Schematic Entry User Manual

SYNARIO

Universal FPGA Design System

Synario is a Data I/O Product

August 1994 090-0602-001

Data I/O Corporation
10525 Willows Road N.E., P.O. Box 97046
Redmond, Washington 98073-9746 USA
(206) 881-6444

Acknowledgments:

Table of Contents

Preface

1. Getting Started

2. Inside SCS

3. Introduction to Hierarchical Design

4. Basic Operation

5. Using the Schematic Editor

6. Using the Symbol Editor

7. Using the Hierarchy Navigator

8. Attributes

9. The SCS INI Editor

10. PCB Design Considerations

Appendixes

A. Generic Interfaces

B. Simulator Interfaces

Index

Preface

This manual describes the *Synario Capture System*™ (SCS™). It is divided into two sections. The first section presents the "what" of SCS, the last section the "how." You don't have to read the first section in order to use SCS. However, it has background information that should increase your understanding of SCS's features and let you take fuller advantage of them. Chapter 4, "Basic Operation," explains how to configure SCS.

Using This Manual

Synario Capture System Basics (Chapters 2 and 3)

This section gives a basic explanation of how SCS organizes symbol, schematic, and netlist files, and the ways in which SCS manages your designs as you enter them.

How to Use the Synario Capture System (Chapter 4 through Appendix B)

This section explains how to use the various Capture System components: the Symbol and Schematic Editors, the Hierarchy Navigator, the INI Editor, and the netlister programs.

Other Documents

See the *SCS Command Reference* for information on all commands available in the Schematic Editor, Symbol Editor and Hierarchy Navigator.

See the *Waveform Tools Manual* for information on the Waveform Viewer or Waveform Editor.

Chapter Contents

The following is a brief outline of the contents of each chapter.

Chapter 1, Getting Started

Explains how to install and license the SCS software for SCS (Windows) and ECS (Unix) product. If you have Synario, SCS is installed during Synario installation.

Chapter 2, Inside SCS

Explains what an SCS symbol is, and how its characteristics and behavior are controlled by attributes. Describes the components that make up an SCS schematic and how they relate to netlists.

Chapter 3, Introduction to Hierarchical Design

Shows how SCS schematics can be combined into a hierarchical structure, and describes the advantages of hierarchical design. Three approaches to hierarchical design are explained. Also describes the principles of hierarchical organization used in the Schematic Editor and Hierarchy Navigator.

Chapter 4, Basic Operation

Discusses the interface and common features of the Schematic and Symbol Editors. Also explains how to configure SCS Executive program and explains the principal features of SCS.

Chapter 5, Using the Schematic Editor

Covers most of what you need to know to add symbols and wiring to a schematic. Explains what nets and buses are, how to create them, and how to name them.

Chapter 6, The Symbol Editor

Shows how to create your own symbols and attach attributes to them. Block symbols (symbols that represent modules or sub-circuits) are also described.

Chapter 7, Hierarchy Navigator

Shows how to view and probe each level of a design using the Hierarchy Navigator. Explains the operation of the Electrical Rules Checker and the Critical Path Viewer.

Chapter 8, Attributes

Explains what an attribute is, and how attributes are assigned to symbols, pins, and nets. Derived attributes, which permit dynamic modification of a design, are also described.

Chapter 9, SCS INI Editor

Explains how the Schematic and Symbol Editors, Hierarchy Navigator, Waveform Viewer and Waveform Editor are configured by changing values and settings with the INI Editor.

Chapter 10, PCB Design Considerations

Explains the differences between PCB and IC designs. Shows how to create netlist files in a variety of formats.

Appendix A, Generic Interfaces

Describes the general-purpose interfaces that are most often used to move the design between systems.

Appendix B, Simulator Interfaces

Describes the interfaces used to output the design to a simulator.

SCS Components

This manual covers the following SCS/ECS components:

SCS Executive

The Executive is a shell for SCS, usually used with the UNIX version, but available from Synario. You can run any of SCS components, as well as any additional design tools of your own choosing, by selecting them from the Executive.

Schematic Editor

The Schematic Editor is SCS's schematic-capture tool. It creates schematic (**.sch**) files that can represent a complete design, or any component of a hierarchical design.

Symbol Editor

SCS comes with a standard symbol library for IC or PCB design. The Symbol Editor is used to create symbols or primitive elements that represent an independent schematic module. You can also create decorative symbols (such as title blocks).

Hierarchy Navigator

The Hierarchy Navigator combines all the components of a multi-level design for viewing and analysis. You can traverse the full design, viewing each component in its full hierarchical context.

INI Editor

The INI Editor modifies SCS INI files, which control the appearance and behavior of SCS. Using the INI Editor, you can set your own preferences for how SCS environment looks and behaves. The INI Editor also manages attributes, which define the electrical behavior of symbols, pins, and nets.

Waveform Viewer

The Waveform Viewer (optional with SCS/ECS standalone) is used to view the results of simulation. You can display the waveform of any net in your design. The Viewer is fully interactive with the Hierarchy Navigator; clicking on a net in the schematic automatically displays the waveform. Waveform Viewer operation is covered in the *Waveform Tools Manual.*

Waveform Editor

The Waveform Editor (optional with SCS/ECS standalone) lets you create the stimulus waveforms for simulation graphically, by drawing directly on the screen. The stimuli can be edited graphically, or by modifying values in dialog boxes. The Editor then converts the waveforms into a stimulus file that your simulator recognizes. Waveform files are also useful as input to automatic test equipment, or as documentation of the circuit's expected behavior. Waveform Editor operation is covered in the *Waveform Tools Manual.*

Chapter 1
Getting Started

If You're a Current ECS User

If you're upgrading from a previous version of the Engineering Capture System (ECS), the Microsoft Write file **schnotes.wri** (in the Windows version) and the **schnotes** text file (in the UNIX/Motif version) has information about new and improved features, and anything you may need to take into account when working with existing projects under the new software. It also has any information that could not be included in the printed documentation.

Installing SCS

The first part of this section covers Windows installation, the second UNIX/Motif installation.

Note: *If you purchased SCS with Synario, SCS is installed during the base Synario installation. You do not need to install it separately.*

Windows Installation

To install SCS or an SCS option:

1. Connect the SCS Security Device (or verify that it is connected).
2. Install the product software.
3. Create a Registration and Enable Code Request form.
4. Send the Registration and Enable Code Request form to Data I/O.
5. Enter the permanent Enable Codes after receiving them from Data I/O.

These steps are explained in the following sections.

Connecting the SCS Security Device

On DOS/Windows machines, SCS products are licensed using a Security Device that attaches to your computer's parallel (Centronics) port and contains an identifier unique to your installation. In addition, each SCS product is individually licensed through its own Enable Code. SCS products can run only on computers that have the appropriate Security Device attached.

The Security Device comes with the SCS product.

To connect the Security Device:

1. Locate a parallel (Centronics) port on your computer. On a DOS machine, there may be more than one (LPT1:, LPT2:, LPT3:). SCS searches all the parallel ports, so you can connect the Security Device to any one of them. The Device does not have to be connected to the same port as the printer.
2. If you are connecting the Security Device to the same port as the printer, disconnect the printer cable.
3. Connect the male connector of the Security Device to the parallel port of your computer. Tighten the screws to firmly attach the Security Device.
4. If you disconnected your printer from this port, connect the printer cable to the female connector of the Security Device.

Note: *If you have other security devices, add the Data I/O Security Device to the group. The Security Device for SCS does not replace any security devices you have for other Data I/O products.*
If you have a Xilinx security device, it must be the farthest device from the computer. If you have the MAX/FLEX Device Kit or the MAX+plus II software, the MAX+plus II Software Guard must be the device closest to the computer.

Note: *The Security Device works whether or not a printer is connected to it. However, on some computers, the printer must be turned on and be "on-line" while the program is running, or the Security Device will not be recognized. If you don't want the printer turned on all the time, you can put the Security Device and the printer on separate ports.*

Installing the SCS Software

To install the software:

5. Insert Disk 1 into your disk drive.
6. From the Windows Program Manager, select Run from the File menu.
7. Enter

   drive:**setup**

   where *drive* is the letter of the disk drive in which you inserted the installation disk (A or B).
8. Follow the instructions on the screen. (They vary depending on which product is being installed.)

Important! *The distribution disks include all Synario Capture Components, including options you may not have purchased. Do not install them if you are trying to maximize your free disk space.*

Note: *In general, you do not need to reboot your computer after installation. The installation program tells you if this is required.*

User Notes and Comment Forms

Important information about the product is placed in the installation directory during installation. These files are in Microsoft Write format (**.wri**) and can be found in the **readme** subdirectory of the installation directory.

The SCS **readme** file is installed as an icon in the Synario program group. This file contains last-minute release information and describes other files installed in the readme directory. Double-click on the icon to load it into Microsoft Write.

Creating a Registration and Enable Code Request Form

You need to register your SCS products with Data I/O to obtain Enable Codes that permanently license your software. You use the Data I/O Registration Editor to register SCS products and enable a 30 day license while you wait for your permanent license codes. The Data I/O Registration Editor is installed as an icon labeled “Data I/O Registration” in the SCS program group.

You can install all the Synario products you purchased, then register them. You do not need to register them as they are installed.

Registering Your Synario Product

All Synario Capture products are registered the same way, whether they come with your initial purchase, or you install them later:

1. Confirm that the Security Device is attached to the computer.
2. Select the “Data I/O Registration” icon from the program group. The Customer Information dialog box is displayed.
3. Click the down arrow in the Security Device combo box and highlight the Security Device Number you want to use. (There is normally only one number, unless you have more than one Data I/O Security Device connected to your computer.)
4. Enter the required information in the edit boxes (Name, Title, Company, and so on). All fields not marked "Optional" must be filled in. If you do not have a fax machine, type the word “None” in the Fax Number field.

   The information you enter allows Data I/O to create the Enable Codes to permanently activate Synario products for your Security Device. The information also helps Data I/O provide better product support, timelier product information, and prompt information about upgrades and new releases.

Note: *If your Shipping Address is different from your Mailing Address, be sure to enter it to ensure timely delivery of updates and upgrades.*

Note: *Be sure the information is complete and correct. If Data I/O needs to contact you for additional information, delivery of your permanent Enable Codes may be delayed.*

5. When all required information has been entered, click the Licensing button. (If the button is not enabled, one or more required items are missing.)
6. The Product Information dialog box is displayed. The Product Description list box shows which products are currently installed (whether or not they are registered).

Note: *If the installation directory is on a network and the directory is shared, products you have not purchased may be listed. However, without a Serial Number (described later) for those products, you cannot obtain the Enable Code for a Permanent license. You can, however, obtain a limited 30 Day license to evaluate those products.*

7. Select the product to register. The fields in the Serial Number Card section change to match the Licensed Software title that appears on your Serial Number Card for that product.
8. Double-click on the product line or click the Enable button. The License Information dialog box appears.
9. Fill in the Serial Number field from the Serial Number Card included in your product documentation. This number is required by Data I/O to identify and verify the product that you purchased.
10. If you want to use the software immediately, select the 30 Day License from the License Type section and click the OK button.

Note: *A 30 Day license can be used only once for each product with your Security Device; it cannot be renewed.*

11. Repeat steps 7 through 10 until all Serial Numbers have been entered.
12. Some products require additional licensing information. When those products are selected from the product list, the Extended button is enabled. See the documentation for those products for instructions on the Extended License Information dialog box.
13. After registering all software, click the Close button to return to the Customer Information dialog box.
14. Click the Print button to print the Registration and Enable Code Request form.
15. Click the Close button to exit the Registration Editor.

Sending the Registration Form to Data I/O for Processing

The Registration and Enable Code Request form contains all the information you entered into the Registration Editor. This form must be sent to Data I/O to obtain your Enable Codes. The methods are listed in order of increasing response time (that is, the first method is the fastest):

- Send the form by fax to the number at the top of the form. Data I/O returns your permanent Enable Codes to the fax number given in the Customer Information section.
- Call the telephone number at the top of the form during the hours listed with the form available. Data I/O will take the registration information over the phone and return your call with your permanent Enable Codes.
- Send the form by mail to the address at the top of the form. Data I/O will process the form and return the permanent Enable Codes by return mail to your Mailing Address.

Entering Permanent License Codes

Once you receive your Enable Codes from Data I/O, enter them as follows:

1. Confirm that the Security Device is attached to the computer.
2. Run the Data I/O Registration Editor from the program group.
3. Select the Licensing button to display the Product Information dialog box.
4. Highlight the product you want to enable. Double-click on the product name or select the Enable button. The License Information dialog box is displayed.
5. Click the "Permanent License" radio button in the License Type section.
6. Enter the Enable Code in the Enable Code Entry field.
7. Click the OK button to exit the License Information dialog box and return to the Product Information dialog box.

Note: *If the Enable Code does not match the other licensing information, you will receive a warning that the Enable Code is invalid. Return to the License Information dialog box and check that the Enable Code was entered correctly and for the correct product.*

8. Repeat steps 4 through 7 until all Enable Codes have been entered. The Enable Code Status in the product list will change to Permanent. After all codes have been entered, select the Close button to return to the Customer Information dialog box.
9. Select the Close button to exit the Registration Editor.

UNIX Installation

Follow these steps to install the Engineering Capture System:

1. Create a directory for ECS.

   ```
   mkdir /usr/ecs
   ```

2. Make the new directory the working directory.

   ```
   cd /usr/ecs
   ```

3. Insert the distribution tape in the tape drive.

4. Copy the contents of the tape with a command similar to the following example:

   ```
   tar xv /dev/rst0
   ```

   The following directories are created:

bin	Motif executable files
data	Help files and master **ecs.ini** file
examples	Example circuits
sym_libs	SCS symbol libraries
mod_libs	SCS model libraries
xfonts	Motif fonts
license	SCS security files

5. Add the **bin** directory to your search path. You can make a temporary assignment with the following command:

   ```
   set path=($path /usr/ecs/bin)
   ```

   You can make the assignment permanent by setting the path in the **.cshrc** file.

6. Add the environment variable ECS_ROOT pointing to the main directory:

   ```
   setenv ECS_ROOT /usr/ecs
   ```

   You can make the ROOT variable permanent by adding it to the **.cshrc** file.

7. The **license** directory contains the following files:

lmgrd	License Manager daemon
dataio	Application Licensing daemon

8. Request a license from Data I/O. See the *ECS User Notes*.

When you receive your license codes, you should create a **license.dat** file in the **/usr/local/flexlm/licenses** directory on each machine that will be running ECS. If you would rather locate it elsewhere, add a setenv command to the **.login** or **.cshrc** file that defines the path as:

```
setenv LM_LICENSE_FILE full_path
```

If you are using other software that uses the Highland Digital network licensing scheme, you can combine the ECS **license.dat** file with the **license.dat** file from the other software.

9. Install the License Manager daemons on the server machine. The server machine's *hostid* should be listed on the SERVER line of the **license.dat** file.

10. Copy the daemons to the **/etc** directory on the server machine, as shown below. You need to be logged in as "root" to be able to write to this directory.

    ```
    cp -p lmgrd /etc/lmgr
    ```

    ```
    cp -p dataio /etc/dataio
    ```

11. Start the daemon by issuing one of the following commands:

    ```
    /etc/lmgrd > /dev/null &
    ```

    (**license.dat** in **/usr/local/flexlm/licenses**)

    ```
    /etc/lmgrd -c full_path_license.dat > /dev/null &
    ```

    (**license.dat** in other directory)

12. Add the following lines at the end of the startup file **/etc/rc.local**:

    ```
    if [ -r /etc/lmgrd ]; then
            /etc/lmgrd > /dev/null &
    fi
    ```

13. You must unpack the symbol and model libraries before using them. They are shipped in the "ar" format to reduce space and loading time. The **listall** and **expand** scripts in the **mod_libs** and **sym_libs** directories are used to list and set up the libraries. Read the comments in the script files for additional information.

Installation is now complete. Exit the root directory and log on as a user.

Chapter 2
Inside SCS

SCS is a schematic capture system. Unless you're using SCS just for documentation, the schematics are actually the starting point of the development process, not the goal. The schematic will eventually be used to analyze the device's behavior (using a simulator and the Waveform Viewer) or to create a printed circuit board.

The schematic file describes your circuit in terms of the components used and how they connect to each other. The schematic is in SCS's own format, and can be used to create netlists in different formats that are read by other development tools, such as simulators and board layout programs.

Symbols are the most basic elements of a schematic. Symbols represent primitive design elements, whether those elements are individual transistors, complete gates, or a complex IC. A symbol can also be the hierarchical representation of a subcircuit (a "Block" symbol).

This chapter explains what an SCS symbol is, and how symbols are combined with wiring (connectivity) to produce useful schematics. It covers the following topics:

- Symbols
- Attributes
- Schematic elements
- Schematics relation to netlists

Symbols

In this discussion, "symbol" refers to an electrical symbol, such as a gate or a subcircuit. You can draw graphic-only symbols (such as title blocks) with the Symbol Editor, but these have no electrical meaning.

Symbol Libraries

Symbol files are usually organized into libraries or library directories. The basic SCS package comes with libraries of "generic" symbols. Any optional device kits you purchase may come with their own symbol libraries.

Any symbols you create are usually stored in the same directory as the project for which they were created. However, you might want to create your own library directories for symbols used in more than one design. These libraries can be added to the Symbol Libraries Search Paths using the INI Editor.

What Does a Symbol Consist Of?

Each SCS symbol is a file ending with a **.sym** extension, and may be included in a library file with a **.lib** extension. The symbol file contains four types of information: *graphics*, *text*, *pins*, and *attributes*.

- *Graphics* are instructions that tell the Symbol Editor, Schematic Editor, and Hierarchy Navigator how to draw the symbol.
- *Text* labels the symbol, or adds supplemental information.
- *Pins* provide electrical connection between the symbol and the schematic's wiring.
- *Attributes* are parameters that describe the symbol's electrical behavior, the symbol's component parts (for example, its pins), and a number of other useful characteristics.

The following sections explain graphics, pins, and attributes in more detail.

Graphics

Graphics are pictures of the symbols. Symbol graphics have no electrical meaning, showing only the position of the component in the circuit. The electrical behavior of a symbol is defined by its attributes and pins, not the graphics that represent it. Explanatory or descriptive text displayed with a symbol is also considered "graphic" information without electrical meaning.

Pins

Symbol pins are the connecting points between the symbol and the schematic wiring. If the symbol represents an individual component, the symbol pin represents the physical pin where a conductor can be attached. If the symbol represents a subcircuit (block symbol), the symbol pin represents a connection to an internal net of the subcircuit.

Attributes

Attributes associate data items with symbols, pins, and nets. ("Nets" are schematic wiring. Net attributes are explained in more detail later in this manual.) The data items describe the electrical characteristics (or other properties) of the symbols and their pins.

An attribute has a name and a value. You can assign or change the values of most attributes at any point in the development process. You can assign some attributes fixed values that cannot change (for example, part numbers). Some attribute values, such as the vendor's part number and the simulation model, are assigned when a symbol is created. These values are automatically applied to all instances of that symbol. You can assign, change, or override other attributes later in development. You can change individual attribute values to achieve a more accurate simulation, since the model can reflect the exact circuit conditions (such as loading and delay) of separate device instances.

A symbol's attribute set is its single most important component. Without attributes, simulation and modeling programs would know nothing about the electrical behavior of the symbol. The second section of this chapter, "Attributes," introduces the characteristics of attributes and their use. For a full discussion of attributes and attribute use, see the following pages and Chapter 8, "Attributes."

Attributes

An attribute is a characteristic or property associated with a symbol, pin, or net. Attributes can describe:

- Width or length of transistors
- Price of a resistor
- Size of a chip or a cell
- Number of connections to a Block symbol
- Delay from input to output
- Number of pins on a package
- Dielectric material of a capacitor
- Length of time taken to design a symbol
- Paint color of a resistor

In short, attributes can describe *anything* in a design.

Attribute Types

There are four attribute types:

Global — Global attributes are constants such as feature size, supply voltage, or identification codes. These attributes are accessible from every sheet of every schematic at every level of hierarchy.

Symbol — Symbol attributes describe features related to the whole symbol. Examples are the width and length parameters of transistors, or SPICE model characteristics. Symbol attributes apply only to the symbol on which they appear. Global attributes (which define such characteristics as supply voltage) apply to all symbols in a design.

Pin — Pin attributes describe features related to individual pins. Polarity, lead number, drive capability, and loading are typical pin attributes.

Net — Net attributes describe characteristics associated with nets. A good example is the stray capacitance of a net routed across a chip. (Nets are part of schematics, not symbols. Net attributes apply only to schematics.)

Attribute Components

An attribute has five components.

Name

An attribute's *name* identifies it to the user. Width, Length, ReferenceDesignator and PinNumber are examples of attribute names.

Number

The attribute's *number* identifies it to the Editors and the Hierarchy Navigator. SCS uses the number—*not* the name—to reference an attribute. This allows a different name to be assigned without changing the meaning or use of the attribute. (Attributes 0–99 are reserved for SCS, the Editors, the Hierarchy Navigator, and simulation. Most of them have predefined meanings.)

Value and Value Modifier

Any attribute can be assigned a *value*. A value is usually a number or a text string.

An *attribute modifier* specifies the conditions under which an attribute's value can be changed. Attribute modifiers are fully described in Chapter 9, "The SCS INI Editor."

Window

Attribute values are displayed in *attribute windows.* Attribute values cannot be displayed unless a symbol has at least one attribute window.

You add attribute windows to a symbol when you define it. Each window is assigned a unique number and the default attribute that will be displayed in that window. When the symbol is placed in a schematic, the value of the assigned attribute appears in the window.

Note: *Attribute windows do not have visible outlines. Rather, they are predefined areas on or near the symbol.*

Creating New Attributes

The generic symbols supplied with SCS (as well as device-specific symbols) have all required attributes defined and given appropriate values.

In addition, each symbol has about 100 undefined attributes. You can define these attributes for any purpose, such as providing additional information to a netlister, or displaying device data in an attribute window.

If you create your own primitive symbols, you must define the attributes needed by the netlister or simulator. Chapter 8, "Attributes," and Chapter 9, "The SCS INI Editor," cover this in detail.

Schematic Elements

A schematic is composed of the following five items:

- Symbols — These can be symbols from the standard SCS libraries, symbols representing other schematics you have drawn (Block symbols), or symbols you have created from scratch.
- Wires — Wires connect the symbols. They can be single-signal ("nets") or multiple-signal ("buses").
- I/O Markers — I/O markers show where signals enter or exit the schematic, and the direction ("polarity") of the signal (that is, whether it's an input, output, or bidirectional).
- Graphics & Text — Graphics and text are usually added to display explanatory data. They are optional and have no electrical meaning.

A schematic must contain the first three components—symbols, wires, and I/O markers. A single, isolated component symbol cannot be the only element in a schematic. The schematic must include I/O markers for the external connections to the schematic, and these markers must be connected to the symbol with wires. Figure 2-1 and Figure 2-2 are examples of valid and invalid schematics, respectively.

Figure 2-1
Valid Schematic

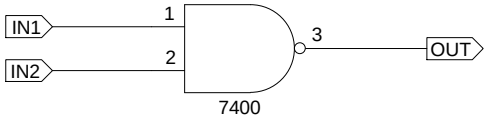
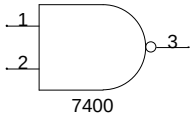


Figure 2-2
Invalid Schematic (no wires or I/O markers)



The following sections discuss schematic elements in more depth.

Symbols

Symbols are graphic representations of components. They have no electrical meaning.

As you place each symbol, the Schematic Editor automatically gives the symbol a unique *instance name* of the form I_*nn* (where *nn* is an integer). The instance name identifies the symbol to the Schematic Editor and netlister programs. You can change the instance name, but the Editor won't let you repeat an existing name.

Some schematics (such as bit-slice designs) use repeated arrays of symbols (such as registers or inverters). The Schematic Editor lets you define an *iterated instance* in which a single symbol represents many instances of that symbol.

Symbol Attributes

Each symbol has a number of predefined attributes that describe its part number, component type, and other unchanging characteristics. (These were discussed briefly in the preceding section, "Symbols.")

Other attributes can be given values after the symbol is placed in the schematic. These attributes can have different values for each symbol instance. This permits detailed customization of a design.

A good example of an attribute that permits customization is the attribute that controls the speed / bandwidth characteristics of a macro cell. Cells driving internal nodes might be set for full speed, whereas cells driving heavily loaded external nodes might be set for a narrower bandwidth, to obtain greater drive capability.

Wires

Wires are the lines that electrically connect the symbol pins. Symbol pins are the only connection points for wires. You cannot connect wires to the symbol body itself.

There are two types of wires: single-wire *nets* and multiple-wire *buses*. Buses allow more than one signal to be routed as single line. (Nets and buses are explained in more detail in Chapter 5, “Using the Schematic Editor.”)

Wire Names

Wires have names. These names identify the wires to the Schematic Editor and netlister programs.

You would normally name all wires that connect to inputs or outputs and any “internal” nets with signals you want to view during simulation. You can use any name you like, but you usually choose a name that suggests the name or function of the signal carried by that wire. If you don’t give a wire a name, the Schematic Editor automatically supplies one, of the form N_*n* (where *n* is an integer).

Multi-wire buses are created by giving a single wire a compound name. You can then tap off any signal you want anywhere along the bus.

Buses are most often used to group related signals, such as a 16-bit data path. However, a bus can be any combination of signals, related or not. Buses are especially useful when you need to route a large number of signals from one side of the schematic to the other.

Buses also make it possible for a single I / O marker to connect more than one signal to a Block symbol. The signal names don’t have to match, but both pins must carry the same number of signals.

Net Attributes

Like symbols and symbol pins, nets (the wiring that connects symbols to each other and makes external connections) can also have attributes. These attributes include the net's name (as assigned in the schematic), plus the net's length, width, unit capacitance, and so on. There are also net attributes that pass parameters to other programs such as simulators or PCB layout programs.

I/O Markers

I/O markers mark the points at which signals leave or enter the schematic. They are required. Any unconnected wire without an I/O marker will eventually be flagged as an error when you try to create a netlist, run the simulator, or load the design into the Hierarchy Navigator.

The I/O marker automatically takes the name of the wire it is attached to. If the wire is a bus, the marker will have the same compound name as the bus.

When a Block symbol and its matching schematic are created, the I/O markers for the signals that enter and leave the schematic must have the same names as the corresponding pins on the Block symbol. The matching names identify which signal attaches to which pin.

Graphics

Although symbols, wires, and I/O markers are visible, graphical items, they also have a functional or electrical meaning. In this context, "graphics" refers to the non-functional graphical parts of the schematic.

For example, you might add graphics showing the expected waveforms at different points in the circuit. Or, you could draw the company's logo and add it to each schematic for identification.

The most common use of graphics is to create a title block. The block shows the name and address of your company, and can include the company logo and blank spaces for the project name, schematic sheet number, and so on.

Text

Text, like graphics, can provide additional information about the schematic or its project. Text can be placed anywhere on a schematic, even if it overlaps symbols or wires.

Schematics Relation to Netlists

A *netlist* is a file listing all the components (symbols) in a schematic, and their *connectivity* (how they are wired together).

There is no single, universally accepted netlist format. The Schematic Editor stores a schematic in its own proprietary connectivity database format. This database format can be converted to almost any netlist format. However, data and data structures not supported by the target format will not appear in the converted file.

The conversion is performed with netlister programs. SCS comes with several that can convert schematics to and from a number of standard formats: ASCII, EDIF, and a generic human-readable format. (Additional netlisters are optional.) The ASCII format retains all the information in the original schematic file.

Using Netlists

The netlist is the “interface” between your design and any other software that needs to read or use the design. These programs include

- Simulators
- PCB layout programs
- IC place-and-route software
- Synthesis software
- Signal integrity tools

Chapter 3
Introduction to Hierarchical Design

SCS supports full hierarchical design. Hierarchical structuring permits a design to be broken into multiple levels, either to clarify its function or permit the easy reuse of functional blocks. This chapter covers the following topics:

- What is a Hierarchy?
- Advantages of Hierarchical Design
- Approaches to Hierarchical Design
- What is Hierarchical Organization?
- Symbols, Schematics, and Hierarchy
- Hierarchical Design Structure
- Hierarchical Naming
- Nets in the Hierarchy

What Is a Hierarchy?

A large, complex design does not have to been drawn as a single schematic. The Schematic Editor lets you add as many sheets as needed, so that a design can extend beyond the original sheet. However, regardless of how many sheets you add, all the components of the design are still at a single level.

Another way to organize a design is to break it up into components or modules. Circuitry for a specific function or interface can be drawn as a separate schematic. A Block symbol is then created for this schematic, which can be placed in other schematics as a single symbol.

The schematic represented by the Block symbol is said to be at one level *below* the schematic in which the symbol appears. Or, the schematic is at one level *above* the Block's schematic. Regardless of how you refer to the levels, any design with more than one level is called a *hierarchical* design. In SCS, there is no limit to the number of hierarchical levels a design can contain.

Advantages of Hierarchical Design

The most obvious advantage of hierarchical design is that it encourages modularity. A careful choice of the circuitry you select to be a module will give you a Block symbol that can be reused.

Another advantage of hierarchical design is the way it lets you organize your design into useful levels of abstraction and detail. For example, you can begin a project by drawing a "top" schematic that consists of nothing but Block symbols and their interconnections. This schematic shows how the project is organized, but does not display the details of the modules (Block symbols).

You then draw the schematic for each Block symbol. These schematics can also contain Block symbols that you have not yet drawn schematics for. This process of *decomposition* can be repeated as often as required until all components of the design have been fully described as schematics.

Breaking the schematic into modules adds a level of abstraction that lets you focus on the functions (and their interaction) rather than on the hardware that implements them. At the same time, you are free to view or modify an individual module.

Although there are many ways of "breaking apart" a complex design, some may be better than others. In general:

- Each module should have a clearly defined purpose or function and a well-defined interface.
- Look for functions or component groupings that can be reused in other projects.
- The way in which a design is divided into modules should clarify the structure of the project, not obscure it.

When trying to decide what a module should contain, consider what Albert Einstein might say: "A module should be as complex as it needs to be, but *no more* complex."

Approaches to Hierarchical Design

The Schematic Editor supports full hierarchical design. Project components can be created in any order, then combined into a complete design. You can draw a schematic first, then create a Block symbol for it, or specify the Block first, then create the schematic for it later.

Hierarchical entry is a convenient way to enter a large design "one piece at a time." It is also a way of organizing and structuring your design and the design process. The choice of the appropriate methodology can speed the design process and reduce the chance of design or implementation errors.

There are three basic approaches to creating a multi-module hierarchical design:

- Top-Down
- Bottom-Up
- Inside-Out (“mixed”)

The following three sections explain the philosophy and techniques of each approach.

Top-Down Design

In top-down design, you do not have to know all the details of your project when you start. You can begin at the “top,” with a general description of the circuit’s functionality, then break the design into modules with the appropriate functions. This approach is called “stepwise refinement”—you move, in order, from a general description, to modularized functions, to the specific circuits that perform those functions.

In a top-down design, the uppermost schematic usually consists of nothing but Block symbols representing modules (plus any needed power, clocking, or support circuitry). These modules are repeatedly broken down into simpler modules (or the actual circuitry) until the entire design is complete.

Bottom-Up Design

In bottom-up design you start with the simplest modules, then combine them in schematics at increasingly “higher” levels. Bottom-up design is ideal for projects (such as interfaces) in which the top-level behavior *cannot* be defined until the low-level behavior is established.

Inside-Out (“Mixed”) Design

Inside-out design is a hybrid of top-down and bottom-up design, combining the advantages of both. You start wherever you want in the project, building “up” and “down” as required.

SCS fully supports the “mixed” approach to design. This means that you can work bottom-up on those parts of the project that must be defined in hardware first, and top-down on those parts with clear functional definitions.

Regardless of the approach you choose, you start from those parts of the design that are clearly defined and move up or down to those parts of the design that need additional definition.

What is Hierarchical Organization?

The Schematic Editor uses a hierarchical system to organize complex designs. Like a series of increasingly detailed maps, a hierarchy of schematics lets the designer move from a general view (the entire country) to more-detailed views (counties, cities, and neighborhoods).

At the top level in the hierarchy, the full design is represented as a complete (but relatively undetailed) schematic. As you descend the hierarchy, you see more-detailed views of smaller circuit elements. Primitive elements (library symbols) are visible at the lowest level.

Although hierarchical designs are created in the Schematic Editor, a schematic can be viewed in its full hierarchical context only from within the Hierarchy Navigator. The Navigator lets you view each circuit component in its hierarchical context, and move up and down in the hierarchy. See Chapter 7, “Using the Hierarchy Navigator,” for a detailed explanation.

Symbols, Schematics, and Hierarchy

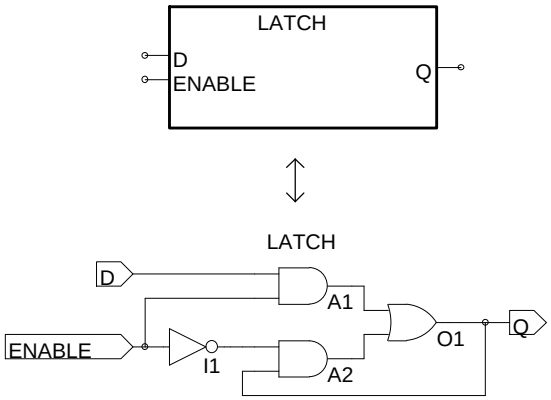
In the Schematic Editor’s hierarchy, a symbol at one level (a “functional block”) represents a more-detailed schematic at the next-lower level. Hierarchical symbols must meet the following three requirements:

1. The symbol must have the same base name as the schematic containing the underlying circuitry. This associates the schematic with the symbol representing it.
2. The pin names on the symbol must match the I/O marker names in the underlying schematic.
3. The symbol should be a Block symbol. If the symbol used is in a model directory, it can also be a Cell symbol.

The Block symbol is associated with its schematic by giving the schematic nets the same names as the corresponding symbol pins. For example, a wire connected to a pin named Q on the symbol is also connected to the net named Q in the underlying schematic. The Consistency Check command of the Schematic Editor and the ERC Check command in the Hierarchy Navigator flag an error if a Block symbol has a pin without a corresponding net in the related schematic.

In Figure Figure 3-1, pin D on the Block symbol corresponds to the net in the schematic, which is named D. The other pins, Q and ENABLE, also correspond to named nets in the schematic.

Figure 3-1
A Block Symbol and Its Underlying Schematic



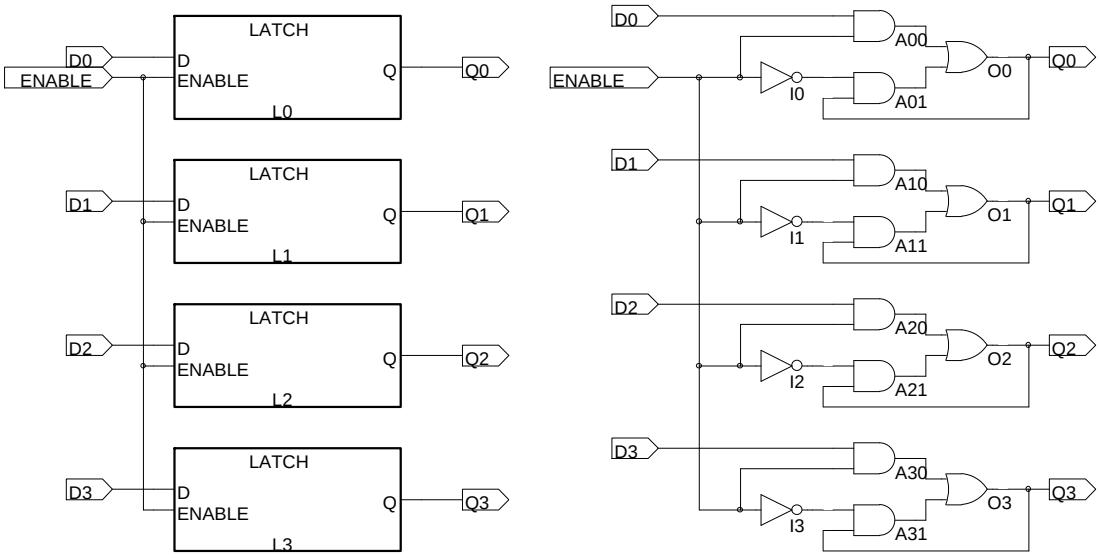
Hierarchical Design Structure

When a symbol is placed in a schematic, the component or sub-circuit the symbol represents is added to the circuit. When you place a latch symbol (for example) you are actually including the OR gate, inverter and two AND gates from the latch's schematic.

Figure 3-2 shows (on the left) a 4-bit register (REG4) constructed from four latch symbols (**latch.sym**). The right side of the figure shows the underlying components. The four **latch** symbols represent a total of eight AND gates, four OR gates, and four inverters.

This hierarchical building process could be repeated by using the Schematic Editor's Matching Symbol command to create a symbol for schematic **reg4**, then placing the **reg4** symbol in a higher-level schematic. If you created a schematic for a 16-bit register, **reg16**, by placing four copies of symbol **reg4**, you would be defining a circuit with a total of 64 gates. But instead of having to view 64 gates on a single level, you can work with symbols that represent gates, at the appropriate level of detail.

Figure 3-2
Circuit REG4 and its Equivalent Circuit



Hierarchical Naming

In the **latch** schematic (Figure 3-2), the inverter has the instance name I1. In schematic **reg4**, four copies of the symbol **latch** are placed and assigned instance names L1 through L4. Schematic **reg4** therefore contains four copies of inverter I1.

The Hierarchy Navigator distinguishes among these otherwise identical inverters by combining the inverter's instance name with the instance name of the latch containing it. The four inverters are therefore named (in the Hierarchy Navigator):

```
L1.I1

L2.I1

L3.I1

L4.I1
```

If we created a 16-bit register by combining four **reg4** symbols (as suggested previously), the resulting schematic would represent a new hierarchical level, containing four copies of **reg4** (named R1 through R4). Each copy of **reg4** contains the four inverters as named above. The Hierarchy Navigator would then name the 16 inverters by combining the instance names of the four **reg4** symbols with each of the four instance names of the inverters as follows:

```
R1.L1.I1, R1.L2.I1, R1.L3.I1, R1.L4.I1

R2.L1.I1, R2.L2.I1, R2.L3.I1, R2.L4.I1

R3.L1.I1, R3.L2.I1, R3.L3.I1, R3.L4.I1

R4.L1.I1, R4.L2.I1, R4.L3.I1, R4.L4.I1
```

When you view an individual latch schematic in the Schematic Editor, you see the instance names of the gates, without the hierarchical context. When the schematic becomes part of a larger design and is viewed in the Hierarchy Navigator, the instance names include the hierarchical path (as shown above) to assure their uniqueness.

Nets in the Hierarchy

The schematic definition for the latch circuit contains both *local* and *external* nets. The output of the inverter is connected to the AND gate with a local net. Two other local nets connect the outputs of the AND gates to the inputs of the OR gate. Assume these nets have been named N1, N2, and N3. When the 16 copies of this circuit are combined in **reg16**, 16 copies of these local nets are created.

The 16 local nets named N1 are individual nets, *not* branches of the same net, so the Hierarchy Navigator creates a unique name for each. The local net name (N1) is prefixed with the instance name of the schematic where the net is defined. A dash separates the net and instance names. The 16 N1s then become:

```
R1.L1-N1, R1.L2-N1... R4.L3-N1, R4.L4-N1
```

The **latch** schematic contains three external nets, D, ENABLE, and Q. The symbol pins on the latch connect these nets to the hierarchical level above.

Automatic Aliasing of Nets

When a design is loaded into the Hierarchy Navigator, nets take the name of the highest (“top”) net in the design. That is, the name of top-level net propagates downward through the hierarchy to override the “local” name. By forcing all nets to the same name, this “aliasing” feature greatly speeds signal tracing in a multi-level design.

In the preceding example, the net name D from the latch is overridden by the higher-level external reference to become D1, D2, D3…. This override becomes the reference at all levels of the hierarchy. If, in the suggested 16-bit register the D0, D1, D2… inputs were connected to wires named and marked Bit0, Bit1, … Bit15, these new names take precedence and the D0, D1, D2… names would no longer be accessible at any level of the hierarchy.

Chapter 4
Basic Operation

This chapter has basic information about the Synario Capture System. It covers the following topics:

- What SCS Can Do
- SCS Programs
- Using the SCS Executive
- SCS Executive Features
- Schematic Exchange / Conversion Utilities
- Customizing the Executive with the pcshell.ini File
- SCS Command Structure
- Using the Mouse
- Prompting and Error Messages
- Error Recovery
- Network Operation
- Naming Design Files
- Saving Files
- Printing and Plotting
- The INI Editor

What SCS Can Do

SCS is a design entry and analysis package for schematic-based IC and PCB designs. It features:

- **Schematic Capture** — The Schematic Editor captures your design logic (either IC or PCB) in schematic form. You can use the schematic files solely for documentation, or as sources for other software that fits your design into ASICs or creates printed circuit boards.

  The schematic file is continuously updated as you draw, so it is always ready for analysis.

- **Netlist Generation** — Your design can be converted into industry-standard EDIF or ASCII netlists for use by other design tools. Additional netlisters are available as an option.
- **Consistency Checking** — You can check your schematics at any time for such errors as unconnected wires or shorted nets. These checks greatly increase the likelihood your design will be correct.
- **Electrical Rules Checking** — Electrical rules checks catch such errors as having too many loads connected to one output. This checking prevents electrical incompatibilities that would keep your design from working.
- **Hierarchical Design** — Designs are not limited to a single schematic. Block symbols can represent a complete schematic or any subsection of a schematic. These symbols can be used to create a hierarchical design, or to create reusable function modules. There is no limit to the number of hierarchical levels a design can contain.
- **Hierarchical Viewer** — The *Hierarchy Navigator* lets you view a multi-level design in its full context. Back annotations can be added as comments or modifications to the design.
- **Symbol Creation** — You can create your own electrical (or decorative) symbols and give them whatever characteristics you want. Or, you can convert a schematic into a Block symbol to make your design easier to understand, or for reuse in other projects.
- **Documentation** — SCS comes with a complete set of generic symbols. You can create schematics to document your design, or produce netlists (in standard EDIF or ASCII formats) for use by other design tools.
- **PCB Interface and Packaging** — SCS can also be used for printed circuit board design. Its packaging features include automatic package assignment, consistency checking, and the ability to back annotate a design with modifications or notes.
- **Simulator Interface** — Most simulators can be used with SCS. EDIF and ASCII netlist generation is a standard feature. Optional netlisters for a variety of simulator formats are available.
- **Waveform Tools** — The *Waveform Viewer* lets you view the results of simulation. The waveform at any node in your design can be displayed. Full cross-probing with the Hierarchy Navigator is supported. Click on a node in the Navigator to view its waveform in the Viewer.

  The *Waveform Editing Tool (WET)* lets you create input stimulus waveforms for your design graphically, by clicking and dragging directly on the screen. The waveform tools are described in the *Waveform Tools Manual.*

SCS Programs

SCS is a program for entering and analyzing schematic-based designs. Designs can be single-level ("flat") or multi-level ("hierarchical"). In all cases, the full design can be converted to a variety of netlist formats for use with other design and production software.

SCS consists of the following principal components:

- **The SCS Executive** — (SCS/ECS Only) This shell program calls the other SCS programs as you need them, and passes any required command line options. It also executes a number of utilities, such as the initialization file editor and netlist converters. You can customize the Executive.
- **The Schematic Editor** — Your schematic designs are captured by this tool. Schematics can be drawn on multiple "sheets" and be of any size.
- **The Symbol Editor** — If you need a symbol that is not in the SCS library, you can create it with the Symbol Editor. You can also design special purpose non-component symbols (such as logos and title bars). The Symbol editor can also create a symbol for any existing schematic, so that that schematic can be used as a module in other schematics.
- **The INI Editor** — The INI Editor configures the initialization ("INI") file for SCS. All default settings and user preferences for the Editors, Hierarchy Navigator, Waveform Viewer, and Waveform Editing Tool are set with the INI Editor. It is also the editor for symbol, pin, and net attributes.
- **The Hierarchy Navigator** — All the elements and levels of a design are combined by the Hierarchy Navigator for viewing and analysis. Analysis is fully interactive. You can back annotate a design, then view the effects of your changes.
- **The Waveform Viewer** — The Waveform Viewer works with the Hierarchy Navigator to display the internal operation of your design. Once you have simulated the design, you can display waveforms from any schematic, at any level in the hierarchy. "Crossprobing" lets you jump at any time between a signal line in the schematic (as displayed by the Navigator) and the signal itself (as displayed by the Viewer).
- **Waveform Editing Tool** — The Waveform Editing Tool (or WET) lets you create the stimulus waveforms for simulation graphically, by drawing them directly on the screen. The stimuli can be edited graphically, or by modifying values in dialog boxes. The WET then converts the waveforms into a stimulus file that your simulator recognizes. Waveform files are also useful as input to automatic test equipment, or as documentation of the circuit's expected behavior.

Using the SCS Executive

(SCS/ECS Only) The SCS Executive is the first thing you see when you run SCS. You can launch the Schematic Editor, Symbol Editor, or Hierarchy Navigator directly from the Executive. The Executive also gives you easy access to a number of utility programs, including schematic exchange/conversion utilities and the INI Editor.

Running the Editors or Hierarchy Navigator

The most common use of the SCS Executive is to launch one of its three principal applications.

1. Click the appropriate radio button: Navigate Hierarchy, Edit Schematic, or Edit Symbol. The label of the edit box changes accordingly, as does the default file extension (***.tre**, ***.sch**, ***.sym**). The list box shows the corresponding files in the current directory.
2. If you want a different file extension (or a specific file name), type it into the edit box and press ENTER. The contents of the list box change to reflect the new specification.
3. If you want a different directory, double-click on a directory or the double dot [..] in the list box to change the directory.
4. Double-click on the file name you want to work with. Or highlight the name and click on RUN. If you want to create a new schematic, symbol, or hierarchy file, click on NEW instead.

You don't have to use the SCS Executive; you can run these applications from the command line or File Manager. However, the Executive simplifies the process by automatically passing the selected file name to the program.

Editing Files

The SCS Executive can be used to search for a file you want to edit and load it into the Notepad editor.

1. Choose the View menu. A list of wildcards for various file types is displayed, along with the universal wild card (*.*).
2. Click on the file type you want to edit. The contents of the list box are immediately updated to show files of the type you selected.
3. If the file you want is not in the current directory, double-click on a directory or the double dot [..] in the list box to change the directory.

4. Double-click on the file name you want to edit. Or highlight the name and click on RUN. The Notepad is invoked and the selected file is loaded. If you want to create a new text file (of any type), click on NEW instead.

Note: *When you select one of the utility programs (such as the Notepad editor, or the utilities described below), the radio button at the bottom of the window changes from "Utilities" to the name of the selected utility. The name remains there until you select another utility. You can call the Editors or Hierarchy Navigator, then click the bottom button to return to the last-used utility without having to reselect it from the menus.*

Schematic Exchange/Conversion Utilities

The conversion utilities work in much the same way as loading or editing a file.

1. Choose the Utilities menu.
2. Select the type of conversion you want to perform. The contents of the list box change immediately to show only files of the appropriate source type.
3. If the file you want is not in the current directory, double-click on a directory name or the double dot [..] in the list box to change the directory.
4. Highlight the name of the file you want to convert, then click on ONE. To convert all files in the current directory, click on ALL. The converted files are placed in the same directory as the source files.

Setup Utilities

With the Windows version, the setup menu gives you the option of setting the current **.ini** file or of editing the current or master **.ini** file. (The default initialization file is **ecs.ini** file from the **config** subdirectory. The INI Editor and the **ecs.ini** file are described in Chapter 9, *The SCS INI Editor*.)

SCS uses a number of other initialization files. These are normally modified in an ASCII text editor (such as Notepad). If an application uses an initialization file, the file is described in the same chapter of this manual where the application is described.

Customizing the Executive with the pcshell.ini File

The **pcshell.ini** file is used only with the Windows version of SCS. If you are using the UNIX / Motif version, you can skip this section.

The **pcshell.ini** file gives you complete control over the SCS Executive. You can select the labeling and function of the radio buttons. You can also add or remove any of the menus, and include whatever commands you want under those menus (including applications not supplied with SCS). A typical **pcshell.ini** file is shown below:

```
[Buttons]
Navigate Hierarchy=hiernav.exe,Design:,\*.TRE,NAV,&F
Edit Schematic=schem.exe,Schematic:,\*.SCH,NEW,&F
Edit Symbol=sym.exe,Symbol:,\*.SYM,NEW,&F

[&Utilities]
Schem to ASCII=ascout.exe,Schematic:,\*.SCH,ALL,&F
ASCII to Schem=ascin.exe,ASCII File:,\*.ASC,ALL,&F
Update Schem=updatesc.exe,Schematic:,\*.SCH,ALL,&F
Symbol to ASCII=asyout.exe,Symbol:,\*.SYM,ALL,&F
ASCII to Symbol=asyin.exe,ASCII File:,\*.SYM,ALL,&F
Write EDIF=edifout.exe,Symbol:,\*.SYM,ALL,&F
Read EDIF=edifin.exe,EDIF File:,\*.EDF,ALL,&F
Back-Annotate Schematic=scback.exe -pads, Schematic:,\*.eco,all,&F

[&View]
Edit *.*=notepad.exe,Text File:,\*.*,NEW,&F
Edit *.INI=notepad.exe,Text File:,\*.INI,NEW,&F
Edit *.V=notepad.exe,Text File:,\*.V,NEW,&F
Edit *.PC=notepad.exe,Text File:,\*.PC,NEW,&F
Edit *.PTN=notepad.exe,Text File:,\*.PTN,NEW,&F
Edit *.HDR=notepad.exe,Text File:,\*.HDR,NEW,&F
Edit *.SCR=notepad.exe,Text File:,\*.SCR,NEW,&F

[&Setup]
Set Current ini File=makeini.exe,Ini File:,\*.ini,No,&F
Edit Current ini File=pcini.exe,,,,-current
Edit Master ini File=pcini.exe,,,,-master
```

pcshell.ini Format

Buttons and Menu Entries

The [Buttons] section controls the labeling and function of the radio buttons. The other sections add menus to the menu bar. You can define up to five buttons and 20 menus. There can be no more than 200 items for all the buttons and all the menus combined.

The [Buttons] section must be labeled "Buttons," but the menu sections can have any name you want. Place an ampersand (**&**) in front of the letter in the section name that you want to be the accelerator key for that menu. For example, if you add a [Verify] section, and you want ALT+F to be the accelerator for the Verify menu, place an ampersand in front of the letter F: [Veri&fy].

Two menus can have the same accelerator key, but you have to press the key twice to access the second menu. If you do not specify an accelerator key, the menu is accessible only with the mouse, not from the keyboard.

The formats for the button items and the menu items are nearly identical. The text to the left of the equals sign (**=**) is the label for the radio button, or the command entry in a menu. The text to the right of the equals sign is the command itself, plus additional display information for the SCS Executive. Look at the second entry in the [&View] section:

```
Edit *.INI=notepad.exe,Text File:,\*.INI,NEW,&F
```

There are five items on the right side of the equals sign, separated with commas (**,**). If you omit an item, you must include the comma as a placeholder. The table below lists each item and its use.

Item	Use
`notepad.exe`	The name of the application to run. You can also run **.bat** or **.pif** files.
`Text File:`	In the [Buttons] section, the label for the radio button. In the menu sections, the label for the filename edit box. (This is usually the file extension or file type.) If this item is omitted (as in the [&Setup] section), the edit box is not relabeled.
`\*.INI`	The default directory and file extension. The entry must start with `\*.`, which represents the current directory (\ ) and the wildcard for the base name (***.**).
`NEW`	The label for the second pushbutton at the bottom.

"NEW" runs the program without passing a file name.
"ALL" runs the program and passes a file name of " * ".
"NAV" creates a Hierarchy Navigator **.tre** file for all schematics in the directory.
"NO" disables the button.
If no label is specified, the program is run and no file name is passed.

`&F` Any command line options (including none). If an ampersand (**&**) appears in the command line, it is replaced by the file name selected from the list box. A letter following the ampersand specifies the format for the file name:

- B Base name (no drive, path, or extension)
- F Base name plus extension (no drive or path)
- E Extension only (including the dot (**.**)
- D Drive plus path
- P Drive plus path plus trailing backslash
- R Drive plus path plus base name
- W Current working directory
- N Fully qualified pathname (drive, path, base name, extension)
- & If your command line requires a literal ampersand (*not* a filename token), enter two ampersands (**&&**).

The command line options are not limited to those listed above. You can add any options your application needs.

Path Specifiers

Your command line can include environment variables defined in the Windows Registration Editor (RegEdit that comes with Windows), or in PATH statements in your **autoexec.bat** file.

To use a variable specified in the Registration Editor, prefix the variable's name with a percent sign (**%**) and follow it with a backslash (\ ), for example,

```
%ROOT\
```

To use a path specified with the SET PATH command in **autoexec.bat**, prefix the path's name with a dollar sign (**$**) and follow it with a backslash (\ ), for example,

```
$my_directory\
```

In either case, the path identifier is interpreted as all the alphanumeric or underscore (**_**) characters following the percent or dollar sign. The path identifier terminates when the first non-alphanumeric character or non-underscore is reached.

If your command line requires a literal dollar sign, enter two dollar signs (**$$**). If your command line requires a literal percent sign, enter two percent signs (**%%**).

Note: *All* **pcshell.ini** *command lines and all command lines in the simulator initialization files (described in Appendix B, "Simulator Interfaces") use this same format for path specification.*

Additional pcshell.ini Features

If you want to add a separator line to a menu, add a line that starts with a hyphen (**–**). The rest of the line is ignored.

You can add, edit, or delete items in **pcshell.ini** as you see fit. Changes do not appear until the next time you run the SCS Executive.

The SCS Executive Command Line

When SCS is installed, the SCS Executive is added to the Synario program group, with this default command line:

```
scs.exe
```

You can modify the Executive's behavior with the following command line options:

-ini *config***.ini**	The specified *config***.ini** SCS initialization file is used instead of **ecs.ini** or the file specified in the INIFILE variable of the [SCS] section of **win.ini**.
-menu *shell***.ini**	The specified *shell***.ini** shell configuration file is used instead of **pcshell.ini**.
-path *path*	The specified directory *path* becomes the working directory, instead of the path specified in the SHELLPATH variable of the [SCS] section of **win.ini**.

SCS Command Structure

Like most Windows or Motif applications, SCS has a graphical interface, with drop-down menus, dialog boxes, and a mouse-driven pointer to organize and simplify the user interface.

This graphical interface means that all Windows or Motif applications have a similar "look and feel." So if you're familiar with one application, you have a basic understanding of how other applications work.

SCS applications use an *action–object* command structure. You select the *action* you want to perform (usually from a menu), then the *object* you want to act on.

For example, to remove a symbol from a schematic, you first select the Delete command from the Schematic Editor's Edit menu. Then you point the mouse cursor at the symbol and click the mouse button to delete the symbol.

Almost all commands remain in effect until you select a different command. For example, if you select the Schematic Editor's Add: Wire command, you can continue to draw wires until you select a different command.

Using the Mouse

The mouse works the same as it does in other Windows or Motif applications. The following is a brief summary of mouse terms and actions.

Point	Move the mouse so the cursor touches a menu command or the object you want to act on.
Click, **Click Left**, **Click Right**	Press the specified mouse button briefly and release it. If no button is specified, the left button is assumed (that is, "Click" is the same as "Click Left").
Click On	Point the mouse cursor at an item or object and click the specified mouse button. If no button is specified, the left button is assumed.
Double-click	Click the left mouse button twice, quickly.
Drag	*Dragging* is used to outline a selected area, or to draw an object (a wire, a circle, a rectangle). Point the mouse cursor at any corner of the area you want to select, or at the starting point of the object you're going to draw. Press *and hold* the left mouse button. Then drag the mouse to outline the area or draw the object. (The area or object is shown as a dotted line.) The drawing or selection operation is completed when you release the mouse button.

Right Mouse Button Functions

The right mouse button has its own set of functions. It:

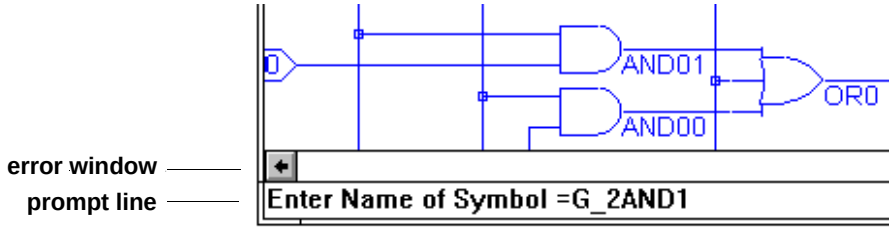
- Resets most commands. For example, if you start drawing a wire but change your mind about where it should go, clicking right deletes the proposed wire (shown as a dotted line) without canceling the Add: Wire command.
- Switches some commands to an alternate mode. For example, when dragging the mouse diagonally to add a wire, clicking right changes the wire's routing.
- Cancels some commands. For example, if you start to paste the contents of the Clipboard but change your mind, click right anywhere in the Editor's window to cancel the Paste command.

Some mice have a third, center button. SCS does not use this button.

Prompting and Error Messages

All commands that require additional information or another action will prompt you for it. The *prompt line* is at the lower-left corner of the window. Look at the prompt line whenever you're not sure what to do. Figure Figure 4-1 shows a prompt line in the Schematic Editor.

Figure 4-1
Schematic Editor Error Window and Prompt Line



The prompt line also displays what you type, such as the names of symbols and signals. You can edit this information as you enter it, using the ARROW, DEL, and BACKSPACE keys.

Immediately above the prompt line is the horizontal scroll bar. When minor errors occur (such as those that prevent a command from completing its action), the Editor replaces the scroll bar with an *error window* that describes the error. (The prompt usually explains how to fix the error.) Major errors are reported in pop-up message boxes.

Error Recovery

The Schematic and Symbol Editors have a feature that increases the chance of fully recovering from a hardware or software malfunction (a "crash" or "lockup"). The first time you save a new design, the Schematic Editor creates a log file named *design*.**_sc** (where *design* is the base name of the schematic file). The Symbol Editor log file is named *design*.**_sy**.

The Editors check for a log file before opening an existing schematic or symbol. If the file exists, it means:

- Someone else on the network has this particular file currently open. (See the following section on network operation.) *Or* …
- The last time the file was open, the user did not exit the Editor normally.

When either Editor finds a log file, the Editor displays a warning message that asks you to verify that no one else is editing the target file. If someone else on the network is editing the file, you should not edit it, as you would overwrite each other's changes. If you tell the Editor that someone else is editing the file, the Editor will not load it.

If you respond that no one else is using the file, the Editor assumes the log file is left from an interrupted editing session. The Editor then asks if you want to recover the file. If you respond Yes, the Editor uses the log file to recover any changes made to the schematic or symbol before the last editing session was interrupted.

The log file is not named until you first save a schematic. Be sure to save new schematics and symbols as soon as you start working on them, so an easily identifiable log file will be available. (Until you save the file and name it, the Editor gives the log file an arbitrary name.) Whenever you exit the Schematic or Symbol Editor normally, the Editor updates the schematic or symbol file (*design*.**sch** or *design*.**sym**) and erases the log file.

Network Operation

When schematics, symbol files, and other project components are shared on a network, some form of overwrite protection is required. The system cannot allow two people to work on the same file at the same time, continually overwriting each other's work.

The Schematic Editor uses the crash-recovery log file (described in the preceding section) to ensure single-user access. If you try to open a file and the Schematic Editor finds a log file for it, you're asked if someone else is using the file. If you answer Yes, you can't access the file.

If you answer No, you'll be allowed to open the file, *even if someone else is editing it*. Check with anyone who might be using the file before answering No.

Naming Design Files

When you name schematic or symbol files, observe the following rules:

- Observe DOS file naming conventions. The dollar sign (**$**) cannot be the first letter of a filename, however.
- Don't enter a filename extension; the Editor will add it automatically. If you enter a non-standard extension, the Editor will replace it with the correct extension (**.sch** for schematics, **.sym** for symbols).
- The system reserves several filename extensions for their own use. Avoid using the following extensions when naming your own files:

._ln	Hierarchy Navigator log file
._sc	Schematic Editor log file
._sy	Symbol Editor log file
._wt	Waveform Editing Viewer log file
._wv	Waveform Viewer log file
.asc	ASCII schematic file
.asy	ASCII symbol file
.bin	Binary waveform file
.edf	EDIF netlist
.err	Error OUTPUT file
.his	Waveform Viewer history file
.nam	Binary waveform name file
.net	Generic netlist file
.pin	Netlist file for generic netlist by pin
.sch	Schematic Editor files
.sym	Symbol Editor file
.tre	Hierarchy Navigator file
.vtr	Hierarchy Navigator temporary file
.wav	Waveform Viewer waveforms and trigger information
.wdl	Waveform Editing Tool database
.wet	Waveform Editing Tool database

Saving the Schematic or Symbol

The Save command from the File menu saves your most recent changes to the schematic or symbol. If the schematic or symbol is new, you're prompted for a name. If the schematic or symbol already exits, changes are saved to the existing file.

The extension **.sch** is automatically added to the schematic's name. If you add your own extension, the Schematic editor discards it and appends **.sch** instead. (The Symbol Editor similarly forces the **.sym** extension.)

Schematics (and symbols) are independent files. A schematic can be stored in a project library for use in many designs, or it can be kept in the design directory for use in a specific design.

Printing and Plotting

Windows

All SCS programs are Windows / Motif applications, so the operating system handles printing for whatever printer(s) you have installed. You print a symbol, schematic, or the screen currently displayed by selecting the Print command from the File menu.

Windows supports a wide range of printers. The drawing or screen display will be printed at the highest quality the printer is capable of.

The default page layout is *landscape* (long side of the paper horizontal). Since most symbol and schematic drawings are wider than they are tall, landscape orientation gives a larger image. In cases where the image to be printed is taller than it is wide, and vertical (*portrait*) orientation would make better use of the paper, use the Print Setup dialog box from the File menu to change the orientation to Portrait.

If you want to use a plotter to print your designs, attach it to the appropriate port (usually COM1: or COM2:), then install and configure the appropriate Windows driver using the Printers dialog box from the Windows Control Panel. Select the plotter driver in the Printer Setup dialog box from the File menu before you print.

UNIX/Motif

UNIX / Motif requires a PostScript® printer. The printer setup information must be supplied in the INI file. If it is not there, add it under the [PostscriptPrinter] header. The INI Editor will not remove or alter these entries.

```
[PostscriptPrinter]
Orientation=Landscape
TypeFace=Helvetica-Narrow
SetupLine=
PrinterName=lpr
PaperWidth=8.5
PaperHeight=11.00
SideMargin=0.25
TopMargin=0.25
```

PostScript Setting	Description
Orientation	*Landscape* (top and bottom edges of schematic printed along the length of the paper) or *Portrait* (top and bottom edges printed along the height)
TypeFace	The typeface to be used when printing. The specified font must be listed in the file $ROOT**/data/fontlist.ini**. If no font is listed, or the listed font is not in this file, the font defaults to Courier.
SetupLine	Special instructions to the printer. Anything on this line is inserted in the printer command line: statusdict begin *setup_line* end.
PrinterName	The name of the printer. Usually **lpr** on a UNIX system. If this field is blank or contains an asterisk (*****), an encapsulated PostScript (**.eps**) file is created on the disk, rather than being sent to the printer.
PaperWidth	The width of the paper in the printer.
PaperHeight	The length of the paper in the printer.
SideMargin	The side margin desired. Most laser printers cannot printer closer than 0.25" to the edge of the paper.
TopMargin	The top and bottom margins desired. Most laser printers cannot printer closer than 0.25" to the top or bottom of the paper.

The INI Editor

An initialization (“INI”) file maintains a wide range of configuration and display options for the Schematic Editor, Symbol Editor, Hierarchy Navigator, Waveform Viewer, and Waveform Editing Tool. This file is in ASCII format and is human-readable. You can modify it with any text editor, but the INI Editor is the easiest and safest way to make changes. Chapter 9, “The SCS INI Editor,” explains the options, and how to select or alter them.

The default initialization file is **ecs.ini** in the **config** subdirectory. You can, however, modify this file, then save it under a different name to create multiple, customized INI files.

Whenever you save a file (whether or not you modified it), the INI Editor asks if you want to make the saved file the “current project.” If you respond Yes, the new file becomes the default INI file. You can change the INI file in use at any time by loading a different INI file into the INI Editor, then using the Save command and responding Yes.

Chapter 5
Using the Schematic Editor

The Schematic Editor has the following significant features:

- Connectivity is created and maintained automatically *as you draw the schematic*. This allows the Editor to catch many common errors as they occur.
- Multi-sheet schematics are supported. Nets with the same name on different sheets are automatically connected.
- Full hierarchical design (top-down, bottom-up, or inside-out) is supported. Schematics can be represented as Block symbols for inclusion in other schematics.
- Symbols can be created without quitting the Schematic Editor. The symbols are immediately available for placement in the schematic.
- Iterated instances, in which one symbol represents any number of identical components, are supported.
- An unlimited number of “undos” and “redos” can be performed.
- The properties of components, pins, and nets can be queried at any time.
- Crash recovery is simple and transparent.

Chapter Contents

This chapter covers the following topics:

- Buses—Names, Types, and Creation
- Bus Taps
- Bus Pins
- Wiring Constraints
- Modifying the Schematic
- Debugging and Verifying a Schematic
- Schematic Editor Display Options

What Is a Schematic?

A schematic is a drawing that represents all or part of an electronic circuit. In terms of SCS, a schematic consists of the following three things:

- References to symbols from the SCS symbol library (or user-created symbols).
- The circuit's *connectivity:* the wiring that interconnects the symbols and links the circuit to "the outside world." SCS automatically creates and maintains the circuit's connectivity as you add or remove symbols and wiring. The schematic's database is always ready for use, so no post-processing is required.
- Attributes (values that define the electrical behavior (and other characteristics) of the schematic's components (both symbols and wiring)).

A schematic can also contain decorative graphics or text annotations.

For a simple circuit, a single schematic may be sufficient to show the entire circuit in terms of its primitive elements. A schematic created in the Schematic Editor can consist of more than one sheet, just as a hand-drawn schematic can extend over more than one piece of drafting paper. You can add as many sheets as you want to extend the original schematic, using the Sheet command from the File menu. The only practical limit is the amount of free RAM.

A schematic can also contain block elements that represent other schematics. A design using one or more block elements is said to be *hierarchical*, because it has more than one level. Additional hierarchical levels can be added as the design's complexity increases.

Schematic Filenames

A schematic's file name is the name of the schematic, plus the extension .**sch**. Since Windows 3.*x* is limited by DOS conventions, the base name of the file can have no more than eight characters. For file compatibility, the UNIX/Motif version uses the same name conventions as DOS.

Schematic Sheets Versus Hierarchical Levels

A complex design can be a single schematic consisting of multiple sheets, or a hierarchy of two or more schematics. Schematic sheets are a "horizontal" expansion of a design—all components are at the same level. Hierarchical levels represent a "vertical" expansion of a design, with some levels "superior" or "subordinate" to other levels.

Schematic sheets and hierarchical levels are two different ways of entering and organizing a design. The choice depends on the complexity of the design, and the relative ease of understanding the design's function and organization.

For example, a design that uses a number of standard modules over and over, or that combines simple modules into complex subsystems, might best be represented hierarchically. On the other hand, a design with little repetition or modularity would be easier to implement as a multi-sheet schematic. You can, of course, freely mix both approaches within a given design.

Schematic Components

A schematic consists of five components: symbols, wires, attributes, graphics, and text.

Symbols

Symbols represent most of the electrical components in a circuit—gates, flip-flops, inverters, multiplexers, and so on.

The electrical connections to symbols are made through pins attached to the symbols. Other than that, symbols have no electrical meaning. Their behavior and characteristics are defined by *attributes* (described below) attached to each symbol.

Wires

The electrical connections in a schematic are represented by lines called *wires*. All pins touched by a wire are electrically connected. A set of interconnected pins and wires is a *net* ("network").

Every net has a name. If you don't assign a name, the Editor assigns one when you save the schematic. The Editor's name consists of the letter N, followed by an underscore (_) and a number from 1 to $2^{32} - 1$ (4,294,967,295).

A wire representing a single conductor is called a *signal*. A wire that represents more than one signal is called a *bus*. (Buses are explained in more detail in a later section of this chapter, "Nets and Buses.") Wires can also be marked as external connections to a schematic with I/O markers (see **Add: I/O Markers** in the *SCS Command Reference*).

Attributes

Each symbol, pin, and net in a schematic has attributes. Attributes can describe any characteristic of a symbol, pin, or net, but are primarily used to define its electrical behavior. Some attributes, like those for device type and instance name, are automatically assigned when the symbol is created or placed in a schematic. Other attributes (usually instance-specific data) are assigned by the user.

Attributes are described in detail in Chapter 8, “Attributes.”

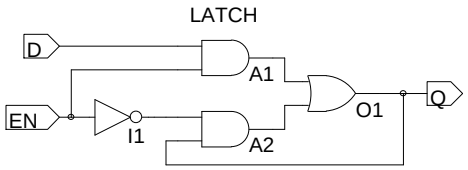
Graphics and Text

Graphics and text can add extra information to a schematic, such as clarifying circuit operation, or the circuit’s relationship to other circuits in the design. They can provide useful information for the reader, but they have no electrical significance (or other meaning) for SCS.

Notes, labels, title boxes, and other symbolic information can be added with the Add menu’s Rectangle, Line, Circle, Arc, and Text commands. These commands are described in detail in the *SCS Command Reference.*

Figure 5-1 shows a simple schematic.

Figure 5-1
A Simple Schematic



Adding Schematic Elements

To begin a new design, select New from the File menu. You can then start placing symbols. (If you want to work on an existing schematic, use the Open command to load it.)

Selecting a Symbol

To place a symbol:

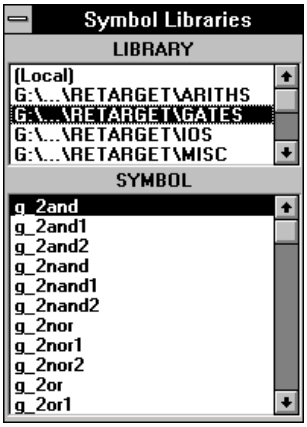
1. Select the Symbol command from the Add menu. You are prompted for the symbol's name.
2. **If you know the symbol's name**, type it on the prompt line, then press ENTER. (Type only the name of the symbol. Do not include the .**sym** file extension.)

   If you don't know the symbol's name, you can select it from the dialog box displayed when the Add Symbol command is selected:

   a. Click on a directory in the top list box. The bottom list box shows the symbols in that directory. (The directories in the top list box are specified in the INI Editor's Symbol Libraries dialog box. The INI Editor is described in detail in Chapter 8, "The SCS INI Editor.")

   b. Click on the desired symbol in the bottom list box.

Figure 5-2
Symbol Library List Box



The Schematic Editor searches the following places in the order listed to find the specified symbol:

- the project directory
- the symbol library search path

The search stops at the first symbol with the specified name. The symbol is then attached to the cursor.

To copy a symbol from the schematic:

If an instance of the symbol you want already appears in the schematic, select the Add Symbol command, then click on the instance. A copy of that symbol is attached to the cursor.

To select a symbol or symbols in the schematic:

- Click on the symbol.
- Shift-click to select additional symbols.
- Draf a box around the desired symbols.
- Use the Find button to select all symbols that match a specified criteria.

Placing the Symbol

Once the symbol is attached to the cursor, click at the desired location to place it. The symbol remains attached to the cursor, so you can place multiple copies without having to reselect the symbol.

The symbol can be mirrored or rotated before it is placed. Refer to the *SCS Command Reference* for a description of the Mirror and Rotate commands.

If you pick the wrong symbol, or change your mind, click right anywhere in the window to remove the symbol from the cursor. You can then select a different symbol. (You do not have to reselect the Add: Symbol command.)

To replace an existing symbol, move the cursor so that at least 50% of the new symbol covers the existing symbol. Click to delete the old symbol and place the new one. (If the overlap is less than 50%, the old symbol is not deleted, and the new symbol overlaps it.) None of the override attributes associated with the previous symbol are transmitted to the new symbol.

Adding Instance Names

Schematic symbols are usually given identifiers to distinguish them. SCS supports two types of identifiers: reference designators and instance names.

Reference designators are used most often to identify physical components. For example, a 7404 IC with six inverters might be given a reference designator of U42. The inverters within this IC are uniquely identified by their pin numbers.

Each symbol instance also has a unique instance name. Instances names are fully hierarchical. Chapter 3, "Introduction to Hierarchical Design," shows how instance names from multiple levels are combined to specify a single gate.

By default, the Schematic Editor automatically adds instance names of the form I_*nn*, (where *nn* is the next unused integer) each time you save the schematic file. You can assign names of your own

To override the Editor-assigned names:

1. Select the Instance Name command from the Add menu.
2. Type the desired name and press ENTER. The name is attached to the cursor.
3. Click on the symbol instance you want to name.

You do not need to assign your own instance names. However, carefully chosen names make it easier to locate specific symbols in a hierarchical design.

Sequential Instances

If you want to add sequential instance names (AND1, AND2, … AND*n*), start by selecting the Instance Name command (or click right to reset it if the command is already selected). Then

- Type the instance name. Add the first number of the sequence at the end of the name, followed by a plus sign (**+**). Or, if you want the sequence to start with 1, you can add just the plus sign, with no number.

  ```
  INV3+      [starts with INV3]

  INV+       [starts with INV1]
  ```

  Press ENTER. The instance name with the number you entered (or the number 1 if you used the plus sign) is attached to the cursor.

Or …

- Click on an existing instance. If the instance ends with a number, the same name, with the number incremented, is attached to the cursor. If the existing name does not end with a number, a numeric suffix of 1 is added to the root name and both the name and suffix attached to the cursor.

You can now click on the instance(s) to be named. The number is automatically incremented for each new instance. Any numbers already assigned to instances with the same base name are automatically skipped.

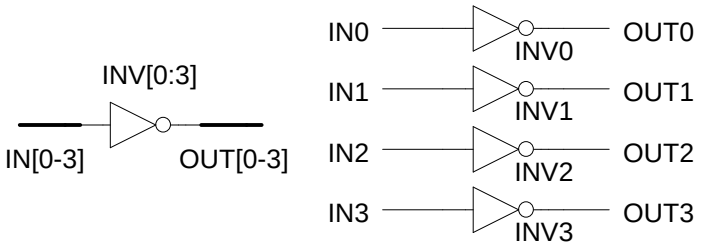
For example, if you are numbering eight inverters starting at INV3, and INV7 and INV8 already exist, the inverters are given labels of INV3 to INV6 and INV9 to INV12.

Please refer to the Instance Name command in the *SCS Command Reference* for a more detailed description of adding instance names.

Iterated Instances (Arrays)

Iterated instances allow a single symbol to represent multiple instances connected in parallel. Figure 5-3 shows two ways of representing four parallel buffers. On the right, four separate inverters are added to the schematic. On the left, one symbol with the instance name of INV[0:3] represents the bank of four inverters.

Figure 5-3
Iterated and Non-Iterated Inverter Arrays



Both examples represent exactly the same circuit connectivity. The iterated instance is a more compact way to represent repeating structures. Iteration is useful in bit-slice designs, for example, where complex structures are repeated many times. It is also convenient when connecting to a multi-bit interface.

A single instance is converted into an iterated instance by giving it a compound instance name, of the form

```
INV[3-10]
```

In this case, eight instances of the symbol you've named INV are created—but the symbol appears only once in the schematic.

The entry format for an iterated instance name is flexible. You can use brackets **[ ]**, parentheses **( )**, or curly braces **{ }** to delimit the numbers. And the numbers can be separated by a hyphen (**–**) or colon (**:**). Any of the following six forms can be used for entry:

```
INV[3-10]   INV[3:10]

INV(3-10)   INV(3:10)

INV{3-10}   INV{3:10}
```

Regardless of which form you use for entry, the Editor converts it to brackets and hyphens for display on the schematic.

You can also enter a list of instance names. A list is a mixture of single names and ranges of names. For example, C1,C3,C(5:20),C22,C24 assigns the instance names C1, C3, C5–C20, C22, C24 to twenty iterations.

Only Block, Cell, and Component symbol instances can be iteratively labeled.

Note: *Although an iterated instance can be "pushed into" in the Hierarchy Navigator, the "individual" symbols cannot be viewed separately. If you need to view them separately, you must create a Block symbol containing individual symbols.*

Supported Characters in Instance Names

The following characters are supported in instance names:

A–**Z**, **a**–**z**, **0**–**9**	All alphanumeric characters. Names are not case-sensitive. INV, Inv, and inv are considered the same instance name.
'	Apostrophe (single quote)
_	Underscore (an underscore cannot be the first or the last character of an instance name)

Wiring the Schematic

Wires electrically connect the symbols. The symbol pins are the connection points for the wires.

Drawing Wires

Wires are added with the Wire command from the Add menu. The simplest wiring is Point-to-Point:

1. Select the Wire command from the Add menu.
2. Click on the first point of the wire. Two dotted lines, one horizontal and one vertical, stretch from the point selected to the cursor. As you move the cursor, the lines change from horizontal to vertical (or vertical to horizontal) depending on the position of the cursor relative to the first point.
3. When the first dotted section has the length and direction you want, click a second time. The first dotted section changes to the wire color and its position is fixed. A new dotted section is now added to the "rubber band" line. You can add additional wire segments by repeating steps 2 and 3.
4. To end the wire, click twice at the same point (or click right). The wire terminates automatically if the point you click on falls on a pin terminal.

Diagonal Wires

You can draw 45° wire segments by pressing SHIFT while clicking on the first point. Hold down SHIFT to add additional segments at 45°.

If you drag diagonally from the first point to the last, you enable a special wiring mode (called "Z and C connection") that simplifies routing complex wire paths. This mode (and other wiring techniques) are described in the *SCS Command Reference* under Add: Wire.

Nets and Buses

Any single- or multi-wire connection between pins is called a *net* ("network"). This section explains how nets are named, and how multi-wire nets (called *buses*) are created and named.

Net Names

The nets (networks) form the electrical connections among the components. Every net has a name, either assigned by you or by the Schematic Editor. Net names have two principal functions, *identification* and *interconnection*.

Identification

Meaningful net identifiers make a design easier to understand. Nets are usually given the names of the signals they carry.

If you don't assign a name, the Schematic Editor automatically assigns a unique name of the form N_*nn*, where *nn* is an integer between 1 and $2^{32} - 1$ (4,294,967,295) when you save the file.

You can override any Editor-assigned name by assigning one of your own. Use the Net Name command from the Add menu.

Interconnection

If a wire segment attached to a symbol pin is given the name of a net or bus, the pin is attached to that net or bus, even if you haven't drawn the connection on the schematic.

Two or more wires with no visible connection on the schematic are automatically connected if they have the same net name. Each wire is called a *branch* of that net. Inter-sheet connections are created in this way.

You can easily find implicit net connections with the Query command from the Object menu. Click on any net or bus. All wires with the same name are highlighted, on all sheets of the current schematic.

Nets with different names cannot be connected; the Editor will warn you if you try to "short" them.

Entering Net Names

Use the Net Name command from the Add menu to assign a name of your own choosing to a net. Your name replaces any name the Schematic Editor may already have assigned. If you assign the same name to two separate nets ("branches"), *they are connected*, even though no connection appears on the schematic. This feature makes it easy to connect widely spaced components without having to draw long wires across the schematic.

Net names you assign are always displayed; Editor-assigned names are not displayed. To avoid cluttering the schematic, you should name only those nets

- That connect to other schematics.
- Whose functions need documentation or clarification.
- Whose signals you want to view in the Hierarchy Navigator.
- Whose signals you want to reference in simulation or timing analysis.

There are many ways to enter net names. After selecting the Net Name command:

- Type a single name and press ENTER. This name is attached to the cursor.
- Type a compound name and press ENTER. (Compound names are explained in the section on buses later this chapter.) The full compound name is attached to the cursor.
- Copy the name of an existing net or bus to the cursor by clicking on the wire.
- Enter a name prefix with a number and a plus sign (+) to define an auto-increment sequence for net names. The first name is attached to the cursor and subsequent clicks attach sequential net names. See also Add: Net Name in the command reference.

Placing the Net Name

Once a net name (or group of names) is attached to the cursor, there are three ways to place it:

- Click on an empty point. (You can place all the net names first, then add the wires later.) You will receive a warning message if a wire is not eventually connected to the name.
- Click on a wire. A name placed at the end of a wire is left- or right-justified. A name in the middle of a wire is centered.

  The position of the name is determined by the segment ends at the time of placement. If both ends are connected, the name is placed in the middle. If neither end of the segment is connected, the name is placed at the starting point. If only one end of the segment is connected, the name is placed at the unconnected end.

- Drag the mouse to simultaneously add a wire segment and its name to a pin, wire, or bus. If either end of the segment connects at right angles to a wire or bus, a bus tap is created at that end. If the wire is not already a bus, it is promoted to a bus.

  Once you drag the mouse to or from a pin, you can place subsequent net names *simply by clicking on a pin***.** (You don't need to drag.) A wire segment is automatically added, of the same length as previously dragged. The name is attached as described above.

Individual elements of a compound name can be sequentially attached to different nets:

1. Enter the compound name.
2. Select the Expanded Bus Name command from the Add menu (or click right). The first name of the sequence is attached to the cursor.
3. Click on a net to place the name. The next name in the sequence then appears on the cursor.

Repeat the process until all the names are assigned. Click right at any time to stop adding names.

Regardless of how you attach the name, the Schematic Editor highlights the net you're attaching the name to as you click.

Legal Characters in Net Names

The following characters can be used in net names:

A–**Z**, **a**–**z**, **0–9**	All alphanumeric characters. Case is not significant.
'	Apostrophe (single quote)
_	Underscore

Reserved Names

If B is the first character of a net name, the underscore cannot follow it as the second character (as in B_). The underscore cannot be used in a net name of the form N_*nn* (where *nn* is any integer). These names are reserved by the Schematic Editor for nets that have not been named by the user.

Logical Inversion

When either the apostrophe or underscore is the first character of a net name, the Schematic Editor draws the name with an overbar. (Overbars are often used to suggest logical inversion.) The apostrophe or underscore is kept in the name and appears in the netlist, but it is not displayed.

Renaming a Net

To rename a net:

1. Select the Net Name command, or click right to reset the command (if it's already selected).
2. Type the new net name and press ENTER. The new name is attached to the mouse cursor.
3. To rename the net (that is, *all branches* of the net, across *all* sheets), hold down SHIFT as you click. You can click anywhere on the net.

   If the renamed net has an I/O marker:

- The I/O marker is removed if you click to rename a specific branch.
- The I/O marker is kept if you hold down SHIFT to rename all branches.

Important! *Remember that renaming a single branch disconnects that branch from the rest of the net and connects it to the net (if any) with the new name.*

Note: *If the name of a branch is displayed two or more times on a single branch, you cannot rename the branch. You must first use the Delete command to remove the extra name(s).*

Specifying Signal Direction

I/O Markers

An I/O marker is a special indicator that identifies a net name as an input, output, or bidirectional signal. This establishes *net polarity* (direction of signal flow) and indicates that the net is externally accessible.

The Schematic Editor's Consistency Check command uses I/O markers to flag any discrepancies in the polarity of marked signals and the symbol pins. Discrepancies in polarity are also flagged each time you run the Hierarchy Navigator.

Adding a Marker to a Net

To label a net as input, output, or bidirectional (or to change its polarity):

1. Select the I/O Marker command from the Add menu.
2. Select the desired polarity from the dialog box. (Or select None to remove an existing I/O marker.)

3. Click at the point where the I/O marker touches the end of a horizontal or vertical wire segment or bus.

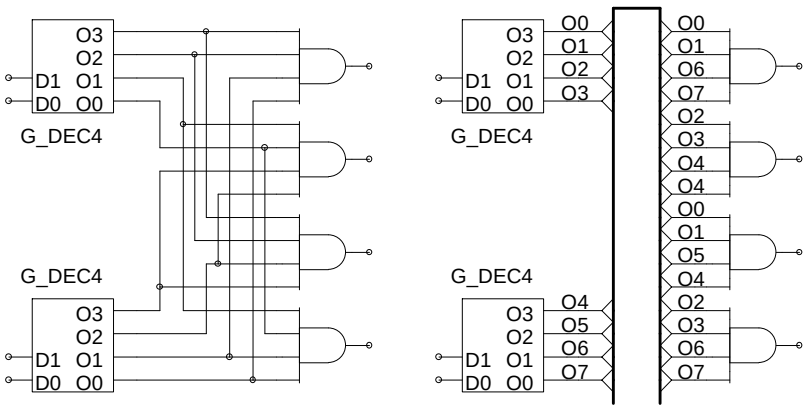
You can place, remove, or change several markers at one time by dragging a box around the wire ends.

An I/O marker can only be added to net names at the end of a horizontal or vertical segment of wire. A net should only have one I/O marker per sheet.

Buses

A *bus* combines two or more signals into a single wire. Buses are a convenient way to group related signals. This grouping can produce a less cluttered, functionally clearer drawing and clarify the connection between the main circuit and a Block symbol. Figure 5-4 shows how a circuit appears before and after a bus has replaced individual wires. The two schematics are electrically equivalent.

Figure 5-4
Same Circuit Without and With an Unordered Bus



The following sections explain the creation and use of buses.

Bus Types

There are two types of buses: *ordered* and *unordered.*

Ordered Buses

An ordered bus has a compound name consisting of the names of the signals that comprise the bus. Any signals can be combined into an ordered bus, whether or not they are related.

A net becomes an ordered bus when it is given a compound name. The net is *promoted* to an ordered bus containing the nets listed in the compound name. (The net is redrawn at twice its regular thickness to indicate that it's now a bus.)

A compound name is a list of two or more net names, separated by commas. For example:

```
READ,WRITE,MYNAME
```

represents the three signals READ, WRITE, and MYNAME. Spaces in compound names are ignored.

A compound name can also be formed by adding a sequence of numbers to a name. The sequence is specified as a starting number, an ending number, and an optional increment (default = 1). The numbers are positive integers, and are delimited by commas (**,**), dashes (**–**), or colons (**:**). The sequence is enclosed in brackets **[ ]**, parentheses **( )**, or curly braces **{ }**.

The following are examples of sequential compound names:

Sequential Name	**Signals**
DATA[0-7]	DATA[0] DATA[1] … DATA[7]
ADDR(0,14,2)	ADDR(0) ADDR(2) ADDR(4) … ADDR(14)
IO{4:23:3}	IO{4} IO{7} IO{10} … IO{22}

If the increment is greater than one, the ending number will not appear in the sequence if it does not equal the starting number plus an integral multiple of the increment (as in the third example above).

A compound name can also combine individual names and compound names in any order:

Sequential Name	**Signals**
CS,DATA{0:7},WR	CS DATA{0} DATA{1} … DATA{7} WR

The order of the signals in the bus is the same as the order in which they are specified. The order is significant only when the bus is connected to a bus pin. (Bus pins are described in a later section, "Bus Pins.")

Unordered Buses

An unordered bus is nothing more than an unnamed wire with *bus taps.* A net with a single name (or any unnamed wire) is promoted to an unordered bus by attaching one or more bus taps to it. The order of the signals within an unordered bus is not defined and has no significance.

Although the order of the signals in an unordered bus has no significance, *you must name the wires connecting to the bus taps,* because the Schematic Editor would otherwise have no way of determining which symbol pin at one end connects to which symbol pin at the other end.

The bus shown in Figure 5-4 is an unordered bus. Unordered buses provide a convenient way to route signals through the schematic with a minimum of visual clutter. They have no other function.

Unordered buses cannot connect to bus pins, because bus pins represent an ordered sequence of signals.

Bus Taps

Signals enter (or exit) a bus at points called *bus taps.* A bus tap can be added to any existing bus, net, or wire. If a net or wire is not already a bus, adding the tap automatically promotes it to a bus. To add a tap:

1. Select the Bus Tap command from the Add menu.
2. Position the cursor on the bus or wire where the tap is required.
3. Drag the mouse to draw a wire perpendicular to the bus.
4. Release the mouse button when the wire is the desired length.

Bus taps can be made only on vertical or horizontal sections of a bus. Tap connections are shown as two diagonal lines.

More than one tap can be taken from the same signal, to simplify routing or permit a cleaner layout. But you cannot add a bus tap to an existing bus tap. You'll get the error message "Forming Multilevel Bus."

Naming the Tap

Once the tap has been added, use the Net Name command from the Add menu to name it. If the tap is from an ordered bus, the tap's name must match the name of a signal in the bus. If it does not, the Schematic Editor or Hierarchy Navigator will flag it as an error.

Note: *Wires entering and leaving any bus (ordered or unordered) must be tagged with a net name to indicate which signal is being tapped. Unnamed taps will eventually be flagged as errors.*

Connecting to Pins

A tapped signal connects to an ordinary symbol pin in the usual way. An ordered bus connects to a *bus pin* (a pin with multiple connections) directly. No taps are needed; the connections are made automatically. The first signal in the bus connects to the first signal in the bus pin, the second to the second, and so forth. Both the bus and the bus pin must contain the same number of signals.

Creating Taps with the Net Name Command

The Net Name command creates the tap, a wire, and the net name.

1. Select the Net Name command from the Add menu.
2. Type a net name contained in the bus to be tapped, then press ENTER.
3. Point to the place on the schematic where you want the free end of the bus tap.
4. Drag the mouse to the bus and release the button. The tap is made, and the free end is labeled with the signal name.

Bus Pins

In the Schematic and Symbol Editors, a *pin* represents either a physical pin on a real component, or a signal from a lower-level schematic.

A *bus pin* represents a group of pins or signals. You create a bus pin by giving a pin a compound name (that is, a list of signals). If the pin connects to a Block symbol, each of the signals listed in the bus pin's name must also appear in the schematic. This defines the connection between the Block symbol and its underlying schematic.

Ordered buses can connect directly to bus pins. The number of bits or signals attached to the bus must match the number of bits or signals attached to the bus pin.

The first signal in the bus (by definition, the first signal in the bus's name) is connected to the first signal represented by the pin. The remaining signals in the bus are connected to the remaining pins in the same order you assigned the signal names to the pins.

Nets on Iterated Instances

Iterated instances allow a single symbol instance to represent multiple instances in a parallel connection. Figure 5-3 shows two ways of representing four parallel buffers. On the right, four separate inverters are added to the schematic. On the left, one symbol with the instance name of INV[0:3] represents the bank of four inverters.

Compound Names

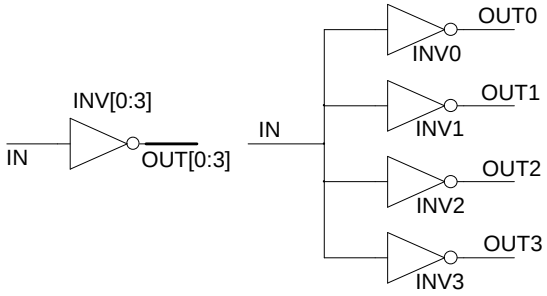
The input and output nets of an iterated instance can be given either single names or compound names. If (as in the example shown in Figure 5-3), the inputs or outputs are given a compound name, their nets are promoted to buses in which each instance's input or output is a separate signal.

Iterated buses work like any other bus. You can attach a bus (with the same number of signals) directly to them, as you would any other bus.

Single Names

If an iterated input or output is given a single net name, there is only one input or output net, and all the inputs or outputs connect in parallel to that single net. In Figure 5-5, the input net is given a single name and the inputs of all four gates are connected in parallel to the net.

Figure 5-5
Iterated Inverters with the Inputs in Parallel



This feature is used most often on inputs, not outputs. Paralleled outputs represent a "wired-OR" configuration, which is usually drawn as separate gates, rather than as an iterated instance.

Bus and Net Connections to Iterated Instances

You can make an iterated instance of any symbol. A simple (non-bus) pin on an iterated instance remains a simple pin. The iteration does not convert the pin to a bus pin. The same rules of connection to a simple pin still apply.

Connections to the nets of iterated instances are made according to the following rules:

Connection	Rule
Simple net to simple pin	The net connects to corresponding pin on each instance.

Bus to bus pin	Successive bus signals connect to successive bus pins on successive instances. The bus and bus pin must have the same number of nets.
Simple net to bus pin	Not permitted. Only a bus can connect to a bus pin.
Bus to simple pin	The *N*th signal of the bus connects to the scalar pin of the *N*th instance. The width of the bus and the number of instances must match.

Wiring Constraints

The Schematic Editor enforces a number of wiring constraints. Most are intended to encourage clean layout and prevent ambiguous wiring patterns.

- All wire segments must end on Primary grid points.
- Wire segments must be oriented on the 45° and 90° axes.
- Wire segments cannot form acute angles. This applies to both crossing and connected wire segments.
- A maximum of two diagonal wire segments can connect at a point. A third vertical or horizontal segment can connect if it does not form an acute angle with either of the other segments.
- A maximum of three wires can connect at a pin or I/O marker.
- A net can have only one name (simple or compound).
- Two I/O markers with different net names cannot be connected by a wire.
- An I/O marker can connect only to a wire segment, never a pin. (Add a wire segment if you want a marker to be near a pin.)
- An I/O marker cannot be placed in the middle of a diagonal line, only at the end.
- An I/O marker cannot be placed at the crossing point of two wires, even if the wires are connected.
- A tap can be only be placed on a vertical or horizontal section of a wire.
- Only one bus tap can be made at any point on a bus.
- A bus can contain only individual signals, not other buses. Attempting to give a net in a bus either a compound name or the name of another bus is flagged as an error.
- The relationship between an ordered bus (that is, a bus with a compound name) and the signals in that bus is strictly enforced. Naming the bus or one of its signals in a way that breaks this relationship is not permitted. For example, you cannot assign a bus tap a name that is not in the bus.

Modifying the Schematic

The Schematic and Symbol Editors provide many commands to edit and modify your work. The section presents a brief introduction to these commands. For a full description, please refer to the *SCS Command Reference.*

Clipboard Commands

The Cut, Copy, and Paste commands use the Clipboard. Executing Cut or Copy replaces the current contents of the Clipboard with whatever you have just cut or copied.

Cut and **Copy** — The Cut and Copy commands place a copy of the selected object on the Clipboard. Either click on the object, or drag a box around the exact section you want to cut/copy. Cut removes the selected object; Copy leaves it in place.

Paste — The Paste command pastes objects on the Clipboard into the current drawing, or a drawing in another Schematic Editor session. Pasting does not remove the object from the Clipboard. It can be pasted as often as you want.

Note: *Schematic objects and symbol objects are not compatible. You cannot paste a schematic object into a symbol drawing, or vice versa.*

Non-Clipboard Commands

The following edit commands *do not* use the Clipboard, and do not change the Clipboard's contents. All these commands can be "reset" (to clear the current selection or start over) by clicking right anywhere in the Editor's window.

Delete — Removes symbols, wires, text, and other components. You can click on individual objects to be deleted, or drag a box around the section to be removed.

Duplicate — Copies objects to another place on the same drawing. Either click on an object, or drag a box around the section to be copied. (Press SHIFT as you click or drag to select multiple objects or sections.) The object or section is attached to the mouse cursor; you can then click where you want a copy. You can continue to place copies until you select another command.

Move	Moves objects directly. Either click on the object, or drag a box around the section to be moved. (Press SHIFT as you click or drag to select multiple objects or sections.) The object or section is attached to the mouse cursor; you can then click where you want the object to go. The Move command is faster than cutting and pasting.
Drag	Repositions objects without breaking their electrical connections. The wires slide, stretch, and form right angle corners (if possible). Click on the object, or drag a box around the section to be moved.
Rotate and **Mirror**	The object (or objects) currently attached to the cursor can be rotated and mirrored. The Mirror command reflects the object around its vertical axis. The Rotate command rotates the object 90°. When the symbol has the desired orientation, click to place it at the desired position.
Undo and **Redo**	Undo reverses your edits one command at a time. If you reverse too many times, use Redo to reverse the Undo. Only editing changes to the schematic or symbol can be undone. View and Sheet commands do not alter the drawing and cannot be undone. There is no limit to the number of Undos or Redos you can perform. However, each time a schematic (or symbol) file is saved, the file is fully updated and all information about the last sequence of edits is discarded. Don't save a file if there are any editing changes you want to Undo or Redo.

Debugging and Verifying a Schematic

The Schematic Editor has two levels of checking that attempt to report or prevent errors early in the design process.

First level errors are detected as you enter your schematic. For example, the Editor will not let you draw an isolated wire that forms a closed loop without connecting to anything else. It won't let you short together nets with different names.

Second level errors are recognized in the context of a complete design. An unconnected wire or pin, or an unnamed signal tapped from a bus, are normal during the first stages of a design. Some potential errors are always indicated, such as the dots on open pins and hanging line ends. Otherwise, errors of this type are reported only when schematics and symbols are combined in the design hierarchy.

You can check for errors and potential errors at any time. Select the Consistency Check command from the File menu. Any errors are written to a file and displayed in a list box. Clicking on an error in the list box displays the section of the drawing with the error and highlights the error with a small "plus" cursor. You can fix the error, then call Consistency Check repeatedly, until you've found and fixed all the errors.

The Consistency Check command warns about the following potential schematic errors:

- Bus taps should be named.
- Isolated I/O markers are not permitted.
- If the Mark Open Ends option is enabled, there should not be any unconnected wire ends.
- Unordered buses should not be connected to a bus pin on a symbol.
- Unordered buses should not be marked with I/O markers.
- Nets should not be marked with more than one I/O marker.
- A bus tap and its bus should not both be marked with an I/O marker.

If there is a symbol for this schematic, the Consistency Check command marks the following as errors:

- Each pin must have a corresponding net with an I/O marker whose direction matches the Polarity of the pin.
- Each net marked with an I/O marker must correspond to a pin.

"Unconnected Pin" Message

You can tell the Schematic Editor's Consistency Check command to ignore intentionally unconnected pins by appropriately setting one of the pin's attributes. This is attribute is #9, named OpenOK. Use the INI Editor to add this attribute and set its Modify Option to "+", Assign in Schematic.

The attribute can be set in either the Schematic or the Symbol Editor. If set in the Symbol editor, the pin is never flagged as unconnected. The attribute can also be selectively set in the Schematic editor to disable the message on specific pins, but not on all instances of the symbol.

Any value entered will inhibit the "Unconnected" message for that pin. A value of Yes, OK, or True is suggested.

Schematic Editor Display Options

The following three sections explain a number of display options available in the Schematic and Symbol Editors.

Schematic Sheets

The Sheet Setup command from the File menu selects which sheet (of a multi-sheet schematic) to view. Each sheet is displayed in a separate window, which can be closed, resized, or repositioned as any other window. When only one sheet window is open, it is enlarged to fill the main window.

Several sheets can be displayed at the same time. Each sheet's window is offset slightly from the previous sheet's window, so that some of the previous sheet can be seen. Click on any sheet's window to bring it to the front.

Multiple Views

You can open up to three views of a single sheet. Each view can have a different magnification. A total of eight sheet windows can be open at the same time. Objects cut or copied from one sheet can be pasted to another.

Resizing and Renumbering Sheets

The Modify option of the Sheet Setup command resizes and renumbers sheets. The sizes are set using the Sheet Sizes dialog box in the INI Editor. The Replace option of the Sheet Setup command replaces whatever sheets are currently displayed with a full-size window showing the selected sheet.

The maximum sheet size is 4095 x 4095 Primary grid units.

You can automatically resequence sheets by selecting File: Resequence Sheets.

Adding Blank Sheets

The Sheet Setup command can also add blank sheets to an existing schematic. To add a fifth sheet to a schematic that already has sheets 1–4, type 5 in the Select Sheet edit box. Then click on Open.

Note: *You can choose any number for a new sheet, even if it isn't the next number in the sequence. For example, if the schematic has three sheets numbered 1 through 3, you can add a fourth sheet numbered 7. (The largest sheet number allowed is 99.)*

To remove unused sheet numbers from a sequence of sheets, use the File: Resequence Sheets command.

Grids

Any elements added to a schematic (including symbols, wires, and buses) are automatically positioned on a grid. The default spacing of this *Primary grid* is one-tenth of an inch (or 2.5 mm). You can change the default with the Sheet Layout dialog box in the INI Editor.

Graphics are also aligned with this Primary grid, but you can align them with a Secondary grid that has two or four times the resolution. This finer grid gives better control over the position of names and graphic embellishments.

The Graphic Options command (described below) controls display of the Primary grid. (The Secondary grid is never shown.) The appropriate Secondary grid must be selected before graphic items can be positioned at the higher resolution.

Controlling Display and Graphics Options

The Display Options and Graphic Options commands from the Options menu sets a number of display options in the Schematic Editor and the Hierarchy Navigator. Changes last only for the current editing session. Use the INI Editor to permanently change the default values.

Display Options

The Display Options command displays a dialog box with the following check box options:

Connect Dots Three wires connected to a symbol pin, and four-wire junctions, are always drawn with a connect dot. When the Connect Dots option box is checked, a connect dot is also displayed when two wires are connected to a symbol pin, and at three-wire junctions.

Border The schematic's border is divided into lettered and numbered zones, to reference the positions of schematic items. When the Border check box is checked, the zone display is turned on.

Symbol Pins When the Symbol Pins check box is checked, unconnected pins are highlighted with a dot.

Pin Attributes When the Pin Attributes check box is checked, pin numbers and names are displayed.

Symbol Text When the Symbol Text check box is checked, fixed text within a symbol is displayed.

Symbol Attributes When the Symbol Attributes check box is checked, text for symbol attribute values is displayed.

Open Ends Wires not terminating on an I/O marker, symbol pin, or another wire are considered errors. When the Open Ends check box is checked, an error dot is displayed at the end of such wires, and on any I/O marker not connected to a wire.

Off Page Connects When this check box is checked, nets that appear on more than one sheet will display a cross-reference to the other sheets, *if you placed the name at the off-page end of the wire.*

Net Attributes When this check box is not marked, text for net attribute values is not displayed. This decreases redrawing time and visual clutter. Note that only net attributes that have net attribute windows defined are displayed even when this check box is marked.

Graphic Options

The Graphic Options command displays a dialog box with the following radio button and check box options:

Text Font Selects the text size. In the Windows version, the choices are First, Second, Third, Fourth, Fifth, Sixth, Seventh, and Eighth. In the UNIX/Motif version, the choices are Small, Medium, and Large. Text Font affects only the text to be added, not existing text.

Justify Text can be left-justified, right-justified, or centered. This parameter applies to fixed graphic text and symbol attribute windows. Justify affects only the text to be added, not existing text.

Grid All electrical elements in schematics and symbols are aligned with Primary grid. Graphic elements and text can be positioned on a finer grid of one-half (Mid) or one-quarter (Sec) the Primary grid.

Display Controls whether the Primary grid is displayed. The grid appears as dots, with one dot at every grid intersection. Every tenth grid point is larger. As you zoom out and the grid points get closer, some grid dots might not be displayed.

Full Cursor You can choose between a small “plus” cursor (the default) and a full screen cursor. The full screen cursor makes it easier to align objects.

Wide Lines	There are two line weights for drawing lines and rectangles. The Wide Line check box selects the heavier weight. Wide lines have the same weight as buses on schematics. This setting affects only lines to be added, not existing lines.
Vertical Text	Fixed text and attribute window text can be horizontal or vertical. Horizontal is the default. Mark the Vertical Text check box for vertical. Vertical Text affects only the text to be added, not existing text.

Setting Attribute Values

You can use the Schematic Editor to override attribute values that were assigned in the Symbol Editor. (Attributes are described in Chapter 8, "Attributes.")

Suppose your schematic has two tristate buffer symbols. One buffer needs to be large, while the other can be small. The Schematic Editor lets you select the buffers individually and customize each instance by changing its transistor size attributes.

Pin Attributes

To modify pin attributes:

1. Select the Pin Attribute command from the Add menu. The Pin Attribute Editor dialog box appears.
2. Click on the pin to be modified, or click on the symbol containing the pin to select all pins in the symbol.
3. Highlight the required attribute from the list.
4. Type the new attribute value and press ENTER.

The new value is changed or added to all selected pins.

Symbol Attributes

Symbol attributes can be assigned in either the Symbol Editor or the Schematic Editor. The editing process is the same as described above for editing pin attributes. You can select multiple symbols and assign attributes for all of them simultaneously as described below

You can control whether an attribute's original value, as defined in the Symbol Editor, can be overridden on the schematic. This is described in Chapter 8, "Attributes."

To select a symbol or symbols in the schematic:

- Click on the symbol.
- Shift-click to select additional symbols.
- Draf a box around the desired symbols.
- Use the Find button to select all symbols that match a specified criteria.

Finding Symbols with Specific Attributes

The Find button in the Add: Symbol Attribute dialog box brings up an Instance Filter dialog box that you can use to select all instances of symbols that match specified attribute criteria.

To select symbols by attribute criteria:

1. Select Add: Symbol Attribute, then click the Find button.
2. Select the attribute to be compared from the left box.
3. Select the comparison function (such as ==, <, >) from the center box.
4. Enter a value for the attributes to be compared against.
5. To add the selected symbols to a list of selected symbols, clear the Replace Current Selection check box.
6. To center a single symbol, select a single symbol and then click the GoTo button.

Net Attributes

Net attributes can be assigned in the Schematic Editor. The editing process is the same as described above for editing pin attributes. You can select multiple nets and assign attributes for all of them simultaneously.

Attribute Windows

An *attribute window is* a predefined area associated with a symbol in which an attribute's value is displayed. The attribute window must have been defined when the symbol was created. Chapter 6, "Using the Symbol Editor," and Chapter 8, "Attributes," explain how attribute windows are added to symbols.

An attribute window can also be associated with a net, and is defined in the Schematic Editor.

The Attribute Display command from the Options menu selects which attributes are displayed in which attribute windows. This is useful if you need to see attributes that are not currently displayed. You can temporarily reassign the attribute window from another attribute to the attribute you want to see.

For example, it's common to have attribute windows for the instance and symbol names, but not simulation delay. If you need to see simulation delay, you can temporarily assign the delay to the instance window. Your reassignments are discarded when you quit the Schematic Editor.

You can also use the Query command at any time to see the values of all attributes associated with a symbol, pin, or net.

Chapter 6
Using the Symbol Editor

The Symbol Editor constructs schematic symbols. Besides the various lines, arcs, and boxes needed to create the symbol, you can also add text to give information about the symbol and its relationship to the rest of the circuit.

Schematics are constructed from symbols. A symbol can represent any electronic component, including capacitors, transistors, integrated circuits, and even microprocessors. Symbols are connected with wires (in the Schematic Editor) to create a complete schematic whose behavior can be verified and simulated.

Note: *A symbol is a picture: it has no inherent electrical meaning. Its electrical characteristics are supplied by attributes that describe the symbol's behavior. (The behavior of a Block symbol is described by the schematic file associated with that Block symbol.) Refer to Chapter 8, "Attributes," for an explanation of how attributes are defined and used.*

SCS is supplied with a extensive set of symbols. You can also use the Symbol Editor to create Block symbols that represent a complete schematic (or part of one). This chapter explains Symbol Editor features you might use to edit these (and other) symbols. It also shows how to display attribute values on a symbol.

You might also want to read the section that explains Master symbols. These can be used to automatically add a title block (or similar annotation) to your symbols and schematics.

This chapter covers the following topics:

- Symbol Components
- Symbol Types
- Creating Symbols
- Adding Attributes to Symbols
- Preparing Symbols for Schematics
- Creating Block Symbols in the Schematic Editor

Note: *You can automatically create symbols using the* **Add: New Block Symbol** *and* **File: Matching Symbol**. *Refer to the SCS Command Reference.*

Symbol Components

A symbol is composed of the following elements:

Graphics

Graphics are the picture of the symbol. They have no electrical meaning; they show the location of the component in the schematic.

Pins

Pins on a symbol are points where a wire can be attached. The pins and wires connect between symbols to circuit elements.

If the symbol represents a physical component, the symbol pin represents the physical pin to which a conductor can be attached. If the symbol represents a subcircuit, the symbol pin represents the connection to an external net of the subcircuit.

Buses cannot be connected to pins unless the pin is a bus pin. Only ordered buses can be connected to bus pins. See Chapter 5, “Using the Schematic Editor,” for more information about net and bus connections.

Attributes

An attribute is a property of a symbol, pin, or net. Attributes can describe *anything*—the length or width of transistors, the price of a resistor, the size of a chip or a cell, the number of connections to a block, or even the length of time it takes to design a cell.

Symbol Types

There are seven symbol types. The symbol type affects the handling of certain symbol attributes.

- Block
- Cell
- Component
- Gate
- Graphic
- Master
- Pin

The symbol type is set in one of three ways:

- A default symbol type can be specified with the INI Editor. The Symbol Editor automatically uses this type when creating a new symbol.
- If no default is set with the INI Editor, the Symbol Editor prompts you for the type when you create a new symbol.
- You can change the symbol type using the Change Type command from the Symbol Editor's Edit menu.

Component and Gate Symbols

These symbols are used in PCB designs. Component symbols represent *complete* physical devices. Gates represent *portions* of physical devices. A complete 7400 quad NAND gate is represented with a Component symbol, while a single NAND gate in a 7400 package is represented with a Gate symbol. Gate symbols let you design a PCB without making the actual packaging assignments. That can be handled later by an automatic PCB packager.

The pins on Component and Gate symbols are numbered to reflect their actual pin assignments in a physical package. Bus pins are not permitted on Gate symbols.

Cell Symbols

Cell symbols represent the primitive cells (transistors, resistors, diodes, and so on) used to design integrated circuits. The pins on Cell symbols are named for identification in netlists.

Block Symbols

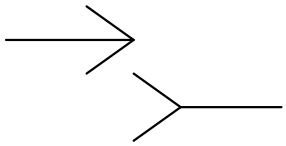
Block symbols are used to build a hierarchical design. A Block symbol represents a schematic at the next-lower level of the hierarchy. Bus pins are permitted on Block symbols.

Pin Symbols

Pin symbols represent physical pins on a PCB. For example, you can define a special type of pin for test points and another for edge connector contacts. Figure 6-1 shows two pin symbols. This symbol represents a test point added to a PCB. The pin symbol includes a pin as well as some graphics to indicate the special purpose of the pin symbol.

Two or more pin symbols can have the same reference designators. This allows you to spread pin symbols across a schematic (even across sheets) and still group them into the same connector by assigning the same reference designator to the different instances. Each instance of a pin symbol with the same reference designator must have a different pin *number*, however.

Figure 6-1
Pin Symbols



Graphic Symbols

Graphic symbols add information that is not part of the circuitry. Graphic symbols are typically used for tables and notes. No pins are associated with Graphic symbols, and they are never included in the hierarchy or netlists.

Master Symbols

Master symbols are used for title blocks, logos, revision blocks, and other standardized graphic symbols. You can add text to a Master symbol to display the company name, address, project description, date, and so on.

You can also display attributes 200 to 206 in attribute windows to automatically show such information as file creation date, sheet number, and schematic name. The use of these attributes is explained in Chapter 8, "Attributes."

Positioning Master Symbols

Master symbols cannot be freely placed on a schematic sheet. Instead, they are automatically positioned at one of the corners. This permits resizing the sheet without having to move the title block (or other annotations).

The sheet corner is determined by the location of the symbol's origin. (Origin placement is explained later in this chapter.) If the origin is placed at the upper-right corner of the Master symbol (for example), the symbol will be positioned at the upper-right corner of the sheet.

Master symbols do not have pins.

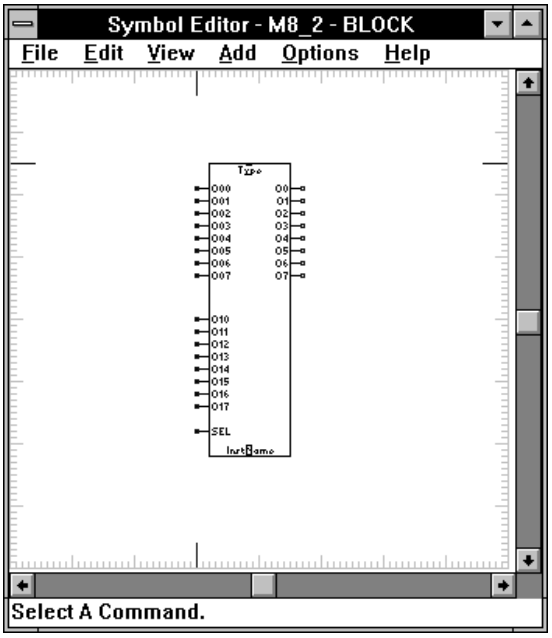
Creating Symbols

This section explains the basics of creating a symbol. The *SCS Command Reference* has more information about the commands in this section.

Starting the Symbol Editor

The Symbol Editor can be run from the SCS Shell, from the Windows menu of the Synario Project Navigator, or using the Symbol command (from the Edit menu) in either the Schematic Editor or the Hierarchy Navigator.

Figure 6-2
Symbol Editor Window



If you're creating a new symbol, the symbol is automatically given the default type specified in the Symbol Controls dialog box from the INI Editor. (You can choose any of the seven types.) If you want a different type, use the Change Type command from the Edit menu of the Symbol Editor.

You can also set the default type to No Default. In this case, the Symbol Editor always prompts you for a type.

Grids

All symbol elements are positioned on a grid. The default spacing of the grid is one-tenth of an inch (or 2.5 mm). (This spacing is set in the Graphic Options dialog box of the INI Editor.)

Graphics are usually placed on the Primary grid, but you can align them with a Secondary grid that has two or four times the resolution. This finer resolution gives more precise control over the position of names, annotations, and graphic embellishments. The Graphic Options command from the Options menu (described below) determines whether alignment is with the Primary or Secondary grids.

Symbol (and Schematic) dimensions are stored as multiples of the Secondary grid units, *not* as absolute lengths. If, for example, you redefine the Primary grid to be 0.2" (when it was previously 0.1"), symbol drawings and schematics will print out at twice their previous size.

Positioning Pins

Although graphics and text can be positioned at any of the three grid spacings, pins *must* be aligned with the Primary grid. Wires are drawn only on the Primary grid. If the pins are not on the Primary grid, you will not be able to attach wires to them.

The tick marks around the outside of the drawing area in Figure 6-2 represent one major grid unit each. The size of the drawing area is initially 80 x 80 Primary grid units. This size can be increased with the Expand command from the File menu. Each time you use the Expand command, the drawing area increases by 20 Primary grid units in each direction. The maximum size of the drawing area is 400 x 400 Primary grid units.

Drawing Graphics and Fixed Text

The graphical rendition of the symbol is created with a combination of lines, rectangles, circles, arcs, and fixed text. Graphic objects can be drawn in two line weights:

Normal	Normal lines are the same width as the wires in a schematic.
Wide	Wide lines are twice the width of Normal lines, the same weight as schematic buses.

The line width is selected from the Graphic Options menu. Changing the line width does not alter the width of lines or objects already drawn, only the width of lines drawn after the change.

The drawing commands are at the bottom of the Add menu. All drawing commands remain active until you click right, or select another command. The *SCS Command Reference* has a full description of the drawing commands.

Lines

When clicking to place the end points, lines are constrained to three principal directions: vertical, horizontal, and 45°. When dragging the line, the line can be at any angle as long as the end points fall on the grid being used. Switch to a finer grid to make it easier to place lines exactly where you want them.

Rectangles

Many symbols are based on a rectangular body. For non-rectangular symbols, such as inverters or multiplexers, use the Line command to draw the outline.

Circles and Arcs

Full circles are placed with the Circle command. Portions of circles can be created with the Arc command. Arcs are useful for the curved sections of NAND and NOR gates.

Negation Bubbles

Negation bubbles are graphical and have no electrical significance. (Adding a negation bubble to a symbol does not change its logic. You must modify the symbol's attributes or underlying schematic file.) You can add small or large bubbles:

- Bubble draws a bubble one-half the Primary grid unit in diameter.
- Big Bubble draws a bubble one Primary grid unit in diameter.

With either command, a bubble is attached to the cursor. Click at the desired point in the schematic to place a bubble.

Text

Text can be added anywhere in the drawing window. Typical uses of text include:

- Notes about the symbol
- Title blocks
- Cross references

Examples of fixed text on a symbol are:

```
Do not exceed 2000 volts ESD on any pin of this device
```

```
This is test point 32
```

Text Size & Justification

Fixed text (as opposed to text appearing in attribute windows) can be drawn in three different sizes. In the Windows version of SCS, eight sizes are available. In the UNIX / Motif version, three sizes are available. Text can be left-justified, right-justified, or centered. The controls for font size and justification are in the Graphics Control dialog box. Defaults for these values can be changed using the INI Editor.

Saving a Symbol

The Save command from the File menu saves the symbol to a disk file. A symbol can be stored in a library for use in many designs, or it can be kept in the design directory for use in a specific design.

If you're saving a new symbol, you're prompted for a name. If you're editing an existing symbol, changes are saved to the existing file.

You can use the Save As command to save the symbol with another file name, which is useful when you're designing several similar symbols. Save the original, modify it, then save the new version with a new name.

Symbol files have the extension **.sym**. The Editor adds this extension automatically when you specify the base name. If you specify a different extension, the Editor replaces it with **.sym**.

Printing the Symbol

Use the Print command from the File menu to print the symbol. The default orientation is *landscape* (long side of the page horizontal). If *portrait* orientation (long side of the page vertical) would allow a larger image, use the Printer Setup command from the File menu to change the orientation.

Editing Symbols

The editing commands provide many ways to modify symbols. A brief description of these commands is presented here. The *SCS Command Reference* has a full description.

Clipboard Commands

Copy, **Cut**, **Paste** — Cut or Copy places the selected object(s) on the Clipboard. Cut objects are removed from the drawing; copied objects remain.

Note: *Symbol objects and schematic objects are not compatible. You cannot paste a schematic object into a symbol drawing, or vice versa.*

Non-Clipboard Commands

Duplicate	Directly copies objects to another place on the same drawing without changing the contents of the Clipboard.
Delete	Deletes previously placed pieces of the drawing without changing the contents of the Clipboard.
Move	Directly moves pieces of a drawing. It's faster than cutting and pasting. The Clipboard's contents are not changed.
Drag	Stretches existing objects. Lines can be lengthened, boxes widened, circles' radii changed and arcs modified. In the Symbol Editor, Drag operates on a single object at a time.
Undo	Reverses the last edit. Any command that changes the symbol can be undone. Commands like View, which don't change the symbol, cannot be undone.
Redo	Reverses the last Undo. Use Redo if you back up too far when Undoing.

Preparing Symbols for Schematics

A symbol needs special *links* so it can be recognized and placed in a schematic. These links consist of pins, attributes, attribute windows, and the symbol origin.

Pins

Symbol pins are connection points for wires. Pins on Gate, Component, and Cell symbols represent the connection points on the device (pins or pads). Pins on a Block symbol represent connections from one level of the hierarchy to the level below. Since Graphic and Master symbols don't represent electrical components, you can't attach pins to them.

Pins are the only symbol elements restricted to locations on the Primary grid, since wires must begin and end on Primary grid points.

Adding Pins

To add a pin to a symbol:

1. Select the Pin command from the Add menu.
2. Click where you want to place a pin.

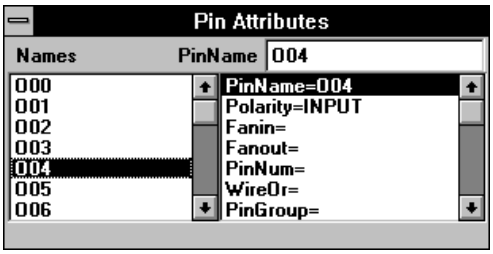
Pins are typically placed at the end of short line segments extending from the sides of the symbol. (Pins can be attached directly to the body of the symbol. However, this makes it more difficult to attach multiple wires to one pin.) Pin names and numbers are displayed next to the pin.

Adding Pin Names

To add pin names:

1. Select the Pin Attribute command from the Add menu to display the Pin Attributes dialog box (Figure 6-3).
2. Select PinName from the attributes listed in the right-hand list box.
3. Click on the desired pin.
4. Click on the Pin Name edit box to select it. Then type the desired name and press ENTER. The name appears on the symbol.

Figure 6-3
Pin Attributes Dialog Box



The Pin Attribute command remains active until you select another command. You can repeat this process to name (or rename) the remaining pins.

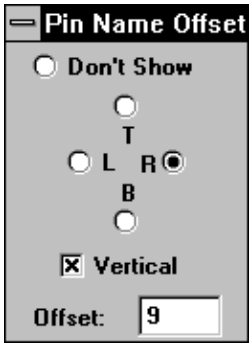
Displaying Pin Names

The Pin Name Location command controls whether a pin's name is displayed, and its position relative to the pin. Selecting the command from the Add menu displays the dialog box below (Figure 6-4):

To change the position of a pin name, click the appropriate radio button (L, R, T, or B), then click on the pin (or drag a box around a group of pins). The L ("left") button displays the name to the right of the pin. The R ("right") button displays the name to the left of the pin. The T ("top") button displays the name below the pin. The B ("bottom") button displays the name above the pin. Mark the Vertical check box to display a pin name as vertical text.

To hide a pin's name, click the Don't Show radio button, then click on the pin.

Figure 6-4
Pin Name Location Dialog Box



To change the distance of the pin name from the pin, click in the Offset edit box and type an offset value (0–127). Then click on the desired pin (or drag a box around a group of pins).

The Offset is measured in Secondary grid units. The maximum Offset is 127. The default value is set in the Symbol Controls dialog box of the INI Editor.

Displaying Pin Numbers

The Pin number display location is automatically calculated based on the position of the pin in the symbol. Pin numbers display can be turned on or off for an entire symbol using the HidePinNumbers attribute (#13).

Bus Pins

A bus pin is needed to connect a bus to a symbol. Naturally, a bus pin must have as many nets or signals as the bus that connects to the pin.

One way to create a bus pin is to give a pin a name of the form:

bus_name[*index1–index2*]

where *bus_name* is the name of an internal bus, and *index1* and *index2* specify the range of signals you want to connect. For example, if you need to connect nine signals, *index1* could be 5 and *index2* could be 13.

Alternatively, a bus pin can be defined by giving it a *compound name*—a list of bus names separated with commas (,):

```
name1, clk, mux[0-3], toggle
```

Bus Pin Limitations

Bus pins are allowed only on Block, Cell, and Component symbols. When a bus pin is created on a Component symbol, the numbers of the physical pins must be specified in the symbol definition.

These pin numbers are a list of pins assigned to the pin attributes BusPin_A through BusPin_H. When assigning bus pins, the normal PinNumber pin attribute *must not* have an assigned value.

The pin list can be divided sequentially among the eight attributes. Each individual attribute can hold about 200 characters. The list is delimited with commas or spaces, and can specify sequences of pins in parentheses **( )** or square brackets **[ ]**. Examples are:

```
BusPin_A = 1, 3, 5, (7:10)    7 pins: 1, 3, 5, 7, 8, 9, 10

BusPin_B = A1 B[2:4] C1       5 pins: A1, B2, B3, B4, C1
```

Attributes

A symbol and each of its pins can have attributes. An attribute has a name and a value.

Attribute names are represented in the symbol's file by an integer. The correspondence between attribute names and integers is defined with the Symbol Attributes and Pin Attributes dialog boxes of the INI Editor. (See Chapter 9, "The SCS INI Editor," for more information.) This arrangement allows the internal numbers to remain constant, while the attribute names change to accommodate local practice or language.

Attribute values assigned in a symbol definition become the default values for each symbol instance. These values are frequently overridden in the completed design. The following sections have brief discussions of symbol attributes. For a full discussion of how attributes are defined and used, please refer to Chapter 8, "Attributes."

Pin Attributes

Pin Attributes are characteristics or properties associated with a pin. PinName, Polarity, Fanin, Fanout, and PinNumber are examples of Pin Attributes. Pin attributes are created with the INI Editor and their values are modified in the Symbol Editor.

To add or change a pin attribute's value:

1. Select the Pin Attribute command from the Add menu. The Pin Attribute Editor dialog box is displayed.
2. Click on the desired pin or pins, or select pin names from the list in the dialog box.
3. Click on the desired attribute in the right-hand list box.

4. Click on the edit box at the top to select it (the box will be labeled with the name of the attribute you selected), and enter the attribute value. The attribute value is assigned to all selected pins.

Most of the standard pin attributes are used by simulators or the Checker. These tools use the pin attributes to analyze the circuit.

Symbol Attributes

Symbol Attributes are characteristics or properties associated with a symbol. Examples of symbol attributes are PartNum, InstName, Width, and Type. The standard symbol attributes (numbered 0–99) are reserved. You can create symbol attributes (numbered 100–199) using the INI Editor.

Any attribute that is not defined as having a fixed value can later be modified in the Schematic Editor or the Hierarchy Navigator using the Attribute command. The procedure is the same as editing pin attributes, described in the preceding section.

Attribute Windows

Attribute windows are predefined areas on or near a symbol or pin in which attribute values are displayed. Attribute windows do not have a visible outline. If no value is displayed, there is no indication that an attribute window has been defined.

Attribute windows are identified by number. The association between an attribute window and an attribute is defined using the Attributes dialog boxes in the INI Editor. An attribute window can have any number; it does not have to match the number of the attribute itself.

When a symbol is rotated or mirrored, the text in attribute windows retains its original position. This keeps it readable.

To add an attribute window to a symbol:

1. Select the Graphics Options command from the Options menu. The Graphics Options dialog box is displayed.
2. Set the Text Font and Justify settings you want for the text in this attribute window.
3. Select the Window command from the Add menu. A list box appears containing all attributes that currently have window numbers.
4. Click on the desired attribute from the list box. The attribute name is attached to the cursor.
5. Click to place the symbol attribute window at the desired position on (or near) the symbol. Be sure the attribute window is placed so the attribute value can be read.

A narrow bar appears above or below the window's name, in a position indicating the text justification. For example, a right-justified name has the bar at the right end of the name.

In the Symbol Editor, the window contains the attribute name. When the symbol is instantiated in a schematic drawing, the attribute window contains the attribute value for that instance, rather than the attribute name.

Set Origin

When a symbol is placed in the Schematic Editor, the symbol is attached to the cursor. The point on the symbol attached to the cursor is called the *origin* of the symbol.

A newly created symbol has no origin. When the symbol is saved, the origin defaults to the upper-left corner of the symbol. You can assign an origin or change the current origin with the Set Origin command in the Symbol Editor.

To set the origin, select the Set Origin command from the Edit menu and click at the desired location. (The origin does not have to be on or within the symbol; it can be outside.) Once an origin is assigned, its coordinates are marked with long pin color tick marks along the border of the symbol window.

Figure 6-2 shows how the origin can be relocated. The long tick marks along the edge of the window point to the origin's new position, at about the center of the symbol.

Note: *The Origin is not part of the symbol. If you move the symbol, the Origin does not move with it. Be sure to reposition the Origin if you move the symbol.*

Checking Symbols

The Check command from the File menu finds errors in finished symbols. The error report is written to a file and then displayed in a pop-up text window. Clicking on an error in the list highlights the error in the drawing.

The following types of errors are detected and reported:

- Block symbols should have a schematic with the same name in the current directory.
- Symbols of type other than Block are usually primitives and should not have a schematic of the same name in the current directory.
- Each pin should have a PinName in a Block or Cell, and a PinNumber in a Gate or Component.

- If a symbol represents a device with more than one gate (such as a 7400 quad NAND gate), then every pin in a gate must appear in the same number of device sections. That is, all the PinNumber attributes must have the same number of entries. (Common, non-unique pins, such as clocks and resets, should repeat the pin number as many times as there are device sections.)
- Pins in Component and Pin symbols can only have one PinNumber.
- Pins in the same group of a Gate must all have the same Polarity, Load and Drive.
- Pins on Block symbols should not have Load or Drive specifications.
- Pins on non-Block symbols should have Load or Drive specified. Input pins should specify Load but not Drive. Output pins should specify Drive, unless the pin is tristate. In that case Load should also be specified, representing the load in the High-Z state. Bidirectional pins should specify both Load and Drive.

"Unconnected Pin" Message

You can tell the Schematic Editor's Check function to ignore intentionally unconnected pins by adding pin attribute #9, OpenOK. Use the INI Editor to add this attribute with the "+" option, "Assign in Schematic."

The attribute can be given a value (Yes, OK, or True) in the Symbol Editor, Schematic Editor, or the Hierarchy Navigator. If a value is assigned in the Symbol Editor, the pin is never flagged as unconnected. Or you can selectively assign a value in the Schematic Editor or Hierarchy Navigator to disable the message on specific pins.

Creating Block Symbols in the Schematic Editor

The New Block Symbol command from the Add menu is a convenient way to create generic Block symbols without leaving the Schematic Editor.

These symbols consist of a rectangle with pin leads. The rectangle's height and width are automatically scaled according to the number of pins and the length of their names. The input pins are placed on the left side and the output pins on the right side. The length of the pin leads is taken from the value of the Default Pin Name Offset parameter in the Symbol Controls dialog box of the INI Editor.

The symbol also has an attribute window (near the top) for displaying the symbol name, and another (near the bottom) for displaying the instance name.

A dialog box lets you specify the symbol name, input pins, output pins, and bidirectional pins. The pin names should be separated with commas. There are four edit boxes, one for each item:

Name	The name of the Block symbol
Inputs	The names of all nets marked as inputs with I/O Markers
Outputs	The names of all nets marked as outputs with I/O Markers
BiDirs	The names of all nets marked as bidirectional with I/O Markers

If a bus name appears in one of these fields, it must be enclosed in equals signs (=*busname*=). If a compound name appears in the pins list and is not surrounded by equals signs, it is expanded to produce individual pins for each name. If the name is enclosed by equals signs, it produces a bus pin. For example:

Name Entered in Input Field	**Pins Created**
`A,B[0:3],C`	6 pins: A B0 B1 B2 B3 C
`A,=B[0:3]=,C`	3 pins: A B[0:3] C
`=A,B[0:3],C=`	1 pin: A,B[0:3],C

Once this information has been entered, the symbol can be edited or placed. To edit the symbol with the Symbol Editor, click the Edit button. To create the symbol without editing it, click the Run button. If you click the Run button, the newly created symbol is attached to the mouse cursor for immediate placement. In either case, the symbol is created in the current directory.

Making a Block Symbol for the Loaded Schematic

You can use the New Block Symbol to create a Block symbol for the currently loaded schematic by clicking on the “Use Data from this Block” button. The edit fields are automatically filled with the correct values from the schematic. However, it is easier to use the Matching Symbol command from the File menu.

Chapter 7
Using the Hierarchy Navigator

The Hierarchy Navigator loads a full hierarchical design all at once, so that you can view it in its complete form (rather than as modules). Every schematic at all hierarchical levels is included. You can trace signals and connectivity throughout the full design.

The Hierarchy Navigator is also the bridge between the individual schematics created in the Schematic Editor and a simulator or other tools that require the entire design to be in a single database.

This chapter covers the following topics:

- Hierarchy Navigator Functions
- Navigating a Design
- Tracing Signals
- Using the Waveform Viewer
- Setting and Overriding Attributes
- Additional Hierarchy Navigator Features
- Analysis Tools
- ERC and PCB Checkers
- Viewing Critical Paths
- The View Report Utility
- Netlists and Interfaces
- The Packager

Hierarchy Navigator Functions

The Hierarchy Navigator performs several important functions:

- It verifies the correctness and consistency of a design's wiring. Verification occurs at each level in the design, and across all the levels, from "top" to "bottom."
- It provides the environment in which you can analyze and optimize the circuit's performance.
- It prepares the design data for later steps in the design process (for example, creating netlists).
- It allows back annotation, to permit modifying or optimizing individual component instances.

The following sections explain the basic concepts of the Hierarchy Navigator.

Navigating a Design

You can start the Hierarchy Navigator from the Synario Project Navigator or from SCS Executive.

To start the Hierarchy Navigator from the Synario Project Navigator:

1. Select the top-level schematic from the Sources window.
2. Double-click on Navigate Hierarchy in the Processes window.

To start the Hierarchy Navigator from the SCS Executive:

1. Click the Navigate Hierarchy radio button. The list box immediately displays all the ***.tre** files in the current directory.
2. If the current directory is not the one you want, double-click on a directory name or the double dots [..] to change the directory.
3. Double-click on the name of the ***.tre** file you want. Or highlight the name of the file and click the RUN button. The Hierarchy Navigator runs and loads the file.

The Hierarchy Navigator's Sheet command lets you traverse a design laterally, at the current hierarchy level. The Push / Pop command moves you down and up (respectively) through the various hierarchical levels. These commands are explained in detail later in this chapter.

Updating Schematics

Symbol files are marked with the time and date they were last modified. Schematic files keep track of this time stamp for each symbol. If a symbol file's time stamp is different from that symbol's time stamp in a schematic, the schematic is out of date.

The Hierarchy Navigator assumes that discrepancies in the date indicate a potential problem—the wrong symbol may have been accessed, or someone has unofficially modified a symbol—and displays warning messages. If only minor changes were made to the symbol, the warnings can be ignored. If a significant change was made (such as a new pin, new pin name, or new symbol origin) any schematics containing that symbol must be revised.

If the changes to a symbol were minor, and the only discrepancy is the date change, the Schematic Editor can update the schematic for you. Load the schematic into the Schematic Editor, then save it. The new symbol dates will replace the old ones. Or, use the Update Schem utility from the File menu of the SCS Executive.

Push/Pop

The Push / Pop command moves you higher or lower in the hierarchy.

Push Moves to a lower (more detailed) level. Click within a symbol to view the symbol's internals. If the symbol represents a lower-level schematic, that schematic replaces the current schematic (unless the symbol is at the lowest level of the hierarchy).

If the lower level file is behavioral (that is, a file with a text description of the circuit, rather than a schematic), the text file will be loaded into the viewer program and displayed.

Pop Moves to a higher (less detailed) level. To pop back to the "parent" of the current schematic, click outside all symbol boundaries. The parent schematic replaces the current schematic.

Alternatively, you can move up or down by typing the instance name of the destination schematic on the prompt line, then pressing ENTER. Enter a period (.) to pop to the top-level schematic

The full hierarchical context is displayed when pushing or popping. Net names are shown with their complete hierarchy. Symbol instance names are shown in an abbreviated format that replaces the leading portion of an instance name with a period (**.**); the leading part is displayed on the title bar of the Navigator. For example, the instance .AB.CD.EF is displayed as .EF, with the prefix .AB.CD in the title bar.

Push / Pop is a "nested" command, so you can call Push / Pop during another command, such as Query or Edit Attribute. Click right anywhere in the window to return control to the previously active command.

Tracing Signals

Although design problems are usually observed at the top level, the source of those problems is often at a lower level. Tracing signals from the primary outputs down through the hierarchy can greatly aid debugging.

There are two Navigator functions to facilitate signal tracing, Mark and Query.

Mark

The Mark command from the Object menu highlights selected nets or symbols. This makes it easy to trace signals from one side of a sheet to the other, across sheets, or through the hierarchy.

Mark is used in some of the simulation interfaces to choose waveforms for display. Any marked item can be displayed in a list box by typing a question mark (**?**) in the prompt line. See the *SCS Command Reference* for further details.

Query

The Query command from the Object menu provides a brief summary of the selected element's *local* attributes, as well as connections to the element. Local attributes can help you find an incorrectly specified simulation parameter or loading characteristic. The connectivity information is helpful in circuit tracing.

Click on the individual connections listed in the query pop-up window, and the display shifts to the area of the schematic containing the selected connections. You can quickly find all connections to a given net, or the Pin driving a particular net.

The Query command can search for elements you type in at the prompt and display information about them in a list box. You can search for an element based on its:

- Instance name
- Pin name
- Net name
- Reference designator

You can query pins, nets, and individual symbols. To query all symbols of one type (for example, all three-input NAND gates), press SHIFT while you click on an instance of that symbol. Individual symbols can be located based on instance name or reference designator. The information is displayed in a text window. The information available for each element is:

Pins

- Pin name
- Attached to which instance
- Attached to which net
- Pin polarity
- All other attributes

Nets

- Net name
- Polarity if input or output node
- Local net name, applicable only on external nets
- Node number in database
- Symbol connections
- All other attributes

Individual Symbols

- The name (for example, NAND2, NOR3) and type (gate, component, block, cell, master, pin) of symbol
- Full path showing location of symbol file in the file structure
- Instance name
- Reference designator
- Other symbol attributes
- Reference location on the schematic (sheet number, vertical border reference, horizontal border reference, for example, 2A6)
- Gate section (A, B, C, etc.) on gate symbols
- Instance number in the hierarchical database
- Pin / net connections
- All other attributes

All Symbols of a Given Type

- Internal net count
- Instance names and locations
- All other attributes

Setting and Overriding Attributes

You can use the Hierarchy Navigator to override attribute values that were assigned in the Symbol Editor or the Schematic Editor. (Attributes are described in Chapter 8, "Attributes.")

If your schematic has two tristate buffer symbols: one buffer needs to be large, while the other can be small. You can customize each buffer instance in the Hierarchy Navigator by changing its transistor size attributes.

The Navigator is the only place where you can change the attribute values of specific instances in the hierarchy (since the Navigator is the only program in which all levels of the hierarchy are combined). Many design modifications are made based on results obtained when running the Navigator and a simulator together, and you can incorporate these changes in the Navigator.

Note: *Some external tools use netlists from a single level of the hierarchy (run from the schematic database rather than the hierarchical database), so attributes assigned in the Hierarchy Navigator are not available. If your tool creates a netlist from a single schematic, assign attributes in the Schematic Editor.*

Pin Attributes

To modify pin attributes:

1. Select Pin Attribute from the Edit menu. The Pin Attribute Editor dialog box appears.
2. Click on the pin to be modified, or click on the symbol containing the pin to select all pins in the symbol.
3. Highlight the required attribute from the list.
4. Type the new attribute value and press ENTER.

Symbol Attributes

Symbol attributes can be assigned in either the Symbol Editor, Schematic Editor, or Hierarchy Navigator. The editing process is the same as described above for editing pin attributes. You can select multiple symbols and assign attributes for all of them simultaneously.

You can control whether an attribute's original value, as defined in the Symbol Editor, can be overridden on the schematic. This is described in Chapter 8, "Attributes."

Net Attributes

Net attributes can be assigned in the Schematic Editor or Hierarchy Navigator. The editing process is the same as described above for editing pin attributes. You can select multiple nets and assign attributes for all of them simultaneously.

Attribute Windows

An *attribute window is* a predefined area associated with a symbol in which an attribute's value is displayed. The attribute window must have been defined when the symbol was created. Chapter 6, "Using the Symbol Editor," and Chapter 8, "Attributes," explain how attribute windows are added to symbols.

An attribute window can also be associated with a net, and is defined in the Schematic Editor.

The Attribute Display command from the Options menu selects which attributes are displayed in which attribute windows. This is useful if you need to see attributes that are not currently displayed. You can temporarily reassign the attribute window from another attribute to the attribute you want to see.

For example, it's common to have attribute windows for the instance and symbol names, but not simulation delay. If you need to see simulation delay, you can temporarily assign the delay to the instance window. Your reassignments are discarded when you quit the Schematic Editor.

Additional Hierarchy Navigator Features

Save

The Save command records the display context of the schematics being viewed. This includes marked nets, the hierarchy level, the particular view and magnification. Save also records any attributes that were added to the hierarchy in the current session.

The context file takes the base name of the root (top-level) schematic. The extension .**tre** is added.

Sheet

The Sheet command from the File menu selects which sheet (of a multi-sheet schematic) to view. Each sheet is displayed in a separate window, which can be closed, resized, or repositioned as any other window. When there is only one sheet window, it is enlarged to fill the main window.

Several sheets can be displayed at the same time. Each sheet's window is offset slightly from the previous sheet's window, so that some of the previous sheet can be seen. Click on any sheet's window to bring it to the front.

Up to three views of a single sheet can be opened. Each window can have a different magnification ("zoom factor"). A total of eight windows can be open at the same time.

Print

Use the Print command from the File menu to print the schematic currently being viewed. The Print dialog box lists all the sheets of the current schematic. To print a specific sheet, click on its number in the list box, then click on This Sheet. To print all the sheets in a multi-sheet schematic, click on All Sheets.

Clicking on the All Instances button prints each component instance on a separate sheet. If your schematic has a large number of instances, the program will print an equally large number of sheets. Be sure that this is what you want before selecting this command.

Select the Print Image command to print a section of the schematic. The mouse cursor turns into cross hairs. Drag the mouse to select a rectangular area for printing. (The area is shown as a dotted rectangle.) This area is sent to the printer when you release the mouse button.

Control

The Display Options command from the Options menu sets a number of display options in the Hierarchy Navigator. Changes last only for the current session. You must use the INI Editor to make permanent changes to the default values.

The following display options can be modified. Please refer to the *SCS Command Reference* for detailed information about each option.

- Connect dots
- Border
- Symbol pins
- Pin numbers
- Symbol text
- Symbol attributes
- Open ends
- Off Page connects
- Simulation values
- Node numbers
- Net attributes

Statistics

The Statistics command from the File menu summarizes how many elements are placed in the design and how much memory is consumed by various records in the database. The quantity of each type of the following primitives is listed:

- Types of symbols
- Primitive cells
- Hierarchical blocks
- Instances
- Instance pins
- Primitive instances
- Primitive pins
- Nets connected

This is followed by a list of ten types of database records. Each record type is followed by a measure of the capacity used in a particular design. Five of the ten record types are fixed length and are reported as number consumed, number available, and percentage of the memory block consumed.

- Definitions (types)
- Instances
- Nets

- Pins
- Generic pins

The five remaining types are attribute records. These records are of variable length, so the number of bytes used and the percentage of the memory block consumed are shown, not the number of records.

- Type Attributes
- Instance Attributes
- Net Attributes
- Pin Attributes
- Generic Pin Attributes

Analysis Tools

The *SCS Command Reference* has a complete description of the Check commands.

Both the Schematic and Symbol Editors have checking functions that catch basic errors, such as unconnected pins and shorted nets. However, some errors are caught only in the context of the complete design. For example, net loading can be influenced across many sheets and at many levels in the hierarchy.

Electrical Rules Checking

The Electrical Rules Checker (ERC) checks basic connectivity and current loading (fanout) on a completed design. The Checker shortens turn-around time by locating many errors prior to simulation.

Printed Circuit Board Checker

The Printed Circuit Board Checker (part of the PCB Package option) performs a packaging check on PCB schematics. It ensures the correct assignment of gates to packages.

Simulator Support

If you want to use a simulator and have the appropriate netlist translator, you can add a simulator menu to the Hierarchy Navigator by specifying a simulator in the System Controls dialog box of the INI Editor, and providing a *simulator*.ini file. Refer to Appendix B, "Simulator Interfaces."

The process called from the menu does not have to be a single program. It can also be a DOS batch (**.bat**) file that calls multiple programs and DOS commands.

ERC and PCB Checkers

This section describes the setup and operation of the Electrical Rules Checker and the Printed Circuit Board Checker.

Checker Attributes

Eight attributes are used by the PCB Checker. These attributes are defined with the INI Editor and provide the information needed to perform many of the required checks. (Attributes and their modification are explained in detail in Chapter 8, “Attributes,” and Chapter 9, “The SCS INI Editor.”)

Polarity Polarity determines which pins are input pins and which are output. If Polarity is not specified, Input is assumed. The possible values are Input, Output, and Bidirectional. Only the first character is needed (I, O, or B).

FanIn FanIn represents the load presented by a pin. Fanout analysis checks that the FanIn loading a net is less than the sum of all FanOuts driving that net. This is a relative measurement, so the units for FanIn can be arbitrary, as long as they match the units for FanOut.

FanOut FanOut represents the driving capability of a pin. Fanout analysis checks that the FanOut driving a net is greater than the sum of all FanIns loading that net. This is a relative measurement, so the units for FanOut can be arbitrary, as long as they match the units for FanIn.

LoadLow LoadLow is used for Hi/Lo Loading Analysis. LoadLow represents the load of the input pin when the net is in the logic low state. This is a relative measurement, so the units for LoadLow can be arbitrary as long as they match the units for DriveLow.

DriveLow DriveLow is used for Hi/Lo Loading Analysis. DriveLow represents the drive capability of the output pin when the net is in the logic low state. This is a relative measurement, so the units for DriveLow can be arbitrary as long as they match the units for LoadLow.

LoadHigh LoadHigh is used for Hi/Lo Loading Analysis. LoadHigh represents the load of the input pin when the net is in the logic high state. This is a relative measurement, so the units for LoadHigh can be arbitrary as long as they match the units for DriveHigh.

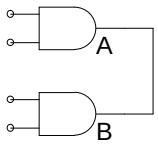
DriveHigh DriveHigh is used for Hi/Lo Loading Analysis. DriveHigh represents the drive of the output pin when the net is in the logic high state. This is a relative measurement, so the units for DriveHigh can be arbitrary as long as they match the units for LoadHigh.

WireOr WireOr indicates that a pin has tristate, open-collector, or wired-OR outputs. The values for this attribute are "Yes" for wired-OR, "O" for open-collector, and "Tri" for tristate. Only the first character (Y, O, or T) is needed. Output pins with WireOr specified as tristate should have both a Load and a Drive. The Load represents the Load of the pin when it's in the tristate condition.

If you connect the outputs of two gates (as shown in Figure 7-1), the Schematic Editor *will not* flag it as an error. The ERC Checker determines its legality according to the WireOr attribute values.

If both gates are wired-OR ("Yes"), both are open-collector, or both are tristate, the connection is valid. If the attribute values are different, the connection is invalid.

Figure 7-1
Two Gates with Outputs Connected Together



The following attributes are used only with the PCB Checker:

RefDes This attribute represents the reference designator of a part on a printed circuit schematic. It must be unique for Component symbols (since two parts can't have the same designator). Gates which are part of the same package have the same RefDes. Pin symbols, used for edge connectors, can have the same RefDes if they belong to the same connector.

PinNumber This pin attribute represents the number of the physical pin on the package. For Components, this attribute is a single number. When a Gate symbol is created, its PinNumber attribute is set to a list of numbers representing the pin number in each of its possible sections.

When the Gate is assigned to a package with the Add Pin Numbers command of the Schematic Editor (or the Pin Numbers command of the Hierarchy Navigator), the PinNumber attribute takes on the actual number of the pin for the section it represents. Some packages have Gates with common clock or enable signals. In this case, the PinNumber list repeats the common pin number for each section.

PinGroup If this pin attribute has a value greater than zero, it indicates that certain pins belong to the same group and can be swapped. If two pins of the same Polarity have the same PinGroup, they can be swapped using the Add Pin Numbers command in the Schematic Editor or Pin Numbers command the Hierarchy Navigator.

CompName Identifies symbols that comprise sections of the same physical component.

These attributes are assigned to the symbol and its pins using the Symbol Attribute and Pin Attribute commands of the Symbol Editor. If a symbol has an assigned attribute, it becomes the default value for each instance of the symbol on the schematic. The default values can be overridden for a particular instance of the symbol on the schematic with the Attribute command of the Schematic Editor or Hierarchy Navigator.

Types of Analysis Performed by the Checkers

Fanout Analysis

Fanout analysis checks that the FanOut on the driving pin of a net is greater than the sum of all FanIns on all load pins. This is a relative measurement, so the units for FanOut can be arbitrary, as long as they match the units for FanIn.

The attributes required for fanout analysis are attached to pins as follows:

Pins	Attributes
Input	Polarity = Input; FanIn specified
Output	Polarity = Output; FanOut specified
	WireOr = T, Y, or O (Tristate, Yes, Open-collector). The WireOr attribute is optional.
Bidirectional	Polarity = Bidir; Both FanIn and FanOut specified

In addition to the above pin specifications, tristate or open-collector output pins should have the WireOr attribute set to Y, O, or T (Yes, Open-collector, or Tristate). Bidirectional pins can have the WireOr attribute set only if the pin is part of a block symbol. Bidirectional pins on primitive symbols should not have the WireOr attribute specified.

Hi / Low Analysis

Hi/Low analysis is more comprehensive than a fanout analysis. The loads and drives are compared for low and high logic values separately. This allows logic families such as TTL, which has strong drive when driving low and weak drive when driving high, to be correctly analyzed. The attributes required for the symbol pins in the Hi/Low analysis are:

Pins	Attributes
Input	Polarity = Input; LoadLow and LoadHigh specified
Output	Polarity = Output; DriveLow and DriveHigh specified
Bidirectional	Polarity = BiDir; LoadLow, LoadHigh, DriveLow, and DriveHigh specified

In addition to the above pin specifications, tristate or open-collector output pins should have the WireOr attribute set to Y, O, or T (Yes, Open collector, or Tristate). Bidirectional pins can have the WireOr attribute set only if the pin is part of a block symbol. Bidirectional pins on primitive symbols should not have the WireOr attribute specified.

Subcircuit Analysis

You might sometimes want to analyze the loading or fanout of only a portion of the circuit. In such cases the input and output pins from the top level block don't have any drive or load (respectively) specified. The electrical Rules Checker handles this in the following way.

The pin attributes of the top level Block symbol determine the load and drive from external sources. The value of the load attribute (FanIn, LoadLow, and LoadHigh) on an input pin represent the minimum external drive capability required. This is checked against the loading on the lower-level schematics.

The value of the driving attribute (FanOut, DriveLow, and DriveHigh) on an output pin represents the maximum external load. This value is checked against the drive on the lower-level schematic. Also, if an external input pin is to be wire-ORed with an internal output pin, the pin in the top level symbol should have its WireOr attribute set.

For PCBs, primitives should be either Gates or Components. Each primitive symbol instance should have a RefDes assigned. Each pin must have a PinNumber. Pins that can be swapped should have the same PinGroup. Pins that represent pins in multiple sections should have a list of numbers in the PinNumber attribute.

Operating the Electrical Rules Checker

The Electrical Rules Checker is typically called from the Processes menu of the Hierarchy Navigator. You specify the command line in the Processes section of the INI file. The Electrical Rules Checker has the following command line options:

`/A`	Automatic mode; do not display dialog box.
`/H`	Perform High / Low current loading analysis.
`/L`	Perform fanout analysis.
`/W`	Issue warnings about nets with no load.

The checker displays the Options dialog box if the /A option is not specified. If other options are specified without the /A option, the dialog box displays the selected options as the defaults. If the /A option is specified the checker does not display the dialog box, but does use any other command line options.

To change the electrical rules checker options:

The default options are /A and /W.

1. Run the INI Editor from the Setup command of the SCS Executive.
2. Choose Open from the File menu to load the INI file you wish to modify.
3. Select Navigator Tools from the Tools menu.
4. Change the options in the Flags edit box.
5. Close the dialog box and choose Save from the File menu to save your changes.

The ERC Checker selection from the Tools menu of the Hierarchy Navigator gives you several check box options, described below.

Check Box	Function
Electrical Rules Only	Performs basic connectivity checking, and warns about open pins and unconnected nets. Each net is checked to confirm that it is driven by the output of some gate. Multiple outputs driving a single net cause warning messages unless all outputs are open-collector or tristate.
Fanout Analysis	Performs fanout analysis in addition to the connectivity checking. An error message is displayed if the sum of the loads (FanIn) on a net exceeds the capacity (FanOut) of the driving gate.
Hi/Lo Current Loading	Performs a loading analysis on both the high and low states. This is useful when the high and low drive capabilities are not the same (as in TTL). This check is the same as Fanout analysis, except it compares LoadLow versus DriveLow and LoadHigh versus DriveHigh.
Report No-Load Nets	Generates a warning for each net that does not drive any inputs.
Check	Invokes the Electrical Rules Checker and writes any errors found to the file *design*.**err**. The View Report command of the Hierarchy Navigator is invoked to display the errors in a list box as described below.
Cancel	Terminates execution of the Electrical Rules Checker.

Each error is written as a separate line in the file *design*.**err** (where *design* is the base name of the design). The error messages are automatically displayed in a list box. Clicking on a message causes the schematic to pan and / or traverse the hierarchy to show the offending net or symbol. This net or symbol is highlighted and a "plus" mark is placed as close as possible to the error condition. After correcting each error, you can call this tool again to find the next error in your design.

Error Messages for the ERC

For the Fanout Analysis, *LOAD* means FanIn and *DRIVE* means FanOut.

For the Hi / Lo Current Loading Analysis, *LOAD* means either LoadHigh or LoadLow and *DRIVE* means either DriveHigh or DriveLow.

`DRIVE Specified for INPUT Pin = pin_name`

An input pin should not have a drive.

`Invalid LOAD specified for Pin = pin_name`

The load attribute must be a number.

`Invalid DRIVE specified for Pin = pin_name`

The drive attribute must be a number.

`LOAD (100) > DRIVE (50) in Net = net_name`

The total of all the loads on the net is greater than the output capacity of the driving pin.

`LOAD Specified for OUTPUT Pin = pin_name`

An output pin should only have a load if it is a tristate pin (WireOr = T).

`No DRIVE Specified for BIDIR Pin = pin_name`

Every bidirectional pin must have a Drive.

`No DRIVE Specified for OUTPUT Pin = pin_name`

Every output pin must have a Drive.

`No LOAD Specified for BIDIR Pin = pin_name`

Every bidirectional pin must have a Load.

`No LOAD Specified for INPUT Pin = pin_name`

Every input pin must have a load.

`No LOAD Specified for Tristate OUTPUT Pin = pin_name`

An output pin must have a load if it is a tristate pin (WireOr = T).

`There are no Loads in Net = net_name`

Every net should drive some input pin.

`There are no PINS in Net = net_name`

Every net should be connected to at least one pin.

There is no Drive pin in Net = net_name

Every net should be driven by some output pin.

Unconnected Input Pin = pin_name

Each Input or Bidirectional pin must be connected to a net.

Warning—Wired OR in Net = net_name

If more than one output pin is driving a net, each of the output pins should have the WireOr attribute set to Tristate, OpenCollector, or Yes. All the WireOr attributes must have the same value.

Operating the PCB Checker

The Printed Circuit Checker is called from the Tools menu of the Hierarchy Navigator. The checker uses two principal criteria to verify the correct assignment of Gates and Components:

- If two symbols are assigned the same reference designator, they must be the same type of Gate. For example, the four NAND gates in a 7400 device are represented by the same symbol with the same reference designator (since all four gates are in the same physical package). The reference designator (RefDes attribute) assigns the symbol to a package.
- Each Gate in a package can be used only once. Each section of a package must have different pin number assignments. If a package has common pins (for example, a 74175 has four D-type flip flops with a common clock and common clear), then the common pins must each be connected to the same net.

Errors are written to a file called *design*.**err** and displayed in list box, one error on each line. Clicking on a line highlights the net or symbol with that error on the schematic. You can use this feature to step through the errors one at a time.

Error Messages for the PCB Checker

Duplicate Assignment of RefDes = name

Components must have unique reference designators.

Encountered CELL Type Symbol = name

Each primitive symbol in a Printed Circuit design should be a Gate or Component. Cell primitives are used in IC designs.

Missing Connector Name on I/O Pin Instance = name

Symbols of type Pin should have a connector name. Use the Reference Designator command to add one.

Missing Pin Number(s) on Symbol Type = name

Each pin must have a pin number.

Missing RefDes on Symbol Instance = name

Each Gate or Component must have a reference designator assigned.

Missing Schematic file for Block Symbol = name

Non-primitive symbols (Blocks) must have an underlying schematic (.**sch**) file.

Two Instances of Pin = name *on Different Nets*

Two Gates of the same part have a common pin wired to different nets. Or, two Gates of a package have the same pin numbers.

RefDes name *Assigned to Different Symbols*

If two symbols have the same reference designator, they belong in the same package. They must be of the same symbol type. (DeMorgan equivalents are considered the same.)

The View Report Utility

SCS has a standardized method for reporting and displaying errors. Errors are written to a file and the file's contents are displayed in a list box. When you click on an error, the display scans to place the error near the center of the window. The error is highlighted (usually with a small cross).

The View Report command is in the File menu of the Hierarchy Navigator. A dialog box prompts you to choose a file. Its contents are displayed in a list box. Netlisters use the standard View Report interface to display error messages in the Navigator. Third-party programs can also use this interface.

The View Report command can jump to and highlight specified nets, instances, pins, or symbol types in a design. When you click on a line in the list box, that line is searched for a keyword followed by a valid identifier. The keyword and identifier are separated by an equals sign (**=**). The equals sign can have any number of or no leading or trailing spaces. The keywords are:

I, **Inst**, or **Instance**	An instance. Identifier is instance name.
N or **Net**	A net. Identifier is net name.
P or **Pin**	A pin. Identifier is pin name.
T, **Type**, or **Symbol**	A symbol type. Identifier is type or filename.

You can enter the keyword and identifier in any combination of upper- and lowercase; they are not case-sensitive. The keyword and identifier can appear anywhere in the line, including within a comment. Examples of lines in the list box are:

```
I=adf.pdiff   Bad SPICE attribute on this line
Unconnected pin  =  adf.pdiff-input1
The following symbol=NAND2 has a connectivity problem
```

Clicking on the first example line causes the Navigator to jump to schematic ADF, which contains instance PDIFF. Instance PDIFF is highlighted. The text of the error message is ignored. This “free formatting” allows great flexibility in formatting the error message.

Viewing Critical Paths

A *critical path* is the signal path in a design that has the longest propagation time. The View Report command can display critical paths. The required syntax is shown in the example below:

```
[path 1]
first net=.input
first inst=.adder.nand3
second net=.adder.nand3.N_23

[path 2]
first net=.cin
first inst=.adder.mux
second net=.adder.mux.control
```

Header information appears inside square brackets. Items in each sublist below the corresponding header are treated as described for View Report. When the command is first invoked, a list box displays the lines from each header. Clicking on a header line displays a second list box containing the individual entries for the selected header.

Clicking on a line in the second list box causes the Hierarchy Navigator to jump to the instance or net in the selected line. The keywords and identifiers described for View Report is valid here, too.

This syntax can be used to view any group of errors or features in a design. For example, a checking program that can identify ten different types of errors could write a file in the following general format:

```
[errors of type 1, nets with more than 7 connections]
first error of type 1 found on net=.input
second error of type 1 found on net=.adder.sub.n_6
third error of type 1 found on net=.clock
…

[errors of type 2, instances with more than 10 pins]
first error of type 2 found on inst =.control_block
second error of type 2 found on inst = .reg.mux
third error of type 2 found on inst = .ram.decode1.I_3
…
```

Netlists and Interfaces

Netlists are the usual mechanism for moving a design from one step in the design cycle to the next. A netlist describes all the components in a design and the way they're connected ("connectivity").

Netlists can also be used as input to your own software. For example, if you've written your own timing verification program, you can output a netlist from the Hierarchy Navigator for the timing verification program to read.

SCS supports back annotation from any analysis tool to the design. You can also use the ASCII interface (with caution) to alter the schematic in ways that are not supported by the supplied back annotation interfaces.

The Hierarchy Navigator supports several standard or generic netlists:

- EDIF
- ASCII
- Generic netlist by net
- Generic netlist by pin
- Verilog
- VHDL
- SPICE (various)
- Timemill
- X-Sim
- Racal-Redac
- PADS PCB
- Cadnetix

These netlisters are available under the Tools or Processes menu in the Hierarchy Navigator, and are described in Chapter 10, "PCB Design Considerations," and in Appendixes A and B.

Note: *Some of the netlisters listed above come with the base product, some come with the PCB Package option, and some come with the Simulation Language option.*

The Packager

The Packager is an optional module that can automatically assign reference designators and pin numbers to PCB gate, component, and pin symbols. The Packager can be accessed from the Schematic Editor for flat designs, or from the Hierarchy Navigator for hierarchical designs.

Once it is installed, the Packager's commands are available only if you use the System Controls dialog box in the INI Editor to set the Application mode to "PCB Only" or "Both IC & PCB." The *SCS Command Reference* has a complete description of the packaging commands.

Chapter 8
Attributes

An attribute is a characteristic or property belonging to, or associated with, a symbol, pin, or net. For example, attributes can describe:

- Width or length of transistors, or the price of a resistor
- Size of a chip or a cell
- Number of connections to a block
- Delay from input to output
- Length of time taken to design a symbol

This chapter covers the following topics:

- Attribute Functions
- Attribute Types
- Attribute Components
- Modifying Attributes
- Creating New Attributes
- Number Notation in Attributes
- Derived Attributes

Attribute Functions

The principal source of information about a symbol's electrical characteristics and behavior is the attribute values attached to it. Your simulator uses these attributes to analyze and simulate the schematics you design.

Attribute values in a symbol definition become the default values for each symbol instance. These values are frequently overridden in the completed design, usually to optimize its performance.

Attributes that apply to all instances of a symbol (such as the vendor part number and the pin polarity) are generally assigned values when the symbol is created. Attributes that apply to a single instance (such as the instance name) are assigned after a symbol has been placed in the design.

The symbol libraries supplied with Synario have predefined values for all the attributes required by the Synario Simulator. Chapter 9, "The SCS INI Editor," explains how to modify attributes.

Attribute Types

There are four attribute types:

Global — Global attributes are constants such as feature size, supply voltage, or identification codes. These attributes are accessible from every sheet of every schematic at every level of hierarchy.

Symbol — Symbol attributes describe features related to the whole symbol. Examples are the width and length parameters of transistors, or SPICE-model characteristics. Symbol attributes usually apply only to the symbol on which they appear.

Pin — Pin attributes describe features related to individual pins. Polarity, lead number, drive capability, and loading are typical pin attributes. Pin attributes are accessible at the instance level and can be modified in both the Schematic Editor and Hierarchy Navigator.

Net — Net attributes describe characteristics associated with nets. A good example is the stray capacitance of a net routed across a chip.

Attribute Components

An attribute has five components: name, number, value, modifier, and window.

Attribute Name

An attribute's *name* identifies it to the user. Width, Length, RefDes and PinNumber are examples of attribute names.

Attribute Number

The attribute's *number* identifies it to the Editors and the Hierarchy Navigator. Synario uses the number, *not* the name, to reference an attribute to allow a different name to be assigned without changing the meaning or use of the attribute. The connection between an attribute's name and its number is defined in the SCS initialization file. (Attributes 0–99 are reserved for Synario, the Editors, the Hierarchy Navigator, and simulation. Most of them have predefined meanings.)

Attribute Value

An attribute can be assigned a *value*. A value is usually a number or a text string.

Attribute Modifier

An *attribute modifier* specifies the conditions under which an attribute's value can be modified. The attribute modifiers are grouped based on where you can edit their values:

- Anywhere in Design (<blank>)
- Not Editable (!)
- Symbol Only (-)
- Symbol or Schematic ($)
- Derived (*)

Attribute modifiers are fully described later in this chapter and in Chapter 9, "The SCS INI Editor."

Attribute Window

Attribute values are displayed in *attribute windows*. Attribute values cannot be displayed unless the symbol has at least one attribute window.

You add attribute windows to a symbol when you define the symbol. Each window is assigned a unique number and the default attribute that will be displayed in that window. (The window number *does not* have to match the number of the assigned attribute.) When the symbol is placed in a schematic, the value of the assigned attribute appears in the window.

You can temporarily change which attribute is displayed in an attribute window, using the Attribute Display command from the Options menu. This is useful when you need to view attributes that are not currently displayed.

Attribute windows in schematics can be repositioned, one at a time, with the Attribute Location command from the Edit menu. Repositioning can make a crowded schematic more readable.

Note: *Attribute windows do not have visible outlines. Rather, they are predefined areas on or near the symbol.*

Modifying Attributes

See the *SCS Command Reference* for directions on editing attributes.

Symbol Attributes

Symbol attributes are characteristics or properties associated with the full symbol, such as PartNum, Prefix, and Value.

Table 9-3, in Chapter 9, “The SCS INI Editor,” lists the default symbol attributes and their meanings.

Pin Attributes

Pin attributes are characteristics or properties associated with pins, such as PinName, Polarity, Fanin, Fanout, and PinNumber.

Pin attributes that are assigned values in the symbol definition can be edited in the Schematic Editor.

Table 9-2, in Chapter 9, “The SCS INI Editor,” lists the default pin attributes and their meanings.

Net Attributes

Net attributes are characteristics or properties associated with pins, such as Cap, Length, Width, and VHDLNetType.

Table 9-1, in Chapter 9, “The SCS INI Editor,” lists the default net attributes and their meanings.

Creating New Attributes

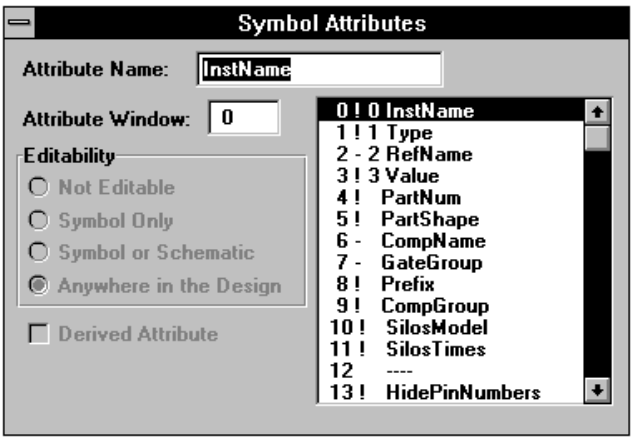
Attributes are defined using the INI Editor.

To create a new attribute:

1. Select **Options: Schematic** in the Synario Project Navigator, or **Setup: Edit Current INI** from the SCS Shell.
2. Choose the Attributes menu, then select the type of attribute you want to create: Symbol, Pin, Net, or Global. A dialog box is displayed. (The Symbol Attributes dialog box is shown below.)

   For Symbol, Pin, or Net attributes, a list box displays all the attribute numbers from 0 to 199. An edit box at the top permits entering the attribute’s name.

Figure 8-1
Symbol Attributes Edit Box



The Symbol and Pin Attribute dialog boxes have radio buttons to select the attribute modifier. The Symbol dialog box also has a second edit box to assign an optional attribute window number.

The Net Attribute dialog box has additional radio buttons to select how the attribute is displayed

For Global attributes, there is a list box with twenty lines numbered 0 to 19. There are two edit boxes, one for the attribute's name, the other for its value.

3. Attributes numbered 0 to 99 are reserved System attributes and should not be altered. Attributes numbered 100 to 199 that are not already assigned can be used to create new attributes. Click on one of these to enter the values for your new attribute. (Global attributes are numbered 0 to 19, and are always user-defined.)

4. Type in the attribute name and attribute window number you want. Click on the attribute modifier desired.

WARNING! *Some of the System attributes (0-99) may not be defined. Do not use these for your own definitions. Future symbol libraries (or libraries for devices that you do not currently use) may use these currently "blank" attributes.*

Table 8-1 below shows some typical attribute entries in the INI Editor.

Table 8-1
Typical Attributes

Attribute Number	Attribute Modifier	Attribute Window #	Attribute Name	Description
0	!	0	InstName	Instance Name
1	!	1	Type	Symbol Name
2	!	2	RefDes	Reference Designator for PCBs
3			Value	General value parameter
35		8	Width	MOS transistor width
36		9	Length	MOS transistor length
102	*	12	Lambda	Smallest processing geometry

The following sections explain how to enter values for each of the new attribute's components.

Attribute Names

Attribute names are text strings and can contain any characters except spaces. Names are not case-sensitive. You can mix cases to improve readability.

Attribute Modifiers

You can edit symbol and pin attribute values on the symbol and override the values in any schematic where the symbol appears. The four attribute modifiers described below control how attribute values can be changed in the schematic.

Modifier	Edit In	Description
blank	Anywhere in Design	These attributes can be assigned or edited in the Symbol Editor, Schematic Editor, or Hierarchy Navigator.
!	Not Editable	Certain attributes are editable only by special "system" commands, such as Instance Names and Net Name Flags. In addition, the ini file may contain attributes that are not editable for the purposes of maintaining compatibility with other versions of the schematic (for example, for a different FPGA or ASIC device family) without losing the attribute name association. These attributes are not listed in the attribute editors in the Symbol Editor, Schematic Editor, or Hierarchy Navigator.
-	Symbol Only	These attributes can only be assigned or modified in the symbol editor. They establish fixed values for all instances of the symbols to which they are attached. "Symbol Only" attributes will be listed in the Symbol Editor attribute editor, but not in the Schematic Editor or Hierarchy Navigator. This modifier cannot be assigned to net attributes.
$	Symbol or Schematic	Attributes designated with this modifier can be assigned or modified in the Symbol Editor or Schematic Editor; they are not editable in the Hierarchy Navigator. These are typically used in conjunction with netlisters that run from the schematic. Since those netlisters do not have access to the hierarchical database, any attributes added through the Hierarchy Navigator would be lost.
*	Derived	Derived attributes can be assigned or modified anywhere in the design through the Symbol Editor, Schematic Editor, or Hierarchy Navigator.

Note: *Attributes 00 through 99 are reserved for Synario definitions. Do not change their numbers or use. Attributes 100 through 199 are available for you to define and use for any purpose.*

For example, in Table 8-1, attribute 0, InstName, is predefined. Although you can change its name, you cannot change its attribute number or its use (storing instance names). The user-defined attribute Lambda has attribute number 102 and does not have the Symbol Only modifier; it can be altered as you like.

Assigning Values to Simple Attributes

Once an attribute has been created, you can assign values to it. Many attribute values are assigned during symbol creation.

An attribute such as Prefix (attribute #8) provides an element's SPICE prefix for use by the SPICE netlister. The value of this attribute is known when the symbol is created as C (capacitor), D (diode), I (current source), L (inductor), M (MOS transistor), Q (bipolar transistor), R (resistor), or V (voltage source). This value can be assigned in the Symbol Editor with the Symbol Attribute command from the Add menu.

To set the Prefix value for a MOS transistor symbol:

1. Select the Symbol Attribute command from the Add menu.
2. Click on Prefix in the list box.
3. Type the letter M.

Any time this MOS transistor is instantiated, the attribute Prefix has M as its default. As there would never be a reason to change the value of this parameter, you could use the INI Editor to add the minus sign (–) modifier ("Symbol Only") to the Prefix attribute. This modifier prevents accidental changes in the Schematic Editor or the Hierarchy Navigator.

You can also provide default values of width 5 and length 2 for the MOS transistor. Scroll through the Symbol Attributes list box, and edit values of the width and length attribute.

Changing Attribute Values in the Schematic Editor

When you create a schematic, you can change the width of the transistor symbol from within the Schematic Editor.

To change an attribute value:

1. Select the Attribute command from the Add menu. The Attribute Editor dialog box is displayed.
2. Click on an item (for example, an instance of the transistor symbol).
3. Select the desired attribute, and enter a new value (for example, under the Width attribute, you can change the value of 5 to 10).

Any value you change in the schematic overrides a symbol's default value. You can also edit attribute values in the Hierarchy Navigator. When a design is finished and logically correct, you can go back (with the Hierarchy Navigator) and adjust device sizes to fit the constraints on rise and fall time.

Removing Attributes from a Netlist

Most netlist programs ignore attributes whose first character is an asterisk (*****). This feature can be used to remove an attribute from a netlisted symbol by prefixing the attribute with an asterisk in the Schematic Editor.

Displaying Attribute Values on a Schematic

Attribute values are displayed in attribute windows. Before an attribute can be displayed on a schematic, an attribute window number must be assigned to the attribute in the Symbol Editor. In Table 8-1, the InstName attribute is displayed in attribute window 0, and the Width attribute is displayed in window 8.

You can add an attribute window number to any value to display it. The attribute window number does not have to match the attribute number.

You can assign one attribute window number to several attributes. The attribute with the lowest attribute number *that has a value assigned* is displayed.

Reassigning Attribute Windows

The attribute value assigned to an attribute window can be changed to permit viewing different attributes. Suppose you're editing a design within the Hierarchy Navigator and want to see the timing delays. You can use the Attribute Display command to temporarily assign the timing delay attribute to a window that normally displays a different attribute.

To reassign an attribute window:

1. Select the Attribute Display command from the Options menu of the Hierarchy Navigator. (The same command is also available in the Schematic Editor.) A list of all attributes and their associated windows is displayed.
2. Click on an attribute (for example, "PDQ") currently displayed on the symbol.
3. Remove the window number from this attribute by pressing BACKSPACE.
4. Add that window number to the timing delay attribute by clicking on that attribute and entering the window number.

The delay times you wanted to see are now displayed in the windows where the "PDQ" attribute was previously displayed.

To return to the original display, assign the attribute windows to their original numbers. Or, do nothing. Reassignments are temporary and are discarded when you exit the Hierarchy Navigator or Schematic Editor.

Title Block Attribute Windows

A special group of attribute windows, numbered 200–206, is reserved for design and file data you might want to display. They are pre-assigned to attribute windows with the same numbers.

The attribute windows in Table 8-2 can be attached to graphics symbols. When these symbols are placed in a drawing, the specified values are automatically displayed. You might, for example, add attribute windows for attributes 200, 202, and 201 to a Master symbol to display the schematic's name, the current sheet number, and the last time the drawing was updated.

Table 8-2
Graphic-Symbol Attribute Windows

Window Number	Window Name	Attribute Window Description
200	FileName	The name of the schematic. If the schematic has not been saved, the default name is UNTITLED.
201	Date	The date the file was last updated (written to disk).
202	Sh#	The current sheet number. Valid sheet numbers are 1 to 99.
203	Sheets	The number of sheets in this schematic.
204	Time	The time that the file was last updated (written to disk).
205	Design	The name of the navigator file (without the **.tre** extension).
206	InstName	The instance name of the current block.

You can define your own symbol attributes (in the range of 100–199) for such information as your company's name and address, or the name of the project engineer. SCS comes with a Master symbol for a Data I/O title block. You can modify it for your own use, or study it for ideas of how to create your own title block.

Using Graphic-Symbol Attribute Windows

To add an attribute window to a Graphic or Master symbol:

1. Select the Window command from the Add menu of the Symbol Editor. A list of available attribute windows is displayed in a list box.
2. Click on one of the attribute windows. The selected window is attached to the cursor.
3. Click at the desired point in the symbol to place the attribute window.

Number Notation in Attributes

Synario has a highly flexible system for handling numerical attributes (width, length, fan-in, impedance, and so forth). SPICE notational conventions are used. Numbers can be entered as integers (100, -5), floating-point (3.14159), scientific notation (1E5, 2.54E-3), or any of the preceding types followed by one of the following unit scale factors:

Table 8-3
Numerical Prefixes in Attributes

Unit Scale Factor	Prefix	Multiplier
T	tera	1e12
G	giga	1e9
MEG	mega	1e6
K	kilo	1e3
M	milli	1e-3
U	micro	1e-6
N	nano	1e-9
P	pico	1e-12
F	femto	1e-15

Alphabetic characters immediately following a number are ignored unless they represent a scale factor. 10, 10V, 10Volts, and 10HZ all represent the number 10, just as 1000, 1000.0 1000hz, 1E3 ,and 1.0E3 all represent 1000.

Each attribute has a default scale factor. Whenever an attribute appears without a scale factor, the default scale factor is used. If an attribute has a scale factor specified, the value is taken exactly as written (the default scale factor is ignored).

For example, the width attribute represents micrometers when it appears on a transistor symbol. The default scale factor for width is U (micro), so a width of 1000 on a transistor would represent 1000 micrometers (1 millimeter). However, a width of 1K on a transistor would mean 1 kilometer because the presence of the unit scale factor, K, overrides the default scale factor.

Table 8-4
Default Scale Factors for Transistor Size

Attribute	Value	Meaning
Width	1	1 micrometer
Width	1U	1 micrometer
Width	1000	1000 micrometers
Width	1K	1 kilometer

Derived Attributes

Derived attributes permit interactive designing. You can change a design and immediately view the effects of those changes, in much the same way you can modify the numbers in a spreadsheet to see what happens to your numerical model.

Derived attributes take their values from:

- Other attributes on the symbol or the symbol's pins.
- Attributes on any other symbol or pin instance.
- Attributes on the symbol's parents or children.
- Attribute tables.
- Global attributes accessible at any level of the design hierarchy.

A derived attribute is not a fixed value. It is specified by a format string that defines how the attribute is to be derived. The syntax of the format string is described below:

- Letters and numbers are transferred from the format string without interpretation, just as if they represented a simple attribute.
- Whenever a pound sign (**#**) is followed by a number, the attribute with that number is copied to the output string. Or, you can use a pound sign followed by the attribute name in square brackets. For example, either #35 or #[width] returns the value of the width attribute.
- If #*number* or #[*attr_name*] is immediately followed by a backslash (\ ) and a number representing a field index, the given field is copied from the attribute. Attribute fields are delimited by spaces, commas, slashes, colons, semicolons, or equal signs. This permits a single value to be taken from a list of values.

- Specific instances are accessed by preceding the instance name with the at sign (**@**). This allows attributes of child instances to be accessed. For example, @inst1#[width] takes the width symbol attribute from instance inst1.
- Attributes on parent instances are accessed using two periods (**..**) to indicate the next-highest level in the hierarchy. For example, @..#35 takes the Width attribute from the parent instance.
- Pin attributes are accessed by specifying an instance name followed by a dash (**–**) and the pin name. The pin name must be terminated with a space, equals sign (**=**), pound sign (**#**), or dollar sign (**$**). If no pin name is given, the reference is to the instance itself (@).

  For example, @inst2-in3#[LoadCap] returns the value of the load capacitance attribute from pin in3 on instance inst2.
- If a pin name in the preceding syntax is followed by an equals sign (@–= or –pinname=), the attributes of the *net* connected to the pin are used instead of the pin's attributes.
- A single period indicates the original instance. For example, @.-in3#42 takes the value of pin attribute 42 from pin in3 on the original instance.
- Global attributes are accessed by a dollar sign (**$**) followed by either the global attribute number or the attribute name in square brackets. For example, both $17 and $[lambda] access global attributes.
- Table data in schematics are accessed with the following syntax:

  `&`*`table_name`*`[row,column]`

  where

  table_name is the table's name. The cases must match.

  row can either be a number or =*value* or =#*attr* or =$*const*. When row contains an equals sign (**=**), the table is searched to find the row that has a label equal to *value* or equal to the value of attribute *attr* on the current symbol instance.

  Row numbers start at one. Row zero is the column label.

  column can either be a number or =*value* or =$*const* or =$*attr*. When a column contains an equals sign (=), the table is searched to find the column that has a label equal to *value* or equal to the value of the global constant *const*.

  Column numbers start at one. Column zero is the row label.

The *SCS Command Reference* explains the Table commands that are used to add tables and modify table data.

Examples of Derived Attributes

The following examples demonstrate the use of derived attributes. The attributes (103, 110, and 111) are first defined as shown in Table 8-5.

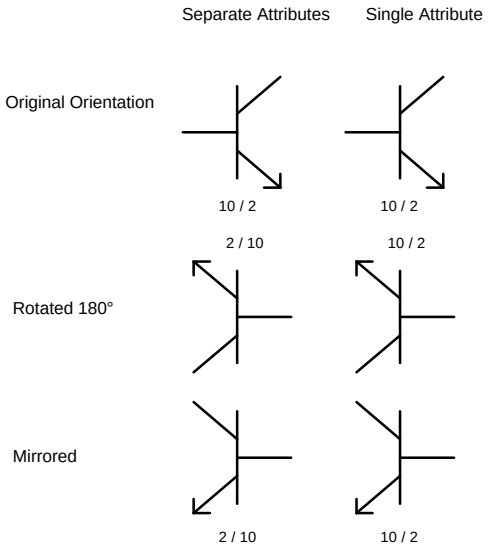
After loading the symbol for a MOS transistor into the Symbol Editor, you can create a new attribute that contains both the width and length. A slash (/) separates them from the letters W= and L=. No matter how the symbol is oriented in a schematic (for example, upside-down or rotated), the width is always to the left of the length when this new attribute is displayed.

Table 8-5
Derived Attributes Defined in INI File

Attribute Number	Attribute Modifier	Attribute Window	Attribute Name	Comments
0	!	0	InstName	Instance Name
1	!	2	Type	Symbol Name
3			Value	
35		8	Width	MOS transistor width
36		9	Length	MOS transistor length
103	*	10	W/L	Ratio of width to length
110	*	11	NewWidth	width derived from 103
111	*	12	NewLength	length derived from 103

Figure 8-2 shows what happens if you display the width and length parameters *separately* on a symbol that is rotated 180°—the order of the width and length is reversed. If you use a *single* attribute string to represent the width followed by the length, the display order is independent of orientation.

Figure 8-2
Effects of Mirroring and Rotation on Attributes Display



Assigning Derived Attributes

You can assign a derived attribute to any unused attribute number. In this case, assume user number 103 is free.

To add a derived attribute to the MOS transistor symbol from the Symbol Editor:

1. Select the Symbol Attribute command from the Add menu. The Attribute Editor dialog box is displayed.
2. Click on the W/L attribute. This is attribute number 103 from Table 8-5.
3. With this attribute in the edit field, type either

   `W=#35/L=#36`

   or

   `W=#[width]/L=#[length]`

   then press ENTER.

The W= is taken literally. It is followed by the value of attribute number 35, which in this case is the width. The /L= is interpreted literally and followed by the value of attribute 36, the length. If you then display attribute number 103 in a window (assuming the width is 10 and length is 2), the following appears:

```
W=10/L=2
```

You can rotate the MOS transistor symbol and not lose the proper order of width-followed-by-length. Also, the derived attributes can be used to set values of Schematic Editor-defined attributes in the attribute-number range 0 to 99.

If attribute 103 is defined as above, you can copy the width and length separately into two new derived attributes, 110 and 111. When you edit the symbol for the MOS device, use the Symbol Attribute command and edit the NewWidth and NewLength attributes to read:

`#103\2`	newly defined Width, attr 110, returns 10
`#[W/L]\4`	newly defined Length, attr 111, returns 2

This copies the second and fourth fields and places them into the NewWidth and NewLength attributes. The W and L are treated as ordinary fields. The equals sign (**=**) and slash (**/**) are delimiters.

If you define a derived attribute 104 and give it the following value:

`#103`	returns W=10/L=2

it returns the exact contents of attribute 103, W=10/L=2. The literal characters W=, L=, and the slash (**/**) *become part of the new attribute.*

Calculated Derived Attributes

Attributes are text strings and have numerical meaning only when interpreted by the Simulator or a netlister. There is one exception—derived attributes inside parentheses **( )** are treated as numbers, not text strings.

In derived attributes, expressions inside parentheses are evaluated to produce a number. The numerical result is limited to an accuracy of three decimal places. The four basic arithmetic operations are allowed, as well as several comparisons. Comparisons are evaluated as shown below.

(a = b)	1 if true, 0 if false
(a != b)	1 if true, 0 if false
(a > b)	1 if true, 0 if false
(a < b)	1 if true, 0 if false
(a >= b)	1 if true, 0 if false
(a <= b)	1 if true, 0 if false

Several examples of attribute definitions containing simple arithmetic calculations follow.

Format String	Result
`AREA=(#[length]*#[width])`	AREA=*nnn*, where *nnn* is length*width from this symbol
`A=(#35*#36* ((#35*#36) > 10)) + (#35*#36*1.2*((#35*#36) <= 10)))`	Sets A equal to length times width if the area is greater than 10. If the area is less than or equal to 10, then A is set equal to the area times the constant 1.2.
`(4* (5+3))`	32
`ABC(1.2/100+ .001)`	ABC0.013

Derived attributes can reference other derived attributes, but there is a limit of four levels of nesting. When a derived attribute references an instance that is not available, the entire attribute string for that derived attribute is left blank.

Derived Attributes and Hierarchy

The previous section showed how to access attributes or attribute fields on the current level of the hierarchy. When a design is loaded into the Hierarchy Navigator, all levels of the hierarchy are accessible, and attributes can also pass information to higher or lower levels. (You cannot pass attributes up and down in the Schematic Editor or Symbol Editor.)

To extract an attribute from another instance, use the at sign (**@**) followed by the instance specification. The instance specification for the parent (higher) symbol is two periods (**..**), and the instance specification for a child instance in the underlying schematic is the specific instance name.

Passing attributes up and down the hierarchy is useful during back-annotation. Transistor-sizing information extracted from a layout can be added to the transistor level in the Schematic Editor. If the attributes are set up properly, the low-level transistor sizes can be automatically transferred up through the hierarchy to the gate level or higher.

Example of Derived Attributes

This example shows how an IC design can be scaled for different processing dimensions.

One goal in IC design is to equalize the delays of rising and falling signals. This is usually done by sizing n-channel and p-channel devices so that the path from V_{dd} to the output has the same resistance as the path from the output to ground.

In a simple CMOS inverter, this results in the p-channel device to V_{dd} having about twice the width of the n-channel device to ground. In an inverter on a 2µ (micron) process, the n-channel device might have a width of 2µ and a length of 4µ. The p-channel device would then have a W/L of 8/2 for a balanced delay. On a 1 process, the corresponding values would be 2/1 for n channel and 4/1 for p channel.

Everything scales linearly with the process dimensions. The following example uses this fact to design a latch that can be automatically scaled to different process dimensions. Dimensionless W/L attributes are attached to transistors. At the primitive symbol level, these dimensionless quantities are multiplied by a process scale factor, Lambda, that assigns the actual dimensions.

A complete design can be scaled to different process dimensions by changing a single attribute, Lambda, inside the lowest-level primitives (the n- and p-channel transistors). The alternative to this approach is to regenerate the library every time a new process is used, and to include the new widths and lengths on all the symbols.

The following attribute definitions are used:

Table 8-6
Attribute Definitions for Derived-Attribute Example

Attribute Number	Attribute Modifier	Attribute Window	Attribute Name	Comments
35	*	8	Width	MOS transistor width
36	*	9	Length	MOS transistor length
102	–		Lambda	smallest processing geometry
105	*	13	Zn	W/L ratio for n channel
106	*	14	Zp	W/L ratio for p channel

Width and length are derived attributes that represent the actual width and length of MOS transistors. Lambda is the scale factor that, when multiplied by the dimensionless width and length terms, gives the actual transistor widths and lengths. Zn and Zp are dimensionless ratios of width to length, in the format W/L. Zn is for n-channel transistors, and Zp for p-channel.

Table 8-7 shows the different attributes and their values at various hierarchy levels. The bottom level is the MOS transistor, with symbols **pch.sym** and **pch.sym**. The width and length attributes are generated by multiplying the process scale factor, Lambda, by the dimensionless ratios Zn and Zp.

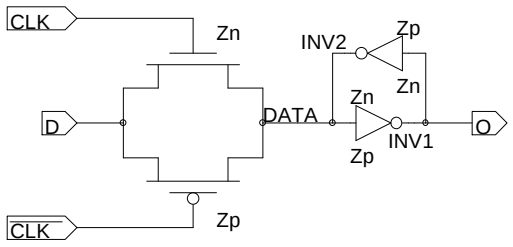
Lambda is defined at the transistor symbol level. The dash (**–**) modifier ensures that it cannot be overridden at higher levels. Zn and Zp are overridden at the latch schematic level and attached to each individual transistor or inverter instance in the schematic. Zn and Zp are then passed down to the transistor symbol level where they are used in the calculation of actual width and length of the transistors. The 10/2 and 20/2 in the **inv.sym** column are the default values. As you can see from the schematics below, all the instances have different values that are set at the **latch.sch** level in the hierarchy.

Table 8-7
Attributes and Values in the INI Editor for IC Example

	NCH.SYM	PCH.SYM	INV.SCH	INV.SYM	LATCH.SCH
Width	(#102*@..#105\1)	(#102*@..#106\1)			
Length	(#102*@..#105\2)	(#102*@..#106\2)			
Lambda	2	2			
Zn			@..#105	10/2	
Zp			@..#106	20/2	

Figure 8-3 shows a latch.

Figure 8-3
LATCH.SCH with Instances of MOS Transistors and Inverters



All transistors have different sizes. Attributes Zn and Zp are edited on n-channel and p-channel devices and passed down to MOS symbols. Attributes Zn and Zp are both edited on each inverter and the values passed down to the inverter schematic. Attributes Zn and Zp are dimensionless width/length ratios for n- and p-channel transistors.

Attributes Zn are Zp are passed down to individual transistors.

Attribute Zn is passed down from above. The dimensionless width is copied from Zn and multiplied by Lambda to make the attribute Width (35). The same is true for attribute Length (36).

These two device characteristics are scaled in proportion to process variable Lambda. This makes it easy to use the same schematic for different fabrication processes, such as 5, 1.2, or 0.5.

The SPICE netlist below shows that the dimensionless transistor widths and lengths are multiplied by the Lambda scale factor (1.2) to give physical dimensions in microns:

```
M1 Q DATA VDD VDD PMOS L=2.4U W=12U

M2 Q DATA GND GND NMOS L=2.4U W=6U

M3 DATA Q VDD VDD PMOS L=1.2U W=1.2U

M4 DATA Q GND GND NMOS L=2.4U W=1.2U

M5 DATA CLK’ D VDD PMOS L=2.4U W=24U

M6 DATA CLK D GND NMOS L=2.4U W=12U
```

Chapter 9
The SCS INI Editor

The Synario Capture System™ uses initialization files (**ecs.ini**) that control the default behavior of the Schematic Editor, Symbol Editor, Hierarchy Navigator, Waveform Viewer, and Waveform Editing Tool. This initialization (INI) file also contains the default values for symbol, pin, and net attributes.

In the Windows version, the master initialization file that comes with SCS is stored in the **config** subdirectory, directly beneath the main SCS directory (**c:\synario**, usually). You can also create "current" INI files that overlay the master in other directories.

In the UNIX/Motif version, the INI file information is built up from the master ecs.ini file stored in the data subdirectory and all other ecs.ini files found along the path from the root (/) to the current directory.

The INI Editor (**pcini.exe**) lets you view and alter the values of the parameters in the INI files. These parameters let you set the default values of the following (and other) functions and parameters:

- Attributes
- Color assignments
- Display parameters (such as Show Border and Show Pin Dots)
- Default symbol type for the Symbol Editor
- Default text editor
- Global net names
- IC or PCB development mode
- Library search paths
- Printer configuration
- Schematic sheet size and layout parameters
- Tools and Processes menu entries

The binary.ini File

SCS applications often have to refer to the contents of the INI file. INI files are ASCII text files, and reading and interpreting an ASCII file takes more time than directly loading the parameters' values in binary form.

Therefore, each time you save an INI file, the INI Editor automatically creates a binary version of the INI file (**binary.ini**) in the directory specified in your DOS TEMP environment variable. If you haven't defined a TEMP variable, the binary file is stored on your hard disk's root directory (usually **c:**). (In the UNIX/Motif version, the binary file is written to the **usr/tmp** directory.)

When the SCS Executive starts running, it looks for **binary.ini** in the TEMP directory (or, if there is no TEMP environment variable, in **c:**). If it cannot find **binary.ini**, it automatically recreates the file with the default values for SCS.

Custom INI Files

SCS is supplied with a default or "master" initialization file called **ecs.ini**. It is stored in the **config** subdirectory. Unless you create other initialization files and make them the default INI file, this is the file that is always used.

You can use the Save As command from the File menu to save the INI file under any other name (and in any other directory, if you wish), to create customized INI files ("project" files). For example, you might want a separate INI file for each device family you use.

When you create a custom INI file and make it the current active initialization file, SCS stores its fully qualified path in the INIFILE entry of the [SCS] section of **win.ini**.

To use a custom file, run the INI Editor and load the file. Then select the Save command from the File command. A dialog box asks if you want to "Set as Current Project?" this INI file. Click on Yes to use this INI file, No to keep using the file currently loaded.

Note: *The custom INI files use the master INI file (***ecs.ini***) as an "overlay." A custom INI file contains only those values and settings that are different from the settings in the master INI file. Any changes to the master file will change the behavior of the custom files. This feature lets you make "global" changes to the custom files without having to edit each one separately.*

Master and Project Mode

The INI Editor has two operating modes—Master mode and Project mode. When you load **ecs.ini** to edit it, the Editor is in Master mode. When you load any other SCS initialization file, the Editor switches to Project mode.

The two modes have slightly different behavior, to accommodate the different requirements when editing the master initialization file or a custom file.

In Master mode:

- The File: New and File: Open commands are disabled.
- The File: Save command saves the master INI file (**ecs.ini**).
- The File: Save As command saves the master INI file with a different name.

In Project mode:

- The File: New command creates an empty project file.
- The File: Open command opens an existing project file.
- The File: Save command saves this project file.
- The File: Save As command saves this project file with a new name.

When you save a project file under a name that is not the current project name, the INI Editor asks if you want to make the modified file the "current project." Click on Yes to make the modified file the new default INI file. Click on No to save the changes, but keep the currently loaded (unmodified) INI files as the default.

If you decide to discard your changes, use the Exit command and select No ("Don't Save Changes") to quit without saving.

Special Treatment of Master File Values

When you are editing a project INI file, the INI Editor displays some of the values that appear in the master INI file (**ecs.ini**).

- If an attribute is defined in the master file, but not in the project file, the master-file value is shown. These values *cannot* be deleted. You should not, therefore, define an attribute in the master file unless you want it to appear in all the project files. Project-file attributes can be defined only with unused attribute numbers.
- The Sheet Layout dialog box includes an edit box labeled "Automatically Add Master Symbols." If one or more symbols have been specified in the master file, their names will appear to the left of the edit box. These names cannot be deleted. As with attributes, do not include Master Symbols in the master file if you do not want them to appear in project files.
- The Global Signals dialog box displays any names assigned to global nets in the master INI file above the edit box for that net. This lets you see the default definitions before you change them.
- The Sheet Sizes, Symbol Tools, Schematic Tools, Navigator Tools, and Processes dialog boxes display the master-file values for these commands in a separate list box. These values cannot be deleted or edited. They are displayed for reference and do not become part of the project file.

- The Search Paths dialog boxes (Project, Symbol, and Model Libraries) display the master-file values for these paths in a separate list box. These values cannot be deleted or edited. They are displayed for reference and do not become part of the project file.

pcini.exe Command Line Options

The following command line options can be used with **pcini.exe**. If you want to use any of these all the time, add them to the command line under [&Setup] in **pcshell.ini**. (See Chapter 4, "Basic Operation," for more information about **pcshell.ini**.)

Option	Use
-current	The editor runs in Project mode on the current INI file. If **ecs.ini** is the current file, it is loaded. (This is the default option in **pcshell.ini**.)
-master	The editor runs in Master mode on **ecs.ini**.

INI Editor Menus

In addition to the File menu, the INI Editor's menu bar displays the following:

- Controls
- Tools
- Attributes
- Search Paths

The settings and controls for each menu are explained in the following sections.

Controls Menu

System Controls

The System Controls affect the general behavior of the Symbol and Schematic Editors and the Hierarchy Navigator.

Application Mode

This parameter configures SCS for IC design, or PCB design, or both. The default setting is IC.

IC Only	Provides instance-name and pin-name support for ASICs.

PCB Only	Provides support for pin numbers and reference designators for board packaging. See Chapter 10, “PCB Design Considerations,” for information about the differences between IC and PCB design using SCS.
Both IC & PCB	Provides support for instance names, pin numbers, and reference designators.

Bus Parentheses

The Bus Parentheses list box lets you select the delimiting character for indices on ordered bits. They can be enclosed in brackets, parentheses, curly braces, or angle brackets, as shown below. The default is square brackets.

`A[2], A[1], A[0]` (default)

`A(2), A(1), A(0)`

`A{2}, A{1}, A{0}`

`A<2>, A<1>, A<0>`

First Character Must be Alphabetic

The first character of a net name or instance is usually a letter. (Numbers are most often suffixes that identify a specific instance or net.) If this check box is unmarked, the first character can be a number. This default setting is for this box to be checked (first character must be alpha).

Coerce Net Names to Upper Case

When this check box is marked, any net name you enter is converted to all upper-case letters. If you are using a netlist program or simulator that expects net names to be all upper-case, you may want to check this box to be sure the names are in the correct format.

Coerce Attributes to Upper Case

When this check box is marked, any attribute name you enter will be converted to all uppercase letters. Attribute names are not case-sensitive, however.

Prefer Descending Buses

By default, when the Waveform and HDL tools encounter bus elements where the ordering of the bus elements is unclear, they arrange them in ascending order. When this check box is marked, these tools will instead arrange them in descending order. This option is ignored if you specify a bus order explicitly anywhere in the schematic.

Simulator

The name in this edit box (*simulator*) is the base name of the simulator configuration file. The Hierarchy Navigator searches for a file named *simulator*.**ini** to establish the simulation parameters. The default setting is "SimCP". See Appendix B, “Simulator Interfaces,” for more information about the *simulator*.**ini** file.

A major feature of the simulation environment is providing hardware description language (HDL) templates in the Symbol Editor and Hierarchy Navigator. This facilitates writing behavioral models for the simulators.

Text Editor

Specifies the editor SCS uses to edit text. The default is **synview.exe**. If you want to use a different editor, enter its name here. If the alternate editor’s directory does not appear in the DOS PATH statement or is not in the SCS base directory, give the fully qualified pathname.

Text Viewer

Specifies the editor SCS uses for viewing text. The default is the Notepad (**notepad.exe**). If you want to use a different editor, enter its name here. If the alternate editor’s directory does not appear in the DOS PATH statement or is not in the SCS base directory, give the fully qualified pathname.

Display Controls

The Display Controls dialog box is a group of check boxes for the following display features. Checking a box displays or enables the corresponding feature. The first eight items can be overridden (for the current session only) with the Display Options command from the Options menu in the Schematic or Symbol Editors. (Use the INI Editor to make any changes permanent.)

Show Border

Turns screen and printer display of the schematic and symbol borders on and off.

Show Pin Dots

Turns the screen and printer display of pin dots (unconnected pins) on and off.

Show Pin Numbers

Turns the screen and printer display of pin numbers on and off. (In the Symbol Editor, pin numbers are displayed only when editing their values.)

Show Symbol Text

Turns the screen and printer display of fixed text inside symbols on and off.

Show Symbol Attributes

Turns the screen and printer display of symbol attributes on and off.

Show Net Attributes

Turns the screen and printer display of net attributes on and off.

Show Solder Dots

Turns the screen and printer display of solder dots (wire connections) on and off.

Show Off Page Connects

On multiple-sheet schematics, you can show references to other sheets at nets that connect across more than one sheet. The display is enabled on wire segments with their names at the end of the wire.

Show Open Ends

Wires not terminating on a net name flag, symbol pin, or another wire are considered schematic errors. This parameter controls whether they are highlighted on the screen and plotter.

Allow Rotated Pin Numbers

The numbers on the pins of a symbol are normally displayed as horizontal text. Optionally, the top and bottom pin numbers can be displayed as vertical text.

Allow Rotated Net Names

If this box is checked, net names at the ends of vertical wires will (in some cases) be displayed vertically.

Show Net Numbers

This parameter turns the screen and plotter display of node numbers on and off. It affects only the Hierarchy Navigator display.

Every node in the circuit has a node number assigned in the SCS database. These node numbers are used internally and can also be used by simulators like SPICE, which require numbers rather than names.

Show Simulation Values

When a simulator is run with the Hierarchy Navigator, the logic levels of each node are displayed on the schematic in the Navigator. When this box is checked, the values are shown on the screen and when the schematic is printed.

Symbol Controls

These control two defaults in the Symbol Editor, Default Symbol Type and Default Pin Name Offset. (In the UNIX/Motif version of SCS, they appear in the System Controls dialog box.)

Default Symbol Type

The Default Symbol Type is assigned to a newly created symbol. You can choose from Block, Cell, Component, Gate, Graphic, Pin, or No Default. These types are described in the list below. You can override the default type by selecting a different type using the Symbol Editor's Change Type command.

Refer to Chapter 6, "Using the Symbol Editor," for a description of the different symbol types.

Block	Hierarchical elements (such as modules)
Cell	Primitive in IC design
Component	Primitive in PCB designs representing complete packaged device
Gate	Primitive in PCB designs representing a single functional element of complete device (for example, one of four NAND gates in a 7400).
Graphic	Non-electrical information, such as tables and notes.
Pin	Physical pins on an edge connector or PCB.
No Default	The Symbol Editor prompts you for the symbol type when you start a new drawing.

The Master type is not available as a default, only by No Default prompting or from the Change Type command. It is assumed Master is the least used symbol type and you would only occasionally create Master symbols.

Default Pin Name Offset

Default Pin Name Offset controls the distance between a pin's name label and the pin itself. It is measured in units of one-quarter the Secondary grid. The default is 36 units. The value can range between 0 and 127.

Graphic Options

This dialog box sets the defaults for the Graphic Options dialog box in the Schematic and Symbol Editors. Any of these can be overridden in the Editor for the current working session.

Text Justification Horizontal text can be left-justified, right-justified, or centered. This parameter applies both to fixed graphic text and text in attribute windows. Any change to this setting affects text added after the change, not existing text.

Text Size You can choose the size of fixed graphic text and text in symbol attribute windows. In the Windows version, the choices are First, Second, Third, Fourth, Fifth, Sixth, Seventh, and Eight. In the UNIX / Motif version, the choices are Small, Medium, and Large. The defaults are Third and Small, respectively. Any change to this setting affects text added after the change, not existing text.

Vertical Text Text is normally drawn horizontally. When the Vertical Text box is checked, the default placement of text is vertical.

Grid Spacing Grid Spacing controls the placement of graphic objects, *not* symbols or wires (which must always fall on the Primary grid). You can set the default increment to the Primary grid spacing, or one-half or one-quarter that value. Smaller values allow more precise placement.

Show Grid Enables the Primary grid display. The grid appears as an array of dots, with one dot at each grid intersection. Every tenth grid point is larger. As you “zoom out” and the grid dots get closer together, some dots may not be displayed.

Full Cursor Selects between the normal cursor (a small “plus” sign) and the full-screen cursor. The full-screen cursor makes it easier to align objects.

Wide Lines When Wide Lines is checked, all graphic elements (*not* symbols or wires) are drawn with double-thick lines. These heavy lines have the same weight as schematic buses. Any change in this setting affects graphics drawn after the change, not existing graphics.

Sheet Layout

The Sheet Layout dialog box sets defaults for the border and the Primary grid.

Zones

The border is divided into horizontal and vertical zones (sections) to simplify locating a specific item. For example, a flip-flop might be in the B7 zone, or an I/O marker in D3.

By default, the horizontal zones are numbered, the vertical lettered. When the "Draw Numbers on Vertical Axis" box is checked, the horizontal zones are lettered, the vertical numbered. When this box is clear, the horizontal zones are numbered, the vertical lettered.

When the "Horizontal Zones Increase toward the Right" and "Vertical Zones Increase toward the Top" boxes are checked, the lowest numbers (or letters) are at the left and bottom. When these boxes are cleared, the lowest numbers (or letters) are at the top and right.

The "Number of Horizontal Zones in Schematic Border" and "Number of Vertical Zones in Schematic Border" edit boxes set the number of border divisions. The permitted range of values is two to nine divisions.

Grid

The Grid Size and Grid Units settings are self explanatory. Inches or centimeters at a 0.1 increment are the most common choices. 0.1 inches is the default.

Symbols do not have an absolute size; they are scaled in grid units. Therefore, selecting a smaller grid may let you place more symbols on a given size schematic. On the other hand, a larger grid produces larger symbols when the schematic is printed.

Automatically Add Master Symbols

Master symbols are used for reference items that appear on every schematic, such as a title bar, the project name, or the company logo. If you want a Master symbol to be added to each schematic, type its file name in the Automatically Add Master Symbols edit box.

The symbol is automatically placed in the same corner of the schematic as its origin. For example, if the symbol's origin is at its lower-left corner, the symbol is positioned at the lower-left corner of the schematic.

More than one Master symbol can be specified by separating them with spaces. (If they have the same origin, they will overlap.) As with other symbols, *do not* specify the path. The Editor will traverse the Symbol Libraries search path for the first symbol file with a matching name.

Sheet Sizes

The Sheet Sizes dialog box sets the permitted drawing sizes. The default sheet size is the size specified at the top of the list box. The sheet size for a particular drawing can be changed with the Sheet Setup command in the Schematic Editor. Typical sheet sizes are:

English (inches)		**Metric** (mm)	
A = 11	8.5	A4 = 297	210
B = 17	11	A2 = 594	420
C = 22	17	A3 = 420	297
D = 34	22	A1 = 841	594
E = 44	34	A0 = 1189	841

The width is the first number. Since schematics are usually wider than they are high, the width is usually greater than the height. Height and width are measured in the Grid Units specified in the Sheet Layout dialog box (inches, centimeters, or millimeters). The maximum supported dimension is 8000 Grid Units.

To add a new size, click on the Add button. A new entry with the designation New and a length and width of zero is added to the list. Press TAB to select the edit boxes (or click on them), then change the Sheet Size, Width, and Height to the values you want.

To change the size of an existing sheet, click on the list box line with its description. The values are copied to the edit boxes, where you can alter them.

The Delete button removes the currently highlighted sheet. The Move Up button swaps the highlighted sheet with the sheet above it. The Move Down button swaps the highlighted sheet with the sheet below it.

Note: *The sheet size names are arbitrary and need not have any relation to either European paper sizes or American drafting paper sizes. You can use any letter, number, or name you want.*

Wave Controls

This dialog box controls the overall appearance of the Waveform Viewer and Waveform Editing Tool. Colors are set in the Wave Colors dialog box.

Padding Around Text	The line spacing of text in the waveform name area in units of one-quarter the text height.
Gap Between Waveforms	The space between waveforms in units of one-quarter the text height.
Characters in Name Field	The width of the waveform name area. If names are longer than the number of characters selected, the names are truncated in the display.
Reverse Bits in Bus	If Yes, the least-significant bit (LSB) of buses is displayed to the left whenever the bus value is displayed as a number. If No, the bus value is displayed with the same bit order as a binary number (the LSB to the right).
Bus Radix	The radix, or base, used to display buses. The choices are Binary, Octal, Decimal, and Hexadecimal.

Global Nets

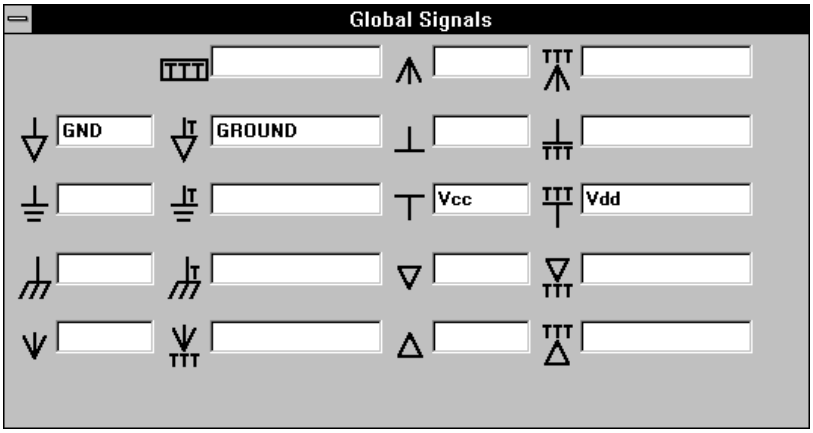
Global nets are net names that have predefined symbols associated with them. When one of these global names is assigned to a net (GND, for example), the corresponding symbol is attached to the net in your schematic.

Global signals can be accessed across all hierarchy levels and across all sheets and schematics in a design. For this reason, names assigned as global net names cannot be used as “local” net names.

Note: *Global ground symbols are drawn only at the bottom of vertical wires, while global supply symbols are drawn only at the top of vertical wires. If the wire is not vertical, the global symbol is not drawn. Instead, its name is displayed inside a box that overlaps the net.*

The available symbols are shown in Figure 9-1. A symbol cannot be used until a name has been assigned to it. (Three of the symbols are already named and can be used immediately.) Click on the edit box next to the symbol and type in the name you want. You can change or remove a name the same way.

Figure 9-1
Global Signals Dialog Box



There are three types of global net symbols.

Labeled Symbols The symbols in columns 2 and 4 of Figure 9-1 are *labeled* symbols. The name you assign to one of these symbols is attached to the symbol to label it, replacing the "T" or "TTT".

You can assign more than one name to labeled symbols. Multiple names are separated with a space (not a comma). Therefore, names cannot contain spaces.

Unlabeled Symbols The symbols in columns 1 and 3 of Figure 9-1 are *unlabeled* symbols. The name you assign is *not* shown on the symbol. The symbol is the only visible indication that the net has been named. Use the Query command to view the name.

You can assign only one name to an unlabeled symbol. This limitation prevents confusion; if you could assign more than one name, you would have no way of knowing which name had been chosen.

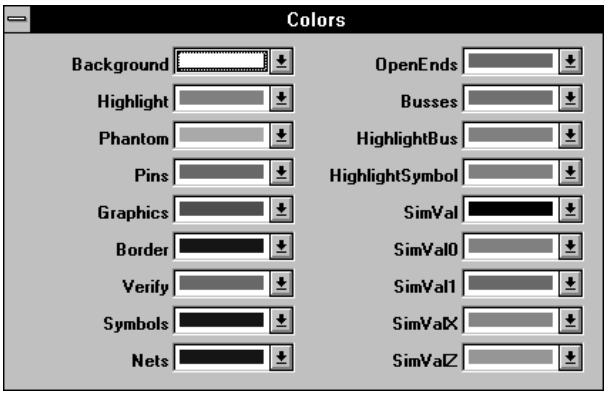
No Symbol You can assign a name to the box with "TTT" at the top of the second column. This name is global. The name is attached to the net, but there is no symbol (other than the box surrounding the name).

You can assign more than one name to this symbol. Multiple names are separated with a space (not a comma). Therefore, names cannot contain spaces.

Colors

The Colors dialog box assigns colors to various Symbol and Schematic Editor display functions. There are 18 combo boxes, one for each Editor display function that can have a unique color.

Figure 9-2
Colors Dialog Box



To change a color, click on the arrow in the combo box, then click on the desired color in the color-bar display that appears. The display functions are listed below.

Background The background color of the Editor window

Highlight When you use the Query command to select a component, the component is marked with diagonal lines of this color. Queried nets are redrawn in this color.

Phantom When you select objects to move or drag, they are redrawn in this color.

Pins The color of symbol pins.

Graphics The color of lines, boxes, circles, arcs, and text. (Although the body of a symbol is graphical, it has its own color attribute.)

Border The color of the lettered and numbered boxes that border the drawing.

Verify	A net selected with the Query command is redrawn in this color. A symbol selected with the Query command is marked with diagonal lines of this color.
Symbols	The color of the body of a symbol.
Nets	The color of wires and net names.
OpenEnds	Hanging wires or unattached net names are marked with a dot of this color.
Buses	The color of buses and bus names.
HighlightBus	A bus selected with the Mark command is redrawn in this color.
HighlightSymbol	A symbol selected with the Mark command is marked with diagonal lines of this color.
SimVal	The color of text showing the logic value of a schematic node.
SimVal0	The color of the small square on a schematic node indicating logic low.
SimVal1	The color of the small square on a schematic node indicating logic high.
SimValX	The color of the small square on a schematic node indicating unknown state.
SimValZ	The color of the small square on a schematic node indicating high impedance.

CAUTION: Don't assign the Background color to any other display function. That function will be invisible against the same-colored background.

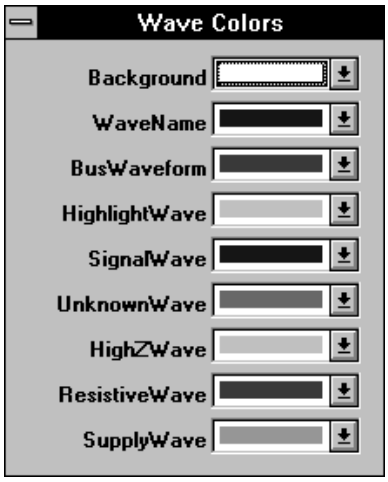
Wave Colors

The Wave Colors dialog box assigns colors to various Waveform Viewer display functions. There are nine combo boxes, one for each Waveform Viewer display function that can have a unique color.

To change a color, click on the arrow in the combo box, then click on the desired color in the color-bar display that appears. The display functions are listed below.

Background	The background color of the Waveform Viewer window.
WaveName	The color of waveform names in the names window.

Figure 9-3
Wave Colors Dialog Box



BusWaveform	The color of bus waveforms.
HighlightWave	When a waveform is selected, its name is marked with diagonal lines of this color.
SignalWave	The color of single-bit waveforms.
UnknownWave	The color of signals that have been assigned a logic value of “unknown.”
HighZWave	The color of signals that have been assigned a logic value of “Hi-Z.”
ResistiveWave	The color of signals that have been assigned a strength of “resistive.”
SupplyWave	The color of signals that have been assigned a strength of “supply” (Vdd).

CAUTION: Don’t assign the Background color to any other display function. That function will be invisible against the same colored background.

Print Controls

Orientation	Specifies the default orientation for printing schematics.
Margins	Specifies the default margins for printing schematics
Clipboard Format	Specifies the clipboard format of items that are cut or copied to the clipboard.

Tools Menu

You can invoke other programs from within the Schematic Editor, the Symbol Editor, or the Hierarchy Navigator. These new programs are listed in the Tools menu of the Editors and the Navigator, and in the Navigator's Processes menu. (If no programs are assigned to these menus, the menus are not displayed.)

There are three dialog boxes for adding programs to the Tools menus, and a fourth dialog box for adding programs to the Process menu in the Hierarchy Navigator. The only distinction between Tools and Processes in the Hierarchy Navigator is that the Processes menu is intended for programs that create netlists. You can ignore this distinction and assign programs to any menu.

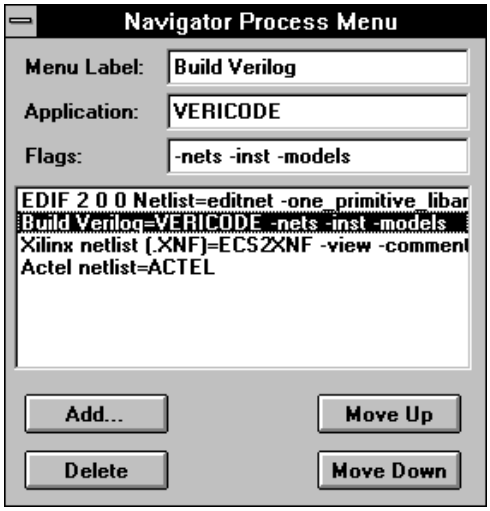
All four dialog boxes are identical and work the same way. The Hierarchy Navigator Process Menu dialog box is shown in Figure 9-4.

- The Menu Label is the text string that appears in the menu to identify the added tool or process. You click on this menu item to run the program.
- The Application is the filename of the program. (You can also specify DOS batch (.**bat**) files.) If the program is not in the SCS root directory or one of the directories in the DOS PATH statement, enter the fully qualified path name.
- The Flags are any command line switches or options needed.

You can manually fill in these edit boxes, or you can use the Add button to browse the programs on your hard disk. The Add button displays the generic Open File dialog box, labeled Choose Application. Find the application you wish to add, then double-click on it. (Or highlight it and click OK.) The Menu Label and Application edit boxes are automatically filled in.

When you use the Add button, the path is not included in the Application edit box. If the program is not in the SCS root directory or one of the directories in the DOS PATH statement, you must add the full path to the application's name.

Figure 9-4
Hierarchy Navigator Process Menu Dialog Box



You must also fill in any command line flags or switches required. You can use any of the filename substitution flags in the command line flags to represent the base name of the currently loaded file. See the section "pcshell.ini Format" in Chapter 4 for filename substitution flag. (You must supply the desired extension.) For example, if the file **design.rat** were currently loaded, and you wanted to pass the name **design.cat**, you would enter

```
&F.cat
```

If you don't use a substitution flag, the current filename is inserted at the end of the command line.

The list box below the edit box displays the tools or processes in the same order they will appear in the menu. The Move Up button swaps the highlighted application with the tool above it. Move Down swaps the highlighted application with the application below it. To delete an application, highlight its name, then click the Delete button.

Symbol Tools

Any utilities, netlisters, or other processes that don't require the Hierarchy Navigator to operate (such as the Notepad, archiving tools, and so on) can be added to the Tools menu of the Symbol Editor. The Verilog and VHDL netlisters are the only tools shipped with SCS that currently work in the Symbol Tools menu.

Schematic Tools

Any utilities, netlisters, or other processes that don't require the Navigator to operate (such as the Notepad, archiving tools, and so on) can be added to the Tools menu of the Schematic Editor. The following tools shipped with SCS work in the Schematic Tools menu.

- EDIFNETS
- Schematic Back annotation
- Verilog netlister
- VHDL netlister

The following are the standard Schematic Tools entries in the default **ecs.ini** file:

```
Code Verilog=vericode
Code VHDL=vhdl
EDIF Netlist=EDIFNETS
```

Navigator Tools

This dialog box adds entries to the Tools menu of the Hierarchy Navigator. These entries start tasks for the Waveform Viewer. You can write your own interface programs and add them to the Tools menu.

The following are the standard Navigator Tools entries in the default **ecs.ini** file:

```
Check Circuit=CHECKCKT

Code Verilog Model=VERICODE
```

Navigator Processes

This dialog box adds entries to the Processes menu of the Hierarchy Navigator. These entries start tasks for netlisting or simulation. SCS supports several netlisters that are accessed through this menu. You can also write your own interface programs and call them from the Processes menu.

Note: *The distinction between the types of programs added to the Navigator's Tools and Processes is arbitrary, and designed only to make it easier to find a specific tool. You can place your own entries in either menu.*

These are the standard Navigator Process entries in the default **ecs.ini** file:

```
Flat Spice Netlist=Spicenet
Hierarchical Spice Netlist=hSpicent
Net List By Pin=pcbnet -pin     (generic list in part / pin order)
```

```
Net List By Net=pcbnet -net     (generic list in net order)
List Marked Nets and Instances=lister -listmark
Partlist=lister -listpart
Instance list=lister -listnet
EDIF 2 0 0 Netlist=edifnet      (EDIF-format netlister (hierarchical))
ASCII Netlist=asciinet          (ASCII-format netlister)
```

Note: *An optional Programmer's Interface Kit (PIK) is available. This kit provides C functions that let you access the SCS database (schematics, symbols, navigator files, and so on). You can use the extracted data as input for your own interface programs.*

Attributes Menu

Attributes are created and assigned using the INI Editor. Refer to Chapter 8, "Attributes" for a description of attributes.

Symbol, Pin, and Net Attributes

There are separate dialog boxes for creating symbol, pin, and net attributes. All have a list box showing the current attribute definitions, and an edit box at the top where an attribute name can be added, deleted, or altered.

Attribute Numbers

The attribute is represented in the attribute database by the attribute number, and SCS uses this number to access it. You can therefore change the name of *any* attribute without changing access to that attribute.

Unused attribute numbers are shown with four dashes in the name field. New attributes can be added to these empty fields.

The first 100 attributes (0–99) are reserved for SCS use. You should not add new attributes to the empty attribute fields in this section. Future versions of SCS or symbol libraries you purchase later may need these currently unused attributes.

The second 100 attributes (100–199) are for your own use. You can define them in any way you like, without worrying about changes or incompatibility.

There are seven attributes (200–206; not shown in the INI Editor) that can be used to display file creation and other data. These are explained in Chapter 8, "Attributes."

Figure 9-5
Symbol Attributes Dialog Box

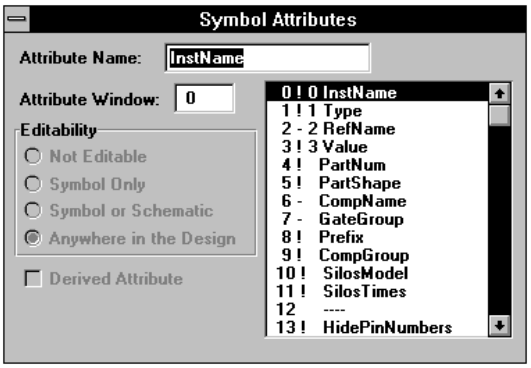
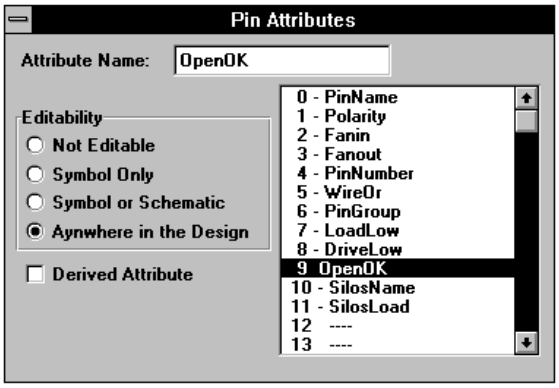


Figure 9-6
Pin Attributes Dialog Box



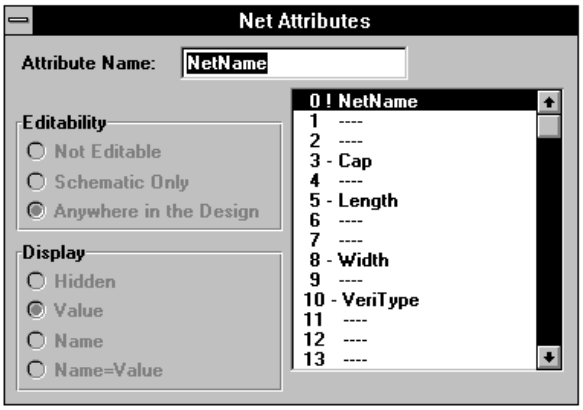
Adding Attributes

To add an attribute:

1. Click on the number (100–199) of an unused attribute.
2. Enter the name of the attribute in the Attribute Name edit box.
3. Select appropriate values for attribute modifiers, window and display.

Attribute names are not case-sensitive. You can mix upper and lower case to make the name easier to read.

Figure 9-7
Net Attributes Dialog Box



Attribute Data Fields

The *attribute modifier* controls where and how attribute values can be entered or altered. The default is a blank (no entry). All the dialog boxes include a group of Edibility radio buttons that select the attribute modifier. See Chapter 8, "Attributes" for more information on attribute modifers.

You can specify how net attributes are displayed. See "Add: Net Attribute Window" in the *SCS Command Reference* for more information.

Attribute Window

Symbol attributes can be displayed on or near the symbol in an area called the “attribute window.” (Attribute windows are described in Chapter 6, “Using the Symbol Editor.”)

If you select an attribute and enter a number in the Attribute Window edit box, the value of the attribute is displayed with the symbol. The attribute window number does not have to be the same as the attribute number.

Attribute Name

You can enter your own names for the user-defined attributes (100–199). You can change the names of the system attributes (0–99) if you want, because the association between an attribute and its value is made with the attribute’s number, not its name.

Modifying Attributes

Use the following procedure to add or modify a symbol attribute. The procedure is nearly the same for pin and net attributes; skip the steps that don’t apply to the selected attribute.

1. Click on the attribute you want to modify. You can select any line in the list box, even if the attribute number is the only field that currently has a value.
2. The attribute's name appears in the Attribute Name edit box. Type the new or changed attribute name.
3. Click on the appropriate radio button to assign the desired attribute modifier. "Anywhere in Design" is the default modifier. If this modifier is selected, the modifier field is left blank in the list box. (Net attributes do not have an attribute modifier.)
4. If you want the attribute displayed, enter a number in the Attribute Window edit box. Window numbers range from 0 to 99, and have no relation to the attribute numbers 0 through 99. (Pin and net attributes do not assign attribute windows in the INI Editor.)

   If two (or more) attributes use the same attribute window, the lowest-numbered attribute *that has a value* is displayed.

Example Attributes

Tables 9-1, 9-2, and 9-3 show example attribute definitions for net, pin, and symbol attributes. Attribute modifiers, numbers, and windows are shown where appropriate.

Chapter 8, "Attributes," has a more detailed discussion about using attributes and creating new ones.

Table 9-1
Standard Net Attributes

Att Number	Att Mod	Attribute Name	Description
0	!	NetName	Net name
3		Cap	Capacitance
5		Length	
8		Width	
10		VeriType	Verilog Net Type
30		VHDLNetType	VHDL Net Type
31		VHDLBusType	VHDL Bus Type
111		RouteLayers	Route Layers
112		ThermalLayers	Thermal Layers
113		NetWeight	Net Weight or Priority
114		ViaPerNet	Number of Vias allowed per Net
115		MinWidth	Minimum Net Width for PCB
116		MaxWidth	Maximum Net Width for PCB
117		MinLength	Minimum Net Length for PCB
118		MaxLength	Maximum Net Length for PCB
119		WidthByLayer	Width List by Layer
120		SpacingByLayer	Spacing List by Layer
121		ConnWidth	Connect Width
122		ReconnType	Reconn Type
123		MatchedPair	Matched Pair
124		Shielding	Shielding
125-130		Reserved	Future PCB

Table 9-2
Standard Pin Attributes

Att Number	Att Mod	Attribute Name	Description
0	–	PinName	Pin name
1	–	Polarity	In, Out, BiDir
2		FanIn	Dimensionless number for IC loads
3		FanOut	Dimensionless number for IC drive
4	+	PinNumber	Used in PCBs for Gate or Component pin numbers; represents physical pin connection
5		WireOr	Tristate, Opencollector, or Yes
6	–	PinGroup	Used in PCB design; indicates can swap pins
7		LoadLow	Current load in low state
8		DriveLow	Current drive in low state
9		OpenOK	OK to be unconnected pin
10	–	SilosName	Identifies pins in models
11		SilosLoad	Numeric load factor for load calculation
14	–	VeriName	Alternate pin name or order for pins
15		LoadHigh	Current load in high state
16		DriveHigh	Current drive in high state
18	–	TimilName	Alternate pin name or order for pins
20		LoadCap	Capacitive load (pF)
21		DriveCap	Capacitive drive (pF)
22	+	KCL	Drive factor for simulation models
23	+	Pin2Pin	Pin to pin delay for simulation models
24	+	DelayBack	Back annotated delay for simulation models
25	+	ChkPulseW	Check for pulse width violation on this pin
26	+	ChkHold	Check for hold time violation on this pin
27	+	ChkSetup	Check for setup time violation on this pin
30		VHDLPinType	Port type if scalar port in VHDL

Att Number	Att Mod	Attribute Name	Description
31		VHDLBusPinType	Port type if vector port in VHDL
32		VHDLDefValue	Default value for Port
33		VHDLNetConv	Type conversion function for port
34		VHDLBusConv	Type conversion function for port
35		VHDLPinUse	Used for port type of BUFFER in VHDL
36		SpiceOrder	Integer that forces order of subcircuit pins
40		HiLoPinName	Alternate pin name used for HiLo simulations
50		XSimPinName	Pin name used for X-Sim primitives
51		PinNegation	Apply negation to this pin
90	–	BusPin_A	First set of pins attached to bus pin
91	–	BusPin_B	Second set of pins attached to bus pin
92	–	BusPin_C	Third set of pins attached to bus pin
93	–	BusPin_D	Fourth set of pins attached to bus pin
94	–	BusPin_E	Fifth set of pins attached to bus pin
95	–	BusPin_F	Sixth set of pins attached to bus pin
96	–	BusPin_G	Seventh set of pins attached to bus pin
97	–	BusPin_H	Eighth set of pins attached to bus pin
111		PinDiameter	Pin Diameter
112		ECLType	ECL Pin Type
113-120		Reserved	Future PCB

Table 9-3
Standard Symbol Attributes

Att Number	Att Mod	Attribute Window	Attribute Name	Description
0	!	0	InstName	Instance Name
1	!	1	Type	Symbol Name
2	!	2	RefName	Reference designator for PCBs
3		3	Value	General value parameter
4			PartNum	PCB part number
5			PartShape	Footprint of PCB part
6			CompName	Identifies gates in the same package. Can be used for DeMorgan-equivalence or non-homogeneous components
7	-		GateGroup	Identifies interchangeable gates
8			Prefix	SPICE element prefix (Q, M, R …)
9	+	9	CompGroup	Component Group used to group components to be placed together on the board
10	–		SilosModel	Primitive model name for SILOS
11			SilosTimes	Delay specifier for SILOS primitives
13	+		HidePinNumber	If set, indicates that pin numbers should not be displayed for this gate
17	–		XSimModel	X-Sim primitive model name
18	+		BodyDelay	Body delay parameter for simulation
20	–		VeriModel	Primitive or model name for Verilog
21			VeriTimes	Delay specification for Verilog
22			VeriStrnth	Strength specification for Verilog
25	–		TimilModel	Primitive or Model name for Timemill

Att Number	Att Mod	Attribute Window	Attribute Name	Description
26			TimilExtra	Extra parameter for Timemill
34			Impedance	SPICE parameter
35			Width	SPICE transistor width
36			Length	SPICE transistor length
37			Multi	Multiplication factor for SPICE
38			SpiceModel	Model card for SPICE
39			SpiceLine	Model parameters for SPICE
40			SpiceLine2	Additional SPICE model parameters
41	*		AreaS	Area of source for SPICE
42	*		AreaD	Area of drain for SPICE
43	*		PeriS	Perimeter of source for SPICE
44	*		PeriD	Perimeter of drain for SPICE
45			NRS	Squares of source diffusion, SPICE
46			NRD	Squares of drain diffusion, SPICE
47			DefSub	Substrate node
60	–		GND	Power connection attribute for PCBs; typ GND
61	–		VDD	Power connection attribute for PCBs; typ VDD
62	–		VCC	Power connection attribute for PCBs; typ VCC
63	–		PCBGlobal3	Power connection attribute for PCBs
64	–		PCBGlobal4	Power connection attribute for PCBs
65	–		PCBGlobal5	Power connection attribute for PCBs
66	–		PCBGlobal6	Power connection attribute for PCBs
67	–		PCBGlobal7	Power connection attribute for PCBs
68	–		PCBGlobal8	Power connection attribute for PCBs
69	–		PCBGlobal9	Power connection attribute for PCBs

Att Number	Att Mod	Attribute Window	Attribute Name	Description
70	–		HiloModel	Specifies model for HILO primitives
71			HiloTimes	Specifies HILO delay times
72			HiloStrength	Specifies HILO driving strength
73			HiloParam	
74			HiloParamValue	
75			HiloDelayScale	
76			HiloVisibility	
78			VHDLConfig	
79			VHDLUseLib	
80			VHDLModel	
81			VHDL1	User-definable for VHDL
82			VHDL2	User-definable for VHDL
83			VHDL3	User-definable for VHDL
84			VHDL4	User-definable for VHDL
85			VHDL5	User-definable for VHDL
86			VHDL6	User-definable for VHDL
87			VHDL7	User-definable for VHDL
88			VHDL8	User-definable for VHDL
89			VHDL9	User-definable for VHDL
99			EllaType	
109			PartDesc	Part Description
111			FPList	Footprint List
112			MirrorFootPrint	Mirror Footprint
113			PowerPin	Power Pin List
114			CompLoc	Location
115			CompLayer	Component Layer on Board
116			CompRot	Component Rotation

Att Number	Att Mod	Attribute Window	Attribute Name	Description
117			CompFixed	Component location is fixed
118			CompLocked	Component location is locked
119			CompKey	Key
120			Decoupler	
121			CompHeight	Component Height
122			Voltage	
123			Wattage	
124			Tolerance	
125-130			Reserved	Future PCB

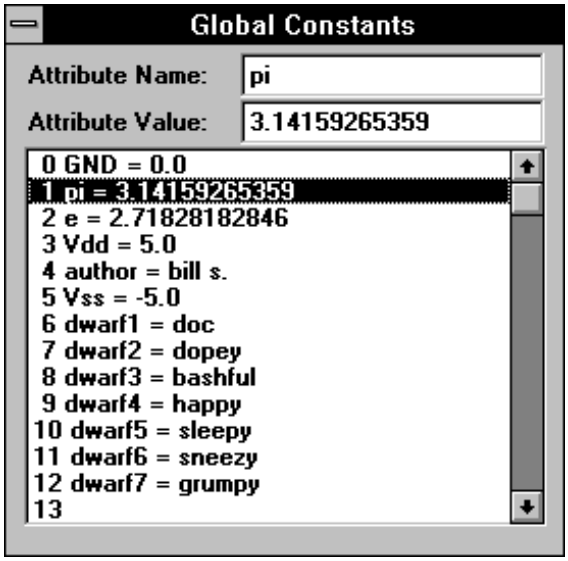
Global Attributes

The Global Constants dialog box defines global attributes in schematics. There are 20 global attributes, all of which are user-defined. They can be used to define anything, but are usually used for attributes that apply to all schematics, such as supply voltage (VDD) or design rule dimensions.

1. Click on the desired global attribute number from the list box.
2. Type the name of the global attribute in the Attribute Name edit field.
3. Type the value of the global attribute in the Attribute Value edit field.

Global attributes can be modified in the Hierarchy Navigator with the Edit: Constants command. Changes are discarded when you exit the Navigator.

Figure 9-8
Global Attribute Editor



Search Paths Menu

Project, Model, and Symbol Libraries

These dialog boxes let you modify the search paths for Projects, Models, and Symbols. The directories are searched in the order they appear in the list.

Libraries are directories that contain symbol or schematic files. SCS also installs libraries that have been compressed into individual files with the .lib extension. Once they have been added to the search path, these compressed symbol libraries are treated the same way as the library directories. You can define your own symbol library directories, but not compressed symbol libraries.

Note: *You can modify symbols from the compressed libraries using the library extraction utiltiy "extlib" to extract a copy of a symbol from a library. You can then edit the symbol and place it in a directory that is searched before the library. The command line is*

```
extlib symbol_name
```

When you highlight a name, it appears in the Path edit box where it can be modified. The paths supplied with the default version of **ecs.ini** names are prefixed with %ROOT. This is the path specified by the Root variable defined during installation. (The default Root path is the directory you installed SCS in, usually **c:\synario**.)

You can add your own path variables to the Registration Database using the Windows program RegEdit:

```
proj_path = c:\projects
my_symbols = d:\user_sym
```

or as SET PATH statements in **autoexec.bat**:

```
SET MY_PATH = e:\my_proj
```

You can access any of these paths in the search path list by prefixing their names with a percent sign (**%**):

```
$proj_path\
%my_symbols\
%MY_PATH\
```

The percent sign (**%**) accesses paths from the registration database. The dollar sign ($) accesses paths from the DOS or UNIX environment.

You do not have to use path variables. You can enter a fully-qualified path for any of your search paths.

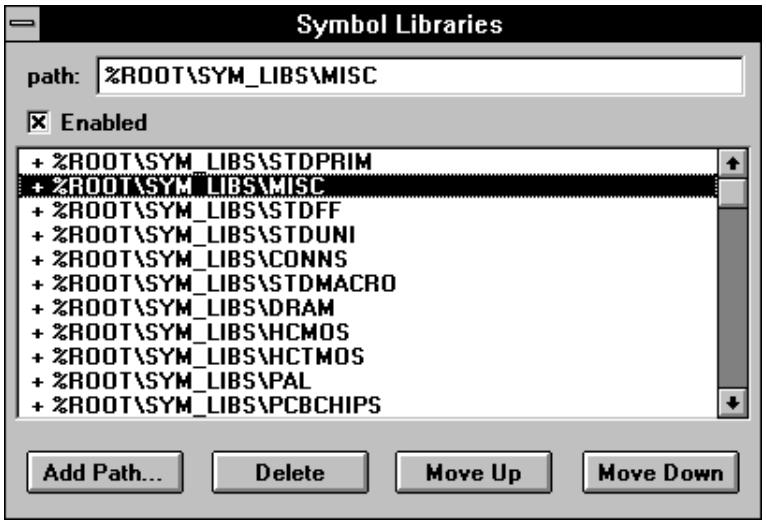
Adding and Deleting Elements to Paths

To add or delete a search path:

1. Select the type of path you want from the Search Paths menu.
2. Click the Add Path button. The generic Open File dialog box appears (labeled Select Project Library, Select Symbol Library, or Select Model Library).
3. Use the controls to select the directory and file type you want.
4. Click on the name of a file in that directory or a library name, then click OK (or just double-click on the filename). The directory containing this file, or the library is added at the bottom of the list.

The Delete button removes the path that is currently highlighted. The Move Up button swaps the highlighted path with the path below it; Move Down swaps the highlighted path with the path above it.

Figure 9-9
Symbol Libraries Search Path Dialog Box



Note: *If you want a path to include one of your own variables, you must enter it manually. The easiest way to do this is to use Add Path to add any directory to the list. You can then modify it in the path edit box.*

Enabling and Disabling Paths

A plus sign (**+**) next to a path means it is enabled, and will be searched. A minus sign (**–**) means the path is disabled and will not be searched. To enable or disable a path, click on its name to highlight it. Then click the Enable check box to mark or clear it.

Libraries and Directory Structures

Libraries are collections of symbols, models, or hierarchical blocks that can be accessed by the schematics of any design. Libraries are stored in directories other than project directories. Using a “common” directory for circuit elements simplifies design organization and makes it easier to ensure that all symbols and models are updated properly.

Program Directories

The following description of the SCS's directory structure applies to the Windows version. The directory structure of the UNIX / Motif version is described in Chapter 1, "Getting Started."

All software and related files used by SCS are located in a master directory called **synario**. (You can, however, select a different directory during installation.) The various files and subdirectories found in the master SCS directory are:

synario	Main directory; contains the SCS executable and help files.
…**config**	Holds the initialization (.**ini**) files and licensing files. (Initialization files for specific designs can also be placed in those designs' directories.)
…**examples****schem**	Contains SCS sample designs.
…**mod_libs**, etc.	Contains model libraries, schematics, and behavioral description files (Verilog, VHDL, and so on).
…**sym_libs**	Contains symbols.

In the Windows version, the installation program automatically adds entries to the Registration Editor that specify the main directory and configuration paths. If you should later move these files (without reinstalling) you should also change these paths in Windows Registration Editor.

User Directories

You generally create a separate directory for each design. However, SCS does not require all the files for a project to be in the same directory. You can put them in any directory.

Library Directories

Libraries store building blocks that can be reused in different designs. A library usually contains related items. For example, a symbol library might consist of symbols for 7400-series TTL devices. Another library might contain symbols for gate array primitives.

SCS interfaces with three different types of libraries:

- Project directories
- Symbol libraries
- Model libraries

You can specify different libraries of each type and can control which ones are on the library search path. Please refer to the section “Search Paths Menu” on page 9-31.

Project Directories

As explained above, SCS does not require all the files for a design to be in the same directory. There are no default entries for this search path, and none are required. Since the Project Directories are searched before the other directories, you might want to add the directories with symbols you have created to the Project Directories search path.

Model Libraries

Model libraries contain schematics that represent higher-level primitives. If the model library is included in the library search path, the Hierarchy Navigator will be able to display a lower-level or more detailed view of the circuit.

Suppose a schematic containing logic gates is drawn and simulated using a gate level simulator. Assume also that a model library exists containing the transistor level schematics of all the logic gates. A netlist for a switch level simulation can be obtained by adding the model library to the library search path, then running the netlister.

Symbol Libraries

Symbol libraries contain the primitive symbols for IC and PCB designs. Typical primitive symbols are PMOS and NMOS transistors, AND gates, NOR gates, or a 74ALS193 counter chip. Symbols are used throughout the hierarchy and at the primitive level in a design.

Library Searching

SCS searches for symbols in a fixed order, and uses the first symbol file it finds with the required name. For example, a symbol *myname* can exist in a project directory and another version of *myname* can exist in a symbol library. Because project directories are searched before symbol libraries, the version of *myname* from the project directory is used.

Symbols are searched for in the following order:

1. The current project directory.
2. The project directories specified in the INI Editor’s Project Libraries dialog box. Project directories closest to the top of the list box are searched first.
3. The symbol libraries specified in the INI Editor’s Symbol Libraries dialog box. Symbol libraries closest to the top of the list box are searched first.

The Model libraries are not searched, because they contain schematics, not symbols.

Important! *If you open a schematic and the symbol search paths are not set correctly, or a symbol file is missing, the Schematic Editor may not be able to locate the correct symbols. If this happens, you'll see the wrong symbols, or blanks where the missing symbols should be. Close the file immediately, without making any changes.*

If symbols are missing from a schematic, it is opened with the name **untitled***, rather than its original name. If you then accidentally save the file, the original version won't be damaged.*

SCS also searches for schematics in a fixed order. It uses the first schematic it finds with the required name. The schematic search order is as follows:

1. The current project directory.
2. The project directories specified in the SCS INI Editor. Project directories closest to the top of the list box in the SCS INI Editor are searched first.
3. The model libraries specified in the SCS INI Editor. Model libraries closest to the top of the list box in the SCS INI Editor are searched first.

The following is a typical symbol search path. %ROOT is the base directory defined in the Windows Registration Editor, usually **c:\synario.**

```
%ROOT\sym_libs\std
%ROOT\sym_libs\misc
%ROOT\sym_libs\ttl
%ROOT\sym_libs\ttl_ls
%ROOT\sym_libs\ttl_als
%ROOT\sym_libs\ttl_as
```

Chapter 10
PCB Design Considerations

This chapter explains the differences in the way the Schematic Editor and Hierarchy Navigator handle IC (integrated circuit) and PCB (printed-circuit board) designs. If you don't use SCS to create PCB layouts, you don't need to read this chapter.

This chapter covers the following topics:

- Configuring for PCB Design
- Differences between IC and PCB Design
- PCB Attributes
- Symbol Types
- DeMorgan-Equivalent Gates
- Instance Names and Reference Designators
- Example PCB Design
- Auto Packaging of PCB Devices
- PCB Back Annotation Interfaces
- PCB Netlisters
- The packlist Bill-of-Materials Program

Configuring for PCB Design

As shipped, SCS is configured for IC design only. If you are doing PCB design, you should reconfigure your system for "PCB Only" or "Both." Also, many of the PCB functions are only available if you purchased the PCB Package option.

To change the configuration:

1. Select "ecs.ini Editor" from the Setup menu of the SCS Executive. When the INI Editor runs, select System Controls from the Controls menu. Then:
2. Click the arrow in the Application Mode combo box.
3. Click on either PCB Only or Both IC & PCB.
4. Select the Save command from the INI Editor's File menu.

The PCB features that can be called from the SCS Executive are now available. PCB features available from the Schematic and Symbol Editors and the Hierarchy Navigator will be available after you have exited from and returned to those applications.

Differences between IC and PCB Design

CAD tools for ICs must be able to handle hierarchical designs with large element counts, but relatively few types of building block cells. PCBs typically don't need as many hierarchical levels, but have large component libraries and must have support for:

- Reference designators
- Pin numbers
- Swapping pins within a package
- DeMorgan-equivalent symbols

An example at the end of this chapter shows how the PCB features are used.

PCB Attributes

Following is a brief description of some of the attributes commonly used for PCB design.

Symbol Attributes

Symbol attributes used in PCB design are shown in Table 10-1.

Table 10-1
PCB Symbol Attributes

Attribute Name	Att. #	Description
RefName	2	**Reference Designator:** This is the physical package designation. More than one symbol may share a reference designator, but all symbols with the same reference designator are considered to be part of the same package. Assign the reference designator prefix (such as U for ICs, R for resistors, C for capacitors) in the Symbol Editor for each symbol type. This prefix is used by the automatic packaging tools in the Schematic Editor and Hierarchy Navigator when assigning packages.

Attribute Name	Att. #	Description
PartNum	4	**Part Number:** Vendor or user part number. Used to identify the actual component associated with this symbol.
PartShape	5	**Part Shape (Footprint):** Indicates the package type (for example, dip14) for use with PCB place and route software.
CompName	6	**Component Name:** Identifies the component name for (possibly different named) symbols. Symbols in the same physical component should share a component name. For example, if there are 2 symbols to represent the DeMorgan equivalents of the same "nand" gate, the symbols would have the same CompName attribute. Likewise, different symbols that are not functionally equivalent but are found in the same package (non-homogeneous gates) would still share the same CompName attribute, but would have different GateGroup attributes (see below).
GateGroup	7	**Gate Group**: Identifies functionally equivalent gates within a package. Gates within a single package with the same value for the GateGroup attribute are considered to be interchangeable.
CompGroup	9	**Component Group**: Component Groups are physical clusterings of components on a printed circuit board. The PCB placement software attempts to place components in the same component group close together.
HidePinNumber	13	**Hide Pin Number:** For components that do not need to display their pin numbers (for example, resistors and capacitors), assigning this attribute with a value of Y or Yes suppresses the pin number display.

Pin Attributes

Pin attributes used in PCB design are shown in Table 10-2.

Table 10-2
PCB Pin Attributes

Attribute Name	Att. #	Description
Polarity	1	**Polarity:** Specifies the signal flow for this pin (input, output, bidirectional).
PinNum	4	**Pin Number:** Indicates the physical pin number of the component connection to this pin. In the Symbol Editor, assign the list of possible connections to each pin, representing each of the functionally equivalent pins in the package (for example, the A input of a 7400 nand gates would be assigned the list 1 4 9 12). The Pin Number and Package Assignment software in the Schematic Editor and Hierarchy Navigator will use this information to override this attribute with specific pin assignments. For Pin or Component symbols, assign a single pin number.
PinGroup	6	**Pin Group:** Identifies functionally equivalent pins within a specific gate. Pins with the same value for the PinGroup attribute are considered to be interchangeable (for example the inputs of a nand or nor gate).
OpenOK	9	**Not an error if left open:** This attribute is used by the electrical rules checker. Normally, the checker would flag any unconnected outputs as an error. Any such outputs with this attribute assigned, however, will not be considered errors.

Symbol Types

The Schematic Editor uses different symbol types to represent primitive elements in ICs and PCBs. The primitives are described below.

Component — In a PCB design, these symbols represent a complete physical package. A typical example is a 7400 quad NAND package. The component symbol has pin numbers that correspond to the physical pin numbers of the package.

Gate — In a PCB design, these symbols represent *one section* of a physical package. A typical example is a single NAND gate from a 7400 quad NAND package. Gates do not need to be assigned to packages as they are placed; the assignment can wait until the logic design is complete.

Gates can be swapped to clean up PCB routing. You can change pin assignments in the Schematic Editor. You can swap gates with the Package: Pin Numbers command in the Schematic Editor or the Hierarchy Navigator.

Note: *Gates that are part of the same component type should have the same CompName attribute. If they are functionally equivalent, they should also have the same GateGroup attribute. Non-homogeneous gates can have different gate group attributes. After they are packaged, gates from the same physical package/component will have the same RefName attribute.*

Cell — These symbols represent the lowest-level primitives used in IC design. Cell symbols do not support pin numbers or reference designators. Cells, therefore, cannot be used in PCB designs.

Pin — Pin Symbols are used to represent physical pins on a printed circuit board. Each pin symbol is a single pin of an edge connector, jumper, or other connector. To identify pin symbols as part of the same physical connector, you assign the same reference designator to them. Pin numbers are reference designators and must be assigned manually to pin symbols (that is, auto packaging options do not work on pin symbols).

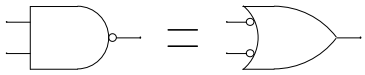
The Schematic Editor supports both pin numbers and pin names. Pin numbers can be used in PCB designs, and pin names are available in both IC and PCB designs.

DeMorgan-Equivalent Gates

DeMorgan-equivalent logic gates have the same function but different schematic representations. A schematic is often easier to understand when the appropriate gate is used.

For example, a circuit whose inputs are ORed to form an error signal can be implemented with NAND gates. However, it's easier to understand the circuit's OR function when it's drawn as OR gates. An example of DeMorgan-equivalent gates is shown in Figure 10-1, where a NAND gate is represented by an OR gate with inversion bubbles on its inputs.

Figure 10-1
DeMorgan Equivalent of a NAND Gate



If different symbols represent the same functionality, you can identify them as DeMorgan-equivalent by giving their CompName attributes the same name. The Schematic Editor will then recognize the gates as being of the same type, and allow them to use the same reference designator.

Note: *Gates that are part of the same component type should have the same CompName attribute. If they are functionally equivalent, they should also have the same GateGroup attribute. Non-homogeneous gates can have different gate group attributes. After they are packaged, gates from the same physical package/component will have the same RefName attribute.*

Instance Names and Reference Designators

The following sections explain instance names and reference designators. Both identify component instances, but they are used differently.

Reference Designators

One difference between PCB and IC designs is the convention used to identify individual components. In a PCB design, elements are identified by a *reference designator*. A reference designator is a unique code for each physical component.

Typical codes are:

Code	Component Type
C1, C2, C3	Capacitors
R1, R2, R3	Resistors
Q1, Q2, Q3	Transistors
U1, U2, U3	ICs

If a component (such as a multi-gate IC or a multi-resistor SIP) contains more than one element, each element uses the same reference designator. The elements are distinguished by their pin numbers.

Instance Names

IC designs use *instance names* to assign a hierarchical identifier. At each level, the lowest-level components derive their full hierarchical names by stringing together the higher-level block names with the primitive's name. A delimiter separates each level in the name. The following are examples of hierarchical instance names.

```
cpu.alu.bit1.carry_lookahead.inverter1
mother_board.disk_controller.lm741
```

Since each component instance is uniquely described by a hierarchical path, two identical devices in different circuits can have the same instance name, without confusion or ambiguity.

The Schematic Editor supports both reference designators and instance names. You can choose the type by setting the Application Mode parameter using the INI Editor. (It appears in the System Controls dialog box under the Controls menu.) Application Mode can be set to the following values:

IC Only Supports instance names and pin names.

PCB Only Supports reference designators and pin numbers.

Both IC & PCB Supports instance names, pin names, reference designators and pin numbers.

Assigning Reference Designators

To assign reference designators:

1. Select the Add Reference Designator command from the Package menu of the Schematic Editor or Hierarchy Navigator.
2. Type the reference designator name and press ENTER. The name is attached to the cursor.
3. Click on the Component or Gate symbol.

You are not warned if you duplicate a reference designator. To find duplicate reference designators, use the Check Packaging command.

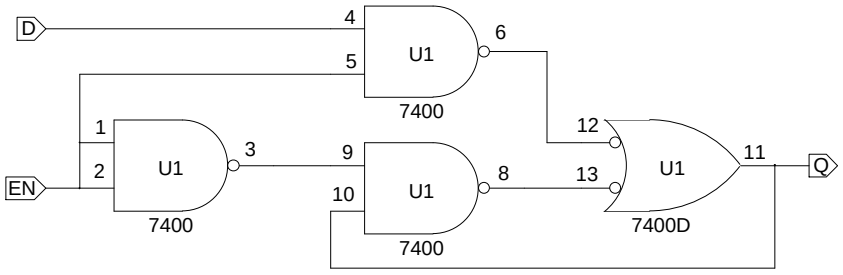
You can also use the Autopackage feature of the Schematic Editor or Hierarchy Navigator to assign reference designators automatically.

Designators for Multi-Gate Devices

The gates in a multi-gate device can have the same reference designator. This is possible because the reference designator refers to the *package*, not to the gates.

Within a multi-gate device, identical gates can be drawn in their normal form, or as their DeMorgan equivalents. (Figure 10-2 shows a latch constructed from a single 7400 IC in which one of the NAND gates is shown in its DeMorgan-equivalent form, as an OR gate with inverted inputs.) A symbol and its DeMorgan equivalent are considered to be the same type, and can have the same reference designator. The symbols are identified as equivalent by sharing the same value for their respective CompName attributes.

Figure 10-2
Latch Constructed from One 7400 Quad NAND IC



Connector pin symbols (symbols of type Pin) are also allowed to have duplicate reference designators. This permits edge connectors to use a single reference designator.

Note: *Gates that are part of the same component type should have the same CompName attribute. If they are functionally equivalent, they should also have the same GateGroup attribute. Non-homogeneous gates can have different gate group attributes. After they are packaged, gates from the same physical package/component will have the same RefName attribute.*

Pin Numbers

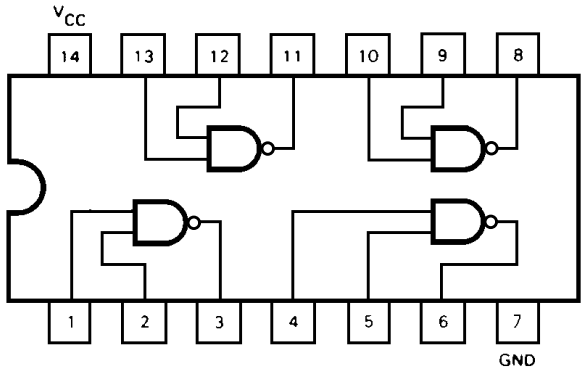
Pin numbers show the correspondence between the pins on a Gate or Component symbol and the pins on a physical device. Only PCB designs require pin numbers.

Component symbols represent a complete physical device whose pins are fixed and cannot be interchanged with other pins in that component. Gate symbols represent *one section* of a multi-section physical device. Gate symbols are assigned groups of pin numbers, where each group includes all the pins that have the same function. For example, the symbol for a 7400 NAND gate has pins with the following names and numbers:

```
Name=IN1  Number=1  4  9 12
Name=IN2  Number=2  5 10 13
Name=OUT  Number=3  6  8 11
```

This is a Gate symbol that represents one-quarter of a complete 7400 IC (as shown in Figure 10-3). When the 7400 IC is given a single reference designator, the pin numbers uniquely identify each gate.

Figure 10-3
Pinout of 7400 Quad NAND Gate



Note: *The use of multiple pin numbers permits pin reassignment for improved routing. On any given Gate symbol, however, only one number per pin is displayed. The number identifies which gate in the component is being used.*

Some Gates have common pins. For example, the 74175 quad D flip-flop has a common clock and clear. The pin number attribute in this case has four identical entries as below.

```
Name=ENAB  Number=4 4 4 4
```

These common pins cannot be swapped, since there would be no benefit in doing so.

Pin numbers are first assigned to Gate and Component symbols in the Symbol Editor.

To initialize the pin numbers:

1. Select the Pin Attribute command.
2. Assign the proper pin numbers to the PinNumber attribute for the various pins.

   Component symbols have only one pin number assigned per pin.

   Gate symbols have one pin number for each gate section in the package. For example, a 74174 IC has six D flip-flops. Each flip-flop has a separate D input and Q output. There is also a common CP clock and common CLR clear. As in the example for the 7400, the attribute entries for the pins look like this:

   ```
   Name=D Number=3 4 6 11 13 14
   Name=Q Number=2 5 7 10 12 15
   Name=CP Number=1 1 1 1 1 1
   Name=CLR Number=9 9 9 9 9 9
   ```

The Add Pin Numbers command in the Schematic Editor assigns pin numbers prior to packaging or layout. The Hierarchy Navigator can also be used to assign pin numbers.

Hiding Pin Number Display

The display of pin numbers on PCB symbols can be suppressed by assinging the HidePinNumbers attribute a value of Y or Yes on the symbol (this is a symbol attribute, not a pin attribute). For example, discrete resistors and capacitors should have a pin number for the purpose of PCB routing, but you may not wish to display them on the schematic.

Gate Assignment

When Gate symbols are placed in a schematic they have the pin assignments from the first section of the package (A). The Autopackage feature can be used to automatically assign the pin numbers.

Automatic assignment might not produce the best board routing, however.

To swap functionally-equivalent gates within a package:

1. Select the Add Pin Numbers command from the Package menu of the Schematic Editor (in a flat design), or the Pin Number command from the Edit menu of the Hierarchy Navigator (in a hierarchical design). The program prompts you for a pin number or gate section.
2. Choose the section to which you wish to reassign the target gate. Enter either the section letter (A, B, C, D) or a unique pin number from the chosen section. (A common clock signal, for example, would *not* be a unique pin.) Press ENTER.
3. Click on the gate to be reassigned. The pin numbers on the gate change to reflect those on to the new gate section.

Reassignment is possible only with Gate symbols, not Components. Gate assignments are swapped *between* packages by reassigning the packages' reference designators.

Pin Swapping

You can also swap functionally identical pins within a Gate for better routing. For example, the inputs of a NOR gate are functionally equivalent.

To swap two functionally-equivalent pins:

Enter the new number for one of the pins and click on that pin. The Hierarchy Navigator checks the Pin Group attribute. If both pins are in the same Pin Group (and the symbol is a Gate from the same section), the selected pin is swapped with the pin with the given pin number.

Because common pins (such as clocks or clears) perform exactly the same function for each gate, swapping them is not permitted.

If the Add: Pin Numbers command in the Schematic Editor is active, you can click on an instance to pick up the current gate letter, or click on a pin to pick up the pin number. This simplifies copying the gate or pin from one instance to another.

You can change the pin numbers in your design using either the Pin Numbers command in the Hierarchy Navigator or the Add Pin Numbers command in the Schematic Editor. If your design is hierarchical, it is usually easier to make these changes in the Hierarchy Navigator.

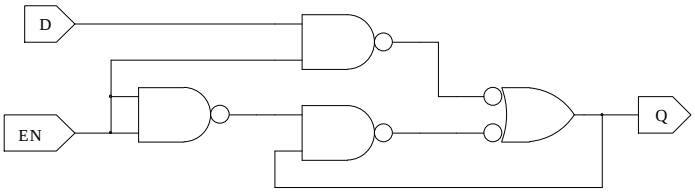
Example PCB Design

The following example builds a latch using a 7400 NAND package. The features described in this example include:

- Use of Gate symbols
- DeMorgan-equivalent symbols
- Pin numbers
- Reference designators
- Assigning gates within a package
- Swapping pins within a gate

Figure 10-4 shows a latch circuit constructed from two-input NAND gates. The OR gate at the output of the latch is a DeMorgan equivalent of a NAND gate.

Figure 10-4
Latch Constructed from 7400 NAND Gates



System Configuration

Before creating a PCB design, set the Application Mode (as described above) to either PCB or Both IC & PCB. This change enables the reference designator and pin number functions in the Schematic Editor.

Creating a Gate Symbol

Two symbols are needed for the latch circuit shown in Figure 10-2. One is a standard NAND gate, taken from the SCS library. The other is the DeMorgan-equivalent NAND gate, drawn as an OR gate with inverted inputs. The following explanation of how to create these symbols assumes the standard pinout for a 7400 quad NAND gate (as shown in Figure 10-3):

- VDD on pin 14
- Ground on pin 7
- Outputs on pins 3, 6, 8, and 11
- Corresponding input pairs on 1 and 2, 4 and 5, 9 and 10, 12 and 13

Note: *Gates that are part of the same component type (for example, 7400 . . .) should have the same CompName attribute. If they are functionally equivalent (as are any of the gates in the 7400 example), they should also have the same GateGroup attribute. Non-homogeneous gates can have different gate group attributes. After they are packaged, gates from the same physical package/component will have the same RefName attribute.*

To create the inverted OR symbol:

1. Click the Edit Symbol radio button in the SCS Executive, then click New. The Symbol Editor runs.
2. Click the Gate radio button in the Type dialog box, then click OK.
3. Draw an OR symbol with two input pins and one output pin. Mark the input pins with inversion bubbles.
4. Use the Symbol Attributes command to set the RefDes attribute to "U." The Schematic Packager uses this default prefix and automatically adds a number to create a unique reference designator.
5. Since this OR symbol is a DeMorgan-equivalent of a NAND gate, be sure its CompName, GateGroup, and PartShape attributes are assigned the same name as the NAND symbol you're using.
6. The following pin attributes are optional. However, they should be assigned as shown so that the design can be properly packaged. Use the Add: Pin Attribute command.

   PinName Choose appropriate pin names, such as IN1, IN2, OUT.

   Polarity Set the polarity of the input pins to In and the polarity of the output pin to Out.

   PinNumber Set one of the input pins to the following list: 1 4 9 12. This corresponds to the pin numbers of all of the IN1 input pins on the 7400 NAND package.

   Set the other input pin to the following list: 2 5 10 13. This corresponds to the pin numbers of the IN2 input pins on the 7400 NAND package.

   Set the output pin to the following list: 3 6 8 11. This corresponds to the pin numbers of the output pins on the 7400 NAND package.

Create the Latch Schematic

To create the latch schematic shown in Figure 10-4:

7. Open a new schematic with the Schematic Editor.
8. Place the four instances as shown in Figure 10-4. All four symbols display pin numbers from the A section of the quad NAND gate. That is, they all have pin numbers of 1, 2, and 3.
9. Wire the circuit to look like Figure 10-4.
10. Use the Package: Reference Designator command to set the reference designator to U1 on all the symbols. This indicates that all the gates are from the same package.
11. Select the Pin Number command. Type 5 (which is a pin in the second gate of the package) and press ENTER.
12. Click on the bottom center NAND gate. The pin numbers of this gate change to those of the second gate in the package: 4, 5, and 6.
13. The Pin Numbers command now prompts for a new pin number or gate section. As before, change the pin numbers of the top center NAND gate to those of the third gate in the package: 8, 9, and 10.
14. Finally, type 11, 12, or 13 and press ENTER. Click on the DeMorgan-equivalent OR gate. Its pin numbers change to those of the fourth gate in the package.

Swapping Packages

If you drew the shortest wiring paths for this schematic, you'd find that there were three crossovers underneath the IC—2–10, 5–11, and 6–13. If the two middle NAND gates were swapped, only two crossovers would be needed (3–9 and 6–12).

To swap the assignments and improve the routing:

1. Select the Pin Numbers command. You are prompted for a pin number or gate section.
2. Type 4 and press ENTER.
3. Click on the top center NAND gate. The pin numbers change to 4, 5, and 6.

   The Schematic Editor checks the PinGroup attribute before performing the pin swap. Pins 4 and 5 belong to the same pin group, so they can be swapped.

4. Type 8 and press ENTER. Click on the bottom center NAND gate. The pin numbers change to 8, 9, and 10.

The Pin Numbers command can also be used to swap functionally-equivalent pins. For example, to swap pins 4 and 5, type 4, press ENTER, then click directly on pin 5 (rather than the body of the gate). The two pins are immediately interchanged.

Note: *The Pin Numbers command allows two or more sections to have the same pin numbers, allowing you to swap sections. However, if you leave two sections with the same numbering, it will eventually be caught as an error.*

Auto Packaging of PCB Devices

The Packager is a utility that assigns reference designators and pin numbers to PCB primitive symbols. Reference designators and pin numbers are usually required for PCB layout, and help document a PCB design.

Iterated symbols are a form of hierarchical structure and must therefore be packaged in the Hierarchy Navigator using the Reference Designator command.

The remainder of this section describes the necessary setup or configuration information required, as well as the individual packaging commands.

The Package menus of the Schematic Editor and Hierarchy Navigator have the eight commands listed below:

Command	Function
Query Packaging	Queries design by packaging information.
Check Packaging	Sets online checking mode of operation.
Clear Packaging	Erases the packaging information on a symbol.
Auto Package	Automatically assigns the packaging information.
Reference Designator	Assigns (or reassigns) individual reference designators.
Pin Numbers	Assigns (or reassigns) individual pin numbers.
Add Reference Designator	Assigns sequences of reference designators.
Add Pin Numbers	Assigns sequences of pin numbers.

Configuration Information

The PCB Packaging software is optional. It must be installed and authorized before it can be used.

The packaging commands are activated in the Schematic Editor and Hierarchy Navigator by setting the Application Mode (using the INI Editor) to PCB or Both IC & PCB. This adds the Package menu to the Schematic Editor. (You must quit the Schematic Editor and return before the menu will appear.)

The packaging commands operate on the RefDes and PinNumber attributes of Gate symbols, and Pin symbols and on the RefDes attributes of Component symbols.

The System accepts reference designators of one or two alpha characters (A–Z) followed by an integer. Pin numbers are limited to four alphanumeric characters.

The Schematic Editor expects the RefDes attribute in a Pin, Gate, or Component symbol definition to be set to the prefix desired for the reference designator. This prefix can be one or two alpha characters. A prefix using any other characters is ignored.

If no value is assigned to the RefDes attribute, a default prefix of "U" is assigned for Gate and Component symbols when packaging in the Schematic Editor.

In a PCB designs, pin symbols are usually used to represent edge connectors. All pins representing an edge connector are given the same reference designator. This is usually done in the Schematic Editor (rather than the Symbol Editor) so that any reference designator can be used. (If the reference designator were assigned in the Symbol Editor, all pins would have the same reference designator.)

In the case of Gate symbols, each pin should have its PinNumber attribute set to the list of pins that the pin could have for each gate in the package. The same pin number can appear more than once in a list, indicating that a pin (such as a clock input) is common to more than one gate. The same number should not appear in lists for different pins.

Query Packaging

The Query Packaging command scans the schematic and lists groups of Gate and Component symbols according to their packaging information. It operates in one of three ways:

- Enter a specific reference designator, such as U12, to display the sheet / zone location and gate section of all elements designated U12 in a list box. Clicking on a symbol does the same thing.

- Enter a reference designator prefix, such as C, to display in a list box the sheet / zone location and gate sections of all elements whose reference designators start with C. Any elements whose reference designators have not been assigned are indicated as such. Clicking on a symbol whose reference designator has not been assigned does the same thing.
- Enter a symbol's file name, such as NAND2, to display the sheet / zone location and gate sections of all NAND2 elements in the schematic in a list box. Clicking on a symbol while holding down SHIFT does the same thing.

Check Packaging

Check Packaging constantly updates the packaging information, except while the Query Packaging command is running. It can be terminated by selecting the Query Packaging command or any command not on the View or Package menus.

Check Packaging scans the schematic for packaging errors. Whenever the other commands change the packaging information, the checking report is updated and displayed in the Packaging Error Report list box.

Any packaging errors are displayed. If the error involves a conflict between two or more symbols, the error is followed by a series of indented lines showing the sheet / zone of each symbol causing the error. Errors that involve only one symbol are listed with the sheet / zone information on the same line.

Clicking on an error from the list box displays the symbol with the error. The cursor is temporarily changed to an "X" and positioned on the symbol. Moving the mouse restores the cursor to its original position and appearance.

The checking command also acts as a repackaging command if no other packaging command is selected. Clicking on any symbol repackages the symbol as if you had executed Clear Package, then Auto Package.

Error Messages

The following errors are reported by the packaging checker:

`Gate Multiply Assigned`

The same reference designator or the same pins appear on two or more gates. The multiply assigned gates appear in an indented list immediately below the error message.

`Invalid Pins on Gate`

The pin numbers on the instance are not a valid set, according to the symbol definition.

`RefDes on Different types`

The same reference designator is used on symbols of differing types (where the type is the CompName attribute, or the base name of the symbol file if no CompName has been specified). The offending symbols appear in an indented list immediately below the error message.

`RefDes Multiply Assigned`

The same reference designator appears on two or more components. The conflicting symbols appear in an indented list immediately below the error message.

`Unpackaged Symbol`

A symbol has not yet been packaged.

Symbols are always shown with their respective sheet / zone locators at the beginning of the line.

This sheet / zone locator has the form of an integer representing the sheet, followed by a letter and number coordinate pair that specifies the X–Y location of the symbol on the sheet. For example, a symbol having a sheet / zone locator of 3D5 would be located on sheet 3 in the region where zones D and 5 intersect.

Clear Packaging

This command clears the packaging information on one or more symbols. It can clear:

- A single symbol. Click on the symbol.
- All symbols. Type ALL, then press ENTER.
- All symbols enclosed in a drag box.
- All symbols of the same type. Hold down SHIFT while clicking on one symbol of the desired type.
- All symbols with a specified prefix. Type the prefix, then press ENTER.

After being cleared, the reference designator returns to the prefix you specified in the symbol definition. The pins on Gate symbols are set to the values for the first gate section in the package.

Auto Package

Auto Package automatically selects an appropriate reference designator for Component symbols, and reference designator and pin assignments for Gate symbols. It ignores Pin symbols.

Reference designators for Component symbols consist of the prefix specified in the symbol's definition, plus the lowest number that has not already been assigned to that prefix. For example, if U1, U2, and U4 already exist, the next U-series reference designator is U3.

If you enter the prefix without a number, the count automatically starts at 1. If you append a number to the prefix, the count starts with that number. In either case, the Editor automatically skips over any numbers that have already been used with that prefix.

If you specify a number, the prompt line shows:

`Package (# > = nn) =`

where *nn* is the smallest unused number for that prefix.

Gate symbols are first assigned reference designators and pin assignments from any free gate sections in existing packages. New packages are assigned only if there are no unused gates. The name is selected for new packages as it is for Component symbols.

The Auto Package command can process one, several, or all unpackaged symbols in the schematic. The command packages:

- A single symbol. Click on the symbol.
- All symbols. Type ALL, then press ENTER.
- All symbols enclosed in a drag box.
- All symbols of the same type. Hold down SHIFT while clicking on one symbol of the desired type.
- All symbols with a specified prefix. Type the prefix, then press ENTER.

To package iterated instances or hierarchical designs, you must use the commands in the Hierarchy Navigator.

Reference Designator

This command explicitly sets the reference designator on a Gate, Component, or Pin symbol. Invalid assignments are allowed, but *only* so you can move reference designators among symbols. Be sure that no invalid assignments remain when you quit this command. Invalid assignments will be trapped later by the PCB Check command.

You first specify the reference designator to be assigned by one of the following methods:

- Enter the full name (for example, U23) and press ENTER. *Or*
- Enter the prefix (for example, U) and press ENTER. The system selects the first unused name with that prefix. *Or*
- Click on a symbol. The system selects a unique name with the same prefix as the selected symbol. *Or*
- Click on a Gate or Pin symbol while holding down SHIFT. The system copies the name from the symbol.

The name you're assigning is attached on the cursor. The command assigns the name to:

- A single symbol when you click on the symbol. The name on the cursor is then incremented to the next available name.
- A single symbol when you click on the symbol while pressing SHIFT. The name on the cursor is attached to the selected symbol but *is not* incremented. Use this method for gate assignments with multiple sections in the same package (which therefore have the same reference designator).
- All symbols enclosed in a drag box. The symbols are assigned unique names starting with the selected name. The name attached to the cursor is incremented to the next available.
- All symbols enclosed in a drag box while you press SHIFT. All selected symbols have the assigned name attached to them. The name attached to the cursor does not change.

To package iterated instances, you must use the commands in the Hierarchy Navigator. The RefDes attribute on an iterated instance consists of a list of reference designators. The list is delimited by commas (**,**) or spaces. A sequence of reference designators is formed by enclosing the numeric range in parentheses **( )**, brackets **[ ]**, or curly braces **{ }**.

For example, a symbol with instance name CC[1:20] specifies 20 iterations of a symbol. If you assign this symbol a reference designator of C1,C3,C(5:20), C22,C24, then the 20 iterations of the symbol are referenced as C1, C3, C5, C6, C7, C20, C22, C24.

Pin Number

This command explicitly sets the pin numbers on Gate symbols and assigns a gate symbol to a particular section of a component. It also lets you swap pins on a Gate symbol.

You must enter the gate section or pin number to be assigned using one of the following methods:

- Type the pin number, then press ENTER. *Or*
- Type a letter corresponding to the gate section in the package (A, B, C, D), then press ENTER. *Or*
- Click on a pin to copy its number. *Or*
- Click on a symbol body to copy its gate section.

The gate section or pin number is attached to the cursor. The gate to be numbered is designated by clicking on its body or on one of its pins. The command checks the requested assignment to be sure it's a valid gate for the symbol.

The command then assigns the correct pin numbers to the gate. If an explicit pin number is requested and if one of the gate's pins is selected, the command attempts to assign that pin by swapping with other pins in the same pin group.

This command does not prevent duplicate gate assignments or common pin conflicts. These errors are detected and reported by the Check command.

PCB Back Annotation Interfaces

SCS provides a number of printed circuit board (PCB) back annotation interfaces. The following netlist formats are currently supported:

- PADS PCB
- Racal-Redac Interchange Format (RINF)

If additional netlist formats are developed, they will be documented in release and update notes.

The Back Annotation Programs

Back annotation can be performed with either of two supplied programs, **pcbback** or **scback**. Unless otherwise noted, both programs behave identically.

The Schematic Back Annotation program (**scback**) reads a file containing back annotation commands for the schematic. **scback** should be called from the Utilities menu of the SCS Shell. You need to add one of the following lines (depending on whether you use PADS PCB or RINF format) to the [&Utilities] section of **pcshell.ini**:

```
Back Annotate Schematic from PADS=scback.exe -pads, Schematic:,\*.eco,all,&f

Back Annotate Schematic from RINF=scback.exe -rinf, Schematic:,\*.irp,all,&f
```

The Hierarchy Back Annotation program runs from within the Hierarchy Navigator. You need to add one of the following lines (depending on whether you use PADS PCB or RINF format) to the Navigator Tools section of **pcshell.ini** using the INI Editor:

```
&Back Annotate from PADS=pcbback.exe -pads &f

&Back Annotate from RINF=pcbback.exe -rinf &f
```

Both back annotation programs take a single command line argument (the *basename* of the schematic) and look for a a back annotation command file. If you do not specify one of the following command line options, the programs look for a file with the name *basename*.**ban**. An alternate format can be specified with one of the following command line formats:

-pads	Read PADS PCB back annotation file *design*.**eco**.
-rinf	Read Racal-Redac RINF format back annotation file *design*.**irp**.

After **scback** runs, you must load (or reload) the schematic to view the results. **scback** does not support iterated instances, buses, or bus pins.

The default format (.**ban**) generic files can contain the following commands. (**pcbback** does not support these commands.)

Command	Function
I	Back annotate instance
N	Back annotate net
P	Back annotate pin
R	Rename component (change reference designator)
S	Swap pins
G	Swap gates
#	Comment line (Blank lines are considered comment lines, and ignored.)

The I, N, and P commands take three arguments:

```
command attribute instance/element_name value
```

- Pin names take the form *instance_name–pin_name*.
- The new attribute *value* is assigned as a schematic override for the specified instance, net, or pin.

The R, S, and G commands work with reference designators and pin numbers rather than names.

- The R command takes two arguments: the old reference designator and the new reference designator.
- The S command takes three arguments: the first is the reference designator of the gate, and the second and third are the pin numbers to be swapped.

- The G command takes two groups of arguments. Specify the reference designator and section letter of each gate, separated by a period (**.**). (For example, U1.A or U3.D.) If the gates are not equivalent (that is, they do not have the same CompName), the program may fail to find one of the specified gates.

The formats of the generic commands are shown below:

```
I attribute instance_name value

N attribute net_name value

P attribute pin_name value

R ref1 ref2

S ref pin1 pin2

G ref1.section1 ref2.section2

#any_text
```

PADS PCB-specific Features

The *design*.**eco** file (where *design* is the name of your design) must have been produced by PADS. The back annotation interface opens this file automatically and updates attributes in the Hierarchy Navigator to reflect any changes. You must perform a Save from the Navigator to preserve the back annotated changes.

The commands for the PADS PCB format operate the same way as the generic R, S, and G commands.

RENPART	rename component (change reference designator)
SWPPINS	swap pins
SWPGATES	swap gates
REM	comment

Each command appears by itself on a line, followed by the component and pin information on separate lines. Each command remains in effect until another command line appears:

```
*RENPART*
ref1 ref2
ref1 ref2
…
*SWPPINS*
ref pin1 pin2
ref pin1 pin2
…
```

PADS PCB Error Messages

`Unable to Open Error File design.err For Writing`

scback could not open a file to list errors. You may be out of disk space, or read / write access is set incorrectly.

`Missing PADS Back Annotation File`

The file *design*.**eco** does not exist or could not be found.

`Insufficient Memory to Process Design`

You need more free RAM to run this module.

RINF-specific Features

The *design*.**irp** file (where *design* is the name of your design) must have been produced by the Racal-Redac program. The back annotation interface opens this file automatically and updates attributes in the Navigator to reflect any changes. You must perform a Save from the Hierarchy Navigator to preserve the back annotated changes.

The commands for the PADS PCB format operate the same way as the generic R, S, and G commands, except that it recognizes only the first pin list format. The command names are also different:

The formats of the RINF-specific commands are shown below:

`.REN_COM` *`ref1 ref2`*

(rename component (change reference designator))

`.SWA_PIN` *`pin1ref1 pin1ref2`*

(swap pins)

`.SWA_GAT` *`ref1 ref2 pin1ref1 pin1ref2 pin2ref1 pin2ref2 pin3ref1 pin3ref2`* …

(swap gates)

`.REM` *`comment`*

(comment)

RINF Error Messages

`Unable to Open Error File design.err For Writing`

scback could not open a file to list errors. This is usually caused by incorrect read / write file or directory permissions.

`Missing RINF Back Annotation File`

The file *design*.**irp** does not exist or could not be found.

`Insufficient Memory to Process Design`

You need more free RAM to run this module.

PCB Netlisters

SCS provides a number of printed circuit board (PCB) netlist interfaces. (These netlisters are an option supplied with the PCB software package.) The following netlist formats are currently supported:

- Cadnetix
- CADSTAR
- PADS PCB
- PCAD
- Racal-Redac Interchange Format (RINF)
- Tango

If netlist conversion programs for additional formats are developed, they will be documented in the release notes for those products.

A single program called **pcbnet** performs these conversions. To use **pcbnet**, add an entry to your INI file using the Navigator Tools dialog box of the INI Editor. The entry consists of the application name **pcbnet** with one or more of the following command line options:

Option	Action
-cadnetix	Write Cadnetix netlist file (.**cdx**).
-cadstar	Write CADSTAR netlist file (.**cdi**).
-pads	Write PADS PCB netlist file (.**pad**).
-pcad	Write PCAD netlist file (.**alt**).
-rinf	Write Racal-Redac RINF format netlist file (.**frp**).
-tango	Write Tango netlist file (.**net**).

-ext=.**abc**	Create a netlist file with the specified extension (that is, *design*.**abc**). This overrides the default extension for the selected netlist format.
-nohead	Suppress printing of time and date header.
-power	Include the "hidden" power and ground pins in the netlist.
-view	Run the specified viewer to display the netlist file immediately after netlist creation.

The remainder of this section describes features common to all PCB netlisters and the features of individual netlisters.

Features Common to All PCB Netlisters

PCB netlists share many features. Elements in the netlist are referenced using RefDes and pin numbers. Each format, however, usually has a unique arrangement of reference designators and pin numbers.

Attributes Needed for Netlisting

The following attributes are commonly used by the PCB netlisters. If these attributes are not already defined in the Attributes section of the INI file, you need to add them for the PCB netlister interfaces to work properly. Not every netlister requires all the attributes defined below.

Attribute	Description
Type	The name of the symbol (symbol attribute #1). For example, NAND2 or 7404.
RefDes	The reference designator attribute used in PCBs (symbol attribute #2). For example, U2 or R23.
Value	A component's value, such as a capacitor's capacitance (symbol attribute #3). For example, 10K or 30u.
PartNum	The PCB part number for a component (symbol attribute #4).
PinName	Pin identifier for PCB interfaces (pin attribute #0). For example, IN1 or Y.
Polarity	Pin polarity (pin attribute #1). Permitted values are INPUT, OUTPUT, and BIDIR, which can be abbreviated as I, O, and B.

PinNumber Component pin numbers (pin attribute #4). For example, the symbol for the 7400 series NAND Gate has pins with the following names and numbers:

PinName= IN1 PinNumber= 1 4 9 12

PinName= IN2 PinNumber= 2 5 10 13

PinName= OUT PinNumber= 3 6 8 11

PartShape Specifies the component package type used by the PCB layout program (symbol attribute #5). For example, DIP14 or PGA128.

CompName Replaces the symbol name in PCB netlists (symbol attribute #6). For example, 74LS00 or ROM2KX8. The value (if specified) is substituted for the symbol's filename in the PCB netlist. This allows component names longer than eight characters to be netlisted. (DOS file names are limited to eight characters, and the UNIX/Motif version of SCS also observes this limit, for file compatibility.)

DeMorgan-equivalent gates can be defined by giving the CompName attribute the same value on different symbols. Two or more gate symbols having identical CompName attributes can share the same reference designator.

GateGroup Identifies whether symbols with the same CompName attribute are swappable (symbol attribute #7). Non-swappable symbols have different letter values for the GateGroup. Swappable have identical GateGroup values. This attribute is used when assigning non-homogeneous gates.

PCB_Global Ten symbol attributes are reserved for hidden power and ground pins in component symbols (symbol attributes #60 to 69). These are described in more detail later in this section.

PinGroup Indicates which other pins in a component have the same function (pin attribute #6). In a 7400 series NAND gate, for example, the inputs have the same PinGroup. Pins with the same PinGroup attribute can be swapped in a Gate symbol.

Width Associates track width with the net (net attribute #8.)

The symbol and pin attributes listed above are assigned using the Symbol Attributes and Pin Attributes dialog boxes in the Symbol Editor. If an attribute is assigned in the Symbol Editor, that attribute becomes the default value for each instance of the symbol on the schematic. These default values can be overridden for any particular instance of the symbol with the Add: Attribute command of the Schematic Editor.

All nets with names longer than the number of characters permitted for a particular netlist are given a unique net name starting with "N__" (N followed by 2 underscores).

The **pcbnet** netlisters use the Block, Component, Gate, or Pin symbol types. The Cell symbol type is invalid in PCB designs. Any other type is ignored. Component or Gate symbols must have their RefDes and PinNums attributes specified.

Non-homogeneous gates are supported in SCS. Refer to the example at the end of this chapter.

Preparing Schematics

The circuit must be fully packaged prior to extracting the netlist. Each symbol should have an assigned reference designator. Pin numbers are assigned according to the gate sections within each package.

Reference designators can be assigned in the Schematic Editor or the Hierarchy Navigator. Assignments in the Navigator override assignments made in the Schematic Editor. If the design contains repeated hierarchy, the Hierarchy Navigator must be used to assign the reference designators and pin numbers to the gates and components in the repeated portions.

The netlisters also require Block symbols to have corresponding schematics. At the lowest hierarchical level, all symbols should be Component, Gate, or Pin. (That is, a Block symbol cannot be at the lowest level. The Block symbol must represent a schematic which is composed of only Components, Gates, and Pins.)

DeMorgan-equivalent symbols are given the same name in their CompName attribute. The processor considers the two symbols to be equivalent and lets them share a package.

PCB Power Pins

Most PCB components have at least one connection to the power supply (usually VCC or GND). You can hide these connections to reduce schematic clutter.

Symbol attributes 60 through 69 represent hidden power and ground pins. The attribute's name corresponds to the name of the global net. The attribute's value corresponds to the pin number on the package. For example, the default INI file has the following attribute names corresponding to the attribute numbers below:

Attribute Number	Global Net
60	GND
61	VDD
62	VCC

Attributes 63–69 can be assigned by the user to other power supply values.

CAUTION: If you assign a name to a hidden power pin, you must also assign the same name to one of the Global Nets in* ecs.ini. *(Use Global Nets from the Controls menu of the INI Editor to make the assignment.) Otherwise, you will eventually receive an error message saying that there is no global net for the hidden power pin.

To apply hidden power and ground pins to a 7400 quad NAND gate:

This example assumes pin 7 is connected to ground and pin 14 is connected to VCC.

1. Load the NAND symbol into the Symbol Editor.
2. Use the Symbol Attribute command from the Add menu to set the value of the GND attribute to 7, the power pin on this DIP.
3. Use the Symbol Attribute command from the Add menu to set the value of the VCC attribute to 14, the power pin on this DIP.

You typically define attributes for GND and VDD. Up to ten different nets (named in attributes 60–69) can be defined as potential connections to these hidden pins. The symbol libraries must contain the default connections to the power and ground pins. If the attribute is not assigned or has the value 0, the net is not connected to that symbol.

If a particular pin is sometimes connected to VDD and sometimes connected to VCC, use the following procedure:

1. Edit the attributes on the symbol so that the VDD symbol attribute is the actual pin number and the attribute for VCC is 0.
2. Find the instances on the schematic that need to have this pin connected to VCC and set the VDD attribute to 0.
3. Then set the VCC attribute on these instances to the actual pin number.

Note: *If you assign a pin number to one of the hidden power net attributes, you must use the corresponding power net in the schematic. Otherwise, you will eventually receive a "Missing Power Net" error message.*

If you use hidden power pins in your schematic, you must add the following command line option to the netlister entry:

```
-power
```

This option tells the netlister to add the power connections to the netlist. If this option is not used, some netlisters will not include the hidden power pins in the netlist. A typical command line (as shown in the INI File) might be:

```
PADS PCB Netlist = pcbnet -pads -power
```

View Report Facility

Errors are normally displayed automatically. If you want to view an error file after it has been closed, or read a file generated by a non-SCS program, you can use the View Report facility.

If you load an error file, the View Report dialog box displays any errors that occurred during netlist generation. It lists the generated netlist and the errors. You can click on any error in the list to highlight it in the schematic. Note that View Report is *not* the same feature as the -view option, which brings up the Notepad after the Netlist is generated.

Error Messages

The following error messages may be issued by the **pcbnet** netlister:

`Missing Pin Number(s) on Symbol Type type`

Pin numbers were not assigned to the symbol. Use the Symbol Editor to add them.

`Missing RefDes on Symbol Instance instance`

A Reference Designator was not assigned to the symbol.

`Missing .SCH file for Block Symbol symbol`

The symbol is a Block symbol, but the matching schematic was not found. You must copy the schematic into the project's directory, or into any library that appears in the Model Libraries path (set in the INI Editor). If this error occurred because the symbol was incorrectly defined as a Block symbol, load it into the Symbol Editor and change its type to Gate or Component.

`Encountered CELL Type Symbol symbol`

The symbol was created as a Cell symbol. This symbol type is intended for IC design and should not be used in PCB applications. Load the symbol into Symbol Editor and change its type to Component or Gate, or replace the symbol with one of the appropriate type.

`Two Instances of Pin pin on Different Nets`

The same pin can appear on different symbols (for example, a common enable pin). This message appears if each instance of the common pin is connected to a different net.

This message also appears if two instances of a symbol are assigned the same Reference Designator and their common pins are not connected to the same net.

`RefDes name Assigned to Different Symbol types`

The same Reference Designator name was assigned to two different symbol types.

`Missing Power Net` `name` *`From Attribute`* `attribute`

An attribute in the range for global power nets (60–69) has been defined, but this net is not used anywhere on the schematic. The global net must be used in any schematic with symbols that have power pins.

`Non-Global Power Net` `name` *`From Attribute`* `attribute`

An attribute in the attribute number range for global power nets has been defined, but there is no global net with the same name.

PADS PCB-specific Features

The following features apply only to the PADS PCB netlist.

The PADS PCB netlister supports the PartShape symbol attribute. The examples below show how the PartShape is formatted in the netlist. It follows the part type and is separated from the part type with an at sign (**@**).

```
U1    LS7400@SO-14
U2    AMD386@PCC128
```

The net attribute Width represents trace widths in PCB applications. This information is netlisted with the proper syntax following the signal definition in the PADS PCB netlist. Net attributes can be added to a design using the Add: Attribute command in the Schematic Editor or the Edit: Attribute command in the Hierarchy Navigator.

The PADS PCB interface creates a netlist file with the name *design***.pad** (where *design* is the base name of your design). This file should be copied to the PADS PCB design directory and renamed with the **.asc** suffix.

The PADS PCB processor combines a checking function with the generation of the netlist. Error reports are inserted in the netlist file. If any errors occur, the processor calls the viewer program and displays the entire netlist, including error reports, before returning to the Hierarchy Navigator.

If the -power command line option is not specified, the PADS PCB netlister assumes you are using the standard PADS PCB library of parts that comes with the PADS PCB product. This library already has power- and ground-pin definitions and therefore does not require SCS hidden power pins.

Net names longer than 12 characters are assigned unique names starting with "N__" (N followed by 2 underscores). The period (**.**) in a net name is converted to a slash (**/**).

RINF-specific Features

The REDAC Interface Format lets you extract a RINF-compatible netlist. This process creates a netlist file with the name *design*.**frp** (where *design* is the base name of your design).

The RINF processor combines a checking function with the generation of the RINF-compatible netlist. Error reports are inserted in the netlist file. If errors occur, the processor invokes the default editor specified in the INI file and displays the entire netlist, including error reports, before returning to the Navigator.

Net names starting with a period (.) have the NetName enclosed in quotation marks (" "). An apostrophe (') in the NetName is converted to a backslash (\).

PCAD-specific Features

The PCAD netlister converts invalid characters (for example, **[ ]**) to underscores (**_**). A bus element named ABC[0] appears in the Netlist as ABC_0_. NetNames longer than 24 characters are assigned unique names starting with "N__" (N followed by two underscores).

To ensure unique net names, do not use an underscore as the last character. For example, if your schematic contains nets named BETA_1_ and BETA[1], the PCAD netlister converts the latter to BETA_1_, the same as the first name.

The PCAD netlister flags duplicate names as commented warnings in the netlist and calls the View Report command to display the netlist. Click on the line with the warning to highlight the net with the duplicate name.

CADNETIX-specific Features

Net names longer than 16 characters are assigned unique names starting with "N__" (N followed by two underscores). The dollar sign (**$**) in net names is converted to an underscore (**_**).

CADSTAR-specific Features

A warning is flagged if more than 1024 nodes are connected to a net.

The allowed value of track width is 0–7, inclusive.

TANGO-specific Features

Net names longer than 16 characters are assigned unique names starting with N__ (N followed by two underscores). Net names with non-alphanumeric characters (except the underscore) are also assigned unique names starting with N__.

Required Attributes

The following attributes are required by various netlisters. Those in *italics* are optional.

Note that if the CompName attribute is not supplied, the netlister substitutes the name of the symobl (filename).

Component Section

Netlist Format	Required Attributes
CADNETIX	CompName, RefName, *PartNum* If PartNum is not set, it is netlisted as "X".
CADSTAR	RefName, PartShape
PADS PCB	RefName, CompName, *PartShape, Value* Setting PartShape or Value to an asterisk (*****) is the same as setting it to null (no value).
PCAD	CompName, RefName
RINF	RefName, PartNum, *PartShape* If PartShape is not supplied, CompName (the Symbol Name) is used
TANGO	RefName, PartShape, CompName, Value

Net Section

Net names are automatically assigned by SCS if you leave them unassigned.

Netlist Format	Required Attributes
CADNETIX	PinNum, *NetName*
CADSTAR	PinNum, *NetName, Width*

PADS PCB	PinNum, *NetName, Width*
PCAD	PinNum, *NetName*
RINF	PinNum, *NetName*
TANGO	PinNum, *NetName*

Non-Homogeneous Gates Example

The following table shows how attributes are defined for non-homogeneous gates in a 747074 six-section multifunction circuit. This component has two inverters, a 3-input NOR gate, a 3-input NAND gate, and two D-type flip-flops.

Symbol	CompName	GateGroup	Gnd (Sym Att #60)	VCC (Sym Att #61)	PinName	PinNumbers
747074_1.sym (inverters)	747074	A	12	24	A B	1, 23 2, 22
747074_2.sym (NAND)	747074	B	12	24	A B Y	3 4 5
747074_3.sym (flip-flops)	747074	C	12	24	CLK _PRE D _Q _CLR Q	6, 18 7, 17 8, 16 9, 15 10, 14 11, 13
747074_4.sym (NOR)	747074	D	12	24	A B Y	20 21 19

In this example, the four symbols are different logic elements that represent the different sections of the components. The two inverters are interchangeable, as well as the two flip flops. The NAND and NOR are not interchangeable. You could provide two different representations of the NAND or NOR sections using DeMorgan equivalence.

DeMorgan-Equivalent Gates Example

DeMorgan equivalence is demonstrated using the 7408 quad two-input AND gates.

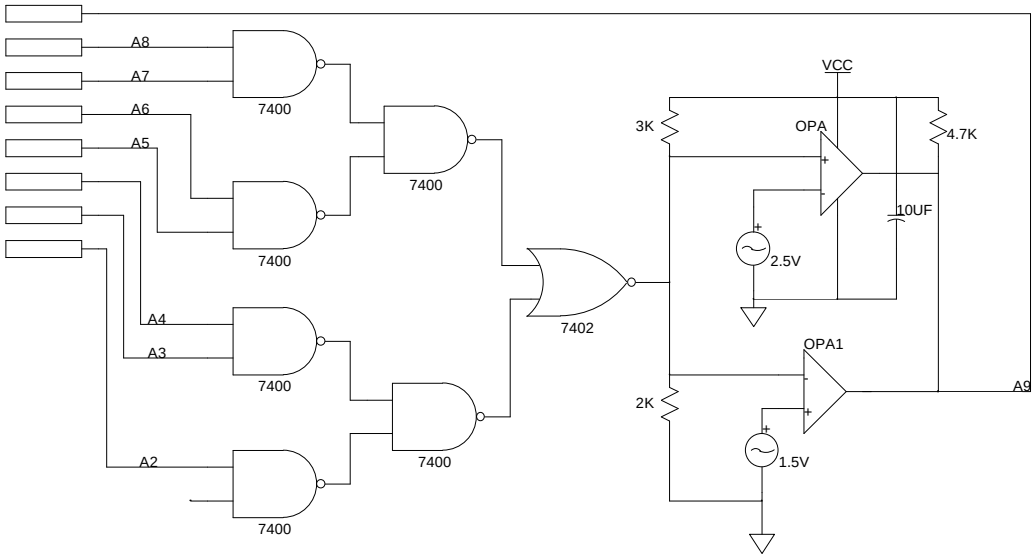
Symbol	CompName	GND pin	VCC pin	PinName	PinNumber
7408.sym	7408	7	14	A	1, 4, 9, 12
				B	2, 5, 10, 13
				Y	3, 6, 8, 11
7408_alt.sym	7408	7	14	A	1, 4, 9, 12
				B	2, 5, 10, 13
				Y	3, 6, 8, 11

The two symbols are the same in this example, except that one symbol draws the AND in the normal way, and the second symbol draws the AND as an OR with inversion bubbles on all the pins.

Netlisting Example

This example shows a simple schematic followed by the netlists that are produced using the various ASCII netlister program. The schematic is **quant.sch**, which appears in the **examples\pcb** subdirectory of your SCS installation.

Figure 10-5
quant.sch Schematic Used in Netlist Examples



ASCII Netlist Format

```
COMMENT asciinet 1.0 - SCS 2.7 Beta 1
COMMENT FILE quant.net - 7/18/94 4:20:22 PM
INSTANCE 7400 I_11
SYMATTR RefName U2
PIN INA       N_3
PINATTR PinNum 4
PINATTR PinGroup 1
PIN INB       N_4

PINATTR PinNum 5
PINATTR PinGroup 1
PIN OUT       N_6

PINATTR PinUse O
PINATTR PinNum 6
INSTANCE 7400 I_12

SYMATTR RefName U2
PIN INA       N_1
```

```
PINATTR PinNum 1
PINATTR PinGroup 1
PIN INB       N_2
PINATTR PinNum 2
PINATTR PinGroup 1
PIN OUT       N_5

PINATTR PinUse O
PINATTR PinNum 3
INSTANCE 7400 I_10
SYMATTR RefName U1
PIN INA       A8
PINATTR PinNum 1
PINATTR PinGroup 1
PIN INB       A7
PINATTR PinNum 2
PINATTR PinGroup 1
PIN OUT       N_1

PINATTR PinUse O
PINATTR PinNum 3
INSTANCE 7400 I_9
SYMATTR RefName U1

PIN INA       A6
PINATTR PinNum 4
PINATTR PinGroup 1
PIN INB       A5
PINATTR PinNum 5
PINATTR PinGroup 1
PIN OUT       N_2
PINATTR PinUse O

PINATTR PinNum 6
INSTANCE 7400 I_8
SYMATTR RefName U1
PIN INA       A4
PINATTR PinNum 9
PINATTR PinGroup 1
PIN INB       A3
PINATTR PinNum 10
PINATTR PinGroup 1
PIN OUT       N_3
PINATTR PinUse O
PINATTR PinNum 8
INSTANCE 7400 I_7
SYMATTR RefName U1
```

```
PIN INA       A2
PINATTR PinNum 12
PINATTR PinGroup 1
PIN INB       N_10
PINATTR PinNum 13
PINATTR PinGroup 1
PIN OUT       N_4
PINATTR PinUse O
PINATTR PinNum 11
INSTANCE 7402 I_5
SYMATTR RefName U3
PIN INA       N_5
PINATTR PinNum 2
PINATTR PinGroup 1
PIN INB       N_6
PINATTR PinNum 3
PINATTR PinGroup 1
PIN OUT       N_8
PINATTR PinUse O
PINATTR PinNum 1
INSTANCE VSRC I_26
SYMATTR RefName S1
PIN +         N_9
PINATTR PinNum 1
PIN -         GND
PINATTR PinUse O
PINATTR PinNum 0
INSTANCE VSRC I_3
SYMATTR RefName S2
PIN +         N_7
PINATTR PinNum 1

PIN -         GND
PINATTR PinUse O
PINATTR PinNum 0
INSTANCE OPA I_29
SYMATTR RefName U4
PIN IN1       N_8
PINATTR PinNum 5
PIN IN2       N_9

PINATTR PinNum 6
PIN IN3       VCC
PINATTR PinNum 8
PIN IN4       GND

PINATTR PinNum 4
PIN OUT       A9
```

```
PINATTR PinUse O
PINATTR PinNum 7

INSTANCE OPA1 I_24
SYMATTR RefName U4
PIN IN1       N_7
PINATTR PinNum 3
PIN IN2       N_8
PINATTR PinNum 2
PIN OUT       A9
PINATTR PinUse O
PINATTR PinNum 1
INSTANCE CAP CAP[0]
SYMATTR RefName C1
PIN A         VCC
PINATTR PinUse B
PINATTR PinNum 1
PIN B         GND
PINATTR PinUse B
PINATTR PinNum 2
INSTANCE CAP CAP[1]
SYMATTR RefName C2
PIN A         VCC
PINATTR PinUse B
PINATTR PinNum 1
PIN B         GND
PINATTR PinUse B
PINATTR PinNum 2
INSTANCE CAP CAP[2]
SYMATTR RefName C3
PIN A         VCC
PINATTR PinUse B

PINATTR PinNum 1
PIN B         GND
PINATTR PinUse B
PINATTR PinNum 2
INSTANCE CAP CAP[3]
SYMATTR RefName C4
PIN A         VCC

PINATTR PinUse B
PINATTR PinNum 1
PIN B         GND
PINATTR PinUse B
PINATTR PinNum 2
INSTANCE RES I_21
SYMATTR RefName R2
```

```
PIN A         N_8
PINATTR PinUse B
PINATTR PinNum 2
PIN B         GND
PINATTR PinUse B
PINATTR PinNum 1
INSTANCE RES I_22
SYMATTR RefName R1
PIN A         VCC
PINATTR PinUse B
PINATTR PinNum 2
PIN B         N_8
PINATTR PinUse B
PINATTR PinNum 1
INSTANCE RES I_27
SYMATTR RefName R3
PIN A         VCC
PINATTR PinUse B
PINATTR PinNum 2
PIN B         A9
PINATTR PinUse B
PINATTR PinNum 1
INSTANCE EDGE I_19
SYMATTR RefName E1
PIN CONNECT   A7
PINATTR PinNum 3
INSTANCE EDGE I_18
SYMATTR RefName E1
PIN CONNECT   A6
PINATTR PinNum 4
INSTANCE EDGE I_17

SYMATTR RefName E1
PIN CONNECT   A5
PINATTR PinNum 5
INSTANCE EDGE I_16
SYMATTR RefName E1
PIN CONNECT   A4

PINATTR PinNum 6
INSTANCE EDGE I_15
SYMATTR RefName E1
PIN CONNECT   A3
PINATTR PinNum 7
INSTANCE EDGE I_14

SYMATTR RefName E1
PIN CONNECT   A2
```

```
PINATTR PinNum 8
INSTANCE EDGE I_30
SYMATTR RefName E1
PIN CONNECT   A8
PINATTR PinNum 2
INSTANCE EDGEOUT I_31
SYMATTR RefName E1
PIN CONNECT   A9
PINATTR PinUse O
PINATTR PinNum 1
NET N_9
NET N_8
NET N_7
NET GND
NET A9
NET VCC
NET N_6
NET N_5
NET N_4
NET N_3
NET N_2
NET N_1
NET N_10
NET A5
NET A6
NET A7
NET A8
NET A4
NET A3
NET A2
```

The Packlist Bill-of-Materials Program

The **packlist** bill-of-materials program writes a text file that includes information about the gates and components of a PCB design, in a format suitable for reading by a database or spreadsheet program. The program lists packages (single components or sets of gates with the same reference designator) rather than single symbol instances. **packlist** is normally run after the design has been packaged (that is, after all gates have been assigned a reference designator).

The output of the program consists of a header line followed by a line of attribute values for each package or set of packages in the design. The lines are sorted based on the attribute values. If two or more packages have the same attribute values, they are combined into a single line in the listing. Optionally, the number of gates or packages represented by each line may be listed. The fields of attribute values on each line are usually separated by a tab, since most spreadsheet programs can read files which have fields separated by tab characters. Optionally, the fields can be formatted into columns separated by spaces.

What Packlist Does

packlist performs the following operations in the order listed:

1. Checks for packaging errors and reports them.
2. Prints a header line. The default for the header line is a list of the names of the attributes which are to be listed. You can override the default header with the “header” entry in **packlist.ini**.
3. Gathers the attribute data for each package in the design. The default set of attributes gathered is part number, symbol type (component name), reference designator, value, and part shape (geometry). You can override the default set of attributes with the “list” entry in **packlist.ini**.
4. Sorts the packages in the design according to their attribute data. The default priority for sorting of attributes is part number, value, and reference designator. You can override the default sorting priority with the “sort” entry in **packlist.ini**.

   The part number or symbol type is usually the most important sorting attribute. The value attribute assures that all resistors and capacitors are listed in order. If two or more packages have the same value for all of the attributes in the sort list, they are combined into one line of output. Therefore, if the value and reference designator are omitted from the sort list, only one line will be written for each type of package.

5. Writes one line for each set of packages in the design. Each line contains a list of the attribute values for the package (or set of packages) on that line. The set of symbols represented by a line consists of the set of packages that have the same value for each of the attributes in the sort list.

   By default, each attribute printed is separated from the preceding attribute with a tab. You can specify the number of spaces to use (instead of a tab) with the "spacing" entry in the **packlist.ini** file.

Command Line Options

The following command line switches can be used to modify **packlist**'s behavior.

Switch	Effect
-ext=.xyz	Sets the extension of the output file to **.xyz**. This overrides the default extension of **.bom**.
-view	Displays the output file up in the Viewer when complete.
-head	Prints two lines with date and program version.
-section=*sec_name*	Reads configuration information from the [*sec_name*] section of **packlist.ini**. The default section name is [Default]. This parameter is ignored if there is no **packlist.ini** in the **config** directory or the **windows** directory. The **packlist.ini** file is described in the next section.

The packlist.ini File

packlist.ini uses the same format as most other **.ini** files used in Windows. The file is divided into sections marked by section names in square brackets. Each section contains entries in the following format:

```
[SectionName]
entry=value
entry=value
entry=value
   …
```

Lines beginning with a semicolon (;) are treated as comments and ignored.

If **packlist.ini** is called with a command line argument of the form:section=*name*, the section with that *name* will be read to obtain the configuration information. If there is no such section, or the -section option is not used, the [Default] section is used.

packlist.ini Entries

Each section of **packlist.ini** contains the following entries.

list	Lists the attribute names (or attribute numbers) to be printed on each line. The names are delimited by spaces. In addition to attribute names (or numbers), list can contain other special identifiers.
#line	Prints a line number.
#gate	Prints how many different symbol instances (or gates) are represented by this line in the bill of materials.
#pack	Prints how many different packages are represented by this line in the bill of materials. Two different symbol instances are considered part of the same package if they have the same reference designator and component name (symbol name).
#unused	Prints a count of the unused gates in the set of packages listed. In the case of packages with non-homogeneous gates, the count of unused gates does not include sections which are entirely unused. For example, consider a package with three 2-input NAND gates and two 3-input NAND gates. If there are two 2-input gates in the design, the bill of materials will list one unused gate. But if there are two 2-input gates and one 3-input gate, the bill of materials lists two unused gates. To display the location of a symbol in sheet and zone format, use attribute #253 (the location attribute).
sort	Lists the attribute names (or numbers) to be used for sorting the packages used in the design. The first attribute specified is the most important for the sort, the next attribute is the next most important, and so on. If two packages have the same value for all the attributes in the sort order, they are listed on the same line. If an attribute is displayed but not used in the sort, the value listed will be the value from the first set of packages represented by the line. Other packages on the same line may have different values for the attribute.

header	The header entry is a single line that is printed at the beginning of the file. The only thing special about this line is that a backslash character followed by a lower case "t" (\t) is printed as a tab character. If there is no header line in **packlist.ini**, the header consists of the attribute names from the list entry. These attribute names are separated by spaces or tabs.
spacing	The spacing entry is the width (in characters) of each column of output. If spacing is 0 (the default value), the columns are separated by tabs instead of spaces. If spacing is a negative number, the text in the columns is left-justified instead of right-justified.

Example 1

When there is no **packlist.ini** configuration file, the default is:

```
[Default]
;list is "line number" PartNum CompName RefName Value PartShape
"gate count"
list=#line 4 6 2 3 5 #gate
;sort order is PartNum CompName Value RefName
sort=4 6 3 2
header=
spacing=
```

Sample Output

```
packlist 1.0 - Engineering Capture System 2.6
xxx.bom - 8/25/93 0:58:39 AM
#Line   PartNum   CompNam   RefName   Value   PartShape #Gates
0                 CAP1      C1        1                 2
1                 CAP1      C2        1                 1
2                 CAP1      C1        2                 2
3                 COMP1     U1                          1
4                 COMP1     U2                          1
5                 GATE1     U4                          4
6                 GATE1     U6                          2
7                 GATE2     U3                          4
8                 GATE2     U5                          3
9                 RES1      R1        1                 4
10                RES1      R2        1                 1
11                RES1      R2        2                 2
```

Example 2

This example produces a list of unique packages and identifies the number of packages of each type. If **packlist.ini** has the [Custom] section shown below, the command line would be

```
packlist -head -section=Custom.
```

```
[Custom] list= #Line PartNum Value RefName CompName PartShape
#pack
sort=PartNum CompName Value
header=#Line  PartNum  Value  RefName  CompName  PartShape  #Packages
spacing=-10
```

Sample Output:

```
packlist 1.0 - Engineering Capture System 2.6
xxx.bom - 8/25/93 1:06:42 AM
#Line  PartNum  Value  RefName  CompName  PartShape  #Packages
0               1      C1       CAP1                2
1               2      C1       CAP1                1
2                      U1       COMP1               2
3                      U4       GATE1               2
4                      U3       GATE2               2
5               1      R1       RES1                2
6               2      R2       RES1                1
```

Appendix A
Generic Interfaces

This chapter explains the generic interfaces that are currently supported for the Synario Capture System (SCS). The following interfaces and topics are covered in this chapter:

- Archive Utility
- ASCII
- EDIF
- Generic Netlists

Archive Utility

The Archive utility gathers all the symbols and schematics needed in a design and places them in a single directory. It is typically used for archiving designs or for transferring designs to a different directory.

Archive is run from the Tools menu of the Hierarchy Navigator and must be added to the tools menu using the INI Editor.

Running the Archive utility without any command-line options does not copy the project files, but only writes a list of the files under the name *design***.lst** (where *design* is the base name of the project).

To actually copy the files, you must add the -save command-line option and specify the path, as shown below.

```
archive -save=path
```

The *path* must be complete and not contain any spaces.

The Archive utility copies or records only electrically significant symbols—Block, Cell, Component, Gate, and Pin symbols. Graphic and Master symbols are not copied or recorded.

ASCII Interface

The ASCII interface converts data in the SCS binary database to an ASCII format file you can edit with any text editor. The interface also converts an ASCII file (in the appropriate format) to the SCS binary format. You can interface with any CAE/CAD system *via* the SCS ASCII interface (though additional manual editing may be required in some cases).

Once a database has been converted to ASCII format, you can modify it and then translate it back to SCS format. This allows various filtering and processing programs to be used in the SCS environment.

The following ASCII interface conversion programs can be accessed from the SCS Executive Utilities menu:

- Schem to ASCII
- ASCII to Schem
- Symbol to ASCII
- ASCII to Symbol

ASCII / Symbol Interface

An SCS symbol file contains the information required to describe a symbol and its attributes. The database entries include graphic elements, fixed text, symbol attributes, attribute display windows, pins, and pin attributes.

SCS calculates an “extent” box for each symbol. This box is a rectangle that encloses all the points that define the various elements of the symbol. Unless you choose a different point, the upper-left corner of this box is the default origin for the symbol.

SCS uses a grid system for coordinates. The grid spacing is defined by the Grid Size and Grid Units parameters in the Sheet Layout dialog box of the INI Editor. The Primary grid is divided into a Secondary grid with four points in each major grid interval.

The Secondary grid is used for all coordinates in the symbol database. When the Symbol Editor stores the symbol data base it recalculates the extent box and adjusts the coordinates so that all coordinates are positive integers. The Symbol Editor also limits the size of a symbol to 400 Primary (or 1600 Secondary) grid units in each dimension.

The ASCII to Symbol and Symbol to ASCII conversion utilities are called from the Utilities menu of the SCS Executive.

Interface Format

The interface is a line-oriented ASCII text format. Each element in the symbol is represented as a single line in the text file. Each line consists of:

- A keyword that begins the line and identifies the type of element described.
- A list of parameters describing the element following the keyword.

The following conventions apply within each line:

- Keywords are not case-sensitive.
- Parameters are delimited by spaces and tabs.
- Text strings appear at the end of the line and can contain embedded blanks.
- Coordinate locations are given as pairs of numbers. X coordinates increase to the right and Y coordinates increase to the bottom of the symbol. All coordinate locations and distances are given in Secondary grid units.

In the following description of the format, keywords are shown in roman text and parameters are shown in italics.

`version` *`number`*

The SCS symbol database version number is specified on the first line of the file.

`symboltype` *`type`*

The symboltype should be the second line of the file. The *type* parameter is BLOCK, CELL, COMPONENT, GATE, GRAPHIC, MASTER, or PIN. If this parameter is not specified, the *type* defaults to BLOCK.

Each of the following formats for graphic elements contains a parameter that specifies line weight. This parameter can have the value NORMAL or WIDE. These keywords can be abbreviated as N or W.

On UNIX systems, the graphic-element format contains a parameter that specifies line style. This parameter can have the value Dash, Dot, or DashDot. If the parameter is missing, the line is drawn solid.

`line` *`width x1 y1 x2 y2 style`*

The *x-y* coordinates are the line's end points.

`rectangle` *`width x1 y1 x2 y2 style`*

The *x-y* coordinates are two opposite corners of the rectangle.

`circle` *`width x1 y1 x2 y2 style`*

The *x-y* coordinates are the opposite corners of a square, with the square enclosing the circle. If a rectangle is specified, it is converted to a square with the same upper-left corner as the rectangle. The sides of the square are the length of the shorter sides of the rectangle.

`arc` *`width x1 y1 x2 y2 x3 y3 x4 y4 style`*

The first two *x-y* coordinates describe a circle (see above) containing the arc. The third and fourth coordinate pairs are the starting and ending points of the arc. If these points are not on the circle, the lines joining these points to the circle's center set the boundaries of the arc. The arc is drawn counterclockwise from the starting to the ending point.

`text` *`x1 y1 justify font string`*

The *x-y* coordinate is the origin for drawing the text.

justify specifies the horizontal (vertical) justification of the text. It can have the value LEFT, RIGHT, CENTER, VLEFT, VRIGHT, or VCENTER. Any value can be abbreviated to just the first two characters. Horizontal (vertical) text is always justified vertically (horizontally) to the center of the character.

font specifies the size of the text and can be 0–7. (In the UNIX / Motif version, there are only three font sizes, 2, 3, and 4. DOS sizes 0–2 are mapped to UNIX size 2, DOS size 3 is mapped to UNIX size 3, and DOS sizes 4–7 are mapped to UNIX size 4.)

string is the text string. The string consists of the first non-blank character after the *font* parameter and all following characters.

`symattr` *`name string`*

name specifies the attribute and must match one of the attribute names listed in the Symbol Attributes section of the INI file.

string is the attribute value assigned, including any attribute modifiers.

`window` *`number x1 y1 justify font`*

number specifies which attribute window is to be drawn.

The *x-y* coordinate is the origin for drawing the attribute value.

justify specifies the horizontal (vertical) justification of the text. It can have the value LEFT, RIGHT, CENTER, VLEFT, VRIGHT, or VCENTER. Any value can be abbreviated to just the first two characters. Horizontal (vertical) text is always justified vertically (horizontally) to the center of the character.

font specifies the size of the text and can be 0–7. (In the UNIX / Motif version, there are only three font sizes, 0–2. Sizes 4–7 are mapped to UNIX size 2, 3 is mapped to size 1, and 0–2 are mapped to size 0.)

`pin` *`x1 y1 justify offset`*

The *x-y* coordinate is the location of the pin. This location must fall on a Primary grid point.

The *justify* and *offset* parameters indicate where to draw the name of the pin. *offset* is the distance from the pin, and can range from 0 to 31 Secondary grids. *justify* indicates the direction the pin name is offset from the pin and can be LEFT, RIGHT, BOTTOM, TOP, VLEFT, VRIGHT, VBOTTOM, VTOP, or NONE. (Use NONE when you are not giving the pin a name.) Any of these can be abbreviated to just the first two characters.

`pinattr` *`name string`*

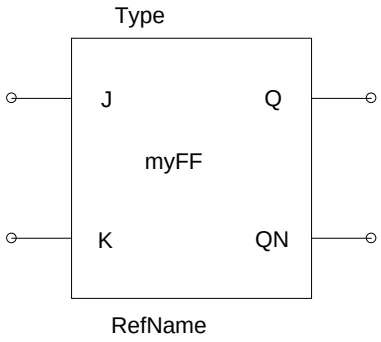
Pin attributes apply to the previously defined pin. *name* is the attribute and must match one of the attribute names in the Pin Attributes section of the INI file.

string is the attribute value assigned, including any attribute modifiers.

Example ASCII Symbol File

The following example is the result of translating the symbol in Figure A-1 to the ASCII format.

Figure A-1
Symbol to be Translated to ASCII




```
Version 3
SymbolType COMPONENT
Line N  8 8  0 8
Line N  8 28  0 28
Rectangle N  8 0  40 36
Line N  40 8  48 8
Line N  40 28  48 28
Text 25 17  CENTER 0 myFF
SymAttr SILOSModel .JKFF
Pin 0 8  LEFT 9
PinAttr PinName J
PinAttr Polarity IN
Pin 0 28  LEFT 9
PinAttr PinName K
PinAttr Polarity IN
Pin 48 28  RIGHT 9
PinAttr PinName QN
PinAttr Polarity OUT
Pin 48 8  RIGHT 9
PinAttr PinName Q
PinAttr Polarity OUT
Window 0  24 44  CENTER 0
Window 1  24 -8  CENTER 0
```

ASCII / Schematic Interface

The schematic database is a compact binary file that is created and modified by the Schematic Editor. The ASCII Schematic Interface allows you to translate the SCS schematic database to and from a format that can be modified by a text editor or other program. This allows conversion of schematics from one system to another.

An SCS schematic file contains the information required to describe a schematic including all of its sheets. The database entries include:

- Wires
- Buses
- Net name flags
- Net attributes
- Bus taps
- Symbol instances (both primitive and hierarchical)
- Symbol attribute overrides
- Pin attribute overrides
- Cosmetic graphics
- Fixed text
- Tables

A schematic file has one or more *sheets*, each representing a separate page of the printed schematic. Each sheet is divided into grids.

The grid spacing is defined by the Grid Size and Grid Units parameters in the Sheet Layout dialog box of the INI Editor. The Primary grid is divided into a Secondary grid with four points in each major grid interval. All connectivity elements (wires, symbols, pins, and attributes) must lie on the Primary grid. Cosmetic graphics and fixed text can lie on either the Primary or Secondary grid.

The Secondary grid is used for all coordinates in the schematic database. However, all the connectivity elements have coordinates that are integral multiples of four, which forces them to fall on the Primary grid.

The size of a sheet in a schematic is determined when the sheet is created. For example, if the units are inches and the grid is 0.1, a sheet 22 inches wide is (22 / 0.1) or 220 Primary grids wide. In Secondary units, the sheet is 880 grids wide.

The ASCII to Schematic and Schematic to ASCII conversion utilities are called from the Utilities menu of the SCS Executive.

Interface Format

The interface is a line-oriented ASCII text format. Each element in the schematic is represented as a single line in the text file. Each line consists of:

- A keyword that begins the line and identifies the type of element described.
- A list of parameters describing the element following the keyword.

The following conventions apply within each line:

- Keywords are not case-sensitive.
- Parameters are delimited by spaces and tabs.
- Text strings appear at the end of the line and can contain embedded blanks.
- Coordinate locations are given as pairs of numbers. X coordinates increase to the right and Y coordinates increase to the bottom of the symbol. All coordinate locations and distances are given in Secondary grid units.

In the following description of the format, keywords are shown in roman text and parameters are shown in italics.

SCS Database Version

`version` *`number`*

The SCS symbol database version number is specified on the first line of the file.

Sheet Size

`sheet` *`number width height`*

The sheet entry should be the second line of the file. *number* is the sheet number and can range from 1 to 99. The *width* and *height* are the dimensions of the sheet description that follows. All other lines following the sheet entry are entities which appear on that sheet. Additional sheet entries represent the beginnings of new sheets.

Net Position (Wire)

`wire` *x1 y1 x2 y2*

The *x-y* coordinates are the wire's end points. These coordinates must lie on a major grid. These coordinates must also satisfy the interconnection constraints for wires. For example, they must be lines at 45 or 90 degree angles to the edge of the sheet.

Flag Position

`flag` *x1 y1 name*

The *x-y* coordinate is the location of the flag. This coordinate must lie on the Primary grid.

name is the net name to be displayed. A wire cannot display two different names. The *name* consists of the first non-blank character after *y1* and all following characters. The name must conform to the rules for net names. A net name cannot contain spaces.

Pin Position and Definition

`IOpin` *x1 y1 in_out*

The *x-y* coordinate is the location of the pin. This coordinate must lie on the Primary grid. *in_out* specifies the polarity and can be IN, OUT, or BIDIR. The first character can be used as an abbreviation.

Bus Tap Position

`bustap` *x1 y1 x2 y2*

The *x-y* coordinates are the end points of the wire forming a bus tap. These coordinates must lie on the Primary grid. The bus tap must be at a 90 degree angle to the bus and can be either horizontal or vertical. It must also satisfy the interconnection constraints. These constraints are listed on page 5-20 of "Using the Schematic Editor."

Symbol Position and Format

`symbol` *`name x1 y1 rot_mir`*

name is the name of the symbol. Since it is the name of a symbol file, it must conform to the file name conventions for SCS.

The *x-y* coordinate is the location of the symbol origin.

rot_mir specifies the rotation and mirroring applied to this symbol instance. *rot_mir* can have values of R0, R90, R180, R270, M0, M90, M180, or M270. The first two characters can be used as an abbreviation.

The R values represent the rotated positions of the object. The M values represent the rotated positions of the object after it has been mirrored left-right. The list below describes the orientation:

***rot_mir* Value**	**Orientation**
R0	Default orientation, no rotation
R90	Default orientation, rotated 90° counter-clockwise
R180	Default orientation, rotated 180° counter-clockwise
R270	Default orientation, rotated 270° counter-clockwise
M0	Mirrored orientation, no rotation
M90	Mirrored orientation, rotated 90° counter-clockwise
M180	Mirrored orientation, rotated 180° counter-clockwise
M270	Mirrored orientation, rotated 270° counter-clockwise

`symattr` *`name string`*

The symbol instance attributes apply to the previously defined symbol instance.

name specifies the attribute and must match one of the attribute names listed in the Symbol Attributes section of the INI file.

string is the attribute value to be assigned. The *string* is the first non-blank character following the name field and all following characters.

Attribute Window Position

`window` *`number x1 y1 justify font`*

The attribute window location applies to the previously defined symbol instance. It overrides the window location in the symbol definition.

number specifies which attribute window is to be drawn.

The *x-y* coordinate is the origin for drawing the attribute value.

justify specifies the horizontal (vertical) justification of the text. It can have the value LEFT, RIGHT, CENTER, VLEFT, VRIGHT, or VCENTER. Any value can be abbreviated to just the first two characters. Horizontal (vertical) text is always justified vertically (horizontally) to the center of the character.

font specifies the size of the text and can be 0–7. (In the UNIX / Motif version, there are only three font sizes, 0–2. Sizes 4–7 are mapped to UNIX size 2, 3 is mapped to size 1, and 0–2 are mapped to size 0.)

Pin Attribute Position

`pinattr` *`x1 y1 name string`*

The *x-y* coordinate is the location of the pin. The specified pin attribute applies to the pin at this location.

name specifies the attribute and must match one of the attribute names listed in the Pin Attributes section of the INI file.

string is the attribute value to be assigned. The *string* is the first non-blank character following the name field and all following characters.

Net Attribute Position

`netattr` *`x1 y1 name string`*

The *x-y* coordinate is the location of a point on the net. The specified net attribute applies to the net at this location.

name specifies the attribute and must match one of the attribute names listed in the Net Attributes section of the INI file.

string is the attribute value to be assigned. The *string* is the first non-blank character following the name field and all following characters.

Each of the following formats for graphic elements contains a parameter that specifies line weight. This parameter can have the value NORMAL or WIDE. These keywords can be abbreviated as N or W.

On UNIX systems, the graphic-element format contains a parameter that specifies line style. This parameter can have the value Dash, Dot, or DashDot. If the parameter is missing, the line is drawn solid.

Lines

`line` *`width x1 y1 x2 y2 style`*

The *x-y* coordinates are the line's end points.

Rectangles

`rectangle` *`width x1 y1 x2 y2 style`*

The *x-y* coordinates are two opposite corners of the rectangle.

Circles

`circle` *`width x1 y1 x2 y2 style`*

The *x-y* coordinates are the opposite corners of a square that encloses the circle. If a rectangle is specified, it is converted to a square with the same upper-left corner as the rectangle. The sides of the square are the length of the shorter sides of the rectangle.

Arcs

`arc` *`width x1 y1 x2 y2 x3 y3 x4 y4 style`*

The first two *x-y* coordinates describe a circle (see above) containing the arc. The third and fourth coordinate pairs are the starting and ending points of the arc. If these points are not on the circle, the lines joining these points to the circle's center set the boundaries of the arc. The arc is drawn counterclockwise from the starting to the ending point.

Table Text Position

`text` *`x1 y1 justify font string`*

The *x-y* coordinate is the origin for drawing the text.

justify specifies the horizontal (vertical) justification of the text. It can have the value LEFT, RIGHT, CENTER, VLEFT, VRIGHT, or VCENTER. Any value can be abbreviated to just the first two characters. Horizontal (vertical) text is always justified vertically (horizontally) to the center of the character.

font specifies the size of the text and can be 0–7. (In the UNIX / Motif version, there are only three font sizes, 0–2. Sizes 4–7 are mapped to UNIX size 2, 3 is mapped to size 1, and 0–2 are mapped to size 0.)

string is the text string. The string consists of the first non-blank character after the *font* parameter and all following characters.

```
table x0 y0 rows cols row_height col_width
first_row_height first_col_width
name_just name_font title_just title_font
upper_left_just upper_left_font first_row_just first_row_font
first_col_just first_col_font rest_of_table_just
rest_of_table_font
```

The justification referred to below can have the following values:

```
BottomLeft
BottomCenter
BottomRight
CenterLeft
CenterCenter
CenterRight
TopLeft
TopCenter
TopRight
None
```

'None' is used if you don't want the title or name to appear on the table.

The name and title are drawn outside the table. When the Bottom options are used to justify the table's name or title, the table's name is printed above the table. The Top options prints the table's name below the table.

Font Parameters

The font parameters referred to below specify the text size and can be 0, 1, or 2. This corresponds to five, seven, or nine Secondary grid units in height.

Parameter	Description
x0 y0	The *x-y* coordinate is the upper-left corner of the table.
rows	Number of rows
cols	Number of columns
row_height	Height in Secondary grid units of all rows except the first

col_width	Width in Secondary grid units of all columns except the first
first_row_height	Height in Secondary grid units of the first row
first_col_width	Width in Secondary grid units of the first column
title_just	Justification for the table's title
title_font	Font size for the table's title
name_just	Justification for the table's name
name_font	Font size for the table's name
upper_left_just	Justification for the text placed in the upper-left cell
upper_left_font	Font size for the text placed in the upper-left cell
first_row_just	Justification for the table's first row (other than the first entry, covered by *upper_left_just*)
first_row_font	Font size for the table's first row (other than the first entry, covered by *upper_left_font*)
first_col_just	Justification for the table's first column (other than the first entry, covered by *upper_left_just*)
name_font	Font size for the table's first column (other than the first entry, covered by *upper_left_font*)
rest_of_table_just	Justification for all other cells, including all cells to the bottom right
rest_of_table_font	Font size for all other cells, including all cells to the bottom right

Table Name and Title Attributes

`tableattr` *`number value`*

The specified table attribute applies to the previously defined table.

number is the table attribute number. There are two choices:

0	Table name attribute
1	Table title attribute

where *value* is the value of the table attribute.

`tabledata` *`row col value`*

The table data refers to the previously defined table.

row and *col* specify the element in the table for which data is being defined. *value* is the data.

EDIF Interfaces

This section describes the Electronic Design Interchange Format (EDIF) interface for SCS symbols and netlists. EDIF is an standard format for ASCII data that allows design information to be exchanged among different CAD/CAE systems. SCS uses a subset of this format as an ASCII interface to its symbol libraries.

SCS can read and write EDIF symbol descriptions, and write EDIF netlists. Third-party interfaces are available to read and write EDIF schematic descriptions.

The EDIF standard is broad and includes many data formats not used in SCS. As a result, not all EDIF files translate correctly into SCS symbol files.

EDIF Netlist Interface

An EDIF netlist format is available in the Processes menu of the Hierarchy Navigator. To add this EDIF netlister to the Hierarchy Navigator menu, use the Navigator Tools dialog box in the INI Editor to create an entry for the **edifnet** or **edifnets** program.

The following command line options are available:

/N	Use Notepad to view netlist immediately after it is written.
/E	Use EXTERNAL statement instead of LIBRARY statement on primitives.
/S	Reduce whitespace in netlist. Makes netlist smaller but less readable.

The **edifnet** program runs from within the Hierarchy Navigator and creates a hierarchical netlist. The **edifnets** program can output only a single level of the hierarchy to a flat EDIF netlist.

Translating SCS Symbols To EDIF Files

To translate existing SCS symbols to EDIF files:

1. Select the Write EDIF command from the SCS Executive Utilities menu.
2. Click the Symbol to EDIF radio button.
3. Change directories within the SCS Executive to the directory containing the symbol files to be translated. The symbols in each directory are shown in the list box.
4. Double-click on the symbols in the list box to be converted. If you want to translate all the symbol files in the list box, click the ALL button. EDIF files with the extension **.edf** are created in this directory.

Translating EDIF Files to SCS Symbol Files

To translate EDIF files to SCS symbols:

1. Select the Read EDIF command from the SCS Executive Utilities menu.
2. EDIF files having an extension of **.edf** are displayed in the list box. If the current directory is not the one you want, click on a directory name or the double dot [..] to change directories.
3. Double-click on the EDIF files in the list box to be converted. If you want to translate all the symbol files in the list box, click the ALL button. SCS symbol files are created in this directory having the file extension **.sym**.

Because EDIF files can contain more than one symbol, the file name of each symbol is the name of the cell in the EDIF file with the extension **.sym**. Errors or warnings are written to an error file named *symbol***.err**, where *symbol* is the name of the EDIF file being translated. Errors are flagged with the line number on which the error occurred.

EDIF File Format

The EDIF format is briefly described here. For a complete description of the EDIF standard refer to the Electronic Industries Association publication, Electronic Design Interchange Format Version 2 0 0 (EIA Interim Standard No. 44).

EDIF statements have the following format:

(*keyword* {*form*})

A *form* is a sequence of identifiers, primitive data, symbolic constants, or other EDIF statements.

keyword	EDIF keywords are not case-sensitive.
identifier	An identifier is the name of an object or data group. Identifiers are used for name definition, name references, and symbolic constants. EDIF identifiers consist of alphanumeric or underscore characters and must be preceded by an ampersand (**&**) if the first character is not alphabetic. The ampersand is not considered part of the name. Identifiers can be up to 255 characters. Case is not significant.
number	Numbers are 32-bit signed integers. Real numbers can be represented by a *scaled integer* EDIF statement. For example, the number 1.4 is represented as (e 14 -1). The e form requires a mantissa and an exponent. The resulting real number is restricted to the range of $\pm$ 1.0 x $10^{\pm 35}$.
string	EDIF strings consist of one or more ASCII characters enclosed in quotes (" "). All ASCII characters are legal. Because quotes and percent signs (**%**) are EDIF delimiters, they must be entered as an escape sequence of the form %*nn*%, where *nn* is the integer value of the ASCII character. For example, "quote %34% character" is the string “quote " character”. (The outer quotation marks are not part of the string; they just delimit it.) ASCII characters not on the keyboard are also entered this way.
whitespace	Blank, tab, linefeed, and carriage return characters (whitespace) are delimiters. Blanks and tabs are significant when they appear in strings.
symbolic constant	A symbolic constant is a predefined EDIF name. For example, LOWERLEFT is used to specify text justification.
point	The pt construct is used to specify coordinate locations. The pt keyword must be followed by the x- and y-locations. For example: (pt 100 200) is at x=100, y=200.

Scaling

Numbers must be scaled to values appropriate for the actually elements they represent. For example, coordinate locations and graphic descriptions (such as line width) are measured in units of distance and must be specified in meters. Each coordinate value is converted to a length in meters by applying the scale factor. Each EDIF library has a [Technology] section which has a required numberDefinition construct. The scale construct is used within numberDefinition to relate numeric values to physical units.

Renaming Object Names

Names express relationships between objects and reference external elements. The name of a figureGroup is only used within the EDIF file as a reference to the EDIF structure which defines the drawing characteristics. Other names, such as cell names or property names, are used by SCS.

Sometimes the external name of an object is not a valid EDIF identifier. When this happens, the rename construct can be used to create a valid EDIF identifier and preserve the external name. For example, the symbol **test$1.sym** could generate the following cell construct:

```
(cell (rename test_1 "test$1")
```

The above example shows that the EDIF string is used to contain the original name, and a new name, test_1, is created as an EDIF identifier.

Formatting EDIF Files

The translation from EDIF to SCS symbol files is not always perfect because of differences between SCS and other CAE systems. Keep the following points in mind when performing EDIF translations:

- The scale for unit distance must be defined correctly in the numberDefinition section of the technology block in each library.
- The SCS requires pins, symbols and nets to fall on the Primary grid. You may therefore have problems lining up symbol pins on the Primary grid points.
- The pathWidth, borderWidth, and textHeight should be defined in a figureGroup in the technology block.
- Since the UNIX / Motif version SCS has only three font sizes, text may not appear to have mapped correctly. (The Windows version has eight font sizes, so this is less likely to occur.)

- In order to correctly interpret symbol and pin attributes, SymbolType should be specified as a property of the cell.
- The cell name defines the name of the translated symbol.
- All pins should be defined using the port construct in the interface section of the view.
- Pin attributes must have definitions in the Pin Attributes section of the INI file.
- Symbol attributes should appear in the [Interface] section before the symbol constructs if there are any attribute display windows with that attribute.
- Symbol attributes must have definitions in the Symbol Attributes section of the INI file.
- Arcs should be carefully specified since there is a possibility of rounding errors in the conversion between a three point arc and a center-and-radius description of the arc.
- The world coordinates must be limited so that the symbol fits within the SCS symbol limit of 400 Primary grid units.
- SCS file names are limited to eight characters for DOS/Windows compatibility. In the UNIX/Motif version of SCS, case is significant.
- In displaying pin names, SCS places pin names from 1 to 15 quarter grids from the pin.
- SCS uses the SymbolType property to distinguish Block, Cell, Component, Gate, and Graphic symbols.
- SCS supports only two line widths.

- SCS uses system-wide specification of color for nets and pins. EDIF allows individual items to have their own colors.
- Attributes which are relevant in SCS may not be the properties relevant in other CAE systems.

Preparing Non-SCS EDIF Files for Translation

Before reading in EDIF files that were produced on another system, the following procedure should be followed:

1. Be sure the Grid Size setting in the Sheet Layout dialog box of the Controls menu of the INI Editor is set to match the grid setting from the source CAE system. Changing the Grid Size changes the size of the symbol, both when displayed and when printed.

2. Add Attribute definitions to the Symbol Attributes and Pin Attributes sections of the INI file to correspond to any attributes that were defined in the source CAE system but which are not currently defined by SCS.

   Check the spelling of the attribute names and change it to match the spelling of properties in the EDIF file. For example, PartNum in SCS might be spelled Part_No in the source CAE system. Once the EDIF files have been converted, you can change the spelling back if you wish. The spelling of attribute names is significant only during translation of files to ASCII or EDIF format.

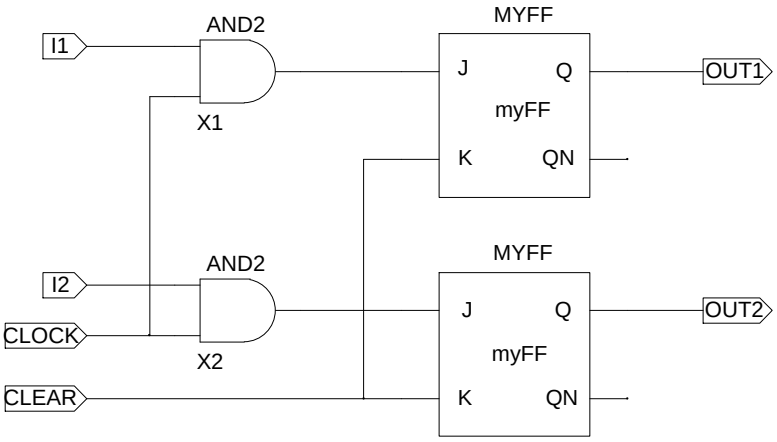
3. Check the cell names in the EDIF file to be sure the names do not conflict with the names of other files after truncation to eight characters. (File names are limited to eight characters under both Windows and UNIX/Motif.)

4. Determine the desired symbol type. The symbol can be a BLOCK, CELL, COMP, GATE, GRAPHIC, MASTER, or PIN. The symbol type is normally determined by having a property on the cell that specifies the type. If this property is not present, the symbols created during EDIF to SCS translation have the type specified by the default SymbolType in the Controls section of the INI file. If there is no SymbolType property in the EDIF file and no SymbolType default in the INI file, the symbol is created as a BLOCK.

Generic Netlists

Several generic netlisters are supplied with the SCS. The ASCII output of these generic netlists is usually in a form that can be easily modified for custom applications.

Figure A-2
Schematic for Netlist Examples



These netlisters are consolidated in a single executable file called **lister**. This program is accessed through the Processes menu of the Hierarchy Navigator. A specific netlister is specified by adding a command-line option to the **lister** entry:

Option	**Action**
-netorder	Net list by net
-pinorder	Net list by instance and pin
-listmark	List only marked nets and instances.
-listinst	List type, location, parent and instance names.
-listpart	List block symbols, count of primitive symbols.

The following additional options can be used with all the netlisters:

-nohead	Suppress printing of time and date header.

-pcb	Use reference designator and pin number instead of instance name and pin name.
-view	Run the currently specified editor to view the netlist file immediately after netlist creation.
-ext=.abc	Create a netlist file with the specified extension (*design*.abc). The specified extension overrides the default extension.

Netlist by Net (netorder)

The -netorder option generates a flat (non-hierarchical) ASCII netlist. Only primitive symbols and their pins are included in the listings. Buses are resolved into discrete signals and do not appear as buses in the listings.

The generated netlist is ordered by net. The listing is divided into two sections. The first section of the listing contains a list of the symbol instances showing the symbol type followed by the instance name. Each instance is on a separate line and instances of the same symbol type are not sorted into groups. A line of dashes ends this list.

The second section of the listing is a list of each net followed by a list of the pins that are contained in the net. The first entry in the line is the net name. The following entries are the pins connected to the net.

If more than one line is required to list a net, the line to be continued ends with an ampersand and the following line is indented.

The following example shows the Netlist format for Net List By Net. The circuit in Figure A-2 is the basis for this example.

```
JKFF    X3
JKFF    X4
AND2    X1
AND2    X2
--------------------
A2      X4-J    X2-OUT
OUT2    X4-Q
CLEAR   X4-K    X3-K
CLOCK   X1-IN2  X2-IN2
I2      X2-IN1
OUT1    X3-Q
A1      X3-J    X1-OUT
I1      X1-IN1
```

Netlist by Pin (pinorder)

The -pinorder option generates a flat (non-hierarchical) ASCII netlist. Only primitive symbols and their pins are included in the listings. Buses are resolved into discrete signals and do not appear as buses in the listings.

The output netlist is ordered by symbol instances. The two entries are symbol type followed by instance name. The entries that follow this represent the symbol pins. Each pin entry contains the name of the pin followed by the name of the net to which it is connected. Unconnected pins have the word “Unconnected” in place of the net name.

The symbol and each of its pins are listed on a separate line. The line with the symbol instance is not indented to differentiate it from the lines with the pins.

The following example illustrates the Netlist format for Net List By Pin. The circuit in Figure A-2 is the basis for this example.

```
JKFF X3
J    A1
K    CLEAR
Q    OUT1
QN   Unconnected
JKFF X4
J    A2
K    CLEAR
Q    OUT2
QN   Unconnected
AND2 X1
IN1  I1
IN2  CLOCK
OUT  A1
AND2 X2
IN1  I2
IN2  CLOCK
OUT  A2
```

Appendix B
Simulator Interfaces

This chapter explains the simulator interfaces that are currently supported for the Synario Capture System (SCS). The following interfaces and topics are covered in this chapter:

- SILOS
- Timewave History File Format
- Simulation Environment
- SPICE
- SPICE Format Conversions
- Timemill
- Verilog
- VHDL
- Export

SILOS

SCS provides an interface to the SILOS Logic Simulator. This interface lets you extract a netlist file for the SILOS simulator.

The SILOS Netlist Processor is selected from the Processes menu of the Hierarchy Navigator. This processor creates a netlist file with the name *design*.dat (where *design* is the base name of your design). This file is a topological description of the hierarchical design and is combined with a user-supplied pattern file that specifies the initial values, clock, and pattern definitions for the simulator. The pattern file is typically named *design*.pat. Any text editor can be used to create the pattern file.

If errors occur, the processor invokes the Notepad (or the text editor you specified in **ecs.ini**) on the netlist file. You can use the Notepad to scroll through the netlist file and identify the errors. Each error is on a separate line surrounded by asterisks. For example:

```
**** Missing Pin Named: EN ****
```

Adding a SILOS Menu Entry to the Hierarchy Navigator

The SILOS netlister is run as a process from the Hierarchy Navigator. You must use the INI Editor to add it to the Navigator's Processes menu.

1. Select the INI Editor from the SCS Executive.
2. Select the Processes command from the Tools menu.
3. Add the following menu entry:

   ```
   SILOS Netlist
   ```

4. Add the following command line entry:

   ```
   silosnet
   ```

Any of the following options can be added to the command line:

/A	Automatic mode; do not display Option dialog box.
/M	Write subcircuit netlist.
/P	Use lowest-level primitives.
/S	Invoke the SILOS simulator. (DOS only)

The netlister displays the Option dialog box if the /A option is not specified. If the /A option is specified the netlister does not display the Option dialog box but does use any other options specified on the command line. If other options are specified without the /A option the dialog box is displayed with the selected options as the defaults.

Required Attributes

Four attributes are reserved for use with the SILOS interface. They are defined in the INI file and provide the information needed to construct the network description.

SILOSModel

This symbol attribute identifies the primitive or macro used to represent the symbol.

Primitives are identified by the first three characters after a leading period (.). For example, .AND, .AND42, and .ANDU2 are seen as being the same primitive. "PDQ" is not a primitive, because it does not begin with a period.

Macros are specified by name without a leading period. Macros must be defined in the **systm.lib** or **macro.lib** library file. The remainder of the attribute (after the macro or primitive name) is copied into the network description without interpretation. Therefore, if a strength qualifier is desired on a primitive, it can be appended to the primitive name in this attribute.

SILOSTimes

This symbol attribute specifies the rise and fall times and multipliers for the gate. Rise time is the first field and fall time is the second field. The multipliers are optional components of each field. If present, this string is inserted into the network description between the SILOSModel and the pins.

SILOSName

This pin attribute is used for macros and several primitives to set the order in which the pins are listed in the netlist. For macros, pins are identified by number and appear in the netlist in the same order as the numbers in the SILOSName attribute.

For some primitives, the pins are identified by name (rather than order). If the PinName is different than the name required to identify the pin for SILOS, the SILOSName attribute can be used. The netlist processor checks the SILOSName before looking at the PinName.

If the first character in this attribute is a minus sign (**–**), the netlister places a leading minus sign in front of the net name connected to this pin. This treats the pin as if there were an inverter on the pin. The symbol should have an inverter bubble on this pin to indicate the inversion and avoid confusion.

SILOSLoad

This pin attribute indicates the optional numerical load factor used in computing the fanout-dependent delay load for all gates connected to this input pin. If present, this string is inserted into the network description between the SILOSModel and the pins.

Optional Attributes

Several other attributes are used by the SILOS interface:

Value

The Value attribute is used on **.cap** symbols to specify the capacitance.

Polarity

The polarity is used by the SILOS netlister for several primitive types to determine which pins are the output pins.

PinName

The pin name is used to identify pins for some primitives if the SILOSName attribute is not present.

These attributes are assigned to the symbol and its pins by use of the Symbol Attribute and Pin Attribute commands of the Symbol Editor. If a symbol has an attribute assigned, it becomes the default value for each instance of the symbol on the schematic. The default values can be overridden for a particular instance of the symbol on the schematic by use of the Add: Attribute command of the Schematic Editor or the Edit: Attribute command of the Hierarchy Navigator.

Creating the Pattern File

The pattern file is one of two files required by SILOS to perform simulations. It provides circuit stimuli, such as clocking signals and any other primary inputs. The pattern file also contains the location of the netlist file. You must create a pattern file for each simulation. The pattern file has the extension **.pat**.

The following is an example pattern file named **demo.pat** for use with a design called **demo.sch**. Text following a dollar sign (**$**) is a comment.

```
!INPUT DEMO.DAT                $ Causes SILOS to read the
netlist
VDD   .CLK 0 S1                $ Define the global power and
ground
GND   .CLK 0 S0
RESET .CLK 0 S1 160 S0         $ Specify all input signals
CLOCK .CLK 0 S0 100 S1 150 S0  .REP  0
```

Please refer to the SILOS *Reference Manual* for more information about clock and pattern inputs.

Controlling Netlist Extraction

The netlist file is one of two files required by SILOS. The netlist file contains the circuit's connectivity.

When the SILOS netlist processor is selected from the Processes menu of the Hierarchy Navigator, a dialog box offers the following options:

Netlist	Writes the netlist only.
Subcircuit	Writes a special version of the netlist for a sub-hierarchy when working with a split hierarchy.
Simulate	Invokes the SILOS Simulator after writing the netlist. (DOS only)

Select the required option and run the process by clicking the Run button.

In addition to the buttons, there is a check box labeled Use Primitives. When the netlist processor encounters a symbol that has the SILOSModel attribute specified, it normally codes that macro or primitive into the netlist without looking for any lower-level schematics. If the Use Primitives option is selected, the netlist processor ignores the SILOSModel on higher-level blocks and codes the netlist using only the lowest-level primitives. This feature makes it possible to switch between simulating at the functional level and the gate level without changing anything on the schematic.

For example, assume a schematic has a symbol called NAND2 and the SILOSModel attribute of this symbol is .NAND. An underlying schematic exists for the symbol NAND2 containing transistor symbols NMOS and PMOS (SILOSModel= .NMOS and .PMOS). The netlist is normally coded .NAND without any reference to the underlying schematic. With the Use Primitives option, NAND2 is coded as a .MACRO using statements containing .NMOS and .PMOS.

SILOS Format Conversions

The SILOS netlister makes the following conversions in the netlist file extracted from the SCS schematic. Keep these changes in mind when interpreting error messages from SILOS.

- The apostrophe (**'**) (that causes names to be written with overbars) is replaced with an underscore (**_**).
- SILOS uses the left parenthesis (**(**) as its hierarchical separator, rather than a period (**.**).

The SILOS netlister writes a hierarchical netlist. Therefore, any instance-specific attribute overrides, that are added by the Hierarchy Navigator, do not appear in the netlist.

Building SILOS Symbol Libraries

SILOS has many built in primitive functions that are used in simulation. Several attributes in these primitives must take on prescribed values. The following list gives the attributes needed for various SILOS primitives.

Combinational Gates

SILOSModel= .INV, .AND, .NAND, .OR, .NOR, .XOR, .XNOR, or .GAT*n*
Polarity= OUT on the output pin

Tri-State Combinatorial Gates

SILOSModel= .TINV, .TAND, .TNAND, .TOR, or .TNOR
Polarity= OUT on the output pin
SILOSName= begins with E on the enable pin
Name= begins with E on the enable pin

And-Or-Invert Gates

SILOSModel= .AOI, or .OAI
Polarity=OUT on the output pin
SILOSName= A, B, ... indicates the groups of pins

Level & Strength Delays

SILOSModel= .DLA, or .SDLA
Polarity=OUT on the output pin

Flip-Flops

SILOSModel=.DNEFF, .DPEFF, .JKNEFF, .JKPEFF, .SRNEFF, .SRPEFF, .TNEFF, or .TPEFF
SILOSName= or Name= one of CLK and Q on all types. CLR, SET and QB are optional on all types. J, K, S, R, and D as appropriate.

Transfer Gates

SILOSModel= .NXFR, .PXFR, .NRXFR, .PRXFR, .CXFR, or .CRXFR
SILOSName= or Name= IN, OUT, and EN and/or EP on enable(s)

CMOS Transistors

SILOSModel= .CMOS
SILOSName= or Name= NS, NGP, NGN, or ND

MOS transistors

SILOSModel= one of .NMOS .PMOS
SILOSName= or Name= NS, NG, ND

Zycad-compatible D Flip-Flop

SILOSModel= .DFF
SILOSName= or Name= Q, D, CLK, or R

Zycad-compatible Latch

SILOSModel= .LAT
SILOSName= or Name= Q, D, EN, R

Resistor

SILOSModel= .RES
Polarity= OUT on the output pin

Capacitor

SILOSModel= .CAP
Value= capacitance ratio (default is 1)

The capacitor symbol should have only one pin. If it has two pins, one must be connected to GND or VSS.

Setup & Hold Detectors

SILOSModel=.SHNE or .SHPE
SILOSName= or Name= CLK or D
Polarity=OUT on the output pin

RAM, ROM, and PLAs

These primitives are not implemented due to the wide range of possible combinations. You should manually code macros for these elements into one of the system library files and set the SILOSModel to the name of the macro.

Macros

SILOSModel= name of the macro
SILOSName= order for pins

Symbols for manually-coded elements require the pin output order to be specified. This is done by using the SILOSName attribute to indicate the numeric order of the pins. The macro definitions should be in either the **systm.lib** or **macro.lib** system library.

Examples

Two examples of SILOS elements are presented. A J-K flip-flop shows how to use a SILOS primitive. A four-bit inverter shows how to use a SILOS macro to build a non-standard function.

Negative Edge-Triggered J-K Flip-Flop

The following example is a negative edge-triggered J-K flip-flop with active-high set and clear. It has CMOS type strengths, a rising delay of 4ns, and a falling delay of 2ns. The special attributes necessary are listed.

Symbol Attributes

SILOSModel= .JKNEFF/C
SILOSTimes= 4 2

Pin Attributes

PinName	SILOSName	Polarity
J		In
K		In
CLK		In
SET	-SET	In
CLR	-CLR	In
Q		Out
QN	QB	Out

In this example, CMOS drive strength is shown by the /c in the SILOSModel definition. The rise and fall delays are shown under SILOSTimes.

The last pin has a PinName of QN, so it is displayed that way on the schematic. The SILOS netlister requires the pin name to be QB, so the SILOSName attribute (QB) is used in the netlist file.

The SET and CLR pins are to be treated as active low rather than the default active high. The minus sign (–) on the SILOSName causes the net connected to this pin to be listed with a leading minus sign, which indicates these are active-low pins.

Four-Bit Inverter

The following example is a macro for a four-bit inverter whose symbol uses bus pins:

Macro Definition

```
.macro INVERT4  DATA0 DATA1 DATA2 DATA3 ENABLE OUT0 OUT1 OUT2 OUT3
OUT3 .TINV ENABLE DATA3
OUT2 .TINV ENABLE DATA2
OUT1 .TINV ENABLE DATA1
OUT0 .TINV ENABLE DATA0
.eom
```

Symbol Attributes

SILOSModel= INVERT4

Pin Attributes

PinName	SILOSName	Polarity
OUT[0;3]	6	Out
ENABLE	5	In
DATA[0;3]	1	In

The numbering sequence for a bus pin should treat that pin as a sequence of individual numbers. In this example, DATA[0:3] is given a SILOSName of 1. Since there are four pins, the next usable number for a SILOSName is 5.

Error Messages

`No Model Defined for symbol`

A primitive symbol did not have the SILOSModel defined.

`Unknown Primitive Type: type`

The SILOSModel attribute is not one of the supported types.

`Missing Symbol for root_schem`

The Subcircuit option requires a symbol for the root schematic.

`Missing Subcircuit File filename`

If a symbol has a SILOSModel specified without a leading period (**.**), there must be a file with the given name and extension **.dat** in the current directory or one of the project or model directories in the search path. Load the schematic into the Hierarchy Navigator, then run the SILOS netlister with the Macro option to create this file.

`Missing Pin Named: name`

A primitive symbol was expected to have a pin with the given Name or SILOSName.

`SILOSName Not Specified for Pin in Symbol name`

Macros require that each pin have its SILOSName specified as a number indicating the numerical order for the pin to be listed. This symbol is a Macro because SILOSModel specified a name without a leading period.

`Incorrect Pin Ordering in macro`

This macro has a gap in the sequence of numbers in the SILOSName attribute. This symbol is a Macro because SILOSModel specified a name without a leading period.

`No Output Pin on name`

This primitive requires a pin with Polarity=Out

`Unconnected Output: output`

All primitives must have the outputs connected except for the QB pin of a Flip-Flop.

`Unconnected Capacitor Pin On: symbol`

The capacitor pin must be connected.

`Capacitance Not to GND or VSS`

If a capacitor symbol has two pins, one of the pins must be connected to GND or VSS.

`Missing or Invalid VALUE (Capacitance) for node`

The Value attribute must be set to the relative capacitance for this node.

Dynamic Simulation Interface

The SCS can include the SILOS Simulator in the design environment. The interface between SCS and SILOS consists of three parts:

Netlist Extraction SCS can extract the design netlist in SILOS input format. The netlist file has the extension **.dat**. Additional SILOS input, including the stimulus commands, is user-generated and placed in another file, typically with the extension **.pat**.

Simulator Control The SILOS Simulator can be launched and controlled from the Hierarchy Navigator. Commands can be defined and added to the Navigator's Simulate menu. The elements on which the commands act (such as nets) are selected by clicking on them in the schematic.

Output Viewing The output of the Simulator can be viewed graphically with the Waveform Viewer. The simulation results can be viewed as the simulation progresses (dynamic mode) or after the simulation is completed (static mode).

The Waveform Editor can run alone or in conjunction with the Hierarchy Navigator. In the latter case, the schematic can be back-annotated with the states of signals as defined in the Simulator.

Setup of the silos.ini File

The Hierarchy Navigator can be configured to control the SILOS Simulator. The first step is to specify the Simulator parameter as "silos" using the INI editor. This causes the Navigator to read **silos.ini**, that must be in either the working directory or in the ROOT**data** (ECS_ROOT/**data**) directory. A typical **silos.ini** file is shown below.

```
[SimTitle]
   title = SILOS

[SimTools]
   Code Verilog = #Make MakeVerilog
   Waveform Editor = wet -nav &B
   Edit silos.ini = notepad %config\silos.ini
   Start Silos = #Simulate Simulate1
   Start Silos Flat = #Simulate Simulate2
   Review Silos Results = waves -nav -silos &B

[MakeVerilog]
   Process = vericode -model &R
   Extension = .v
   Path =
   ModelAttribute = 20

[Simulate1]
   Command = silos3w
   Waves = waves -nav -silos &B
   SimAppName = silos3
   ExitMsg = quit
   FirstMsg = FirstSilos1
   Command = rsh host_name /home/bin4/silos &R.pat &R.dat
   Waves = waves -nav -silos -sim &B
   First = siloswav -sendbuf -wav@ \n in &R.pat \n in &R.dat \n sim 0

[Simulate2]
   Command = silos3w
   Waves =
   SimAppName = silos3
   ExitMsg = quit
   FirstMsg = FirstSilos2

[FirstSilos1]
   1 = RESET ALL
   2 = CON .CUSTREPORT .SYN=0 .SAV=2
   3 = LIB .\common{.v} .\symbus{.vi} .\top{.v} .{.v} c:\test\bliflib.v
   4 = File .SAV=&B
   5 = File .STO=&B.rep
   6 = Input &B.tf
   7 = Sim 0
```

```
[FirstSilos2]
   1 = RESET ALL
   2 = CON .CUSTREPORT .SYN=0 .SAV=2
   3 = LIB .\common{.v} .\symflat{.vi} .\top{.v} .{.v} c:\test\bliflib.v
   4 = File .SAV=&B
   5 = File .STO=&B.rep
   6 = Input &B.tf
   7 = Sim 0

[SimControls]
   MaxLine = 512
   Terminator =
   Separator = ,
   FlattenBusses = Yes
   HierChar = (
   RootPrefix = (t(m(
   RootIOPrefix = (t(
   InstPrefix = (t(m(
   SubPrefix = (t(m(
   GlobalPrefix =
   InstParen = _

[SimMenu]
   Set = SET %n = %{ 0 1 X Z }
   Force = FORCE %n = %{ 0 1 X Z }
   Release = FREE %*n
   Simulate to: = Sim to ?{}
   Stop = ^C
   Finish = QUIT

[SimSources]
   .abl =
   .v =
   .vf = vf\

[Export]
   exp -verilog -ext=.tf
```

The First entry under the [SimControls] section works as follows. The Navigator replaces the at sign (**@**) with the host number for the local host (typically,192.9.200.*nnn*) and the &R with the path name to the root schematic. It then sends the **siloswav** command to SILOS followed by the two **in** commands to read in the pattern file (**.pat**) and the netlist file (**.dat**). This is followed by the **sim** 0 command. (The individual command lines are separated with a “newline” character—a backslash followed by the letter n (**\n**).)

Note: *The paths to the netlist files must refer to the host running SILOS.*

Netlist Extraction

The SILOS netlist extraction program, **silosnet**, is accessed from either the Processes menu or the Simulate menu of the Hierarchy Navigator. In either case, the netlist of the hierarchical design has the root schematic's base name and the extension **.dat**.

The Code SILOS command in the Hierarchy Navigator is enabled by setting the Simulator option to "silos" using the INI editor. There must be a file named **silos.ini** in the working directory, or in the ROOT**data** directory (the ECS_ROOT/**data** directory in the UNIX/Motif version). The preceding section, "Setup of the silos.ini File" has a sample **silos.ini** file.

Running SILOS with the Hierarchy Navigator

After SILOS has been relinked and the **silos.ini** file set up, you can run SILOS from the Navigator using the following procedure:

1. Set the Simulator parameter in the INI file to "silos" using the INI editor.
2. Start the Navigator on the top level of your design. SILOS appears on the menu bar. If SILOS does not appear, the setup is not correct or the **silos.ini** file was not found in either the working directory or in the ROOT**data** (ECS_ROOT/**data**) directory.
3. Select the Code SILOS Model command in the SILOS menu. This extracts the netlist for the design.
4. Select the Start Simulator command in the SILOS menu. The Simulator Control window opens showing the Delta button. Below the button is a window displaying the commands:

```
siloswav
sim 0
```

The SILOS sign-on messages appear, then the SILOS prompt:

```
Ready:
```

If the -waves option was requested, the Waveform Viewer opens.

You are now prepared to run SILOS. The menu changes to show the commands specified in the **silos.ini** file. Use the Hierarchy Navigator menu and the control buttons on the Simulator window to control SILOS. Commands not in the menu can be entered directly into SILOS by clicking in the control window and typing the command.

Running SILOS Without the Hierarchy Navigator

SILOS can also be run without the Navigator by starting the Waveform Viewer, then SILOS, with the commands:

```
waves -sim -silos &
silos design.pat design.dat
```

After SILOS is launched, enter:

```
siloswav -sendbuf -wavaddr
sim 0
```

where *addr* is the Internet address of the host

The Waveform Viewer runs. Enter all SILOS commands from the keyboard.

Running SILOS in Static Mode

SILOS can also be run in static mode with or without the Hierarchy Navigator. When SILOS completes, it creates a history file named **save.sim**. Use the Waveform Viewer to view the simulation results. This feature lets you run a long simulation unattended.

If you want to use the Hierarchy Navigator, add an entry to the Navigator Tools section of the INI Editor that specifies the command to launch the Waveform Viewer and access the history file:

```
waves -nav -silos
```

If the Waveform Viewer is to be run by itself, enter the command:

```
waves -silos
```

Timewave History File Format

The Timewave history file format was designed to provide an ASCII file format passing data from the Simulator to the Waveform Viewer. The history file is usually used to store simulation results for later analysis using the Waveform Tools.

The format supports times from zero to two billion, with four different simulation states for up to 65,000 nodes.

The file is stored with the same base name as the netlist filename, but with the extension .his. The file has four sections: the preamble, the header, the node names, and the node transitions. The first section can be omitted.

Preamble

The preamble contains comments and/or date information. Lines in this section can begin with any non-digit character. All lines (including blanks) are skipped until a line beginning with a digit is encountered.

Header

This is a single line containing three integer values and terminated with a new-line (\n) character. The format is

decimals end_time node_count

The values are

decimals	Number of decimal places in time values (0, 1, or 2).
end_time	Last simulation time.
node_count	Number of nodes that are monitored.

Node Names

This section is a single line that follows the header containing the node names. The first name represents node number 0 in the state data.

- Names are separated by spaces
- Names are limited to 256 characters each

- The number of net names is limited to 65535.
- The Node Names section should be terminated with 3 new-line characters (\n).

Node Transitions

This section follows the node names and specifies the node transitions. For each time when at least one node changes, there is one record which lists all the nodes that changed. The format of the record is the time followed by one or more pairs of node number and state values. Each record is terminated with a new-line (\n) character. The first record should be at time zero, and every node should be listed with its initial value. After the last time any signal changes, there should not be any more events. The following rules apply to node transition records:

- All fields in a record are delimited with spaces.
- Nodes are indexed from zero.
- Buses are referenced by their constituent signals. The bus Data [0:7] is treated as 8 separate signals (Data [0], Data [1] …).
- The legal state values are:

0	Logic low
1	Logic high
2	Unknown
3	High-Z

Example History File

This example is a 2-input AND gate. The simulation runs from 0 to 512. The nodes are GND, VDD, IN1, IN2, and OUT. The gate has a rise time of 2 and a fall time of 1. The inputs are 0, 0 at time 0 and change at time 138, 256, and 384.

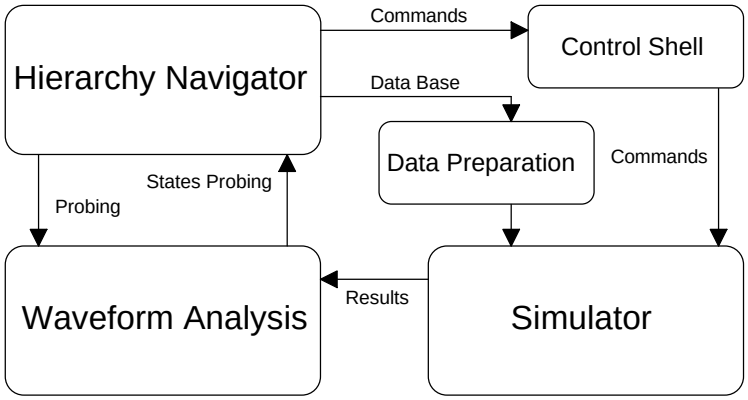
```
This file is an example history file
and this is the Preamble section.
0 512 5
GND VDD IN1 IN2 OUT


0 0 0 1 1 2 0 3 0 4 2 1 4 0
1 4 0
128 2 1
256 3 1
258 4 1
384 2 0
385 4 0
```

Simulation Environment

The Simulation Environment is a generic interface for running a simulator within the SCS environment. The operational model consists of five functions that can be interconnected, as shown in Figure B-1. Ideally, the environment should include all the functions. You can, however, omit one or more functions while retaining the interaction among the remaining modules.

Figure B-1
Operational Model of SCS / Simulation Environment



The Hierarchy Navigator is the parent process. You begin your work session by invoking the Navigator to view your design. The Navigator combines the symbols and schematics to create a hierarchical data structure, which is displayed in the Navigator window. The Navigator also provides a Simulate menu with the following options:

Model	Action
Code Simulation Models	The Hierarchy Navigator runs the Data Preparation module to extract the model information from the database.
Compile Model	The Hierarchy Navigator runs the process that compiles the models according to the simulator's requirements. This process can be omitted for some simulator environments.
Start Simulator	You can run the simulator only after you have run the two preceding Model options.

You can add additional options to the simulator initialization file.

Once the simulation is running, you send commands to the simulator through the Simulate menu in the Hierarchy Navigator. In the UNIX/Motif version you can also click buttons in the control shell or type commands in the control shell's main window. These commands enter the simulator as its standard input (stdin) as if they were typed into a normal operating system command window. You should not need to modify the simulator to accommodate this portion of the interface.

The Hierarchy Navigator can also invoke the Waveform Viewer. When run, the Viewer opens a socket and awaits the connection of the Simulator. The Simulator can then connect to the Viewer, using a routine that is supplied with the (optional) Programmer's Interface Kit.

When connected, the Waveform Viewer displays the simulation results. It also passes the results (at the query cursor) to the Hierarchy Navigator for display of net state values. All other functions of the Viewer and the Navigator, such as cross-probing, are also available.

When all modules are operational and linked, the process is controlled through user interactions with the Hierarchy Navigator, the Waveform Viewer, and the SCS Shell.

A subset of the environment can also be used. Simulators with their own waveform-viewing capabilities can omit the Waveform Viewer. If you omit the Waveform Viewer, however, cross probing and display of simulation results on the schematic are not available. When long simulations are run in a batch environment, the Navigator/Waveform Viewer combination provides a powerful tool for viewing the results of the simulation from a history file.

Simulation Environment Setup

The simulation environment is defined in a separate configuration file. This lets you define an interface for any simulator without modifying the SCS programs.

The **ecs.ini** file contains a Simulator parameter (set in the System Controls dialog box of the INI Editor), which is the base name of this configuration file. The **.ini** extension is added to the Simulator parameter to form the complete file name. (The Simulator name is forced to lower case in the UNIX/Motif version, and is case-insensitive in the Windows version.) The system expects to find this *simulator*.ini control file in the working directory, or in the directory specified by the CONFIG variable in the Windows Registration Editor, or (in the UNIX/Motif version) in the ECS_ROOT/**data** directory.

Each section of the simulator configuration file has an identifying header within square brackets. All lines following the header, up to the end of the file or another header, are part of that section.

The commands in each section are described below.

[SimTitle]

```
Title = simulator name
```

Adds the simulator's name to the menu bar of the Hierarchy Navigator. If this section is omitted, the default name "Simulator" is displayed.

[SimTools]

The commands in this section are available from the simulator's menu. It can contain any number of entries of the form:

```
menu_label = command_line
```

The is *menu_label* displayed in the simulator's menu. It can be any text. The *command line* is the command passed to the operating system when you select this item. Any of the "path tokens" described in Chapter 4, "Basic Operation" can be added to the *command_line* and will be properly expanded.

A number of special commands are available, all starting with the pound sign (**#**):

```
menu_label = #compile section
menu_label = #maker section
menu_label = #simulate section
menu_label = #reload
```

The first three commands replace the *section* name with the information in the corresponding [*section*] of the initialization file. (The format of these sections is described later.) This feature allows a set of commands to be defined once, then conveniently reused.

The #reload command reloads the *simulator***.ini** file. It's used when you've modified the file after starting the Hierarchy Navigator.

[SimControls]

This section specifies how net and instance names are "mangled" before being passed to the simulator or the Waveform Viewer.

`MaxLine = nn`

Specifies the maximum command-line length before the Hierarchy Navigator breaks the line by inserting a continuation character and a carriage return / linefeed. Maxline can be no larger than 1024.

`Continue=char`

Specifies an continuation character when breaking long lines. If none is specified, the default is the backslash (\).

`RootIOPrefix = string`

Specifies the prefix to be added to the names of nets external to the root schematic. Nets external to the top-level schematic are primary I/O ports. Should end with the HierChar. (See the HierChar entry.)

`RootPrefix = string`

Specifies the prefix to be added to the names of nets local to the root schematic and not primary I/O ports. Should end with the HierChar. (See the HierChar entry.)

`SubPrefix = string`

Specifies the prefix to be added to nets that do not appear in the root schematic (appear as local nets in lower level schematics). Should end with the HierChar. (See the HierChar entry.)

`InstPrefix = string`

Specifies the prefix to be added to the hierarchical path name used to set the scope of the simulator when the schematic is pushed or popped. Should end with the HierChar. (See the HierChar entry.)

`GlobalPrefix = string`

Specifies the prefix to be added to global nets. Usually the same as RootIOPrefix. (Must be left blank for SILOS.) If it is not blank, it should end with a hierarchy character. (See the HierChar entry.)

`FlattenBusses = Yes | No`

Buses are normally flattened to scalar signals (default = Yes). If this option is turned off (No), the bus name is placed in the command. The name is the local name and the simulator must understand the hierarchical context. The name is converted into the simulator's format, including the required parentheses, range delimiter, and the delimiter between signals in the complex names. (See the Concatenate entry.)

`HierChar = char`

Specifies the character denoting hierarchy levels in hierarchical names sent to the simulator. This character replaces the period (**.**) and dash (**–**) in names within the SCS environment.

`BarChar = B`

Complemented signals have an apostrophe (**'**) or underscore (**_**) as the leading character. This control attaches a prefix of B to these signals. This provides compatibility with VHDL simulators, that do not permit leading underscores in a net name.

`Separator = char`

Specifies the character used to delimit lists of nets in commands. The default is a blank space. The comma (**,**) is typically used.

`Terminator = char`

Each command sent to the simulator has a terminator character appended before the final newline. The default is not to append a terminator. The semicolon (**;**) is typically used.

`Parentheses=ccc`

Specifies a substitute for the parentheses and delimiter in bus names. The first and last character are the parentheses and the middle character is the delimiter. The default is to use the parentheses in the schematic and the colon (**:**) as a delimiter.

For example, if the schematic has A(0–7), and you have set Parentheses = [:], then the name passed to the simulator is A[0:7].

`Concatenate=`*`char`*

When buses are not flattened, complex buses (such as A,B,Data[0:3]) are processed by replacing the commas and backslashes with the Concatenate character.

[*simulate***]**

The name of this section is the same as the section name specified in the #simulate command line of the [SimTools] section.

`Command =` *`command_line`*

The command line that calls the simulator.

`Waves =` *`command_line`*

The command line that calls the Waveform Viewer.

`SimAppName =` *`appname`*

The name the simulator uses to identify itself to the Hierarchy Navigator. (For example, for silos3w it's "Silos3".) If not specified, the base name of the simulator executable file is used.

`FirstMsg =` *`section`*

section is the name of the section with the commands to be sent to simulator at startup.

`ExitMsg =` *`message`*

This message is sent to the simulator when the Hierarchy Navigator is closed. If no *message* is specified, the Simulator tells the Navigator to quit (under Windows) or sends a SIGKILL message to the Navigator (under UNIX/Motif).

`PushPop=`*`string`*

Specifies the command sent to the simulator when first starting and when the Hierarchy Navigator performs a push or pop. The command is formed by replacing the first percent sign (**%**) in *string* with the RootPrefix= value and the Navigator's current hierarchical context. If there is no percent sign in *string*, the current hierarchical path is appended to *string*.

`Prompt =` *`char`*

UNIX/Motif only. The character the simulator displays as the prompt. It's usually ">".

[SimMenu]

This section contains the list of menu buttons displayed on the Hierarchy Navigator's Simulate menu. Up to 25 entries can be added to this menu. The syntax of the entries in this section is:

```
menu_item = format_string
```

where *menu_item* is the string that appears on the menu and *format_string* describes the command sent to the simulator.

The *format_string* defines the operation of the command to the Hierarchy Navigator. The string is a template of the command sent to the simulator. Within the template, controls define the operation of the command to the Navigator.

Both simple and interactive commands can be specified. Simple commands are executed immediately when you select the command from the menu. These commands do not have any of the special format indicators described below (%n, %*n, or %i). The *format_string* for immediate commands must not contain any special characters other than a single percent sign (**%**).The percent sign must not be followed by n, i, or *. When the menu item is selected, the percent signs (if any appear in the *format_string*) are replaced by the name of the design file, and the resulting message is sent to the simulator.

Interactive commands let you select nets or instances by clicking on them. These commands include the special substitution indicator described below (%n, %*n, or %i).

When you select one of these commands from the Simulator menu, the system enters an interactive command mode allowing you to select nets or instances. The Hierarchy Navigator displays a special prompt indicating which command it is executing.

Each time you select a net or instance the special substitution characters in the *format_string* are replaced by the names of the selected element(s) and the resulting command is sent to the simulator. One format string can contain only a single substitution indicator. The substitution indicators are shown below.

%n	The Hierarchy Navigator allows you to click on individual nets. Each time a net is selected the name of the net replaces the %n and the command is sent to the simulator.

%*n	The Hierarchy Navigator lets you select a group of nets by: • Clicking on the net. • Clicking on a symbol instance to select all • Dragging a box around name flags and pins. • Holding down SHIFT to perform multiple selections. The list of nets replaces the %*n and the resulting string is sent to the simulator.
%i	The Hierarchy Navigator accepts the selection of a symbol instance. The full hierarchical name of the instance is placed in the string and the command is sent to the simulator.
%{ a bc }	This option defines a set of choices displayed in a dialog box when the command is active. Each time a message is issued, the selected choice is substituted into the command. This option is intended for commands that are used to force states in the simulator. A choice of a question mark (**?**) causes a type-in field to be included as a choice in the dialog box.

The menu commands are user-definable. The following suggested commands can be included in the [SimMenu] section:

Command	**Effect**
Free	Causes SILOS to report state changes on the selected signals and the Waveform Viewer to show the signals
Force	Forces a state on the selected signal until released with a Free command
Set	Temporarily sets the state on the selected signal
Exit	Leave SILOS

[SimButtons] (UNIX/Motif only)

The Simulator Control Window has three buttons whose behavior is controlled by this setting. This section defines the text string that is sent to the simulator each time the button is clicked. If no string is specified, the button does not appear.

`Run = string`

Clicking the Run button sends this *string*, followed by the terminator character, to the simulator.

`Stop = string`

Clicking the Stop button sends this *string*, followed by the terminator character, to the simulator. If the first two characters in the string are "^C", they are replaced with the ASCII control character (ASCII 003). This feature does not operate properly if the simulator is run by specifying a script file in the command.

`Delta = string`

A message is formed by replacing the percent sign (**%**) in the *string* with the integer value in the type-in field. If the string does not contain a percent sign, the value is appended to the string. The message is terminated with the terminator character and a \n (newline).

[*maker*]

The name of this section is the same as the section name specified in the #maker command line of the [SimTools] section.

`Process = command_line`

If the Process method is specified, the menu entry Code Simulation Models is added to the Navigator's Simulate menu. Choosing Code Simulation Models executes *command_line* with the appropriate substitutions as described below.

When the command is selected, *command_line* is expanded by replacing any "path tokens" with the actual path. The format for these substitutions is explained in Chapter 4, "Basic Operation."

If the character immediately preceding the path token (**_**), the preceding string is inserted into the path as a prefix to the file name. That is,

```
proc &F abc/_&F
```

becomes:

```
proc /pathname/schem /pathname/abcschem
```

The following controls only apply to the Schematic/Symbol method of extraction:

`RootProcess = command_line`

This line is needed only if the root-level schematic is netlisted differently than the rest of the design. In VHDL code, for example, it is common to netlist the top level of the design along with a test bench. This ensures that all the top-level I/O parts are defined properly for VHDL.

When you execute Code Simulation Models from the Hierarchy Navigator, if you have defined an entry for Root Process, the top level schematic is netlisted according to the instructions in *command_line*.

`Extension = file_extension`

Specifies the file extension for the model files. It is used by the Hierarchy Navigator to find the model files.

`Path = file_path`

This control specifies the directory with the model files. The path can either be the full path, or a directory relative to the directory containing the symbol file.

`ModelAttribute = symbol_attribute_number`

If a value is specified for this attribute on a symbol, the system assumes that the model is already available, and does not generate a model. (Setting this attribute effectively prevents SCS from coding a model.) When performing a Push or Pop in the Hierarchy Navigator, the model specified here is searched for, not the symbol file.

[*compiler***]**

The name of this section is the same as the section name specified in the #compile command line of the [SimTools] section.

Some simulators require the models to be compiled prior to invoking the simulator. This section provides the necessary controls.

`Process = command_line`

If the Process method is specified, the entry Compile Simulation Models is added to the Navigator's Simulate menu. Choosing Compile Simulation Models executes the *command_line* with the appropriate substitutions as described above.

The following controls apply only to the Schematic/Symbol method of extraction:

`RootProcess = command_line`

RootProcess is needed only if the root-level schematic is netlisted differently than the rest of the design. In VHDL code, for example, it is common to netlist the top level of the design along with a test bench. This ensures that all the top-level I/O parts are defined properly for VHDL.

If you have defined a RootProcess entry, executing Compile Simulation Models from the Hierarchy Navigator compiles the top-level schematic according to the instructions in *command_line*.

`Ext = file_extension`

The extension for the compiled model files. It is used by the Hierarchy Navigator to find the compiled model files.

`ModelExt = file_extension`

The extension for the netlist files.

`Path = file_path`

Specifies the directory where the compiled files are to go.

`ModelPath = file_path`

Specifies the directory where the model (netlist) files are.

`ModelAttribute = symbol_attribute_number`

The attribute number of the attribute that has a value only if a symbol is a simulator primitive.

[SimSources]

This section is a list of extensions and paths to the sources that are part of a design. When you use the Push or Pop command in the Hierarchy Navigator, this list is searched for a source file.

```
[SimSources]
   .abl =
   .v =
   .vf = vf\
```

If no path is specified, the symbol directories are searched.

[*FirstMessage***]**

This section is a series of numbered lines that are output, in sequence, to the simulator on startup. The name of the section corresponds to the name given in the FirstMsg entry of the [*simulate*] section, as shown below:

```
[Simulate2]
   Command = silos3w
   Waves =
   SimAppName = silos3
   ExitMsg = quit
   FirstMsg = FirstSilos2

[FirstSilos2]
   1 = RESET ALL
   2 = CON .CUSTREPORT .SYN=0 .SAV=2
   3 = LIB .\common{.v} .\symflat{.vi} .\top{.v} .{.v} c:\test\bliflib.v
   4 = File .SAV=&B
   5 = File .STO=&B.rep
   6 = Input &B.tf
   7 = Sim 0

[Export]
   exp -verilog -ext=.tf
```

The [Export] section contains the command line that calls the export program (the program that converts Waveform Editor data to a stimulus file).

Running the Waveform Viewer

The following describes the various command-line options used in launching the Waveform Viewer.

Windows

The Waveform Viewer program is named **waves**. It can be started with the Run command from the Program Manager, from the Hierarchy Navigator's Tools menu, or as part of the Navigator's Simulation Environment. In all cases, the program is run with a combination of command line arguments that specify the program's operating mode:

Option	Function
-nav	**waves** is under control of the Hierarchy Navigator. This option should be included in the Waves= control of the Simulation Environment or the Navigator Tools entry in the INI file.

-pcsilos	**waves** runs with the PC / SILOS history file.
-sim	**waves** runs dynamically with a simulator. If this option is missing, **waves** runs in the static mode using a history file for the simulation results.
-time	**waves** accepts its data from a history file written in the Timewave history file format.
basename	Specifies the base name for the display save file (**.wav**), the timewave history file (**.his**) (if the -time option is specified), and the binary save file (no extension).

UNIX/Motif

The Waveform Viewer program is named **waves**. It can be launched from a UNIX command shell, from the Hierarchy Navigator's Tools menu, or as part of the Navigator's Simulation Environment. In all cases, the program is run with command-line arguments that specify the program's operating mode:

Option	Function
-nav	**waves** is under control of the Hierarchy Navigator. This option should be included in the Waves= control of the Simulation Environment or the Navigator Tools entry in the INI file.
-sim	**waves** runs dynamically with a simulator. If this option is missing, **waves** runs in the static mode using a history file for the simulation results. **waves** opens a socket to which the Simulator connects. The default port number is 1703. This can be overridden by following the -sim option with the desired port number (for example, -sim1705).
-silos	**waves** runs with the SILOS simulator in either dynamic mode or with a SILOS history file in static mode.
-time	**waves** accepts its data from a history file written in the Timewave history file format.
basename	Specifies the base name for the display save file (**.wav**), the timewave history file (**.his**) (if the -time option is specified), and the binary save file (no extension).

SPICE

SCS provides an interface to several different versions of the SPICE Simulator. This interface extracts a netlist file in either a hierarchical or flattened format for simulation with Spice.

Both the flat and hierarchical netlisters generate netlists that are compatible with Berkeley Spice, HSPICE, PSpice, or LVS (layout versus schematic). The netlist can be written with the actual node names or with SCS-assigned node numbers.

If errors occur, the processor invokes the Notepad on the netlist file. You can use the Notepad to scroll through the netlist and identify the errors. Errors are displayed one to a line, surrounded by asterisks. For example:

```
**** Unconnected Pin on Transistor: N1 ****
```

Setting up the INI File

To run the SPICE netlister, you must enter the following information in the SCS INI file.

Adding a Menu Entry to the Hierarchy Navigator

The SPICE netlister is run as a Process from the Hierarchy Navigator. To create a Process menu option for the hierarchical SPICE netlister in the Hierarchy Navigator, use the following procedure:

1. Select the INI Editor from the Setup menu of the SCS Executive.
2. Select the Processes command from the Tools menu of the INI Editor.
3. In the dialog box, create a menu entry using one of the following statements (depending on whether you need a flat or hierarchical netlist).

   ```
   Flat SPICE
   ```

   ```
   Hierarchical SPICE
   ```

4. Enter the command line as:

   ```
   Spicenet (for flat SPICE netlist)
   ```

   ```
   Hspicent (for hierarchical SPICE netlist)
   ```

The following options can be appended to the command line.

Hierarchical Spice Command Line Options

The following four options are mutually exclusive. Choose only one:

/B	Use Berkeley Spice conventions.
/H	Use HSpice conventions.
/L	Use LVS conventions.
/P	Use PSpice conventions.

One or more of the following options can be specified:

/A	Automatic mode; do not present dialog box.
/G	Allow use of .GLOBAL instruction (hierarchical netlist only).
/M	Write subcircuit netlist (hierarchical netlist only).
/N	Use node names instead of numbers.
/U	Do not add u to length or width and do not add p to areas.
/X	Ignore Prefix of X on subcircuits and Use Primitives (hierarchical netlist only).
-ext=.abc	Create a netlist file with the specified extension (*design*.abc). The specified extension overrides the default extension for that netlist format.

If the /A option isn't specified, the netlister displays the option dialog box. If the /A option is specified, the netlister does not display the dialog box but does use any other options specified on the command line. If other options are specified without the /A option, the dialog box is displayed with the selected options as the defaults.

Required Attributes

Several attributes are reserved for use with the SPICE interface. These attributes are defined in the INI file and provide necessary information for constructing the network description.

Prefix	This symbol attribute identifies the type of element represented by the symbol. Only the first character is significant. Examples are Q, M, D, C, and R.

SpiceModel	This optional symbol attribute specifies the model name listed for this symbol. The model name should match the name on a .MODEL statement in the SPICE control file.
SpiceLine	This optional symbol attribute is used to add parameters to the SPICE network description of this symbol. If present, it is copied to the network description without interpretation. It can be used on any primitive symbol. It can also be used on instances of blocks in hierarchical netlists, allowing parameters to be passed in hierarchical SPICE netlists. The SpiceLine attribute automatically wraps to another line if it is longer than 80 characters.
SpiceLine2	This attribute is copied to the netlist as part of the [subckt] header when used on Block symbols. For primitive symbols this attribute is treated in the same as SpiceLine. It can be used on any primitive symbo, and also on definitions of blocks in hierarchical netlists. SpiceLine2 is used primarily to define parameters on subcircuits. The parameters defined using SpiceLine2 can then be passed in to individual instances using the SpiceLine attribute. The SpiceLine2 attribute automatically wraps to another line if it is longer than 80 characters.
Impedance	Characteristic impedance for transmission lines.
Multi	This optional symbol attribute indicates a multiplier factor for multiple devices in parallel. It works for primitives and subcircuits (except for transmission lines, diodes, voltage and current sources).
AreaS	The source area for transistors. It is defined as a derived attribute, allowing you to set its value as a function of other attributes.
AreaD	The drain area for transistors. It is defined as a derived attribute, allowing you to set its value as a function of other attributes.
PeriS	The source perimeter for transistors. It is defined as a derived attribute, allowing you to set its value as a function of other attributes.

PeriD The drain perimeter for transistors. It is defined as a derived attribute, allowing you to set its value as a function of other attributes.

NRS Equivalent number of squares of source diffusion in series with a transistor's source. It is used by SPICE to calculate source resistance.

NRD Equivalent number of squares of drain diffusion in series with a transistor's source. It is used by SPICE to calculate drain resistance.

DefSubstrate Default net to connect to bulk node on transistor. It is used if a transistor is shown with only three terminals. The fourth terminal (the bulk node) is defined by this attribute. The name referenced by this attribute must be a global net that exists on the schematic.

Several other attributes are used by the SPICE interface.

Value Several primitive types use this attribute to represent the basic value of the primitive. Elements using this attribute include: capacitor, resister, inductor, voltage source, current source, and transmission line.

Width transistor width

Length transistor length

PinName The pin name is used to identify pins for some elements.

SpiceOrder This attribute forces the netlisted order of subcircuit connections. If this pin attribute is not set on individual pins in a subcircuit, the pins are netlisted in the following order:

1. Inputs sorted alphabetically
2. Bidirectionals sorted alphabetically
3. Outputs sorted alphabetically

The netlist pin order on a subcircuit can be forced by setting the SpiceOrder attributes of each pin to numerical values ranging from 1 (first in order) to the number of pins. This option can be used to interface with the PSPICE digital libraries.

These attributes are assigned to the symbol and its pins by use of the Symbol Attribute and Pin Attribute commands of the Symbol Editor. If a symbol has an attribute assigned, it becomes the default value for each instance of the symbol on the schematic. The default values can be overridden for a particular instance of the symbol on the schematic by use of the Add: Attribute command of the Schematic Editor or the Edit: Attribute command of the Hierarchy Navigator.

Refer to your SPICE Manual for a detailed explanation of the SPICE format.

Controlling The Netlist Extraction

The Hierarchical SPICE Netlist Processor (**hspicent**) is selected from the Processes menu of the Hierarchy Navigator. This processor creates a netlist file with the name *design*.spi (where *design* is the base name of your design). This file is a topological description of the hierarchical design and can be combined with user-supplied files that specify the input stimuli and monitor the output.

The Flat SPICE Netlist Processor (**spicenet**) also creates a netlist file with the name *design*.spi. However, it converts hierarchical designs into flat (non-hierarchical) netlist form. Block (and other high-level) symbols are converted into primitive symbols and the nets connecting them. This version is useful when the design contains instance-specific attribute overrides.

When the netlist processor is selected from the Process menu of the Hierarchy Navigator a dialog box offers several program options. These options are:

Option	Effect
Berkeley Spice	Use Berkeley Spice conventions.
HSpice	Use HSPICE conventions.
LVS	Use LVS conventions.
PSpice	Use PSpice conventions.

In addition to the buttons, there is a check box labeled Node Names that lists nets by name rather than by number. This box is marked by default for PSpice and HSpice.

The Hierarchical SPICE Netlister has three additional check boxes:

Check Box	Effect
Subcircuit	Produces a special version of the netlist for a sub-hierarchy when working with a split hierarchy.
Globals	Permits the use of the .GLOBAL HSpice instruction. Any global nets are declared in a .GLOBAL statement. Global nets are not explicitly passed inside subcircuits through the node connectivity list in the subcircuit definition. When the PSpice option is selected, global nets are specified with a $G_ prefix in the netlist. If the option is not specified, no nets in the netlist are treated as global. In this case, all nets required in subcircuits are passed in through the node connectivity list in the subcircuit definition, including Vdd, but not including Gnd.
Use Primitives	Determines whether the netlist expands Block symbols having an X prefix. If the Use Primitives option is not specified, and the netlist processor encounters a symbol that has an X prefix, it searches for a subcircuit SPICE file with the same name as the symbol (but with extension **.spi**) and reads that file into the netlist without looking for any lower-level schematics. Both the flat and hierarchical SPICE netlist generators are modified to recognize the X prefix as a user-defined model. In both cases, the ModelName symbol attribute is used as the model name if it is defined. The symbol's name is used if no model name is available. If the Use Primitives option is selected, the netlist processor ignores symbols with an X prefix and codes the netlist using only the lowest-level primitives.

Both the flat and hierarchical SPICE netlist generators are modified to recognize the X prefix as a user-defined model. In both cases, the ModelName symbol attribute is used as the model name if it is defined. The symbol's name is used if no model name is available.

SPICE Format Conversions

When netlists are written with node numbers, a special preprocessor is run that converts names in a control file into the corresponding numbers. This lets you generate the control file using meaningful node names and have SCS automatically convert the names to numbers for SPICE.

The Hierarchical SPICE netlister creates artificial nodes whenever it encounters unconnected pins on block symbols. Each instance of an unconnected pin is given its own unique node. In the node number format this node is a number larger than any number on the real nodes. In the node name format this node is assigned a name of the form UNC*nn*, where *nn* is a unique number for each occurrence of an unconnected pin. Avoid assigning names with this format.

The Hierarchical SPICE netlister gives instances alphanumeric names with the Prefix attribute as the first character, followed by the instance name. Unnamed instances and nets are given names beginning with I_ and N_ by the Hierarchy Navigator. The Hierarchical SPICE netlister converts the I_ from unnamed instance names and replaces it with UN. Similarly, the N_ from unnamed nets is replaced with UN. Avoid assigning net names or instance names of the form UNC*nn* (where *Mnn* is an integer).

The Hierarchical SPICE netlister writes a hierarchical netlist. Therefore, any instance-specific attribute overrides that are added by the Hierarchy Navigator do not appear in the netlist.

Working with the Hierarchical Netlist

You sometimes need to know which element in a netlist corresponds to which instance in the schematic. When interpreting a particular SUBCKT in the SPICE netlist, first convert the element name to the original instance name by removing the prefix and, if necessary, convert the leading UN to I_. When using the normal option, with Node Numbers instead of names, the comments at the front of each SUBCKT description tell which nets represent which numbers.

To find the corresponding element on the schematic::

1. Use the Push / Pop command to find the schematic that represents the SUBCKT.
2. Select the Query command.
3. Type i= followed by the instance name from the netlist and press ENTER. The required element is highlighted.

Some SPICE netlists permit node names to be used, rather than converting them to numbers. Using the Node Names option with these formats retains the original names of the nets, making it easier to find the net you want.

Working with the Flat Netlist

The Flat SPICE netlister can produce large files, since every repeated hierarchical block is replicated and assigned unique instance and net names.

The alphanumeric element names are formed by taking the Prefix (M, Q, and so on) and attaching it to the instance number (for example, M123 or Q17).

The flat netlist is difficult to read. The Query command in the Navigator can help.

To find the net with a given node number:

1. Select the Query command.
2. Type the required node number on the prompt line and press ENTER. The Query command pushes or pops to the correct schematic and highlights the requested node.

To find the instance with a given element name:

1. Select the Query command.
2. Type i=*nnn* (where *nnn* is the number portion of the element name) and press ENTER. The Query command pushes or pops to the correct schematic and highlights the requested instance.

Preprocessing a Control File

Many versions of SPICE require that you specify nodes with numbers rather than names. This can be confusing if you are working with the numbered nodes for SPICE and the named nodes on your schematic. The SPICE netlister has a preprocessor that maps the names on your schematic into node numbers for SPICE. You can write the SPICE control files using the names on your schematics rather than numbers.

The preprocessing feature of the SPICE netlister works as follows:

1. Place all SPICE control instructions in a file having the file extension **.spp**.
2. Place quotes (" ") around every net name.

3. Run the SPICE netlister with the node number option. The netlister creates a copy of the **.spp** file where the names in quotes are replaced with the corresponding numbers. This new file has the extension **.spc**.

Attributes Required for SPICE Elements

Different SPICE elements require different attributes to be assigned to them. This section lists the various attributes used with the different SPICE primitives.

Bipolar Junction Transistor

Prefix=	Q
SpiceModel=	name of Model for this device
Multi=	optional multiplication factor
SpiceLine=	optional additional parameters
SpiceLine2=	optional additional parameters
PinName=	first letter C for Collector
PinName=	first letter B for Base
PinName=	first letter E for Emitter
PinName=	first letter S for optional Substrate

A junction transistor is a four-terminal device, with the fourth terminal the substrate connection. Transistor symbols are usually drawn showing only the base, collector, and emitter. The substrate has an implied connection to ground. SPICE requires this connection whether or not it's shown on the symbol. The DefSubstrate attribute on transistor symbols lets you connect the substrate node to any global net in the schematic.

MOSFET, MESFET, and JFET Devices

Prefix=	M for MOSFET, B for MESFET, or J for JFET Devices
SpiceModel=	Name of Model for this device
Width=	Width in meters
Length=	Length in meters
AreaS=	Area of substrate
AreaD=	Area of Drain

PeriS=	Perimeter of substrate
PeriD=	Perimeter of drain
NRS=	Number of squares of source diffusion
NRD=	Number of squares of drain diffusion
Multi=	Optional multiplication factor
SpiceLine=	Optional additional parameters
SpiceLine2=	Optional additional parameters
PinName=	First letter D for Drain
PinName=	First letter G for Gate
PinName=	First letter S for Source
PinName=	First letter B for optional Base (substrate)

A MOSFET is a four-terminal device, with the fourth terminal the substrate connection. MOSFET symbols are usually drawn showing only the drain, gate, and source connections. The bulk has an implied connection to ground or Vdd. The PSpice and Berkeley netlisters require the substrate connection either directly or through the DefSubstrate attribute. The DefSubstrate attribute on MOSFET symbols lets you connect the substrate node to any global net in the schematic.

Transmission Line

Prefix=	T
Impedance=	Characteristic impedance
Value=	Transmission line delay
SpiceLine=	Optional additional parameters
SpiceLine2=	Optional additional parameters
PinName=	First two letters I1 for first input
PinName=	First two letters R1 for first reference node
PinName=	First two letters I2 for second input
PinName=	First two letters R2 for second reference node

Diode

Prefix=	D
SpiceModel=	Model name
SpiceLine=	Optional additional parameters
SpiceLine2=	Optional additional parameters
PinName=	+ for cathode pin, anode name is optional

Capacitor

Prefix=	C
Value=	Capacitance in farads
SpiceModel=	Model name (PSpice, HSpice, and LVS)
Multi=	Optional multiplication factor
SpiceLine=	Optional additional parameters
SpiceLine2=	Optional additional parameters
Name=	Optional name for both pins

Independent Sources

Prefix=	I for independent current source
Prefix=	V for independent voltage source
Value=	Voltage or current specification
SpiceLine=	Optional voltage or current specification
SpiceLine2=	Optional additional parameters
PinName=	+ for positive pin, negative name is optional

Linear Dependent Sources

Prefix=	E for Voltage controlled voltage source
Prefix=	G for Voltage controlled current source
Value	Dependent gain for source
InitCond=	Optional initial conditions
SpiceLine=	Optional voltage or current specification
SpiceLine2=	Optional additional parameters

PinName=	+ for positive pin, negative name is optional
PinName=	P for positive controlling node
PinName=	N for negative controlling node

Inductor

Prefix=	L
Value=	Inductance in henries
SpiceModel=	Model name (PSpice, HSpice, and LVS)
Multi=	Optional multiplication factor
SpiceLine=	Optional additional parameters
SpiceLine2=	Optional additional parameters
PinName=	Name is optional for both pins

Resistor

Prefix=	R
Value=	Resistance
SpiceModel=	Model name (PSpice, HSpice, and LVS)
Multi=	Optional multiplication factor
SpiceLine=	Optional additional parameters
SpiceLine2=	Optional additional parameters
PinName=	Optional name for both pins

Subcircuits (Hierarchical SPICE Only)

Multi=	Optional multiplication factor
SpiceLine=	Optional additional parameters on subcircuit instances
SpiceLine2=	Optional parameter definition in subcircuit header
SpiceOrder=	Optional pin attribute; forces order of pins in subcircuit header

Examples

Two examples of SPICE specifications are presented. An independent voltage source shows the use of the SpiceLine attribute. A MOSFET example shows the typical attribute assignments for transistors in a CMOS IC design.

Independent Voltage Source

Symbol Attributes

InstName=	IN7
Prefix=	V
SpiceLine=	0.001 AC 1 SIN (0 1 1MEG)

Pin Attributes

PinName=	+ (This is the pin attribute on the positive pin.) The plus pin is connected to node 10 and the other pin is connected to GND.

Netlist Output

```
VIN7 10 0 0.001 AC 1 SIN (0 1 1MEG)
```

MOSFET

Symbol Attributes

InstName=	DIFF
Prefix=	M
SpiceModel=	P_PWELL
Width=	10E-6
Length=	1E-6
AreaS=	30E-12
AreaD=	30E-12
SpiceLine=	IC=0.2,3.7,-2.1

Pin Attributes

PinName=	D this pin corresponds to drain and is node 3

PinName= G this pin corresponds to drain and is node 4

PinName= S this pin corresponds to source and is node 5

PinName= B this pin corresponds to substrate and is node 2

Netlist Output

MDIFF 3 4 5 2 P_PWELL L=1E-6 W=10E-6 AD=30E-12 AS=30E-12 IC=0.2,3.7,-2.1

Error Messages

`Missing or Invalid Prefix Code on Symbol Type type`

A primitive symbol did not have the Prefix defined.

`Invalid Prefix on Element element`

The Prefix attribute is not one of the supported types.

`Missing Symbol for filename`

When using the Subcircuit option, you must have a symbol for the root schematic.

`GND not defined as a GlobalNet.`

The Controls: SCS Controls section of the INI file must have an entry for globals that includes GND. Example: GlobalNets=VDD GND

`Unconnected Pin on Bipolar Transistor transistor`

Bipolar transistors (Prefix=Q) must have nets wired to the Collector, Base and Emitter pins.

`Unconnected Pin on Transistor transistor`

MOSFET (Prefix= M) , MESFET (Prefix= B) and JFET (Prefix= J) must have nets wired to the Source, Gate and Drain pins.

`Unconnected Pin on Transmission Line line`

Transmission Lines (Prefix= T) must have nets wired to the pins labeled I1, I2, R1 and R2.

`Unconnected Pin on Diode diode`

Diodes (Prefix= D) must have nets wired to the two pins one of which is named with a plus sign (+).

`Unconnected Pin on Source source`

Independent Current Source (Prefix= I), Independent Voltage Source (Prefix= V), Voltage Controlled Voltage Source (Prefix= E) and Voltage Controlled Current Source (Prefix= G) must have nets wired to the two pins, one of which is named with a plus sign ("+").

`Unconnected Pin on component`

Resistor (Prefix= R), Inductor (Prefix= L) and Capacitor (Prefix= C) must have net wired to the two pins.

`Failed to find Net (nn)`

This is a System Error; you should not see this.

`Missing or Unrecognized Default Substrate for transistor`

MOSFET transistors must have either four terminals or a valid DefaultSubstrate attribute. The DefaultSubstrate attribute must be the name of a global net and that net must exist on the schematic containing the transistor.

Timemill

SCS provides an interface to EPIC Design Technology's Timemill Simulator and Pathmill Critical Path Analyzer. This interface lets you extract a netlist file for Timemill or Pathmill. The Hierarchy Navigator is used to examine the Pathmill results and the Interactive Logic Analyzer is used to examine the Timemill results.

The Timemill Netlist Processor is selected from the Processes menu of the Hierarchy Navigator. This processor creates a netlist file with the name *design*.ntl, where *design* is your design's name. This file is a topological description of the hierarchical design. For simulation, it is combined with a user-supplied stimulus file that specifies the initial values, clock, and pattern definitions for the simulator. The pattern file is named *design*.io.

For critical path analysis, the netlist is combined with a user-supplied static setup file that specifies the source and sink nodes. Any text editor can be used to create the stimulus file or static-setup file.

Any instance-specific attribute overrides to the Width or Length attributes on transistor elements are written to a separate configuration file named *design*.ovr. This file is used as input to Timemill and Pathmill.

If errors occur, the processor invokes the Notepad on the netlist file. You can use the Notepad to scroll through the netlist file and identify the errors. Errors are displayed one to a line, surrounded by asterisks. For example:

```
**** Missing Pin Named: EN ****
```

Setting up the INI File

To run the netlister, you must specify the following information in the INI file.

Adding a Menu Entry to the Hierarchy Navigator

The Timemill netlister and simulator are run as a Process from the Hierarchy Navigator. To create a netlist Process menu option for Timemill in the Hierarchy Navigator, use the following procedure.

1. Select the INI Editor from the SCS Executive.
2. Select the Processes command from the Tools menu of the INI Editor.
3. Create the following menu entry:

   `Timemill Netlist`
4. Add the following command line entry:

   `timenet`

Any of the following options can be added to the command line:

/A	Automatic mode; do not display dialog box.
/M	Write subcircuit netlist.
/P	Use lowest-level primitives instead of models.

The netlister displays the Option dialog box if the /A option is not specified. If the /A option is specified the netlister does not display the dialog box but does use any other options specified on the command line. If other options are specified without the /A option the dialog box has the selected options as the defaults.

Required Attributes

Three attributes are reserved for use with the Timemill interface. These attributes are defined in the INI file and provide the information needed for constructing the network description.

TimilModel This symbol attribute identifies the primitive or model to be used to represent the symbol. Primitives are N, P, R, C. Anything else is treated as a model.

TimilExtra This symbol attribute specifies the complete syntax of any parameters, states or initial values required by this type of symbol from blocks to transistors. If present, this string is appended to the network description after all the pins are listed.

If a functional model is written that requires allocation of memory for states the symbol must contain the attribute TimilExtra= (ST=*n*), where *n*is the number of states required.

TimilOrder This pin attribute determines the order that pins are listed in the netlist. For models, pins are identified by number and appear in the netlist in the same order as the numbers in the TimilOrder attribute. For primitives, the pins are identified by name.

If the PinName is different from the name required to identify the pin for Timemill then the TimilOrder can be used as an alternate name. The netlist processor checks the TimilOrder before looking at the PinName.

The following attributes are also used by the Timemill interface.

Value The Value attribute specifies capacitance values in nanofarads on capacitors and resistance values in ohms on resistors.

Width The Width attribute specifies the channel width of transistors in microns.

Length The Length attribute specifies the channel length of transistors in microns.

Polarity The Polarity attribute determines which pins are output pins.

PinName The PinName attribute identifies pins for the primitives if the TimilOrder attribute does not specify the name.

These attributes are assigned to the symbol and its pins with the Add: Symbol Attribute and Add: Pin Attribute commands of the Symbol Editor. If a symbol has an attribute assigned, it becomes the default value for each instance of the symbol on the schematic. The default values can be overridden for a particular instance of the symbol on the schematic by use of the Add: Attribute command of the Schematic Editor or the Edit: Attribute command of the Hierarchical Navigator.

Preparing Additional Files

Timemill requires a stimulus file containing vectors for primary inputs. Pathmill requires a static setup file specifying the input and output nodes to be searched. Both of these files are manually created using a text editor.

Sample Timemill Stimulus File

The following is an example stimulus file for use with a design called **updown**. This file must be named **updown.io**.

```
(is=clk)(en=reset)(ot=R)(st=5)(ov=0)(sv=0,5,500);
(is=clk)(en=set)(ot=S)(st=5)(ov=1)(sv=0,400,800);
(is=clk)(en=yy)(ot=Y)(st=5)(ov=1)(sv=0,500,800);
(is=clk)(en=xx)(ot=X)(st=5)(ov=0)(sv=0,200,800);
(is=clk)(en=ck1)(ot=CK)(st=5)(ov=1)(sv=0,20,40);
(is=chk_out)(en=updown.tst)(it=Q0,Q1,Q2,Q3)(st=3);
```

Refer to the Timemill Reference Manual for more information about clock and pattern inputs.

Sample Pathmill Static Setup File

The following is an example static setup file for use with a design called **updown**. This file must be named **updown.s**.

```
no_warning
source_node S
source_node Y
source_node CK
source_node X
source_node R
sink_node Q3N
sink_node Q3
number_of_long_path 10
```

Refer to the Pathmill Reference Manual for more information about the static setup file.

Controlling the Netlist Extraction

When the netlist processor is selected from the Processes menu of the Hierarchy Navigator, a dialog box offers the following options:

Netlist Writes the netlist only.

Subcircuit Writes a special version of the netlist for a sub-hierarchy when working with a split hierarchy.

In addition to the buttons, there is a check box labeled Use Primitives. When the netlist processor encounters a symbol that has the TimilModel attribute specified, it normally codes that model into the netlist without looking for any lower-level schematics. If the Use Primitives option is selected, the netlist processor ignores the TimilModel on higher-level blocks and codes the netlist using only the lowest-level primitives. This feature makes it possible to switch between simulating at the functional level and the switch level without changing anything on the schematic.

For example, a design contains a schematic with a symbol called NAND2 and the TimilModel attribute of this symbol is NAND. An underlying schematic exists for the symbol NAND2 containing transistor symbols NMOS and PMOS (TimilModel= N and P). The normal option codes the netlist with NAND without any reference to the underlying schematic. With the Use Primitives option, NAND2 is coded as a subcircuit using n-channel and p-channel transistor switch level elements.

Timemill Format Conversions

The Timemill netlister assigns names to nodes whenever it encounters unconnected pins on block symbols. The format of the name is UNC*nn*, where *nn* is a unique number for each occurrence of an unconnected pin. Do not assign names using this format in the schematic.

Building Timemill Symbol Libraries

Timemill has several built in primitive functions that are used in simulation. Several attributes in these primitives must take on prescribed values.

The following list gives the attributes needed for various Timemill primitives.

CMOS Transistors

TimilModel= either N or P
Width= channel width, default scale factor is microns
Length= channel length, default scale factor is microns
TimilOrder= or PinName= S, G, D

Resistor

TimilModel=
Value= resistance in ohms, default scale factor is 1.
TimilOrder= or Name= S, D

Capacitor

TimilModel= C
Value= capacitance, default scale factor is nanofarads

This symbol should only have one pin or if it has two pins one of them must be connected to GND.

Models

TimilModel= name of the model
TimilExtra= optional specification of states and initial values

Symbols that represent functional models require the pin output order to be specified. This is done with the TimilOrder attribute. It indicates the numeric order that the pins are to be output. The pins are grouped according to Polarity.

Examples

Two examples of Timemill models are presented. A J-K flip-flop shows the use of the TimilOrder and TimilExtra attributes. The second example shows how buses can be automatically ordered.

Functional Model for J-K Flip-Flop

Symbol Attributes

TimilModel= xjkff
TimilExtra= (st=2)

Pin Attributes

PinName	TimilOrder	Polarity
J	2	In
K	4	In
CLK	3	In
SET	5	In
CLR	1	In
Q	1	Out
QN	2	Out

In the above example, the nets connected to the input pins are listed in the order CLR, J, CLK, K, and SET. The nets connected to the output pins are listed in the order Q and QN. This functional model requires two states to store Q and QN, so the TimilExtra attribute is specified to allow for these two states.

Dummy Symbol Using Bus

Symbol Attributes

TimilModel= DUMMY

Pin Attributes

PinName	TimilOrder	Polarity
OUT		Out
CLOCK	1	In
DATA[0;3]	2	In
RESET	5	In

In the above example, the nets connected to the input pins are listed as follows CLOCK, DATA0, DATA1, DATA2, DATA3, RESET. The output pin does not require TimilOrder to be specified since there is only one output pin.

The numbering sequence for a bus pin should treat that pin as a sequence of individual numbers. In this example, DATA[0:3] is given a PinName of 1. Since there are four pins, the next usable number for a PinName is 5.

Error Messages

`No Model Defined for` *`symbol`*

A primitive symbol did not have the TimilModel defined.

`Missing Symbol for` *`schematic`*

The Subcircuit option requires a symbol for the root schematic.

`Missing Pin Named:` *`attribute`*

A primitive symbol is expected to have a pin with the given PinName or TimilOrder attribute.

`Missing Pin Order for TimeMill On` *`model`*

Models require that all pins of a given Polarity have TimilOrder specified as a number indicating the numerical order for the pin to be listed. If order doesn't matter, then all the pins of the same Polarity should have the TimilOrder left blank.

`Incorrect Pin Ordering in` *`model`*

This model has a gap in the sequence of numbers in the TimilOrder attribute.

`Unconnected Capacitor Pin On:` *`capacitor`*

The capacitor pin must be connected.

`Capacitance Not to GND`

If a Capacitor symbol has two pins, one of the pins must be connected to GND.

`Missing or Invalid VALUE (Capacitance) for` *`capacitor`*

The Value attribute must be set to the capacitance for this node in nanofarads.

`Missing or Invalid VALUE (Resistance) for` *`resistor`*

The Value attribute must be set to the resistance in ohms for this resistor.

`Missing or Invalid WIDTH for` *`transistor`*

The Width attribute must be set to the channel width of this transistor in microns.

`Missing or Invalid LENGTH for` *`transistor`*

The Length attribute must be set to the channel length of this transistor in microns.

Verilog

SCS provides the means of including the Verilog Simulator in the design environment. The interface between SCS and Verilog consists of three parts:

Netlist Extraction SCS can extract the structural module description from the schematic. It can also extract a template description from a symbol that represents a behavioral model.

Simulator Control (UNIX / Motif only) The Verilog simulator can be launched and controlled from the Hierarchy Navigator. Various commands can be defined and added to a menu and the elements (such as nets) are selected by clicking on the schematic.

Output Viewing (UNIX / Motif only) The output of the simulator can be viewed graphically with the Waveform Viewer. The simulation results can be viewed as the simulation progresses (dynamic mode) or after the simulation is completed (static mode) by using a history file.

The Waveform Viewer can either run alone or in conjunction with the Hierarchy Navigator. In the latter case, the schematic can be back-annotated with the states of signals as defined in the simulator.

The necessary code is in the directory **/interfaces/verilog** as supplied on the UNIX distribution tape. A test example is included in the **/interfaces/veritest** directory. A sample **verilog.ini** file is included in the **/data** directory on the distribution tapes or disks.

Setting up the Verilog Environment in SCS

The Verilog Netlister is run from the Simulator menu of the Hierarchy Navigator for hierarchical netlists. The Verilog netlister creates a top-level netlist file with the name *design*.v (where *design* is the top-level name of your design hierarchy) and individual netlist files for sub-level blocks.

The Verilog netlister is also run from the Tools menu of the Schematic Editor and the Tools menu of the Symbol Editor to create Verilog netlists of individual schematics or modules.

Setting up the INI File

To run the interface, you must enter the following information in the INI file.

The Simulator parameter in the Controls section of the INI file enables the Simulator menu of the Hierarchy Navigator with the name and options of the selected simulator. Enter “verilog” for the Simulator parameter.

The remaining parameters for the Verilog simulation environment are kept in the **verilog.ini** file in the CONFIG directory specified in the Windows Registration Editor, or the ECS_ROOT**/data** directory (under UNIX / Motif). Each design can also have a separate **verilog.ini** file in its working directory.

Adding a Tools Menu Entry to the Schematic Editor

The Verilog netlister can be run as a Tool from the Schematic Editor, independent of the Simulator parameter. The program generates a module netlist of the schematic block. The module declaration is extracted directly from symbol files, so a symbol for the schematic must exist for the netlist to be written. This gives the user independent control of module information and its underlying netlist.

To create a Tools menu option for Verilog in the Schematic Editor, use the following procedure:

1. Select the INI Editor from the SCS Executive.
2. Select the Schematic Tools command from the Tools menu of the INI Editor.
3. In the dialog box, create the following menu entry:

   ```
   Verilog Netlist
   ```

4. Add the following command line entry:

   ```
   vericode
   ```

Any required options are added after "vericode" on the command line.

Adding a Tools Menu Entry to the Symbol Editor

The Verilog netlister can be run as a Tool from the Symbol Editor, independent of the Simulator parameter. The program generates a module statement for the symbol being edited.

To create a Tools menu option for Verilog in the Symbol Editor, use the following procedure:

1. Select the INI Editor from the SCS Executive.
2. Select the Symbol Tools command from the Tools menu of the INI Editor.

3. In the dialog box, create the following menu entry:

   ```
   Verilog Netlist
   ```

4. Add the following command line entry:

   ```
   vericode
   ```

Any required options are added after vericode on the command line.

Command Line Options

The following options are used with the **vericode** netlister:

-n	Display in selected editor
-NETS	Force net and pin names to uppercase
-nets	Force net and pin names to lowercase
-INST	Force instance names to uppercase
-inst	Force instance names to lowercase
-model	Force VeriModel names to lowercase
-noprim	Code primitive types as modules

The selected editor is specified with the Text Editor option in the System Controls section of the INI file.

Required Attributes

Five attributes are reserved for use with the Verilog interface. These attributes are defined in the INI file and provide information necessary for constructing the network description.

VeriModel

This symbol attribute identifies the primitive or model used to represent the symbol. It is used in conjunction with the VeriName pin attribute to generate the instantiation line for primitive and model blocks. Any name that is not one of the primitives is treated as a model. This attribute cannot be overridden in the schematic editor or the Hierarchy Navigator on an instance basis because the symbol files are used to determine the Verilog model names. Thus any such change will not have any effect on the Verilog netlist that is created.

The remainder of the attribute (after the model or primitive name) is copied into the network description without interpretation. Therefore, if a strength qualifier is desired on a primitive, it can be appended to the primitive name in this attribute. If the -model option is used, the Verilog model name is forced to lower case. If it is not used, the Verimodel attribute is netlisted with the case specified on the symbol definition.

VeriName

The Verimodel and Veriname attributes are useful only for blocks with hand-coded models. These attributes are used to manually match the model and pin names of the hand-coded models with the model and pin names of the symbols that represent these models. If present, this pin attribute is used to determine the pin names that are netlisted in the instantiation line for models only if they have a VeriModel attribute assigned. If not present, the SCS PinName attribute is used to determine the pin names. This attribute should be used if the PinName attribute is different from the name required to identify the pin in the Verilog netlist.

In cases where a pin must be ignored during netlisting (hidden pins are one example), it can be given a name beginning with a two dashes (—) using the VeriName attribute. Note that the Verimodel attribute must be assigned a value for the VeriName attribute to be used.

VeriStrength

This symbol attribute specifies the strength for the outputs on this gate. If present, this string is inserted into the network description between the VeriModel and the VeriTimes.

VeriTimes

This symbol attribute specifies the delay for this gate. If present, this string is inserted into the network description between the VeriModel and the pins.

VeriComp

This attribute causes a defparam line to be generated as needed by the Logic Automation Incorporated libraries. If a VeriComp attribute is specified, the line

`defparam "`*`instance_name`*`" COMPONENT="`*`VeriComp_value`*`"`

precedes the instance line.

Several other attributes are used by the Verilog interface.

Polarity

The Polarity attribute is used by the Verilog netlister to determine which pins are the output pins for several of the primitive types. Legal values are INPUT, OUTPUT, BIDIR.

PinName

The PinName is used to identify pins for some primitives and for all models if the VeriName attribute is not present or is not used.

These attributes are assigned to the symbol and its pins with the Symbol Attribute and Pin Attribute commands of the Symbol Editor. An assigned attribute becomes the default value for each instance of the symbol on the schematic. Note that the Verilog netlister writes a hierarchical netlist. Therefore, any instance-specific attribute overrides added by the Hierarchy Navigator do not appear in the netlist.

Setting Up the verilog.ini File

The Hierarchy Navigator can be configured to control the Verilog simulator. The first step is to specify the simulator parameter to be Verilog using the INI editor. This causes the Navigator to read the **verilog.ini** file that must be in the working directory, or in the CONFIG directory specified in the Registration Database, which can be changed with the Windows program RegEdit, or the $ECS_ROOT**data** directory (under UNIX/Motif).

The **verilog.ini** file contains all the necessary commands and directives to control the Verilog simulator.

A sample **verilog.ini** file is shown below. A copy of this file is provided with the SCS release tape for your convenience, and it's recommended that you use this file until you become more familiar with the SCS simulation environment. Refer to the section "Simulation Environment" in this chapter.

```
;NOTE: This file is for running silos from hiernav
;Silos uses verilog and to write all the netlists from
;the Navigator using "make" we add [MakeVerilog]
;the name "MakeVerilog" is arbitrary.  It matches the value specified after
;#Make in a command in the [SimTools] section
[MakeVerilog]
;need a process to make the netlists from the Navigator
   Process = vericode -model &R
;We use the default extension .v so we need the following line to let the
;Navigator know this.
   Extension = .v
;we want to put the .v files in the same directory with the symbols so
;we leave the Path blank.
   Path =
;the Verilog Model attribute is number 20.  If attribute #20 has a value
;it means that the symbol is a verilog primitive or has a .v file supplied
;by the user.  So, if we run make from the Navigator we must have the
;following line
   ModelAttribute = 20

[SimTitle]
;If there is no Title the menu in the Navigator will be "Simulator"
;We would like the menu to say "Silos III" so we need the following line
   Title = Silos III

[SimTools]
;the next line invokes the "make" facility to write the verilog netlists.
   Code Verilog = #Make MakeVerilog
   Waveform Editor = wet -nav &B
   Edit silos.ini = notepad %config\silos.ini
;the next line invokes the "simulate" facility to start silos using the
;commands from "Simulate1"
   Start Silos = #Simulate Simulate1
;the next line invokes the "simulate" facility to start silos using the
;commands from "Simulate2"
   Start Silos Flat = #Simulate Simulate2
   Review Silos Results = waves -nav -silos &B

;we want to be able to start Silos from the Navigator so we add [Simulate1]
;the name "Simulate1" is arbitrary.  It matches the value specified after
;#simulate in a command in the [SimTools] section
[Simulate1]
;Command is the entire command line to start the simulator
   Command = silos3w
;Waves is the entire command line to start the Waveform Viewer
   Waves = waves -nav -silos &B
;SimAppName is the name the simulator uses to identify itself to the Navigator
   SimAppName = silos3
;ExitMsg is the command to tell the simulator to quit
   ExitMsg = quit
;FirstMsg is the name of the section that contains the messages to send to
;the simulator when it starts up.
   FirstMsg = FirstSilos1
```

```
;We have several commands to send to the simulator when it starts so we add
;[FirstSilos1].  the name "FirstSilos1" is arbitrary.  It matches the value
;specified for the FirstMsg entry in the section that starts the simulator.
[FirstSilos1]
;the values on the left side of the equal sign are not used but they must
;be unique within the section
;we use numbers just for simplicity
   1 = RESET ALL
   2 = CON .CUSTREPORT .SYN=0 .SAV=2
;the LIB command points to where the library files are stored on this machine
;rather than hard code path names, these could be specified in environment
;variables...
   3 = LIB .\common{.v} .\symbus{.vi} .\top{.v} .{.v} E:\test\bliflib.v
   4 = File .SAV=&B
   5 = File .STO=&B.rep
   6 = Input &B.tf
   7 = Sim 0

;we want to be able to start Silos from the Navigator so we add [Simulate2]
;the name "Simulate2" is arbitrary.  It matches the value specified after
;#simulate in a command in the [SimTools] section
[Simulate2]
;Command is the entire command line to start the simulator
   Command = silos3w
;Waves is the entire command line to start the Waveform Viewer
   Waves =
;SimAppName is the name the simulator uses to identify itself to the Navigator
   SimAppName = silos3
;ExitMsg is the command to tell the simulator to quit
   ExitMsg = quit
;FirstMsg is the name of the section that contains the messages to send to
   FirstMsg = FirstSilos2

;We have several commands to send to the simulator when it starts so we add
;[FirstSilos2].  the name "FirstSilos2" is arbitrary.  It matches the value
;specified for the FirstMsg entry in the section that starts the simulator.
[FirstSilos2]
;the values on the left side of the equal sign are not used but they must
;be unique within the section
;we use numbers just for simplicity
   1 = RESET ALL
   2 = CON .CUSTREPORT .SYN=0 .SAV=2
;the LIB command points to where the library files are stored on this machine
;rather than hard code path names, these could be specified in environment
;variables...
   3 = LIB .\common{.v} .\symflat{.vi} .\top{.v} .{.v} E:\test\bliflib.v
   4 = File .SAV=&B
   5 = File .STO=&B.rep
   6 = Input &B.tf
   7 = Sim 0
```

```
[SimControls]
;MaxLine may not exceed 1024
   MaxLine = 512
   Terminator =
   Separator = ,
   FlattenBusses = Yes
;Silos uses left paren as its hierarchy separator
   HierChar = (
;In our examples we have named the top module "t" and the instance of our
;design "m".  So we need the following prefixes:
   RootPrefix = (t(m(
   RootIOprefix = (t(
   InstPrefix = (t(m(
   SubPrefix = (t(m(
;Silos requires that the globals be specified outside of any module.
;to make this happen we must set GlobalPrefix to blank
;normally GlobalPrefix would be the same as RootIOPrefix
   GlobalPrefix =
;verilog does not allow brackets in instance names so we must set InstParen
;to underscore.  Both the left and right brackets [] from names of iterated
;instances will be converted to underscores.
   InstParen = _

;to send commands to the simulator we must have a [SimMenu] section
[SimMenu]
    Set = SET %n = %{ 0 1 X Z}
    Force = FORCE %n = %{ 0 1 X Z}
    Release = FREE %n
;Simulate doesn't work well.  the user must type end the end time
    Simulate to: = Sim to ?{}
    Stop = ^C
    Finish = QUIT

;to get the Navigator to bring up source files when the user clicks on
;primitive symbols, we add the [SimSources] section.
[SimSources]
;we leave the right side of the equal sign blank.  If our source files were
;not in the same directory with the symbols we could set the right side of
;the equal sign to the relative or absolute path of the sources
   .abl =
   .v =
   .vf = vf\
   .foo = c:\foo\

;[Export] is for the waveform Editor and must have only one line
[Export]
   exp -verilog -ext=.tf
```

Verilog Command Line

Generally, the Verilog -f option is used to include most of the command line information that is needed by Verilog. The commands can also be included in the **verilog.ini** file itself. The command line typically contains:

The command to start Verilog

```
/verilog
```

or if using a remote host

```
rsh remote_host /verilog
```

A password option:

```
-p1234abcd
```

The option to include a commands file:

```
-f /cmds_file
```

The option to run Verilog in interactive mode:

```
-s
```

The netlist file for the design to be simulated:

```
&R.v
```

```
and the option to include a library path to be scanned:
```

```
-y /lib +libext+.v
```

The paths (/) to the commands file and the netlist files must be from the view of the host running Verilog. The directories containing these files must therefore be mounted on the host. See your network administrator for assistance in determining the correct way to access these files.

Building Verilog Symbol Libraries

Verilog has several built-in primitive functions used in simulation. Several attributes in these primitives must take prescribed values. The following list gives the attributes needed for various Verilog primitives.

Combinational Gates

VeriModel= one of AND, NAND, OR, NOR, XOR, XNOR, BUF, NOT
Polarity= OUT on the output pins
Polarity= IN on the input pins

Bidirectional Pass Gates

VeriModel= one of TRAN RTRAN
Polarity= BI on both pins

Tri-State Bidirectional Gates

VeriModel= one of TRANIF0, TRANIF1, RTRANIF0, RTRANIF1
Polarity= BI on the two main pins
Polarity= IN on the control pin

Tri-State Drivers

VeriModel= one of BUFIF0, BUFIF1, NOTIF0, NOTIF1
Polarity= OUT on the output pins
VeriName= or PinName= IN on the input
VeriName= or PinName= EN on the enable pin

MOS Gates

VeriModel= one of NMOS, PMOS, RNMOS, RPMOS
VeriName= or PinName= D, G, S

CMOS Gates

VeriModel= one of CMOS, RCMOS
VeriName= or PinName= D, GN, GP, S

Pullup & Pulldown Gates

VeriModel= one of PULLUP, PULLDOWN
Polarity= OUT on the pin

Models — User-defined Primitives

VeriModel= Name of the primitive
Polarity= OUT on the output pin
Polarity= IN on the input pins
VeriName= User-defined name

The pin names of the user-defined primitives (or models) are netlisted in alphabetical order. The pin name matching is done by name, not by sequence.

VeriOrder Attribute

The VeriOrder attribute changes the pin ordering in the produced netlist.

To change the default pin ordering:

Set the VeriOrder pin attribute of the pins you want to re-order to any sequential numbers. When you produce a netlist, the pins will be output in the order you specified.

Example

The following example is a model for dummy symbol with a bus.

Symbol Attributes

VeriModel= DUMMY

Pin Attributes

Pin Name	VeriName	Polarity
OUT		Out
CLOCK		In
DATA[0:3]		In
RESET		In

In the above example the pins are listed as follows:

CLOCK, DATA[0:3], OUT, RESET

Netlist Extraction

The Code Verilog commands in the Hierarchy Navigator are enabled by setting the Simulator option to "verilog" using the INI editor. A file named **verilog.ini** must exist in either the working directory, or in the CONFIG directory specified in the Windows Registration Editor, or the ROOT**data** directory (under UNIX/Motif).

Selecting Code Verilog from the Simulate menu of the Hierarchy Navigator creates a set of structural module definitions for each schematic linked into the hierarchy, including the top level.

Selecting Verilog Netlist from the Tools Menu of the Schematic Editor creates a complete structural module definition from the schematic and the associated symbol file.

Selecting Verilog Netlist from the Tools Menu of the Symbol Editor creates a template module definition from the symbol only if an associated schematic is not available. The intent is for the user to complete the behavioral description in text mode.

All three versions of this function create (or update) files with the base name of the schematic or symbol and a **.v** extension.

Controlling The Netlist Extraction

The following parameters and controls let you customize the netlist extraction.

VeriModel Attribute

The VeriModel symbol attribute is the most useful way to control Verilog Netlist extraction. When the netlist processor encounters a symbol that has the VeriModel attribute specified, it normally codes that model into the netlist with the specified VeriName pin attributes (if any), without looking for any lower-level schematics.

If the VeriModel attribute is not specified, the netlister ignores the VeriName pin attributes of that symbol and codes the lower-level schematics associated with that symbol. The VeriModel and VeriName attributes are used only to code the module instantiations (that is, calls to modules) and not module declarations.

-noprims Command Line Option

The -noprims command-line option of the netlister causes it to skip the test for the Verilog primitive names and to code all instances as normal modules, even if they are Verilog primitives.

For example, assume a design contains a schematic with a symbol called NAND2 and the VeriModel attribute of this symbol is NAND. An underlying schematic exists for the symbol NAND2 containing transistor symbols NMOS and PMOS (VeriModel = NMOS and PMOS). NAND2 is normally coded with the NAND primitive name without any reference to the underlying schematic. With the -noprims option, NAND2 is coded as a subcircuit using n-channel and p-channel transistor switch level elements.

Existing Verilog Files

In some cases, template module definitions are generated from symbol files, which are then completed with user-defined Verilog behavioral models or stimulus. In order to preserve this user-defined data the next time the file is updated (after a change in the number of pins of the symbol, for example) the Verilog netlister replaces only the header comment, the module statement, and any lines that identify the polarity of the pins in the module statement. If a schematic is associated with the symbol under consideration, a completely new file is generated each time, and any manually entered data is discarded.

Pin Ordering

The Verilog netlister uses explicit naming to instantiate modules. The symbol files are used to extract pin names for module declaration lines as well as instantiation lines. In both cases, the pins are sorted alphabetically without regard to their polarity (Input, Output, Bidir). In the case of module instantiations, each pin is matched with the connected net before the sort.

To change the default ordering:

Set the VeriOrder pin attribute of the pins you want to re-order to any sequential numbers. When you produce a netlist, the pins will be output in the order you specified.

Vectors

Vectors with their bits spread over many pins are grouped together. For example, for the pin assignments below

```
|
|----o DD[0:3]
|----o A,B,DD[4:5]
|----o DD[6]
|----o DD[8]
|
```

the vector DD[0:8] will be used as the port. (Note that bit DD[7] is generated, even though it is not used.) It is the user's responsibility to insure that all the bits of a vector are coded with the same polarity. The netlister randomly selects one of the pins to determine the polarity of the vector.

For module instantiations, vector pins are coded with the vector name and the list of signals connecting to the vector. If any bits are not connected, either because of an unconnected pin or an embedded bit that was not on a pin, an extra comma is inserted to the netlist. When a pin's netlist includes one or more commas, it is surrounded by curly braces **{ }** inside the parentheses **( )**. If there is a mismatch between the number of buses and the number of bus pins, the extra signals are dropped (if there are too many signals) or the remaining bits become unconnected (if there are too many pins).

Running Verilog (UNIX/Motif)

To simulate, the netlists are first combined with a user-supplied stimulus file that specifies the initial values, clock, and pattern definitions for the simulator. Then the Verilog Simulator is started from the Simulate menu of the Hierarchy Navigator. This displays the simulator window, as well as the Waveform Viewer to view the simulation results dynamically:

1. Start the Hierarchy Navigator on the top level of your design. Verilog appears on the menu bar. If Verilog does not appear, the setup is not correct or the **verilog.ini** file was not found in either the working directory or in the ROOT**data** (ECS_ROOT/**data**) directory.

2. Select the Start Simulator command in the Verilog menu. The Simulator Control window opens showing the 3 buttons (Run, Stop and Delta). Below the buttons is a window showing the command:

   `$ecs_init (192.9.200.`*`nnn`*`);`

   where *nnn* is the Internet node address.

   The Verilog sign-on messages appear. Any problems starting Verilog are reported. You will eventually see two Verilog prompts:

   ```
   C1 > C2 >
   ```

   If the -waves option was requested, the Waveform Viewer runs at this time.

You are now prepared to run Verilog. The menu changes to show the commands specified in the **verilog.ini** file. Use the Hierarchy Navigator menu and the control buttons on the Simulator window to control the Verilog program. Commands not included in the menu can be entered directly into Verilog by clicking the mouse in the control window and typing the command.

VHDL Interface

The Synario Capture System provides the means of including the VHDL Netlister (vhdl) and Simulator in the design environment. The interface between SCS and VHDL consists of three parts:

Netlist Extraction — The Synario Capture System can extract the structural module description from the schematic. It also has the ability to extract a template description from the symbol that represents a behavioral model.

Simulator Control — The VHDL simulator can be launched and controlled from the Navigator. Various commands can be defined in which the command is selected from menu and the elements (such as nets) are selected graphically from the schematic.

Output Viewing — The output of the simulator can be viewed graphically with the Waveform Viewer (WAVES). The simulation results can be viewed as the simulation progresses (dynamic mode) or after the simulation is completed (static mode) by using a history file. The Waveform Viewer can either run alone (stand alone) or in conjunction with the navigator. In the latter case, the schematic can be back annotated with the states of signals as defined in the simulator.

The scope of this document is to describe the Netlist extraction process using the VHDL Netlister. For details on setting up the simulator refer to the *Simulation Environment* section. Many SCS terms are called by a different name in VHDL. Some common terms, along with their VHDL counterpart, which will be used in this document are shown in the table below:

SCS Term	VHDL Equivalent
Pin	Port
Net	Signal
Symbol	Entity
Polarity	Mode
Bus	Array of Signals
Bi-directional	Inout
Symbol Attributes	Generics
Pin Attributes	Port Declaration

Setting up the VHDL Environment in SCS

The VHDL Netlister is invoked from the **Simulator** menu of the Hierarchy Navigator for hierarchical netlists. It creates a top-level netlist file with the name *design_name.vhd* where *design_name* is the top-level name of your design hierarchy, and individual netlist files for sub-level blocks. The VHDL Netlister can also be invoked from the **Tools** menu of the Schematic Editor and the **Tools** menu of the Symbol Editor to create VHDL netlists of individual schematics or modules.

Setting up the SCS.ini File

In order to properly generate the VHDL code, some initial setup is required in the **ecs.ini** and **vhdl.ini** files. The master copy of these files is located in the $ECS_ROOT/data directory. Copies of these files can also be placed in the local directory. The values in the local directory override those set in the master files. The ecs.ini configures the SCS tools and allows the customization of various elements in the SCS environment. Refer to Chapter 9 for more details about the ecs.ini file. The vhdl.ini file is described later in this section.

The Simulator parameter in the **ecs.ini** file enables the Simulator menu of the Hierarchy Navigator with the name and options of the selected simulator.

To set up the VHDL menu:

Enter **VHDL** for the Controls: Simulator parameter.

In SCS, the mechanics for associating information with graphical objects is through attributes. These attributes can be defined for symbols, pins and nets with each attribute having a name and a value. By assigning values to the required attributes, the port names, types, generics and default values for each entity (symbol) can be defined. Some attributes are reserved for use with the VHDL interface. The following sections and attributes of the ecs.ini file are used by the VHDL Netlister.

Description of Symbol Attributes

[SymAttrNames}

InstName = !0,0
Type = !1,1
RefName = !2.2
VHDL_Cfg = +78
VHDL_Use / Lib = -79
VHDL_Model = -80
VHDL_Gen1 = 81
VHDL_Gen2 = 82
VHDL_Gen3 = 83
VHDL_Gen4 = 84
VHDL_Gen5 = 85
VHDL_Gen6 = 86
VHDL_Gen7 = 87
VHDL_Gen8 = 88
VHDL_Gen9 = 89

VHDL-cfg (#78)

Defines the entity / architecture pair of the configuration name to be used in the configuration section of the netlist, instead of the default. This value is typically assigned during schematic editing, where it might be required to override the default configuration for a particular symbol. If left blank, then either the model name (attribute #80), if defined, or symbol name will be used. For example,

```
VHDL_Cfg = configuration Work.cfg
   VHDL_Cfg = entity Work.and (A)
```

VHDL_Use/Lib (#79)

Specifies the section to be copied from the vhdl.ini file (in place of the section specified by the *use_lib_section* parameter). For example,

```
VHDL_Use/Lib = Mylib_Defaults
```

VHDL_Model (#80)

Indicates that the symbol represents a precompiled model whose component instantiation statement is defined in one of the packages. This attribute therefore determines if a component instantiation statement needs to be written in the VHDL netlist. For example,

```
VHDL_Model = orgate    specifies that the current symbol
represents orgate which is already
compiled into one of the libraries
```

VHDL_Gen*n* (*n* is 1 to 9) (#81-89)

This is the mechanism for defining generics symbol attributes) for an entity (symbol). Any of these attributes that contain values are copied "as is" into the generic declaration of that entity. The format for the value field is *param:type:=value*. For example,

```
VHDL_Gen1 = delay:time:=0 ns (In symbol editor)
```

These attributes can be overriden in the Schematic Editor where the symbol is instantiated. The format of the overriding values should be: *name=value*. For example,

```
VHDL_Gen1 = delay = 10ns
```

If none of these attributes is assigned a value, generic declarations are not coded. If more than 9 generics are needed then multiple generics can be specified in the same attribute by separating them with semicolons (;).

Description of Pin Attributes

```
[PinAttrNames]
   PinName = -0     ;Name of Signal on Symbol Pin
   Polarity = -1    ;Specifies Input, Output or Bi-directional
Polarity
   VHDL_PinType = -30      ;Port Type if Scalar Port
   VHDL_BusType = -31      ;Port Type if Vector Port
   VHDL_Def_Value = +32    ;Default Value for Port
   VHDL_NetConv = +33      ;Type Conversion Function for Scalar Port
   VHDL_BusConv = +34      ;Type Conversion Function for Vector Port
   VHDL_PinUse = -35       ;Polarity of the Port (Buffer or Linkage)
```

VHDL_PinType #(30)

Specifies the VHDL type of port (pin) for scalar objects. If left blank, this value is taken from the *Defaults* section of the vhdl.ini file. For example,

```
VHDL_PinType = BIT
VHDL_PinType = MVL7
```

Note: *If the net attributes VHDL_NetType is blank, the pin attributes value is used.*

VHDL_BusType (#31)

Specifies the VHDL type of the port (pin) for vector objects. If left blank, this value is taken from the Defaults section of the vhdl.ini file. For example,

```
VHDL_BusType = Bit_Vector
VHDL_BusType = MVL7_Vector
```

Note: *If the net attributes VHDL_BusType is blank, the pin attributes value is used.*

VHDL_Def_Value (#32)

Sets a default value for the port (pin) in the entity (symbol) declaration when the value is assigned in the Symbol Editor. If no default value is needed, this argument can be left blank. For example,

```
VHDL_DefValue = '0' ;
VHDL_DefValue = X"00";
VHDL_DefValue = "ZZZ"
```

VHDL_NetConv (#33)

This attribute is assigned a value when the pin is connected to a signal of a different type. It specifies the VHDL type conversion function to be associated with the port (pin), if it is scalar. For an input pin, the function is associated with the signal it connects to in the schematic, for an output pin the function is associated with the signal in the port map. This value can be specified on a per instance basis in the schematic. For example,

```
VHDL_NetConv = bit2mvl7 ;
VHDL_NetConv = mvl2bit
```

VHDL_BusConv (#34)

Specifies the VHDL type conversion function to be associated with the port if it is a bus. The rules of the association are the same as described above for the scalar pins. For example,

```
VHDL_BusConv = bv2mv ;
VHDL_BusConv = mv2bv
```

VHDL_PinUse (#35)

Specifies the mode (polarity) of the port (pin) other than IN, OUT, INOUT. Could be either buffer or linkage (the only two legal values). To make a *buffer* port, the I/O marker on the schematic must be a BIDIR and the matching pin on the schematic must also be bi-directional. The symbol pin must have this attribute set to *buffer*. With this combination of attributes, the port netlists as BUFFER. The *buffer* value can be assigned to a single pin attached to a single net. It can not be assigned to a Bus Port. For example,

```
VHDL_PinUse = Buffer ;
```

In addition to the attributes mentioned above, the following optional sections can contain directory path names to define symbol libraries:

```
[SymbolLibraries]

$ECS_ROOT/your_lib_path = Yes
```

The remaining parameters for the VHDL simulation environment are kept in the **VHDL.ini** file in the *$ECS_ROOT/*data directory. Each design can have a separate **VHDL.ini** file in its local directory.

Description of Net Attributes

```
[NetAttrNames]
   VHDL_NetType = -30   ;Port Type if Scalar Port
   VHDL_BusType = -31   ;Port Type if Vector Port
```

VHDL_NetType #(30)

Specifies the VHDL type of signal for scalar objects. If left blank, the pin attributes are used. If the pin attributes are blank, this value is taken from the *Defaults* section of the vhdl.ini file. For example,

```
VHDL_NetType = BIT
VHDL_NetType = MVL7
```

VHDL_BusType (#31)

Specifies the VHDL type of the signal for vector objects. If left blank, the pin attributes are used. If the pin attributes are blank, this value is taken from the Defaults section of the vhdl.ini file. For example,

```
VHDL_BusType = Bit_Vector
VHDL_BusType = MVL7_Vector
```

Adding a Tools Menu Entry to the Schematic Editor

The VHDL Netlister can be run as a **Tool** from the Schematic Editor, independent of the Simulator parameter. The program generates a module netlist of the schematic block. A symbol for the schematic has to exist in order for the netlist to be written since the module declaration is extracted directly from the symbol files. This gives the user independent control of module information and its underlying netlist. Note that the symbol for a schematic can be created from within the schematic using the File--Create Symbol command.

Adding a Tools menu Entry to the Symbol Editor

The VHDL Netlister can be run as a **Tool** from the Symbol Editor, independent of the Simulator parameter. The program generates a module statement for the symbol being edited.

Command Line Options

The following options are used with the VHDL Netlister:

Option	Function
-s	Suppress the generation of the configuration statements
-t	Generate a VHDL testbench template for a given symbol or schematic
-n	Run the default text editor (specified in the Editor field of the ecs.ini) after the VHDL code is generated
flat	Flatten buses when generating VHDL
-e	Do not output entity lines in the generated VHDL

VHDL Testbench Template

In VHDL, a *testbench* is the code that generates the input test stimulus and optionally performs the expected response verification. The -t option of the Netlister provides a template for the testbench (minus the behavioral description) as a convenience to the user.

Setting up the VHDL.ini file (for use with Hierarchy Navigator)

The Hierarchy Navigator can be configured to control the VHDL simulator. The first step is to specify the Simulator parameter to be VHDL using the ecs.ini editor. This causes the Navigator to read the VHDL.ini file, which must be in either the local directory or in the $ECS_ROOT/data directory. The **VHDL.ini** file contains all the necessary commands and directives to control the VHDL simulator. Note that empty sections (that is, sections without any contents) are not allowed. However, there may be cases when library and use statements are not needed at the top of each .vhd file generated by the Netlister. In such cases, a section may be defined consisting only of the VHDL comment characters (--), as shown:

[None_Default]

--

--

A sample **VHDL.ini** file along with a brief explanation is shown below. (Refer to Appendix B, "Simulator Interfaces" for more details on the topic of setting up the simulator).

Description of Default Control Parameters in vhdl.ini

These parameters are used in the [Defaults] section of the vhdl.ini file.

net_type

Default VHDL type of scalar nets and pins.

For example,

```
net_type = mvl7
net_type = bit
```

bus_type

Default VHDL type of vector pins and busses. If the bus_type is terminated with a (), a range is added to the declaration when VHDL netlists are extracted (that is, the type is assumed to be unconstrained).

For example,

```
bus_type = bit_vector
bus_type = MVL7_VECTOR
```

work_lib

Default name of the library where the configurations are found. This parameter is used while generating configuration statements in the VHDL netlist.

For example,

```
work_lib = work   (Generates the use statement in the
cfg.declaration as:  use configuration
WORK.cfg_symbol).
```

model_lib

Default name of the library where the configuration of the model is found. This is also used while generating configuration statements in the VHDL Netlist.

For example,

```
model_lib = Your_Own       (Generates the use statement in the
cfg.declaration as: use entity
Your_Own.symbol(arch)).
```

use_lib_section

Default name of the section in the vhdl.ini file, to be copied to the top of each VHDL netlist. Typically this section contains library and use packages information.

For example

```
,  use_lib_section = MyOwn_Defaults
```

This section copies information for the section of the vhdl.ini file marked as [MyOwn_Defaults] to the beginning of each VHDL netlist. From our example, that would produce:

```
library Mylib;
use Mylib.types.all;
use Mylib.components.all;
```

tb_entity

Default entity name to be used in the automatically generated test bench template. If no value is specified, "E" is used.

For example,

```
   tb_entity = tb_andor
   tb_entity = tb_toplevel
```

tb_arch

Default architecture name of the test bench entity, to be used in automatically generated test bench template. If no value is specified, "A" is used.

For example,

```
   tb_arch = tb_struct
   tb_arch = tb_behavioral
```

tb_inst

Default instance name of the top level schematic to be used in the automatically generated test bench template. If no value is specified,"UUT" is used.

These attributes are assigned to the symbol and its pins by use of the **Add->Symbol Attribute** and **Add->Pin Attribute** commands of the Symbol Editor. If a symbol has an attribute assigned, it becomes the default value for each instance of the symbol on the schematic. Note that the VHDL Netlister writes a hierarchical netlist. Because of this, any instance specific attribute overrides added by the Hierarchy Navigator do not appear in the netlist. Please refer to the *Attributes* section in this manual for more details on attributes in the SCS environment.

ModelAttribute = 80

This parameter specifies the number of the symbol attribute used to store the VHDL model names. This parameter is used extensively by the Hierarchy Navigator and the VHDL Netlister, and it must be included in the **VHDL.ini** file. See also the VHDL_Model attribute defined in the ecs.ini file.

ECS Only Controls (UNIX)

FlattenBusses = No

This parameter causes busses not to be flattened when they are monitored from the Hierarchy Navigator. The same value should be assigned to this parameter as the command line option.

RootPrefix, RootIOPrefix, InstPrefix, SubPrefix = %

These parameters cause the system to use fully hierarchical net and instance names instead of local names. They must be included in the **VHDL.ini** file in order to successfully monitor and cross-probe nets between the Hierarchy Navigator and the Dynamic Waveform Tool.

Run = .Delta = #% $ecs_stop;.

The **$ecs_start** command has been eliminated. Thus the Run and Delta commands do not include that VHDL function call.

Monitor = $ecs_waves ("%n")

The argument of the **$ecs_waves** function call has to be a string. Note that the *%*n* notation has been changed to *%n* when this task is called.

Netlist Extraction

The **Code VHDL** commands in the Hierarchy Navigator are enabled by setting the Simulator option to *VHDL* using the ecs.ini editor. A file named *VHDL.ini* must exist in either the local directory or in the *$ECS_ROOT/data* directory.

Selecting **Code VHDL** in the Simulator menu of the Hierarchy Navigator creates a set of structural module definitions for each schematic linked into the hierarchy including the top level. If, however, a structural module already exists and the schematic has not been modified since, it is not recompiled.

Selecting **VHDL Netlist** in the Tools Menu of the Schematic Editor creates a complete structural module definition from the schematic and the associated symbol file. Note that a symbol for the schematic must exist so that the module netlist can be completed, since the module declaration of the block is extracted from the symbol file.

Selecting **VHDL Netlist** in the Tools Menu of the Symbol Editor creates a template module definition from the symbol *only if* an associated schematic is not available. The intent is that the user then completes the behavioral description in the text mode.

All three versions of this function create (or update) files with the base name of the schematic or symbol and a *.vhd* suffix.

Generating VHDL Netlist from Symbols

The Netlister produces a VHDL template (that is, a VHDL description without the architecture body) from symbols. The following steps describe the process of netlist extraction from a symbol named *abc*.

A netlist file called abc.vhd is opened and the following information is written to it.

Library and Package Declaration

The library declarations are coded first. If attribute #78 (VHDL_Use/Lib) of the symbol has a value, then that value is matched to the section names in the vhdl.ini file and the contents of the section are copied into the netlist. If no value has been assigned to attribute #78, the contents of the section specified by the value of parameter *use_lib_section* in the vhdl.ini are copied.

Entity Declaration

The name of the entity is the same as the name of the symbol. The entity declaration consists of generic declaration (symbol attributes) and port declarations (pin attributes).

Generic Declaration(s)

If any of the symbol attributes in the range of 81-89 (VHDL_Gen1 - VHDL_Gen9) has a value, then the generic declaration is written. The values of these attributes is copied verbatim.

Port Declaration(s)

Port names are taken from pin attribute #0 (PinName). If the port is a scalar, the pin name is taken as it is. If the pin name is a vector, then the range description is stripped from the name.

Port mode is taken from pin attribute #35 (VHDL_PinUse) if the mode is buffer or linkage or pin attribute #1 (Polarity) if the mode is In, Out or Inout.

Port type is taken from either pin attribute #30 (VHDL_PinType) for scalar port or pin attribute #31 (VHDL_BusType) for vectors.

Architecture Declaration

An empty architecture body is coded with default architecture name as *behavioral*. Anything you type inside the architecture body is retained during the next invocation of the Netlister for the same symbol.

Configuration Declaration

Default configuration statements are written with the configuration name cfg_symbolname.

Generating VHDL Netlists from Schematics

The process for generating Netlists from schematics is similar to the process previously described for symbols. The major difference is that in schematics, in addition to the entity declaration there are component statements and component instantiation statements.

To generate a VHDL description from a schematic called *xyz*:

1. A netlist file called xyz.vhd is opened and the following VHDL code is written.
2. A library declaration and an entity declaration similar to the one described for the symbol are written.
3. An architecture declaration is written and it consists of:

   Signal Declaration

Signal names are taken from the netlists used in the schematic. If the net is a bus, the range is stripped from the name. The signals inherit type from the pin they are connected to. If a signal connects to pins of different types, the type is taken from one of the pins (chosen arbitrarily) that does not have a type conversion function specified, the signal type is taken from the value of the parameter net_type or Bus_type in the vhdl.ini file.

Component Declaration

A component statement is written for each symbol in the schematic that does not have a value assigned for symbol attribute #80 (VHDL_Model).

Architecture Body

An architecture called *schematic* is written. The architecture body of the schematic consists of component instantiation statements for each of the symbols drawn on the schematic diagram. The name of each component instance is taken from the name assigned to the symbol on the schematic. The generic map statement is written if needed. The port map statement is then written.

4. A configuration file called *cfg_schematic_name* is written.

For instances of the same symbol, the list of instance names are written followed by the model name (attribute #80) or the symbol name. If configuration is specified for the instance (that is, if symbol attribute #78 for the instance has been assigned a value), then it writes

```
use (attr#78)
```

If the symbol is a precompiled model, then it writes

```
use entity model_lib.(attribute#80)(A)
```

where the value of model_lib is taken from the vhdl.ini. If it is not a model, then it writes

```
use configuration work_lib.cfg_symbolname
```

where the value of work_lib is taken from the vhdl.ini file.

Generating VHDL Netlists from the Navigator

The VHDL Netlister program can also be invoked from the Navigator to generate the VHDL code for the schematic. Optionally, a testbench template can also be generated for the root on the schematic. The advantage of invoking the VHDL Netlister from the Navigator rather than from the Symbol/Schematic editor is that the Netlister generates the VHDL code not only for the root schematic, but also for the symbols within the schematic and all underlying hierarchy. Invoking the Netlister from the Navigator results in the following steps being performed:

1. The design is traversed in a depth first manner. Blocks specified as predefined models are not traversed.
2. For each block traversed, the directory specified by the Path control in the *ModelBuilder* section of the vhdl.ini is searched for the corresponding VHDL file. If the VHDL file is not found or if it is older than the symbol block, the VHDL file is regenerated for that block.

Special Topics (Buses, Resolution Function and Testbench Mode)

Some topics which require special attention while using the VHDL Netlister are discussed below.

Using Buses

Buses are used to represent a group of related signals in the SCS schematics. In the corresponding VHDL netlist, they are represented as a one dimensional array of signals. The following rules must be followed for the proper generation of VHDL code for busses.

1. Bus names on the schematic must have the following format:

   ```
   busname (from : to)
   ```

   Busnames of the format *Busname (from, to, step)* which are legal in *SCS* are illegal using the VHDL Netlister.

2. The bus pin type may be specified by the pin attribute #31 (VHDL_BusType). The value of this attribute must have the following format:

   ```
   bustype
   ```

   or

   ```
   bustype ( )
   ```

The parentheses are necessary if *bustype* is an unconstrained array type. However, if *bustype* is a user-defined, constrained subtype, then no parentheses are required.

3. The VHDL code generated for the port declaration is of the following format:

   ```
   busname
   ```

   or:

   ```
   busname (left to right)  if left < right
   busname (left downto right)   if left > right
   ```

   The range specification is added if bustype is an unconstrained array type (as indicated by the presence of parentheses in the value for pin attribute #31). The range is derived from the name of the pin and is added to the basetype (bustype), if needed. Examples of VHDL code generated by different type of pin attributes are shown below:

Pin Attributes	VHDL Code Generated
Pin_Name (Pin Attr. #0) : a (0:3)	a : in bit_vector (0 to 3) ; VHDL_ BusType (Pin Attr. #31) : bit _vector ( )
Pin_Name : a (7:0)	a : in BusX (7 downto 0); VHDL_BusType : BusX ( )
Pin_Name : a (7:0)	a: in byte ; VHDL_BusType : Byte

If no attribute is assigned to the pin attribute #31 (VHDL_BusType), the type is derived from the value assigned to the parameter *Bus_Type* in the *Default* section of the *vhdl.ini* file.

Using VHDL Type Resolution Functions

A resolution function is used to return the value of a signal when the signal is being driven by multiple drivers. It is illegal in VHDL to have a signal with multiple drivers without a resolution function attached to that signal. The resolution function will examine the values of all of the drivers and return a single value called the resolved value of the signal. The way to specify resolution functions for signals on a net by net basis in SCS is by defining a subtype to be a resolved type and then assigning the subtype to the desired pins. It is recommended that the resolved subtype be assigned to all the pins in the primitives so that the signals which connect to these pins automatically become the resolved types, provided the reflexive attribute is associated with the resolution function. This attribute is an assertion that the function has the property that when given an input vector with exactly one element, it returns that value.

Using Indexed Array Slices in PinNames

A pin on a symbol can be an indexed array slice of the form

slice_name (index)

If two or more pins belong to the same array, the VHDL netlister collapses them to generate a single port declaration. For example, if a symbol contains pins with PinNames as inp(0), inp(1), inp(2) and inp(3), the corresponding entity declaration in VHDL looks like:

```
port (input : in BusX (0 to 3))
```

The direction of the range of the bus is determined by the following rules:

- If at least one of the slices contain a range of the form

  `bus_name (left:right),  left > 0 , and  right > 0`

  the direction of that range is used for the entire bus.

- If the above rule does not apply, the direction specified in the VHDL_BusType attribute is used. You can specify this by inserting the character ’t’ for "to" and ’d’ for "downto" between parentheses in the VHDL_BusType (For example, BusX (d)).
- If no direction is specified, the direction is assumed to be ascending.

Automatic Testbench Template Generation

A VHDL testbench template can be generated for the root schematic by specifying the -t command line option for the VHDL Netlister program. The option is specified in the ModelBuilder section of the vhdl.ini file by making the following assignment:

```
RootProcess = vhdl -t %
```

The testbench consists of the following:

- The standard library / use statements.
- An empty entity whose name is specified by the tb_entity parameter in the *Default* section of the vhdl.ini file.
- An architecture, whose name is specified by the tb_arch parameter in the Default section of the vhdl.ini file and containing:
  - A signal declaration for each pin on the top level symbol
  - A component declaration for the symbol (with the same name as the symbol)
- A single instance of the component, whose name is specified by the tb_inst parameter in the Defaults section of the vhdl.ini file, with port mapping the ports to the signals.
- An empty block statement (labeled TB) for users to add test vectors if desired.
- A configuration statement for the testbench, named CFG_TB_COMP_NAME.

Structure of Output VHDL Files

VHDL Code Explanation

```
library your_own;| Specified by symbol attr. #79 or by
use your_own.types.all;| use_lib_section of the [Defaults]
use your_own.components.all;| section of the vhdl.ini file

entity AOI is| AOI is the name of the symbol/file
generic (N : positive : = 5 ;|This is specified in symbol
attrs. #81 - 89
tLH : Time : = 0 ns ;
tHL : Time : = 5 ns) ;
port ( In1 : in BIT : = '1' ;| Polarity specified by pin attr.
#1 or #35
In2 : in BIT : = '1' ;| '1' = Default; specified by pin attr.
#32
In3 : in BIT_VECTOR (2 downto 0);| InX are pin names specified
by pin attr. #1
In4 : in BIT_VECTOR (0 to 2);| BIT/BIT_VECTOR is VHDL type
| specified by pin attr. #30, 31
   Output : out BIT ) ;
end AOI;

architecture BEHAVIORAL of AOI is| Boiler plate code.  The
contents of this
begin  | section are preserved if/when VHDL code
end BEHAVIORAL;| is regenerated

configuration CFG_AOI of AOI is
for BEHAVIORAL

end for;

end CDG_AOI;|
```

Controlling the Netlist Extraction

The VHDL symbol attribute is the most useful way to control VHDL Netlist extraction. Normally, when the netlist processor encounters a symbol that has the VHDL attribute specified, it codes that model into the netlist with the specified VHDL pin attributes, if any, without looking for any lower level schematics. If the VHDL attribute is not specified, the netlister would ignore the VHDLpin attributes of that symbol and code the lower level schematic associated with that symbol. Note that the VHDL attributes are used only to code the module instantiations (that is, calls to modules), and not module declarations.

Existing VHDL Files

In some cases, template module definitions are generated from symbol files, which are then completed with user-defined VHDL behavioral models or stimulus. In order to preserve this user-defined data the next time the file is updated (for example, after a change in the number of pins of the symbol), the VHDL Netlister replaces only the header comment, the module statement and any lines that identify the polarity of the pins in the module statement. Note that if there is a schematic associated with the symbol under consideration, a completely new file is generated each time, and no manually entered data is preserved.

Some Restrictions

The maximum number of generics that can be declared for any symbol is nine. However, this limit can be overcome by defining multiple generics in a single symbol attribute separated by a semicolon (;). For example, the following string for the symbol attribute VHDL_Gen1 (attribute #81) is legal:

```
delay : time := 2 ns ; fanout : positive : = 8
```

Note that there is no semicolon (;) at the end of the string. The total number of characters in all the generics combined can not be more than 255.

Sample vhdl.ini File

```
;NOTE: This file is for running vhdl from hiernav

;Set the title of the Simulation menu in the hierarchy navigator
[SimTitle]
  title = VHDL

;we want to be able to write all the net lists from
;the Navigator using "make" so we add [MakeVhdl]
;the name "MakeVhdl" is arbitrary.  It matches the value specified after
;#Make in a command in the [SimTools] section
[MakeVhdl]
;need a process if we are going to make the net lists from the Navigator
   Process = vhdl &R
;we use the extension .vhd so we need the following line to let the
;Navigator know this.
   Extension = .vhd
;we want to put the .vhd files in the same directory with the symbols so
;we leave the Path blank.
   Path =
;the Vhdl Model attribute is number 80.  If attribute #80 has a value
;it means that the symbol is a vhdl primitive or has a .vhd file supplied
;by the user.  So, if we run make from the Navigator we must have the
;following line
   ModelAttribute = 80
```

```
;we want to be able to compile the net lists from
;the Navigator using "compile" so we add [CompileVhdl]
;the name "CompileVhdl" is arbitrary.  It matches the value specified after
;#Compile in a command in the [SimTools] section
[CompileVhdl]
   Process = zvan -nc &R
;the RootProcess runs only on the top schematic
   RootProcess = zvan -nc &Ptb&F
;the compiled files have extension .sim
   Extension = .sim
   ModelExt = .vhd
;we want to put the .sim files in the same directory with the
;symbols so we leave the Path blank.
   Path =
;the .vhd files are in the same directory with the symbols so
;we leave the ModelPath blank.
   ModelPath =
;the Vhdl Model attribute is number 80.  If attribute #80 has a value
;it means that the symbol is a vhdl primitive or has a .vhd file supplied
;by the user.  So, if we run compile from the Navigator we must have the
;following line
   ModelAttribute = 80

;the SimTools section directs the simulation interface to the appropriate
;information for generating VHDL models (using the built-in "make" facility),
;compiling the models, and starting the simulator.  Each of these functions
;is represented by an entry in the first section of the simulation menu
;of the hierarchy navigator
[SimTools]
;the next line invokes the "make" facility to write the vhdl net lists.
   Code vhdl = #Make MakeVhdl
;the next line invokes the "compile" facility to compile the vhdl net lists.
   Compile vhdl = #Compile CompileVhdl
;the next line invokes the "simulate" facility to start vhdl using the
;commands from "Simulate"
   Start vhdl = #Simulate Simulate

;we want to be able to start vhdl from the Navigator so we add [Simulate]
;the name "Simulate" is arbitrary.  It matches the value specified after
;#simulate in a command in the [SimTools] section
[Simulate]
;Command is the entire command line to start the simulator
   Command = dummysim cfg_tb_%
;Waves is the entire command line to start the Waveform Viewer
   Waves =
;SimAppName is the name the simulator uses to identify itself to the Navigator
   SimAppName =
;ExitMsg is the command to tell the simulator to quit
   ExitMsg =
;FirstMsg is the name of the section that contains the messages to send to
;the simulator when it starts up.
   FirstMsg =
```

```
[SimControls]
   MaxLine = 40
   RootPrefix  = /e/uut
   HierChar = /
   Continue = \

[SimButtons]
   Run = run
   Stop = ^C
   Delta = run

;Entries in the SimMenu section get transferred directly to the simulation
;menu in the hierarchy navigator.  The corresponding  commands are sent to
;the simulator when the menu item is selected.
[SimMenu]
   Monitor = Monitor %*n
   Trace=Trace %n
   Hold Val = Hold %{ ????? '0' '1'} %*n
   List All = List *

;The Defaults section specifies the default values for a number of parameters
;in the output VHDL
[Defaults]
   net_type = DotX
   bus_type = BusX
   work_Lib = WORK
   model_lib = ZYCAD
   use_lib_section = Myown_Defaults

;This section provides default header information included in all VHDL files.
;The name of the section is arbitrary and is specified by the "use_lib_section"
;entry in the "Defaults" section.  This information may be overwritten by
;the VHDL_Use/Lib symbol attribute (#79), if specified.
[Myown_Defaults]
   library Mylib;
   use Mylib.types.all;
   use Mylib.components.all;
```

Exporting the Stimulus File

You create a stimulus file for simulation by creating waveforms in the Waveform Editor and exporting into the stimulus format of the target simulator and written to the appropriate files.

To export to a stimulus file:

1. Create waveforms in the Waveform Editor.
2. Select Export from the Waveform Editor File menu.

See Also

"Exporting the Stimulus File" and "Changing the Waveforms for Exporting" in the *Waveform Tools Manual*

Index

A

B

C

D

I

O

P

Q

R

S

T

U

V

W