertical line, **route** draws a diagonal line.
- **route** will not run a wire through any existing objects.

The Dot Command

To clearly mark the connection of the two clock signals, you can use the **dot** command. There are two styles of dots: open and filled.

open dots: ○○○

filled dots: ●●●

Open dots are the default. If you want filled dots, enter this GED command:

set dots_filled

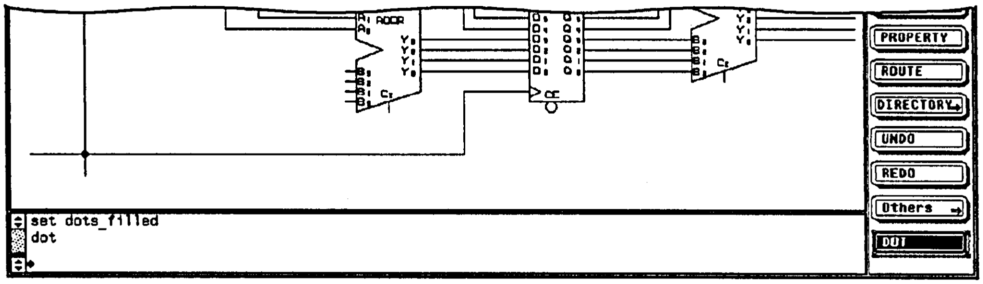
See Lesson 7 for instructions on entering commands into your startup.ged *file.*

If you want to use filled dots on a regular basis, you can enter the **set dots_filled** command into your *startup.ged* file.

Now mark the clock signal connection. The **dot** command is very simple to use:

1 Select **dot** from the GED menu.

2 Move the cursor to the intersection of the two wires and click the left button. A dot appears.



```
set dots_filled
dot
```

Tips on Using the Dot Command

- ✓ In a complex circuit, it is time–consuming to place every dot by hand. Instead, you can use **show connect** to check that no extra connections have been created by mistake. This command places an asterisk temporarily on the drawing to highlight each connection point. Check that the connection points are all correct. Use the refresh command (**zoom** ;) to remove the asterisks from the screen. Then type **auto dot** and all of the wire junctions are automatically dotted.

- ✓ The system convention is that a T–junction is automatically a connection, whether or not it is dotted. A four–way intersection (+) is not a connection *unless* it is dotted.

The Split Command

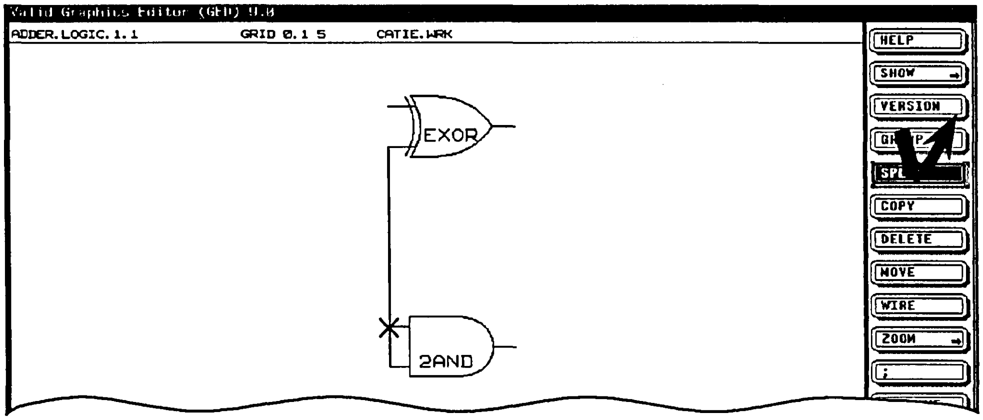
Another useful command, particularly for modifying a drawing, is the **split** command. This command lets you:

- Separate a dot from a wire
- Separate a wire from a pin
- Split one wire into two

The figure below shows the pins of a body wired together by mistake.

You could use the following procedure to split the wires from the pins and then reconnect the wires.

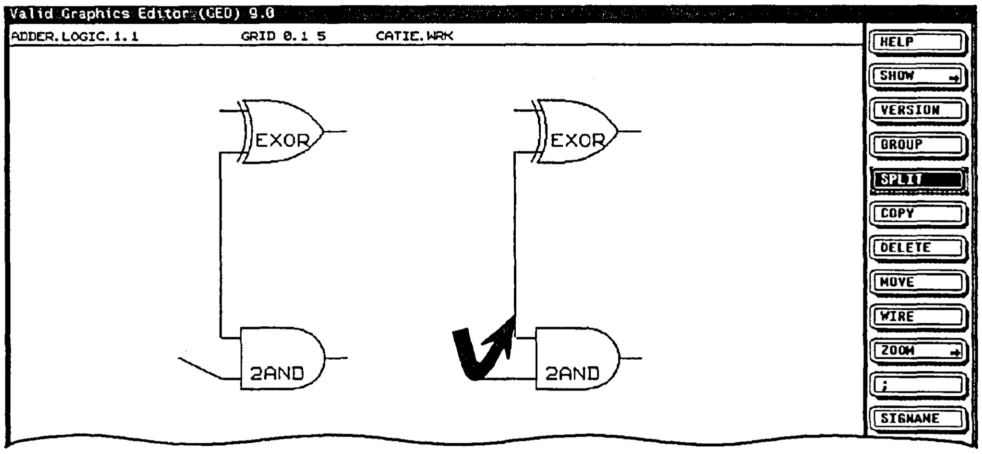
1. Select **split** from the menu.
2. Move the cursor to the upper input pin of the 2AND where the pin and wire join. The figure below shows an X where you should place the cursor.



3 Click the right button. Move the cursor away and watch what moves.

4 There are two wire segments connected here. You want the lower one. If you get the upper one, just move the cursor back to the junction, and click the right button again.

5 When you have picked up the correct segment, move it out perpendicular to the body, and click the left button to set down the split wire.



Valid Graphics Editor (GED) 9.0

ADDER.LOGIC.1.1   GRID 0.1 5   CATIE.WRK

Tips on Using the Split Command

✓ The **split** command is very handy. When several objects are all connected together and you place the cursor on top of them, each click of the right button selects a different object to move away from the others. This lets you cycle through the various options until you reach the one you want. So don't worry about pointing to the "correct" object. Just point to all of them and click. Click once or several times to select the object to move.

Choosing Signal Names

Signal names not only identify each important signal in a circuit, they also are entered in the database for the drawing. They are therefore a powerful tool for entering data about signals that can be used by verification programs.

Signal name syntax is explained in detail in the SCALD Language Reference Manual.

For now you need only know one detail of this syntax, the pointed brackets (<>). These are used to designate bits of a bus. The signal names

A<3>
A<2>
A<1>
A<0>

are understood by the system to be bits 3–0 on the bus named A.

Review Lesson 2 if you forget how to use the **signame** *command.*

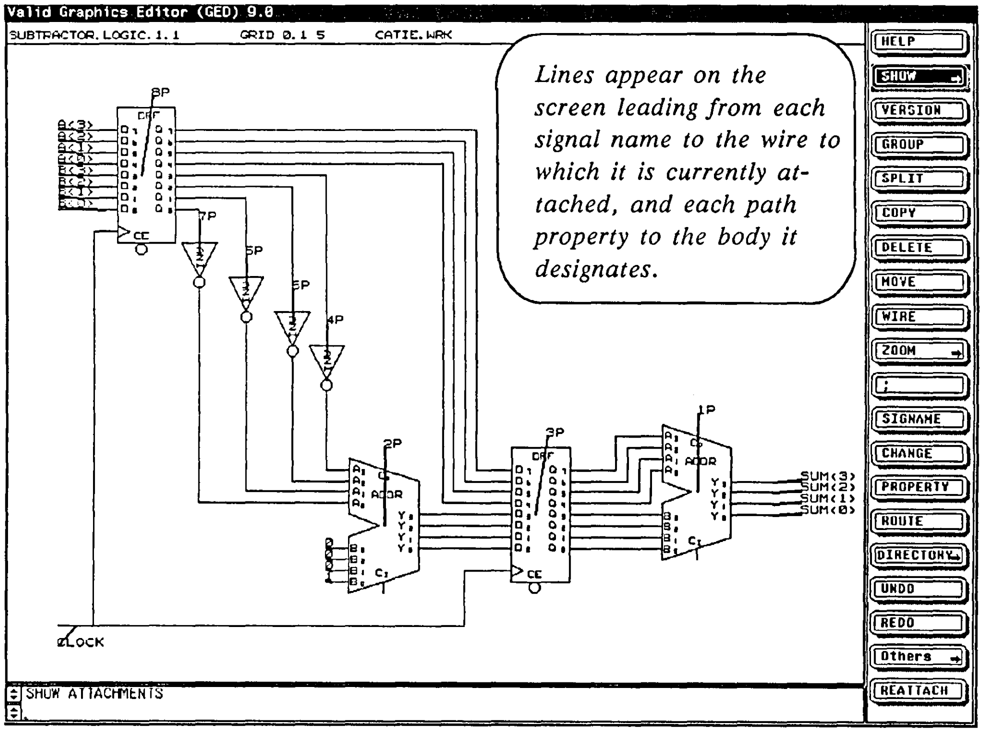
Use the **signame** command to attach names to each of the signals on the drawing. Use the signal names shown in Figure 6–1.

Note: Don't forget to add the signals 0 and 1, as shown in Figure 6–1, to your drawing. These are two special signals that represent logic 0 and logic 1.

The Reattach Command

After you have given the signal names, it is important to check that each name has been properly assigned to the correct signal. To do so, type:

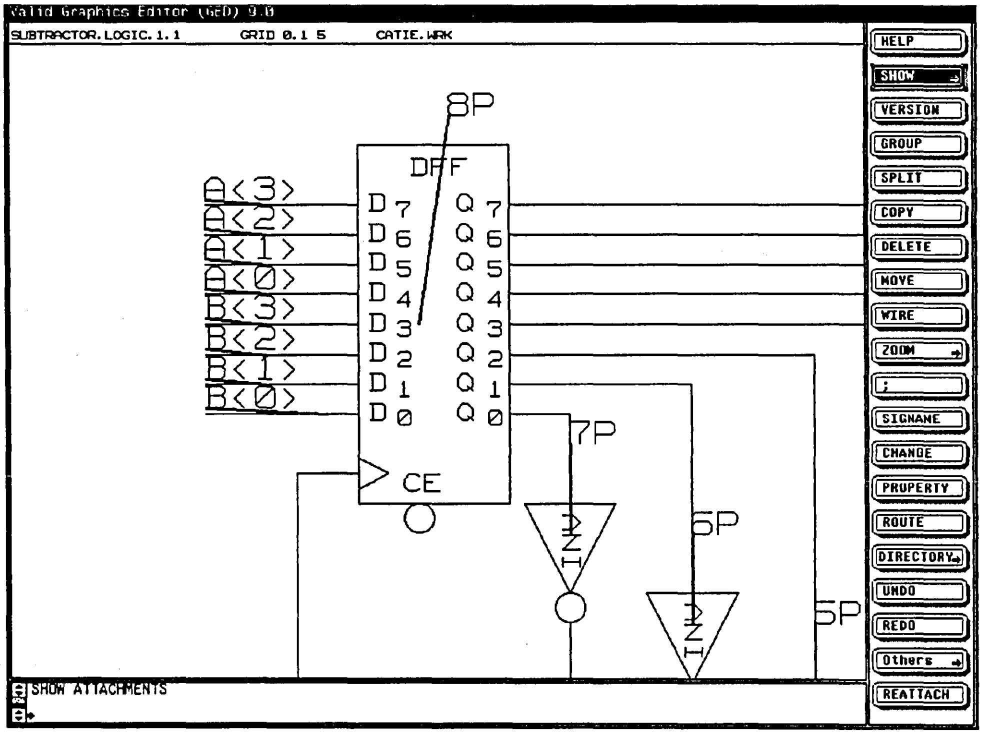
```
show attach
```



SHOW ATTACHMENTS

Lines appear on the screen leading from each signal name to the wire to which it is currently attached, and each path property to the body it designates.

To see the attachments more clearly, you can use the **zoom** command to zoom in on a section of your design.



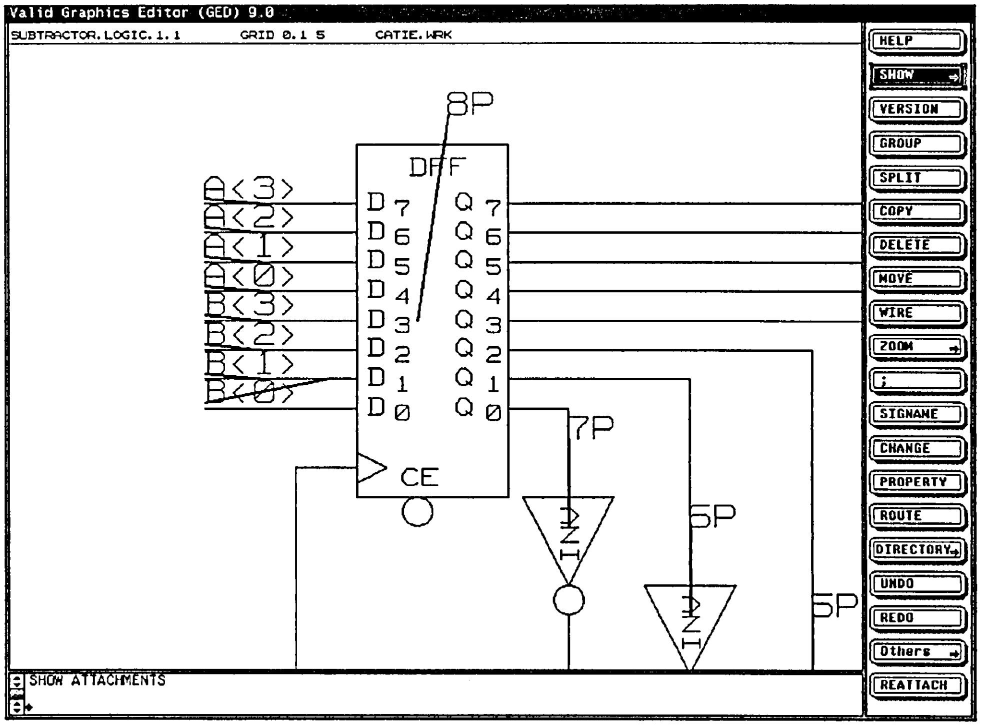
If your attachments are correct, use the refresh command (**zoom ;**) to refresh the screen.

When signals are close together (as are the inputs and outputs on the subtractor circuit), it takes a little practice to properly attach each signal name. The GED command **reattach** lets you fix incorrectly assigned signal names very easily. Follow these steps:

1 Type

```
show attach
```

to show the signal name assignments. Note that the signal name **B<0>** is incorrectly attached.



SHOW ATTACHMENTS

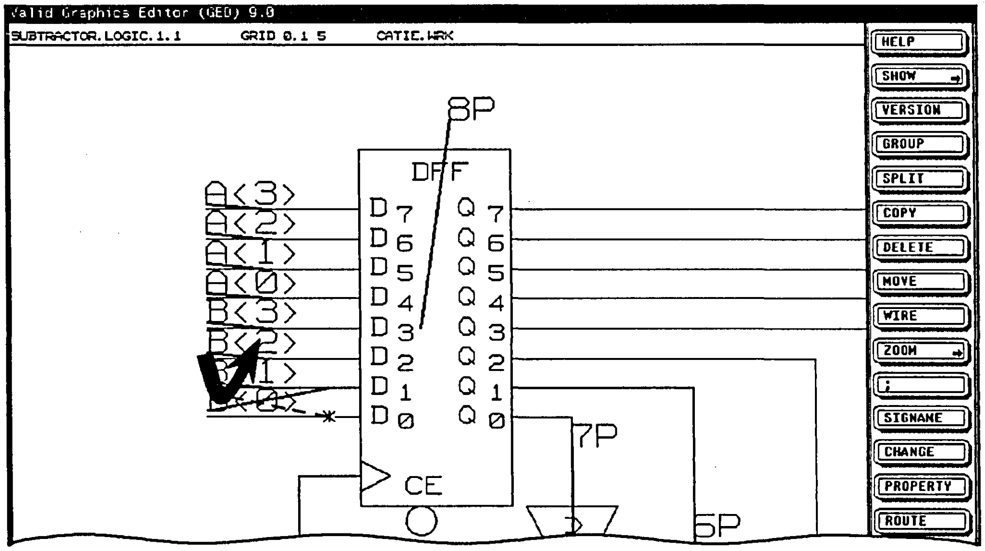
2 Type:

```
reattach
```

3 Move the cursor to the signal name that you want to reattach, and click the left button.

4 Move the cursor, and you will see a wire following the cursor (shown as a dotted line below).

5 Move to the wire that corresponds to the signal name and click again; an asterisk appears on the screen marking the new attachment.



Use the window refresh command (**zoom** ;) to remove the temporary lines.

Adding an Abbreviation to the Drawing

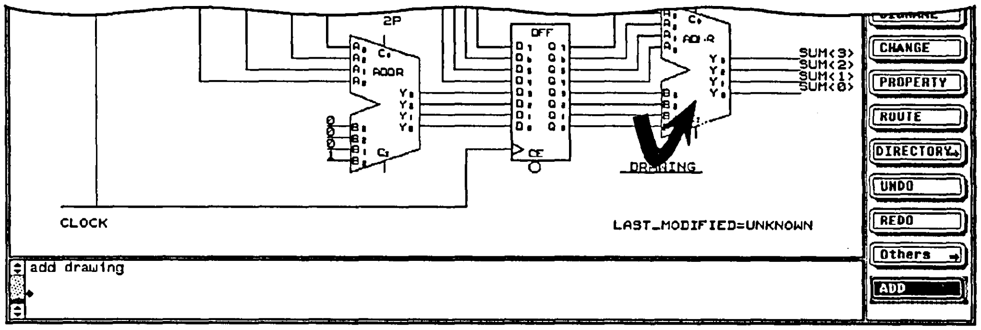
In order to uniquely identify each signal and library part on every drawing, the system must be able to identify each drawing by an abbreviation. If you do not specify an abbreviation for your drawing name, the system makes one for you. But since you later need to know the abbreviation, it is much better to specify it yourself and have it noted on the drawing.

The drawing body is stored in the Standard library.

To specify an abbreviation and have it be recognized as pertaining to the entire drawing, you need to add a special body called a *drawing body*. Enter this command:

```
add drawing
```

The word "drawing" appears under the cursor. This is the drawing body. Place it down in a corner of the drawing just as you place down any other body (this one just looks a little different).



```
add drawing
```

Notice the words "Last Modified" on the drawing body. Each time you write the drawing, GED consults the internal clock and time stamps your drawing here. If you include a drawing body on your drawing, you will always have a record of when the drawing was last modified.

Now you need to attach an abbreviation to this drawing. To do so you use the **property** command.

The Property Command

A property is a name and value that conveys information about your design to the analysis tools. The information represented by the properties in a drawing is interpreted by the Compiler and then passed on to the other programs.

The **property** command is used to attach properties to:

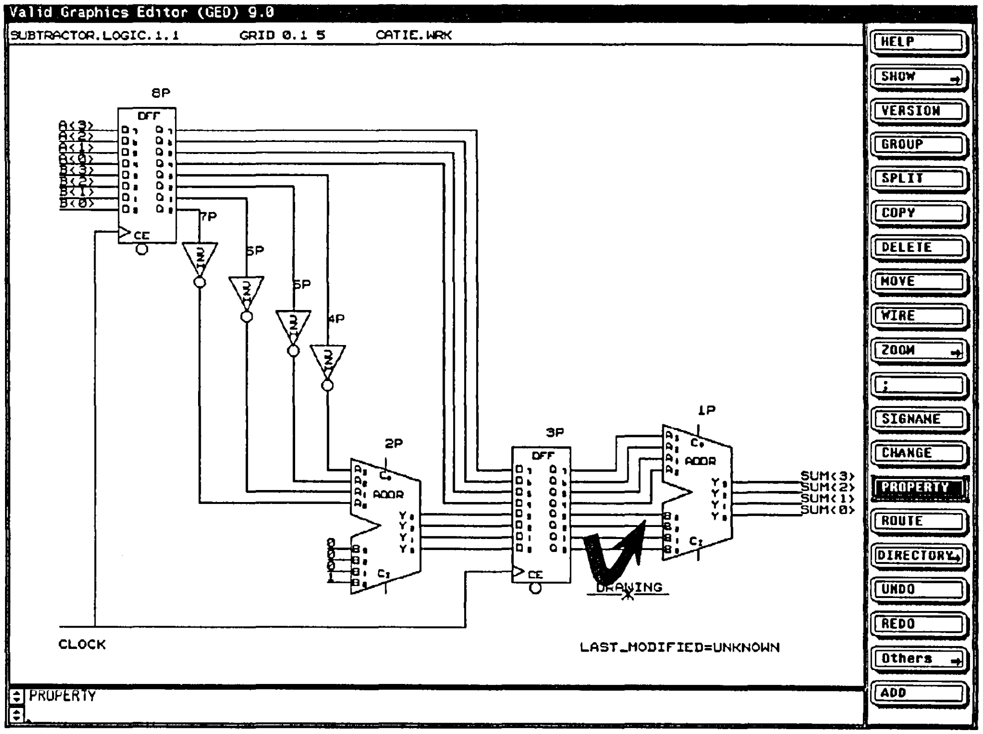
- Bodies
- Signal names
- Wires

To attach a property to an entire drawing, you must first add a drawing body and then attach the property to the drawing body.

There are lots of different properties. When you specify a property, you first give the name of the property (in this case, ABBREV), and then the value you want to assign to that property (in this case, SBT). The following procedure shows you how to add the abbrev property to the drawing body.

1 Select **property** from the menu.

2 Move the cursor to the word “Drawing” on the drawing body and click the left button. This identifies the object to which you want to attach a property.

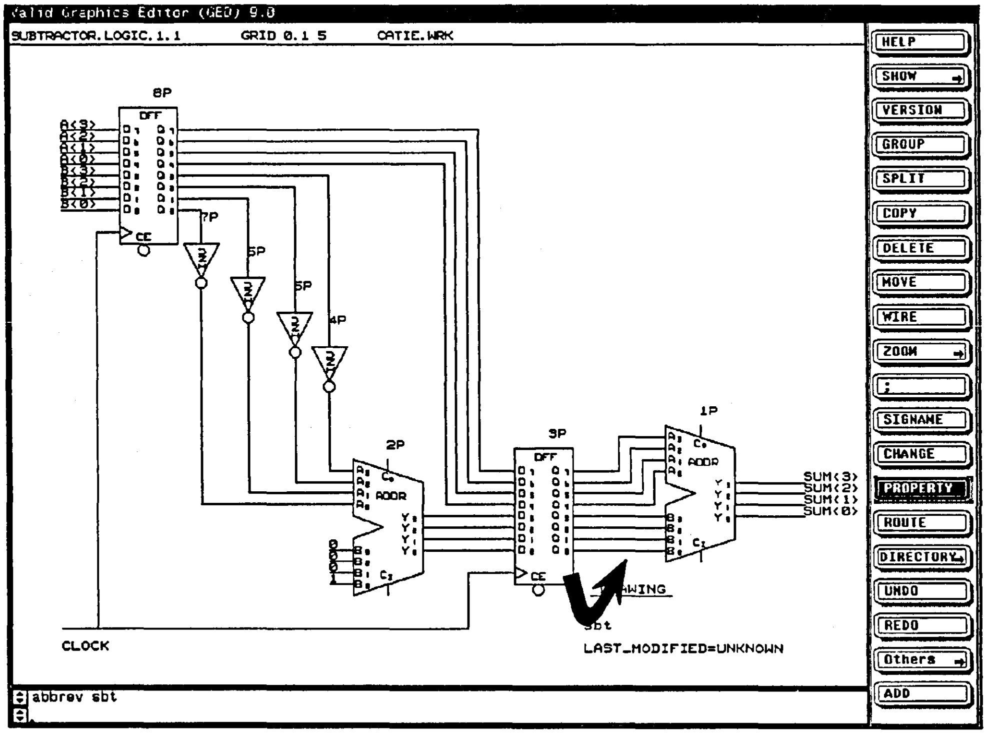


Be sure to leave a space after "abbrev." The **property** *command distinguishes property names from property values by this space.*

3 Type

abbrev sbt (Return)

4 Place the abbreviation down on the drawing above the words "Last Modified" by clicking the left button.



```
Valid Graphics Editor (GED) 9.0
SUBTRACTOR.LOGIC.1.1        GRID 0.1 5        CATIE.WRK
```

```
abbrev sbt
```

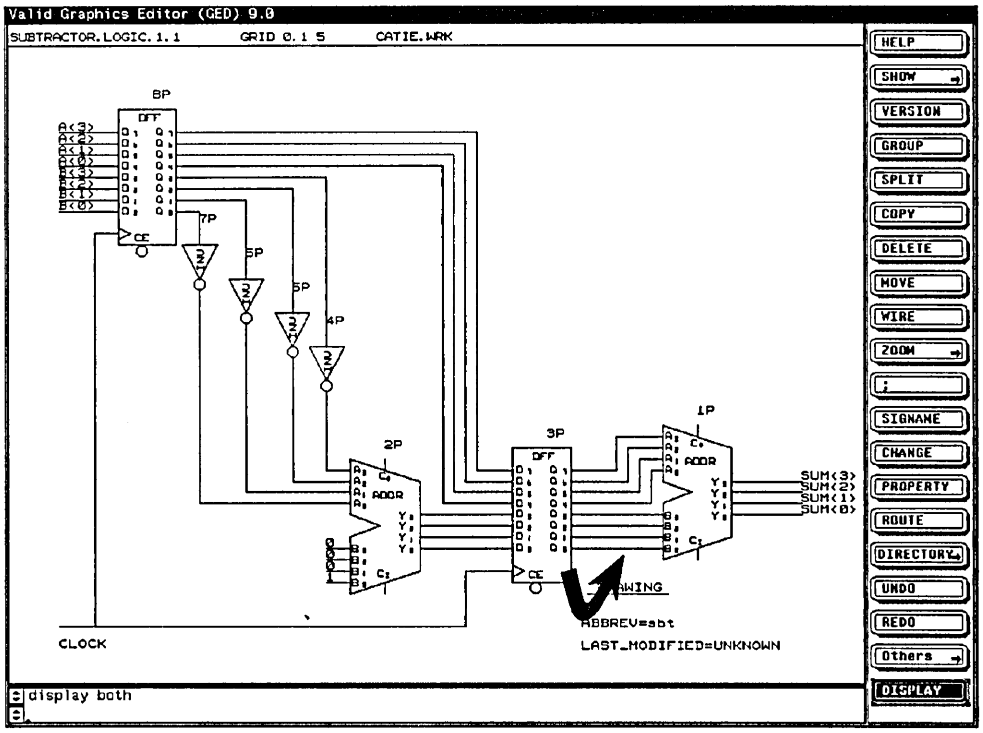
It is a good idea to have the abbreviation listed on your drawing so you know that SBT *is the abbreviation for the drawing name.*

5 Enter the command:

```
display both
```

then move the cursor to the abbreviation on the drawing and click the left button.

The **display both** command tells GED to display both a property name and the property value instead of just the property value.



```
display both
```

Adding a Title to the Drawing

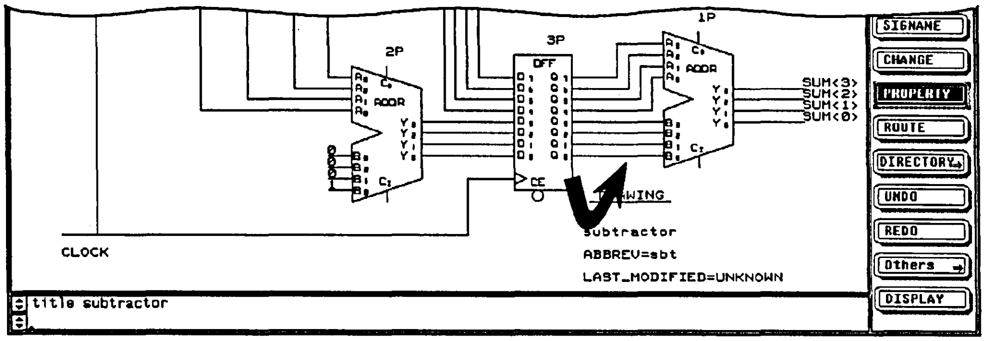
It's a good idea, although not required, to also add the property TITLE to the drawing body. This lets you record the name, or *title*, of the drawing on the drawing itself. When you are working with a drawing in GED, the title shows in the status line. But this title does not appear on a printed copy of the drawing unless you add the TITLE property to the drawing. When you add a TITLE property to a drawing, the title must exactly match the GED drawing name. Follow these steps:

1. Select **property** from the menu.
2. Move the cursor to the word "Drawing" on the drawing body and click the left button.
3. Type:

   `title subtractor` Return

   Be sure to leave a space after "title."

4. Place the title down on the drawing above the abbreviation.



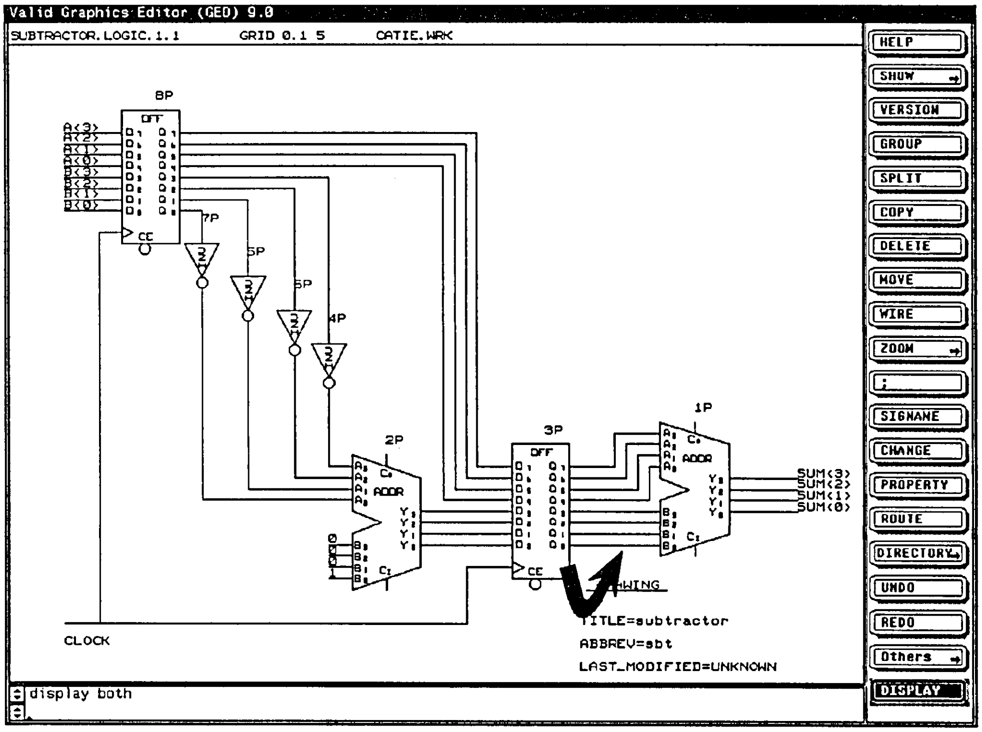
```
subtractor
ABBREV=sbt
LAST_MODIFIED=UNKNOWN
```

```
title subtractor
```

5 Enter the command:

```
display both
```

then move the cursor to the title on the drawing and click the left button. Now the drawing displays both the property name and the property value.



```
Valid Graphics Editor (GED) 9.0
SUBTRACTOR.LOGIC.1.1        GRID 0.1 5        CATIE.WRK
```

```
TITLE=subtractor
ABBREV=sbt
LAST_MODIFIED=UNKNOWN
```

```
display both
```

6 Save your drawing with the **write** command.

Checking a Drawing for Errors

Now that your drawing is complete, GED can check it for certain types of errors. If a wire is connected at only one end and has not been given a signal name, this is usually an error. If more than two wires intersect at one connection point, it is usually an error (as when you accidentally wire several pins of a library part together). It is important that you check for these kinds of errors before you leave GED and go on to use the Compiler and other programs. The other programs assume that your drawing is correct and complete.

If you use the Compiler, for example, on a drawing where many pins are wired together, it assumes that you did so intentionally. When the Compiler processes your drawing, it cannot detect the error condition in your wiring. Needless to say, it is better to check the drawing before you go on.

The **check** command is included automatically in the **write** command, but you have the option of using the **check** command by itself. To check a drawing for possible wiring errors, type:

```
check
```

GED checks quickly for errors. When it is finished, it displays short messages on the screen, usually one for each error. The last line reads "Done checking."

The Error Command

To get more precise information on each of the errors that were found, type:

```
error
```

GED draws an asterisk at the location of the first error and displays a message describing the error. Select **error** again to see the next error.

When your drawing is error-free, be sure to save it with the **write** command.

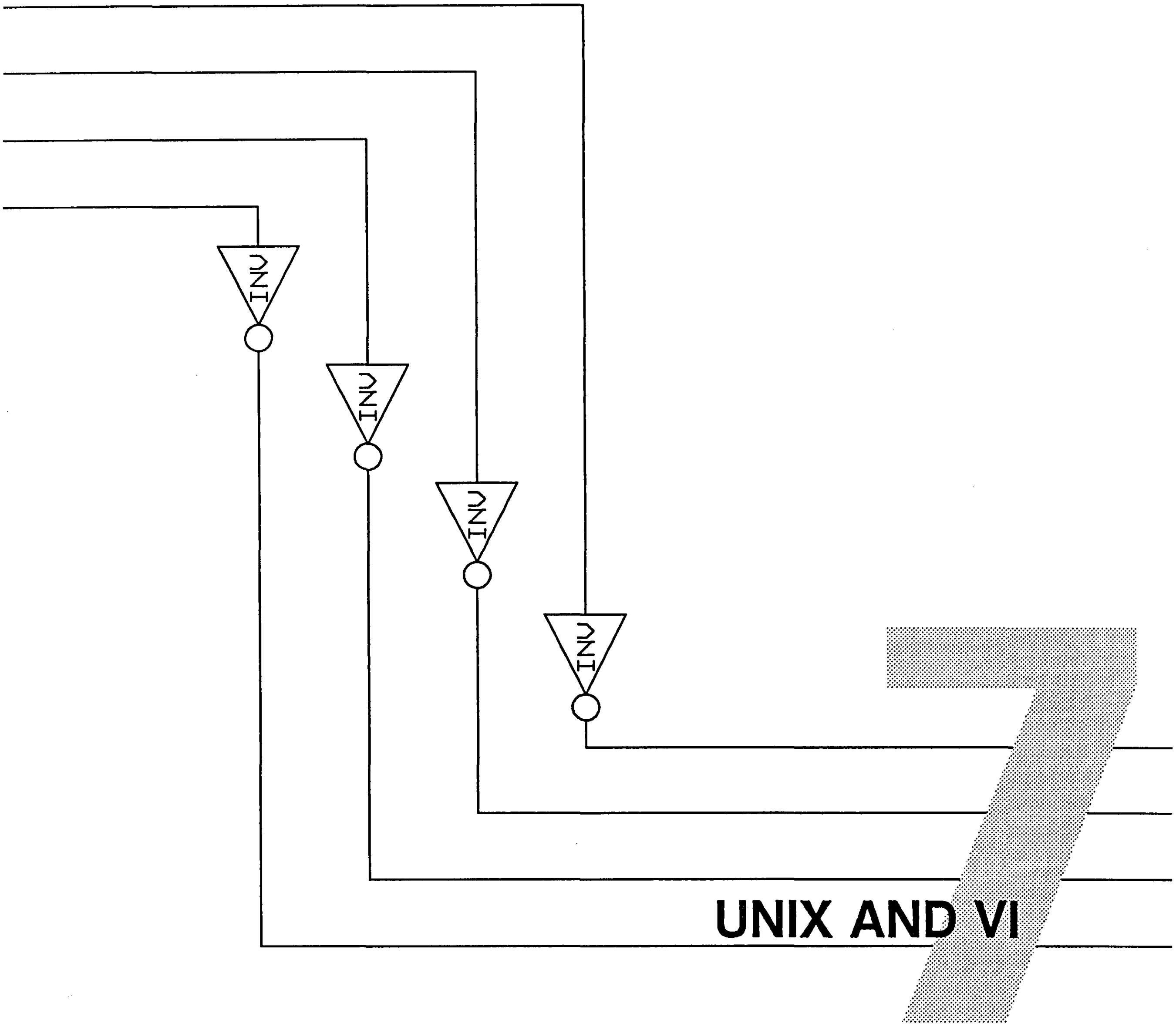
Congratulations!

See Appendix A for a handy list of common GED commands.

You have now learned over 30 of the most important GED commands and have sufficient proficiency to enter your own schematics.

Now you are ready to learn more about UNIX, the Compiler, and the Packager.

LESSON 7



UNIX AND VI

In Lesson 6, you successfully entered the subtractor schematic into your workstation using GED. You now have a picture of your schematic "on-line," but, more importantly, you have stored information about your circuit in a database that can be accessed by other programs.

This lesson introduces the creation and manipulation of windows on your workstation and these UNIX commands:

New Commands

file	**lpr**	**ls**
more	**vi**	

Using UNIX Windows

The Packager and other SCALD design tools are programs that you access from the UNIX prompt, just as you accessed GED. One way to access these programs is to exit from GED and use the original UNIX window for the other programs. A better way is to create a second window on the screen.

Windows are described in detail in the Sun manual Windows and Window Based Tools: Beginner's Guide.

When you create multiple windows, you can run programs in each window independently. It is like being able to log in through several terminals. The following procedures describe the basic window operations.

Creating a Second UNIX Window

Follow these steps to create a second UNIX window:

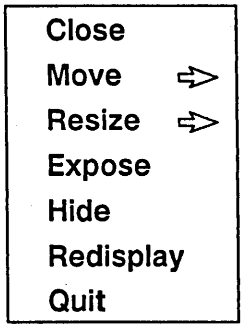
1. Move the cursor to the right and outside of the GED window.
2. Press and hold the right mouse button to display the Suntools menu.
3. While still holding the button, move the cursor down over ShellTool and then release the button. A second Shelltools – /bin/csh window is drawn over the GED window.

Moving the Second UNIX Window

When the second window is created, it fits within the GED window. To move back and forth between the GED and the UNIX window, the new window should be moved so that some portion of its top border extends beyond the GED window.

To move the second window:

1 Position the cursor in the border across the top of the newly created window and press and hold the right mouse button to display the frame menu.



2 While still holding the right mouse button, move the cursor down to "Move" and then release the button. When the button is released, the following message is displayed:

```
Press the left or middle mouse button near the
side or corner you wish to drag and hold the
button down while dragging the bounding box to
the location you want; then release the button.
To cancel, press the right mouse button now.
```

3 Move the cursor to the upper left corner of the window and press and hold the left (or middle) button.

4 While still holding the button, use the mouse to move the window to the lower right corner of the screen and then release the button.

Resizing the Second UNIX Window

For most SCALD applications, the size of the second window can be reduced to show more of the GED window.

To resize the second window:

1 Position the cursor in the border across the top of the window and press and hold the right mouse button to display the *frame* menu.

2 While still holding the right mouse button, move the cursor down to "Resize" and release the button. When the button is released, the following message is displayed:

```
Press the left or middle button near the side or
corner you wish to adjust and hold the button
down while adjusting the bounding box to the
shape you want; then release the button.  To
cancel, press the right mouse button now.
```

3 Move the cursor to the upper left corner of the window and press and hold the left (or middle) button.

4 While still holding the button, use the mouse to shrink the size of the window and then release the button.

Switching Between Windows

After the new UNIX window has been positioned and resized, move the cursor within the window area. The UNIX cursor changes from an outline to a filled box to indicate that the second window is *active,* or ready to accept a UNIX command.

To return to the GED window, simply move the cursor out of the UNIX window and into the GED window.

The *frame* menu can be used to place one window on top of the other so that the full window can be displayed. To position one window over another window:

1. Move the cursor to the top border of one of the windows and press and hold the right mouse button to display the *frame* menu.

2. If the window you have selected is to be placed on top of the other window, move the cursor to Expose and release the button. If the window you have selected is to be placed behind the other window, move the cursor to Hide and release the button.

When you Hide the GED window, the initial UNIX window is exposed over the GED window. Use the *frame* menu to again hide this window.

Closing a Window

The *frame* menu also allows you to close a window. When you close a window, the window disappears from the screen and an icon for the window is displayed across the top of the screen.

To close a window:

1 Move the cursor to the top border of the window to be closed and press and hold the right mouse button to display the *frame* menu.

2 While still holding the right button, move the cursor down to Close and then release the button. When the button is released, the window disappears and a /bin/csh icon appears at the top of the screen.

3 To reopen a closed window, simply move the cursor over the icon and press the left button.

Exiting a Second Window

To exit a second window, it is only necessary to type *exit* at the window's UNIX prompt. Do not use the Suntools menu to exit the second window as this operation terminates the GED session with an open drawing.

Using UNIX: ls, more

You already know one UNIX command. This is:

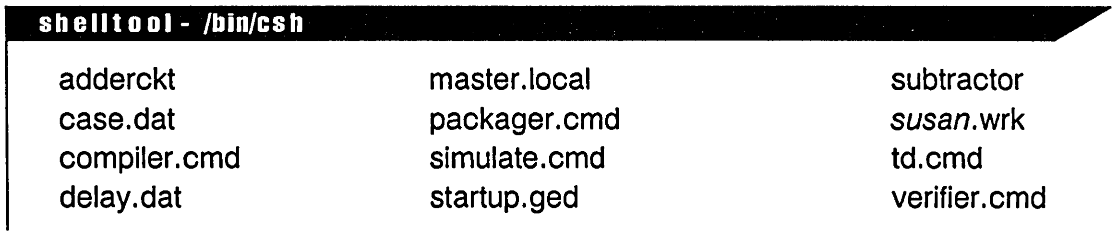
ged

Notice that the command *ged* is in all lower case letters, but inside GED, a lot of things appear on the screen in uppercase. This points out a difference between UNIX and GED that is important: UNIX is case-sensitive; GED is not. UNIX commands are always entered in lowercase, and system directory and file names are almost always in lowercase. In GED, commands and drawing names can be entered in either uppercase or lowercase.

There are two simple UNIX commands that let you look at your directories and files on the screen. The first one is *ls*. This is the list command. It lets you list the names of all the subdirectories and files in your current directory on the screen. Since the Sun workstation is set up so that all of the files you need are in your login directory, let's list them. At the UNIX prompt, type:

```
ls
```

Your screen will look like Figure 7-1.



```
adderckt        master.local     subtractor
case.dat        packager.cmd     susan.wrk
compiler.cmd    simulate.cmd     td.cmd
delay.dat       startup.ged      verifier.cmd
```

Figure 7-1. Contents of the Login Directory for User Susan

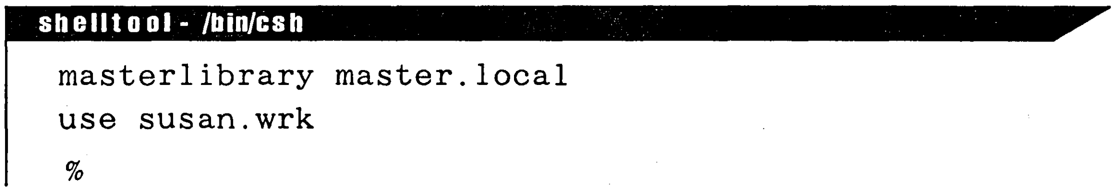
Each of these files and directories is briefly described in Appendix C. Remember, the *ls* command just lists the names of your files and directories, not their contents. To look at the contents of a file on the screen you need to know the exact name of the file you want to see. Use the *ls* command to find out the name of the file, and then use the *more* command to see the file.

The More Command

Let's look at the file named *startup.ged.* From the UNIX prompt, type:

```
more startup.ged
```

The *startup.ged* file appears on the screen, and at the bottom you see the UNIX prompt ready for your next command. The *startup.ged* file is shown in Figure 7–2.



```
masterlibrary master.local
use susan.wrk
%
```

Figure 7–2. Looking at Startup.ged with the *more* Command

To see one line of the file at a time, press Return.

If the file is too long to fit on one screen, the *more* command shows you one screenful of the file. To see the next screenful, press the space bar. To get out of a very long file without stepping all the way to the end of it, type:

q *means* quit.

```
q
```

Now you know how to get a list of your file and directory names and how to look at files on the screen. Try looking at some of the other files, just for practice, using *ls* and *more*.

The File Command

If you don't know whether something is a file or a directory, you can use the *file* command like this:

```
file adderckt
```

or

```
file startup.ged
```

The response to the first command is:

```
adderckt: directory
```

The response to the second command is:

```
startup.ged: ascii text
```

Try the file command on any of the files and directories listed in Figure 7–1.

Getting a Printed Copy of a File

When you work with the Compiler and the Packager later in this tutorial, you may want to get a printed copy of an output file or any other file. For some jobs, seeing things on the screen just isn't enough. Remember that the **hardcopy** command in GED is only for drawings. You can use it to get a copy of any drawing in GED, but you cannot use it to get a copy of a UNIX file. To print a UNIX file, you use the UNIX command *lpr*. The *lpr* command sends a copy of the file you specify to the printer. To get a printed copy of the *compiler.cmd* file, type this from the UNIX prompt:

```
lpr compiler.cmd
```

Remember, *lpr* is a UNIX command you use to get printed copies of text files. **hardcopy** is a GED command you use to get printed copies of drawings.

Command Files

To help you access the Compiler and the Packager, we provide you with a command file for each of these programs. The command file for the Compiler is named *compiler.cmd*. This is the file you just listed to the screen with the *more* command. The Packager command file is named *packager.cmd*. Both of these files contain a list of directives (or commands) that you use to instruct the program.

The command files in your login directory have default entries. Thus, to use these programs you will have to tailor the defaults to your own needs. This involves minor editing of the appropriate file. You may have to add a line or change part of a line already in the file.

Using the *Vi* Editor

Additional vi *commands are listed in Appendix B.*

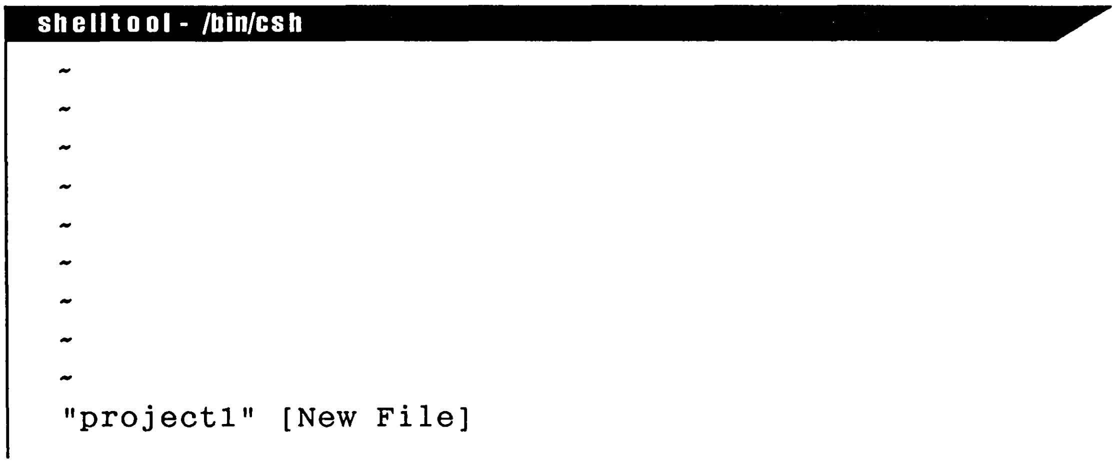
To do minor editing, you use an editor program called *vi*. *Vi* is a full screen editor with lots of capabilities. Here we will teach you just enough of the very basic commands to do your job.

The first thing you'll want to know about *vi* is how to call up the program, and how to get out of the program once you are in it. To edit a file you type:

filename *is the name of a file to edit.*

```
vi filename
```

If *filename* is an existing file, *vi* brings that file up on the screen and waits for your commands. If *filename* is not an existing file, *vi* assumes you want to create a new file, and the screen looks like Figure 7–3.



```
~
~
~
~
~
~
~
~
~
"project1" [New File]
```

Figure 7–3. Creating a New File with *Vi*

wq *stands for write and quit.*

When you are in *vi* and want to stop and exit, type

```
:wq
```

This command saves the current version of the file you are editing and returns you to the UNIX prompt.

To get out of *vi* without saving your current file, type:

```
:q!
```

This command exits you from the *vi* program without saving the current version of your file.

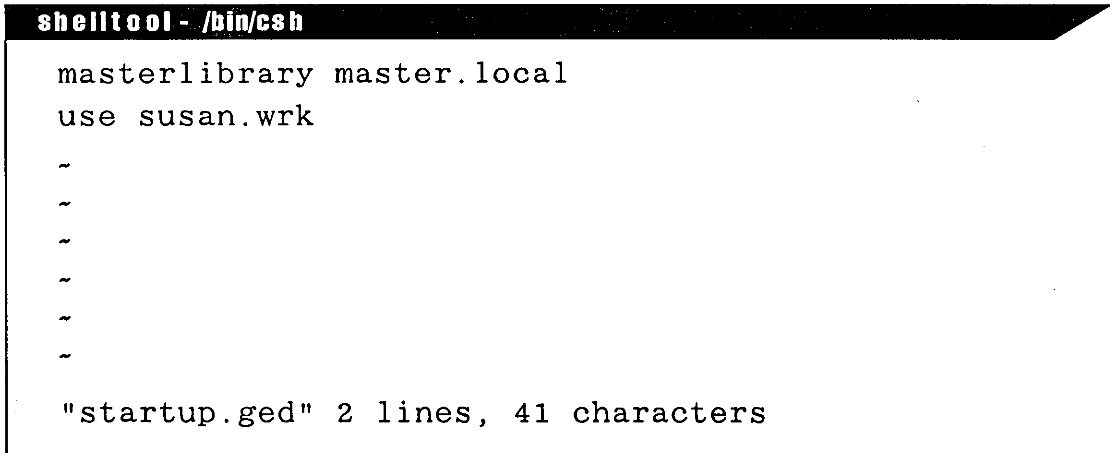
Editing Your Startup.ged File

The best way to learn how to use *vi* is by doing some editing. So we'll show you, step by step, how to edit your *startup.ged* file. Adding a few lines to your *startup.ged* file makes it easier for you to use GED. Follow these steps:

1 Type:

```
vi startup.ged
```

The screen looks like Figure 7-4.



```
shelltool - /bin/csh
masterlibrary master.local
use susan.wrk
~
~
~
~
~
~

"startup.ged" 2 lines, 41 characters
```

Figure 7-4. Default Startup.ged File

2 Move the cursor down one line by pressing `Return`.

3 Press **o** to tell *vi* that you want to add a new line to the file.

4 Type `library tutorial`.

5 Press Return.

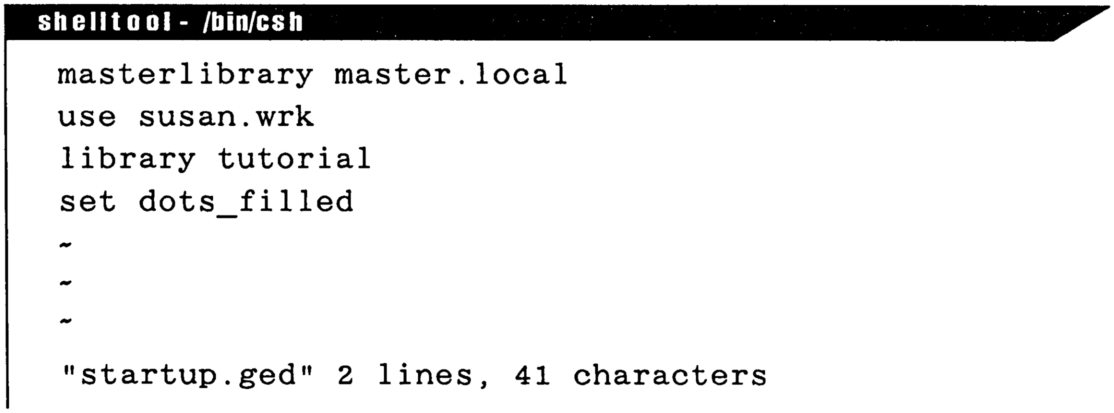
6 Type `set dots_filled`.

7 Press Return.

8 Press Esc to tell *vi* that you have finished inserting text.

9 Type `:wq`.

You have now added two lines to your *startup.ged* file and have saved the new version of your file. The new file should look like Figure 7–5. If you're not sure that it does, use the *more* command to look at the file. The next time you enter GED, the tutorial library is used and dots added with the **dot** command are filled.



```
masterlibrary master.local
use susan.wrk
library tutorial
set dots_filled
~
~
~
"startup.ged" 2 lines, 41 characters
```

Figure 7–5. New Startup.ged File

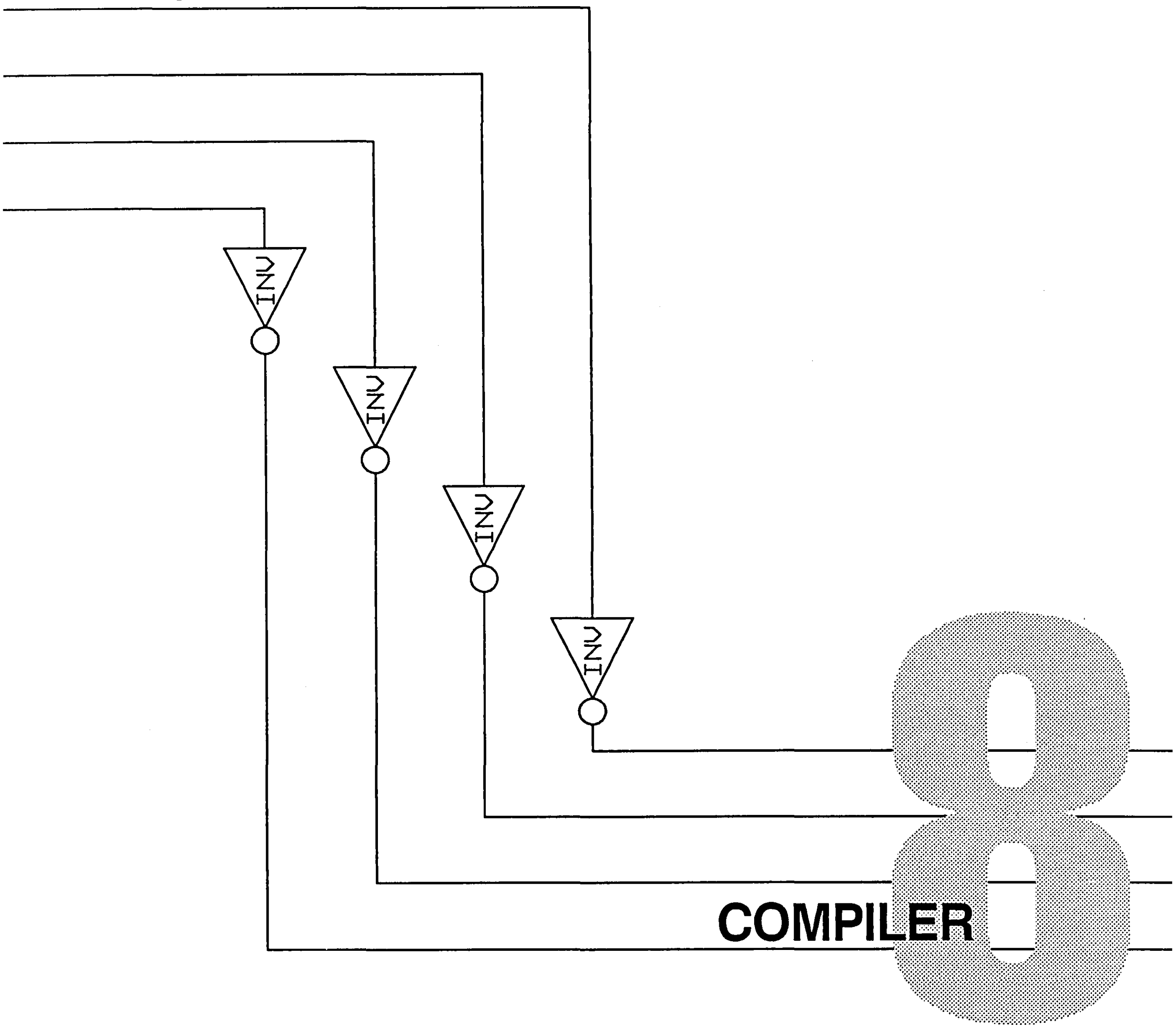
Basic *Vi* Commands

The following commands describe a number of the basic *vi* commands. Additional *vi* commands are described in Appendix B. For more information on *vi* and the other UNIX text editors, see the Sun documentation included with your workstation.

Table 7-1. Simple Vi Commands

Action	Command	Explanation
Adding	a	**add command:** add text after the cursor.
	i	**insert command:** add text before the cursor.
	Back Space	**erase** a character(s) you just typed while still in insert or add mode. This is how you fix typos as they occur.
	Esc	**stop adding or inserting text:** you are still in the editor and can now move to another part of the file.
Deleting	x	**delete character:** delete the character the cursor points to.
	dw	**delete word:** delete from cursor to end of the word.
	dd	**delete line:** delete the entire current line.
Moving	Return	**next line:** move cursor to the beginning of the next line of the file. If you are on the last line, you hear a beep.
	- (minus)	**previous line:** move cursor to the beginning of the previous line of the file. If you are on the first line, you hear a beep.
	Space Bar	**forward:** move the cursor one space forward.
	Back Space	**backward:** move the cursor one space backward.

LESSON 8



COMPILER

Now that you have completed your subtractor design and have checked for possible wiring errors, the next step is to compile and package your design. In this lesson you learn about the Compiler and how to prepare for compiling. Since the Packager calls the Compiler automatically, you actually compile your drawing along with the packaging in the next lesson. Before you start this lesson, be sure you have saved your current drawing (with the **write** command).

Resizing UNIX windows is explained in Lesson 7.

If you have not already done so, make a second window smaller than your GED window. The window should be tall enough for about 10 lines of text and about three quarters the width of the screen.

What Does the Compiler Do?

The Compiler prepares your design for another program and while doing so, checks for errors you may have made. For example, it checks for syntax errors in signal names and elsewhere, and it checks that the bit width of signals and pins is consistent.

The Compiler performs two jobs as it prepares your design for another program. It *expands* the information in your design (including any abbreviations) so that all of the signal names and library parts are fully defined, and it *selects* just those files that are needed by the particular program for which you want to prepare.

The Packager automatically calls the Compiler. When you package your circuit, the Compiler produces files that the Packager uses to prepare a design for a physical design system. These files contain the connectivity information for the parts in your design.

A timing model is a description of a part's timing behavior.

The Timing Verifier also automatically calls the Compiler. When you use the Timing Verifier, the Compiler produces the files needed to check the timing behavior of the circuit. These files contain a timing model for each part used in the design.

The Logic Simulator also calls the Compiler automatically. When you simulate your circuit, you produce files (with the Compiler) that the Logic Simulator uses to check the simulation behavior of the circuit. These files contain simulation models of the parts used in the design.

The entire scope of the Compiler is explained in the ValidCOMPILER Reference Manual.

This tutorial covers only compilation for the Packager. You can use the Compiler separately to specifically prepare a design for the Packager, the Timing Verifier, or the Logic Simulator. In this case, you specify the compilation purpose as *logic* (for the Packager), *sim* (for the Simulator), or *time* (for the Timing Verifier). You could even develop your own programs and use the Compiler to prepare your design for use with these products. The Compiler is generic and flexible.

Preparing for Compilation

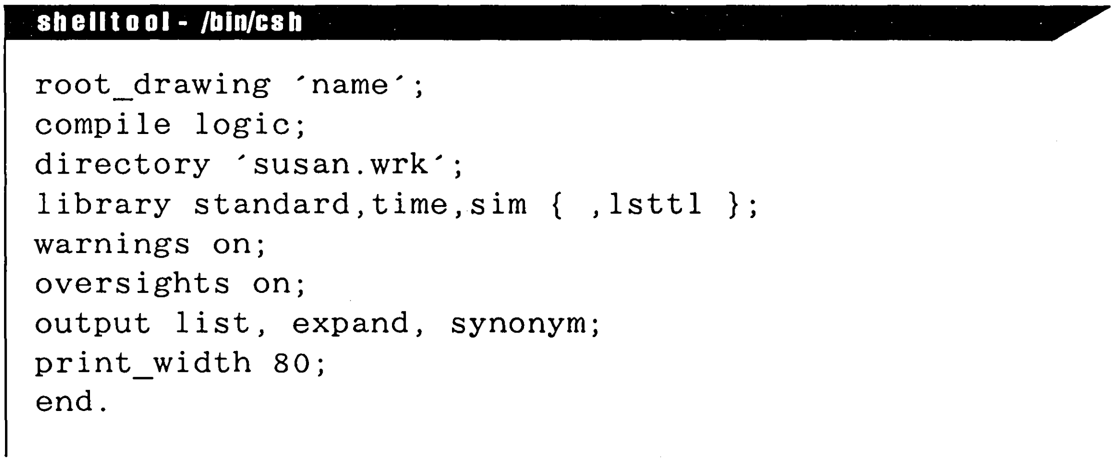
There are a number of different instructions that you might want to give the Compiler before having it called by the Packager. For this purpose, there is a file of *compiler directives* that you use to define the detailed instructions for the Compiler. A default, ready-made Compiler directives file is provided, but you can make minor adjustments to this file to suit your own needs. The directives file is named *compiler.cmd* and is created for you in your working directory when your account is established.

Compiler Directives File

To look at the Compiler directives file, from the UNIX prompt, type:

```
more compiler.cmd
```

Figure 8-1 shows the default directives file for user *susan*. If the entire listing does not appear on the screen, press the space bar to see the rest of the list. Check that your file matches the one displayed in Figure 8-1.



```
root_drawing 'name';
compile logic;
directory 'susan.wrk';
library standard,time,sim { ,lsttl };
warnings on;
oversights on;
output list, expand, synonym;
print_width 80;
end.
```

Figure 8–1. Default Compiler Directives File (compiler.cmd)

Each line in this file is a directive (an instruction) for the Compiler. There are many possible Compiler directives; each is discussed in detail in the *ValidCOMPILER Reference Manual.* This tutorial provides a brief description of the ones in this file. First, a few general comments.

Notice that each line of this file ends with a semicolon and that the last line of the file reads *end.* When you edit this file, be sure that the semicolons and the final line are not removed. Without these, the Compiler is not able to read this file and follow the instructions in it.

Notice also the curly braces in the fourth line of the file. They are used to surround a message or comment that is for your information, but that you don't want the Compiler (or any of the other programs) to act upon. In this case, *lsttl* is the name of a library, and could be inserted on this line; it reminds you that this is where additional libraries are entered.

Required Compiler Directives

Two of the directives shown are necessary for the Compiler to run and produce meaningful results. The others are optional. Most of these have default values that have been set for your convenience. The two necessary directives are:

DIRECTORY
LIBRARY

The DIRECTORY directive tells the Compiler what directories to search for the drawing it is going to compile. This directive is mandatory unless all drawings of interest are in libraries. If there is no DIRECTORY or LIBRARY directive given, or if the directory you enter is misspelled, you get a fatal error.

For more information on master.lib *and UNIX path names, see Lessons 13 and 14.*

The LIBRARY directive is where you list all of the libraries you used to make your design. Any library that is listed in the master library file on your system (*master.lib*) can be entered on this line. Do not remove the libraries Standard, Time, or Sim. They are needed for correct compilation of your designs. To use a library that is not in the *master.lib* file, enter its name (using the full UNIX pathname) on the directory line.

Other Compiler Directives

The remaining six directives in the default file are directives that you don't need to change. As you get more proficient with the Compiler, you may want to change them later. They are included in the file so that they can be changed simply by editing the line.

The ROOT_DRAWING directive tells the Compiler the name of the drawing you want to compile. Unless you are running the Compiler separately, you can ignore this line. Even when you run the Compiler as a separate program, you can also enter the drawing name directly in the command line, so this directive is not required. Any drawing name that you put in the Compiler directives file with the ROOT_DRAWING directive is overridden when you enter a name on the Compiler command line, or when the Packager, Timing Verifier, or Logic Simulator is used to call the Compiler.

The COMPILE directive tells the Compiler the type of compilation. Unless you are running the Compiler separately, you can ignore this line. Even when you run the Compiler as a separate program, you can also enter the compilation type in the command line, so this directive is not required. The default compilation type is *logic*. Any compilation type that you enter in the directives file with the COMPILE directive is overridden when you enter a type on the command line, or when the Packager, Timing Verifier, or Logic Simulator is used to call the Compiler.

The WARNINGS and OVERSIGHTS directives tell the Compiler that you want to know not only about the "errors" it finds, but also about the "warnings" and "oversights" it finds. Warnings and oversights are like errors, but less severe.

The output files are described in detail in Lesson 9.

The OUTPUT directive tells the Compiler what files you want it to produce. Unless you are running the Compiler separately, you can ignore this line.

The PRINT_WIDTH directive specifies how wide to print the output files. The optimum width is 80 if you are sending output to a printer/plotter with the *lpr* command.

There are many other Compiler directives for specific purposes. Each is described in the *ValidCOMPILER Reference Manual.*

Editing the Compiler Directives File

Before you use the Compiler on the subtractor design, you need to make a few changes to the Compiler directives file:

- Add the library "tutorial" to the LIBRARY directive.
- Add these two lines to your Compiler directives file (the reasons are provided later):

```
bubble_check off;
suppress 196;
```

Follow these steps to make these changes:

1 From the UNIX prompt, enter the command:

```
vi compiler.cmd
```

2 Press Return three times to move the cursor to the library directive (the fourth line of the file).

3 Press the space bar to move the cursor to the space after “sim.”

4 Press the **i** key (lower case ‘i’) to insert text at the cursor.

5 Type:

```
,tutorial
```

Tutorial is the name of the library you used to make the subtractor circuit.

6 Press Esc to stop inserting text.

7 Press Return until the cursor is positioned at the beginning of the next to the last line (print_width 80;).

8 Press the **o** key (lower case ‘o’) to open a new line below the print_width line. This command automatically places you in the text-insert mode.

9 Type these two lines:

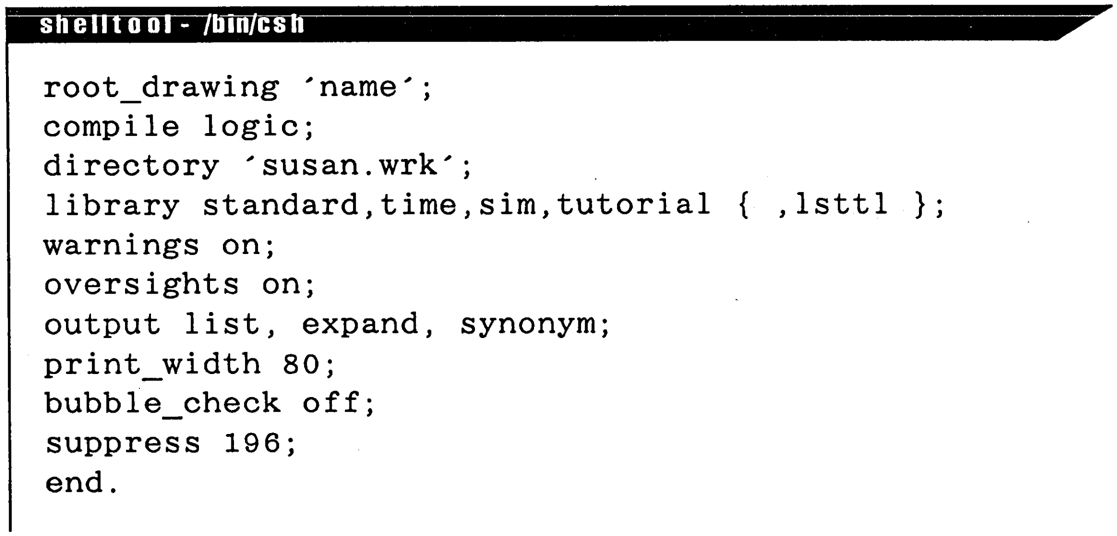
```
bubble_check off;
suppress 196;
```

Don't forget the underscore in "bubble_check" and the semicolons at the end of each line.

10 Press Esc to stop entering text.

11 Type :wq to save your changes and to exit from the editor.

Your *compiler.cmd* file now looks like Figure 8–2.


shelltool - /bin/csh

```
root_drawing 'name';
compile logic;
directory 'susan.wrk';
library standard,time,sim,tutorial { ,lsttl };
warnings on;
oversights on;
output list, expand, synonym;
print_width 80;
bubble_check off;
suppress 196;
end.
```

Figure 8–2. Edited Compiler Directives File

The use of bubble checking is beyond the scope of this tutorial. See Tutorial II, Using Your Validation Tools on a Sun Workstation *and the* ValidCOMPILER Reference Manual.

The BUBBLE_CHECK OFF directive tells the Compiler you are not using the bubble check feature. This feature checks that the assertion of a signal matches the bubble state of the pin, and produces an error message when they do not match. The purpose of this check is to allow the designer to verify that the signal is as intended. Bubble checking is an important feature that is very useful in the debugging of large designs.

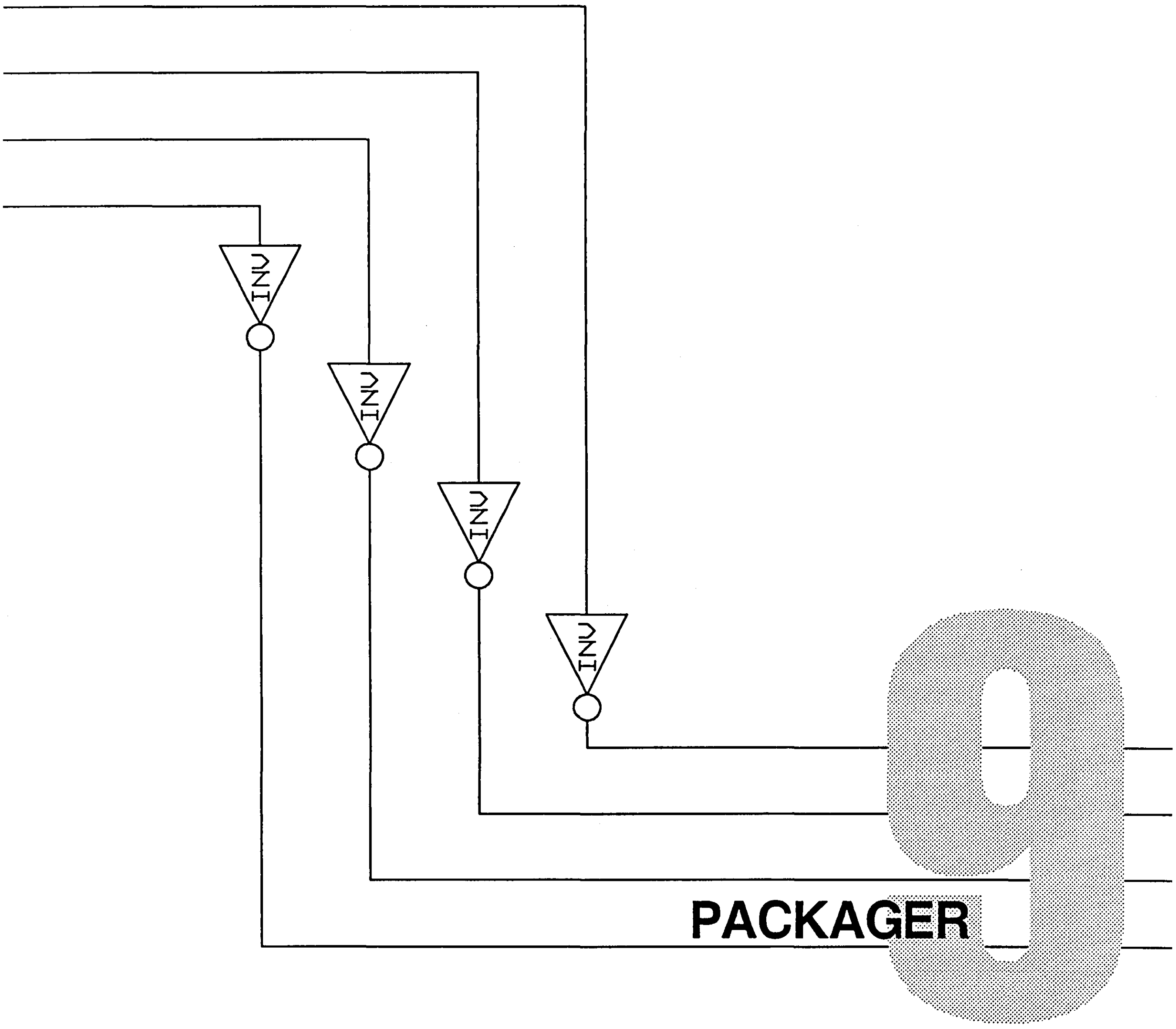
You cannot suppress error messages.

You use the SUPPRESS directive, followed by the number of the oversight or warning to be suppressed, to tell the Compiler not to report this particular warning.

Warning 196 tells you that the specified body (that is, the specified library component) has not been given the SIZE property, so the Compiler assumes that you want that part to have a size of 1. Size of 1 means that the inputs and outputs of the part are one bit wide. The Compiler assumes a size of 1 (SIZE = 1) unless you specify otherwise (for example, SIZE = 4). Parts having a size of 1 by default are almost always what you want, and so usually you would not be interested in receiving this message.

Your drawing is now ready to be compiled. When you package your drawing in Lesson 9, the Packager automatically calls the Compiler. You learn about Compiler output files and error messages in Lesson 9.

LESSON 9



PACKAGER

The Packager:

- Assigns the logical components in your drawing to physical packages
- Tests your design for loading violations and possible wiring errors
- Prepares your design for use by a physical design system

When you run the Packager, it automatically calls the Compiler, reads the Compiler output files, and expands this information into a set of output files that contain physical reference designators (U-numbers), section assignments, and pin numbers for the parts used in the design. Certain of these files are then used as the input to a physical interface program that reformats the Packager output for a specific physical design system.

The Packager assigns the logical parts in your design to sections of physical parts (according to a set of rules), and then describes the design in terms of these physical part assignments and node–to–net (pin–to–wire) connections. Some of these physical part assignments are likely to be changed later in the design cycle, either by you (the designer) or by the physical design system.

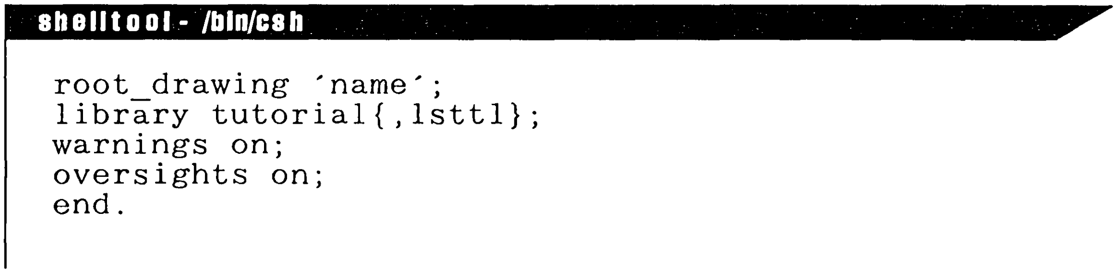
Keeping track of all of these section assignments along the way, and seeing that the GED drawing (the schematic) accurately reflects the assignments made by both the Packager and the physical design system, can be a demanding task. The Packager is set up to do all of this tedious work for you.

Using the Packager

Like the Compiler, the Packager has a command file, called *packager.cmd*, where you can enter directives to the Packager. Your account has a default packager directives file. To look at this file, type:

```
more packager.cmd
```

Your file should match the one shown in Figure 9-1.



```
root_drawing 'name';
library tutorial{,lsttl};
warnings on;
oversights on;
end.
```

Figure 9-1. The Packager Directives File

Before you use the Packager, you need to make some changes to this directives file.

The Root_Drawing Directive

The ROOT_DRAWING directive tells the Compiler the name of the drawing you want to compile. You can also enter the drawing name directly on the Packager command line, so this directive is not required. Any drawing name that you put in the Packager directives file with the ROOT_DRAWING directive is overridden when you enter a name on the command line. Without this directive, or if you don't enter a name on the command line, the Packager does not call the Compiler.

The Library Directive

For more information about this directive, see the ValidPACKAGER Reference Manual.

The LIBRARY directive is where you list the libraries you used to make your design. Your subtractor drawing uses components from the tutorial library. Therefore, "tutorial" must be included in the LIBRARY directive. Remember that the curly braces surround a message or comment that is for your information, but that you don't want the Packager to act upon. In this case, the *lsttl* library name reminds you that this is where additional libraries are entered.

For more information on master.lib, *see Lessons 13 and 14.*

Any library that is listed in the master library file on your system (*master.lib*) can be entered in this directive. The LIBRARY directive tells the Packager to read the individual physical library files (*chips_prt*) within a library directory. The Packager uses these files to obtain all of the physical information for each part in your design. If your design uses more than one library, you must enter the name of each library you use (except for the Standard, Time, Sim, or Phantom libraries, because the Packager doesn't use them).

Follow these steps to change the drawing name in the ROOT_DRAWING directive.

1 To edit the *packager.cmd* file, from the UNIX prompt enter the command:

```
vi packager.cmd
```

2 When the file appears on the screen, use the space bar to position the cursor over the first letter of the word `name` in the ROOT_DRAWING directive.

3 Type

```
cw
```

to change the word `name`. A dollar sign appears at the end of the word you are changing.

4 Type the name of your drawing:

```
subtractor
```

5 Press Esc to stop entering text.

6 Type

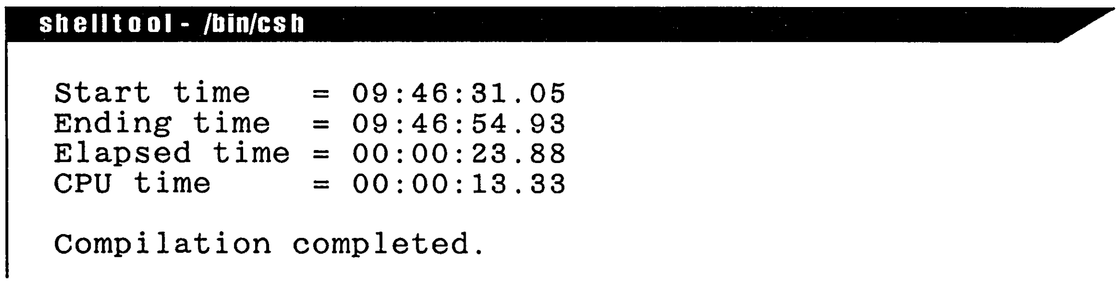
```
:wq
```

to save the new version of the file and to exit from the editor.

Now you are ready to run the Packager. At the UNIX prompt, enter the command:

```
package
```

A series of messages scrolls by in the message window to inform you of the progress of the Compiler program. A report from the Packager follows the Compiler messages. When the Compiler finishes, you see timing information and the Compiler tells you that the compilation is complete.



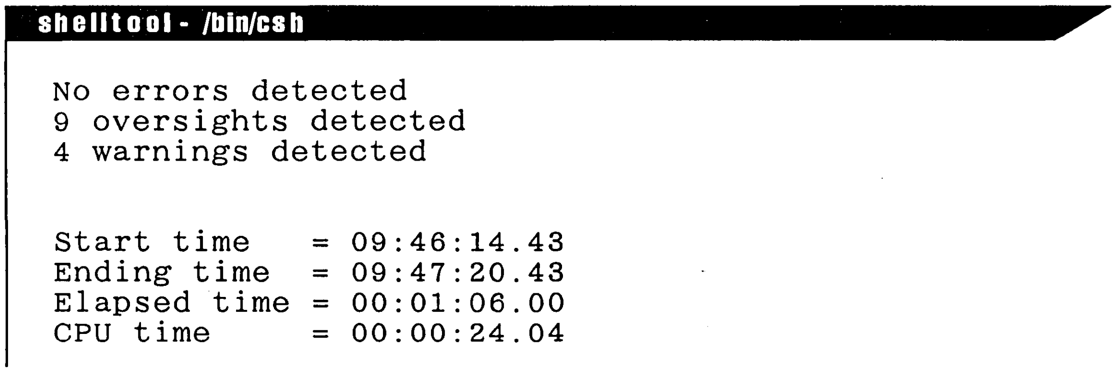
```
Start time    = 09:46:31.05
Ending time   = 09:46:54.93
Elapsed time  = 00:00:23.88
CPU time      = 00:00:13.33

Compilation completed.
```

Compilation errors are discussed on page 9–23.

If any compilation errors are reported, the Packager run is stopped and you must go back into GED and correct the errors in the drawing. If you followed this tutorial carefully, there are no Compiler errors. Congratulations!

The remaining messages report on the progress of the Packager. When the Packager finishes, you see a report of the errors found by the Packager. The last seven lines of the screen listing are shown below.



```
No errors detected
9 oversights detected
4 warnings detected


Start time    = 09:46:14.43
Ending time   = 09:47:20.43
Elapsed time  = 00:01:06.00
CPU time      = 00:00:24.04
```

Packager Listing File

Next, you want to see the listing file to take a closer look at what the Packager did and what errors, warnings, and oversights it found. To do so, type

```
more pstlst.dat
```

to list the file to the screen or type

```
lpr pstlst.dat
```

to get a hardcopy of the listing file.

All of the Packager output files have names that begin with the letters *pst*, which stand for *post processing*. You are now post processing your design for use with another system.

A Packager Listing File is shown in Figure 9–2. The listing file is full of useful information. The paragraph letters below correspond to the callout numbers on Figure 9–2.

A The first part of the listing file is the header. It tells you what version of the Packager you are using and the date and time you used it.

B The next part of the listing file is the directives list. It tells every directive that was in effect for this run of the Packager. Many of these directives have default settings. You don't need to know what each one of them does.

C The next part of the listing file contains the drawing name and the date/time that the drawing was compiled.

D The next several items on the listing file are process statements. As the Packager does its work, it reports at each stage. If errors are found during a particular operation, they are reported and appear in the listing file between the process statements. This way you know what the Packager was doing when it found the error.

E After the process statements, the listing file ends with a recap of the number of errors, oversights, and warnings that it found, and the elapsed time and CPU time.

```
     Valid Logic Systems, Inc.  ValidPACKAGER 1.0 Sun-B7  1Aug1988

A    Packager run on 1-January-1989 at 17:22:03.00

     *******************************
     * Starting to read directives *
     *******************************

     -----Directives-----

B    ANNOTATE BODY,
              PIN;
     DOCUMENT_ERRORS ON;
     FREE_GROUPING OFF;
     HARD_GROUPING ON;
     HARD_LOC_SEC ON;
     INCLUDE_IO_LIST OFF;
     LIBRARY 'TUTORIAL',
                  'LSTTL';
     NET_NAME_LENGTH 24;
     OUTPUT EXPANDEDNETLIST,
            EXPANDEDPARTLIST,
            LOGICALCHANGES,
            PHYSICAL CHANGES,
            BINDINGCHANGES,
            BACKANNOTATION,
            CHIPSFILE;
     OVERSIGHTS ON;
     PART_NAME_LENGTH 16;
     PART_TYPE_LENGTH 255;
     PRINT_PIN_LIST OFF;
     REPORT SPARES,
            PARTSUMMARY;
     ROOT_DRAWING 'SUBTRACTOR';
     SUPPRESS <none>;
     UNNAMED_CHANGES ON;
     UNNAMED_NETS OFF;
     USE_PIN_GROUP ON;
     USE_STATE_FILES ON;
     WARNINGS ON;

     *******************************
     * Starting to read directives *
     *******************************

     -----Design information-----

C    ROOT_DRAWING='SUBTRACTOR';
     TIME=' COMPILATION ON 1-Jan-1989 AT 17:22:03.00';

     ****************************************
D    * Starting to read library description *
     ****************************************
```

Figure 9-2. Packager Listing File

```
********************************
* Starting to read state file *
********************************

*************************************************
* Starting assignment of SIZE replicated parts *
*************************************************

*******************************
* Starting TIMES replication *
*******************************

****************************
* Starting to thread nets *
****************************

**************************************
* Starting to assign physical parts *
**************************************

  Using part bindings state file

******************************
* Starting to evaluate nets *
******************************

#1 OVERSIGHT(131):  No output on net
   Error in both states
   Log net name:  CLOCK

#1 WARNING(132):  No input on net
   Error in both states
   Log net name:  SUM<3>

#2 WARNING(132):  No input on net
   Error in both states
   Log net name:  SUM<2>

#3 WARNING(132):  No input on net
   Error in both states
   Log net name:  SUM<1>

#4 WARNING(132):  No input on net
   Error in both states
   Log net name:  SUM<0>

#2 OVERSIGHT(131):  No output on net
   Error in both states
   Log net name:  B<3>

#3 OVERSIGHT(131):  No output on net
   Error in both states
   Log net name:  B<2>
```

Figure 9-2. Packager Listing File (continued)

```
#4 OVERSIGHT(131):  No output on net
   Error in both states
   Log net name:  B<1>

#5 OVERSIGHT(131):  No output on net
   Error in both states
   Log net name:  B<0>

#6 OVERSIGHT(131):  No output on net
   Error in both states
   Log net name:  A<3>

#7 OVERSIGHT(131):  No output on net
   Error in both states
   Log net name:  A<2>

#8 OVERSIGHT(131):  No output on net
   Error in both states
   Log net name:  A<1>

#9 OVERSIGHT(131):  No output on net
   Error in both states
   Log net name:  A<0>

***************************
* Starting to check nodes *
***************************

******************************************
* Starting to assign physical part names *
******************************************

*****************************************
* Starting to assign physical net names *
*****************************************

  Using signal bindings state file

*******************************************
* Starting to assign physical group names *
*******************************************

*********************************
* Starting to perform pin swaps *
*********************************

  Using pin swap state file

***********************************
* Starting state files generation *
***********************************

***********************************
* Starting output list generation *
***********************************
```

Figure 9–2. Packager Listing File (continued)

```
Descriptions of the errors encountered during packaging
-------------------------------------------------------

OVERSIGHT #131:  No output on net

    An input pin is not connected and hence gets no drive current.
    Make sure that the net printed by the Packager is wired as
    intended.  Note that this message does not indicate an error
    if the pin is a primary input.  In this case, you may want to
    attach the NO_IO_CHECK property to the pin to suppress
    input/output checks.  This property has the form:
              NO_IO_CHECK = logic_state
              where logic_state may be 0, 1, or BOTH

    If I/O checks are not desired for either logic state, use the
    value BOTH.  To quiet this error message altogether, use the
    directive SUPPRESS 131.

WARNING #132:  No input on net

    An output pin has no subsequent blocks to drive.  Make sure
    that the net printed by the Packager is wired as intended.
    Note that this message does not indicate an error if the pin
    is a primary output.  In this case, you may want to attach the
    NO_IO_CHECK property to the pin to suppress input/output
    checks.  This property has the form:
              NO_IO_CHECK = logic_state
              where logic_state may be 0, 1, or BOTH

    If I/O checks are not desired for either logic state, use the
    value BOTH.  To quiet this error message altogether, use the
    directive SUPPRESS 132.

E   Packager run on 1-Jan-1989 at 17:22:03.00

Design name:
       SUBTRACTOR

Design compilation:
       COMPILATION ON 1-Jan-1989 AT 17:22:03.00

Library creation:
       COMPILATION ON THU JUN 6 11:07:06 1985


No errors detected
9 oversights detected
4 warnings detected


Start time   = 17:22:03.00
Ending time  = 17:22:33.00
Elapsed time = 00:00:30.00
CPU time     = 00:01:47.45
```

Figure 9-2. Packager Listing File (continued)

Packager Error Messages

Next, look at the errors the Packager found. They are listed in Figure 9–3. Four occurrences of Warning 132 and nine occurrences of Oversight 131 are listed. These two error messages tell you which nets on your drawing are connected to only one library part. Warning 132 tells you that your four outputs, SUM<3>, SUM<2>, SUM<1>, and SUM<0>, do not act as inputs for any library components. This message is correct, because these signals are your primary outputs.

Oversight 131 tells you that your nine inputs (CLOCK, A<3>, A<2>, A<1>, A<0>, B<3>, B<2>, B<1>, and B<0>) do not receive output from any library component. This message is also correct because these signals are your primary inputs.

Oversight 131 and Warning 132 are most useful for detecting wiring errors in a complex design, specifically dangling nets. To avoid receiving unwanted messages, you should indicate on your design each instance where a net (a wire) is intentionally connected to only one pin (one node). These are your primary inputs and outputs.

```
#1 OVERSIGHT(131):  No output on net
   Error in both states
   Log net name:  CLOCK

#1 WARNING(132):  No input on net
   Error in both states
   Log net name:  SUM<3>

#2 WARNING(132):  No input on net
   Error in both states
   Log net name:  SUM<2>

#3 WARNING(132):  No input on net
   Error in both states
   Log net name:  SUM<1>

#4 WARNING(132):  No input on net
   Error in both states
   Log net name:  SUM<0>

#2 OVERSIGHT(131):  No output on net
   Error in both states
   Log net name:  B<3>

#3 OVERSIGHT(131):  No output on net
   Error in both states
   Log net name:  B<2>

#4 OVERSIGHT(131):  No output on net
   Error in both states
   Log net name:  B<1>

#5 OVERSIGHT(131):  No output on net
   Error in both states
   Log net name:  B<0>

#6 OVERSIGHT(131):  No output on net
   Error in both states
   Log net name:  A<3>

#7 OVERSIGHT(131):  No output on net
   Error in both states
   Log net name:  A<2>

#8 OVERSIGHT(131):  No output on net
   Error in both states
   Log net name:  A<1>

#9 OVERSIGHT(131):  No output on net
   Error in both states
   Log net name:  A<0>
```

Figure 9–3. Packager Error Messages

Flagging Primary Inputs and Outputs

To flag nets intentionally connected to only one pin, go into GED and attach the NO_IO_CHECK property with the value BOTH to each net (wire) that is a primary input or output. This property tells the Packager not to perform input/output checking on this net. The value BOTH tells the Packager not to do this checking in both the High and Low states.

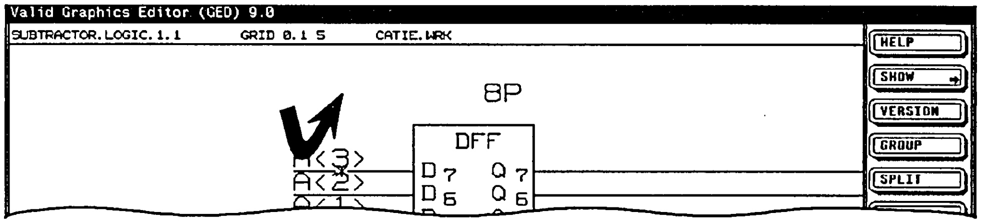
Follow this procedure to add the NO_IO_CHECK property to primary input and output nets:

1 Enter GED and edit subtractor.

2 Use the **zoom** command to zoom in on the top left corner of your drawing.

3 Select **property** from the GED menu.

4 Point to the wire labeled A<3> and click the left button. An asterisk appears on the wire.

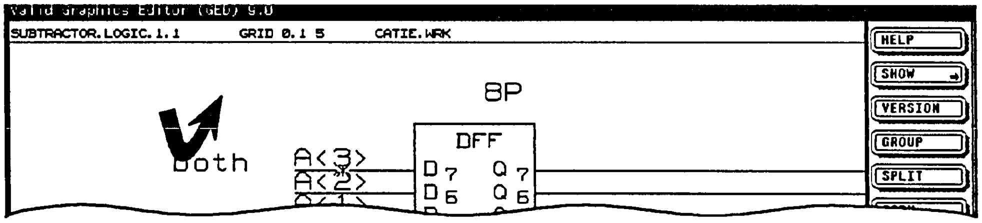


5 Type in this line exactly:

```
no_io_check both
```

The word "both" appears at the cursor on the screen. This is the property *value.* (NO_IO_CHECK is the property *name.*)

6 Move the word `both` to the left of signal A<3>, and click the left button to place it down on the screen.

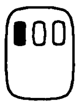


7 Now copy the property. This is a little different from copying bodies:

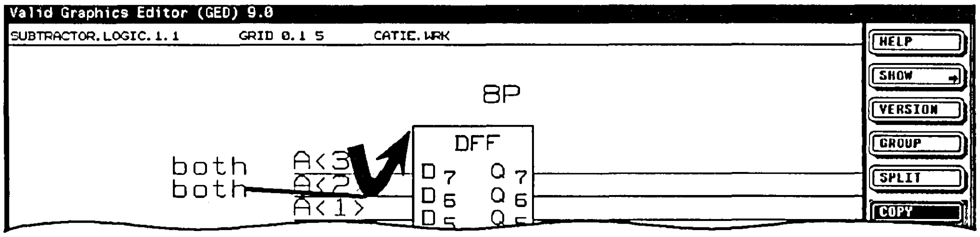
1) Select **copy** from the GED menu.

2) Point to the word `both` with the cursor, and click.

8 Move the new word "both" to just below the original word "both," and click to place it down. A line appears attached to the cursor.



9 Point to the wire labeled A<2> to indicate where you want this property attached, and click the left button. The line attached to the cursor disappears.



10 Repeat steps 7 through 9 to copy the new property to signals A<1>, A<0>, B<3>, B<2>, B<1>, and B<0>.

Now that you have attached a new property to each of these input signals, you should check that they are all correctly attached. Follow these steps:

1 Type `show attach` and watch lines appear from each word "both" to the signal next to it. The reason you place the word "both" off to the side is so that you can see these attachment lines. If you place the word "both" on the net, you can't see the lines drawn by the **show** command, and it would be difficult to verify correct attachment.

2 Use `zoom ;` to refresh the screen.

3 After verifying the correct attachment of the property, you can erase the word "both" from the screen. Type `display i`, and click over each word "both" in turn. They disappear one by one. This is the **display invisible** command. It suppresses the display of a property that you do not want to appear on your schematic.

4 Now add the property NO_IO_CHECK BOTH to the input signal CLOCK and to the four output signals SUM<3>, SUM<2>, SUM<1>, and SUM<0>. Use the above procedure to check the attachments and to erase the property value (both) from the screen.

5 Write your drawing to save it when you have finished.

Congratulations!

You have now successfully flagged your primary inputs and outputs. Now, move to a UNIX window and type `package` to run the Packager again. This time your Packager run should be free of errors, oversights, or warnings.

Tips on Adding Properties

- ✓ Point to the object to which you want to attach a property (for example, the wire labeled A<3>) *immediately* after selecting the property command. If you try to type in the property name and value before you point, GED does not understand. You get an error message.

- ✓ Don't point to a wire close to a body. If you do, GED might think you're pointing to a pin. A pin and a wire are two separate objects to GED although they are not visually distinct on the screen.

- ✓ When adding a property to a net (a wire), always point to the wire itself rather than the signal name (the text string).

- ✓ Be careful to type the property name and value correctly. GED distinguishes a property name and a property value from the SPACE you enter; that's why you use underscores to enter NO_IO_CHECK. If you type in NO IO as a property, GED thinks you mean a property named NO with a value of IO. No such property exists.

- ✓ The word "both" appears on the screen because, by default, GED displays only property values and not property names. To force GED to do otherwise on an individual basis, use the command **display both**, **display invisible**, or **display name**.

For more information on locating invisible properties, see the **find** *command in the* ValidGED Command Reference Manual.

After you use the **display invisible** command on a property, that property acts differently than other properties; an invisible property cannot be moved or deleted. Use **display visible** and point to the property to make it visible again.

✓ When you copy the property, be sure to point to the new signal name to indicate where you are attaching it. If you inadvertently point to the wire, you get this error message:

```
Can only attach properties
to SIGNAME properties
```

Just try the **copy** command again and point to the signal name this time.

Compiler Output: What the Compiler Produces

The Compiler produces two files and a directory when called by the Packager. The directory is called *xshadowx* and is used by the Compiler for organizing and storing intermediate results. The two files are *cmplst.dat* and *cmplog.dat*. These two files and the directory are listed in your directory along with your other default files. The *cmp* stands for Compiler and the extension *.dat* means that these are data files that you can read. Here is a description of each file:

cmplst.dat

Compiler listing file (*lst* stands for list). The Compiler creates this file *only* if it finds an error in a drawing.

This file is for your benefit and gives you a record of the errors and other conditions found by the Compiler. The file provides detailed information about each error, warning, and oversight found by the Compiler. It is important that you look at the listing file and correct errors, so that a subsequent Packager run can finish successfully. Warnings or oversights do not prevent the Packager from finishing.

See the ValidCOMPILER Reference Manual *for more information on* **comperr**.

The Compiler automatically creates the listing file if it finds errors in a drawing. If only warnings or oversights are found, the Compiler does not create the listing file. To create a listing file when only warnings or oversights are found, use the compiler error utility by typing `comperr`.

cmplog.dat	Compiler log file. If there are no errors found in a Compiler run, and therefore no *cmplst.dat* file is generated, the *cmplog.dat* file contains any warnings and oversights found by the Compiler. The *cmplog.dat* file also contains additional information on the Compiler run that is useful to Valid personnel in solving unusual customer problems. Certain error messages ask you to save this file and contact your Valid representative. You don't otherwise need this file.

Another important thing to know about the Compiler output files is that each time you run the Compiler, it sends output to the SAME FILES. This saves disk space and means that you do not constantly have to delete files. But you have to plan your work strategy to take advantage of this feature. If you have output from a previous compilation, the next time you run the Compiler, that output disappears because it is overwritten by the new output.

Compiler Errors

Since one of the most important functions of the Compiler is to provide information about errors found in your design, it is important for you to learn how to read the error documentation in the Compiler listing file (*cmplst.dat*). But first, you have to make an error! To learn about Compiler errors, go back into GED, edit your *subtractor* drawing, and change the signal name SUM<3> to SUM<3. This change produces a syntax error because the right bracket is missing. Follow these steps:

1 Enter GED by typing:

```
ged subtractor
```

2 **delete** the signal name SUM<3>.

3 Re-enter the signal name SUM<3, using the **signame** command.

4 **write** the drawing to save it.

Now move to a UNIX window and run the Packager again. Look at the listing file (*cmplst.dat*).

Compiler Listing File: cmplst.dat

To see the file on the screen and review what the Compiler did and the errors it reports, type:

```
more cmplst.dat
```

Or, if you want to make a hardcopy of the listing file, you can enter the command:

```
lpr cmplst.dat
```

The first thing to know about errors is that when you see a message like

```
WARNING 193
```

or

```
ERROR 202
```

it does not mean that the Compiler found 193 different warnings or 202 different errors. What it does mean is that warning condition 193 or error condition 202 was found. There are about 250 conditions on which the Compiler reports. Some are errors, some are warnings, and others are oversights. This is why the SUPPRESS directive takes just a number. The number alone is sufficient to identify the warning or oversight.

When the Compiler processes the pages of a design, warnings, oversights, or error messages for each individual page are stored in that page's error list file. Then the pages are automatically linked together by the linker utility and another error list file is created. The Comperr utility gathers these messages together into a single listing file (*cmplst.dat*) that you can look over.

Now look at the listing file (*cmplst.dat*) from this Packager run, shown in Figure 9–4.

```
A   Valid Logic Systems, Inc.  ValidCOMPERR 1.3 Sun-B6
    (C) Copyright 1982, 1987 Valid Logic Systems, Inc

    ---- <COMPERR> Retrieving ValidLINKER error messages ----


    Valid Logic Systems, Inc.  ValidLINKER 1.3 Sun-B5
    (C) Copyright 1984, 1987 Valid Logic Systems, Inc

    #1 ERROR (243):  Compiler errors in a drawing
       Drawing:  "SUBTRACTOR".LOGIC.1
       No parameters
       Page 1 contains 2 compiler errors

    1 Linker Error
    No Linker Oversights
    No Linker Warnings


    ---- <COMPERR> Retrieving ValidPAGECOMP error messages ----
                   (for pages with WARNINGs or worse)

    **********************************
    * Compiling SUBTRACTOR.LOGIC.1.1 *
    *          No parameters         *
    **********************************

    Compilation on Mon Aug 1 09:18:28 1988

    'SUM'<3
           ^
B   #1 ERROR(30):  Unexpected symbol in bit subscript
       Drawing:  "SUBTRACTOR".LOGIC.1
       Page=1
       Name of body=ADDR (path prop=5P)

    'SUM'<3
           ^
    #2 ERROR(11):  Expected >
       Drawing:  "SUBTRACTOR".LOGIC.1
       Page=1
       Name of body=ADDR (path prop=5P)

    2 errors detected
    No oversights detected
    No warnings detected

    ---- <COMPERR> End ValidCOMPERR ----
```

Figure 9-4. Compiler Listing File

A

The first part of the listing file is the header. It tells you what version of **comperr** you are using. It also lists linker information. Since you never have to deal directly with the linker, you don't have to be concerned with these messages at this time.

B

The second part contains descriptions of the errors found by the Compiler. For each error condition, there is a brief message (four or five lines). For example, look at the message:

The signal name is listed on the first line, and the ^ character on the second line marks the location of the error. For other kinds of errors, these lines do not appear.

```
SUM <3
      ^
#2 ERROR (11): Expected >
Drawing name=SUBTRACTOR.LOGIC.1.1
Page=1 Name of body=ADDR (path prop=5P)
```

The first two lines of the message report where the problem was detected. The error reported is a syntax error.

The next line tells you the error count (#2), meaning that this is the second error the Compiler found. Then the error identifier number (11), and the error description (Expected >) are listed.

The next line tells you on what drawing the error was found.

The final two lines tell you on what page of the drawing the error was found, and the body and path property to which that signal was attached. The path property of the ADDR may be a different number on your drawing.

All of the error messages follow this general format. Knowing what to look for makes understanding the error messages from other programs easier.

Correcting Compiler Errors

When you find errors in the Compiler listing file, it is important that you go back into GED, correct the errors, and then run the Packager again. You can choose not to correct warning and oversight conditions, but error conditions *must* be corrected before the Packager can use the Compiler output.

To correct the syntax error in your drawing, follow these steps:

1. Enter GED and edit the subtractor drawing.
2. Select **change** from the command menu and point to the signal name "SUM <3." The signal name appears at the top of the screen.
3. Press Control-E to move to the end of the signal name, and insert the missing bracket.
4. Press Return to exit the text editor.
5. Attach a NO_IO_CHECK property to the SUM<3> signal you just created. (The property was deleted when you deleted the signal name.) This will prevent a Packager warning when you run the packager again.
6. Save your corrected drawing with the **write** command.

For additional information on text editor commands, press Control-Q *while still in the text editor.*

Now move to a UNIX window and run the Packager again.

Common Compiler Errors & What to Do

The syntax error previously described is a common type of error reported by the Compiler. Here are some other common errors and how to fix them:

Error 191: Drawing not found in the Directories.

This error usually means that you misspelled the root drawing name, either in the directives file or on the command line. This error generates error 217 (fatal error report) and stops the Compiler run.

Error 206: Cannot open specified directory file.

This error usually means that you misspelled the name of your SCALD directory (*susan*.wrk) in the directives file. This error also generates error 217 (fatal error report) and stops the Compiler run.

Error 200: Pin name and signal widths do not match.

This error usually means that you either forgot to attach the SIZE property to a body that is attached to a bus or that you made a typing error in one of the signal names that makes up a bus, causing a syntax error. GED thinks the bus is not the same width as the body. Check the area in question on your GED drawing.

Data Flow Through the Packager

At the beginning of this lesson, you learned that the Packager not only tests your design for connectivity errors and prepares it for a physical design system, but that it also keeps track of your section and pin assignments. These additional duties mean that the Packager produces quite a few more output files than the Compiler does. If you list your default files now (*ls* command), you see a whole group of files have been added that all have the prefix *pst*. These are the output files from the Packager.

The Packager makes some output files to pass on to other programs and some files for you, the user, to consult. The Packager also makes another type of output file, for its own use, called a *state file*.

State Files

To keep track of physical assignments for you, the Packager makes a set of state files for its own use. These files contain notes about what the Packager did to your design the last time it was packaged. The Packager then uses these files as additional input the next time you run the program so it can repeat its previous assignments of parts and nets as much as possible. Since state files are created and maintained entirely by the Packager for its own use, you must never edit, change, or delete a state file.

First Run of the Packager

Figure 9–5 shows what happens the first time you run the Packager on a design. The Packager takes as its input the linked page expansion data from the compilation and the directives file *(packager.cmd)* and produces several groups of output files. Notice that the state files, the output files that are passed on to other programs, and the user files are all clearly marked on the figure.

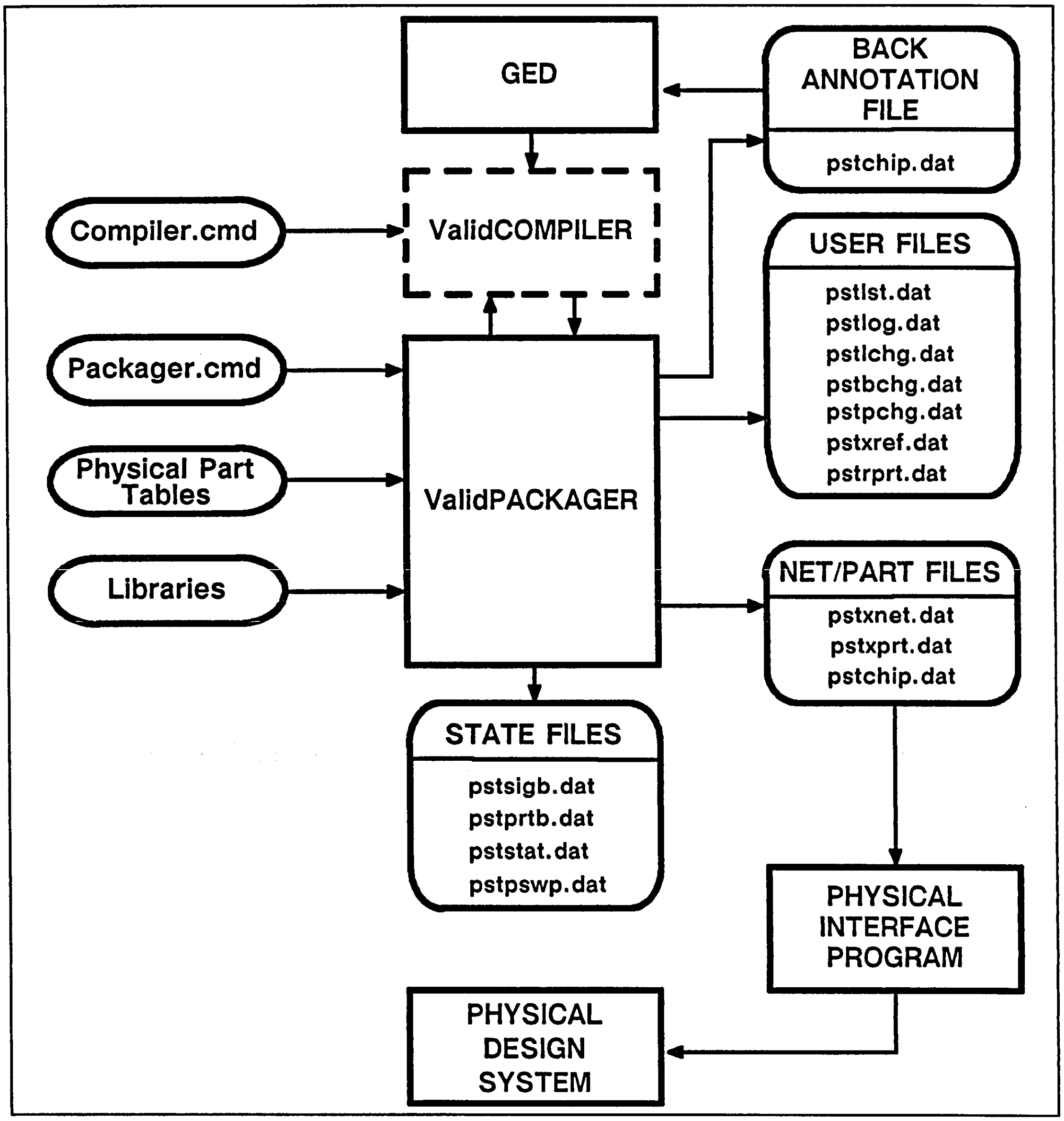


Figure 9–5. First Run of the Packager

The Packager produces more user files than the Compiler. In addition to the listing file *(pstlst.dat)* that you have already looked at to get information on your errors, there is a log file produced *(pstlog.dat)*. Like the Compiler log file, this file contains information that can be useful to Valid personnel in solving certain problems. The other user files are just to provide you with some additional information. They are described in more detail later.

You learn how to back annotate in Lesson 10.

Another important file shown in Figure 9-5 is the back annotation file *(pstback.dat)*. This file is used by GED to update (or back annotate) your schematic design.

Subsequent Packager Run

Now look at Figure 9-6. This figure shows what happens the next time you run the Packager. Your input is the same, but now the Packager reads the state files generated from the previous run. These files tell the Packager what it did the last time it packaged the design and allow the Packager to keep as many of its original assignments as possible.

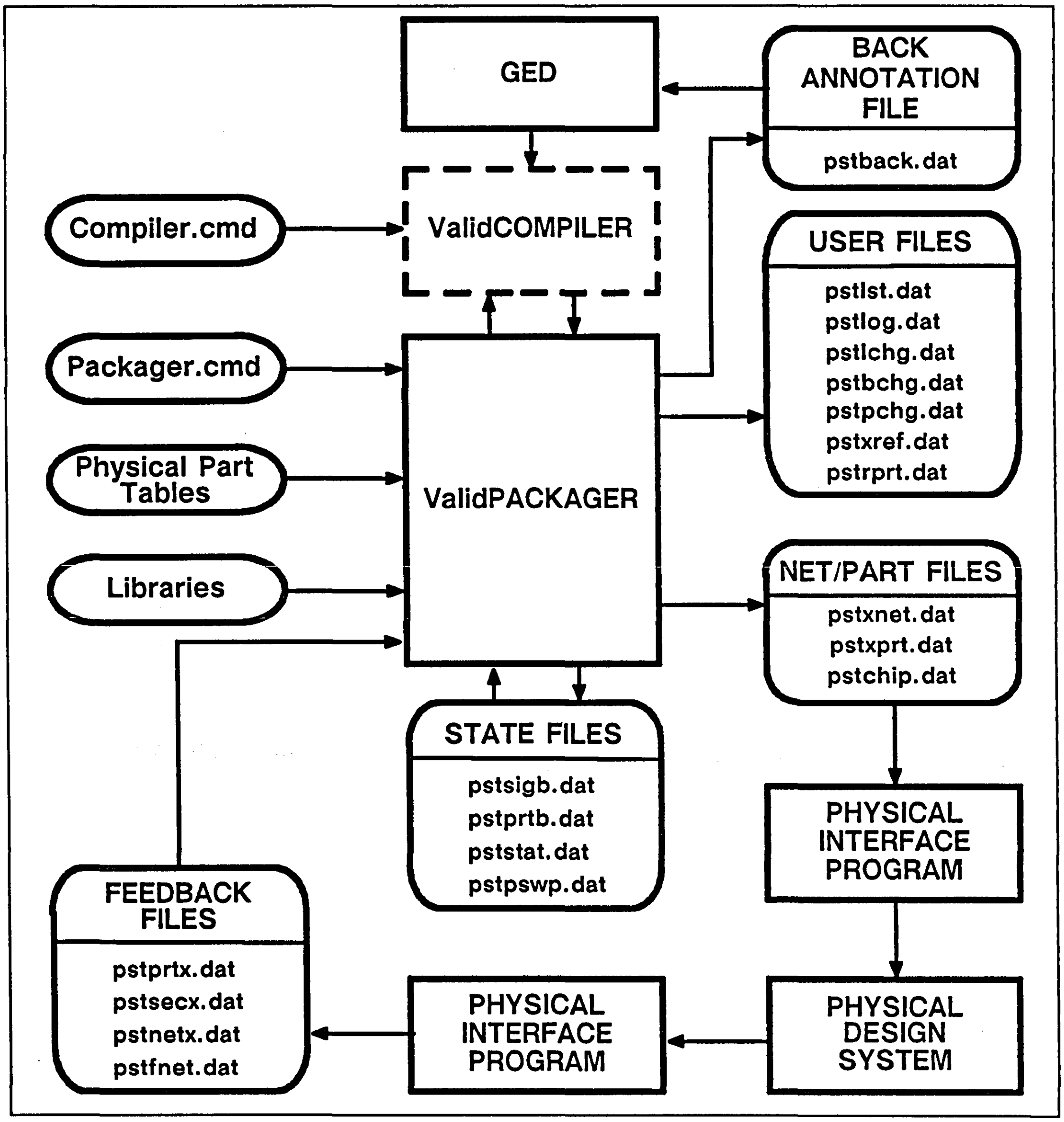


Figure 9–6. Subsequent Packager Run

Feedback Files

Feedback files are produced by the physical design system to tell the Packager what physical assignment changes were made in the design. As you learned in the beginning of this lesson, when you package your design in preparation for sending it to a physical design system, the Packager assigns section assignments and pin numbers on a best–guess basis. You then send the Packager output to a physical interface program, and the output from the interface program is sent to the physical design system.

One of the jobs of a physical design system is to optimize a design by shortening all the wire lengths as much as possible. Often this optimization is achieved by changing the section and pin assignments made by the Packager. As part of its output, the physical design system produces feedback files that the Packager uses to change its physical assignments to match those produced by the physical design system. Looking again at Figure 9–6, the feedback files from the physical design system must be run through the interface program before they can be used as input to the Packager. The feedback files save the designer from the tedious and error–prone task of making these changes manually.

Table 9–1 lists all of the Packager files by category and gives their file names. A description of each of the output and input files follows the table.

Table 9-1. Packager Files

Type	Name	System Name
First Run Input Files	Packager Directives File [Compiler Expansion Data]	packager.cmd
	Chips File	lsttl.prt,...etc.
Output Files: For use by another program	Expanded Net List	pstxnet.dat
	Expanded Parts List	pstxprt.dat
	Back Annotation File	pstback.dat
	New CHIPS File	pstchip.dat
	Pin List (for Flattener)	pstpin.dat
For use by user	Listing File	pstlst.dat
	Log File	pstlog.dat
	Reports	pstrprt.dat
	Logical Changes Summary	pstlchg.dat
	Physical Changes Summary	pstpchg.dat
	Binding Changes Summary	pstbchg.dat
	Cross Reference	pstxref.dat
For use by Packager in later runs (state files)	Logical Signal Name to Physical Net Name Bindings	pstsigb.dat
	Logical to Physical Part	pstprtb.dat
	State File	pststat.dat
	Pin Swap File	pstpswp.dat
Later Runs Input Files	(same as "First Run Input Files" and "state files" above plus "Feedback Files" below)	
Feedback Files	Physical Net Name Transformations	pstnetx.dat
	Physical Section Transformations	pstsecx.dat
	Feedback Net List	pstfnet.dat
	Physical Part Designator Transformations	pstprtx.dat
Other Input	Physical Part Tables	<named by user>

Output Files for Use by Another Program

- **Expanded Net List (pstxnet.dat)**

  This file lists each net on your design (alphabetically by signal name) and the nodes connected to it (by U–number). This file is used by any interface to go to a physical design system.

- **Expanded Parts List (pstxprt.dat)**

  This file lists each physical part in the design in order (by U–number) and tells what logical part (by P–number) is assigned to each section. This file is used by any interface to go to a physical design system.

- **Back Annotation File (pstback.dat)**

  This file lists the information in the expanded net list and the expanded parts list ordered by body name and P–number so that GED can write in U–numbers and pin numbers for each body on the drawing.

- **New CHIPS File (pstchip.dat)**

  This is the physical information extracted from the library chips files and physical part tables for each different library part used in your design. This output file is any interface to go to a physical design system. It is also passed on to subsequent programs for them to use.

- **Pin List File (pstpin.dat)**

  See the ValidPACKAGER Reference Manual *for details.*

  This file (not shown) is necessary to run the optional Schematic Flattener (ValidFLAT/Transcribe™). The pin list file is an extraction of the chips file and contains library information for the Schematic Flattener. This file is not generated by default. To generate it, use the PRINT_PIN_LIST directive in the *packager.cmd* file.

Output Files for Use by the User

- **Listing File (pstlst.dat)**

  This file provides process information and error messages for the user

- **Log File (pstlog.dat)**

  This file provides process information and other data for use by Valid personnel.

- **Reports File (pstrprt.dat)**

  This file lists the remaining spare sections (if any) and how many packages of each physical part your packaged design requires.

- **Logical Changes Summary (pstlchg.dat)**

  This file lists the logical parts that were added to the design or deleted from the design since the last run of the Packager.

- **Physical Changes Summary (pstpchg.dat)**

  This file lists all physical parts that were added to the design or deleted from the design during the last run of the Packager.

A binding is a mapping of a logical part to its allocated physical section.

- **Binding Changes Summary (pstbchg.dat)**

  This file lists all bindings that were added to the design or deleted from the design during the last run of the Packager.

- **Cross Reference (pstxref.dat)**

  This file lists, for cross reference purposes, signal names and the net names to which they correspond, and logical part names (library part names and P-numbers) and the physical assignment (U-number, section, and pin numbers) to which they correspond. This file is not generated by default.

See the ValidPACKAGER Reference Manual *for more information on the cross reference file.*

Output Files for Use in Later Packager Runs (State Files)

- **Logical Signal name to Physical Net Name Binding (pstsigb.dat)**

  This file contains the information in the expanded net list, in a somewhat different format.

- **Logical to Physical Part Designator Binding (pstprtb.dat)**

  This file contains the information in the expanded parts list, in a somewhat different format.

- **State File (pststat.dat)**

  This brief file time stamps the current Packager run and identifies the Compiler run used for input.

- **Pin Swap File (pstpswp.dat)**

  This file lists the pins swapped during the current Packager run. If no pins have been swapped, the file includes only a header and "end."

Input Files for Later Runs

- First Run Input Files (see above)
- State Files (see above)
- Feedback Files

  Physical Net Name Transformations (*pstnetx.dat*)
  Physical Part Designator Transformations (*pstprtx.dat*)
  Physical Section Transformations (*pstsecx.dat*)
  Feedback Net List (*pstfnet.dat*)

  The four feedback files are created by the physical design system and include all changes made to the design by the physical design system in a format that can be read by the Packager. They allow the Packager to repackage the design to conform exactly to the physical design. Feedback files may also be generated manually by the designer to make changes to the packaged design without returning to GED.

Physical Part Tables

Physical Part Tables are files used to modify library information. See the *ValidPACKAGER Reference Manual* for details.

Packager Checklist

Now that you know how the Packager works, just follow these steps next time you use the Packager. If you don't remember how to do some thing, or need a more extensive explanation, just look back through this lesson for the appropriate section.

1 Edit the Packager directives file:

```
vi packager.cmd
```

2 Check that the ROOT_DRAWING and LIBRARY directives are correct. You need one LIBRARY directive for each library used in your design.

3 Save the corrected Packager directives file.

4 Run the Packager:

```
package
```

Wait for completion - look for the UNIX prompt to appear.

5 If Compiler errors are reported look at the Compiler listing file to see errors reported:

```
more cmplst.dat
```

6 Go into GED and correct errors in the drawing.

7 Run the Packager again (make sure there are no Compilation errors and that the Packager run finishes).

8 Look at the Packager listing file to see errors reported:

```
more pstlst.dat
```

9 Go into GED and correct errors in the drawing.

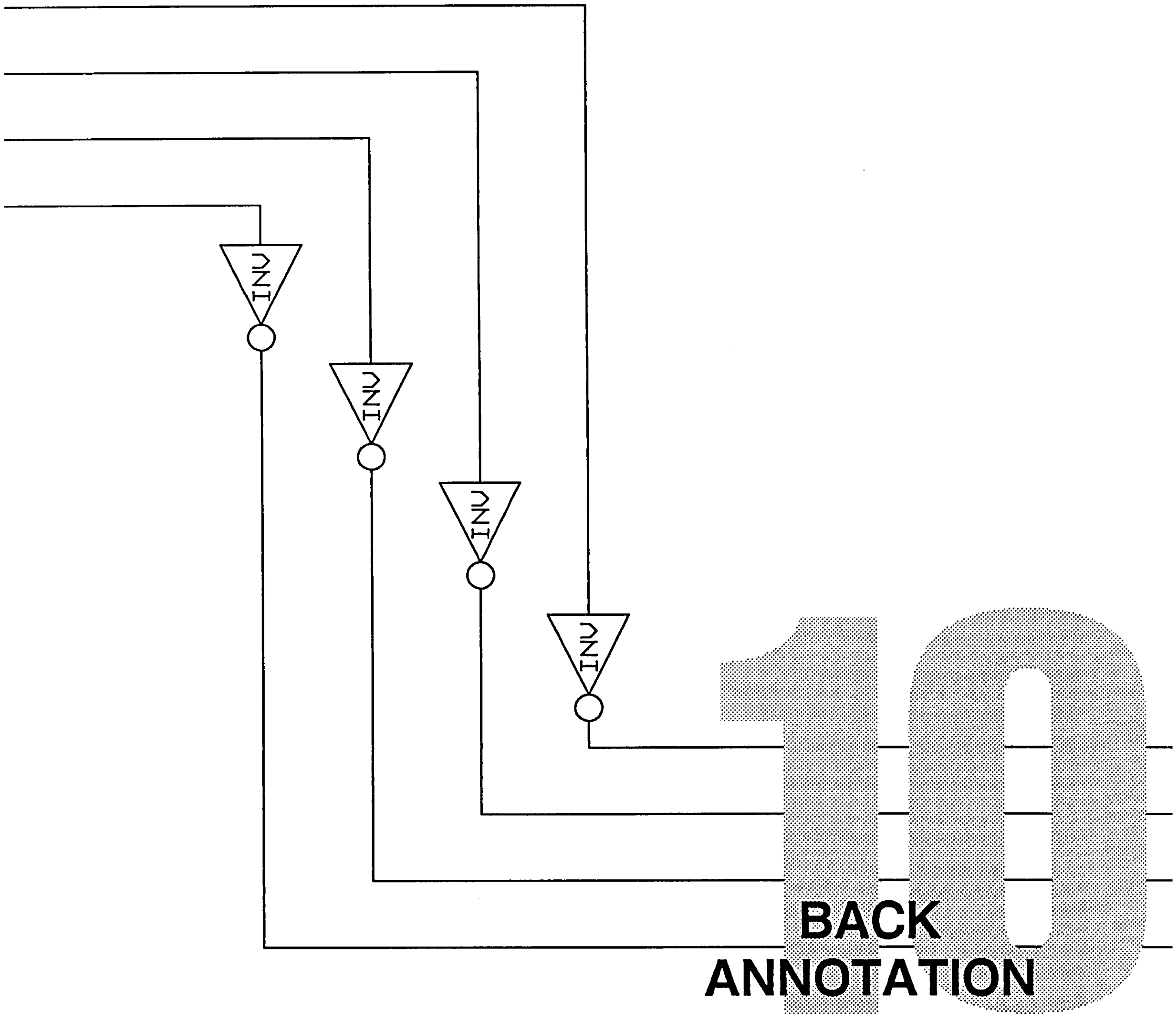
10 Run the Packager again.

11 Look at the listing file to verify that errors have been corrected:

```
more pstlst.dat
```

That's all. Now you are ready to back annotate your drawing or to run any of the physical interface programs. Congratulations!

LESSON 10



BACK ANNOTATION

After you run the Packager, your next task is to annotate your schematic (your GED drawing) with the physical reference designators (U-numbers) and to make adjustments to those physical assignments when necessary. This lesson shows you how to do these jobs.

New Commands

backannotate **section**

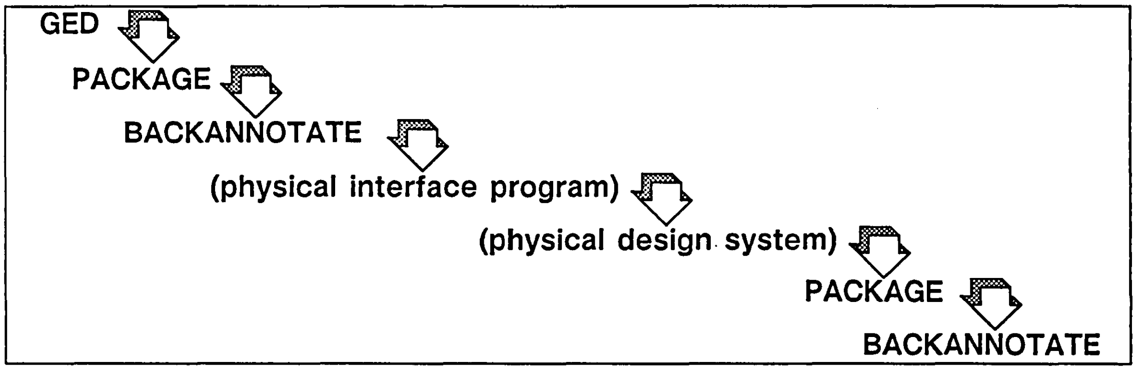
The Backannotate Command

One of the files output by the Packager is the back annotation file, called *pstback.dat*. This file contains the physical part assignments that the Packager made in a format that GED understands. You use this file and the GED command **backannotate** to automatically add the physical assignments made by the Packager to your drawing.

Back annotation saves you a lot of time and tedious work and ensures that your drawing accurately reflects the physical part assignments.

When to Back Annotate

There are two different times in the design cycle when it is important to back annotate your GED drawing. The first of these is after the first error-free run of the Packager, and the second is after the design has been sent to a physical design system (producing feedback files) and the design has been repackaged to reflect the changes in physical assignments. The typical order of design steps from the Packager through a physical design system, including back annotation, is shown below.



The first time you back annotate your design is after your first error-free run of the Packager. After you have corrected any errors found by the Packager and have run the Packager on the corrected design, you want to record the physical assignments (U-numbers and pin numbers) on your GED drawing for reference. Remember, however, that these physical assignments may be modified by your physical design system. That is why you have to back annotate again later.

After this first back annotation, you send the Packager output to a physical interface program to format it for a physical design system, and then to a physical design system. The physical design system creates feedback files that are used as input files to the Packager. You can also create your own feedback files to force the Packager to make certain assignments. You then run the Packager again, and it reassigns parts on the basis of the instructions in the feedback files. Now you want to update your GED drawing again so that it corresponds exactly with the physical design you have produced. This is the second time that you back annotate your design.

Using the Backannotate Command

The **backannotate** command is very easy to use:

1 At the UNIX prompt, type:

```
ged subtractor
```

2 After your drawing appears on the screen, give the command:

```
backannotate pstback.dat
```

You'll see GED automatically annotate the drawing. When it finishes, it automatically saves the updated version (writes it). You don't need to do anything except watch!

Figure 10–1 shows a portion of your updated drawing showing physical part designators (U-numbers) and pin numbers.

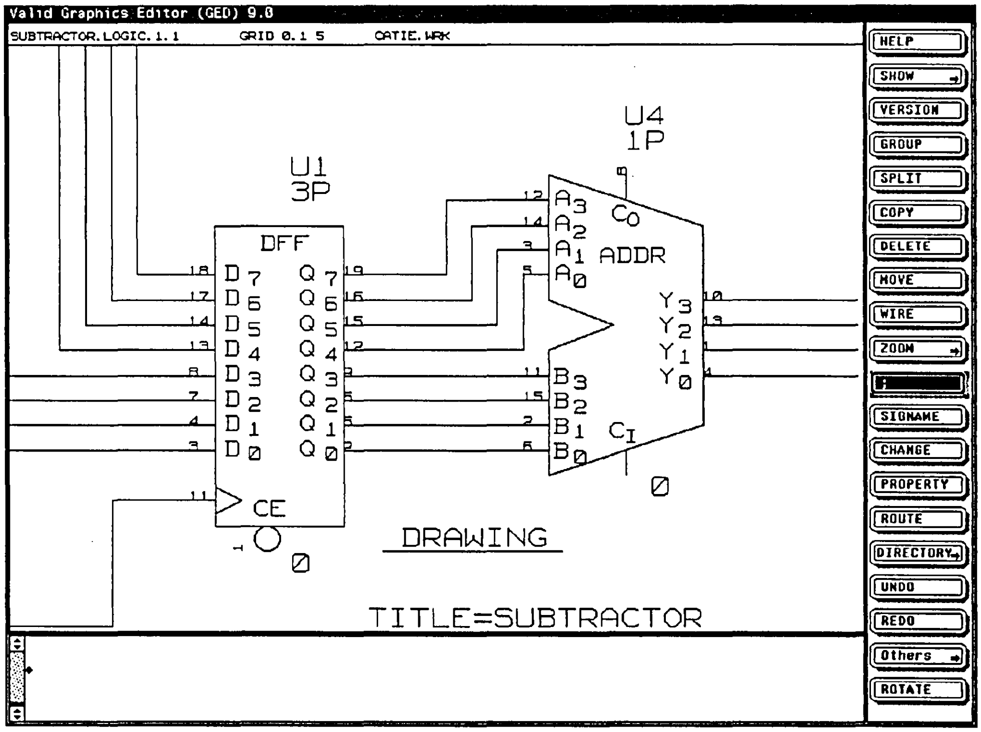


Figure 10–1. Back Annotation of Schematic

Moving Pin Assignments in GED

Signal names and pin numbers are considered objects.

When you back annotate a drawing, the system sometimes places pin numbers directly over signal names. If this happens, you can use the **split** command to move the pin assignments.

When you used the **split** command in Lesson 6, you clicked the left button to select a wire. To separate objects, you use the right-hand button to select the object you want. Follow these steps:

1. Use the **zoom** command to enlarge the area with overlapping signal names and pin numbers.
2. Select **split** from the menu.
3. Move the cursor to a group of overlapping objects and click the right-hand button.

   If you are closer to the pin itself (to the right of the objects), the cursor selects the pin number. If you are to the left of the objects, the cursor selects the signal name.

   If the cursor selects the wrong object, you can click the right-hand button to cycle through the overlapping objects. Each click selects an object to move away from the others; if a click selects the wrong object, just click again until you select the object you want.
4. When the signal name is selected, move it to the left of the pin number and click the *left* button to place it in the new location.

5 Move to the next group of overlapping objects, and repeat Steps 3 and 4 until all signal names and pin numbers have been separated.

6 **write** your drawing to save it.

Assigning Physical Locations and Sections in GED

After your drawing has been updated and the U–numbers are in place, you may want to make some physical reference assignments yourself rather than accepting the ones made by the Packager. There are three GED commands that you use to do this:

- The **property** command to attach a LOCATION property.
- The **section** command to assign sections.
- The **pinswap** command to swap the pin numbers belonging to the same pin group on a body.

The Location Property

When you know that you want a particular logical part on your design to be assigned to a particular physical location, you can make the assignment in GED. Go into GED, edit your drawing, and follow these steps:

1 Use the **zoom** command to enlarge the appropriate part of your drawing so that the U–numbers are clearly visible.

2 Select the **property** command from the menu.

3 Point to the origin of the body to which you want to assign a different physical reference designator (U-number).

4 Type:

```
location
```

followed by a space and the new U-number (for example, "`U25`.")

5 Place the new U-number down on the drawing. Notice that the old U-number disappears. You don't have to delete it.

6 **write** the drawing and package it.

Tips on the Location Property

✓ You can attach a LOCATION property to a body either before or after you package the design. Although the LOCATION property is introduced here, you may want to attach this property as you first enter the design in GED.

✓ When you attach a LOCATION property, the property value is not restricted to U-numbers. The property value may be any alphanumeric string. For example, you can call a resistor "R5" or a capacitor "C3."

The Section Command

You use the **section** command to select which section of a physical part is assigned to a particular logical part. To change the section assignment of one of the inverters in the subtractor drawing, edit that drawing and follow these steps:

1 Use the **zoom** command to enlarge the appropriate part of your drawing so that the pin numbers are clearly visible.

2 Type:

```
section
```

to select the **section** command.

3 Point to one of the pins that you want to assign to a different section, and click the left button.

Each click selects a different section of the physical part. Watch the pin numbers on the entire body change accordingly.

4 When the pin numbers correspond to the desired section, you are done. You can select another body to section, or select another GED command.

5 **write** your drawing and package it.

Tips on the Section Command

- ✓ You can use the **section** command on a body either before or after you package the design. Although the **section** command is introduced here, you may want to use this command as you first enter the design in GED.

- ✓ An alternate way to use the **section** command is to type in the desired pin number and then point to a pin. This can save time cycling through the various sections when you know exactly which one you want.

- ✓ When changing section assignments after back annotation, you do not need to reassign all sections of a given part. Just assign the sections you want to force and leave the others. Your back annotated schematic may temporarily carry duplicate section numbers. This is not a problem. When you save your drawing and send it to the Packager, the Packager reassigns the remaining sections for you.

See the ValidGED User's Manual *for details on manual section assignment.*

- ✓ When you use the LOCATION property and the **section** command, remember that the Packager and the physical design system do not override these assignments. To change a manual assignment, you can either replace the LOCATION property or use the **section** command again. To delete a manual assignment, you must delete the LOCATION and SEC properties.

The Pinswap Command

The **pinswap** command swaps the pin numbers on a body that are defined to be in the same pin group. You use this command only after using the **section** command. Pin swapping can occur only between pins that have been defined in the library as swappable. For example, 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, go into GED, edit your drawing, and follow these steps:

1 Type:

```
pinswap
```

2 Point to the two pins to be swapped,

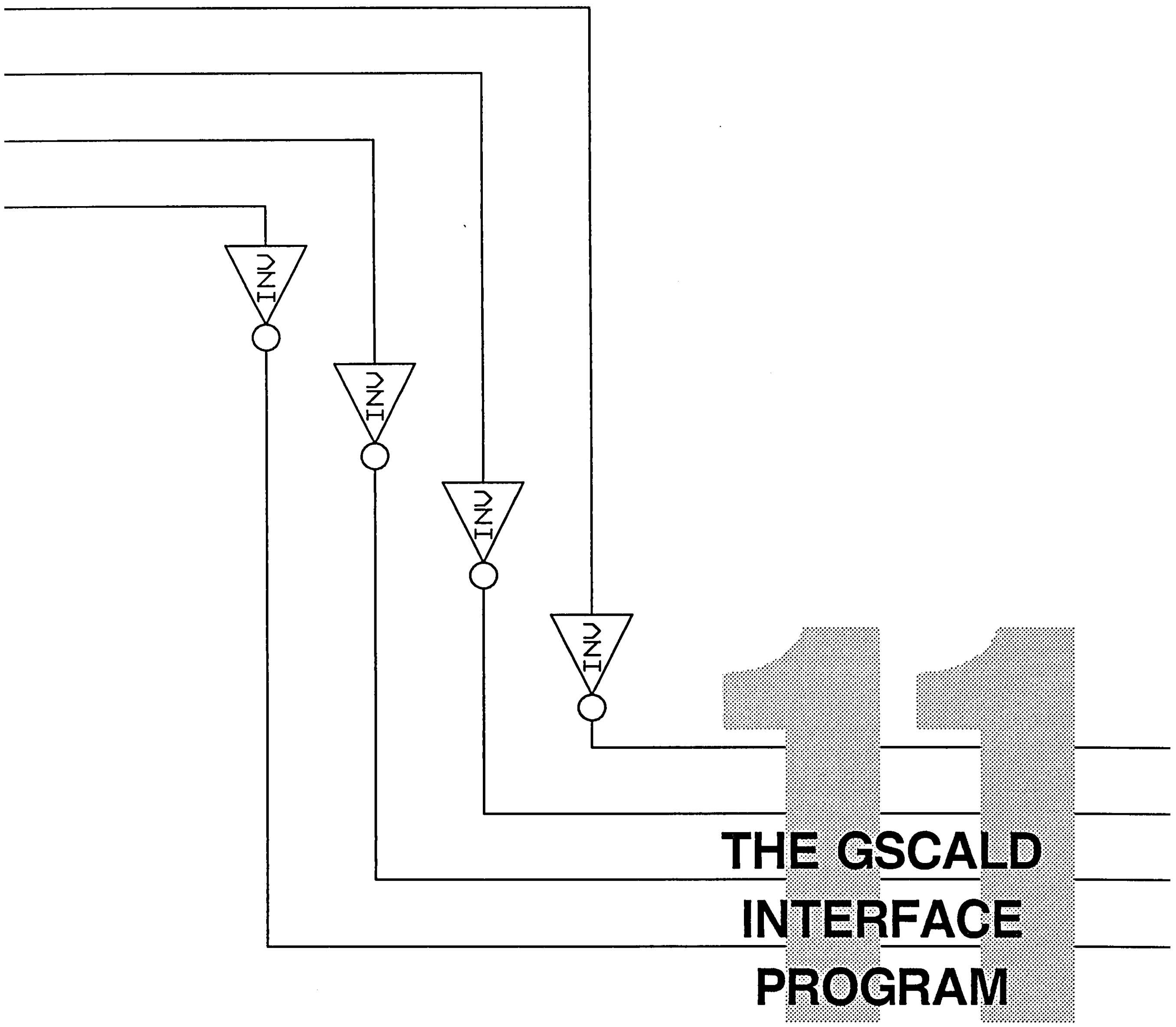
or

Type in a new pin number and then point to the desired pin. The selected pin is swapped with the pin having the pin number you specified.

The properties attached by the **pinswap** command cannot be changed, only deleted and moved. Once pins on a part have been swapped, the part cannot be resectioned using the **section** command.

Now you are ready to use a physical interface program. Back annotation and the interface programs are completely independent of each other. You can back annotate your design first and then use an interface program, or vice versa.

LESSON 11



THE GSCALD INTERFACE PROGRAM

The GSCALD interface is one of several interface programs available on workstations that include the Packager. If your workstation only supports schematic capture (that is, if it includes only GED), the Standard Interfaces are not included and you can skip this lesson.

There are two basic types of interface programs:

- Logical interface programs
- Physical interface programs

All the interface programs work the same way. The only difference is that the logical interface programs accept the Compiler output files as input, and the physical interface programs accept the Packager output files as input. Each interface program accepts input files (either from the Packager or from the Compiler), and formats this information in the manner required by the destination system. For example, GALLEGRO formats the Packager output for the Allegro printed circuit design system, and GTEGAS formats the Compiler output for use by the TEGAS V simulator.

Using GSCALD

GSCALD is a general purpose interface program that formats concise net and parts lists from the Packager output files. GSCALD does not directly target a particular physical design system. GSCALD is introduced here as an example of a physical interface program. The other interface programs work the same way and are just as easy to run.

You run the GSCALD interface program from UNIX. As do all of the interface programs, GSCALD requires a directives file in which you specify a library file directive. Follow these steps:

1 From the UNIX prompt, type:

```
vi scald.cmd
```

You do not have a default directives file for GSCALD, so your screen reads "New File."

2 Press the **i** (lower case 'i') to insert text at the cursor, and enter these lines:

Remember, pstchip.dat *is the short form of the library chips files that the Packager produces.*

```
library_file 'pstchip.dat';
end.
```

3 Press the Esc key to stop entering text.

4 To save the directives file and exit, type:

```
:wq
```

5 Type:

```
gscald
```

Tips on Using GSCALD

✓ If you have trouble running GSCALD, check that the GSCALD program is installed on your system. The full UNIX pathname of the program is:

```
/usr/valid/tools/bin/gscald
```

See Lesson 13, *Understanding UNIX*, for instructions on how to locate a file on your system.

Output Files

GSCALD produces the following output files:

Concise Net List File
Body Ordered Net List File
Concise Part List File
Stuff List File
Power and Ground List File
Listing File
Cross Reference File

Here is a brief description of each file produced:

- **Concise Net List (dialcnet.dat)**

  The Concise Net List is a list of the nets in the design that have at least two nodes. Nets connecting one pin are excluded.

- **Body Ordered Net List (dialbonl.dat)**

  The Body Ordered Net List contains the same information as the Concise Net List, but is ordered by physical part designator (U-number) rather than by net.

- **Concise Part List (dialcprt.dat)**

  The Concise Part List lists all physical parts (by name) and quantities used in the design.

- **Stuff List (dialstf.dat)**

  The Stuff List is a list of physical parts and the corresponding physical part designator (U-number) for each. It is a useful list of the parts required to stuff the board.

- **Power and Ground List (dialpgnd.dat)**

  The Power and Ground List is a list of the pins connected to power supplies and ground connections on each physical part (by U-number).

- **Listing File (scald.lst)**

  The Listing File provides process information on the program and lists error messages when any occur. It is similar to the listing file of the Compiler or the Packager.

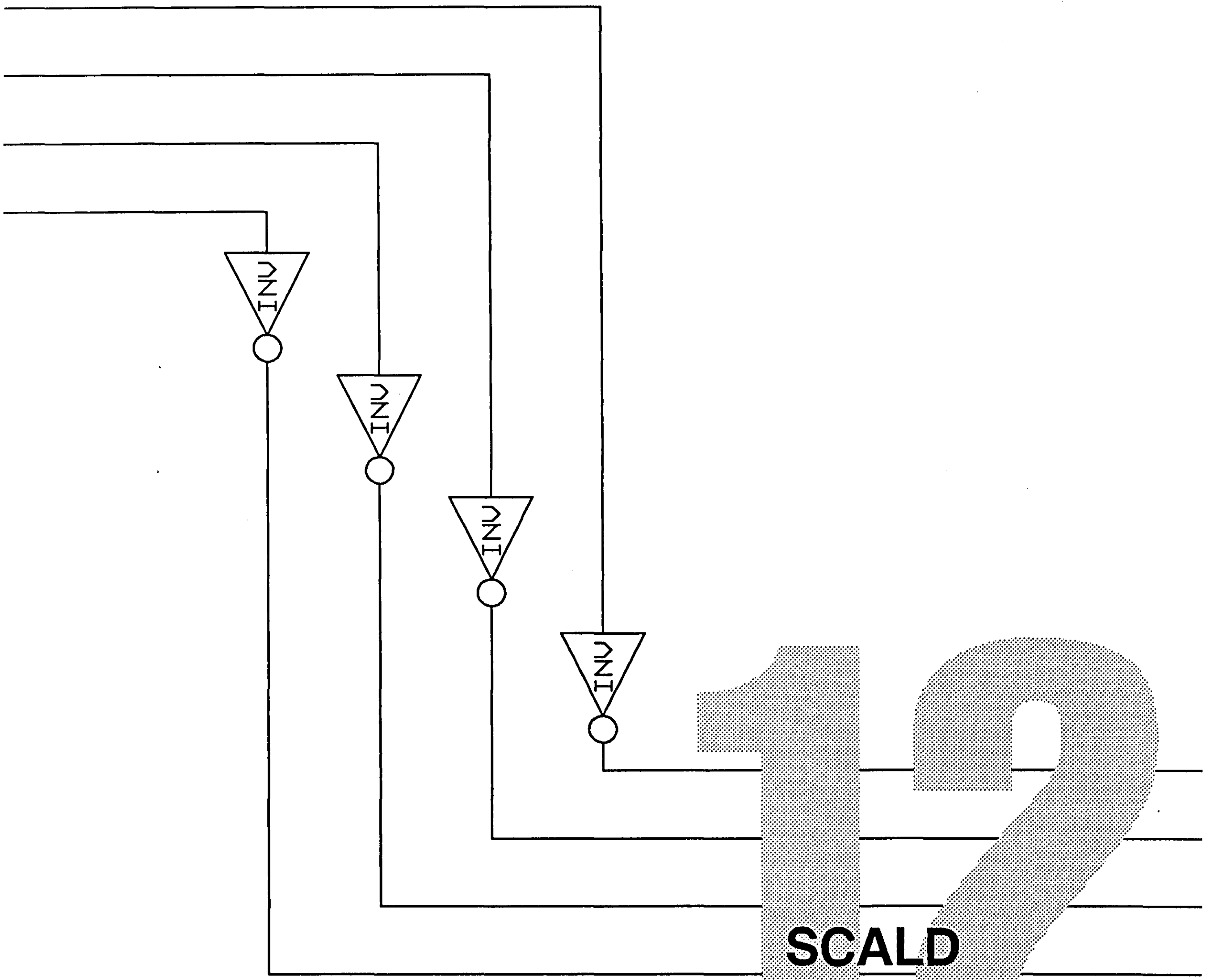
- **Cross Reference File (scald.xrf)**

  The Cross Reference File lists the local part, global part, and global cross references.

Accessing a Physical Design System

To send your design to a physical design system such as Allegro, package your design, run the GALLEGRO interface program just as you ran the GSCALD program, and then submit the GALLEGRO output netlist and device files directly to the Allegro physical design system. If the physical design system is networked to your workstation, you can transfer the netlist and device files from the interface over the network (if the physical design system is a networked resource) or use magnetic tape. Consult your system administrator for the appropriate procedure.

LESSON 12



SCALD DIRECTORIES

This lesson teaches you about SCALD directories, and the next lesson teaches you about the UNIX operating system. You need to read this material to be able to effectively use the SCALD design tools on your workstation. In this lesson, you learn about SCALD directories and how to use drawings stored in different directories. You also learn how to use the **directory** command in GED to list your available library parts, and how to borrow a drawing from another directory.

New Commands

directory <*>	**masterlibrary**
remove	**use**

Now that you've taken a drawing through a complete design cycle from schematic entry through compilation, packaging, and back annotation, you have a good understanding of the design cycle and how the SCALD design tools work together. The data that you enter when you draw a schematic in GED is retrieved and manipulated by a number of different programs such as the Compiler and the Packager. You can also define your own tools to be used in conjunction with your design workstation, perhaps a different kind of simulator or an interface to a particular physical design system.

The reason your information can be retrieved and manipulated so easily is because storage issues (where to store the different data, in what form to store the data, how to retrieve the data) have been carefully reviewed. Since Valid recognized that the users of SCALD systems are design engineers who think in terms of drawings and schematics, the SCALD system is designed around drawings. The SCALD system does most of the routine file manipulation for you so that you won't have to deal directly with UNIX.

Since the role of the SCALD directory is intrinsic to the operation of the SCALD system, it's important for you to know about SCALD directories. If you understand SCALD directories, then you only need to know a little bit about UNIX. If you don't understand the function of a SCALD directory, even a good deal of UNIX knowledge is just going to be confusing.

What is a SCALD Directory?

A SCALD directory is a filing system that GED uses to store your drawings in UNIX and retrieve them from UNIX without you having to worry about how UNIX works and where your drawings are stored.

As you've noticed from your work with the full-adder circuit and the subtractor, all you had to do was go into GED and issue the **edit subtractor** command, draw the schematic, and save it with the **write** command. Your drawing gets saved somewhere, and GED knows where to find it each time you request it.

GED uses a small index file to find your drawings. For each drawing you create, GED writes the name you gave the drawing and its corresponding UNIX name into this file. (The UNIX name is the UNIX directory where the drawing is stored.) So, when you give the command **edit subtractor** to GED, GED looks in its index file for a drawing with that name, goes and gets it for you, and it appears on the screen.

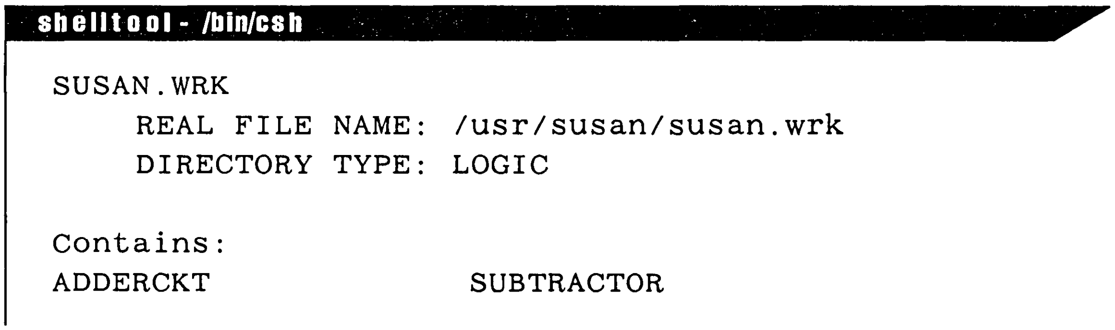
susan.wrk *is a SCALD directory name used in some of the tutorial examples.*

The index file that GED makes for itself and uses to store and find your drawings is called a *SCALD directory*. The name of your SCALD directory for this tutorial project has been your login name with the extension *.wrk*.

Where you see SCALD Directories

You probably remember that the right–hand field of the status line (when you are in GED) tells you the name of your current working directory. Now you know that another name for this is your "SCALD directory." Whenever you see a file named *susan*.wrk, you know that it is a SCALD directory and that it contains a list of a user's drawings.

You probably also remember that when you used the **directory** command (in Lesson 3), you saw this on the screen:



```
SUSAN.WRK
     REAL FILE NAME: /usr/susan/susan.wrk
     DIRECTORY TYPE: LOGIC

Contains:
ADDERCKT                  SUBTRACTOR
```

The screen says *SUSAN*.WRK at the top because the **directory** command goes to your SCALD directory and tells you four things:

- The name of your SCALD directory (*susan*.wrk).
- The UNIX pathname to your SCALD directory (so you could find it in UNIX if you had to).

Other directory types are reserved for library development purposes and documentation.

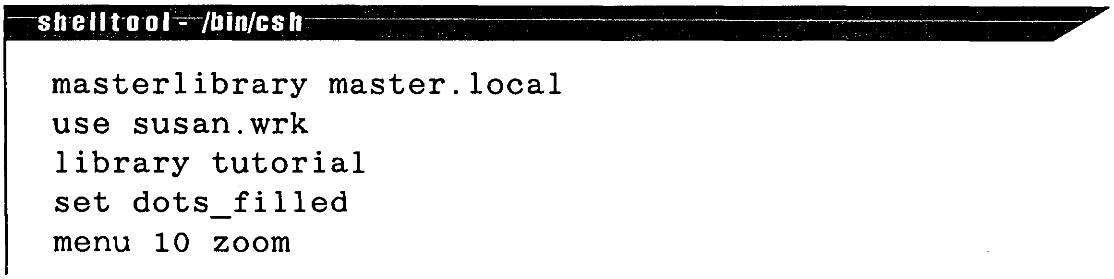
- The directory type: this tells you what sort of things are stored here. If these are user–created schematics, the type will always be "LOGIC."
- The names of each of your drawings, in alphabetical order.

The SCALD Directory in Your Startup.ged File

Your SCALD directory also appears in your *startup.ged* file. In fact, the entry in this file is what makes the whole SCALD directory system work. Let's look at your *startup.ged* file again. From the UNIX prompt, type:

```
more startup.ged
```

This is what you see:



```
masterlibrary master.local
use susan.wrk
library tutorial
set dots_filled
menu 10 zoom
```

Remember, you added the the line "use *susan*.wrk" in Lesson 7. Now you can understand that it tells GED to use a SCALD directory named "*susan*.wrk." As a reminder, a *startup.ged* file with a **use** statement is created for you when your user account is made. Just be sure you don't delete or change this line.

The Use Command

The **use** statement in your *startup.ged* file gives the name of your SCALD directory. It looks like this:

```
   use susan.wrk
```

This statement tells GED to look in the SCALD directory named *susan*.wrk to find the drawings you are going to edit. When you save a drawing with the **write** command, this statement tells GED to record the location of the edited drawing in the same SCALD directory (*susan*.wrk.)

You can put more than one **use** statement in your *startup.ged* file, and you can also give the command:

```
   use proj1.wrk
```

while you are working in GED. In fact, the **use** command in your *startup.ged* file is the same command. All the *startup.ged* file does is issue the specified commands as soon as you enter GED.

Multiple Use Statements

When you put two **use** statements in your *startup.ged* file, like this:

```
   use susan.wrk
   use proj1.wrk
```

GED executes the first command and then the second command. If you watch the top of the screen very carefully when you first go into GED, you'll see it say "*susan*.wrk" briefly, and then "proj1.wrk." This is because GED first reads the contents of *susan*.wrk and then the contents of proj1.wrk. The order of the **use** statements is very important. The SCALD directory in the *last* **use** statement in your *startup.ged* file is your current working directory in GED.

The Search Stack

When you place multiple **use** statements in your *startup.ged* file, what you are really doing is making a search stack. You are telling GED to look for your drawings in certain files, but you are also telling GED where to look first for the drawing you request and where to look next. The SCALD directory you list **last** is put on the top of the stack. This is where GED first looks for your drawings and where GED puts each new drawing unless told otherwise.

When you give a **use** command in GED like this:

```
use proj1.wrk
```

You see the top of your screen change to read "proj1.wrk," and now the SCALD directory named proj1.wrk is on the top of the stack.

If there is not yet an existing SCALD directory named proj1.wrk, GED creates it when you write a drawing into it.

The Library Command

You used the **library** command in Lesson 1 when you entered the command:

```
   library tutorial
```

while you were using GED. Later you learned how to enter a **library** command into your *startup.ged* file. The **library** command is just like the **use** command, except that it is used for libraries.

For more information on libraries, see Lesson 14.

When you give the **library** command, you make a library available for use and you enter that library into the search stack. Libraries are like SCALD directories because they are directories that contain drawings. But libraries are different from SCALD directories because although you use the drawings of the parts in them, you don't change the part drawings. Since the parts in each library usually have unique names (e.g., LS00, 74S00, 7400, 74ALS00), the order of libraries in the search stack is not important. When you add a part, the first library in the stack is searched and, if the part is not found, the next library is searched.

The Directory Command

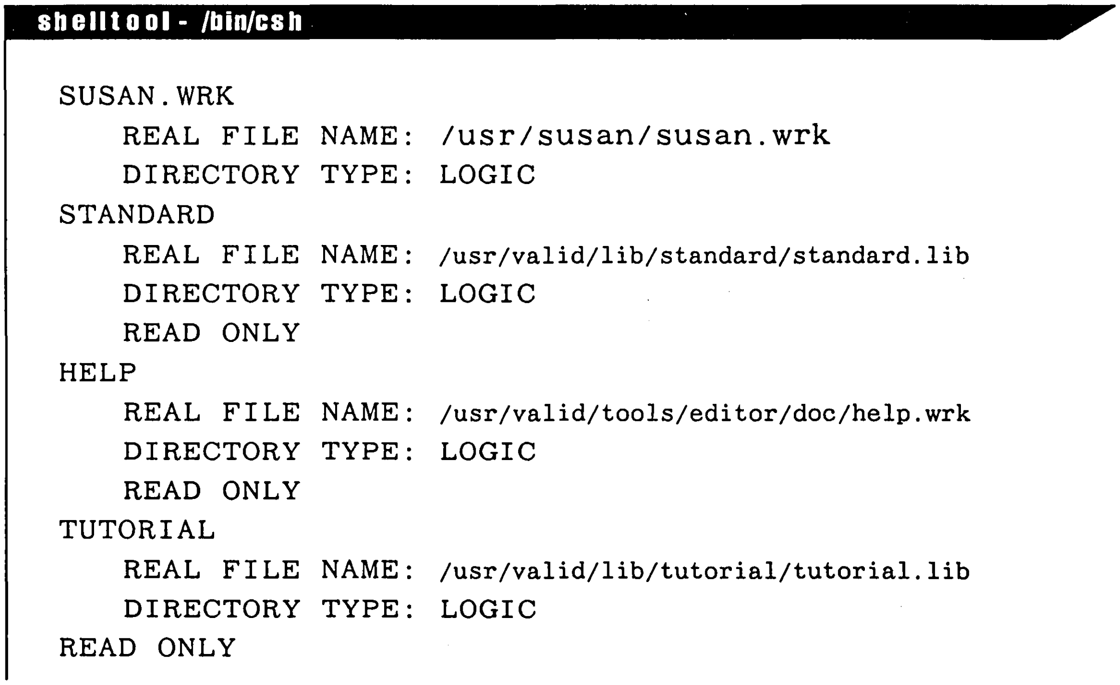
In Lesson 3, we introduced the **directory** command and showed you how to get a listing of the drawings in your current directory (or more specifically, your current SCALD directory). This is the SCALD directory on the top of your search stack. Now we'll show you some more things that the **directory** command does for you.

The basic **directory** command you used in Lesson 3 lists your current SCALD directory and the drawings in it. The **directory** command can also be used to see all of the SCALD directories you are currently using, like this:

```
   directory <*>
```

The angle brackets < > surround a SCALD directory name, and the asterisk is a wildcard. The command means that you want to see a list of the names of all of your SCALD directories.

The list of all of your directories looks something like this:


shelltool - /bin/csh

```
SUSAN.WRK
    REAL FILE NAME: /usr/susan/susan.wrk
    DIRECTORY TYPE: LOGIC
STANDARD
    REAL FILE NAME: /usr/valid/lib/standard/standard.lib
    DIRECTORY TYPE: LOGIC
    READ ONLY
HELP
    REAL FILE NAME: /usr/valid/tools/editor/doc/help.wrk
    DIRECTORY TYPE: LOGIC
    READ ONLY
TUTORIAL
    REAL FILE NAME: /usr/valid/lib/tutorial/tutorial.lib
    DIRECTORY TYPE: LOGIC
READ ONLY
```

Figure 12-1. A List of Your SCALD Directories

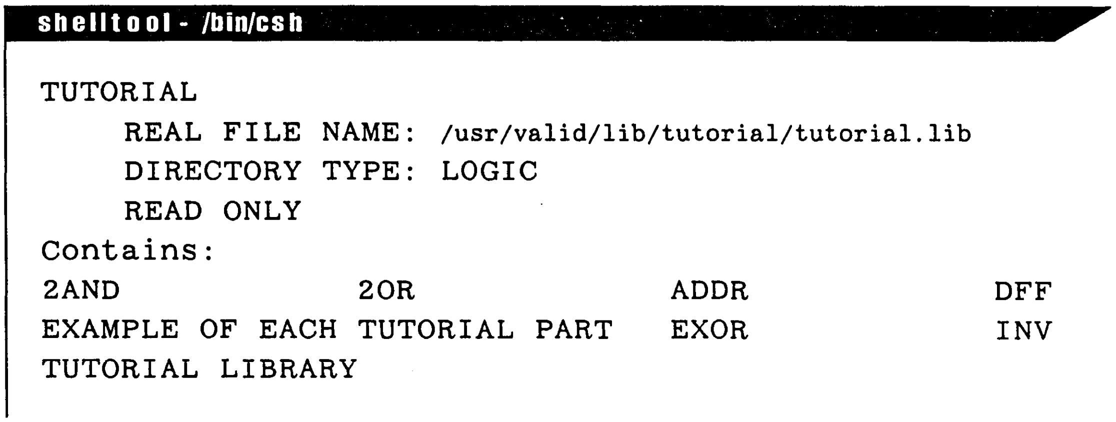
You can see that your libraries are listed along with your current SCALD directory. This is a convenient place to see what libraries you have available currently in this GED session.

You can also use the **directory** command to get a list of all of the parts in one of the libraries you are using, like this:

```
directory <tutorial>*
```

The directory name here is "tutorial" and the asterisk is a wildcard. This command means that you want to see a list of all of the drawings in the tutorial library.

This is what you see on the screen:



```
TUTORIAL
    REAL FILE NAME: /usr/valid/lib/tutorial/tutorial.lib
    DIRECTORY TYPE: LOGIC
    READ ONLY
Contains:
2AND               2OR                  ADDR               DFF
EXAMPLE OF EACH TUTORIAL PART           EXOR               INV
TUTORIAL LIBRARY
```

Figure 12–2. Drawings in the Tutorial Library

To learn more about libraries, see Lesson 14.

Each name in the list is a drawing in the Tutorial library. Notice the names of each library part in this list.

The Search Stack Listing: directory <*>

When you listed your active directories with the **directory** <*> command, you may have noticed that the listing did not appear in alphabetical order. That's because this command not only tells you what directories you are currently using, but also the order in which they appear in the search stack. Look at Figure 12–1 again; the libraries are on the bottom of the list.

Borrowing a Drawing from Another User

GED always writes a new drawing to the current SCALD directory (the directory on the status line) and always writes an existing drawing back into the SCALD directory where the drawing was found unless specifically told to write the drawing into a different directory. The ability to write a drawing to another directory allows you to borrow a drawing from another user or from another of your own directories by reading (with the **edit** command) a drawing from its directory and then writing a copy of the drawing into your directory.

The following procedure describes how to borrow a drawing. The procedure assumes that you are borrowing drawing "circuit 1" in local user Steve's SCALD directory "proj1."

To borrow this drawing:

1 Specify the SCALD directory that contains the drawing to be borrowed with the **use** command:

```
use /usr/steve/proj1/proj1.wrk
```

2 Read the drawing from the specified SCALD directory with the **edit** command:

```
edit circuit 1
```

3 Write the drawing to your directory:

```
write <susan.wrk>
```

The brackets around the directory name are required. Optionally, you can specify a different name for the drawing with the **write** command. For example, the command

```
write <susan.wrk>ckt a
```

renames the original drawing (circuit 1) "ckt a."

This procedure writes a copy of circuit 1 into the SCALD directory *susan*.wrk while leaving the original drawing in /usr/steve/proj1/proj1.wrk untouched. After borrowing the drawing, again specify your directory (type `use susan.wrk`) so that you edit the copy of the drawing in your SCALD directory.

The Masterlibrary Command

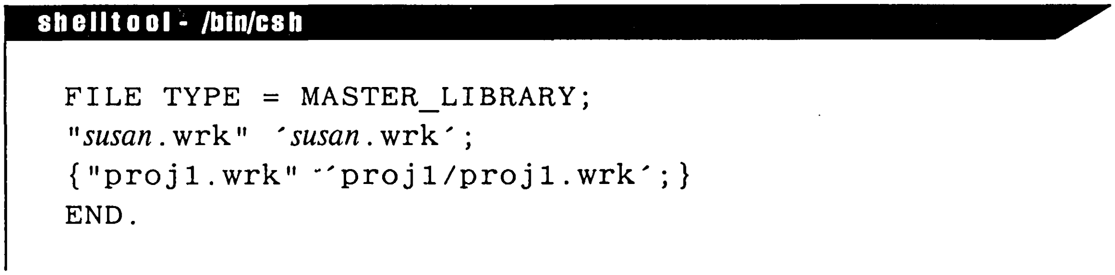
The first line in your *startup.ged* file contains a **masterlibrary** command. This command references a file that contains short forms or abbreviations of SCALD directories that you regularly use.

Once you make an entry in this file, you can use just the short form or abbreviation of the directory name rather than the complete pathname. Short forms and abbreviations can be used with any command that requires a SCALD directory or library name. Entries in this file can include both the SCALD directories of other users as well as your own SCALD directories.

A default file, named *master.local*, is already set up in your login directory. To look at this file, type:

```
more master.local
```

This is what you see:



```
FILE TYPE = MASTER_LIBRARY;
"susan.wrk" 'susan.wrk';
{"proj1.wrk" ''proj1/proj1.wrk';}
END.
```

In the file, *susan* is replaced with your login name. The third line of the file shows the entry format. This line is actually a comment since it is enclosed in curly braces. The line shows a hypothetical entry for the SCALD directory "proj1.wrk" located in UNIX directory "proj1" under your login directory.

Using the example from the previous section on borrowing a drawing, adding the line

```
"pr1" '/usr/steve/proj1/proj1.wrk';
```

to your *master.local* file would allow you to enter simply "`use pr1`" in the example rather than the full pathname (/usr/steve/proj1/proj1.wrk).

For complete details on the masterlibrary *command, see the* ValidGED Command Reference Manual.

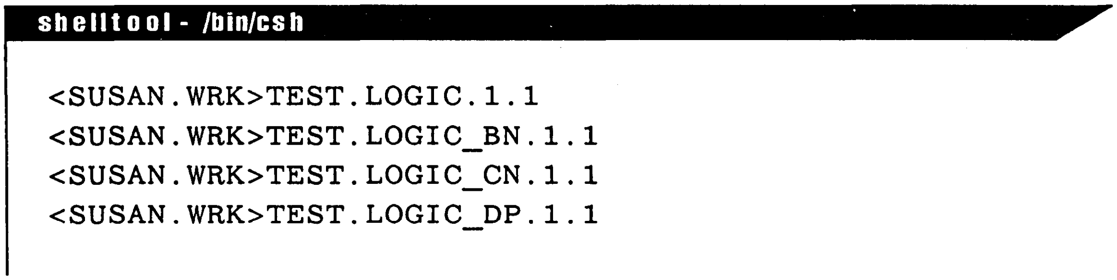
The **masterlibrary** command is powerful and can save you considerable typing.

The Remove Command

To delete a GED drawing, you use the **remove** command. To delete a drawing named TEST.LOGIC.1.1 you type:

```
remove test
```

GED then displays the names of all of the drawings that have TEST in the first field of their name, like this:



```
<SUSAN.WRK>TEST.LOGIC.1.1
<SUSAN.WRK>TEST.LOGIC_BN.1.1
<SUSAN.WRK>TEST.LOGIC_CN.1.1
<SUSAN.WRK>TEST.LOGIC_DP.1.1
```

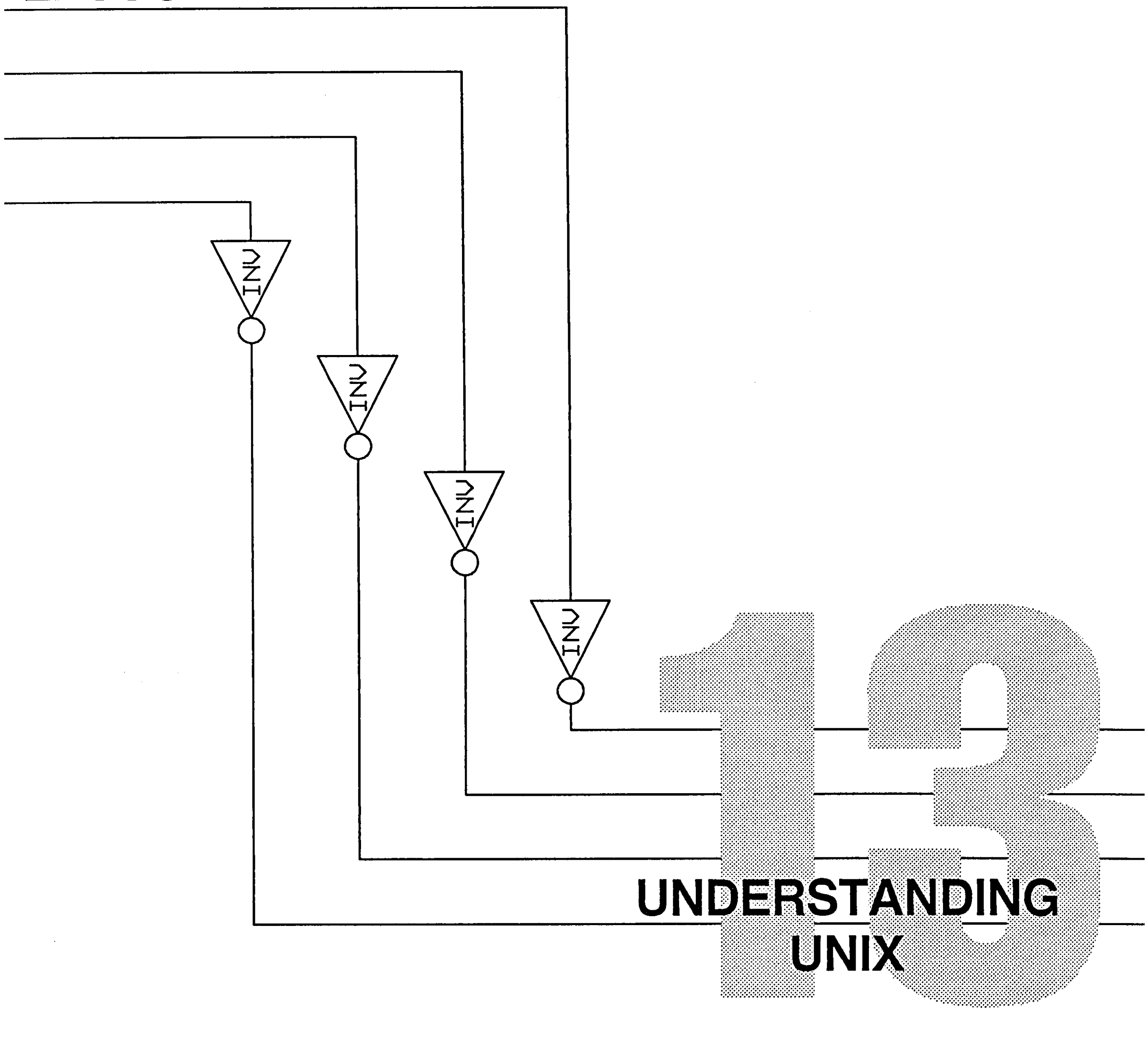
If you wish to remove all of the listed drawings and files, you type ';' (semicolon). Instructions are given on the screen.

To remove a drawing TEST.BODY.1.1 while retaining a drawing TEST.LOGIC.1.1, you type:

```
remove test.body
```

GED then removes only the specified drawing.

LESSON 13



UNDERSTANDING UNIX

In this lesson you learn about the structure of the UNIX file system and how to move from one directory to another in the UNIX system. This lesson introduces these new UNIX commands:

New Commands

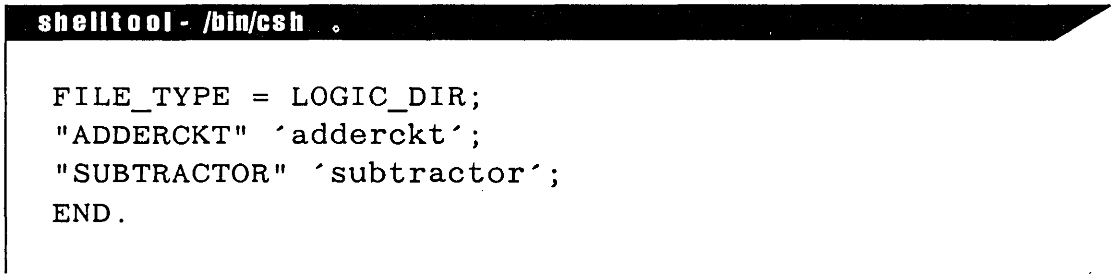
cd	**pwd**

SCALD Directories and UNIX

In Lesson 12 you learned that the SCALD Directory acts as the go-between for GED and UNIX. You now need to understand the other half of the picture: the UNIX half. When you used UNIX in Lesson 7 (the *ls* command) to look at your default files in your user account, you saw one of the files listed there was named *susan*.wrk. This is your SCALD directory. It is a UNIX file (in fact everything on your workstation is stored as a UNIX file). Let's look at it just for your information. From the UNIX prompt, type:

```
more susan.wrk
```

The file appears as shown in Figure 13-1.



```
FILE_TYPE = LOGIC_DIR;
"ADDERCKT" 'adderckt';
"SUBTRACTOR" 'subtractor';
END.
```

Figure 13-1. The Contents of a SCALD Directory from UNIX

Don't ever edit this file or delete it. If you do, GED won't work properly.

The UNIX Tree

Directories are shown as boxes on the diagram; files are shown as circles.

The UNIX operating system stores everything in a branching file structure of directories and subdirectories that resembles an upside-down tree. Figure 13-2 shows you a portion of the UNIX file structure. You move around in the file system by following the branches (the lines on the diagram). This usually involves moving up, then over, then down a different branch.

You can go and look at any text file in the system (with the *more* command), but you can only change (using *vi*) a file that you "own." (These are usually the files in your user account.)

The very top directory of the file structure is what we call "root" (remember this is an upside-down tree). As you might have guessed, if you log on to the system as "root," you have access to everything and permission to change everything. Root is sometimes called the *superuser*, and you need to give a certain password to log on.

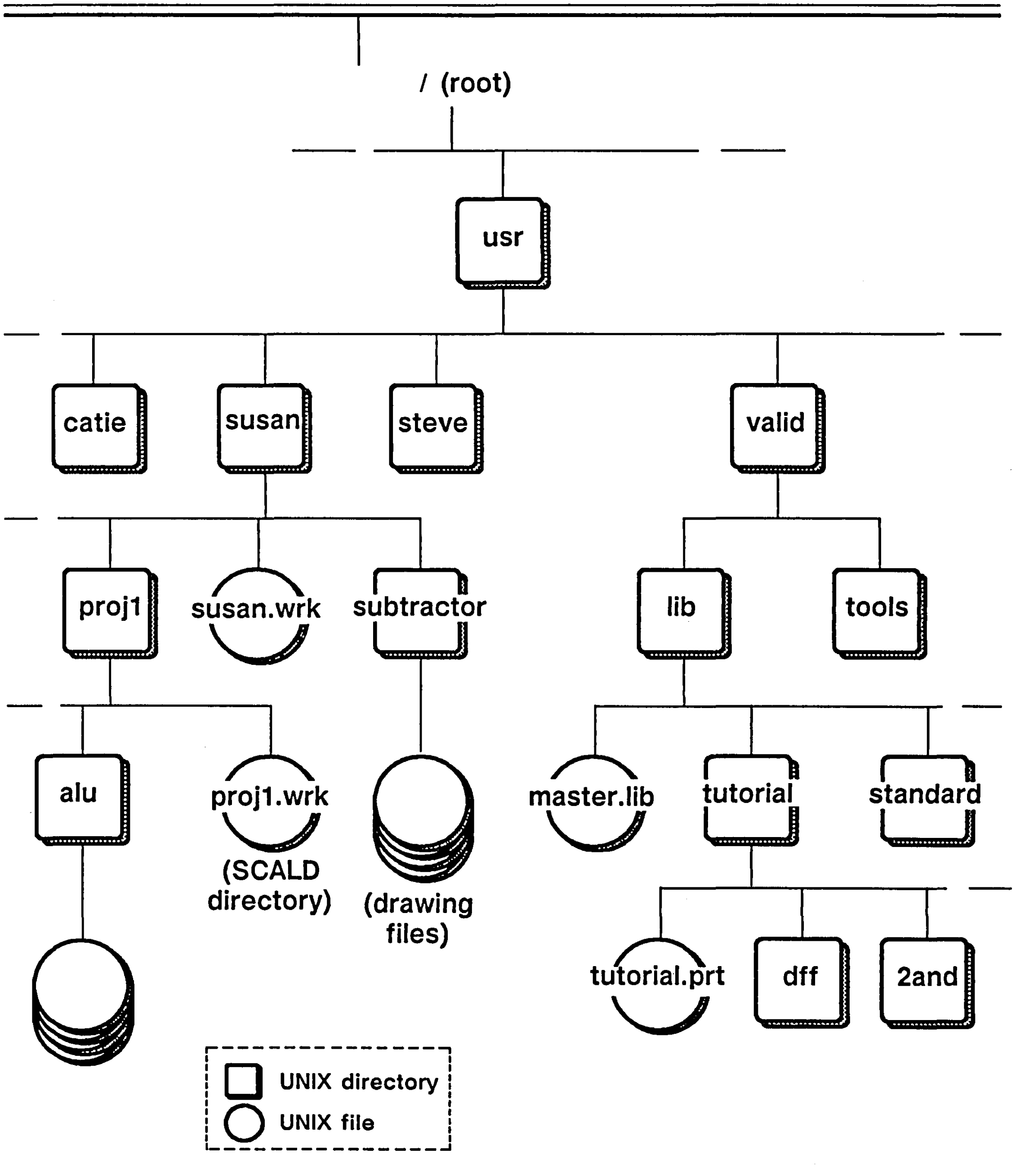


Figure 13-2. UNIX File Structure

GED Drawings Stored in UNIX

Now that you know that everything on your workstation is stored somewhere in UNIX, you are probably curious to know where your GED drawings are stored. Look at Figure 13–2 again and find the box named **subtractor**. The key to Figure 13–2 tells us that the boxes represent UNIX directories and the circles represent UNIX files. Your drawing named SUBTRACTOR (like any other GED drawing) is stored in a UNIX directory. Under this directory is a whole group of UNIX files that store different information about your drawing. One file stores the information to make your drawing appear on the screen, another stores the information that is passed on to the Compiler, and so on. Don't try using the *more* command to list these files to the screen; some of them aren't even text files. And don't ever try to alter one of these files. To edit a drawing, ALWAYS USE GED.

Moving Around in the UNIX File System

So far, you have looked only at the files in your home directory and have not needed to move around the UNIX tree to find other files. But now you're ready to learn how to go and look at files outside of your home directory. To look at a file elsewhere in the UNIX file system, you use the *cd* (change directory) command. You use the *cd* command to move to the directory that contains the file you want to see, and then you look at the file using the *more* command. To use the *cd* command, you need to know where the directory is that you want to move to, and you need to know where you currently are located. This is because movement with the *cd* command is often relative movement, movement in relation to your current position. So, let's find out where you currently are in the system.

The Pwd Command

At the UNIX prompt, type:

```
pwd
```

This command stands for "print working directory," and can be entered at any time from the UNIX prompt. It prints the full pathname of the directory to which you are currently connected. The answer to your query is

`/usr/`*`susan`*

where *susan* is the name of your user account. This is your login, or *home*, directory, the directory to which you are connected each time you log in. The message "usr/*susan*" means that your current directory is named "*susan*" and that it is under a directory named "usr." If you look at the UNIX tree in Figure 13-3, you can see that the directories "catie," "susan," and "steve are all at the same level and that they are under a directory named "usr." "Usr" is a directory we call the user area, because this is where each user's login directory (or home directory) is located and where all the users' files and subdirectories are stored.

The Cd Command

The *cd* command connects you to another directory. It can be used with the following three options:

cd *dirname*

This connects you to a directory called *dirname* that is directly under your current directory.

cd ..

This connects you to the directory immediately above your current directory. (Be sure to type a space between **cd** and ..)

cd

This command (without an argument) connects you to your home directory from wherever you are. It is very useful when you have been travelling around in the UNIX tree and want to go "home."

Full UNIX Path Names

When you used the *pwd* command above, the reply was:

```
/usr/susan
```

This is the full UNIX pathname of the directory *susan*. A full UNIX pathname tells you, in a short (and rather cryptic) notation, how to get to a particular directory or file from root (the top of the UNIX tree). Full UNIX pathnames are used extensively in the UNIX operating system because they uniquely identify any directory or file in the system. From every branch of the UNIX tree there is one unique path back to the top (root).

One of the beauties of the UNIX system is that without restricting the names that a user is allowed to use, each directory and file can always be uniquely identified. If I make another directory named "susan" from my home directory, these two directories can each be uniquely identified. As you see in Figure 13–3, the pathname of the first is /usr/susan and the pathname of the second is /usr/susan/susan.

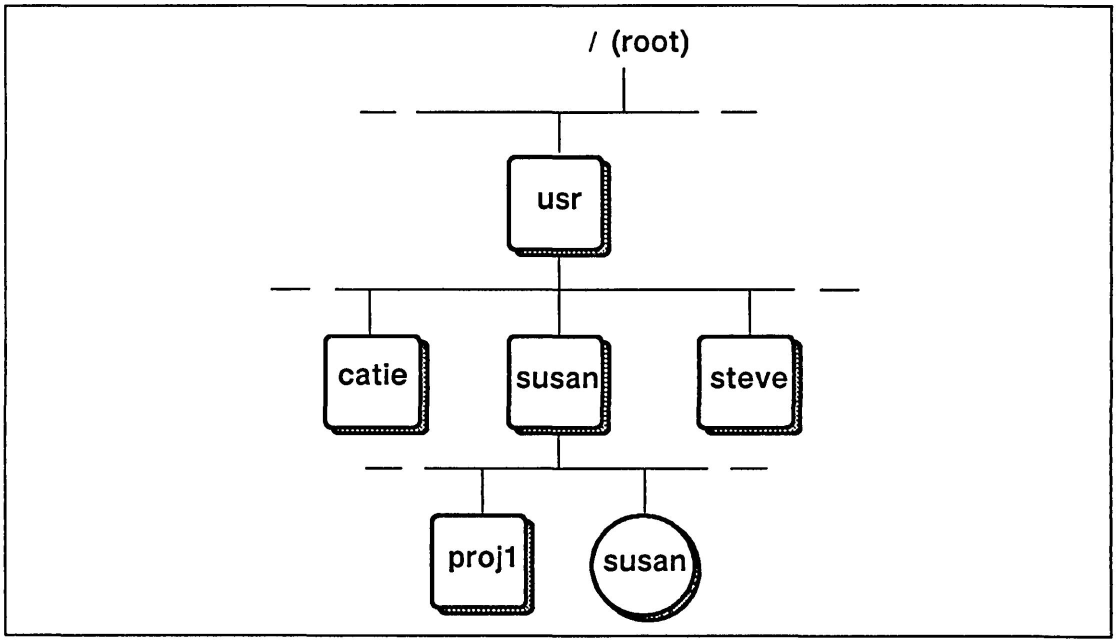


Figure 13–3. Naming Subdirectories

The only restriction is that two directories or files AT THE SAME LEVEL cannot have the same name. If you try to make a new directory of the same name as an existing directory at that level, you get an error message.

The one thing that is a little hard to remember about a full UNIX pathname is that it always starts with a / (slash). The way this is usually explained is that the slash stands for root. But only the leading slash in a pathname stands for root, not the others.

Another way to remember how to write a full UNIX pathname correctly is to pretend that you are at the top of the tree (root) and then to give directions to the file or directory, like this:

First I connect to "usr," then I connect to "susan."

Now replace each occurrence of the phrase "I connect to" with a slash (/). You get:

```
/usr/susan
```

So think about the slash as indicating movement (like a path does). If you are at root and you want to go somewhere else, the first thing you have to do is move somewhere. So the slash comes first.

Moving Around in UNIX: Exercise

Now that you have all the tools you need to move anywhere in the UNIX tree, let's practice going places. Here are three short excursions; each starts from your home directory. Refer to Figure 13–4 for the location of each destination.

1 To /usr/valid/lib/tutorial/tutorial.prt and home again.

Solution:

```
cd ..
cd valid
cd lib
cd tutorial
more tutorial.prt
q (to leave the file without going to the end)
cd
```

2 To /usr/valid/lib/master.lib and home again.

Solution:

```
cd ..
cd valid
cd lib
more master.lib
q (if necessary)
cd
```

3 To root and home again.

Solution:

```
cd ..
cd ..
cd
```

Remember, to find out where you are at any time, just use *pwd*. When you are at "root" and you use *pwd*, you'll see just a slash (/).

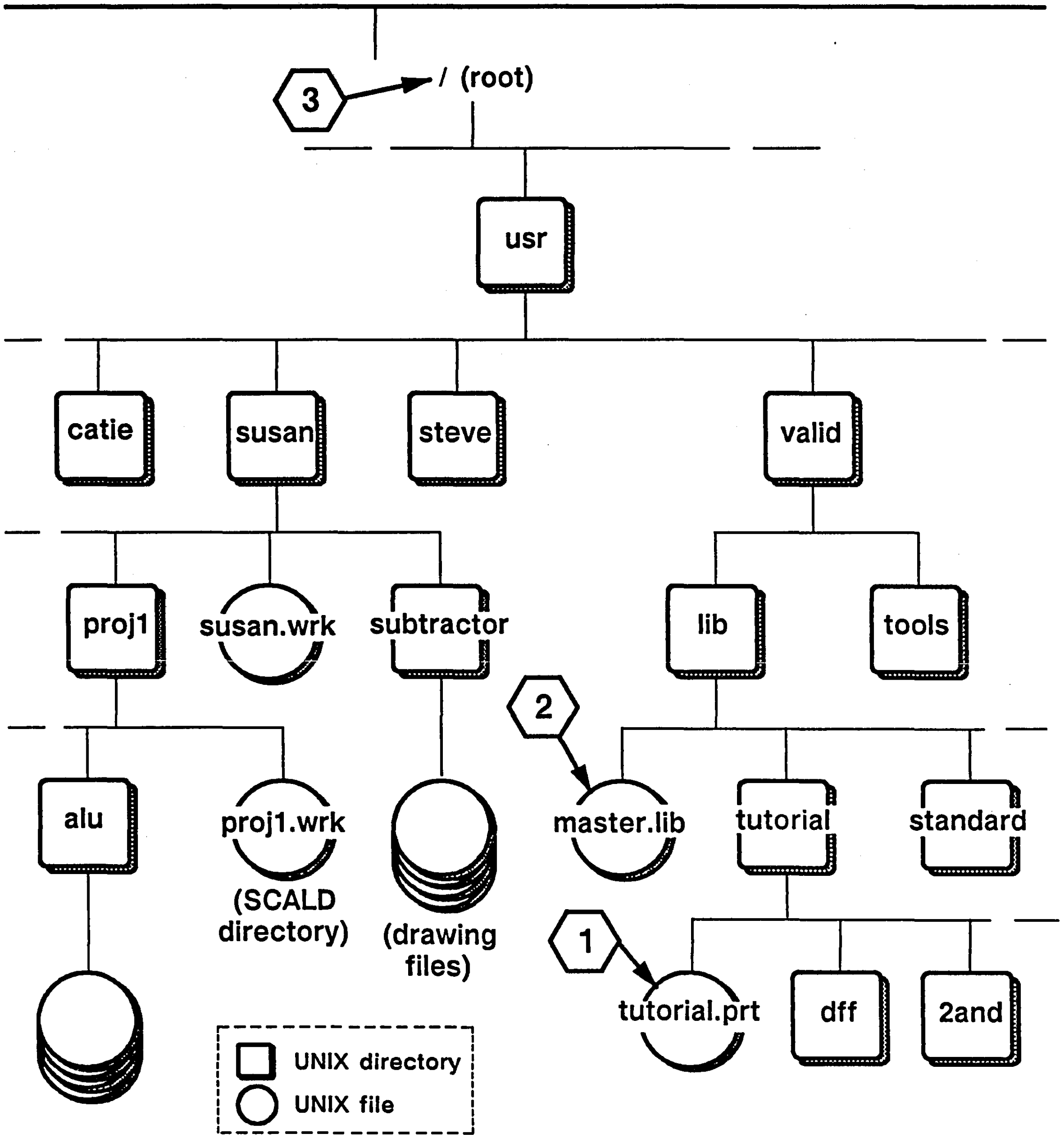


Figure 13–4. Moving Around in the Directory Structure

Moving Faster

Now that you understand how the *cd* command works, we'll show you the fast way to move to a directory. There are two different techniques: one for directories that are under your current position, and one for directories that are elsewhere in the UNIX tree. But be careful, the difference between the two techniques is subtle and very important.

Suppose you are in /usr/susan and you want to go to your directory /usr/susan/proj2/alu. This is a directory under your current position. You can go there directly in one step by using the command:

```
cd proj2/alu
```

This is just like the "cd *dirname*" command we showed you earlier. You only give the last half of the pathname to the destination directory because you are already half way there. Remember the movement is *relative* to your current position.

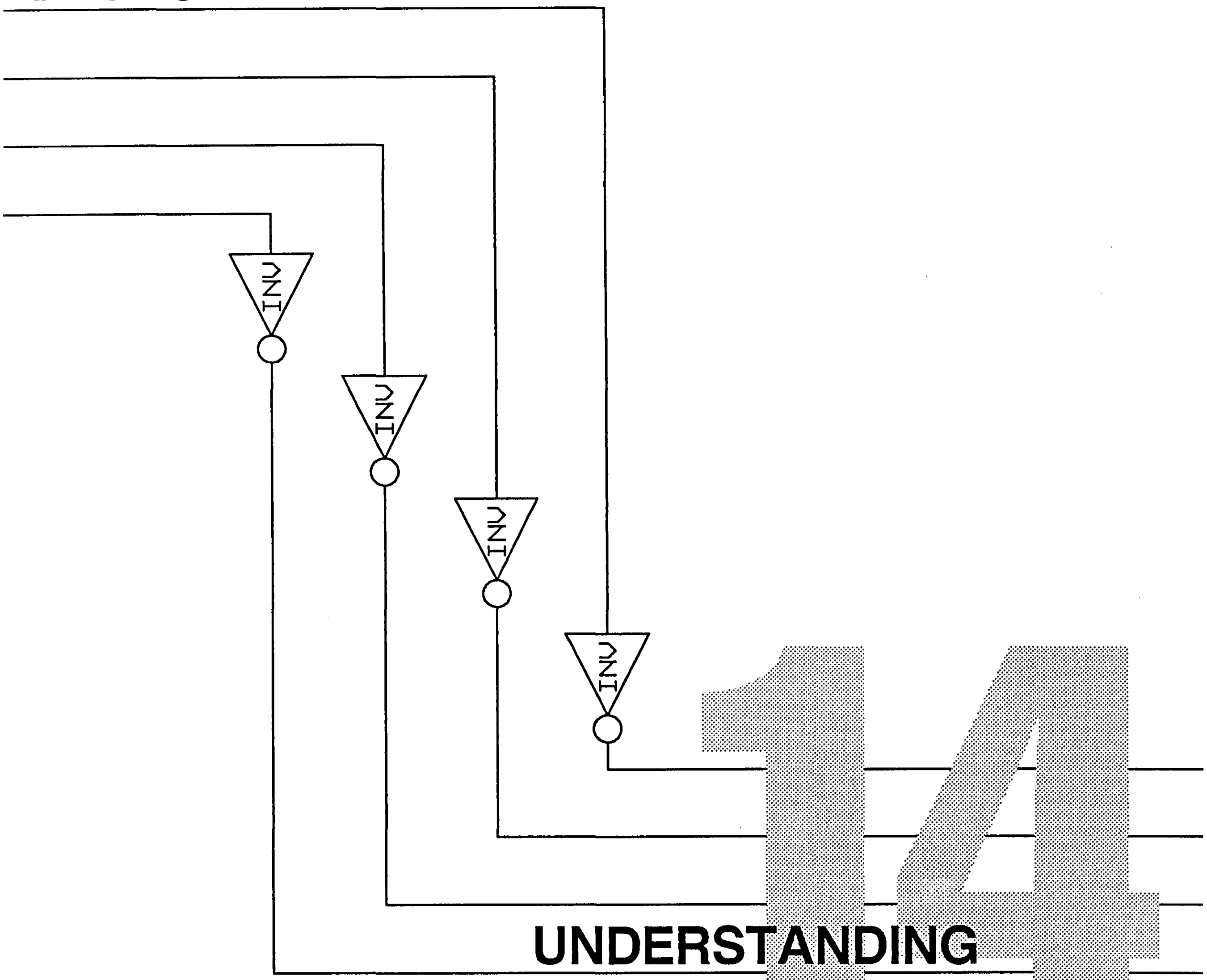
If you want to move quickly to a directory that is NOT under your current directory and you know the full pathname of your destination, you can go there directly by giving the full UNIX pathname like this:

```
cd /usr/jane/design/cpu
```

When the *cd* command is followed by a pathname that begins with a / (slash), it interprets that pathname as a full pathname. The initial slash tells the *cd* command that this defines an *absolute* location. If the *cd* command is followed by a pathname that does not begin with a / (slash), it interprets that pathname as being relative to your current position.

Now you know quite a lot about SCALD Directories and the UNIX file system. You are now ready to learn about the libraries of components that Valid supplies and maintains for your use with the SCALD system.

LESSON 14



UNDERSTANDING LIBRARIES

This lesson explains how libraries are organized and describes the generic files and directories that are common to all Valid-supplied libraries.

The Library Directory: /usr/valid/lib

The directory "usr" is the user area where user files are stored. This directory contains a directory named "valid" where all of the Valid applications programs are located. Within this directory is a directory named "lib" where all of the Valid-created libraries installed on your system are stored. If you go to /usr/valid/lib and list the contents of that directory (*ls* command), you'll see the name of each library installed on your system and a few other files and directories. It will look something like this:

shelltool - /bin/csh

```
configure     standard
ged           tutorial
lsttl         util
master.lib    utilities
```

This is a good place to check quickly whether a particular library is installed on your system. Each library is stored in a directory bearing its name. The full pathname of the time library is:

/usr/valid/lib/time

and the full pathname of the tutorial library is:

/usr/valid/lib/tutorial

Figure 14–1 shows the tutorial library and its place within /usr/valid/lib. Look at this chart for reference while you're reading the rest of this lesson.

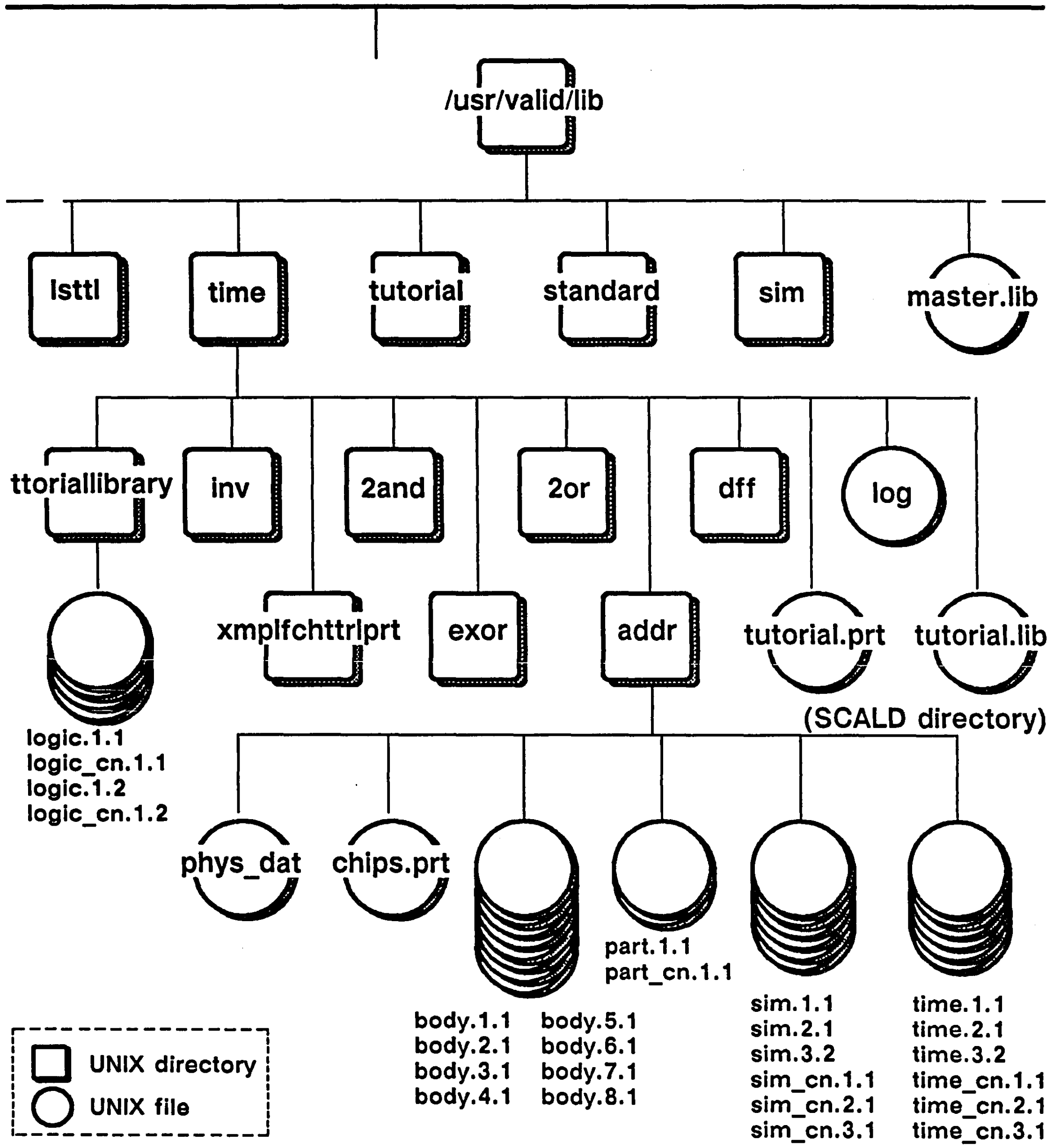


Figure 14-1. Tutorial Library

The Master Library File: master.lib

One important file in /usr/valid/lib that you need to know about is *master.lib*. When you use the **library** command in GED (and in your *startup.ged* file), GED looks in the *master.lib* file for the full UNIX pathname to the library you specified. This library can reside anywhere on the system as long as it's listed correctly in *master.lib*. So if you run into problems using the **library** command, look at *master.lib* and see if the library you want is listed there. If it isn't, you need to add it.

A Sample Library: Tutorial

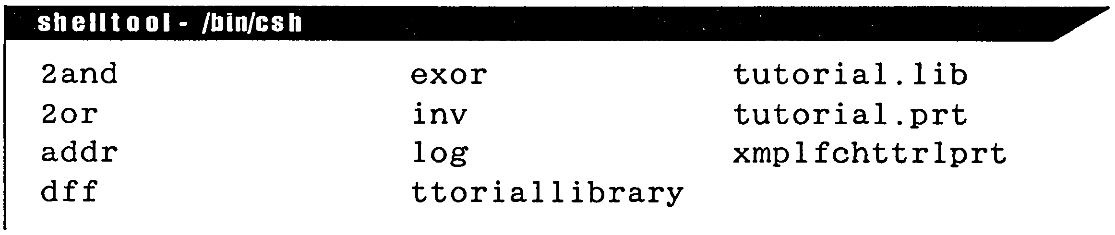
All of the Valid-maintained libraries on your Sun workstation have a similar structure. Knowing a little bit about how these libraries are set up will make you a better and more knowledgeable library user. We'll use the tutorial library as an example because it is very small (it only has a few parts in it).

Let's look at the contents of the tutorial library. To do so, you need to connect to the directory /usr/valid/lib/tutorial and list its contents like this:

1 `cd /usr/valid/lib/tutorial`

2 `ls`

Here is a listing of the contents of the directory /usr/valid/lib/tutorial:



shelltool - /bin/csh

```
2and          exor              tutorial.lib
2or           inv               tutorial.prt
addr          log               xmplfchttrlprt
dff           ttoriallibrary
```

Of the items listed, `2and`, `2or`, `addr`, `dff`, `exor`, and `inv` are the library parts. These are the parts you added in GED with the **add** command. Each of these library parts is a drawing and so, like any other drawings, each is stored in a directory. Two other items are also drawings:

These two drawings are used for documentation and reference purposes and are described in the next section.

```
xmplfchttrlprt
ttoriallibrary
```

The remaining items in the tutorial library are files. Here is a brief description of each one:

log

Each library has a log file, maintained by Valid personnel, of all updates made to the library since its initial release.

tutorial.lib

This is the SCALD Directory for the library named tutorial. Just as the extension *.wrk* is reserved for user SCALD Directories, the extension *.lib* is reserved for library SCALD Directories.

tutorial.prt

This file contains physical information about each part in the library. It is used by the Packager and by the physical interface programs.

Reference Drawings

The tutorial library includes two reference drawings. Their GED drawing names are EXAMPLE OF EACH TUTORIAL PART and TUTORIAL LIBRARY. These two drawings are stored in the UNIX directories “xmplfchttrlprt” and “ttoriallibrary.”

Each library includes two such reference drawings. You can see that the names of the UNIX directories are not exactly the same as the drawing names. This is the work of your SCALD Directory. The UNIX system does not accept spaces and special characters in names. Also, for convenience, each drawing name is shortened to just 14 characters.

The "EXAMPLE OF EACH ..." drawing includes every version of every part in the library. The "EXAMPLE OF EACH ..." drawing for each library is included in the *Library Reference Manual* for documentation.

The "TUTORIAL LIBRARY" drawing includes the first version of every part in the tutorial library. It is used for library development.

In summary, the tutorial library is made up of one directory for each part in the library, two additional directories that each contain a reference drawing, and three files. Each Valid-created library follows this pattern. Under each of these directories is a group of files that together define the library part. Refer back to Figure 14-1 to see the directories, subdirectories, and files of the tutorial library.

A Sample Library Part: DFF

You learned above that each library part has a UNIX directory allocated to it because each is a GED drawing. Actually, each library part is defined by several related drawings. For the library part named DFF, these drawings are named:

dff.body
dff.part

The part of the name after the dot is the *drawing type*. Drawing types were first introduced in Lesson 3. You need to know about three types for now.

Logic Drawings

First of all, remember that the SUBTRACTOR drawing and the ADDERCKT drawing that you created using GED both have the drawing type "LOGIC." All user-made drawings are given this drawing type by default. You will always want your drawings to have the type LOGIC unless you are doing hierarchical design. This is an advanced topic that is covered in Tutorial II. For now, just think of all user drawings as having the type LOGIC.

Body Drawings

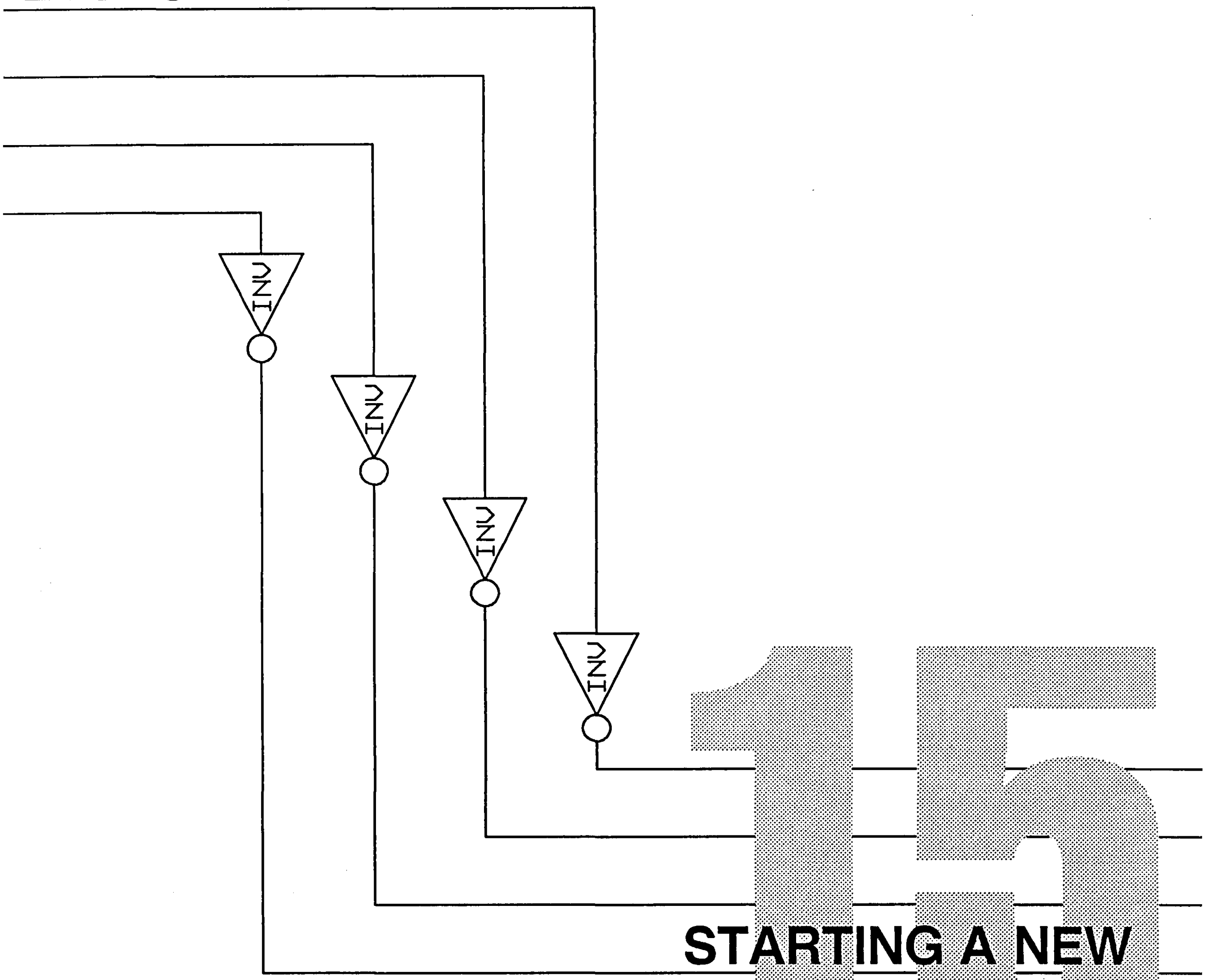
When you add a library part to your drawing, the symbolic representation that appears on the screen is the BODY drawing for that part. When you add the library part DFF, the drawing DFF.BODY appears on the screen. This drawing defines the shape, pins, and general properties of the library part. When you save your drawing (with the **write** command), this information about each library part is saved with the other information on your drawing.

Part Drawings

The PART drawing includes global part information that defines a given part for the Packager. When the Compiler compiles for LOGIC, it includes, for each body on your drawing, this information from the PART drawing. This information is then passed on to the Packager.

There is only one more lesson to this tutorial, *Starting a New Project*. It teaches you two useful procedures for starting a new project, because now you are ready to put all this information to work on a design project of your own.

LESSON 15



STARTING A NEW PROJECT

This is the last brief lesson of this tutorial. To get you started on your work, you learn how to set up a new directory for a new project. This lets you use your SCALD tools to their best advantage.

This lesson covers these new UNIX commands:

New Commands	
cp	**mkdir**

Setting Up a Directory for a New Project

To most effectively use your SCALD tools, you will want to keep each design in a separate UNIX directory. Within this directory you will want to make a single SCALD Directory. If you do so, then your SCALD Directory will contain just the drawings for a single project and there will be no possibility of confusion.

To set up a directory to start on a new project, follow these steps:

1 Move into your home directory (if you are not already there):

```
cd
```

2 Make a new directory named "proj1" under your home directory:

```
mkdir proj1
```

3 Copy your default files into this new directory. In UNIX, the best way to do this is to stay in your home directory (where these files are located) and to use the *cp* (copy) command. The command

```
cp *cmd startup.ged proj1
```

copies all of your command files (all files ending with the letters *cmd*) and the *startup.ged* file into the directory "proj1" using the same file names.

4 Go into the new directory:

```
cd proj1
```

5 Edit the *startup.ged* file:

```
vi startup.ged
```

6 Change the "use" line to read:

```
use proj1.wrk
```

7 If you are taking advantage of the *master.local* file, edit the "masterlibrary" line to include the UNIX pathname to the *master.local* file in your home directory:

```
masterlibrary ../master.local
```

8 Add library commands for all the libraries you need for this project.

9 Save your new *startup.ged* file:

```
:wq
```

10 Use the editor *(vi)* to change the name of your SCALD directory in the *compiler.cmd* and *td.cmd* files to read:

```
directory `proj1.wrk´;
```

Tips on the Copy Command

✓ Where do you want to be when you run the UNIX *cp* (copy) command?

A good rule is: To copy files into a destination directory that is under the source directory, use the *cp* command from the source directory.

To copy files into a destination directory that is elsewhere, use the *cp* command from the destination directory.

✓ Wildcard characters.

The * (asterisk) in Step 3 is a wildcard character. In UNIX commands, an asterisk stands for any string. The string ***cmd** therefore matches all strings that end in *cmd*. UNIX looks through your current directory for all files that end with *cmd*, and performs the command (in this case, *cp*) on them. Using an asterisk is a powerful and quick way to make UNIX do a lot of work for you. But if you do not think carefully before using the asterisk, you can get into a lot of trouble.

For more information on wildcard characters, see the UNIX documentation included with your workstation.

✓ Copy command syntax.

The UNIX *cp* command accepts either two file names, like this:

```
cp file1 file2
```

or a file name followed by a directory name, like this:

```
cp file1 dirname
```

or even a list of files followed by a directory name, like this:

```
cp file1 file2 filex dirname
```

The copy command can distinguish your existing file names from your directory names. Spaces delimit the names. Full or partial UNIX pathnames may be used for any of the file or directory names.

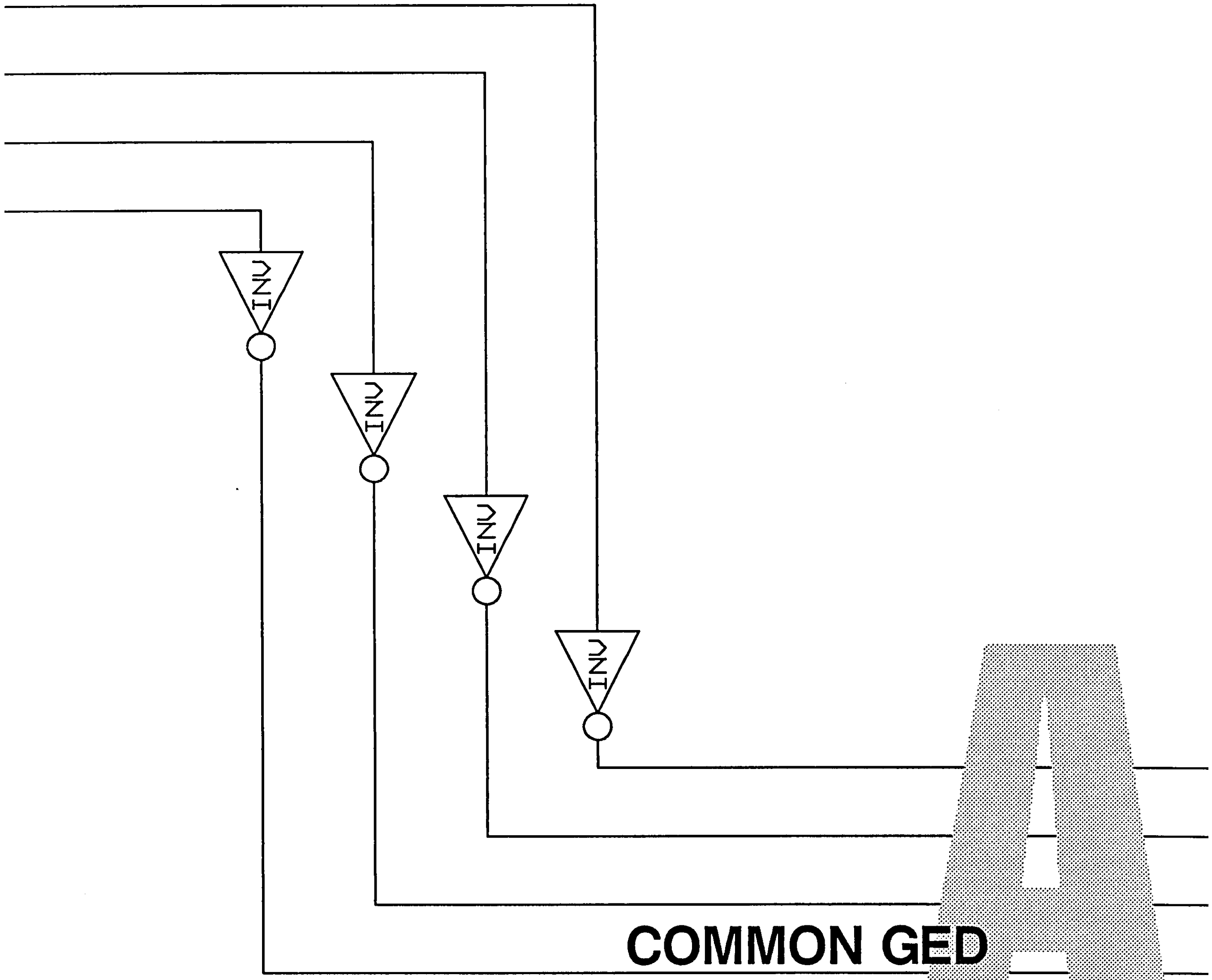
Now you are ready to start a new project. When you log in, connect to your "proj1" directory (**cd proj1**) and then just type

```
ged
```

to go into GED and start drawing.

This concludes Tutorial I. You are now well on your way to becoming a proficient SCALD user. Keep this tutorial nearby and refer to it as needed for specific procedures.

APPENDIX A



COMMON GED COMMANDS

A

The Graphics Editor (GED) commands are case insensitive so that you can enter them in either upper or lower case. To enter the Graphics Editor, at the UNIX prompt type:

```
ged
```

To exit from GED, type:

```
quit
```

or

```
exit
```

The following commands are used for drawing and for checking your drawing. The portion of the command name that is **underlined** is the smallest command abbreviation that can be entered; the applicable lesson numbers appear in parentheses following the command description.

add	Add a body (library part) to the drawing (2,6)
auto dot	Add dots to all wire junctions recognized by GED (6)
backannotate	Add the U-numbers and pin numbers assigned by the Packager to a drawing (10)
check	Check for wiring errors in a drawing (6)
change	Invoke an editor to change text strings (9)
copy	Add a copy of an object or group of objects to the drawing (2,6)

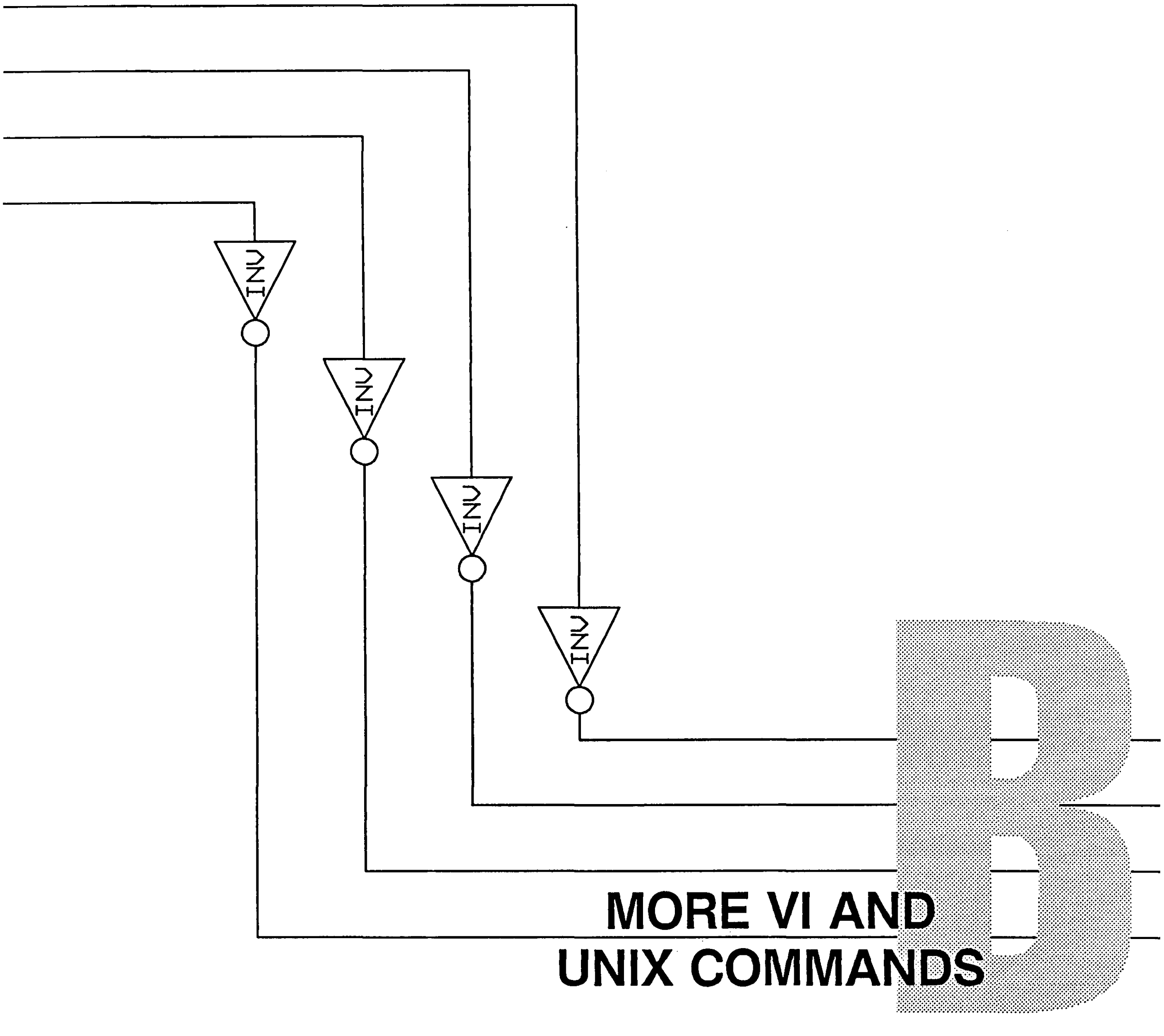
delete	Delete an object or group of objects from the drawing (2)
display both	Display both the property name and property value (6)
display invisible	Suppress both the property name and property value from being displayed on the drawing (9)
display visible	Return a property value to visible after the **display invisible** command; displays only the property value (9)
dot	Add a dot to the drawing (6)
error	Display location and type of each error found by the **check** command (6)
masterlibrary	Allows you to specify your own library–like abbreviations for SCALD directories (12)
move	Move an object or group of objects by sliding them across the screen (2)
pinswap	Swap pin numbers on bodies that are defined to be in the same pin group (10)
property	Specify a property name and property value and attaches this pair to a body, wire, or pin (6,9)
reattach	Change the object to which a given signal name or property is attached (6)
rotate	Rotate a body 90 degrees at each click (6)
section	Assign a logical part to a particular section of a physical part (10)

select	Provide a stretchable rectangle to specify the boundaries of a group (2)
set dots_filled	Make all subsequently–added dots appear as filled dots (6,7)
show attach	Display objects to which signal names and properties are attached (6,9)
show connect	Highlight all connection points on wires with a temporary asterisk (2,6)
signame	Attach a user–defined signal name to the specified wire (2)
split	Separate joined objects and divides wires into two segments (6,10)
undo	Undo the last command issued; repeated use returns step by step to state of drawing at last read or write (2)
version	Select an alternate version of a body; repeated clicks cycle through all available versions (6)
wire	Add wires to the drawing; grid points or pins may be specified as end points (2)
write	Save a drawing by writing it to disk (2)
zoom	Alter the view of the drawing; many options are available (2,4,5)

The following commands are used to edit drawings, borrow drawings, and keep track of your drawings. The portion of the command name that is underlined is the smallest command abbreviation that can be entered; the applicable lesson numbers appear in parentheses following the command description.

directory	List your current SCALD directory, your search stack, or the drawings within them (3,12)
edit	Access a drawing and brings it to the screen (3)
hardcopy	Send a drawing to the plotter (2)
library	Add a library to the bottom of your search stack (1,12)
remove	Remove a drawing from your SCALD directory (12)
use	Add a SCALD directory to the top of your search stack (12)

APPENDIX B



MORE VI AND UNIX COMMANDS

Vi Commands

To edit a file:	vi *filename*
To move around in a file:	
Forward one line	$\boxed{\text{Return}}$
Backward one line	– (minus sign)
Forward one word	w
Backward one word	b
End of current word	e
Forward one char	$\boxed{\text{Space Bar}}$
Backward one char	$\boxed{\text{Backspace}}$
Beginning of file	1G
End of file	G
Line number *x*	*x*G (e.g., go to line 5 = 5G)
To delete:	
Delete a line	dd
Delete a word	dw
Delete a character	x
Delete to end of line	D
To insert text:	
Before cursor	i
After cursor	a
Below current line	o
Above current line	O
To stop inserting:	$\boxed{\text{Escape}}$
To change a word:	cw*new text*$\boxed{\text{Escape}}$

To replace a single character:	r*new character*
To replace several characters:	R*new characters* Escape
To exit:	
With write	**:wq**
With write	**ZZ**
Without write	**:q!**
To save without exiting:	:w
Current line number: :	**.=**
To search for a string: (forward search, automatic looparound)	*/string*
To search for a string: (backward search, automatic looparound)	*?string*
Next occurrence of string:	**n**

UNIX Commands

Looking at files:

ls *dirname*	List the files and subdirectories in your current directory
ls -s *dirname*	Same as *ls*, but also gives you the size of each file
ls -l *dirname*	Same as *ls*, but gives you extensive information, including permissions and date of last modification
more *filename*	List a file to the screen a screenful at a time (press the Space Bar for another screenful; press **q** to quit; press Return for next line)

Changing directories:

pwd	Print name of current directory
cd	Connect to home directory
cd *dirname*	Connect to *dirname* under current directory
cd ..	Connect to next highest directory

Moving files:

cp *file newfile*	Copy *file* to *newfile*
mv *file newfile*	Move or renames *file* to *newfile* (if *newfile* is an existing file, it is overwritten)
mkdir *dirname*	Make a new directory named *dirname*

Removing files:

rm *file*	Remove *file* from system
rmdir dirname	Remove *dirname* (only works on empty directories, remove files first)

Other commands:

file *file*	Report whether *file* is a directory or a file
lpr *file*	Print UNIX text file
passwd *user*	Change or installs a password for *user*
who	List names of users currently logged on

APPENDIX C

DEFAULT FILES IN LOGIN DIRECTORY

The following files and directories are in your login directory:

adderckt	**startup.ged**
***case.dat**	**subtractor**
compiler.cmd	*susan*.**wrk**
***delay.dat**	***td.cmd**
master.local	***verifier.cmd**
packager.cmd	**xshadowx**
***simulate.cmd**	

Each GED drawing (**adderckt** and **subtractor**) is a UNIX directory that contains files. You should not make any changes to the files in these directories directly from UNIX. If you do, GED will not be able to function properly.

xshadowx is also a UNIX directory. This directory is created the first time you run the Compiler (i.e., when it is called by the Packager) and contains other directories that contain intermediate compilation result files. The information is these files is either used or updated during subsequent compilations.

The other names listed are all files. The file names noted with an asterisk (*) support the Timing Verifier and Logic Simulator analysis tools. The function of each of the above files is listed below.

case.dat This file is used to enter data to the Timing Verifier when you want to test the timing for specific cases.

compiler.cmd This file is the command file for the Compiler.

delay.dat	This file is used to feed back delay data to the Timing Verifier from a physical design system.
master.local	This is the default file specified by the **masterlibrary** command. The file contains short forms or abbreviations for SCALD directories that you regularly use.
packager.cmd	This file is the command file for the Packager.
simulate.cmd	This file is the command file for the Simulator.
startup.ged	This file is used to tell GED which directories and libraries you want automatically connected every time you use GED. You also enter any other GED commands you want set as defaults (for example, **set dots_filled**).
susan.wrk	This file is your SCALD Directory. It is a small index file that GED creates for its own use. Each time you save a drawing, GED writes the name that you give the drawing and the corresponding UNIX name into this file. (The UNIX name is the directory where the drawing is stored.) This way, GED can retrieve the drawing for you whenever you want it. Do not edit this file manually. If you do, GED will not work properly. GED will not be able to find your drawings!
td.cmd	This file is the command file for the Plottime program that plots timing diagrams generated by the Timing Verifier or Simulator.
verifier.cmd	This is the command file for the Timing Verifier.

Index