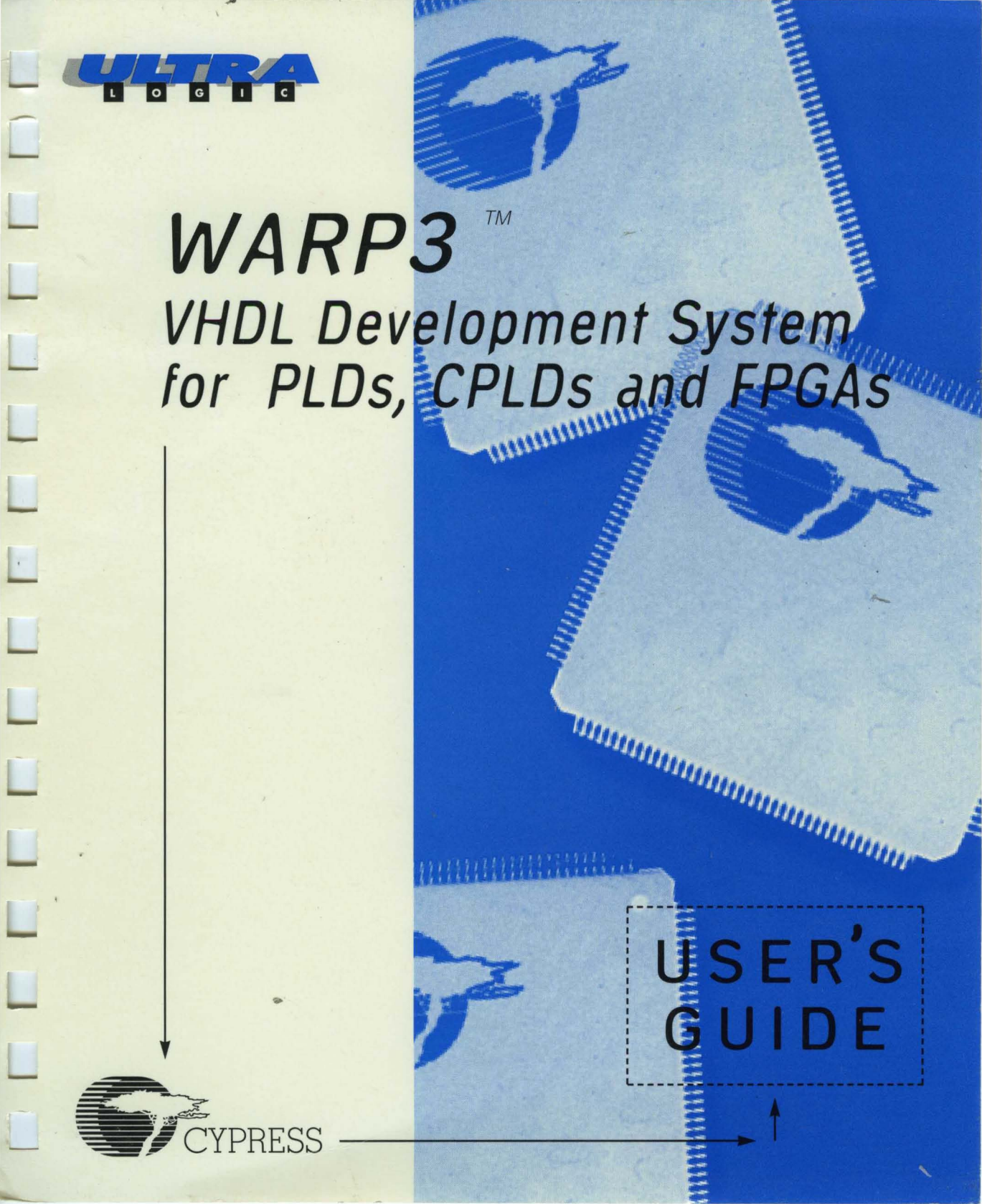

ULTRA
LOGIC
WARP3 ™
VHDL Development System
for PLDs, CPLDs and FPGAs
USER'S
GUIDE
CYPRESS

Warp3™

VHDL Development System

User's Manual

Cypress Semiconductor
3901 North First Street
San Jose, CA 95134
(408)943-2600
JANUARY 1995

Cypress Software License Agreement

1. LICENSE. Cypress Semiconductor Corporation ("Cypress") hereby grants you, as a Customer and Licensee, a single-user, non-exclusive license to use the enclosed Cypress software program ("Program") on a single CPU at any given point in time. Cypress authorizes you to make archival copies of the software for the sole purpose of backing up your software and protecting your investment from loss.

2. TERM AND TERMINATION. This agreement is effective from the date the diskettes are received until this agreement is terminated. The unauthorized reproduction or use of the Program and/or documentation will immediately terminate this Agreement without notice. Upon termination you are to destroy both the Program and the documentation.

3. COPYRIGHT AND PROPRIETARY RIGHTS. The Program and documentation are protected by both United States Copyright Law and International Treaty provisions. This means that you must treat the documentation and Program just like a book, with the exception of making archival copies for the sole purpose of protecting your investment from loss. The Program may be used by any number of people, and may be moved from one computer to another, so long as there is **No Possibility** of its being used by two people at the same time.

4. DISCLAIMER. THIS PROGRAM AND DOCUMENTATION ARE LICENSED "AS-IS," WITHOUT WARRANTY AS TO PERFORMANCE. CYPRESS EXPRESSLY DISCLAIMS ALL WARRANTIES, EXPRESSED OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE IMPLIED WARRANTY OF MERCHANTABILITY OR

FITNESS OF THIS PROGRAM FOR A PARTICULAR PURPOSE.

5. LIMITED WARRANTY. The diskette on which this Program is recorded is guaranteed for 90 days from date of purchase. If a defect occurs within 90 days, contact the representative at the place of purchase to arrange for a replacement.

6. LIMITATION OF REMEDIES AND LIABILITY. IN NO EVENT SHALL CYPRESS BE LIABLE FOR INCIDENTAL OR CONSEQUENTIAL DAMAGES RESULTING FROM PROGRAM USE, EVEN IF CYPRESS HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. CYPRESS'S EXCLUSIVE LIABILITY AND YOUR EXCLUSIVE REMEDY WILL BE IN THE REPLACEMENT OF ANY DEFECTIVE DISKETTE AS PROVIDED ABOVE. IN NO EVENT SHALL CYPRESS'S LIABILITY HEREUNDER EXCEED THE PURCHASE PRICE OF THE SOFTWARE.

7. ENTIRE AGREEMENT. This agreement constitutes the sole and complete Agreement between Cypress and the Customer for use of the Program and documentation. Changes to this Agreement may be made only by written mutual consent.

8. GOVERNING LAW. This Agreement shall be goverend by the laws of the State of California. Should you have any question concerning this agreement, please contact:

   Cypress Semiconductor Corporation
   Attn: Legal Counsel
   3901 N. First Street
   San Jose, CA 95134-1599

   408-943-2600

Table of Contents

Chapter 1

Warp3 Process Overview

About This Chapter

Overview

This chapter provides an overview of the design process using *Warp3*[1] tools.

1. *Warp3* is a trademark of Cypress Semiconductor.

1.1. What do you do with *Warp3*?

Warp3 is a collection of tools for designing, synthesizing, and simulating circuits to be transferred to programmable logic devices.

With *Warp3*, you can:

- use VHDL descriptions, schematics, or both to describe a design;
- compile and synthesize the resulting design description. ("Compile" means check the design description to make sure it's syntactically correct and contains no obvious logical errors, such as unconnected inputs, etc. "Synthesize" means realize the design description into logic circuits to be implemented in a PLD or FPGA);
- fit the resulting logic circuits in a particular PLD, or place and route the design in an FPGA. The resulting files may be used for programming the device;
- verify the design with a timing simulator.

There are other tasks you can perform, but at the highest conceptual level, that's it in a nutshell.

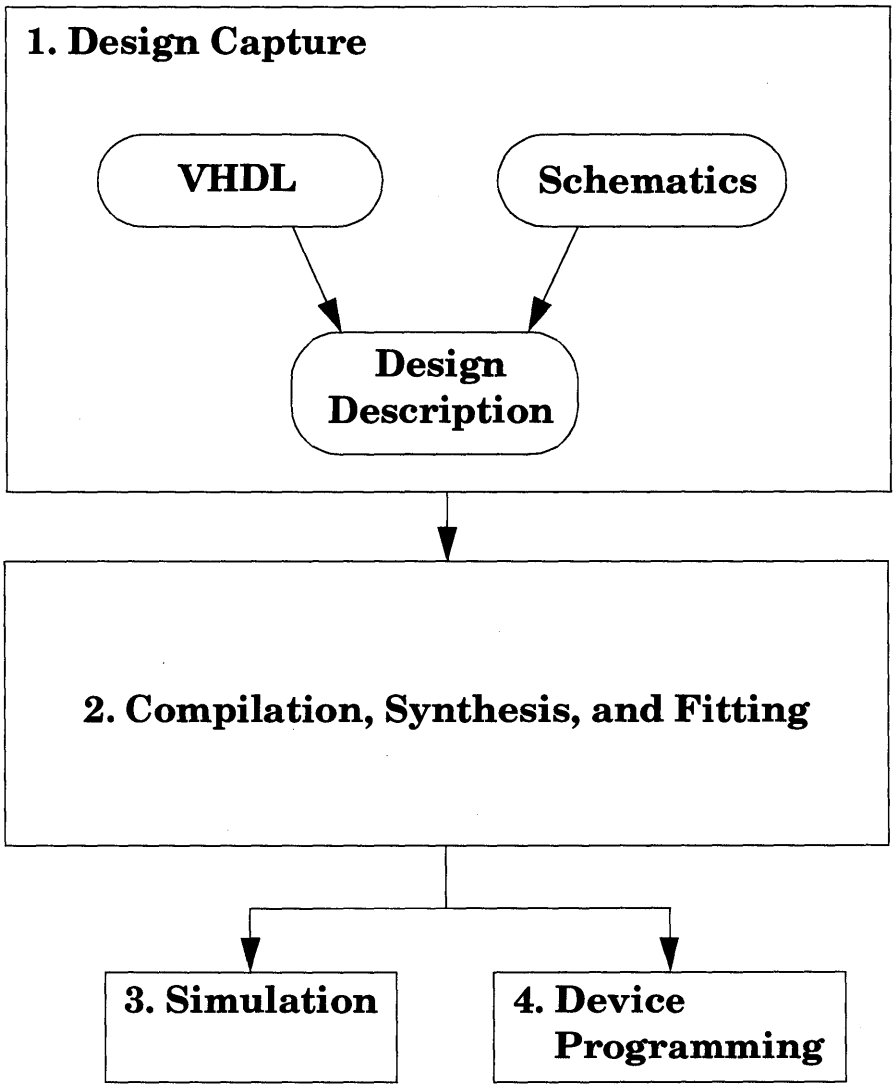


Figure 1-1. Warp3 Conceptual Process Flow.

1.2. What kinds of devices can I target using Warp3?

Warp3 supports a wide variety of Cypress programmable devices. *Warp3* features a common user interface so that the process of targeting one device is much like that of targeting any other.

Table 1-1 lists the parts for which you can target designs when using *Warp3*. These parts can be divided into five families:

- PLDs: the bulk of the Cypress programmable logic device family.
- MAX: the Cypress CY7C34X multiple-array matrix, high-density EPLD family.
- 37X: the Cypress FLASH370 high-density flash complex PLD family. Note that in Release 1.1, C372 and C373 are supported at the alpha level.
- 38X Programmable ASICs (pASICs[1]): the Cypress pASIC380 very-high-speed CMOS FPGA family.

1. pASIC is a trademark of QuickLogic Corporation.

Table 1-1.
***Warp3*-Supported Parts**

Family	Part #'s			
PLD's	C16L8	C16R4	C16R6	C16R8
	C16V8	C20G10	C20G10C	C20RA10
	C22V10	C22VP10	C331	C335
MAX CPLD's	C341	C342	C343	C344
	C346	C346H		
FLASH370 CPLD's	C371	C372	C373	C374
	C375			
pASIC380 FPGA's	C381A	C382A	C383A	C384A
	C385A	C386A		

1.3. What's the design process in *Warp3*?

The usual order of design in *Warp3* is as follows: (1) invoke the cockpit (project manager), (2) enter the design, (3) generate an IEEE1076 netlist (if necessary), (4) compile, synthesize, and fit the design, (5) place and route the design (FPGA's only), (6) create a timing model for the circuit, (7) simulate the design, (8) back-annotate pin assignment information, and (9) use the output files from the *Warp3* design flow to program the target device.

Project Management (Cockpit)

The cockpit provides an easy point-and-click interface for starting the design tools within the *Warp3* design flow. The Cockpit provides a means to manage your designs by specifying a current project. When you run tools from within the Cockpit, the current design name and project directory can be passed from one tool to the next.

The following paragraphs describe each of the steps in the usual order of design. Figure 1-2 diagrams the *Warp3* design process.

Design Entry (ViewDraw, VHDL)

In *Warp3*, you can describe your designs in VHDL, schematic, or a combination of both.

VHDL designs can be written behaviorally or structurally. Behavioral design descriptions make use of VHDL constructs which permit designs to be described in terms of algorithms or equations. Structural designs are netlists which instantiate components and connect them together, much like a textual description of a schematic. Viewdraw also allows VHDL descriptions to be converted to symbols (using Viewtext, started from Viewdraw) and included as component instantiations within the hierarchy.

Schematics can be entered in Viewdraw. VHDL designs as well as schematic designs can be hierarchical.

Generating a VHDL Netlist (EXPT1076)

This tool does not need to be run for designs that are described entirely as VHDL. The Warp synthesis engine is a VHDL compiler. Because Warp takes IEEE1076 VHDL as input, all schematics must be converted to VHDL by using the EXPT1076 tool. This tool converts schematics to a VHDL netlist (a structural VHDL description) and flattens the hierarchy.

Compiling, Synthesizing, and Fitting the Design (Galaxy)

Galaxy is the GUI (graphical user interface) for Warp. Warp takes VHDL as input, checks the design for proper VHDL syntax, and synthesizes the design to logic equations. (Synthesis is the realization of design descriptions into logic circuits).

For CPLDs, the logic equations produced by synthesis are used by the fitter which attempts to "fit" the logic into a particular device, using algorithms to determine how to optimally use the available resources and macrocell and I/O configurations. The output of the fitter is a JEDEC map used for programming the device after simulation.

For FPGA's, the logic equations produced by synthesis are translated into a netlist specific to the FPGA. This netlist describes the interconnection of logic elements that can map directly to logic cells of the FPGA.

Placing and Routing an FPGA (Place & Rte)

The Place and Route tool is used only for FPGA designs. As its name suggests, this tool uses the FPGA netlist from synthesis to place logic within logic cells, place the logic cells in the device, and route the necessary signals between logic cells and I/O. This tool uses algorithms for optimizing logic implementation within logic cells and for minimizing signal delays and resource utilization. A static timing analyzer is provided to evaluate worst-case delays for all paths and to specify constraints for a timing-driven place and route. The LOF file necessary for programming FPGA's can be exported from this tool

Creating a Viewsim Timing Model (pASIC->VSM)

pASIC->VSM is used only for FPGA designs. This tool converts a delay table and design description produced by the place and route tool to a Viewsim Timing model. Timing models for CPLDs can be produced from within Galaxy by selecting the appropriate option.

Simulating the Design (ViewSim)

Viewsim is used to simulate the design with full timing information. With Viewsim, you can verify the functionality of your design and determine if the design meets timing requirements. Viewsim provides an interface to Viewtrace, which displays waveforms and allows direct interaction with the simulator. Simulation values can also be annotated onto Viewdraw to assist in debugging designs. A sophisticated command language is available for running simulations.

Back annotating pin assignment (Cypback)

Cypback is used to annotate pin assignments onto your schematic after place and route. It is primarily used to establish pin assignments so that subsequent place and routes will use the same pin assignments.

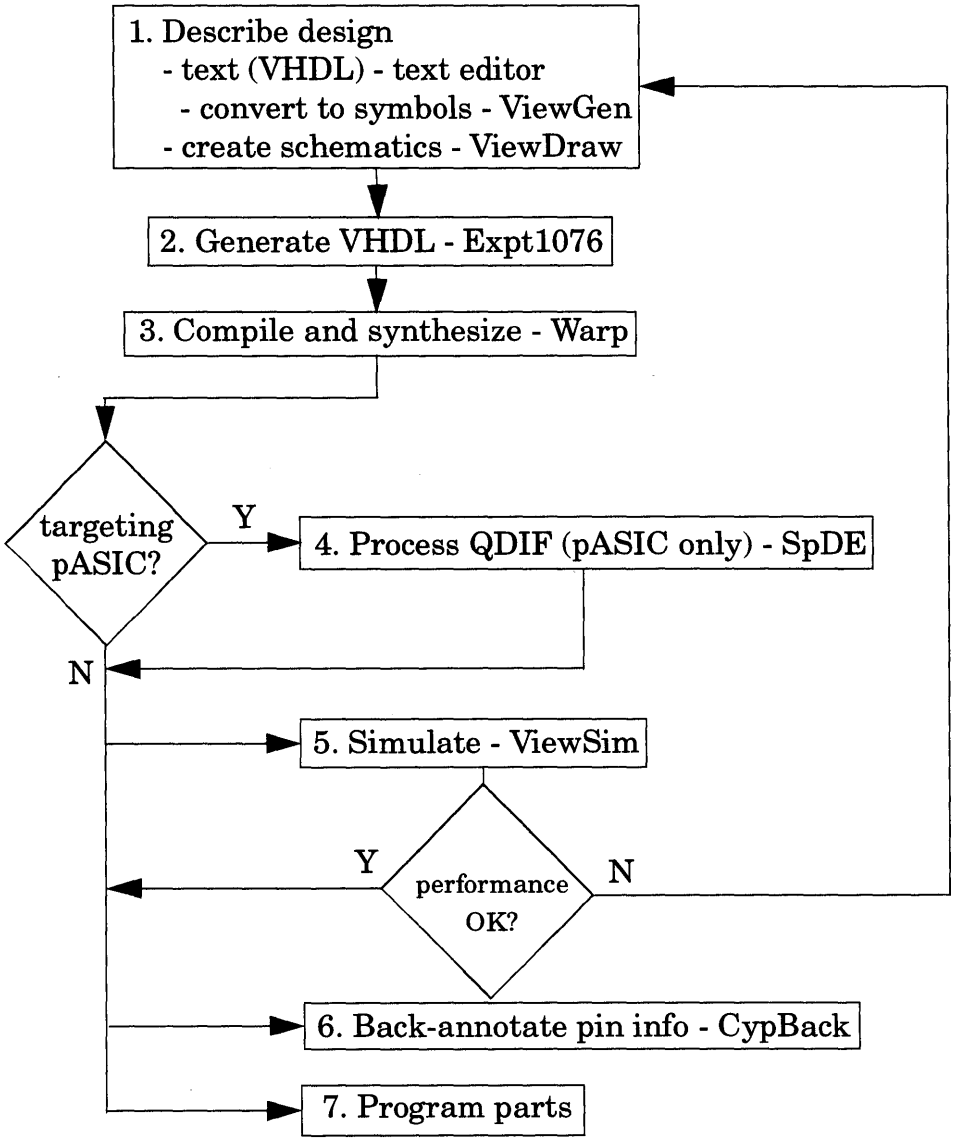


Figure 1-2. Warp3 Process Overview (Detailed).

1.4. What are library components?

Summary: Library components are directories in which descriptions and symbols of components are stored. A comprehensive set of parts libraries comes with *Warp3*.

Warp3 provides libraries containing the following types of components:

- adders
- multipliers
- counters
- common logic gates
- input/output components
- memory elements
- multiplexers
- registers
- shifters
- TTL components.

You can open the VHDL description of any library component for viewing with any text editor. The default location of the VHDL descriptions of library components is:

\WARP\LIB\COMMON*FAMILY*.VHD

where *family* is the generic name of a particular component group (e.g., COUNTER.VHD, GATES.VHD).

To instantiate a component in a VHDL description, use a VHDL component instantiation statement. See the description of components in Chapter 5 of the *Warp Synthesis Compiler Manual*.

For more information about the libraries, refer to the *Warp System Library Reference Manual*.

1.5. Where can I call if I have problems?

Cypress Semiconductor has offices around the world. If you need information about (or if you experience any difficulties using) any Cypress product, call one of the numbers listed below.

Country	Toll- Free Number	Country	Toll-Free Number
N. America	1-800-419-1481	Japan	0031-11-1731
Australia	0014-800-125-203	Korea	008-1-800-942-8203
Belgium	11-8729	Netherlands	06-022-5303
Denmark	8001-0413	Norway	050-12068
Finland	9800-10065	Singapore	800-1758
France	05-90-1251	Spain	900-99-1163
Germany	0130-81-1902	Sweden	020-795-637
Hong Kong	800-7214	Switzerland	045-05-8808
Israel	00-17-942-1803	UK	0800-89-7339
Italy	1678-97-034		

Chapter 2 *Warp3* Tools

About This Chapter

Overview

Warp3 comprises a set of tools for creating and synthesizing designs for programmable logic devices. This chapter describes each tool in the set. The tools are discussed in their typical order of use.

2.1. Starting *Warp3*

To start *Warp3* from an IBM PC or compatible, double-click on the *Warp3* icon from the Windows program manager.

To start *Warp3* from a Unix workstation, type "powerview &<CR>" from the command line of a shell window.

Figure 2-1 shows the *Warp3* icon in the Windows Program Manager. Double-clicking on this icon brings up the *Warp3* program group. Double-clicking on the Cockpit icon within this group brings up the Workview PLUS Cockpit.

Typing the powerview command on a Unix workstation brings up the Powerview Cockpit.

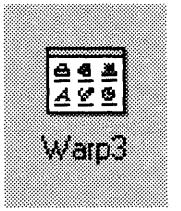


Figure 2-1. Warp3 icon (IBM PCs and compatibles)

2.2. The Cockpit

The Cockpit is the central access point for all *Warp3* tools.

Figure 2-2 shows the Workview PLUS cockpit, which appears when you invoke *Warp3* on IBM PCs and compatibles. Figure 2-3 shows the Powerview Cockpit, which appears when you invoke *Warp3* on Unix workstations.

The main feature of the Cockpit is a collection of icons in the area to the upper right. This area contains one icon for each *Warp3* tool. To start a tool, double-click on its icon.

Other features of the Cockpit include:

1. the Red Square menu in the upper left corner, which contains items that provide access to operations common to all ViewLogic tools;
2. four pull-down menus (Project, Library, Process, and Config), which contain items that provide access to project and library management commands;
3. the Tool Status area (left-half of the Cockpit, below the Red Square and pull-down menus), which provide a convenient way to set the current toolbox, drawer, project type, project, and library; and
4. the message area (bottom portion of the Cockpit), where messages from the various tools appear.

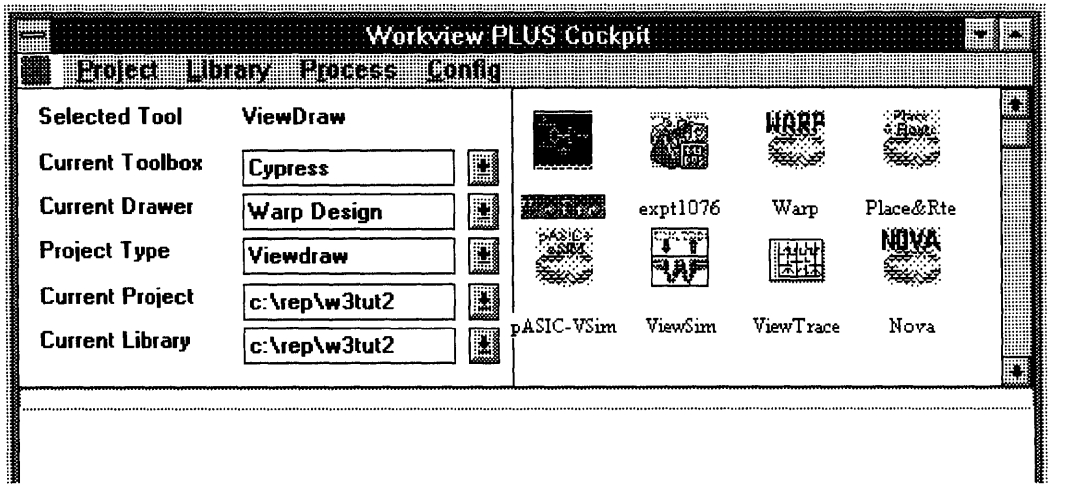


Figure 2-2. Workview PLUS cockpit (IBM PCs and compatibles)

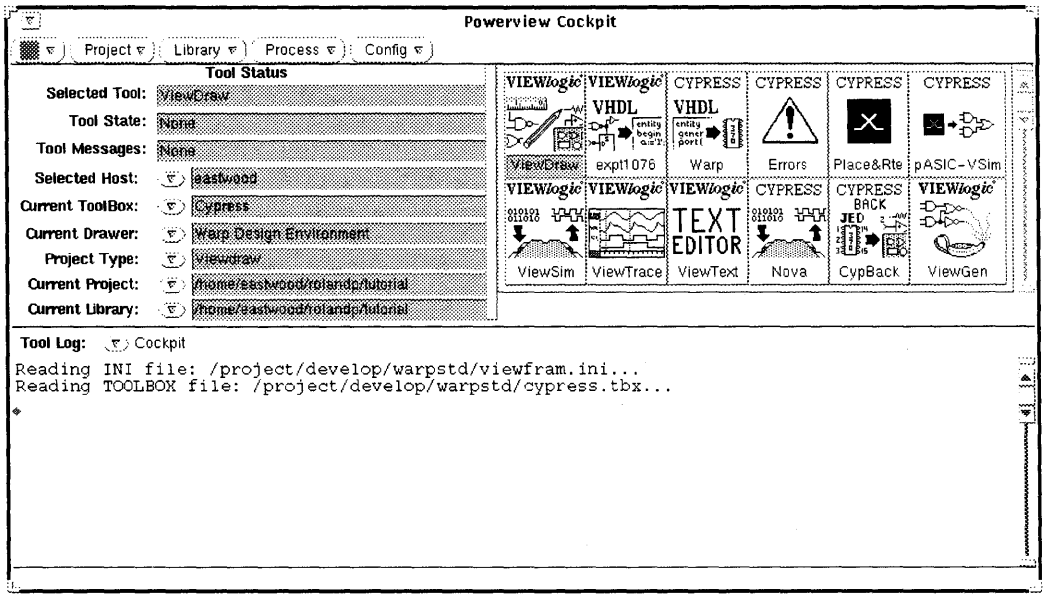


Figure 2-3. Powerview cockpit (Unix workstations)

The following pages discuss several topics related to the Cockpit that you should know about in order to use *Warp3*:

1. the Current Project;
2. the Search Order; and
3. the Warp Design and Circuit Design drawers. (These are named the Warp Design Environment and Digital Circuit Design drawers, respectively, when using *Warp3* on Unix workstations.)

For additional information about the Cockpit, refer to *Using Workview Plus on Windows* or *Using Powerview*, from the Viewlogic documentation set.

2.2. The Cockpit

2.2.1. The Current Project

The Current Project is the directory where *Warp3* expects to read input from and write output to.

Warp3 is easiest to use when each project is contained in its own directory. Therefore, whenever you start a new design, you should create a new project directory for it.

NOTE

The current project has no relationship to the directory that you started *Warp3* from. It must be specified separately.

To create a new project:

1. Choose the **Create** menu item under the **Project** menu.
2. Enter a project pathname.
3. Click on **OK**. The software reports "Project <*name*> created successfully."

To change the current project:

1. Click on the arrow next to **Current Project**.
2. Select a project from the list (Figure 2-4).

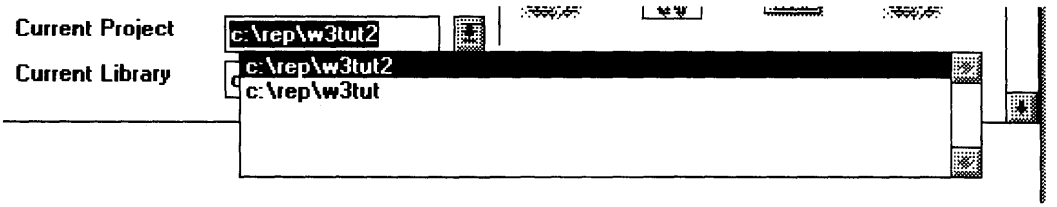


Figure 2-4. Setting the Current Project (IBM PCs and compatibles)

For more information about the Current Project, see *Using Workview Plus on Windows* or *Using Powerview*, from the Viewlogic documentation set.

2.2. The Cockpit

2.2.2. The Search Order

The Search Order is the sequence of directories that *Warp3* searches to find symbols for components instantiated in Viewdraw schematics.

From time to time, you may want to change the search order for a project. For example, you may develop your own library of components that you want to make available for use in a particular schematic.

On IBM PCs or Compatibles:

To change the search order for the current project:

1. Select Search Order from the Project menu in the Cockpit. The Search Order dialog box appears.

2. To edit a search order entry, click on a line in the Search Order dialog box, then click on the Edit... button. When a new dialog box appears, set the directory's type and modify the path or the library name as needed, then click on OK.

3. To insert a new search order entry, click on a line in the Search Order dialog box, then click on the Insert Before... button. When a new dialog box appears, set the new directory's type, enter the path, then enter the name of the new library, then click on OK. The new library will be inserted before the highlighted line.

4. To add an entry to the end of the search order list, click on the Append... button. When the dialog box appears, set the new directory's type, enter the path, then enter the name of the new library, then click on OK. The new library will be added at the end of the search order.

5. To delete an entry from the search order, click on a line in the Search Order dialog box, then click on the Delete... button.

Click on OK in the Search Order dialog box to accept the changes. Click on Cancel to return to the Cockpit without changing the Search Order.

On Unix Workstations:

To change the search order for the current project:

1. Select Search Order from the Project menu in the Cockpit. The Search Order dialog box appears.

2. To edit a search order entry, click on a line in the Search Order dialog box, then click on the field you want to edit.

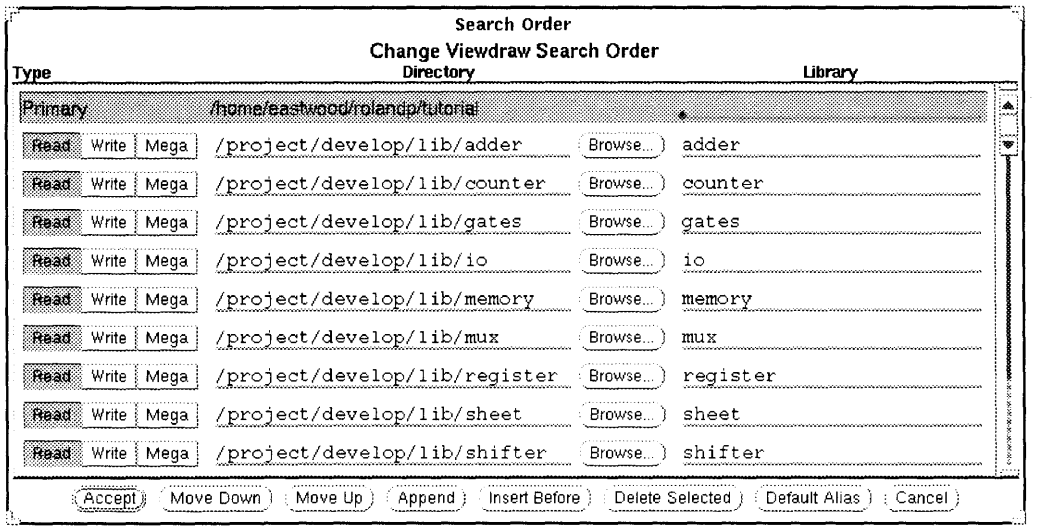


Figure 2-5. Search Order dialog box (Unix workstations).

3. To insert a new search order entry, click on a line in the Search Order dialog box, then click on the Insert Before button. A new blank line appears in the Search Order dialog box. Set the new directory's type, enter its path, and enter the name of the new library on this line.

4. To add an entry to the end of the search order list, click on the Append button. A new blank line appears at the end of the Search Order dialog box. Set the new directory's type, enter its path, and enter the name of the new library on this line.

5. To delete an entry from the search order, click on a line in the Search Order dialog box, then click on the Delete Selected button.

6. To move an entry higher in the search order, click on the entry, then click on the Move Up button.

7. To move an entry lower in the search order, click on the entry, then click on the Move down button.

Click on Accept in the Search Order dialog box to accept the changes. Click on Cancel to return to the Cockpit without changing the Search Order.

For more information about the Search Order, see *Using Workview Plus on Windows* or *Using Powerview*, from the Viewlogic documentation set. See also *Schematic Design User's Guide* from the Viewlogic documentation set.

2.2. The Cockpit

2.2.3. The Warp Design and Circuit Design Drawers

The Cockpit contains two "drawers," or collections of tools. The Warp Design drawer contains the tools most commonly used with *Warp3*. The Circuit Design drawer contains all the tools in the *Warp3* tool set.

Most often, you will be able to get by using only tools from the Warp Design drawer. The complete suite of tools in the Circuit Design drawer is provided for specialized uses.

2.3. ViewDraw

ViewDraw is a hierarchical design entry editor for easy input of schematics or symbols.

ViewDraw is a Viewlogic tool found in both the Warp Design and Circuit Design drawers. ViewDraw is used to create schematics, as well as symbols that can be instantiated in other schematics.

ViewDraw takes as input either the name of a schematic or the name of a symbol. On IBM PCs and compatibles, you specify whether the input is a schematic or symbol by means of the -sym or -sch option on the command line. On Unix workstations, you specify the same thing in the ViewDraw dialog box. The output from a "File=>Write" command is the data for the schematic or symbol window stored in a database, and for schematics a wirelist description file (**.wir**).

To run ViewDraw, double-click on the ViewDraw icon in the Cockpit. The ViewDraw File Open window will be displayed. Select whether you would like to edit a "Schematic", or "Symbol", and which library the schematic or symbol resides in. Now select the file to open from the list of files displayed, or enter a name on the "Enter name:" line. Double-clicking on the "Accept" button will bring up the ViewDraw Window. Make the desired modifications to the schematic, or symbol. Use the "File=>Write" command to write out the modifications to the database, check the schematic, and create a wirelist file. Selecting "Quit viewdraw..." from the Red Square menu will exit you from ViewDraw.

For additional information on ViewDraw, refer to the "ViewDraw Reference Manual" of the ViewLogic documentation set.

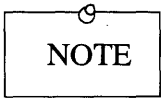
2.4. expt1076

Expt1076 translates the ViewDraw schematic design into one or more VHDL models.

Expt1076 is found in both the Warp Design and Circuit Design drawers. This program translates ViewDraw designs into one or more VHDL models. Expt1076 can produce VHDL output in a hierarchical, flattened, or block structure format. The only output format currently supported by *Warp* is the flattened output format. This is the default setting for expt1076.

Expt1076 takes as input the name of the design to translate and produces a file with the name matching the VHDL entity name and the extension ".vhd".

To run expt1076, select the expt1076 icon from the Cockpit. Enter the name of the ViewDraw design to be translated.



Double-clicking on the "expt1076" icon does not run the ViewLogic tool of that name, but instead runs a sequence of Cypress-created tools that perform the same function.

For additional information about expt1076 consult the *VHDL Reference Manual*, Chapter 7, of the Viewlogic documentation set.

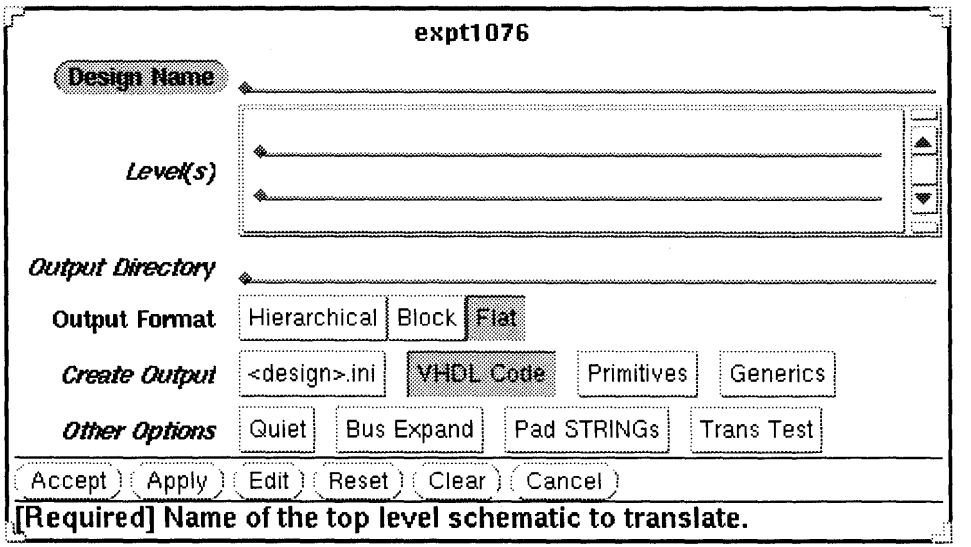


Figure 2-6. Dialog box for Expt1076 tool (Unix workstations).

2.5. Galaxy

Galaxy is the graphical user interface for *Warp*, *Warp3*'s VHDL compilation and synthesis tool.

Galaxy comes up when you double-click on the *Warp* icon from within the Workview PLUS Cockpit (on IBM PCs and compatibles) or the Powerview Cockpit (on Sun workstations).

For complete information on *Warp* and Galaxy, see the *Warp Synthesis Compiler Manual*.

2.6. Warp

Warp is *Warp3*'s VHDL compilation and synthesis tool.

Warp takes a VHDL description of a circuit as input. *Warp* produces JEDEC file (used for programming PLDs), HEX files (used for programming PROMs), or QDIF file (used by the SpDE tools when targeting pASICs).

For complete documentation of *Warp*, see the *Warp Synthesis Compiler Reference Manual*.

2.7. Place & Rte

The "Place&Rte" tool is used to process a QDIF file in order to target programmable ASICs (pASICs). This tool is used to perform automatic place and route, delay modeling, critical-path timing analysis, automatic test vector generation, and device programming and test.

The "Place&Rte" tool is a tool provided by QuickLogic found in both the Warp Design and Circuit Design drawers. This tool is used to perform automatic place and route, delay modeling, critical-path timing analysis, automatic test vector generation, and device programming and test.

"Place&Rte" takes as input a QDIF file generated by *Warp*, and produces **.atr** (design attribute file), **.dtb** (delay information), **.var** (delay information), and **.vl** (x-platform ASCII) files. If a save is done on the SpDE run a **.chp** file will also be produced.

To run SpDE, double-click on the "Place&Rte" icon. The SpDE window fills the screen. Load the QDIF file into SpDE by clicking on SpDE's **File** menu and selecting **Import QDIF**. Select the QDIF file you wish to run SpDE on and click on "OK". Click the **Tools** menu and select **Run All Tools**.

To create a .LOF file to program a part after all SpDE tools have been run, select File/Export/LOF, supply a file name in the dialog box, then click on "OK".

WARNING

Do NOT use the Program menu in the SpDE window to attempt to program parts! The correct procedure in *Warp3* is to create a .LOF file and transfer it to your device programmer, as described in this section.

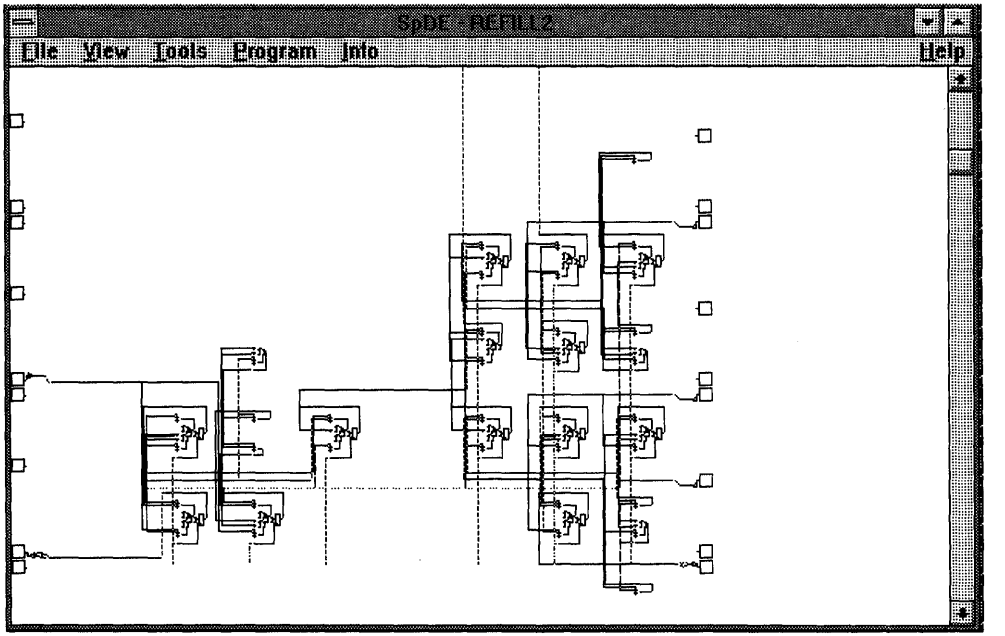


Figure 2-7. SpDE window.

SpDE creates the .LOF file, then asks if you would like to create a compressed version of it. (.LOF files can be rather large.) If you reply "Yes", a compressed version is created if you have a copy of the PKZIP program installed on your computer; otherwise you simply return to the main SpDE window.

(Data I/O's Unisite device programmer, as well as other popular third-party programmers, can read the ZIPped version of the .LOF file. Some device programmers cannot. Check the documentation that comes with your device programmer to be sure.)

For more information about the SpDE tools, see the *SpDE / Warp System User's Manual*, included in this documentation set.

2.8. pASIC-VSim

The "pASIC-VSim" tool reads the output information from a SpDE Place&Rte operation and generates a Viewsim simulation file.

The pASIC-VSim tool is a QuickLogic tool found in both the Warp Design and Circuit Design drawers. This tool is used for generating WIR files that can be used in Viewlogic.

The pASIC-VSim tool takes as input a file generated by SpDE and produces a new WIR file, that can be used by Viewsim.

To run pASIC-VSim, double-click on the "pASIC-Vsim" icon in the Cockpit and enter a file name. Selecting "OK" runs the tool.

On IBM PCs and Compatibles

When you invoke pASIC-VSim on an IBM PC or compatible computer, it brings up a dialog box that prompts you to enter a command line. The command line options you can use are as follows:

input file name	The name of the file generated by SpDE, without any extension.
output file name	Optionally enter a file name for the new WIR file, without any extension.

On Unix Workstations

When you invoke pASIC-VSim on a Unix workstation, it brings up a dialog box (Figure 2-8) that lets you set options for spde2vl's execution. Options you can set are as follows:

Input file name	Enter the name of the file generated by SpDE, without the file extension.
Output File name	Optionally enter a file name for the new WIR file, without the file extension.

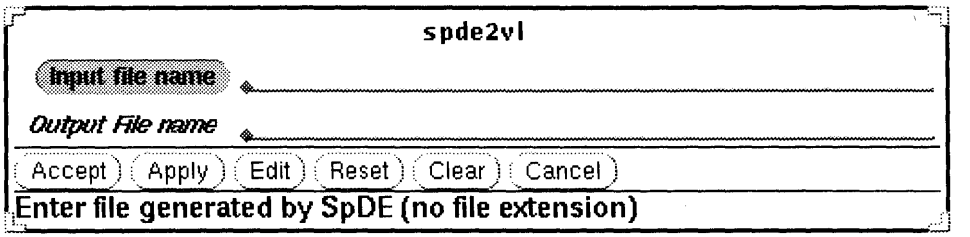


Figure 2-8. Dialog box for pASIC-VSim tool (Unix workstations).

2.9. vsm

Vsm creates a flattened netlist description of a structural design for logic simulation with Viewsim.

Vsm is a Viewlogic tool for taking a WIR file and translating it into a ViewSim netlist. This tool is found in both the Warp Design and Circuit Design drawers.

Vsm takes as input the name of the top level schematic to translate and produces a flattened netlist file (.**vsm**) for logic simulation with Viewsim.

To run **vsm,** double-click on the **vsm** icon in the cockpit. Enter the name of the schematic to translate and select **OK**. When vsm has completed running the window header will change to (Inactive vsm). To exit the window, select **Close** from the system window menu.

For additional information on vsm, refer to the "*ViewSim Wirelisting*" chapter of the "*ViewSim Reference Manual*" in the Viewlogic documentation set.

On IBM PCs and Compatibles

When you invoke **vsm** on an IBM PC or compatible computer, it brings up a dialog box that prompts you to enter a command line. The command line options you can use are as follows:

design name	Specifies the name of the top level schematic to translate.
<level>	Optional list of levels to descend to.
-f	Directs wirelist output into a file other than design_name.vsm.

-h	Generates full hierarchical net name equivalences in the wirelist, allowing you to observe simulation values of logically equivalent nets at any level in the design hierarchy. This is the default.
-s	Performs the function complementary to the **-h** option: all nets are referred using the netname of the top level net as it appears in the design.
-v	Specifies the name of a variable definition file which contains a list of parameterized attribute definitions.
-w	Generates a flattened connectivity (\WIR file format) description of the design.
-t	Specifies that the wire list is to include additional records specifically for fault simulation.
-d	Specifies the name of a delay back-annotation table file with the default extension **.dtb.**

On Unix Workstations

When you invoke **vsm** on a Unix workstation, it brings up a dialog box (Figure 2-9) that lets you set options for vsm's execution. Options you can set are as follows:

Design Name	Specify the name of the top level schematic to translate.

Level(s)	The Level string(s) list the LEVELs to descend to. The LEVEL string(s) are matched against LEVEL attributes on symbols in your schematic design. Any symbol with a matching string will be treated as a primitive.
Output Format	This options tells the netlister which output format to use. LONG generates full hierarchical net name equivalences in the wirelist. SHORT generates net names that refer to the top level net name. FAULT specifies that the wire list is to include additional records specifically for fault simulation.
Output Filename	Optionally specify the name of the output file. No extension should be specified since **.vsm** will be added.
Delay Table File	Optionally specify the name of the delay back-annotation table file, with the default extension **.dtb**.

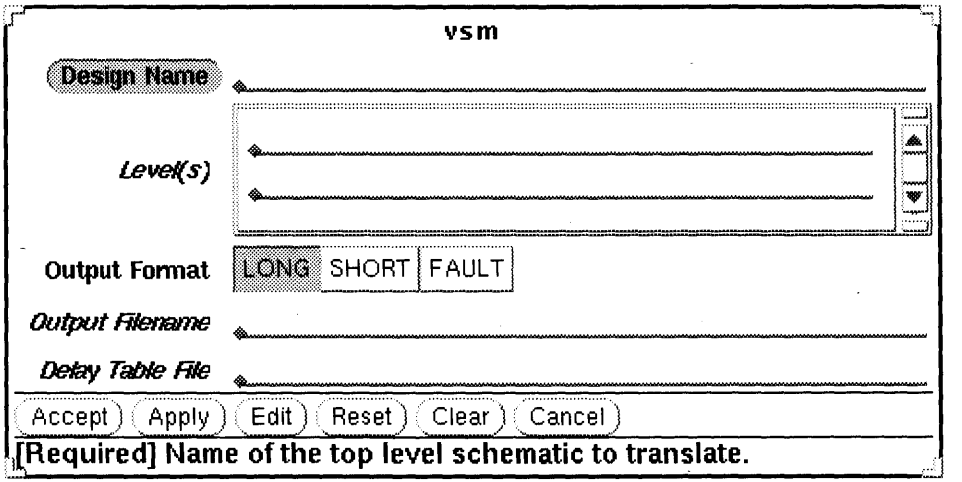


Figure 2-9. Dialog box for vsm tool (Unix workstations).

2.10. ViewSim

ViewSim is a high-capacity, full-timing simulator and (optional) debugger.

ViewSim is Viewlogic's powerful simulator designed to maximize the development and test environment. This tool can be found in both the Warp Design and Circuit Design drawers.

ViewSim will take as input either a **.vsm** file produced by the wirelister **vsm**, or **.vsm** and **.vli** files produced by the VHDL Analyzer when simulating an isolated behavioral model. ViewSim also accepts user-generated command files or command files created by ViewTrace. After the wirelist has been simulated, ViewSim outputs a waveform data stream (**.wfm**) for ViewTrace, simulation log file, and screen display information.

To run ViewSim, double-click on the ViewSim icon. Enter the design name and optionally the command file name on the command line. Select "OK". After the simulation has completed running you can use "Quit WVSim...", from the Red Square Menu to exit from ViewSim.

For additional information on ViewSim, refer to the "ViewSim Reference Manual", of the Viewlogic documentation set.

On IBM PCs and Compatibles

When you invoke ViewSim on an IBM PC or compatible computer, it brings up a dialog box that prompts you to enter a command line. The command line options you can use are as follows:

design name	Specify the name of the network file for your design.

-cmdfile	Name of a command file to run on startup.
-nographics	Disables the graphical interface to the ViewSim simulator.
-nofault	Disables capture of a fault simulation license.
-nodebug	Disables the VHDL debugging commands.

On Unix Workstations

When you invoke ViewSim on a Unix workstation, it brings up a dialog box (Figure 2-10) that lets you set options for ViewSim's execution. Options you can set are as follows:

Design Name	Specify the name of the network file corresponding to the level at which you netlisted. If no filename extension is supplied, **.vsm** is assumed
Command File	The command file can specify any of the simulation commands that would be issued during a simulation load/run.
Graphical Interface	This option enables or disables the graphical interface to the ViewSim simulator
VHDL Source Window	This window is used to display the VHDL source code while editing and debugging.

Fault Simulation This option instructs the simulator whether to capture a fault simulation license at startup. If disabled, another attempt will be made when the first fault simulation command is issued.

VHDL Debugging This option enables or disables the VHDL debugging commands.

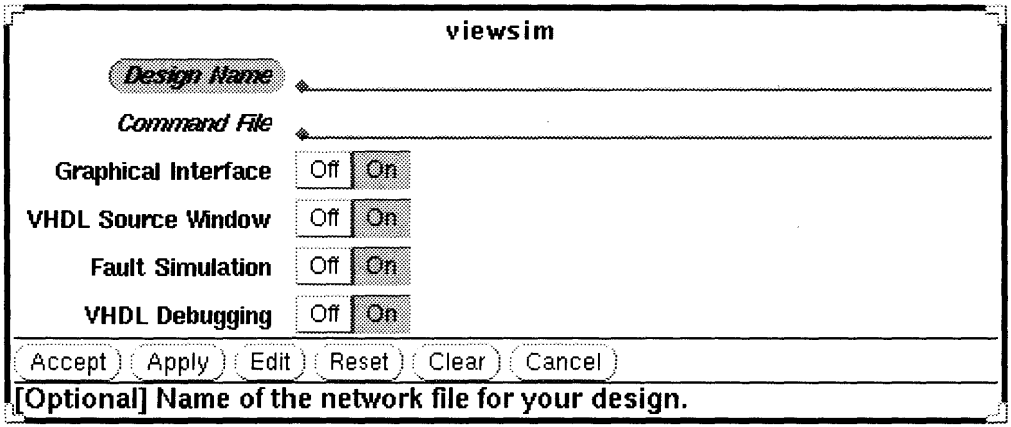


Figure 2-10. ViewSim Dialog Box on Unix workstations.

2.11. ViewTrace

ViewTrace is a graphical waveform analysis tool that directly ties to ViewDraw and ViewSim.

ViewTrace is Viewlogic's waveform analyzer tool. This tool is found in both the Warp Design and Circuit Design drawers.

ViewTrace enables you to read, display and process simulation results. ViewTrace takes as input a digital waveform stream file (**.wfm**), produced by ViewSim, and outputs a ViewSim command file (**.cmd**).

You will usually start ViewTrace by executing a wave command from Viewsim. To run ViewTrace by itself, double-click on the ViewTrace icon. Enter the name of the waveform file on the command line or select a file to open from the ViewTrace file selection dialog box. Perform the analysis on the displayed waveform using the available ViewTrace commands. Exit ViewTrace by selecting "Quit VIEWTRACE..." from the Red Square menu.

For additional information on ViewTrace , refer to the *"ViewTrace User's Guide"* in the Viewlogic documentation set.

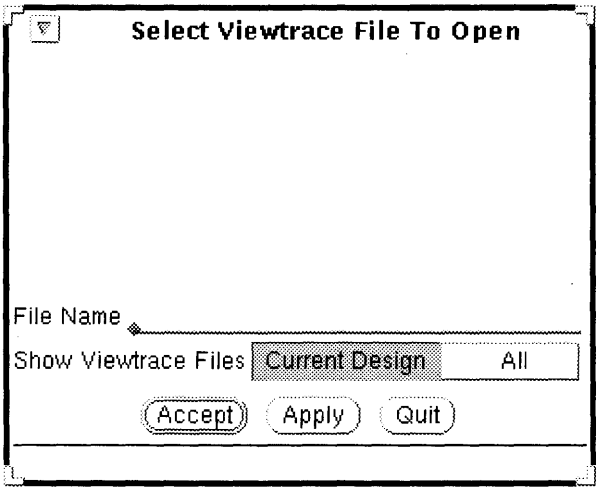


Figure 2-11. ViewTrace Dialog Box, Unix Workstations.

2.12. CypBack

CypBack back annotates pin assignment information added to the design during compilation and synthesis.

CypBack is a Cypress tool found in both the Warp Design and Circuit Design drawers. CypBack takes the pin assignment information found in either the **.vhd** file produced by *Warp*, or the **.atr** file produced from running Place and Route and writes the pin assignment information back to the schematic.

CypBack will modify the original schematic, adding the pin number attribute to the specified input and output ports. To view the new attributes select File/Read from the ViewDraw window and re-read the schematic.

To run CypBack, select the CypBack icon from the Cockpit and specify the design to back-annotate to.

On IBM PCs and Compatibles

When you invoke CypBack on an IBM PC or compatible computer, it brings up a dialog box that prompts you to enter a command line. You need enter no command line options to run CypBack. Make the command line read "CYPBCK <*schematic-name*>", then click on "OK".

On Unix Workstations

When you invoke CypBack on a Unix workstation, it brings up a dialog box (Figure 2-12) that lets you set options for CypBack's execution. Options you can set are as follows:

Design Name	enter the name of the design to be back-annotated

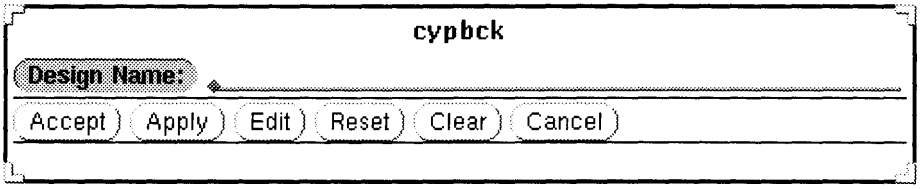


Figure 2-12. Dialog box for CypBack tool (Unix workstations).

2.13. ViewGen

In *Warp3*, ViewGen generates symbols from schematic drawing.

ViewGen is a Viewlogic tool found in both the Warp Design and Circuit Design drawers.

In *Warp3*, you would use ViewGen to generate a symbol for a circuit from a schematic. The resulting symbol could then be instantiated in other, higher-level schematics.

To run ViewGen in *Warp3* on IBM PCs and compatibles, double-click on the ViewGen icon, enter the name of the WIR file, followed by the arguments "-makesym" and "-noschem", then click on "OK".

To run ViewGen in *Warp3* on Unix workstations, double-click on the ViewGen icon, enter the name of the WIR file, make sure that "Make Schematic" is OFF and "Make Symbol" is ON, then click on "OK".

For additional information on ViewGen, refer to the "ViewGen Reference Manual" found in Viewlogic's online documentation set.

On IBM PCs and Compatibles

When you invoke ViewGen on an IBM PC or compatible computer, it brings up a dialog box that prompts you to enter a command line. Command line options required in *Warp3* are:

-makesym	Specifies that a top level symbol should be created for the design.
-noschem	Specifies that the generation of a schematic should be inhibited.

On Unix Workstations

When you invoke ViewGen on a Unix workstation, it brings up a dialog box (Figure 2-13) that lets you set options for ViewGen's execution. Options you can set are as follows:

Design Name	Enter either library_alias:name or the name of the WIR file.
Output File Name	Optionally enter an output file name.
Command File	Optionally enter a command file name.
Make Schematic	Set to "OFF" to suppress schematic generation.
Make Symbol	Set to "ON" to generate a symbol for the design.
Sheet Size	Does not apply in *Warp3*.
Layout	Does not apply in *Warp3*.

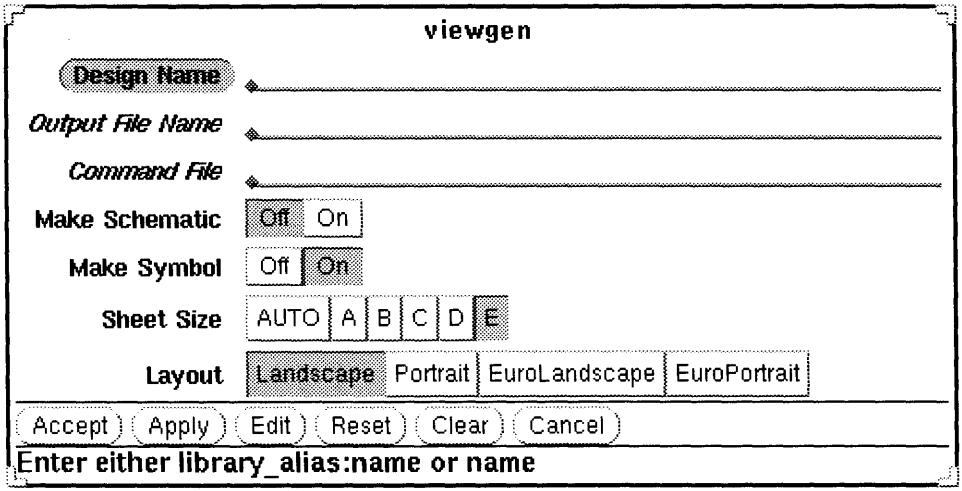


Figure 2-13. Dialog box for ViewGen tool (Unix workstations).

2.14. ViewText

ViewText is Viewlogic's ASCII text editor.

ViewText is a Viewlogic tool for editing an ASCII text file. This tool is available in both the Warp Design and Circuit Design drawers on Unix Workstations. On IBM PCs and compatibles, ViewText is available from most tool windows through the Red Square menu, except the Workview PLUS cockpit.

ViewText allows you to edit VHDL files, configuration files and initialization files. To run ViewText, double-click on the ViewText icon. Select the desired file from the ViewText file list window and click on "OK" ("ACCEPT" on Unix Workstations). This brings up the text editing window. To exit ViewText, select "Dismiss Window" from the Red Square menu.

One of the nice features of the *Warp3* text editor is the command "File=>VHDL to Symbol". This command will take the VHDL file being edited and translate it into a symbol.

For additional information on ViewText, refer to the chapter entitled, "Text Editing with ViewText", in "Using Workview PLUS," or "Using Powerview," in the Viewlogic documentation set.

On Unix Workstations

When you invoke ViewText on a Unix workstation, it brings up a dialog box (Figure 2-14) that lets you specify the file name for ViewText's execution. Options you can set are as follows:

Filename	The name of the file to edit.
Filter	Use the filter to specify the names to be displayed in the Files list.
Path	The full path of the current working directory.
Directories	List of directories available in the current working directory.
Files	List of files available in the specified directory.

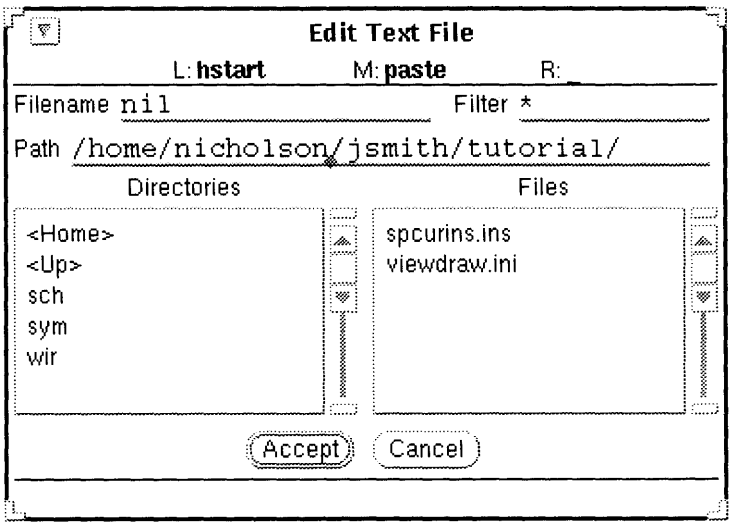


Figure 2-14. Dialog box for ViewText tool (Unix workstations).

On IBM PCs and Compatibles

The ViewText tool is not available from the Cockpit on the PC versions of *Warp3*. To access ViewText when working on an IBM PC or compatible, do the following:

1. Select and open one of the Viewlogic tools available in the cockpit.
2. From the Red Square menu of the selected tool, use the "Edit text file" command.

2.15. vhdl->sym

The vhdl->sym translator utility converts a VHDL model into a ViewDraw symbol.

Vhdl->sym is a Viewlogic tool that can be found in the Circuit Design drawer. The 'VHDL to Symbol' translator analyzes a VHDL model and automatically generates a symbol. You can confirm the correctness of the VHDL model by executing the VHDL Analyzer before executing vhdl->sym.

Vhdl->sym takes as input the name of either a VHDL source file or a library and translates each VHDL model into a ViewDraw symbol. The symbol name and the filename will have the same name as the VHDL entity name with an extension of "**.1**".

For additional information about vhdl->sym, refer to the *VHDL Reference Manual*, Chapter 8, in the Viewlogic documentation set.

On IBM PCs and Compatibles

When you invoke vhdl->sym on an IBM PC or compatible computer, it brings up a dialog box that prompts you to enter a command line. The command line options you can use are as follows:

***VHDLfile*[.ext]**	Specify the name of the VHDL source code file to be translated. If no filename extension is supplied, .**vhd** is used.
-l *input_library*	The **-l** option directs vhdl2sym to translate all VHDL models in the specified library.
-d *output_library*	Specifies a library to write the output file (symbol) to.

On Unix Workstations

When you invoke vhdl->sym on a Unix workstation, it brings up a dialog box (Figure 2-15) that lets you set options for vhdl2sym's execution. Options you can set are as follows:

VHDL Source Filename	Specify the name of the VHDL source code file for translation. If no filename extension is supplied, **.vhd** is used.
-or- Library Name	Specify the name of the library of VHDL source code files.
Output Library	Use this field to specify a library to write the output file (symbol) to. If not filled in, output goes to the library used or to the primary library.

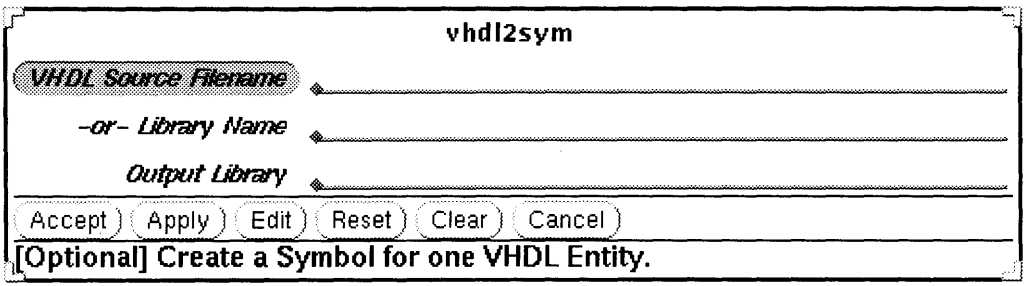


Figure 2-15. Dialog box for vhdl->sym tool (Unix workstations).

2.16. Errors

The *Warp* Error Tracker highlights the line of text in the VHDL file that contains the error.

The Errors tool is a Cypress-created tool for highlighting errors found by *Warp* back to the VHDL file. This tool is found in both the Warp Design and Circuit Design drawers, on Unix versions of *Warp3* only.

Errors takes as input a **.ver** file produced by *Warp*. This file is used to highlight the identified errors back to the specified **.vhd** file.

To run the *Warp* Error Tracker select the Errors icon from the Cockpit. Enter the name of the error file you wish to view *(file_name.ver)*. From the Alert box select the error line you would like to correct. Selecting the "Visit (Next)" button will invoke the ViewText editor with the error highlighted in the **.vhd** file. Selecting the "Continue" button will return you to the *Warp* Error Tracker dialog box.

On Unix Workstations

When you invoke the *Warp* Error Tracker on a Unix workstation, it brings up the dialog box shown in Figure 2-16.

Enter the name of the error file you wish to view on the line labeled "Error File", then click on Accept. Clicking on Browse allows you to search the directory structure for error files to view. Clicking on Cancel cancels the operation.

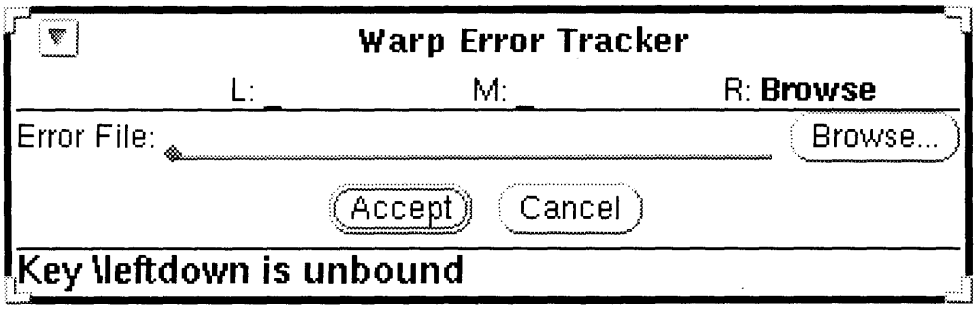


Figure 2-16. Dialog box for the Warp Error Tracker (Unix workstations).

On IBM PCs and Compatibles

The Errors tool is not provided on PC versions of *Warp3*. To locate sources of errors when working on an IBM PC or compatible, do the following:

1. Bring up ViewDraw and select the Red Square Menu.
2. From the Red Square Menu select "Help=>Track Errors from file...". This will bring up a dialog window asking for the name of the Error File. *Warp* creates a **.err** file if there were errors found in the VHDL file.
3. Enter the name of the error file (*file_name.err*). The "Browse..." button may also be used to select an error file from a list of error files. Selecting OK will bring up the Alert box with the list of errors.
4. Select the error to view and choose the "Visit (Next)" button. This will bring up the ViewText editor with the error line highlighted.

2.17. analyzer

The analyzer reads behavioral models in the form of VHDL source files, and produces an intermediate format file which ViewSim interprets to simulate behavioral models.

The **analyzer** is a Viewlogic tool that can be found in the Circuit Design drawer. This tool reads a VHDL source file, checks for errors, and creates the following files:

1. A listing file (*source_file*.**lis**) which indicates the location of errors in the source file. This file is always created.

2. An intermediate format file (*source_file*.**vli**). The intermediate format file is a functional description required by ViewSim. This file is only created if no errors occurred during analysis.

3. Optionally, the analyzer can create a netlist file (*source_file*.**vsm**) of a single instantiation of the model, allowing simulation of the isolated model.

4. An error message file (*source_file*.**elf**) which is used by the source level debugger to step through the error and warning messages that were issued by the analyzer.

The analyzer can be invoked from the Cockpit by selecting the analyzer icon. It can also be accessed from a ViewSim window, using the Analyze command button, or it can be invoked as a stand-alone program directly from the operating system.

The analyzer is not needed when targeting pASICs, because pASIC->VSim creates the .vsm file. Additionally, the analyzer is run automatically when you check the "Create Viewsim Model" box in *Warp*.

For additional information on the Analyzer, refer to the *VHDL User's Guide*, Chapter 3, in the Viewlogic documentation set.

On IBM PCs and Compatibles

When you invoke the analyzer on an IBM PC or compatible computer, it brings up a dialog box that prompts you to enter a command line. The command line options you can use are as follows:

-**e**[rrlevel]=	3-Error reporting 2-Error, warning reporting 1- Error, warning, extension reporting 0-Error, warning, extension, note reporting
-**d**[ebug]	This options adds source level debugging code (default: debugging disabled)
-**n**[ocheck]	This option specifies: No array bounds violation checking (default: checking enabled)
-**p**[ath]=<dir>	Search <dir> for **.msg, .ini**
-**v**[sm]	Generate ViewSim wirelist (default: no wirelist generated)
-**w**[ork]=<library>	Equate library WORK with <library> (default: WORK = <default dir>)

On Unix Workstations

When you invoke the analyzer on a Unix workstation, it brings up a dialog box (Figure 2-17) that lets you set options for the analyzer's execution. Options you can set are as follows:

VHDL Source Filename	Specify the name of the source file.
Source Level Debugging	Turning this value ON instructs the analyzer to record line numbers from the source file in the intermediate (**.vli**) file.
Array Bounds Checking	Turning this value ON instructs the analyzer to add extra code which ensures that array references are always within the defined bounds. This option should not be turned OFF until the model has been fully tested and debugged.
Create vsm File	This option tells the analyzer to create a ViewSim wirelist file which has the base name of your model and the extension VSM. This netlist can be used to simulate your model in isolation without entering ViewDraw to create a symbol and enclosing schematic for your model. The network contains one instance of the analyzed behavioral model; each port of the model is connected to a net of the same name.

Error Reporting

This option selects the kinds of errors to report. The choices are:

- Errors
- Errors and Warnings
- Errors, Warnings and Extensions
- Errors, Warnings, Extensions and Notes

Library Name

This option is used to specify the name of the library where this model resides. If the analyzer can not find the source file in the local directory, it searches the specified library. (The library name must match a Viewdraw library.) If a library name is NOT specified, AND the analyzer does not find the source code file in the local directory, each of the Viewdraw project libraries are searched sequentially.

Directory for .ini Files

This option is used to specify the path name of a directory containing the files workview.ini and viewdraw.ini. If a path is not specified, the analyzer searches (in order) the current directory and the directories specified by the WDIR environment variable. If a path is specified, the analyzer searches the current directory and then the specified path.

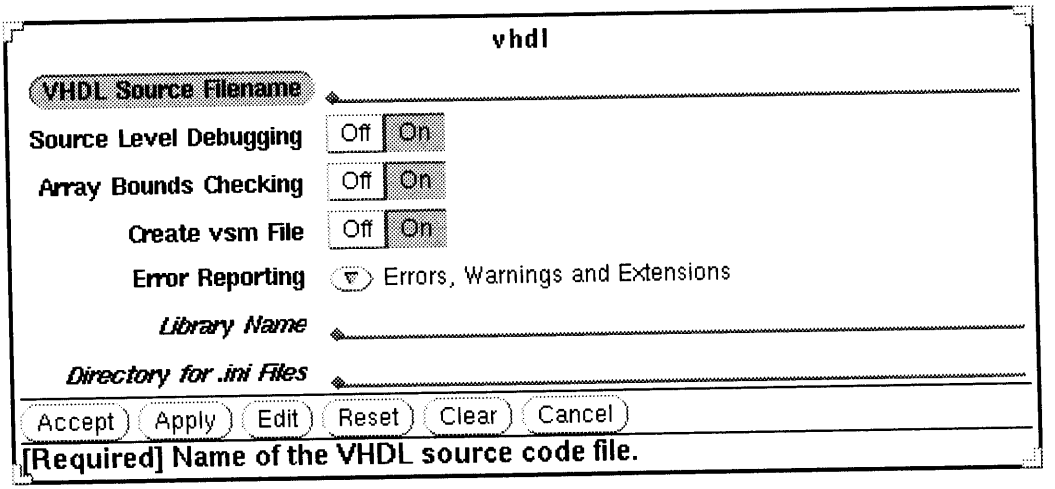


Figure 2-17. Dialog box for analyzer tool (Unix workstations).

2.18. ViewNav

ViewNav allows the user to find and/or visit various objects in a schematic design.

ViewNav is a Viewlogic tool found in the Circuit Design drawer. ViewNav allows the user to find objects within a ViewDraw design or between ViewDraw and another tool. An "object" can be a net, component, pin, or refdes (reference designator).

ViewNav takes as input the name of the design you wish to navigate through. It will query you as to how you want to search the design and use this information to highlight items in the schematic design.

To run ViewNav, double-click on the ViewNav icon. Enter the design name you wish to navigate and select "Accept". (Note: ViewDraw must be running, and the schematic you wish to navigate through should be loaded.)

The ViewNav top-level window will now be displayed for you to select between searching for objects, tracking errors from an error file, or editing parameters in the preferences window.

To exit the navigator, select "Quit vnav..." from the Red Square menu, and "Accept" from the Quit dialog box.

For additional information on ViewNav, refer to the online documentation section entitled, "Navigator and Cross Prober".

On Unix Workstations

When you invoke ViewNav on a Unix workstation, it brings up a dialog box (Figure 2-18) that lets you set options for ViewNav's execution. Options you can set are as follows:

Design Name	The name of the design you want to navigate through.
-co *configfile*	Optionally, enter the name of a configuration file on the Design Name line.

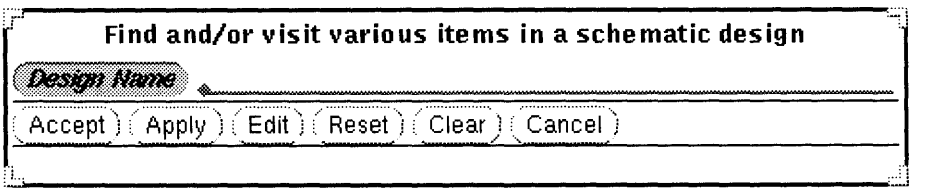


Figure 2-18. Dialog box for ViewNav tool (Unix workstations).

On IBM PCs and Compatibles

The ViewNav tool is not provided on PC versions of *Warp3*.

2.19. check

The check utility checks the current sheet or project for minor connectivity violations and creates a wirelist description file.

The check utility is a Viewlogic tool that is found in the Circuit Drawer. It takes either one sheet, the entire design, one library, or all schematics and performs a connectivity check on them.

Running the check utility creates a **.wir** file and places it in the wir subdirectory. This file contains information such as component type, component label, attributes, and connectivity, as well as any error messages that were reported during the check.

The check utility is automatically run when you write out a schematic in Viewdraw. You may also use the Utils => Check command from the Viewdraw menu, or select the Viewlogic check icon from the Cockpit.

For additional information on the check Utility, see the *ViewDraw Reference Manual*, Appendix B, in the Viewlogic documentation set.

On IBM PCs and Compatibles

When you invoke the check utility on an IBM PC or compatible computer, it brings up a dialog box that prompts you to enter a command line. The command line options you can use are as follows:

-all	The **-all** option checks all writable schematics in all writable directories in the viewdraw.ini file.

-f	The **-f** option (include with only the -l or -p options) retains symbols in memory between checking schematics.
-l	The **-l** option checks all projects in the specified library (-l *library_alias*).
-p	The **-p** option checks the specified project and underlying hierarchy.(-p *project_name*)
-s	The **-s** option checks only the specified schematic sheet. If no sheet number is specified, sheet one is checked. (-s *project.n*)
-v	The **-v** option (include with any option) reports additional check status.

On Unix Workstations

When you invoke the check utility on a Unix workstation, it brings up a dialog box (Figure 2-19) that lets you set options for check's execution. Options you can set are as follows:

One Sheet	The **One Sheet (-s)** options checks only the specified schematic sheet. If no sheet number is specified, sheet one is checked.
Entire Design	The **Entire Design (-p)** option checks the specified project and underlying hierarchy.
One Library	The **One Library (-l)** option checks all projects in the specified library.

All Schematics	The **All Schematics (-all)** option checks all writable schematics in all writable directories in the viewdraw.ini file.
Check What?	Specify the *project.n* for the **One Sheet** option, specify the *project_name* for the **Entire Design** option, or specify the *library_alias* for the **One Library** option.
-f	The **-f** option (include with only **-l** or **-p** options) retains symbols in memory between checking schematics.
-v	The **-v** option (include with any option) reports additional check status.

If you click on a check option and then click on **Edit**, you can modify the command line to include the **-f** and **-v** options.

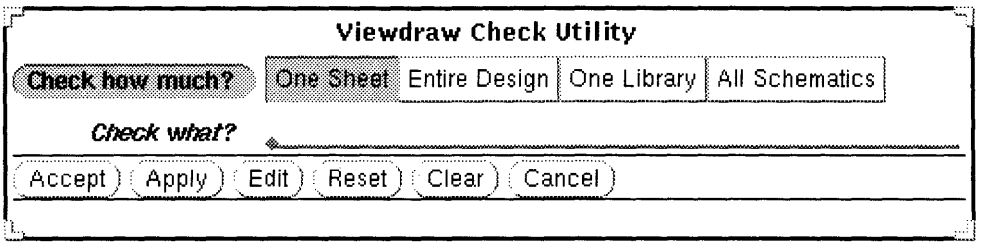


Figure 2-19. Dialog box for check tool (Unix workstations).

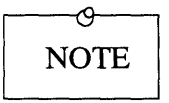
2.20. netlist in

The EDIF Netlist Reader is based on the industry standard EDIF (Electronic Design Interchange Format) to transfer netlist information from other vendor's EDIF translators.

The EDIF Netlist Reader (edifneti) is a Viewlogic tool that is available in the Circuit Design Drawer.

The EDIF Netlist Reader (edifneti) takes as input an EDIF 2 0 0 netlist and produces Powerview WIR files. Its purpose is to provide for connectivity linkage between "foreign" schematic capture systems and Powerview-supported physical design and verification tools.

To run the EDIF Netlist Reader select the "netlist in" icon in the Cockpit and enter the EDIF netlist file name. Clicking on "Accept" invokes the tool.



A word of caution about EDIF file transfers: importing or exporting EDIF files may not pass the entire design description. Libraries used to create the design may be needed in the environment to which the design is transferred.

For additional information, refer to Viewlogic's online documentation under the section entitled "EDIF 2 0 0 Netlister".

On IBM PCs and Compatibles

When you invoke the EDIF Netlist Reader on an IBM PC or compatible computer, it brings up a dialog box that prompts you to enter a command line. The command line options you can use are as follows:

<infile>	Refers to the input EDIF netlist filename. You must specify the filename extension.
-a *<filename>*	Specifies that the filename is the attribute name/value swap pairs configuration file. The default filename is **edifatts.cfg**.
-l *<labmapfn>*	Specifies that the labmapfn is the label swap configuration file. The default Label Swap Config File is **ediflabs.cfg.**
-o *<outpath>*	Specifies the output directory where the WIR files are to be written. The default is the WIR subdirectory of DIR 0 as specified in the **viewdraw.ini** file. Output filename(s) will be generated automatically.

On Unix Workstations

When you invoke the EDIF Netlist Reader on a Unix workstation, it brings up a dialog box (Figure 2-20) that lets you set options for **edifneti**'s execution. Options you can set are as follows:

EDIF netlist file	Enter the input EDIF netlist filename without the **.edn** extension
Attribute swap configuration file	Enter the name of the attribute name/value swap configuration file if the default, **edifatts.cfg**, is not being used.

Label swap configuration file	Enter the name of the label swap configuration file if the default, **ediflabs.cfg**, is not being used.
Pathname for output files	Enter the path where the WIR files are to be written. If the pathname is not specified, the WIR files will be written to the primary project directory.
-u	If specified on a Unix platform, the output file names will be written in lowercase.

If you enter the EDIF netlist filename and then click on **Edit**, you can modify the command line to include the **-u** option.

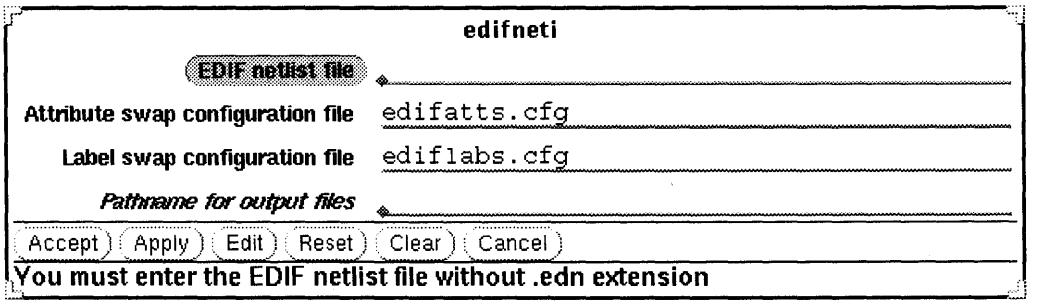


Figure 2-20. Dialog box for netlist in tool (Unix workstations).

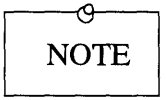
2.21. netlist out

The EDIF Netlist Writer (edifneto) is based on the industry standard EDIF (Electronic Design Interchange Format) to transfer netlist information to other vendor's EDIF translators.

The EDIF Netlist Writer is a Viewlogic tool found in the Circuit Design Drawer.

The EDIF Netlist Writer (**edifneto**) takes as input an up-to-date WIR file and produces a file (the "netlist") that describes the connectivity of a schematic in EDIF 2 0 0 format. This file may then be used as input to a foreign system's EDIF netlist reader.

To run the EDIF Netlist Writer select the **netlist out** icon and either provide the command-line options followed by a selection of the "**OK**" button, or on a Unix Workstation, respond to the dialog box options and click on the "**Accept**" button.



A word of caution about EDIF file transfers: importing or exporting EDIF files may not pass the entire design description. Libraries used to create the design may be needed in the environment to which the design is transferred.

For additional information on the EDIF Netlist Writer, refer to Viewlogic's online documentation under the section entitled "EDIF 2 0 0 Netlister".

On IBM PCs and Compatibles

When you invoke the EDIF Netlist Writer on an IBM PC or compatible computer, it brings up a dialog box that prompts you to enter a command line. The command line options you can use are as follows:

-f	Flatten the netlist without evaluating attributes.
-e	Flatten the netlist and evaluate attributes
<project>	The top-level project name.
-l *<level>*	The **-l** option specifies that the ***level*** string indicates what level the netlist is to stop. Multiple **-l** command-line options may be used. In that case, the flattener will flatten an instance down to the highest level listed among the provided strings.
<outfile>	This specifies the output EDIF filename. The default name is ***project*.edn**
-a *<author>*	The **-a** option specifies the ***author*** string. When specifying the ***author*** string on the command line it must be one string with no spaces.
-c *<filename>*	The **-c** option specifies the ***filename*** of the port/supply type of the configuration file. The default is **edifptyp.cfg**
-n *<nameswfp>*	The **-n** option specifies the filename of the name swap configuration file. There is no default.

-p	The **-p** option specifies that the use of library aliasing, in the naming of cells, within the EDIF file, be inhibited.

On Unix Workstations

When you invoke **netlist out** on a Unix workstation, it brings up a dialog box (Figure 2-21) that lets you set options for **edifneto**'s execution. Options you can set are as follows:

Select type	Select between the three ways of extracting the EDIF Netlist. Selecting **Hierarchical netlist** will produce a hierarchical netlist. Selecting **Flattened netlist - no evaluation** will produce a flattened netlist without parameterized attribute evaluation. Selecting **Flattened netlist - evaluated parameters**, will produce a flattened netlist with the evaluation of parameterized attributes.
Wire file name	Enter the name of the netlist (WIR file) to be translated into the EDIF format.
Level	If you do not want the EDIF netlister to flatten down to the lowest level, enter a string representing the level to netlist down to.
Output filename	Specify the name of the output file for the EDIF Writer. The default is the Wire file name followed by the **.edn** extension.

Author name Optionally enter the author name as one string to be stored as author string in the EDIF output file.

Port/Supply type config file Optionally specify a configurable file other than the default, **edifptyp.cfg**.

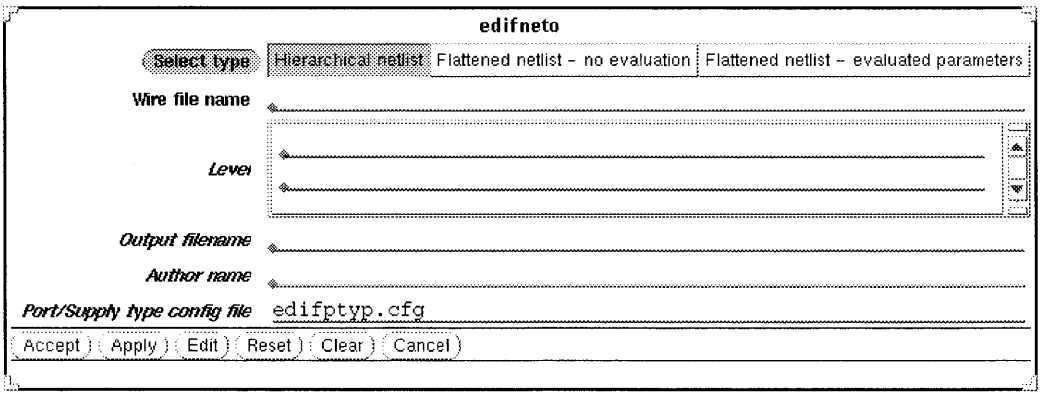


Figure 2-21. Dialog box for netlist out tool (Unix workstations).

2.22. sym->vhdl

The sym->vhdl translator utility converts a Viewdraw symbol into a VHDL model.

Sym->vhdl is a Viewlogic tool that can be found in the Circuit Design drawer. The 'Symbol to VHDL' translator analyzes a ViewDraw symbol and automatically generates a VHDL model skeleton. The tool can be directed to convert a single symbol or an entire library of symbols into VHDL models.

Sym->vhdl takes as input the name of a symbol, or a library, and translates each ViewDraw symbol into a VHDL file. The VHDL file has the same name as the symbol, except when modification is needed to create a legal VHDL name. The VHDL entity name is the same as the filename (minus the **.vhd** file extension).

For additional information about sym->vhdl, refer to the *VHDL Reference Manual*, Chapter 9, in the Viewlogic documentation set.

On IBM PCs and Compatibles

When you invoke sym->vhdl on an IBM PC or compatible computer, it brings up a dialog box that prompts you to enter a command line. The command line options you can use are as follows:

-i	Write the default initialization file into the working directory. When processing a single symbol, this option creates a *symbol_name*.**ini** file. When processing a library, this option creates a *library_name*.**ini** file.

-l *library*	Writes the VHDL model for all symbols that exist in the specified library.
-d *dir_name*	Writes the VHDL model file into the specified directory. If the directory does not exist, sym2vhdl will create it.
***symbol*[.ext]**	Writes the VHDL model for the symbol, *symbol*.ext. If the extension is not specified, **.1** is assumed.

On Unix Workstations

When you invoke sym->vhdl on a Unix workstation, it brings up a dialog box (Figure 2-22) that lets you set options for sym2vhdl's execution. Options you can set are as follows:

Symbol Name[.Ext]	Creates a VHDL entity for one symbol. If the symbol name is not followed by an extension, .1 is assumed
-or- Library Name	Creates VHDL entities for all symbols in the named library.
Output Directory	Use this field to specify a directory to write the output files to. If not filled in, output goes to the specified library or to the local directory.
Create .ini file	Turning this option "On" instructs the translator to create a template initialization file which can be edited to control numerous aspects of the translation process.

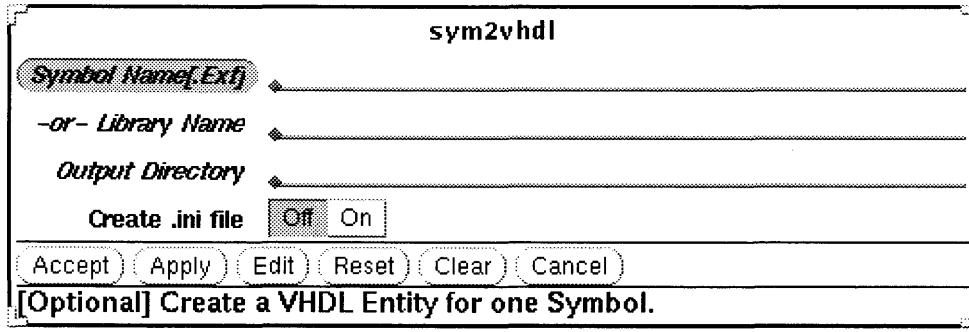


Figure 2-22. Dialog box for sym->vhdl tool (Unix workstations).

2.23. Nova

Nova is *Warp2*'s functional simulator.

Nova is the functional simulator that comes with *Warp2*, Cypress Semiconductor's original VHDL compilation and synthesis product. It is included with *Warp3* for the sake of compatibility and completeness only. For simulating designs created in *Warp3*, we recommend use of Viewsim.

Index

Warp3™

VHDL Development System

Tutorial

Cypress Semiconductor
3901 North First Street
San Jose, CA 95134
(408)943-2600
JANUARY 1995

Cypress Software License Agreement

1. LICENSE. Cypress Semiconductor Corporation ("Cypress") hereby grants you, as a Customer and Licensee, a single-user, non-exclusive license to use the enclosed Cypress software program ("Program") on a single CPU at any given point in time. Cypress authorizes you to make archival copies of the software for the sole purpose of backing up your software and protecting your investment from loss.

2. TERM AND TERMINATION. This agreement is effective from the date the diskettes are received until this agreement is terminated. The unauthorized reproduction or use of the Program and/or documentation will immediately terminate this Agreement without notice. Upon termination you are to destroy both the Program and the documentation.

3. COPYRIGHT AND PROPRIETARY RIGHTS. The Program and documentation are protected by both United States Copyright Law and International Treaty provisions. This means that you must treat the documentation and Program just like a book, with the exception of making archival copies for the sole purpose of protecting your investment from loss. The Program may be used by any number of people, and may be moved from one computer to another, so long as there is **No Possibility** of its being used by two people at the same time.

4. DISCLAIMER. THIS PROGRAM AND DOCUMENTATION ARE LICENSED "AS-IS," WITHOUT WARRANTY AS TO PERFORMANCE. CYPRESS EXPRESSLY DISCLAIMS ALL WARRANTIES, EXPRESSED OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE IMPLIED WARRANTY OF MERCHANTABILITY OR

FITNESS OF THIS PROGRAM FOR A PARTICULAR PURPOSE.

5. LIMITED WARRANTY. The diskette on which this Program is recorded is guaranteed for 90 days from date of purchase. If a defect occurs within 90 days, contact the representative at the place of purchase to arrange for a replacement.

6. LIMITATION OF REMEDIES AND LIABILITY. IN NO EVENT SHALL CYPRESS BE LIABLE FOR INCIDENTAL OR CONSEQUENTIAL DAMAGES RESULTING FROM PROGRAM USE, EVEN IF CYPRESS HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. CYPRESS'S EXCLUSIVE LIABILITY AND YOUR EXCLUSIVE REMEDY WILL BE IN THE REPLACEMENT OF ANY DEFECTIVE DISKETTE AS PROVIDED ABOVE. IN NO EVENT SHALL CYPRESS'S LIABILITY HEREUNDER EXCEED THE PURCHASE PRICE OF THE SOFTWARE.

7. ENTIRE AGREEMENT. This agreement constitutes the sole and complete Agreement between Cypress and the Customer for use of the Program and documentation. Changes to this Agreement may be made only by written mutual consent.

8. GOVERNING LAW. This Agreement shall be governed by the laws of the State of California. Should you have any question concerning this agreement, please contact:

   Cypress Semiconductor Corporation
   Attn: Legal Counsel
   3901 N. First Street
   San Jose, CA 95134-1599

   408-943-2600

Table of Contents

Chapter 1 - Introduction

Chapter 2 - Exercise 1 : Behavioral Description

Chapter 3 - Exercise 2: Structural Description

Chapter 4 - Exercise 3: Using Attributes in pASIC Design

Chapter 5 - SYNTHESIS_OFF & BUF

Chapter 1 Introduction

About This Chapter

Overview

The *Warp3*[1] Tutorial walks you through a common sequence of operations in using *Warp3*. The Tutorial shows you how to create, compile, synthesize, and simulate a design.

This chapter presents:

- introductory information about *Warp3*;
- a discussion of what this manual is and where it fits in the larger scheme of *Warp3* documentation;
- assumptions about tools you should be familiar with in order to run the Tutorial, including sources of information about those tools;
- some conventions about typography, wording, and illustrations used in this manual;
- installation and licensing requirements for running the Tutorial;

1. *Warp3* is a trademark of Cypress Semiconductor Corporation.

- the objectives of the Tutorial; and
- the contents of the Tutorial.

1.1. Introduction to *Warp3*

Warp3 is a Cypress Semiconductor software product that enables users to combine VHDL (text-based) descriptions with schematic drawings of electronic designs.

Finished files can be "synthesized" onto (i.e., used to program) Cypress parts, such as PLDs, CPLDs, and pASICs.

Warp3 consists of software offerings from three different companies brought together in one convenient package:

- Cypress provides the *Warp* VHDL compiler, the Nova functional simulator, and their user interfaces. *Warp* translates VHDL text descriptions into files that can be mapped onto programmable parts. Nova is a simple simulator that allows you to investigate design behavior.
- Viewlogic Corporation provides the Workview PLUS design environment (for IBM PCs and compatibles) and the Powerview design environment (for Unix workstations). Viewlogic provides the Cockpit, which is the central access point for all tools in the *Warp3* system. The Cockpit contains icons that bring up any tool the user needs. Viewlogic's design environments provide a large set of design tools in and of themselves. Among them are Viewdraw, a hierarchical schematic and symbol editor; Viewsim, a full-featured simulator that accurately models timing delays; Viewtrace, a waveform viewing tool that helps you analyze simulator results; and numerous other tools for special needs.

- QuickLogic Corporation provides the SpDE toolkit and its user interface. SpDE contains a set of tools for fitting designs onto pASICs. This tool set includes a placer, router, logic optimizer, path analyzer, and an automatic test vector generator, among others.

1.2. Assumptions

The *Warp3* tutorial assumes: (1) you have installed *Warp3* software on your system, and (2) you are familiar with the user interface conventions that control software operation on your specific platform.

Introduction

Warp3 tools used in this Tutorial include:

Tool	Use	Document
Cockpit	Project Management	*Using Workview PLUS*
ViewDraw	Design Entry	*Schematic Design User's Guide*; *ViewDraw Reference Manual*
ViewText	Text Entry	*Using Workview PLUS*
expt1076	Schematic-to-VHDL Conversion	*Warp3 VHDL Synthesis Reference*, Chapter 7
Warp	Design Compilation and Synthesis	*Warp3 User's Guide*, *Warp3 VHDL Synthesis Reference*
Place&Rte	pASIC Placement and Routing	*Warp3 User's Guide*
pASIC-VSim	Spde-to-ViewLogic Netlist Conversion	*Warp3 User's Guide*
ViewSim	Digital Simulation	*ViewSim Reference Manual*
ViewTrace	Design Analysis	*ViewTrace User's Guide*
CypBack	Back-Annotation	*Warp3 User's Guide*

1.3. Conventions

The following conventions are used throughout this manual:

Notational Conventions:

Menu items	Whenever an item from a menu is referenced, the reference takes the form menu-name/item-name/item-name... The first entry in the reference is the name of the menu; the second is the name of the menu item; the third and succeeding entries indicate choices from sub-menus. Example: "Select File/Level/Push/Schematic" tells you to pull down the **File** menu, select the **Level** item from it, select the **Push** item from the Level sub-menu, and then select the **Schematic** item from the Push sub-menu.
Italic	Words are italicized for emphasis or to draw attention to new terms.
`Courier`	Denotes the contents of a text file, or indicates that displayed text is system output. Also indicates the text of typed commands.

File-Naming Conventions

Warp3 runs on two different types of platform: IBM PC's and compatible computers, and Unix workstations.

IBM PC's and compatible computers specify file locations by designating a disk drive and using a backslash (\) character to distinguish directory levels, e.g., "c:\level1\level2\myfile".

Unix workstations specify file locations using a forward slash (/) and no disk drive designator, e.g., "/level1/level2/myfile".

For consistency and brevity, this manual uses the same notation as IBM PC's and compatibles when referring to file locations. Unix platform users are asked to make appropriate translations.

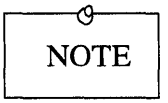
Mouse Conventions

The following terms describe common actions you might perform in using this manual:

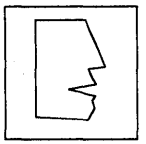
Click	Place the mouse cursor over an object, then press and release the appropriate mouse button.
Double-click	Position the mouse cursor over an object, then press and release the appropriate mouse button twice in rapid succession.
Drag	Position the mouse cursor over an object or at a specified location. Press and hold the appropriate mouse button. While holding down the mouse button, move the cursor to the new location. Finally, release the mouse button.
Press	When you are instructed to "press" a specified key, locate the key on the keyboard and press it.
Select	When you are instructed to "select" an option, move the cursor over the option, then click the appropriate mouse button.

Other Conventions

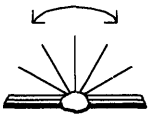
The following visual indicators denote special situations you may encounter while using this manual:



This icon indicates a *note:* a point in the tutorial: where you must exercise special caution, or; where the procedure might vary depending on platform you're using, or: where there's something else you should know about that doesn't fit into the main flow of the text.



This icon indicates a *hint:* a point in the tutorial where you could save a little time or a few key-strokes or much frustration, if you had known of the respective hint.



This icon indicates a *shortcut.* This tutorial has been designed so that you can choose which parts of the *Warp3* design, compilation, synthesis, and simulation process you want to work on. When you see this symbol, you're being given a choice to skip to another section of the tutorial.

1.4. Installation and Licensing

In order to run the *Warp3* Tutorial, you must install the *Warp3* software correctly (including necessary WIN32S files), and have a valid license to run the *Warp3* software.

Installation

Complete installation instructions are included in the *Warp3 Installation Notes*.

Licensing

In order to run the *Warp3* Tutorial, you must have a license to use the *Warp3* tools. Refer to the *Warp3 Installation Notes* for instructions on obtaining licenses.

1.5. Objectives

The objective of the *Warp3* Tutorial is to demonstrate the use of *Warp3* tools for creating, compiling, synthesizing, and simulating designs.

Tools to be demonstrated in the *Warp3* Tutorial include:

- Cockpit: the "toolbox" or central repository for all *Warp3* tools.
- ViewDraw: draws schematics and component symbols.
- expt1076: converts schematics into VHDL descriptions.
- Warp: compiles VHDL descriptions, produces JEDEC files to program PLDs and CPLDs, and produces QDIF files used in programming pASICs.
- Place&Rte: performs placement, routing, and other functions on QDIF files for programming pASICs.
- Viewsim/Viewtrace: simulates the behavior of designs in order to verify correct design performance.
- CypBack: adds ("back-annotates") the pin assignment information generated during compilation and synthesis to schematic files.

1.6. About the Tutorial

The *Warp3* Tutorial shows how to use tools to create, compile, synthesize, and simulate designs.

Chapters 2 and 3 of this Tutorial demonstrate, in step-by-step fashion, how to create designs using *Warp3* tools. The design we create in each chapter is called "refill."

Consider the following problem: we wish to design a controller for a soft-drink dispensing machine. The machine has two bins which dispense Pepsi and Coca-Cola, respectively. Each bin holds three cans of soft drink. (This could be any positive integer, but three is an easy number to simulate with.)

We want the circuit to dispense a beverage when the user presses a button for that beverage and one or more cans of the beverage are available. We want the circuit to NOT dispense a beverage when no cans of that beverage are available. We want to get a REFILL signal when both bins are empty. We also want to be able to press a RESET signal to tell the circuit when the machine has been replenished and the bins are full again.

We will create two solutions to this design problem, one using a combination of behavioral and structural modeling, the other using structural modeling alone.

The first solution, presented in Chapter 2, proceeds as follows:

First, we'll describe a circuit in VHDL that controls the operation of one bin. It will respond appropriately to a "get_drink" signal (i.e., by providing a drink when one is available); keep count of the number of cans left in the bin; and set an EMPTY signal when its bin becomes empty and is in need of resetting. In addition to writing a text description of the circuit's behavior, we'll add appropriate VHDL to make a symbol out of the circuit. This will allow us to use the circuit in higher-level schematics.

Next, we'll do just that (i.e. create a schematic that instantiates our VHDL description twice, once for each bin in the machine). We'll also add input and output port symbols, and an AND gate to tell us when both bins are empty and the machine needs to be replenished. We'll wire it all together, and the end result will be a drink machine controller.

After that, we'll obtain a VHDL description of the entire schematic; synthesize that VHDL description into a JEDEC file (for PLDs and CPLDs) and a QDIF file (for pASICs); process the QDIF file for transfer to pASICs; simulate the behavior of the entire circuit; and finally, back-annotate pin assignment information added during synthesis to the original schematic.

Figure 1-1 shows the sequence of operations used in Chapter 2's Tutorial exercise.

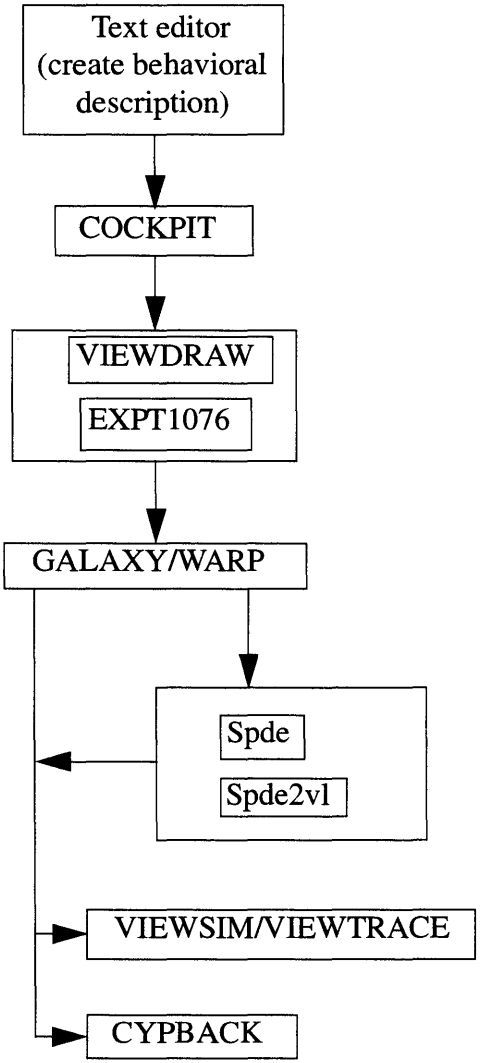


Figure 1-1. Flow Diagram of Warp3 Tutorial, Chapter 2.

The second solution, presented in Chapter 3, will create a functionally identical circuit. Instead of a behavioral description of the circuit that controls a single bin, we'll create a structural description (i.e., a schematic of the circuit). We'll convert this schematic into a symbol that can be instantiated on other schematics, then proceed as was done in Chapter 2.

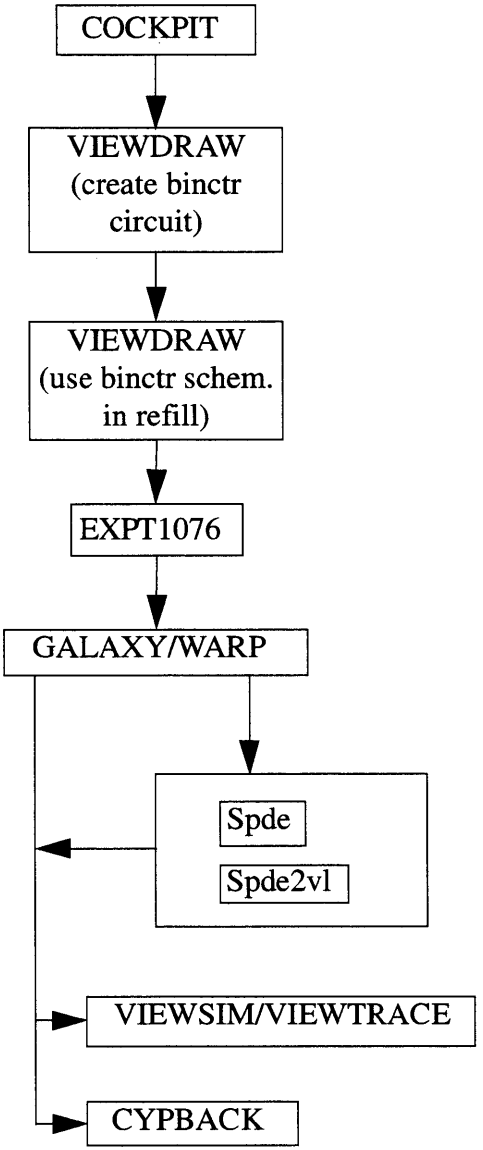


Figure 1-2. Flow Diagram of Warp3 Tutorial, Chapter 3.

1.6. About the Tutorial

1.6.1. File/Directory Management

As you work through the Tutorial, doing the Tutorial exercises (or any other time, for that matter), don't write anything into the *Warp* directory. Instead, create a separate directory to practice in (e.g., c:\w3tutor).

By default, the *Warp3* installation procedure for IBM PC's and compatible computers installs *Warp3* software into a directory named c:\warp. On Sun workstations, the default location for the *Warp3* software is the directory pointed to by the CYPRESS_DIR environment variable.

1.6. About the Tutorial

1.6.2. Differences in Operating Systems

Warp3 operates identically on both Unix and Windows systems. However, there are differences in the way you start *Warp3*, and in the appearance of various objects on the display.

Starting *Warp3*

On Windows systems, you start *Warp3* by double-clicking on the Cockpit icon from the *Warp3* directory.

On Unix systems, you start *Warp3* by typing "powerview<CR>" from within a shell window.

On both platforms, the Cockpit should be displayed immediately after you start *Warp3*. The Cockpit is labeled "Workview PLUS Cockpit" on Windows systems, "Powerview Cockpit" on Unix systems.

Differences in Display

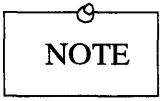
There are minor differences between the Windows and Unix versions of *Warp3* in the appearance of the Cockpit, the dialog boxes, and the prompt boxes.

In the Windows environment, *Warp3* features movable icons in the Workview PLUS DOS Cockpit. The icons are not movable in the Powerview Cockpit on Unix systems.

Although dialog boxes and prompts may differ in appearance between the two platforms, their functionality is identical. Screen captures in this manual are taken from the Windows version of *Warp3*. Differences in the Unix version are identified when necessary. In most cases, adjustments needed to go from Windows to Unix versions of *Warp3* displays will be obvious.

Naming Restrictions

When working in the Windows environment, you *must* keep file names restricted to eight or fewer alpha-numeric characters (the first of which must be an alphabetic character), plus a file extension of up to three characters (e.g., ".exe", ".vhd"). This is important to remember should you need to transfer data between Windows and Unix implementations of *Warp3*.



Keep in mind also that some Unix systems have trouble with spaces embedded in file names. That can be true for IBM PCs and compatible systems, too. *Don't use embedded spaces in file names.*

Chapter 2

Exercise 1: Behavioral Description

About This Chapter

Overview

This chapter takes you step-by-step through the tutorial example, using a low-level behavioral description and a high-level structural one. When you complete this chapter, you will know how to:

- write an entity declaration, architecture, and package declaration for a VHDL description of a simple circuit;
- create a symbol for the VHDL description to allow it to be instantiated as a component;
- create a schematic for a larger circuit, instantiate and position components, label input and output ports, wire components together, and save the schematic;
- "export" the schematic file to obtain a VHDL description of the entire schematic;
- run Warp to synthesize the schematic's VHDL description;
- simulate the behavior of the design; and
- back-annotate pin information added during synthesis.

2.1. Start *Warp3*

To start *Warp3*, double-click on the *Warp3* icon in the Windows program manager. (On Unix workstations, type "powerview<CR>" on the command line of a shell window.)

When *Warp3* starts, the Workview PLUS Cockpit (hereafter referred to as "the Cockpit") appears (Figure 2-1). This window is named the Powerview Cockpit on Unix systems.

The Cockpit is the access point for all *Warp3* tools. To start a tool, double-click on the tool's icon from within the Cockpit window.

The Cockpit is organized into "toolboxes," which are themselves organized into "drawers." At any time, the Cockpit will display the tools available in the current drawer of the current toolbox. By default, the Current Toolbox is named "Cypress," and the Current Drawer is named "Warp Design."

To change the current toolbox or drawer, click on the down-arrow next to the "Current Toolbox" or "Current Drawer" line, then select the toolbox or drawer from the available ones listed.

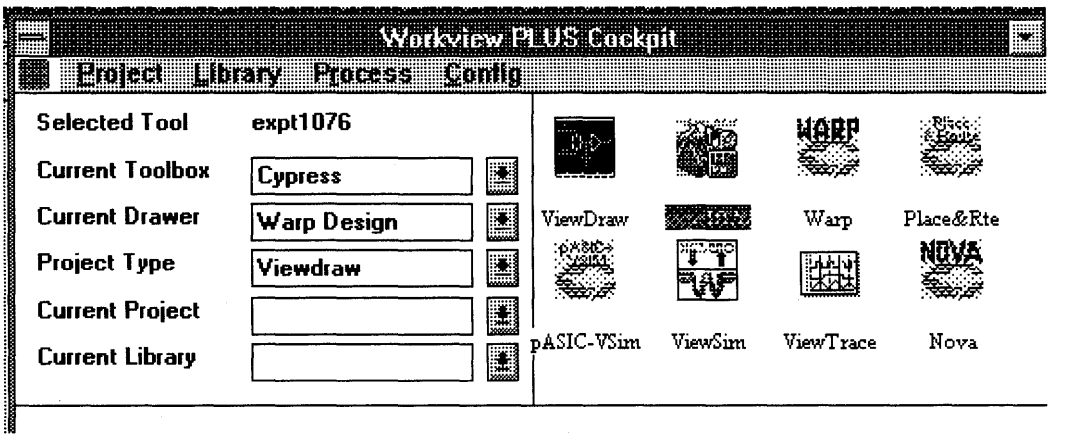


Figure 2-1. WorkView PLUS cockpit.

2.2. Create a Project

The project directory should be set just before running any *Warp3* tools. *Warp3* stores all generated files and creates sub-directories in the project directory.

To create a new directory and set the project directory to it, select Project/Create from the Cockpit menu bar. When a dialog box appears, type a complete pathname, then click "OK". The named directory is created if it doesn't already exist, and made the project directory.

To set the project directory to an existing project, select Project/ Set Current from the Cockpit menu bar. A dialog box appears, listing current projects. Select one, then click "OK".

For the sake of this Tutorial, set the project directory to c:\w3tutor. On the PC, select Project/Create, type "c:\w3tutor"), then click on "OK". In UNIX, go to the directory $CYPRESS_DIR/w3tutor.

2.3. Start Viewdraw

In this Tutorial, we'll use Viewdraw to create a schematic for the "refill" circuit. We'll also use Viewdraw's text editor to write the VHDL file for a lower-level component of the "refill" circuit that we'll call "binctr".

To start Viewdraw, double-click on the Viewdraw icon from the Workview Plus Cockpit, then click "OK" in the ensuing dialog box. Viewdraw loads. (It takes a few seconds.)

The Viewdraw File open dialog box appears (Figure 2-2). Type "refill" on the "Enter name:" line in the dialog box, then click "OK". A new, blank schematic sheet appears.

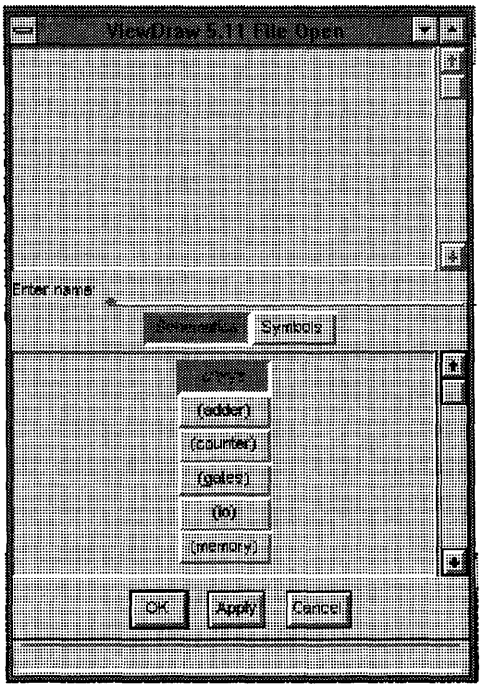


Figure 2-2. Viewdraw File Open dialog box.

2.4. Start Viewtext (Viewdraw's Text Editor)

ViewText is Workview PLUS's (and Powerview's) ASCII text editor.

We'll use ViewText to create the VHDL description for the "binctr" component. We'll convert this description into a symbol, which we'll then instantiate onto the "refill" circuit.

To start ViewText, select "Edit Text File..." from the Red Square menu in the ViewDraw window.

NOTE

If you are using Viewdraw on an IBM PC or compatible computer:

The Viewtext text editor is not available from the Red Square in the Cockpit. It's only available from the Red Square menu in Viewdraw or Viewsim or other tools *contained* in the Cockpit.

Make sure you pull down the Red Square menu from the Viewdraw window.

The Viewtext dialog box appears (Figure 2-3). Type "BINCTR.VHD" on the line labeled File Name, then click "OK".

The Viewtext window appears (Figure 2-4). Now you can start entering the text of the BINCTR.VHD file.

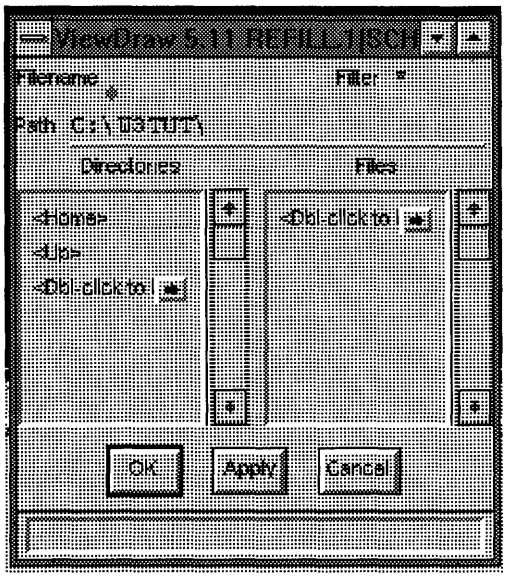


Figure 2-3. Viewtext Dialog Box.

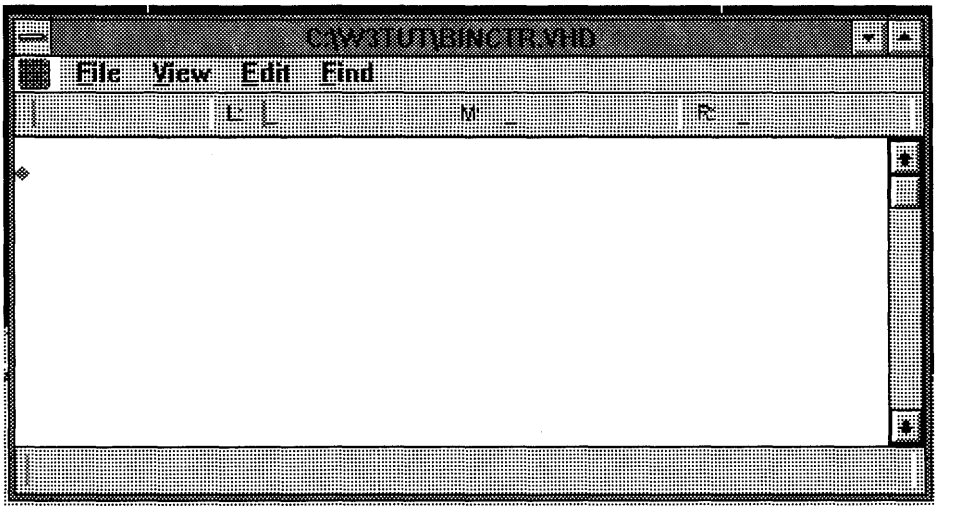


Figure 2-4. Viewtext Window.

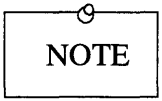
2.5. Write the "binctr" VHDL description

The "binctr" VHDL description controls the behavior of a single bin in the drink machine. It acknowledges drink requests, dispenses drinks, keeps count of the number of drinks remaining, and sets an output signal when the bin becomes empty.

The "binctr" VHDL description will be written in three parts:

- the ENTITY declaration declares the name, direction, and data type of each port of the component;
- the ARCHITECTURE describes the behavior of the component;
- the PACKAGE declaration provides the information to the Warp compiler to allow binctr to be used as a component in a schematic.

The following pages discuss the contents of each of these sections of the VHDL description.



If you would rather not type in the BINCTR.VHD file, you can copy it from the Warp directory.

The default location for the BINCTR.VHD file is c:\warp\examples\w3tutor\binctr.vhd on the PC. For UNIX, the default location is $CYPRESS_DIR/examples/w3tutor. From here, copy binctr.vhd to your project directory.

Copy this file to directory c:\w3tutor. Then, read along for the next few pages, to help you understand the purpose of each section of a VHDL source file.

2.5. Write the "binctr" VHDL description

2.5.1. Write the Entity Declaration

The ENTITY declaration declares the name, direction, and data type of each port of the component.

Figure 2-5 lists the entity declaration for the binctr component.

The ENTITY declaration declares that entity "binctr" has five external interfaces, or "ports." It has three input ports of type BIT, named reset, get_drink, and clk, respectively. It has two output ports, also of type BIT, named give_drink and empty, respectively.

```
entity binctr is port(
  reset, get_drink, clk : in bit;
  give_drink, empty: out bit);
end binctr;
```

Figure 2-5. ENTITY Declaration of binctr Component.

2.5. Write the "binctr" VHDL description

2.5.2. Write the Architecture

The ARCHITECTURE portion of a VHDL description describes the behavior of the component.

Figure 2-6 lists the architecture portion of the binctr component's VHDL description. The architecture appears after the entity declaration in the .VHD file.

The first line declares an architecture named "archbinctr" of entity "binctr".

NOTE

In Warp3, the first line of an architecture must always take the form:

ARCHITECTURE arch<entity> OF <entity> IS

The next two lines declare a constant and a signal, respectively.

- The constant, named "full", determines how many drinks are in a full bin.
- Signal "remaining", of type integer with a range from 0 to the value of constant full, keeps track of how many drinks are left in the bin.

The BEGIN that follows the signal declaration marks the start of the architecture body.

There follows a process declaration, marked by the keyword PROCESS and an ensuing BEGIN.

```
architecture archbinctr of binctr is
  constant full:integer:=3; -- max of 3 drinks/bin
  signal remaining:integer range 0 to full;
begin
  process begin
    wait until clk = '1';
    if reset = '1' then
      remaining <= full;
      empty<='0';
      give_drink<='0';
    elsif remaining=0 then
      empty <= '1';
      give_drink <= '0';
    elsif get_drink = '1' then
      remaining <= remaining - 1;
      give_drink <= '1';
    elsif get_drink = '0' then
      give_drink <= '0';
      end if;
    end process;
  end archbinctr;
```

Figure 2-6. ARCHITECTURE of binctr Component.

The line "WAIT UNTIL clk='1'" synchronizes all activity within the process with transitions of signal clk to '1'.

Subsequent signal activity is handled in the following order:

- If signal "reset" is '1', then signal "remaining" is set to full, signal "empty" is set to '0', and signal "give_drink" is set to '0'. (Notice that this means reset has priority over get_drink; if reset and get_drink are both true on the same clock cycle, the bin is reset, but no drink is given.)

- Otherwise, if signal "remaining" has value 0, then signal "empty" is set to '1' and signal "give_drink" is set to '0'.
- Otherwise, if signal "get_drink" is '1', then signal "remaining" is decremented by one and signal "give_drink" is set to '1'.
- Otherwise, if signal "get_drink" is '0', the signal "give_drink" is set to '0'.

Several lines ending the IF statement, process, and architecture follow. Note that the END statement ending the architecture must be accompanied by the name of the architecture, which *must* match the name shown on the first line of the architecture.

2.5. Write the "binctr" VHDL description

2.5.3. Write the Package Declaration

The PACKAGE declaration provides the information to the *Warp* compiler to allow binctr to be used as a component in a schematic.

Figure 2-7 lists the package declaration for the binctr component. The package declaration must appear before the entity declaration or architecture in the .VHD file.

The first line in Figure 2-7 declares the name of the package. The name of the package must be distinct from the name of any component declared within that package. Using the convention "<entity>_pkg" works nicely.

The second line declares a component named "binctr". The component name that appears on this line must match the name of an accompanying entity.

The port statement declares the name, direction, and type of each port in the component. You can copy the port statement from the entity declaration for this purpose.

An "end component" and "end binctr_pkg" statement conclude the package declaration. Note that the package named in the END package statement must match that shown in the first line of the package declaration.

At this point, save the file as c:\w3tutor\binctr.vhd and minimize the Viewtext and Viewdraw windows.

```
package binctr_pkg is
  component binctr
    port(reset, get_drink, clk : in bit;
         give_drink, empty: out bit);
  end component;
end binctr_pkg;
```

Figure 2-7. PACKAGE Declaration of binctr Component.

2.6. Run Warp

Warp is the VHDL synthesis compiler for *Warp3*.

Warp takes VHDL descriptions as input. Warp has two functions in the *Warp3* system: (1) to verify that VHDL descriptions are syntactically correct, and (2) to produce JEDEC or QDIF files to program output devices.

At this point in the Tutorial, we'll use *Warp* to verify that the BINCTR.VHD file is syntactically correct. This step isn't strictly necessary; we could simply go on and build the schematic for the "refill" circuit. But it's always a good idea to compile any VHDL description once you've completed it. That way, you can spot problems in your VHDL description when they are easiest to identify and correct. Later on, should you encounter problems with the larger circuit, you can at least be assured that you have taken care of any bugs at the lower levels of the hierarchy.

2.6. Run Warp

2.6.1. Start Galaxy

Galaxy is the user interface for the *Warp* VHDL synthesis compiler.

To start Galaxy, double-click on the Warp icon in the Cockpit. Click on "OK" from the ensuing dialog box. Close the ensuing "About" window. The Galaxy window appears (Figure 2-15).

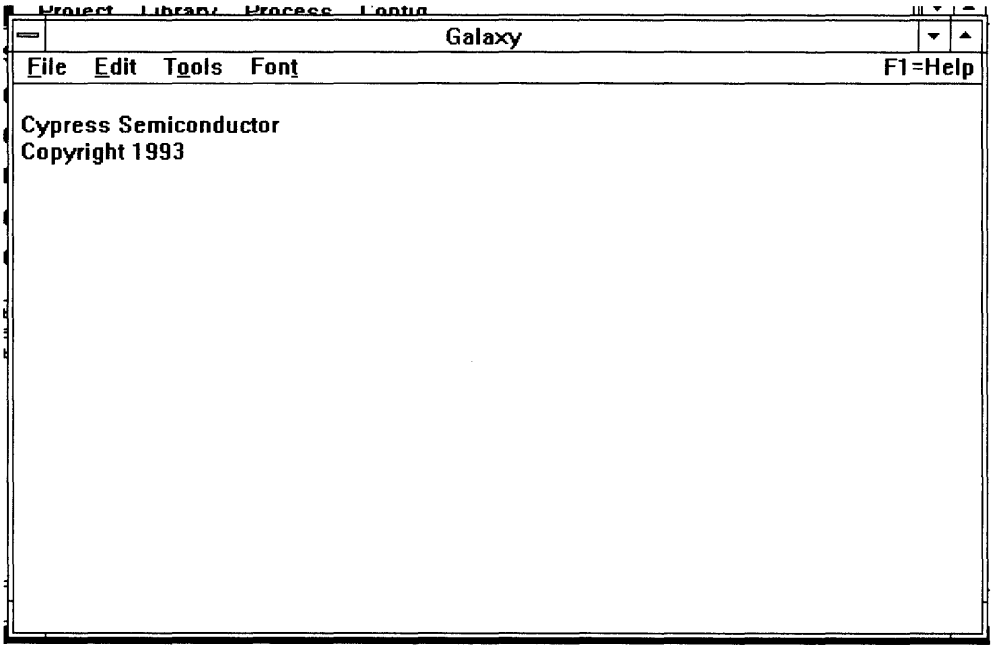


Figure 2-8. The Galaxy window (1).

2.6. Run Warp

2.6.2. Compile binctr.vhd

To bring up the *Warp* dialog box, select Tools/Run Warp Menu from the Galaxy menu bar.

The *Warp* dialog box appears (Figure 2-16).

This dialog box lists the VHDL files available in the current directory on the left side, and the VHDL files selected for compilation/synthesis on the right side.

To select the binctr.vhd file for compilation, select binctr.vhd from the list of files on the left-hand side, then click on the "Add" button.

Then, select "Compile Only" from the "Build" button group near the lower-left corner of the dialog box.

Then, click on "OK". *Warp* runs, printing messages to keep you appraised of its progress.

The compilation process should run to completion, without any error messages.

NOTE

If you do get error messages, check to make sure that the various parts of the binctr.vhd file read EXACTLY as they are listed on the preceding pages. Better yet, copy the binctr.vhd file from the \examples\w3tutor sub-directory of the Warp directory, then run *Warp* again.

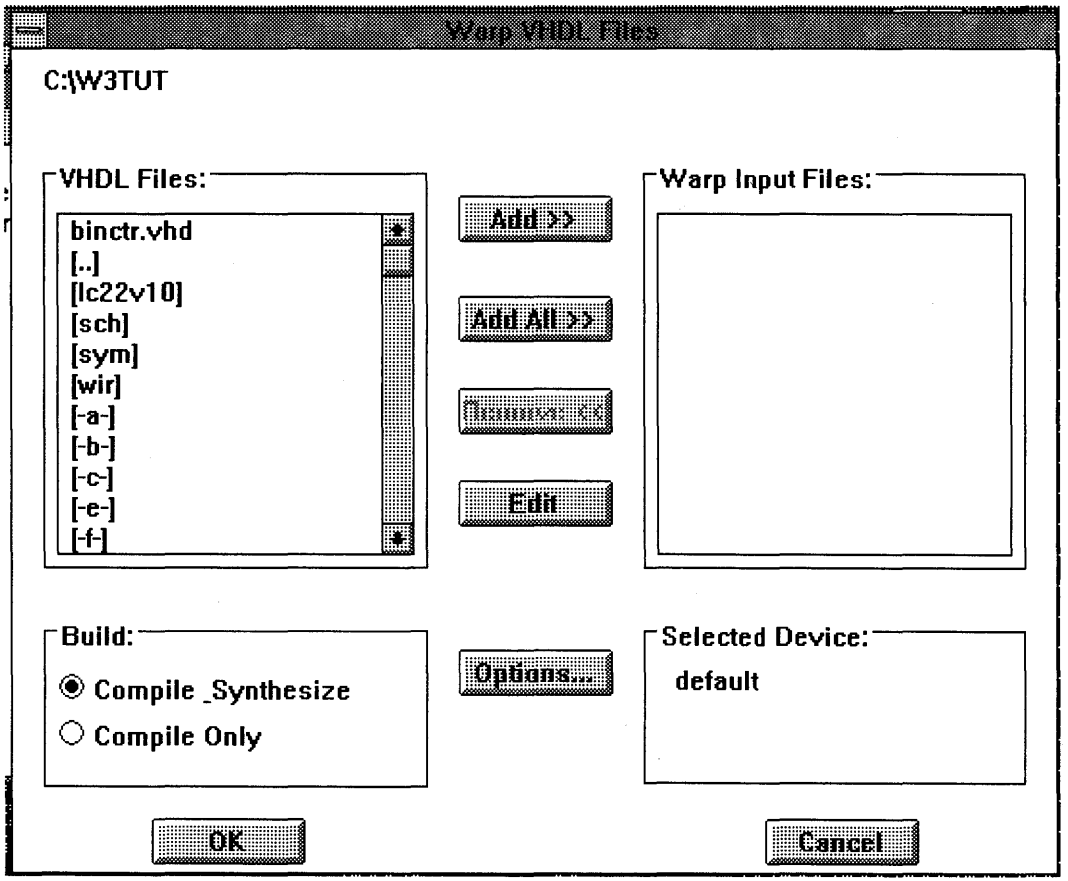


Figure 2-9. Warp dialog box (1).

2.7. Generate a symbol for the "binctr" VHDL description.

Once the binctr VHDL description is written, the next step is to create a binctr symbol that can be instantiated onto a schematic.

To create the symbol, double-click on the icon for the Viewtext window that you previously minimized.

When the Viewtext window appears, select "VHDL to Symbol" from the File menu. This runs the "vhdl->sym" application.

The vhdl->sym window appears, informing you of the application's progress. When the window banner reads "inactive vhdl->sym", you know the application is complete.

If no errors were reported, close the vhdl->sym window, then dismiss the Viewtext window and go on to the next step of the Tutorial.

If errors were reported, go back and carefully edit the BINCTR.VHD file. Make sure that its three parts--the package declaration, entity declaration, and architecture--read exactly as shown in Figures 2-5 through 2-7.

NOTE

Experienced users can use ViewDraw to verify that the BINCTR symbol has the correct number of ports, the correct port names, and the correct properties--ESPECIALLY the "vhdluse" property, which should be set to the name of the package (e.g., "binctr_pkg").

2.8. Create the schematic for the "refill" circuit.

Once you have a symbol for the binctr component, you can create a schematic that incorporates that component (and several others) into a larger circuit.

In the following pages, you will:

1. call up the components of the refill circuit from a library and place them onto the schematic sheet;
2. position the components relative to each other, in preparation for wiring;
3. label input and output ports;
4. wire the components together; and
5. save the finished schematic.

To start, double-click on the icon of the Viewdraw window, which you previously minimized.

2.8. Create the schematic for the "refill" circuit.

2.8.1. Instantiate components

The process of calling up a component from a library and placing it on a schematic sheet is called "instantiating" a component. In this section of the Tutorial, you'll instantiate the components in the refill circuit.

Components to be instantiated include: two binctr components, a two-input AND gate, one clock pad, four input ports, and three output ports.

To instantiate a component:

Select "Add/Comp" from the Viewdraw menu bar. The Viewdraw Add Component dialog box appears (Figure 2-10).

The Add Component dialog box is similar to the File Open dialog box, except it includes no option for opening schematics. The lower half of the dialog box lists libraries from which you can select symbols. The name of the current library is darkened. The upper half of the dialog box shows the names of the symbols available in the current library, in a scrollable list.

Find the component you want to instantiate by clicking on a library name in the lower half of the dialog box, then clicking on a component name from the upper half with the left mouse button. Then, move the cursor to somewhere (anywhere) in the schematic. Click the middle mouse button to place an instance of the component in that spot. Move the cursor, and click the middle mouse button to place another instance of the component. Repeat until you have instantiated the component as often as necessary. After selecting all the components, click on "Cancel".

Don't worry about placement; you can re-position each instance later. For now, just get instances onto the schematic sheet.

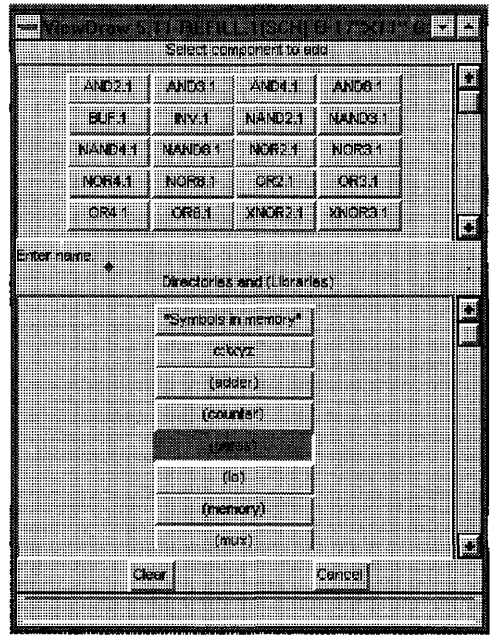


Figure 2-10. Viewdraw Add Component dialog box.

The components you'll be instantiating, the libraries in which they can be found, and the names of the symbols are as follows:

Quantity	Component	Library	Symbol Name
2	binctr	current project directory	BINCTR.1
1	two-input AND	(gates)	AND2.1
1	clock pad	(io)	CKPAD.1
4	input port	(io)	IN.1
3	output port	(io)	OUT.1

When you get finished, the sheet should look something like Figure 2-11. At this point, you can dismiss the dialog box by clicking "Cancel".

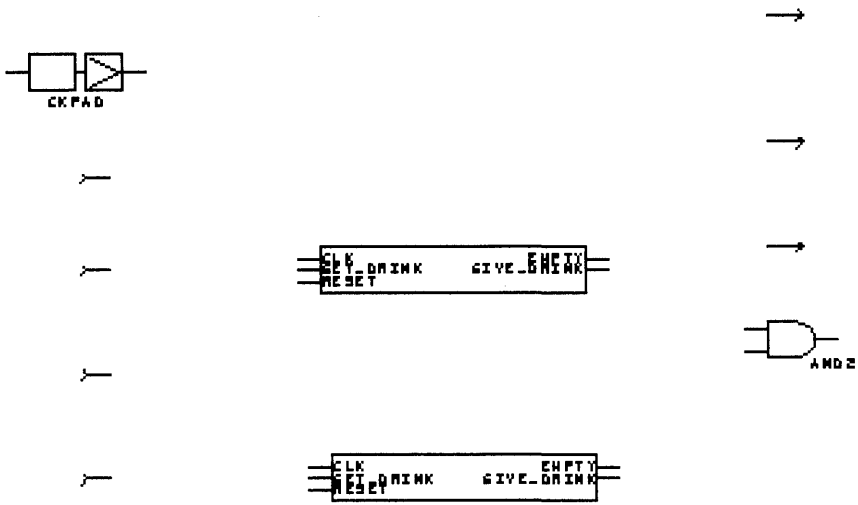


Figure 2-11. Components for the REFILL Schematic.

2.8. Create the schematic for the "refill" circuit.

2.8.2. Position Components

Once the components are instantiated on the schematic sheet, you will want to position them to enhance the readability and ease-of-wiring of the schematic.

You'll want to position the input ports on the left side of the schematic, the binctr components just to the right of the input ports, the AND gate just to the right of the binctr components, and halfway between them vertically, and the output ports on the right of the schematic. (See Figure 2-12.)

To position a component:

- Select the component, using the left mouse button;
- Press 'm' on the keyboard, or select Edit/Move from the menu bar;
- Move the cursor to where you want the component to be;
- Click the middle mouse button.

Repeat until all components are positioned as shown in Figure 2-12.

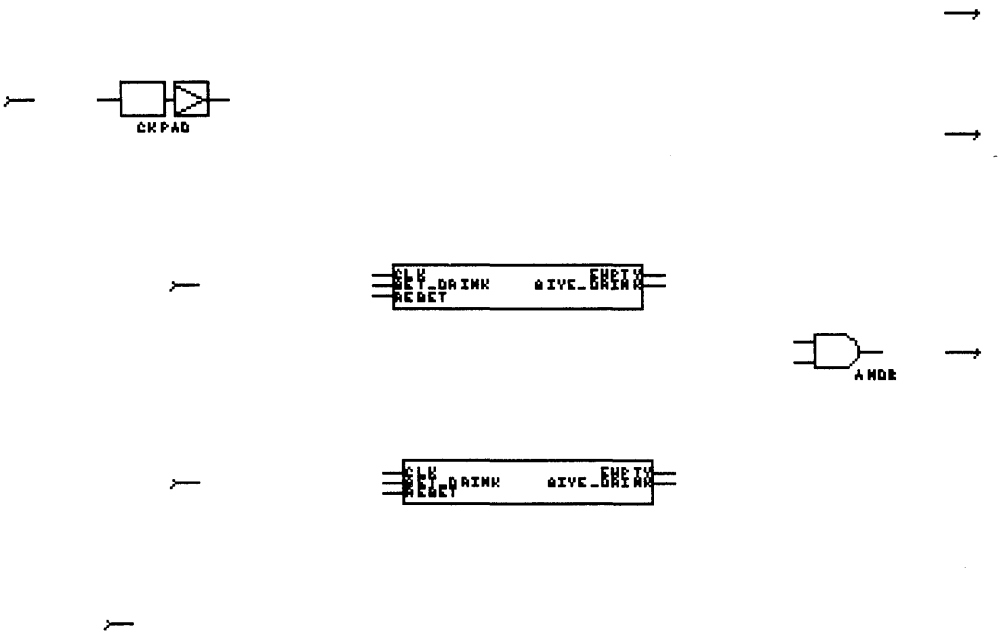


Figure 2-12. Component Positions for the REFILL Schematic.

2.8. Create the schematic for the "refill" circuit.

2.8.3. Label Ports

Once components are positioned on the schematic, it's a good idea to label the input and output ports of your design. Doing so greatly facilitates correct wiring later.

From top-to-bottom on the schematic, we'll name the input ports "clk", "get_pepsi", "get_coke", and "reset". We'll name the output ports "give_pepsi", "give_coke", and "refill". When this step is completed, the schematic should look like Figure 2-13.

To label the component ports:

- Select the component, using the left mouse button.
- Select "Add/Label..." from the Viewdraw menu bar, or type "l" ("el") at the keyboard.
- Click on the "label" line of the dialog box.
- Delete the contents of the label line, if any.
- Type the new label, e.g., "clk". Case doesn't matter; all characters will be upper-case on the schematic.
- Click "OK".
- Move the cursor to where you want the label to appear (i.e., next to the port its associated with).
- Click the middle mouse button to place the label.

Repeat until all ports are labeled.

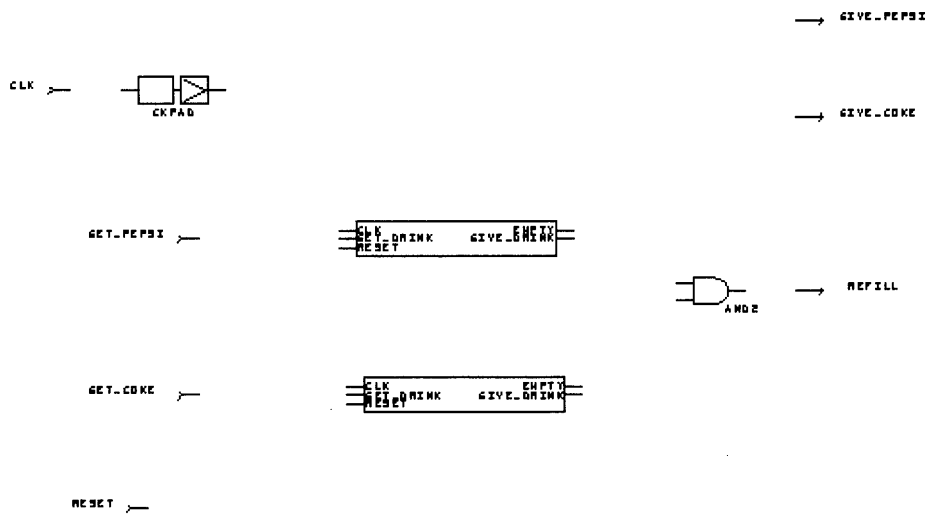
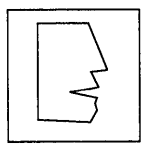


Figure 2-13. Components Positioned and Ports Labeled, for the REFILL Schematic.



Here's a shortcut method for adding several labels to input and output ports of a component at once:

1. select a port;
2. select "Label" from the Add menu;
3. type the names of the ports to be labeled as one comma-separated list, DO NOT insert spaces after the commas. The string should look like:
   "clk,get_pepsi,get_coke,..."
4. click the middle mouse button. The first label in the list appears.
5. position the label, then click the middle mouse button.
6. click the left mouse button to select the next to be labeled.
7. repeat steps 4 through 6 for each port to be labeled.

2.8. Create the schematic for the "refill" circuit.

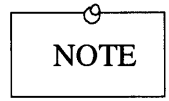
2.8.4. Wire Components Together

Once components are positioned and labeled, it's time to wire them together to make the circuit.

The final circuit should look like Figure 2-14.

To connect two components:

- Select Add/Net from the Viewdraw menu bar, or type 'n' at the keyboard;
- Move the cursor to the origin of the net and click the middle mouse button;
- Form the net, specifying points along the net by clicking the middle mouse button. Click the left mouse button once to back up one segment on the net, twice to back up two segments, etc.;
- To connect the net to a component pin, move the cursor to a point on the pin and click the middle mouse button;
- To connect the net to another net or bus, move the cursor to a point on the net or bus and click the middle mouse button;
- To leave the net dangling, form the net and click the middle and then the right mouse buttons.



Make sure to connect all input pins on components to something. Viewdraw does not allow unconnected inputs.

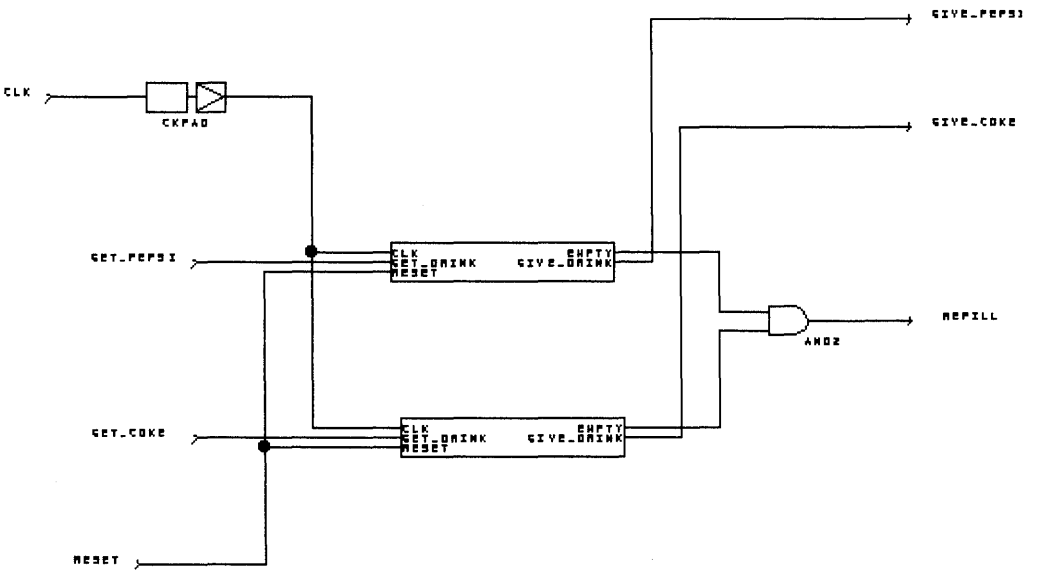


Figure 2-14. Final REFILL Schematic.

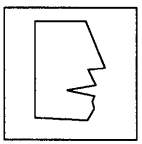
2.8. Create the schematic for the "refill" circuit.

2.8.5. Save the Schematic

Once all the components are positioned, labeled, and connected, save the schematic.

Select File/Write from the Viewdraw menu bar, or type "w" at the keyboard. Viewdraw saves the schematic and warns you about unconnected pins, unlabeled ports, and other potential problems.

Actually, it's a good idea to write the file frequently while drawing a schematic. Just ignore Viewdraw's warning messages until you think you're finished. Then, an error message could be telling you that you aren't finished yet....



If Viewdraw gives you an error message when you write a file, you probably forgot to wire or label something.

If Viewdraw produces any error messages at this stage, go back and make sure that:

- all component pins are connected to something;
- all input and output ports are labeled;
- all nets are wired so that the schematic looks EXACTLY like Figure 2-14.

If no error messages appear, you're finished with the schematic. Quit Viewdraw and go back to the Workview Plus Cockpit.

2.9. Generate a VHDL Description of the Schematic

The expt1076 program generates a VHDL description from a schematic.

To run expt1076, double-click on the expt1076 icon from the Cockpit.

On IBM PC's and compatibles, a dialog box appears, showing a command line to execute. The program name is CYPEXPT; the name of the schematic appears to the right of it. Make sure that the command line reads "CYPEXPT REFILL", then click "OK".

A window appears, informing you of the progress of the expt1076 application. When the banner on this window reads "inactive expt1076", the application is complete. Close the application window.

On Sun workstations, type "refill" on the line labeled "Design Name", then click "Accept".

2.10. Run Warp

Warp is the VHDL synthesis compiler for *Warp3*.

Warp takes VHDL descriptions as input, and produces JEDEC or QDIF files as output. JEDEC files are used to program PLDs and CPLDs. QDIF files are used as input to the SpDE tools, which produce output files used to program pASICs.

The first time you ran *Warp*, earlier in the Tutorial, it was simply to verify that the binctr.vhd file was syntactically correct.

On the following pages, you'll run *Warp* to produce a JEDEC or QDIF file. You'll perform the following steps:

- Start *Warp*;
- Select appropriate options and run *Warp* to produce a JEDEC file targeting a PLD (a C22V10), OR
- Select appropriate options and run *Warp* to produce a QDIF file targeting a pASIC (a C382A).

2.10. Run Warp

2.10.1. Start Galaxy

Galaxy is the user interface for the *Warp* VHDL synthesis compiler.

To start Galaxy, double-click on the Warp icon in the Cockpit. Click on "OK" from the ensuing dialog box. Close the ensuing "About" window. The Galaxy window appears (Figure 2-15).

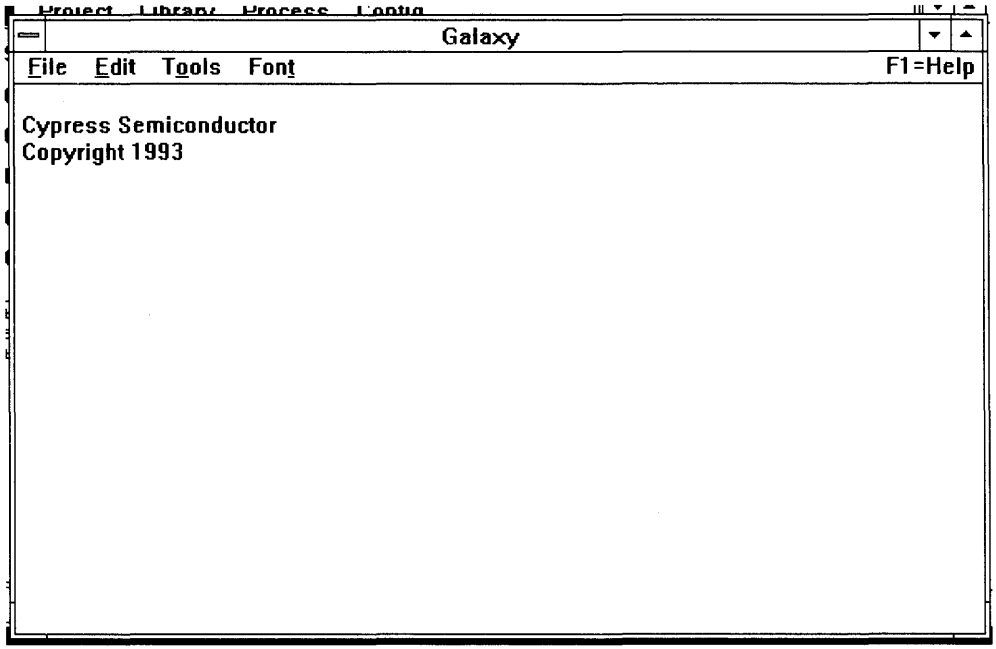


Figure 2-15. The Galaxy window (2).

2.10. Run Warp

2.10.2. Start *Warp*

To bring up the *Warp* dialog box, select Tools/Run Warp Menu from the Galaxy menu bar.

The *Warp* dialog box appears (Figure 2-16).

This dialog box lists the VHDL files available in the current directory on the left side, and the VHDL files selected for compilation/synthesis on the right side.

To select a VHDL file for compilation/synthesis, click on the name of the file on the left side, then click on the "Add" button.

To de-select a file for compilation/synthesis, click on the file's name on the right side, then click on the "Remove" button.

The asterisk (*) alongside the name of the REFILL.VHD file indicates that *Warp3* has processed this file and identified it as the highest-level description of a multi-file VHDL hierarchy. (The asterisk is NOT part of the file name, however; it's just a visual aid, to help you identify the highest-level files in projects.)

Now, let's specify a target device...

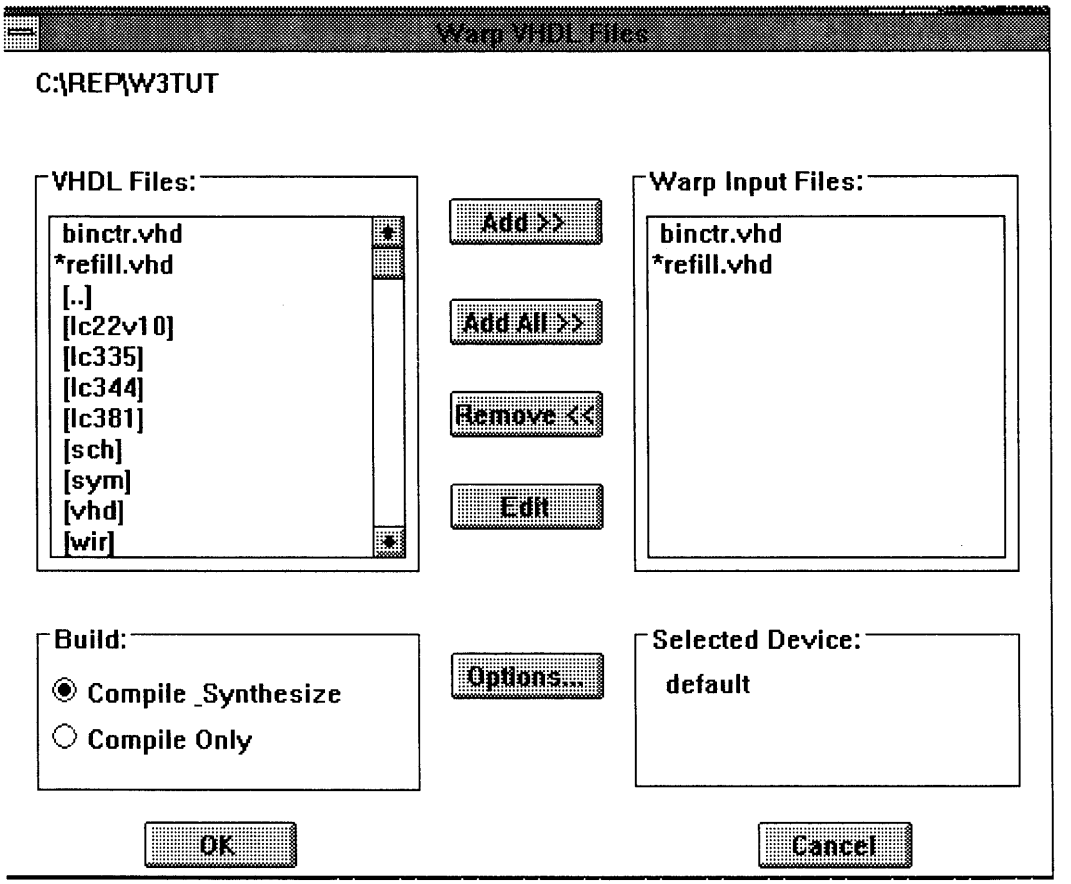


Figure 2-16. Warp dialog box (2).

2.10. Run Warp

2.10.3. Synthesize a VHDL Description

To specify a target device (along with other compilation/synthesis options), click on the Options button from the *Warp* dialog box.

The *Warp* options dialog box appears (Figure 2-17).

For this tutorial, you can synthesize the VHDL description in file REFILL.VHD to either a C22V10 PLD or a C382A pASIC.

To select the C22V10, scroll through the "Devices:" list and highlight "C22V10". Click on "Create Viewsim Model" in the Output portion of the dialog box, then click "OK".

To select the C382A, scroll through the "Devices:" list and highlight "C382A", then click "OK". (The "Create Viewsim Model" box is grayed out when you target a pASIC; the Viewsim mo that *Warp3* needs will be created later, in Section 2.11.1.)

When the *Warp* dialog box re-appears, click on "OK". *Warp* runs, printing messages to keep you appraised of its progress.

If you are targeting a PLD and "Create Viewsim Model" is checked, the VHDL Analyzer window comes up when compilation and synthesis conclude. When the banner on this window reads "inactive Warp", the application is complete. Close the VHDL analyzer window. (On Sun workstations, no window comes up for the VHDL Analyzer; the application's messages simply appear in the Powerview Cockpit's text area.)

NOTE

On IBM PC's and compatibles, *Warp* compiles each device's libraries the first time that you target a compilation and synthesis for that device. This takes extra time. *Warp* runs faster on subsequent runs.

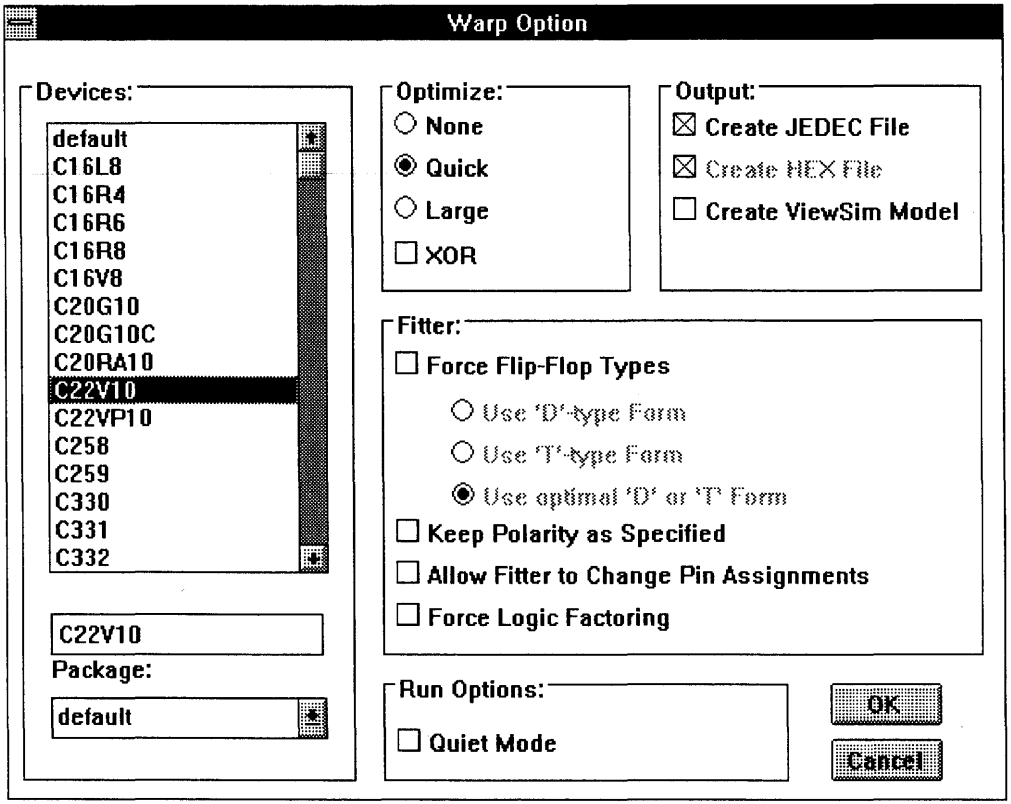


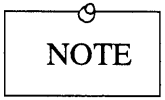
Figure 2-17. Warp Options dialog box.

This operation generates three files of particular interest (among others):

- The first is named REFILL.JED (if you targeted the 22V10) or REFILL.QDF (if you targeted the C382A). The .JED file can be used to program the 22V10. The .QDF file is used by the SpDE tools to produce a .LOF file, which can be used to program the C382.

- The second file of interest is found in a sub-directory that *Warp* creates called "vhd", and is also named REFILL.VHD. This .VHD file, however, is a structural description of the synthesized circuit.
- The third is named REFILL.RPT. It contains pinout and timing information, along with lots of other information about the final synthesized design.

Exit the Galaxy window. (Choose "Exit" from the File menu.)



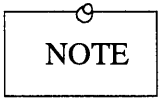
If compilation errors occur, do the following:
1. make sure the text of your binctr.vhd file is entered exactly as shown earlier in this chapter; better yet, copy it from the \w3tutor sub-directory of the *Warp* directory.

2. run expt1076 again.

3. run *Warp* again.

2.11. Run SpDE tools to process the QDIF file

If you targeted a pASIC (i.e., any C380-series part) when synthesizing a design in *Warp*, you must process the resulting .QDF file with the SpDE tools in order to produce a file with which you can program the device.



Perform the steps in Sections 2.11 and 2.11.1 only if you targeted the C382A pASIC in step 2.10.3. Otherwise, skip to step 2.12

To process a QDIF file in order to target pASICs, double-click on the "Place&Rte" icon in the Cockpit.

Click on "OK" in the ensuing dialog box. The SpDE window appears (Figure 2-18):

- Select File/Import/QDIF from the SpDE menu bar.
- Find your project directory, then select REFILL.QDF and click"OK".
- Select Tools/Run All Tools from the SpDE menu bar.

The SpDE tools will run. A small window appears, to keep you informed about the progress of the SpDE tools.

When a dialog box appears, telling you that all SpDE tools ran successfully, click on "OK".

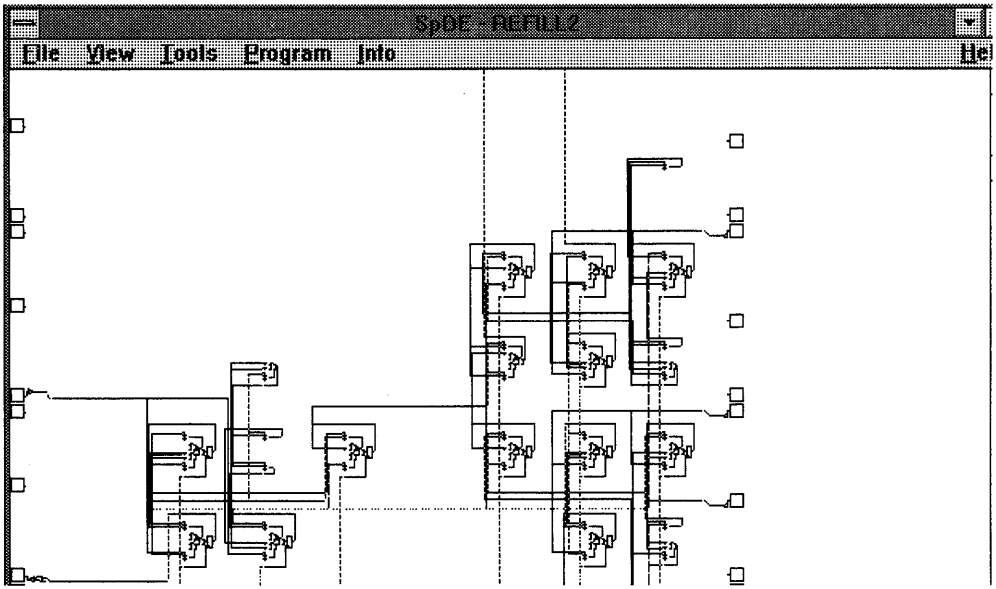


Figure 2-18. SpDE window.

Finally, select File/Save from the SpDE menu bar, save your changes, and File/Exit to exit.

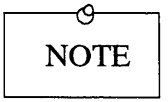
2.11. Run SpDE tools to process the QDIF file

2.11.1. Run pASIC->VSim to Generate a Viewsim Model

To generate a Viewsim model from the output of the SpDE tools, double-click on the "pASIC->VSim" icon in the Cockpit.

A dialog box appears, containing a command line to be executed. Make the command line read "SPDE2VL refill", then click on "OK".

A window appears, informing you of the progress of the application. When the banner of this window reads "inactive pASIC->VSim", the application is complete.



Ignore the messages at the bottom of the window. If the application reports 0 errors and 0 warnings in the sixth text line from the bottom, the application ran successfully.

Close the pASIC->VSim window.

2.12. Run CypBack to back-annotate pin assignment information

To back-annotate pin assignment information added to the design during compilation and synthesis, double-click on the "CypBack" icon from the Cockpit.

A dialog box appears, with a command line to be executed. Edit the command line to read "CYPBCK refill", then click on "OK".

A window appears, informing you of the progress of the application. When the banner of this window reads "inactive CypBack", the application is complete. Close the window.

NOTE

1. To see the modifications that CypBack made, do a READ on the schematic from within View-Draw. The modifications (i.e., the pin numbers) will be displayed.

2. Pin numbers do not appear on pins connected to hi-drive pads, clock pads, or busses.

2.13. Run ViewSim to simulate the behavior of the design

Once the design is synthesized, it's a good idea to simulate its behavior, to ensure that it functions as intended.

NOTE

Before performing this step of the Tutorial, copy the refill.cmd file from the Warp directory (its default location is c:\warp\examples\w3tutor\ refill.cmd) to your project directory.

To simulate the behavior of the design, double-click on the Viewsim icon in the Cockpit. A dialog box appears. Make sure the command line reads "WVSIM refill", then click on OK.

Viewsim starts up. When the Viewsim window appears ('SIM>'), type "refill" at the command line. The refill.cmd file runs, executing the following sequence of Viewsim commands (not necessarily echoed to the screen):

```
> wave REFILL.wfm clk reset get_pepsi give_pepsi get_coke
   give_coke refill
> clock clk 0 1
> h reset
> l get_pepsi
> l get_coke
> cycle
> l reset
> cycle
> h get_pepsi
> cycle 4
> l get_pepsi
> h get_coke
> cycle 4
> l get_coke
> h reset
> cycle
> l reset
```

This sequence of commands does the following:

- sets up the waveforms to be traced (clk, reset, get_pepsi, give_pepsi, get_coke, give_coke, and refill);
- sets up the clock signal;
- initializes the inputs to the simulation, sets reset high for one clock cycle, then sets reset to low;
- sets get_pepsi to high for four clock cycles;
- sets get_pepsi to low, then sets get_coke to high for four clock cycles;
- sets get_coke to low, then sets reset to high for one clock cycle.

Note how the fourth request to get a Pepsi does not result in "give_pepsi" going high. (The Pepsi bin is empty.) The refill signal does not go high, however, because both bins are not empty. Later, the refill signal goes high after the third Coke is delivered. A fourth request for a Coke is ignored. Finally, the refill signal goes low when reset is asserted to indicate that the bins have been replenished.

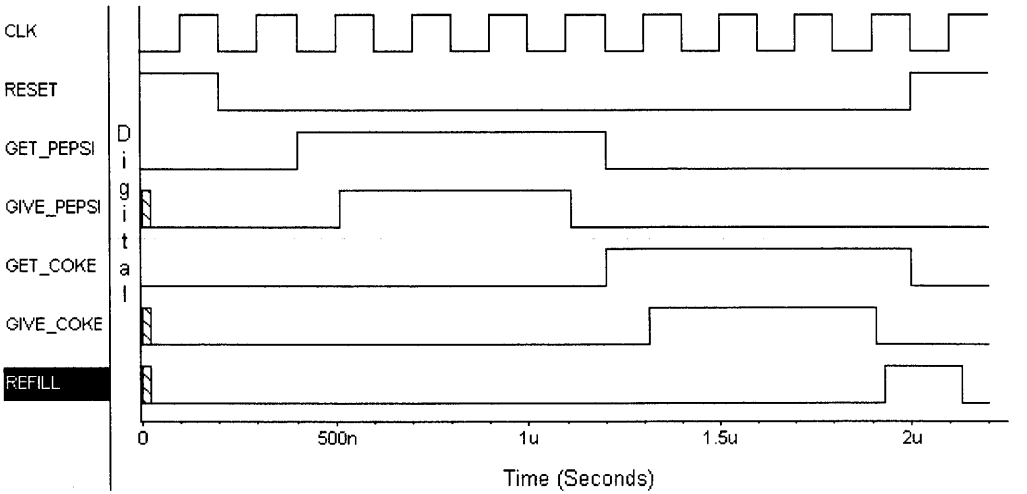


Figure 2-19. Traces from Simulation of REFILL Schematic.

NOTE

If you are simulating the circuit you designed in Chapter 3 (i.e., the one in which binctr is described in a schematic versus a VHDL description), you may notice a difference between simulations:

In the schematic version, the refill signal goes high at the same time as the last "give_coke" signal. In the behavioral version, the refill signal goes high one clock period after the last "give_coke".

This is a good illustration of the need to study simulation results carefully. In the case of a drink machine, it makes no difference if the machine realizes it's empty as it gives the last drink, or one clock cycle later. In a different application, however, such a difference could be critical.

Code could be added to the behavioral description, or components added to the schematic, in order to make the two simulations identical.

2.14. Conclusion

Warp3 produces two files in the project directory that can be transferred to device programmers: JEDEC files (for PLDs/CPLDs) and LOF files (for pASICs).

JEDEC files (.JED extension) generated by *Warp* can be transferred to device programmers for PLDs and CPLDs. LOF files (.LOF extension) generated by SpDE can be transferred to device programmers for pASICs. See the manual that came with your device programmer for information on transferring and using this file to program devices.

Chapter 3 Exercise 2: Structural Description

About This Chapter

Overview

This chapter builds on the design developed in Chapter 2, by converting it to a two-level structural description. In this chapter, we'll substitute a structural description of the binctr component for the VHDL description we created in the previous chapter. Then, we'll make the binctr symbol in the refill schematic point to the schematic instead of the VHDL description, to demonstrate how easily *Warp3* lets you switch between the two representations.

The steps you will follow in this chapter include:

- making the binctr symbol on the refill (high-level) schematic "point to" a structural description—another schematic—instead of a VHDL description;
- creating the schematic for the low-level design; instantiating and positioning components; wiring components together; and labeling input and output ports;
- "exporting" the new refill schematic file to obtain a VHDL description of the entire schematic;

- running *Warp* to synthesize the schematic's VHDL description;
- simulating the behavior of the design;
- back-annotating pin information added during synthesis.

3.1. Make the "binctr" symbol point to a schematic

The first step in this Tutorial exercise is to make the binctr symbol on the refill schematic point to a schematic representation of the binctr component, as opposed to a textual one.

To do this, use the following steps:

1. Bring up the refill schematic (from the Viewdraw icon in the Cockpit, open the refill schematic.
2. Select the binctr symbol on the refill schematic.
3. Select "Level/Push/Symbol" from the File menu.
4. Select "Block/Type" from the Change menu. Click on the "Composite" button. This changes the symbol to point to a schematic ("composite") instead of a VHDL description ("module").
5. Select "Write" from the File menu to save the symbol.
6. Select "Level/Pop" from the File menu. This returns you to the refill schematic.
7. Select "Write" from the File menu to utilize the new symbol information.
8. Select "Level/Push" from the File menu (ensuring binctr symbol is still selected). A new blank schematic sheet appears.

3.2. Create the schematic for the "binctr" circuit.

The next step in this Tutorial exercise is to create the schematic for the binctr circuit.

In the following pages of the Tutorial, you will:

1. call up the components of the binctr circuit from a library and place them onto the schematic sheet;
2. position the components relative to each other, in preparation for wiring;
3. label input and output ports;
4. wire the components together; and
5. save the finished schematic.

3.2. Create the schematic for the "binctr" circuit.

3.2.1. Instantiate components

The process of calling up a component from a library and placing it on a schematic sheet is called "instantiating" a component.

Components to be instantiated include: one TTL163 counter, two two-input AND gates, two inverters, one D-type flip-flop, one Vcc symbol, one Gnd symbol, three input ports, and two output ports.

To instantiate a component:

Select "Add/Comp" from the Viewdraw menu bar. The Viewdraw Add Component dialog box appears (Figure 3-1).

The Add Component dialog box is similar to the File Open dialog box, except it includes no option for opening schematics. The lower half of the dialog box lists libraries from which you can select symbols. The name of the current library is darkened. The upper half of the dialog box shows the names of the symbols available in the current library.

Find the component you want to instantiate by clicking on a library name in the lower half of the dialog box, then clicking on a component name from the upper half. Then, move the cursor to somewhere (anywhere) in the schematic. Click the middle mouse button to place an instance of the component in that spot. Move the cursor, and click the middle mouse button to place another instance of the component. Repeat until you have instantiated the component as often as necessary.

Don't worry about placement; you can re-position each instance later. For now, just get instances onto the schematic sheet.

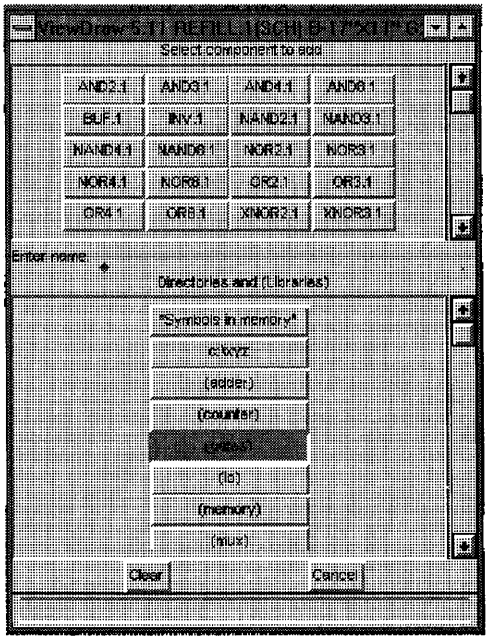


Figure 3-1. Viewdraw Add Component dialog box.

The components to instantiate, the libraries in which they can be found, and the names of the symbols are as follows:

Quantity	Component	Library	Symbol Name
1	TTL163 counter	(ttl)	TTL163.1
2	two-input AND	(gates)	AND2.1
2	inverter	(gates)	INV.1
1	D-type flip-flop	(memory)	DFF.1
3	input port	(io)	IN.1
2	output port	(io)	OUT.1
1	Vcc	(io)	VCC.1
1	Gnd	(io)	GND.1

When you get finished, the sheet should look something like Figure 3-2. At this point, you can dismiss the dialog box by clicking "Cancel".

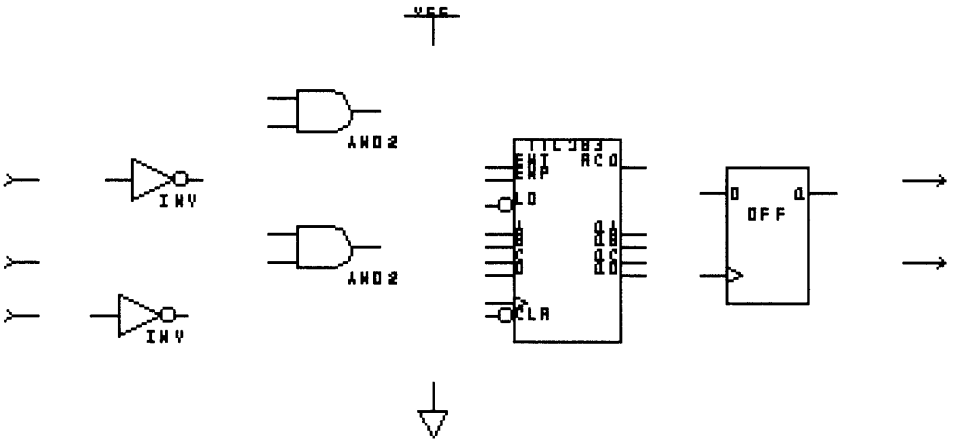


Figure 3-2. Components for the BINCTR Schematic.

3.2. Create the schematic for the "binctr" circuit.

3.2.2. Position Components

Once the components are instantiated on the schematic sheet, you will want to position them to enhance the readability and ease-of-wiring of the schematic.

You'll want to position the TTL163 in the center of the schematic, the input ports on the left side of the schematic, the output ports on the right, and the other logic elements at places where it will be easy to see the logic flow within the circuit. (See Figure 3-3.)

To position a component:

- Select the component, using the left mouse button;
- Press 'm' on the keyboard, or select Edit/Move from the menu bar;
- Move the cursor to where you want the component to be;
- Click the middle mouse button.

Repeat until all components are positioned.

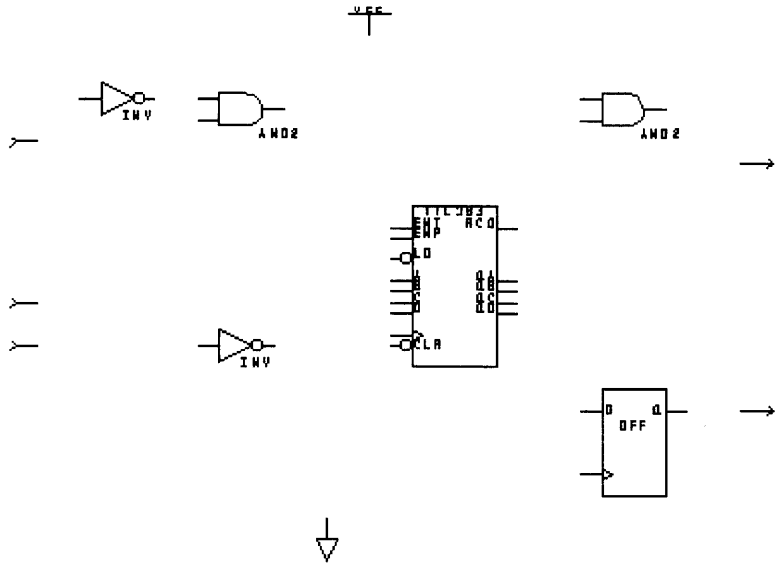


Figure 3-3. Component Positions for the BINCTR Schematic.

3.2. Create the schematic for the "binctr" circuit.

3.2.3. Label Ports

Once components are positioned on the schematic, it's a good idea to label the input and output ports of your design. Doing so greatly facilitates correct wiring later.

We'll name the input ports "clk", "get_drink", and "reset". We'll name the output ports "give_drink", and "empty". When this step is completed, the schematic should look like Figure 3-4.

To label the component ports:

- Select the component, using the left mouse button.
- Select "Add/Label..." from the Viewdraw menu bar, or type "l" ("el") at the keyboard.
- Click on the "label" line of the dialog box.
- Delete the contents of the label line, if any.
- Type the new label, e.g., "clk". Case doesn't matter; all characters will be upper-case on the schematic.
- Click "OK".
- Move the cursor to where you want the label to appear (i.e., next to the port its associated with).
- Click the middle mouse button to place the label.

Repeat until all ports are labeled.

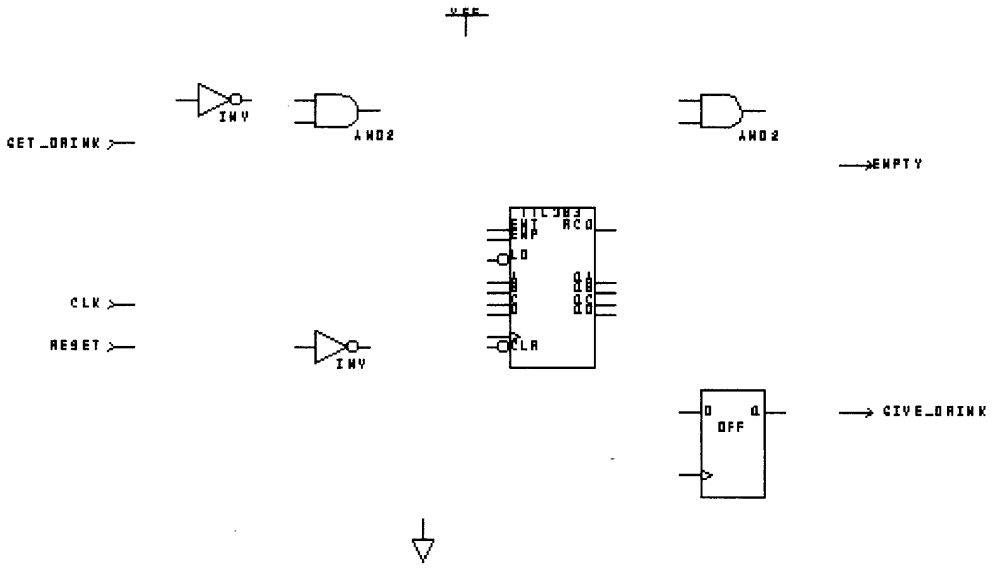
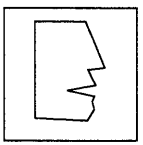


Figure 3-4. Components Positioned and Ports Labeled, for the BINCTR Schematic.



Here's a shortcut method for adding several labels to input and output ports of a component at once:

1. select a port;
2. select "Label" from the Add menu;
3. type the names of the ports to be labeled as one comma-separated list, DO NOT insert spaces after the commas. The string should look like:
   "clk,get_drink,reset,..."
4. click the middle mouse button. The first label in the list appears.
5. position the label, then click the middle mouse button.
6. click the left mouse button to select the next to be labeled.
7. repeat steps 4 through 6 for each port to be labeled.

3.2. Create the schematic for the "binctr" circuit.

3.2.4. Wire Components Together

Once components are positioned and labeled, it's time to wire them together to make the circuit.

The final circuit should look like Figure 3-5.

To connect two components:

- Select Add/Net from the Viewdraw menu bar, or type 'n' at the keyboard;
- Move the cursor to the origin of the net and click the middle mouse button;
- Form the net, specifying points along the net by clicking the middle mouse button. Click the left mouse button once to back up one segment on the net, twice to back up two segments, etc.;
- To connect the net to a component pin, move the cursor to a point on the pin and click the middle mouse button;
- To connect the net to another net or bus, move the cursor to a point on the net or bus and click the middle mouse button;
- To leave the net dangling, form the net and click the middle and then the right mouse buttons.

NOTE

Make sure to connect all input pins on components to something. Viewdraw does not allow unconnected inputs.

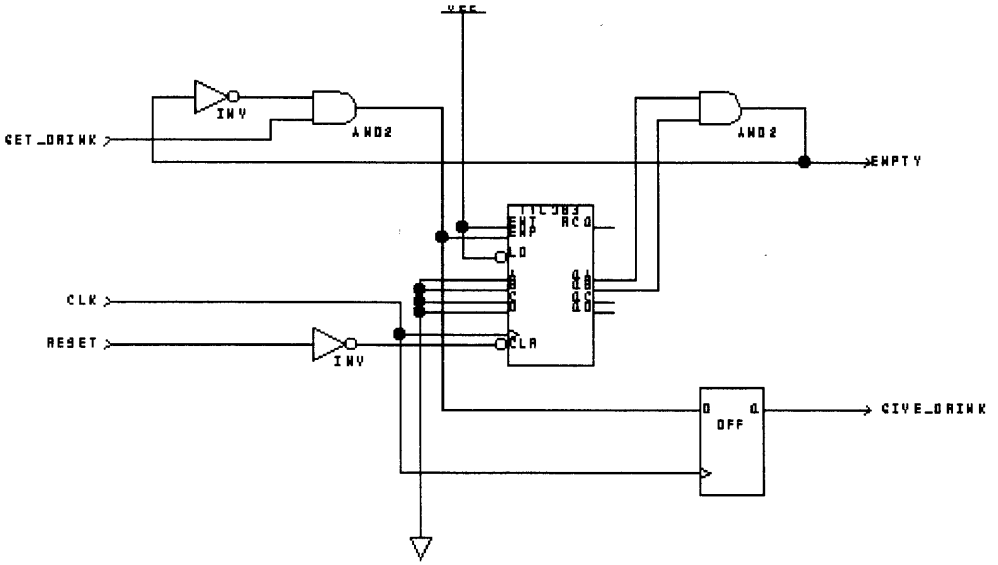


Figure 3-5. Final BINCTR Schematic.

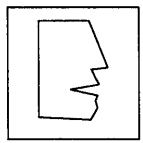
3.2. Create the schematic for the "binctr" circuit.

3.2.5. Save the binctr Schematic

Once all the components are positioned, labeled, and connected, save the binctr schematic.

Select File/Write from the Viewdraw menu bar. Viewdraw saves the schematic and warns you about unconnected pins, unlabeled ports, and other potential problems.

Actually, it's a good idea to write the file frequently while drawing a schematic. Just ignore Viewdraw's warning messages until you think you're finished. Then, an error message could be telling you that you aren't finished yet....



If Viewdraw gives you an error message when you write a file, you probably forgot to wire or label something.

If Viewdraw produces any error messages at this stage, go back and make sure that:

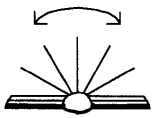
- all component pins are connected to something;
- all input and output ports are labeled;
- all nets are wired so that the schematic looks EXACTLY like Figure 3-5.

If no error messages appear, you're finished with the binctr schematic. Select "Level/Pop" from the File menu to return to the refill schematic. Select File/Write from the Viewdraw menu bar to save the refill schematic.

3.3. Export the refill Schematic, etc.

The steps to be completed in the remainder of this Tutorial exercise are identical to those in Exercise 1.

Having generated a schematic for the binctr component and making the refill schematic use the symbol for this component, you can now complete the Tutorial exercise by exporting the refill schematic with expt1076, compiling and synthesizing it with *Warp*, etc., as in Exercise 1.



Turn to Section 2.9 of Exercise 1.

You can complete this exercise from that point.

Chapter 4

Exercise 3: Using Attributes in pASIC Design

About This Chapter

Warp3's tool set can convert schematic diagrams into VHDL, synthesize the VHDL, and place and route the resulting file for a pASIC380 FPGA, all via sophisticated automatic algorithms.

Sometimes, however, the output of these automatic algorithms is not satisfactory, for one reason or another:

- perhaps there is a critical set of I/O pins whose signals must get into and out of the chip as quickly as possible;
- perhaps you wish to hand-pack a design, to get the last nanosecond of performance out of the part, or to help fit a design that is proving difficult to fit using the automatic algorithms;
- or perhaps you don't want all the optimization performed by the *Warp3* tools. Often, the optimization process makes internal signals "disappear," i.e., combines logic elements in such a way that a signal or component that appeared in the original schematic does not appear in the final layout. You may want a particular component to come through the synthesis-placement-routing process with all its input and output signals preserved.

In such cases, you may wish to use the FIXED_FF and DONT_TOUCH attributes to modify the synthesis and the placement and routing of the final layout.

The FIXED_FF attribute specifies the logic cell location in which a flip-flop will appear in the final layout. Each logic cell in a pASIC has a column-and-row designation, where the columns are alphabetic characters and the rows are numbers. Thus, a logic cell designation of "B3" indicates the second cell from the left and the third cell down from the upper-left corner of the layout.

The DONT_TOUCH attribute specifies that a PAFRAG component and its input and outputs are to pass through the synthesis, placement, and routing process with its structure, signal naming, and signal ordering preserved.

This tutorial demonstrates the use and effect of the FIXED_FF and DONT_TOUCH attributes on pASIC designs. In addition, the conventions used by the *Warp3* tools to assign names to internal signals are also discussed, as well as how knowledge of these conventions can help you follow the placement and routing of your original design in the completed layout. The tutorial chapter ends with some notes about the use of markers, pads, and triout/bufoe components in pASIC designs.

This tutorial assumes that you have already run the previous two tutorials, and so are already familiar with the *Warp3* design process and how to run the various tools. If you are not, please go through those earlier tutorials now; it will save you many questions and much frustration.

4.1. Modify the binctr Schematic

To start this tutorial exercise, we will modify the binctr schematic from Chapter 3, substituting a PAFRAG_A component for two other components.

To do this:

1. Bring up the binctr schematic in Viewdraw.
2. Delete the inverter and AND2 components from the upper left of the schematic.
3. Insert a PAFRAG_A component from the MCPARTS library. Connect the GET_DRINK input to pin A5 of the PAFRAG_A component. Tie pins A1 and A3 of the PAFRAG_A to Vcc. Tie pins A2 and A4 of the PAFRAG_A to GND. Connect pin A6 to the output of the AND2 component in the upper right quarter of the schematic. The result should look like Figure 4-1.
4. Write the file to BINCTR2.
5. Quit Viewdraw.
6. Run Expt1076 on the BINCTR2 file.
7. Run *Warp* on the resulting BINCTR2.VHD file. In the Warp Options screen, target the C382A device, and the CY7C382A-0JC package. Use the default option settings. Click OK from the Warp Options window, then select Compile and Synthesize and click OK in the main *Warp* window. Exit *Warp* when compilation and synthesis are completed.

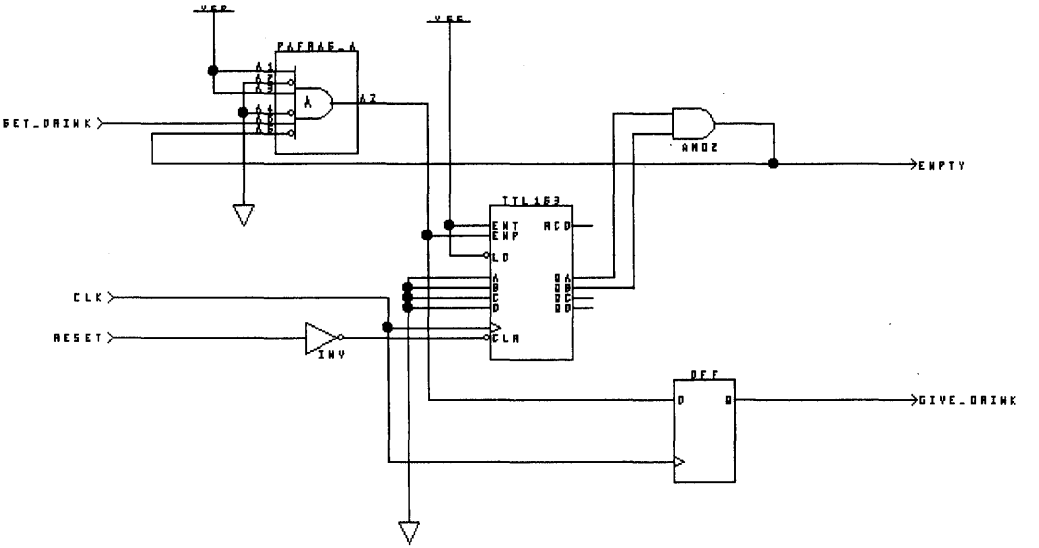


Figure 4-1. Modified BINCTR Schematic.

8. Run the SpDE tools by double-clicking on the "Place&Rte" icon in the Cockpit. Choose "Import->QDIF" from the File menu, then select the BINCTR2.QDF file from the ensuing dialog box, and click on "OK". Ignore any warning messages about unused gates.

9. Once the BINCTR2.QDF file has been imported into SpDE, select "Run Tools..." from the Tools menu. Select all the tools from the ensuing dialog box except ATVG (it isn't needed for this tutorial), then click on "Run".

10. Save the result. You may wish to print the resulting layout, to facilitate later comparisons.

4.2. Add Attributes to Binctr2 Schematic

Having saved a second version of binctr called binctr2, we will create yet a third version by assigning the DONT_TOUCH attribute to a component and the FIXED_FF attributes to a signal.

To do this:

1. Bring up the BINCTR2 schematic in Viewdraw.
2. Select the PAFRAG_A component by clicking on it.
3. Select "Attr..." from the Add menu.
4. Type DONT_TOUCH=TRUE on the "Attribute:" line of the ensuing dialog box (making sure the "Visible" button is also selected), then click on OK. Position the attribute text near the PAFRAG_A component on the schematic, where it is readily visible.
5. Select the output signal from the DFF component in the lower right quadrant of the schematic by clicking on it.
6. Select "Attr..." from the Add menu.
7. Type FIXED_FF="G5" on the "Attribute:" line of the ensuing dialog box (making sure the "Visible" button is also selected), then click on OK. Position the attribute text near the AND2's output signal on the schematic, where it is readily visible.
8. Write the file to BINCTR3, then quit Viewdraw.
9. Follow steps 6 through 10 given in Section 4.1 to create, save, and print a physical layout for the BINCTR3 design.

4.3. Compare binctr2 and binctr3

Having synthesized, placed, and routed both binctr2 and binctr3, a comparison of the resulting layouts shows the effects of the DONT_TOUCH and FIXED_FF attributes.

The most noticeable effects were:

1. The binctr2 layout fit in three cells on the chip, while the binctr3 layout took four cells to fit. This occurred because the DONT_TOUCH attribute restricted the optimization process, so that the design could not be implemented as efficiently. While this has negligible effect on our (tiny) example design, it could make the difference between a larger design's fitting or not fitting on a target device. Moral: use the DONT_TOUCH attribute carefully.

2. In the binctr3 layout, signal GIVE_DRINK_OUT (the internal signal attached to the pin labeled GIVE_DRINK) is positioned in cell G5. This positioning was forced by the FIXED_FF attribute that we attached to the GIVE_DRINK signal. The other cells used in the design were all positioned near G5, in order to minimize the use of routing resources. In the binctr2 layout, the random seed used in the placement and routing algorithm could have caused the cells used on the chip to be placed anywhere (and probably did).

   The positioning capability that the FIXED_FF attribute gives could be used to specify the position of logic cells implementing registers attached to signals with critical timing requirements.

4.4. Modify the binctr3 Schematic

We will modify the binctr3 schematic one more time, to remove the DONT_TOUCH attribute from the PAFRAG_A component.

To do this:

1. Bring up the BINCTR2 schematic in Viewdraw.
2. Select the DONT_TOUCH=TRUE attribute from the PAFRAG_A component by clicking on it.
3. Select "Cut" from the Edit menu.
4. Write the result to BINCTR4, then quit Viewdraw.
5. Follow steps 6 through 10 given in Section 4.1 to create, save, and print a physical layout for the BINCTR4 design.

4.5. Compare binctr3 and binctr4

Comparing the binctr3 and binctr4 layouts shows the results of removing the DONT_TOUCH attribute while keeping FIXED_FF.

Removing the DONT_TOUCH attribute gave SpDE's automatic placement and routing algorithms greater freedom in laying out the design. As a result, fewer resources (pASIC cells) were needed to fit the binctr4 layout: three cells instead of four.

Note, however, that the FIXED_FF attribute still specified the logic cell location of the flip-flop whose output was signal GIVE_DRINK_OUT. Thus, the cells used in the physical layout of the binctr4 design are all near cell G5. In large designs, this restriction on positioning could affect the ability of the automatic algorithms to complete placement and routing.

Note also that, as a result of removing the DONT_TOUCH attribute from the PAFRAG_A component, the AND gate that was previously used for this component has been merged with other components, and may not be easy to find.

4.6. Getting Signal Names from the ViewSim Simulator

Once you have a physical layout for a pASIC design, you may wish to simulate it, both to check its behavior and to get a list of the signal names used in the design.

To obtain a ViewSim model file:

1. Double-click on the "pASIC-VSim" icon in the Cockpit.
2. In the ensuing dialog box, include the root name of the design that you wish to simulate, after the SPDE2VL keyword. For example, to generate a ViewSim model file for design binctr4, make this line in the dialog box read "SPDE2VL binctr4", then click on "OK".
3. Close the inactive window when SpDE2VL completes.

To run ViewSim:

1. Double-click on the "ViewSim" icon in the Cockpit.
2. In the ensuing dialog box, include the root name of the design that you wish to simulate, after the WVSIM keyword. For example, to simulate the binctr4 design, make this line in the dialog box read "WVSIM binctr4", then click on "OK".

To obtain a list of signal names:

1. Type the DISPLAY command at the SIM> prompt. For example, the command "DISPLAY *" lists all the signal names in the simulation file, and their current values.

If the number of signal names in the file is too large (and in a big design, there could be thousands), you can do one of two things:

1. Restrict the list of signals displayed, by the judicious use of signal-name fragments and wildcard characters. For example, the command "DISPLAY DATA_IN*" displays the names and current values of all signals in the design beginning with the characters "DATA_IN". The "*" wildcard character matches any sequence of characters. The "?" wildcard character matches any single character.

2. Re-direct the output of the DISPLAY command to a file. For example, the command "DISPLAY * >SIGNALS.TXT" writes the names and current values of all signals in the simulation file to a text file named SIGNAL.TXT.

   NOTE: When re-directing the output from a DISPLAY command to a file, you should follow the DISPLAY command with a FLUSH command, in order to write the file out immediately.

4.7. Signal Naming Conventions

During the synthesis and optimization processes, designs may be modified in such a way that signal names that appear on the original schematic no longer exist. This section discusses the signal naming conventions used by the *Warp3* tools, in order to make it easier to follow designs through simulation during debugging.

When debugging a design, you use ViewSim and ViewTrace to exercise the final representation of the design, and determine if it is performing as expected. If all is well, it is sufficient just to examine the external signals and verify that the design produces the correct outputs for a given set of inputs. If you find a design bug, however, it is often convenient to examine internal signals in order to locate the source of the error and correct the design.

This can present an unexpected problem, however. During optimization, designs may be modified in such a way that signal names that appeared on the original schematic no longer exist in the final layout, i.e., they may have been optimized out. Some signals may have been merged with others during optimization, and entirely new signals are likely to have been created.

There are two steps you can take to make it easier to simulate and debug designs:

1. learn what kinds of components are never optimized out of a design, as well as what attribute you can apply to certain components to specify that they must not be optimized out of the design; and

2. learn the signal naming conventions used by the *Warp3* tools, to make it easier to follow the progression of a signal name through various modifications as it progresses through the layout.

Keeping Components From Being Optimized Out of Designs

If it is important to you that a certain portion of your design not be optimized out by the *Warp3* tools, it will be helpful to remember that the following are never optimized out:

1. outputs from registers (flip-flops);
2. input buffer outputs from external pins;
3. components to which the DONT_TOUCH attribute has been assigned.

Note, however, that indiscriminate use of the DONT_TOUCH attribute could lead to unnecessary over-utilization of system resources, making large designs hard to fit on a chip.

Signal Naming Conventions

Although there are no hard-and-fast naming rules governing the synthesis and optimization process, knowledge of some general naming conventions may help in debugging designs:

1. Pins (external signals) in the schematic always get the same name in the final layout.
2. Labeled signals always get the same name in the final layout as they were given in the schematic, if they appear in the final layout. Thus, if you label a signal on a component that you know will not be optimized out (e.g., the output from a register, or a component to which the DONT_TOUCH attribute has been applied), you can be certain that a signal of the same name will appear in the final layout.
3. Internal signals attached to pins will usually get a name consisting of the pin's name, followed by a brief suffix. For example, the internal signal between a logic cell and an output pin named "RESULT" might be given the name

"RESULT_OUT". Similarly, the output from an input buffer attached to an input pin named "ENABLE" might be given the name "ENABLE_IN".

4. Another common suffix that you might see on internal signals is "_QTnnn" where 'nnn' is a number. Such signals are created due to logic factoring, where the original equation might be larger than a single logic cell can hold, causing intermediate signals to be created. Due to bubble pushing, logic optimization and packing, sometimes the original signal names might disappear and be replaced by a signal that is suffixed by the "_QTnnn" string.

5. When hierarchical designs are created by instantiating a component multiple times, signal names in the lower levels of the hierarchy are prefixed with an instance number assigned in Viewdraw. This instance number can be found in the upper-level schematic by selecting the instantiated component; its instance number is displayed in the lower left corner of the Viewdraw window. With this information, you can identify signal names that have been modified to guarantee uniqueness, associate the new name uniquely with one of the original instantiations, and reference the signal in the simulation.

   For example, suppose a signal named DATA_A, an output from a register, appears in a component instantiated twice onto a schematic. The two component instantiations might have instance numbers of VL1I1 and VL1I2, respectively. In that case, in the final layout after the synthesis and optimization process, you should be able to find two signals named VL1I1_DATA_A and VL1I2_DATA_A, corresponding to the two instantiations of the component in which the DATA_A signal appears.

6. Signals that start out as buses in a schematic (or as bit vectors in VHDL) are converted to discrete signals in physical layouts and in the simulation environment.

   The good news about this conversion process is that the *Warp3* tools maintain the association between the discrete signals and the original bus, so you can reference the bus directly in simulation command files.

   The bad news is that the conversion may cause inadvertent collisions to occur between bus signals and non-bus signals. For example, a bus named a[3:0] in a schematic or in VHDL will be converted to four discrete signals, named a_3, a_2, a_1, and a_0. If a non-bus signal should already exist with one of these names, a naming collision will occur, followed by unpredictable results.

   This problem can be easily avoided, of course, by simply not giving any other signals in your design the same name as the root name of a bus.

   With regard to referencing buses and individual bus signals within the simulation environment, the following list shows some typical bus/signal names that might appear in a schematic, along with the ways you might reference them when running a simulation:

In Schematic	In Simulation
DATA_OUT(0)	DATA_OUT_0_OUT (internal) DATA_OUT_0 (at the pin)
DATA_IN(0)	DATA_IN_0_IN (internal) DATA_IN_0 (at the pin)
DATA_IN[3:0]	DATA_IN_[3:0]_IN (internal) DATA_IN_[3:0] (at the pin)

Note that in the last case, a vector must be defined in the command file. This is done using the command

```
vector vector_name DATA_IN_[3:0]_IN
```

for referencing the internal signal.

4.8. I/O Components in pASIC Designs

pASIC designs require that some special consideration be shown in the use of certain I/O components, discussed here.

The IO library consists of markers, pads, directional buffers, and tristate buffers, as well as gnd and vcc.

Markers

Markers are required for all signals that will be placed on device input, output, or I/O pins. Use the IN markers for signals that are inputs. Use the TRI markers for bi-directional or three-state signals. Use the OUT marker for signals that are outputs that are not permanently enabled.

Triout/BUFOE Components

In addition to using one of the TRI markers for bi-directional or three-state signals, a TRIOUT or BUFOE component must be used for the enable and feedback signals.

Figure 4-2 shows examples of the use of markers and TRIOUT/ BUFOE components.

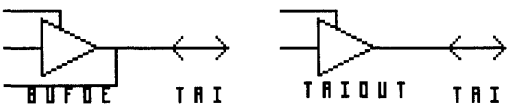


Figure 4-2. Use of markers with TRIOUT and BUFOE Components.

Pads

Pads have no meaning in PLD and CPLD designs, and so their use will be ignored in that context.

For pASIC380 FPGA designs, pads should be used for system clocks, system resets, and high-fanout nets.

A brief summary of the routing resources available in the Cypress pASIC380 family of FPGAs will help to explain when to use pads.

Three types of input and output structures are provided on pASIC380 FPGA's: dedicated inputs, bidirectional I/O cells, and clock/dedicated input cells. Each member of the family has six dedicated inputs, and two clocks that can also be configured as dedicated inputs. The dedicated inputs have about twice the driving capability of I/O cells, and are useful for distributing high-fanout signals. Each dedicated input can drive both the signal and the signal's complement. The clock/input cells can be configured as a clock, input to logic, or both. When used as an input, these cells also provide driving capability for the dedicated inputs.

To use the dedicated inputs, you must instantiate a high-drive pad (HDPAD) from the IO library or the PAINCELL (for both polarities) from the MCPARTS library. Likewise, to use a clock pad, instantiate a CKPAD or the PACKCELL (for clocking and input) from the MCPARTS library. To use an I/O cell, instantiate either the OUT marker or the TRI marker along with a BUFOE or TRIOUT component, as described above.

Dedicated inputs must use express wires or quad wires to route signals within the device. Table 4-1 summarizes the available express and quad wires within the device family. Express wires run the entire length or width of the logic cell array. Quad wires run the length of four logic cells.

Signals on high-drive pads must route on express wires or quad wires. Because the CY7C381A through CY7C384A each have four horizontal and vertical express wires per channel, a situation can exist in which more than four signals routed on express wires are required in a given channel. The signals cannot route in this case.

Table 4-1. Routing Features of Cypress pASIC380 FPGA's

		C381A	C382A	C383A	C384A	C385A	C386A
	Density	1K	1K	2K	2K	4K	4K
	XxY	8x12	8x12	12x16	12x16	16x24	16x24
Vertical Routing	#Exp	4	4	4	4	4	4
	#Quad	0	0	0	0	4	4
	#Clk	2	2	2	2	2	2
	#Seq	16	16	16	16	16	16
Horizontal Routing	#Exp	4	4	4	4	0	0
	#Quad	0	0	0	0	4	4
	#Seq	8	8	8	8	8	8
Additional Top/Bot.	#Exp	4	4	4	4	6	6
	#Quad	0	0	0	0	0	0
	#Seq	0	0	0	0	0	0

If you find that you have more than four channels on express wires that are required in a given channel, then you may need to remove a signal from a high-drive pad, and use a bi-directional pad in its place.

Do not use both the true and complement output of the high-drive pad. Choose only one polarity. If you require both the true and the complement of a signal on a high-drive pad, then bring in only one sense of the signal and use an inverter to obtain its complement. This will reduce the number of express wires needed.

Sometimes even a High Drive pad may not be sufficient to handle the loading on a given signal. For situations like these, HDnPAD components are provided in the library, where 'n' represents the number of High Drive pads that are combined (for example, HD2PAD). However, there are restrictions on their use. HDnPAD's should be pre-placed (using the pin_numbers attribute in VHDL or the '#' attribute in schematics) so that they fall on the same side of the chip in the final physical layout.

If you need to use the high-drive pad because of the I/O requirements of your design, then choose signals with a small fanout. These signals will route to fewer channels and reduce the probability that more than four signals will be required in a given channel.

Finally, if you must continue to use high-drive pads with high-fanout signals, then immediately buffer the signal by using a BUF component from the GATES library. The signal on the high-drive pad will still use an express wire, but because it will have a fanout of one (the buffer), the signal will route to only one vertical channel. The signal will be buffered in a logic cell and subsequently use segmented wires to route elsewhere.

In summary, signals on high-drive pads must use express or quad wires. High-drive pads are particularly useful for high-fanout nets. However, you should limit the use of high-drive pads and double-buffering (multi-buffering also requires express or quad wires) to less than the total number of express wires and quad wires per channel.

Chapter 5 Design Techniques: SYNTHESIS_OFF & BUF

5.1. Description

This chapter describes two design techniques that are useful in both CPLD and pASIC designs giving the user greater flexibility and control over fitting quality and fanout constraints.

Synthesis_off is also covered in the *Synthesis Reference* manual. There is also a discussion of buffering in the *SpDE* section of the *Warp3 User's Guide*.

This chapter provides examples for these design techniques.

5.2. Function and Use of the SYNTHESIS_OFF Attribute

The Synthesis_off attribute causes a signal to be made into a factoring point for logic equations, and keeps the signal from being substituted out during optimization.

The attribute is useful for the following reasons :

1. It gives the user control over which equations or sub-expressions need to be factored into a node.

2. It helps in cutting down on compile time for designs which have a lot of 'signal redirection' (signals getting inverted/ reassigned to other signals). This attribute provides the Logic optimizer a better control over the optimization process, by reducing the number of signals it needs to deal with.

3. Provide better results for designs where a signal with a large functionality is being used by many other signals. If let alone, the fitter would collapse all the internal signals (which is desirable in most cases) and may drive the design's resource requirements beyond the available limits.

By using the synthesis_off attribute, the user can assign the commonly used signal to a node and bring down the resource utilization.

5.2.1. Issues with CPLDs

A side effect of using the Synthesis_off attribute is that the design will now take an extra pass through the array to achieve the same functionality.

The user is recommended to use this attribute only on combinatorial signals. Registered signals are assigned to a node by natural factoring and the Synthesis_off attribute on these signals results in unnecessary factoring and resource utilization.

5.2.2. Issues with pASICs

For pASIC architectures, this attribute prevents logic factoring from further breaking the signals equations down so that they can be shared with other signals which may contain these sub-equations. The gates that form this signals' equation will not be shared with other signals. This attribute should be used on signals whose performance is critical and where sub-expression factoring is not desired. In some rare cases, factoring also causes more utilization of resources and this attribute will aid in those kind of situations also.

In general, for VHDL designs targeting pASIC architectures, it is a good practice to enable the logic factoring option in galaxy and place this attribute on certain critical signals only.

5.2.3. Syntax and Accessibility

```
attribute synthesis_off of SIGNAL_NAME:signal is true;
```

This attribute can be used for signals declared in a VHDL code and cannot be accessed with schematic capture.

5.3. Function and use of the BUF

Instantiating the BUF element in a VHDL code is a means to achieve Synthesis_off structurally. The BUF element is primarily used to assign a signal to a node.

For pASICs, the BUF element can be used to improve the signal drive capability. For further details on this, please refer to the SpDE Manual.

The BUF element can be used in schematics to assign different labels to the same net. This however has a penalty of an extra delay and extra resource being used.

BUF elements can also be used to create delay chains in a design.

5.3.1. Issues with CPLDs

In CPLD designs the BUF element is typically used, when the user wants to assign a signal with a large functionality to a node, pass the output from the node through the array, to be recombined with other equations. This reduces the design complexity significantly.

The fitter will optimize the BUF element if it is presented with a few BUF elements without any logic in between them.

5.3.2. Issues with pASICs

The BUF element is used in pASICs in exactly the same way as in the CPLDs. The fitter can be instructed to retain any BUF element/s (even those which have no logic in between them) by applying the Don't_touch attribute in conjunction with the BUF elements. This would force the fitter to use a FRAG for the BUF element. A similar feature is currently unavailable for the CPLD family.

5.3.3. Syntax and Accessibility

```
i1 : buf port map(INPUT_SIGNAL,OUTPUT_SIGNAL);
```

Both the above signals are to be of type BIT.

This element is accessible to the users through VHDL and schematic capture.

If a BUF element is being instantiated in a VHDL code, the user should include the line "USE WORK.RTLPKG.ALL;" in his code. This needs to be placed outside the entity and the architecture definitions. Without this line, the BUF element will not be recognized by the fitter.

It is also to be noted, that the overuse of synthesis_off attribute results in corresponding overuse of chip resources and in some cases an inability to fit a design on a chip. This is a very useful attribute, but needs correct handling.

5.4. SYNTHESIS_OFF

The VHDL code attached below describes a 4-bit adder and targets a CY7C371. The 4-bit adder is built using two 2-bit full adders and letting the intermediate carry "c2" ripple between the two groups. When the synthesis_off attribute is not used, the fitter tries to collapse the functionality of the signal "c2" with the signals "sum2", "sum3" and "co". This pushes the product term count on each of these 3 signals beyond 16, and the fitter is forced to sum-split the product terms to make the design fit. The design ends up using 13 macrocells, 137 product terms and takes 2 passes through the product term array.

When the Synthesis_off attribute is used on the signal "c2" as shown in the VHDL code attached, the signal "c2" is assigned to a node. The output from the node is fed back through the array and is used in the signals "sum2", "sum3" and "co". This still takes two passes through the array, but uses just 6 macrocells and 45 PTs.

Using the attribute synthesis_off can provide the user a lot of flexibility if used well.

```
-- 4-Bit Adder
USE WORK.MATHPKG.ALL;
USE WORK.ttlpkg.ALL;

ENTITY add4 IS
        PORT (CI : IN BIT;
              A3,A2,A1,A0: IN BIT;
              B3,B2,B1,B0: IN BIT;
              SUM3,SUM2,SUM1,SUM0 : OUT BIT;
              CO: OUT BIT);
END add4;

ARCHITECTURE archADD4 of ADD4 IS

signal  c2: bit;
attribute synthesis_off of c2 : signal is true;
```

```
BEGIN

sum0 <= a0 xor b0 xor CI;
sum1 <= a1 xor b1 xor ((a0 and b0) or (a0 and CI) or (b0
and CI));
c2   <= (a0 AND b0 AND b1)
        OR (a0 AND b0 AND a1)
        OR (ci AND b0 AND b1)
        OR (ci AND b0 AND a1)
        OR (ci AND a0 AND b1)
        OR (ci AND a0 AND a1)
        OR (a1 AND b1);

sum2 <= a2 xor b2 xor c2;
sum3 <= a3 xor b3 xor ((a2 and b2) or (a2 and c2) or (b2
and c2));
co   <= (a2 AND b2 AND b3)
        OR (a2 AND b2 AND a3)
        OR (c2 AND b2 AND b3)
        OR (c2 AND b2 AND a3)
        OR (c2 AND a2 AND b3)
        OR (c2 AND a2 AND a3)
        OR (a3 AND b3);

END archADD4;
```

5.5. BUF

As stated earlier, the BUF element can be instantiated both through VHDL and schematic capture.

5.5.1. Instantiating BUF through VHDL

5.5.1.1. CPLDs

The adder design presented earlier is modified here to illustrate the use of the BUF element. The performance of this design is the same as in the case when the attribute Synthesis_off is used.

```
-- 4-Bit Adder

USE WORK.MATHPKG.ALL;
USE WORK.RTLPKG.ALL;
USE WORK.ttlpkg.ALL;

ENTITY add4 IS
        PORT (CI : IN BIT;
              A3,A2,A1,A0: IN BIT;
              B3,B2,B1,B0: IN BIT;
              SUM3,SUM2,SUM1,SUM0 : OUT BIT;
              CO: OUT BIT);
END add4;

ARCHITECTURE archADD4 of ADD4 IS

signal  c2,c2temp: bit;

BEGIN

i1:     buf port map(c2temp,c2);

sum0 <= a0 xor b0 xor CI;
sum1 <= a1 xor b1 xor ((a0 and b0) or (a0 and CI) or (b0
and CI));
c2temp   <= (a0 AND b0 AND b1)
        OR (a0 AND b0 AND a1)
        OR (ci AND b0 AND b1)
```

```
        OR (ci AND b0 AND a1)
        OR (ci AND a0 AND b1)
        OR (ci AND a0 AND a1)
        OR (a1 AND b1);

sum2 <= a2 xor b2 xor c2;
sum3 <= a3 xor b3 xor ((a2 and b2) or (a2 and c2) or (b2
and c2));
co   <= (a2 AND b2 AND b3)
        OR (a2 AND b2 AND a3)
        OR (c2 AND b2 AND b3)
        OR (c2 AND b2 AND a3)
        OR (c2 AND a2 AND b3)
        OR (c2 AND a2 AND a3)
        OR (a3 AND b3);

END archADD4;
```

5.5.1.2. pASICs

This example illustrates the use of the BUF element to achieve some pASIC-specific applications like :

1. DOUBLE BUFFERING
2. BUFFERING

These help in reducing fan-out, increasing drive capability and increase the speed of operation. Please refer to the SpDE manual for further details on the applications for these features.

Double buffering:

```
use work.resolutionpkg.all;

...

architecture archdemo of demo is
...
signal fast: multi_buffer bit;
```

```
...
begin
        b1: buf port map(a, fast);
        b2: buf port map(a, fast);
```

This code fragment shows that two buf components are instantiated, each having the same input and output.

Another possible use of the buf component in FPGA designs follows. This technique is referred to as buffering, and is also described in the SpDE Manual. The following two code examples show that signal enable is has a fanout of sixteen. By instantiating a BUF component, the fanout can be reduced.

Enable has fanout of 16:

```
for i in 0 to 15 generate
      u16: mot port map(enable, a(i), b(i));
end generate;
```

Enable has fanout of 9:

```
buf1: buf port map(enable, enable2);

      g1: for i in 0 to 7 generate
                              u8: mot port map(enable,
a(i), b(i));
                  end generate;

      g2: for i in 8 to 15 generate
                              u8b: mot port map(enable2,
a(i), b(i));
                  end generate;
```

In these code fragments, "mot" is a user defined component. In the first case, signal enable has a fanout of 16. In the second case, enable has a fanout of nine (eight mot components and one buf component). The buf component is used to drive the other eight mot components.

The buf component will introduce an additional logic delay. The additional delay must be balanced with the fanout to produce the desired timing performance. If one instance can be singled out as requiring the enable signal before the other instances, then you could reduce the fanout for that particular instance by giving it a fanout of one, and using enable2 to drive the remaining instances.

5.6. Instantiating BUF Through Schematic Capture

The user can invoke the BUF element by selecting it from the Cypress GATES library. This selection menu is available from the VIEWlogic VIEWdraw window. The user has the same flexibility with the BUF element through VHDL and structural instantiation.

The functionality achieved in the VHDL code described in previous section can also be replicated in schematic capture.

The BUF component is used in FPGA designs primarily to reduce the fanout or delay of critical signals.

A signal may be driven by two logic cell outputs provided that the outputs are from A fragments with identical inputs, and provided that express wire and quad wire resources are available to drive a multiply driven signal. For a discussion of express and quad wire resource, refer to multi-buffering discussion in the SpDE Manual, Chapter 4.

The BUF component can provide an easy, but not the only, means to double buffer a signal. The following code fragment shows that the package called resolutionpkg is used, that two signals are declared us multi_buffer bits, the buffers have the same inputs, and the buffer outputs are the same node.

Double buffering:

```
use work.resolutionpkg.all;

...

architecture archdemo of demo is
...
signal fast: multi_buffer bit;
...
begin
      b1: buf port map(a, fast);
```

```
b2: buf port map(a, fast);
```

This code fragment shows that two buf components are instantiated, each having the same input and output.

Another possible use of the buf component in FPGA designs follows. This technique is referred to as buffering.

The following two code examples show that signal enable is has a fanout of sixteen. By instantiating a BUF component, the fanout can be reduced.

Enable has fanout of 16:

```
for i in 0 to 15 generate
  u16: mot port map(enable, a(i), b(i));
end generate;
```

Enable has fanout of 9:

```
buf1: buf port map(enable, enable2);

g1: for i in 0 to 7 generate
      u8: mot port map(enable, a(i), b(i));
    end generate;
g2: for i in 8 to 15 generate
      u8b: mot port map(enable2, a(i), b(i));
    end generate;
```

In these code fragments, "mot" is a user defined component. In the first case, signal enable has a fanout of 16. In the second case, enable has a fanout of nine (eight mot components and one buf component). The buf component is used to drive the other eight mot components.

The buf component will introduce an additional logic delay. The additional delay must be balanced with the fanout to produce the desired timing performance. If one instance can be singled out as requiring the enable signal before the other instances, then you could reduce the fanout for that particular instance by giving it a fanout of one, and using enable2 to drive the remaining instances.

Index

Warp3™

VHDL Development System

Nova User's Manual

Cypress Semiconductor
3901 North First Street
San Jose, CA 95134
(408) 943-2600
JANUARY 1995

Cypress Software License Agreement

1. LICENSE. Cypress Semiconductor Corporation ("Cypress") hereby grants you, as a Customer and Licensee, a single-user, non-exclusive license to use the enclosed Cypress software program ("Program") on a single CPU at any given point in time. Cypress authorizes you to make archival copies of the software for the sole purpose of backing up your software and protecting your investment from loss.

2. TERM AND TERMINATION. This agreement is effective from the date the diskettes are received until this agreement is terminated. The unauthorized reproduction or use of the Program and/or documentation will immediately terminate this Agreement without notice. Upon termination you are to destroy both the Program and the documentation.

3. COPYRIGHT AND PROPRIETARY RIGHTS. The Program and documentation are protected by both United States Copyright Law and International Treaty provisions. This means that you must treat the documentation and Program just like a book, with the exception of making archival copies for the sole purpose of protecting your investment from loss. The Program may be used by any number of people, and may be moved from one computer to another, so long as there is **No Possibility** of its being used by two people at the same time.

4. DISCLAIMER. THIS PROGRAM AND DOCUMENTATION ARE LICENSED "AS-IS," WITHOUT WARRANTY AS TO PERFORMANCE. CYPRESS EXPRESSLY DISCLAIMS ALL WARRANTIES, EXPRESSED OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE

IMPLIED WARRANTY OF MERCHANTABILITY OR FITNESS OF THIS PROGRAM FOR A PARTICULAR PURPOSE.

5. LIMITED WARRANTY. The diskette on which this Program is recorded is guaranteed for 90 days from date of purchase. If a defect occurs within 90 days, contact the representative at the place of purchase to arrange for a replacement.

6. LIMITATION OF REMEDIES AND LIABILITY. IN NO EVENT SHALL CYPRESS BE LIABLE FOR INCIDENTAL OR CONSEQUENTIAL DAMAGES RESULTING FROM PROGRAM USE, EVEN IF CYPRESS HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. CYPRESS'S EXCLUSIVE LIABILITY AND YOUR EXCLUSIVE REMEDY WILL BE IN THE REPLACEMENT OF ANY DEFECTIVE DISKETTE AS PROVIDED ABOVE. IN NO EVENT SHALL CYPRESS'S LIABILITY HEREUNDER EXCEED THE PURCHASE PRICE OF THE SOFTWARE.

7. ENTIRE AGREEMENT. This agreement constitutes the sole and complete Agreement between Cypress and the Customer for use of the Program and documentation. Changes to this Agreement may be made only by written mutual consent.

8. GOVERNING LAW. This Agreement shall be governed by the laws of the State of California. Should you have any question concerning this agreement, please contact:

   Cypress Semiconductor Corporation
   Attn: Legal Counsel
   3901 N. First Street
   San Jose, CA 95134-1599
   408-943-2600

Table of Contents

Chapter 1 Introduction

1.1. Introduction

Nova is Cypress Semiconductor Corporation's name for its jedec or jedec- functional simulator.

The Nova user interface gives you an easy way to:

- specify JEDEC files to simulate;
- read or write stimulus files;
- convert files from .JED to ViewSim format;
- edit input waveform traces;
- simulate the behavior of a design;
- alternate between various views (i.e., collections of signals), and specify signals to be included in each view;
- specify the length and resolution of a simulation;
- specify segments, where you can re-apply and edit initial conditions, in order to compare results of differing initial conditions side-by-side;
- and lots of other useful capabilities.

This manual tells you how to use Nova to simulate designs. It assumes that you are already familiar with common user interface operations for your computer, such as the use of scroll bars, menu buttons, opening and closing windows, etc.

Chapter 2 Using Nova

2.1. Starting Nova

On Sun workstations, typing "nova" on the command line brings up the Nova window. On PC's and compatibles, double-clicking on the Nova icon in the Cypress group window brings up the Nova window.

By default, Nova comes up ready to run on a color screen.

To start Nova on a monochrome Sun workstation, type "nova -m" on the command line.

To set Nova to come up in monochrome mode when running Windows on an IBM PC or compatible computer:

1. select the Nova icon from the Cypress group window;
2. select "Properties" from the File menu;
3. edit the "Command Line" entry to include the -m option;
4. click OK.

2.2. The Nova Window

The Nova window (Figure 2-1) consists of a menu bar with several items across the top; a column of buttons along the left side, listing pin and node numbers and signal names; an area for displaying traces, and scroll bars across the bottom and right sides.

Menu Bar

The menu items are File, Edit, Simulate, Views, and Options. Under each of these items are menus for selecting related actions. The menus are ordered so that the most common operation is at the top. The contents of each menu are described in greater detail later in this manual.

Only two menu items, Open and Exit, are enabled in the File menu when you first enter Nova. When you open a ".jed" file, the other menu items will be enabled.

Node Numbers, Signal Names

The left-hand side of the Nova window consists of a column of buttons, displaying pin and node numbers and their associated signal names. A "node" is an area of a circuit containing one or more points whose locations you may wish to trace. (For information about different values within a node, refer to Section 2.4.4., "Nodes".)

To change the width of the buttons where signal names are displayed, use the Signal Name Size item in the Options menu.

Trace Area

The trace area displays the values of the nodes/signals listed in the left-hand column.

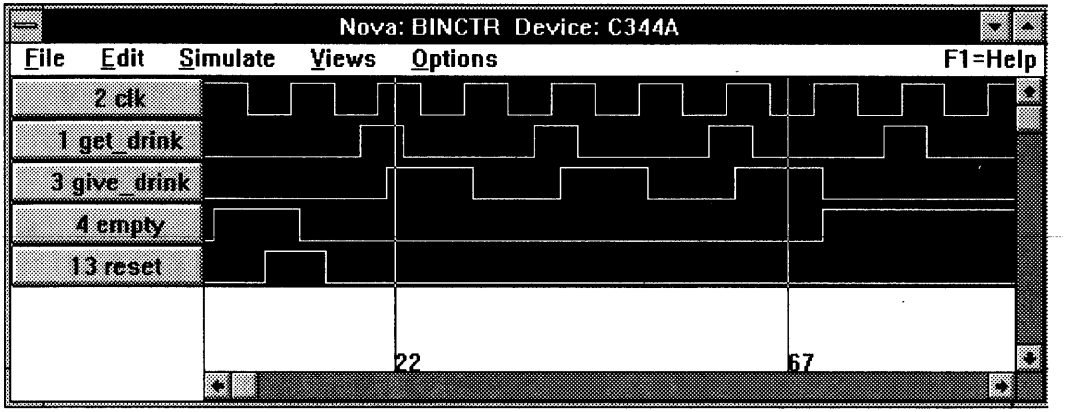


Figure 2-1. Main Nova Window.

You can display up to two measuring cursors, which allow you to see precisely the value(s) of several signals at a single time. To display the first cursor, click at the bottom of the trace window. To display a second cursor, click at the bottom of the trace window while pressing and holding the Shift key.

To change the position of either cursor, click and hold on the cursor at the bottom of the trace window, then drag the cursor to its new position. The cursor's horizontal position in simulation tics is displayed next to each cursor.

Note that a simulation tic does not represent any set amount of real-time delay. Instead, a simulation tic is simply a unit delay of simulation time.

2.3. The File Menu

The File Menu contains items related to opening JEDEC files for simulation, reading and writing stimulus files, and saving output files in various formats.

The File menu (Figure 2-2) in the Nova dialog box contains the following items:

- **Open...**
- **Write Sim (*.sim)**
- **Write Trace (*.psd)**
- **Read Stimulus File**
- **Write JEDEC Vectors**
- **Write JEDEC File (*.jed)**
- **Disassemble to ViewSim Format (*.vhd)**
- **Exit**
- **About...**

The operations of each of these menu items are discussed in greater detail on the next few pages.

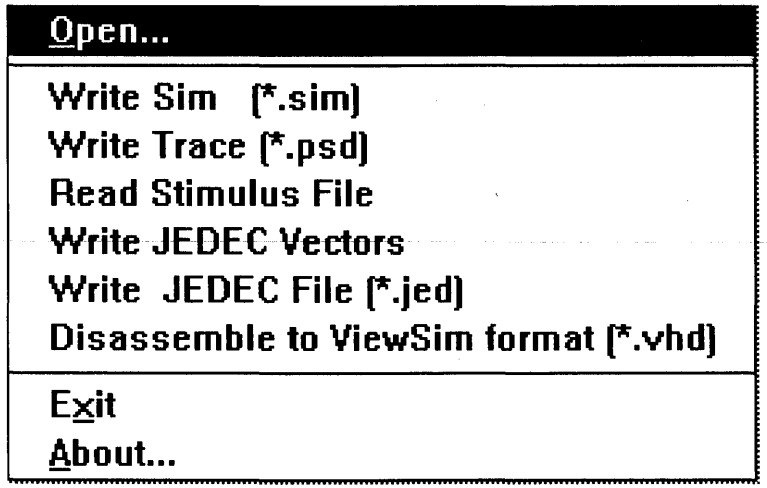


Figure 2-2. Nova File Menu.

2.3. The File Menu

2.3.1. Opening Files

The **Open...** item in the Files menu selects which .JED file to open, and tells Nova what device is targeted in simulation.

Selecting **Open...** brings up the Open Files dialog box (Figure 2-3). The "File Name" line specifies the names of files to view in the Files window, or to open. By default, this line reads "*.JED".

To open a file, you can select a file from the list shown in the Files window, or type the name of the file on the "File name" line. Selecting a .JED file and clicking on Open closes the dialog box and displays traces. (If a stimulus file of the form *filename*.sim or *filename*.stm exists, it is also read automatically.) Clicking on Cancel closes the dialog box without opening a file.

The Select Device dialog box (Figure 2-4) comes up when you click on Open in the Open Files dialog box, and the file to be opened is a .JED file not created by *Warp2*. The Select Device dialog box maps a JEDEC file to a device.

Selecting a device with the wrong number of fuses brings up a message box stating: "Wrong device type for this jedec - QF doesn't match." This indicates that the number of fuses in the selected device don't match the number in the JEDEC file.

NOTE

If Nova says that it can't find file DEVICES.DAT, check to make sure your CYPRESS_DIR environment variable is set correctly. Nova uses this file to find the proper pin names and numbers for each target device and package.

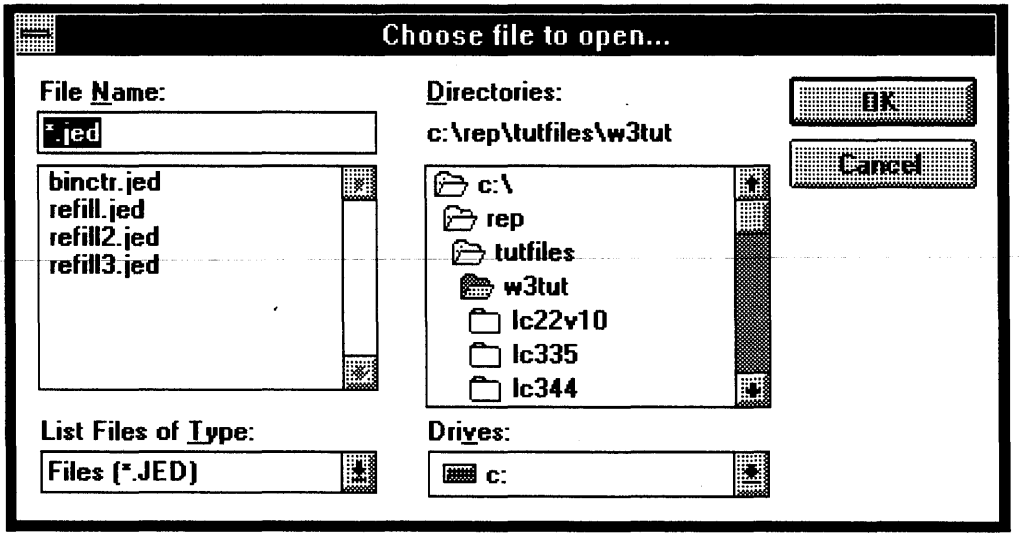


Figure 2-3. Open Files Dialog Box.

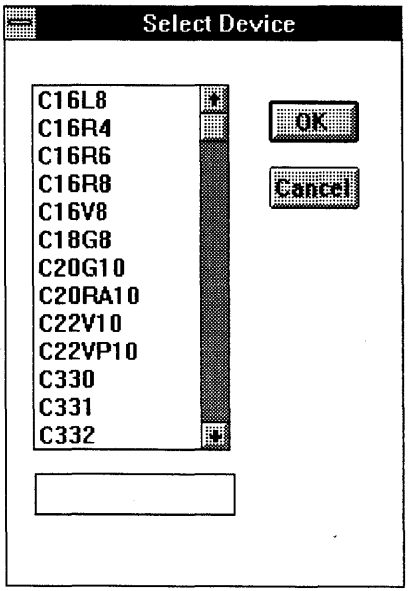


Figure 2-4. Select Device Dialog Box.

2.3. The File Menu

2.3.2. Reading and Writing Stimulus Files

Write Sim and **Write Trace** save simulation data. **Read Stimulus File** reads data stored by a previous Write Sim operation.

Write Sim saves the current simulation data to *filename*.sim, where *filename* is the prefix of the file you are simulating. If a ".sim" file already exists with this filename, the new simulation data overwrites the old. The .sim file (see Figure 2-5) can be re-read by **Read Stimulus File**.

Write Trace saves the trace information to *filename*.psd, where *filename* is the prefix of the file you are simulating. The .psd file (see Figure 2-6) provides a column-oriented, human-readable record of trace values during the simulation. Bus values are not written to the file.

Read Stimulus File reads simulation data from a .sim file. Because reading in the simulation file may change some of the settings the user has set for the current simulation, a message box is displayed, asking if the stimulus file should be read in. A "Yes" reply reads in the .sim file. A "No" reply returns you to the main Nova window. The *filename*.sim file is automatically read when the *filename*.jed file is opened.

```
1
clock_pin1
F83E0F83E0F83E0F83E0F83E0F83E0F83E0F83E0F83E0F83E0F83E0F83E0F83E0F83E
0000000000000000000000000000000000000000000000000000000000000000
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
9999
2
pin2
0000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
9999
3
nickel_pin3
00FF8000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
9999
```

Figure 2-5. Portion of .sim File.

```
 0: 1 0 0 0 L L L
 1: 1 0 0 0 L L L
 2: 1 0 0 0 L L L
 3: 1 0 0 0 L L L
 4: 1 0 0 0 L L L
 5: 0 0 0 0 L L L
 6: 0 0 0 0 L L L
 7: 0 0 0 0 L L L
 8: 0 1 0 0 L L L
 9: 0 1 0 0 L L L
10: 1 1 0 0 L L L
11: 1 1 0 0 L L L
12: 1 1 0 0 L L L
13: 1 1 0 0 L L L
14: 1 1 0 0 L L L
15: 0 1 0 0 L L L
16: 0 1 0 0 L L L
17: 0 0 1 0 L L L
18: 0 0 1 0 L L L
19: 0 0 1 0 L L L
20: 1 0 1 0 L L L
21: 1 0 1 0 L L L
22: 1 0 1 0 L L L
23: 1 0 1 0 L L L
24: 1 0 1 0 L L L
25: 0 0 1 0 L L L
```

Figure 2-6. Portion of .psd File.

2.3. The File Menu

2.3.3. Writing JEDEC Vectors

Write JEDEC Vectors appends vector information to the JEDEC file. The vectors can be used to test parts after they are programmed.

Write JEDEC Vectors appends vector information to the JEDEC file that you are simulating. If the JEDEC file already contains vector information, the new vector information overwrites the old.

2.3. The File Menu

2.3.4. Converting Between File Formats

The File menu includes items that allow you to convert vector information into different file formats, depending what you want to do with it.

Figure 2-7 shows the various file types that can be input to or output from *Warp*, Nova, and a device programmer.

Write JEDEC File (*.jed) writes out a JEDEC file from the data available to the simulator. The dialog box includes options to include an instruction in the JEDEC file to blow the security fuse when the device is programmed, and to write the JEDEC file using a compressed, "K-field" hexadecimal representation.

Disassemble to ViewSim format (*.vhd) writes out a Viewlogic vhdl file which is used to simulate the design in ViewSim in the subdirectory vhd.

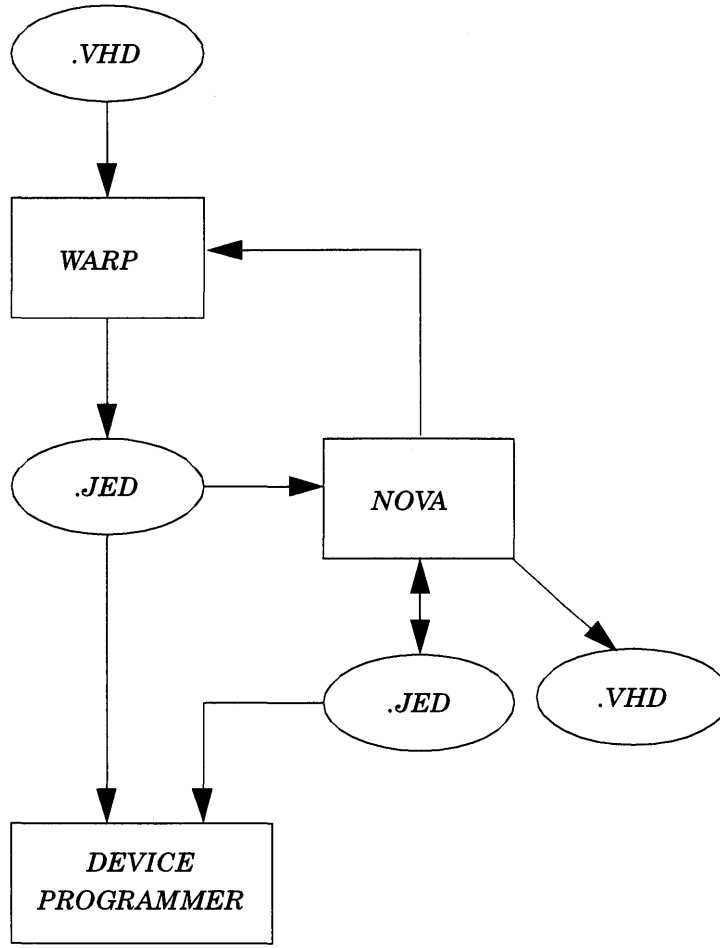


Figure 2-7. Possible Data Paths and File Formats.

2.3. The File Menu

2.3.5. About and Exit

The File menu's About item displays some basic information about the Nova simulator. The Exit item exits the simulator.

Besides displaying version information about the Nova simulator, the About dialog box also includes a Help button. Clicking on Help brings up help about Nova.

2.4. The Edit Menu

Use the items in the Edit Menu to modify trace information displayed on the screen. With the Edit menu, you can set the selected range of a trace, create and delete view nodes, create, delete and edit buses, and change the bus radix.

Items in the Edit Menu (Figure 2-8) include:

- **High, Low:** sets the selected trace or portion of a trace to 1 or 0, respectively.
- **Clock:** sets up repetitive pulses.
- **Pulse:** sets up a single pulse.
- **Node Defaults:** specifies the default source for the displayed value of a node.
- **Create View Node:** creates a new trace, selects the point within a node at which the displayed value is measured.
- **Delete View Node:** deletes traces from the simulation.
- **Create Bus:** groups traces for display as a single entity called a bus; used when it is more convenient to think of groups of signals as a single value. Bus values are only displayed when a measuring cursor is present.
- **Delete Bus:** un-defines a previously defined bus.
- **Edit Bus:** adds or removes signals from a bus.
- **Bus Radix:** specifies radix used to display a bus value.

These items are described in greater detail on the following pages.

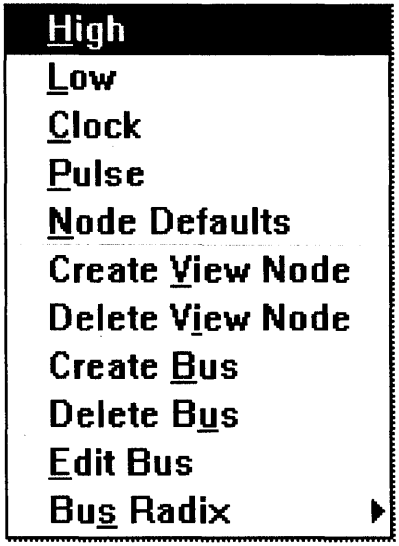


Figure 2-8. Nova Edit Menu.

2.4. The Edit Menu

2.4.1. Setting Signals High or Low

With the Nova user interface, you can easily set the value of all or a selected portion of an input signal high or low.

To set an entire input signal high or low:

- click on the button containing the name of the signal in the Nova window to select it. On color monitors, the button changes color and the trace turns blue when selected. On monochrome monitors, the button goes to inverse video and the trace changes to a dotted line when selected.
- select High or Low from the Edit menu as desired, or type '1' or '0'.

To set a portion of an input signal high or low (see Figure 2-9):

- de-select the signal;
- click and hold the mouse button on the trace at the point where you want the left edge of the selected area to be;
- drag the mouse to the point where you want the right edge of the selected area to be;
- then, select High or Low from the Edit menu, as appropriate, or type '1' or '0'.

Both the left and middle buttons of a 3-button mouse perform the same action when clicked to position an edge.

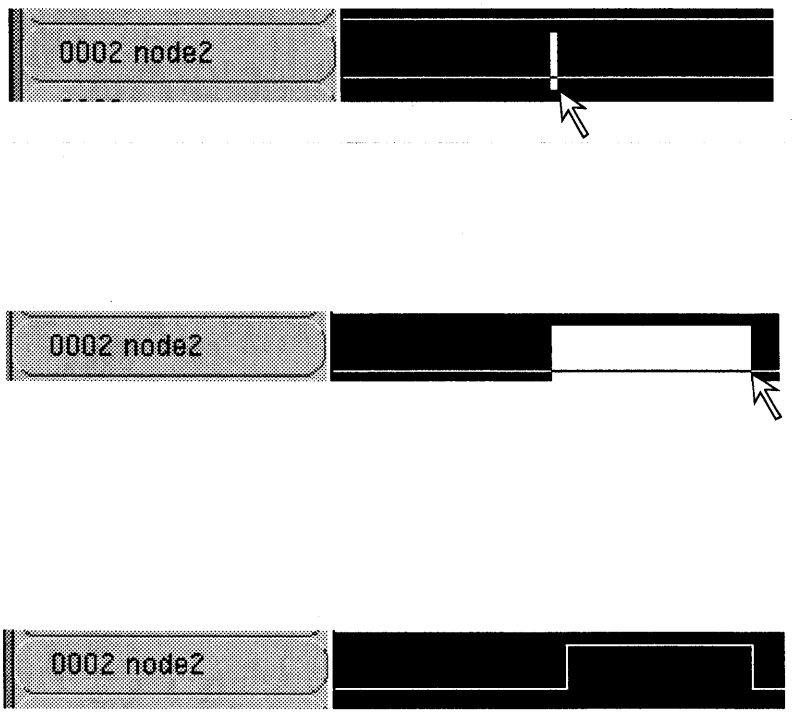


Figure 2-9. Setting a Portion of a Signal High or Low. Top: press and hold the mouse button at the left edge of the selected area. Middle: drag to the right edge of the selected area, and release the mouse button. Bottom: select High or Low from the Edit menu, or type '1' or '0' from the keyboard.

2.4. The Edit Menu

2.4.2. Setting Up Clock Signals (Repetitive Pulses)

The Clock item under the Edit menu lets you set up repetitive pulses on a selected signal or portion thereof.

To set up a repetitive pulse or clock signal, select a signal or a portion of a signal, then select the Clock item under the File menu. This brings up the Clock dialog box (Figure 2-10), which lets you fill in various parameters about the repetitive pulse you wish to set up:

- **Clock Period** specifies the period of repetition for the pulse in simulator "tics".
- **Clock Delay** specifies the number of simulator tics to wait (beginning with the left edge of the selected area) before starting the repetition. The default is 0 tics.
- **Clock High Time** specifies the amount of time that the selected signal should be set to '1' during each repetition. The default is 5.
- **Start High** and **Start Low** specify whether each repetition starts with the signal set to '0' or '1';
- **OK** sets up the repetitive pulse.
- **Cancel** closes the Clock dialog box without affecting the trace.

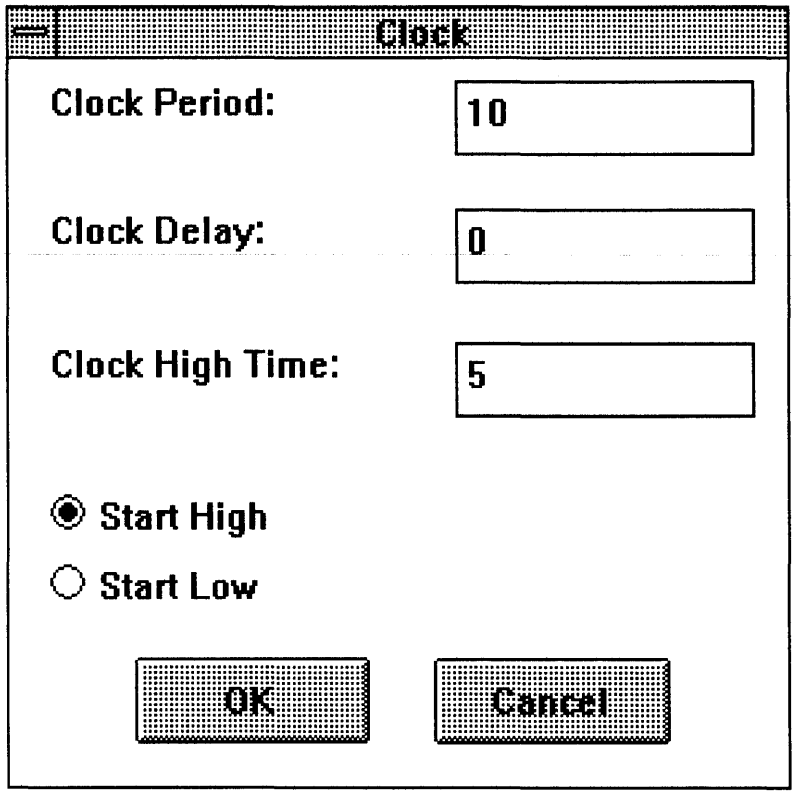


Figure 2-10. Clock Dialog Box.

2.4. The Edit Menu

2.4.3. Setting Up Non-Repetitive Pulses

The Pulse item under the Edit menu lets you set up single pulses on a selected signal.

To set up a single pulse on a signal, select a signal or a portion of a signal, then select the Pulse item under the File menu. This brings up the Pulse dialog box (Figure 2-11), which lets you fill in various parameters about the pulse you wish to set up:

- **Pulse Duration** specifies the length of the pulse, in simulator "tics".
- **Pulse Delay** specifies the number of simulator tics to wait (starting from the start of the simulation) before applying the pulse. The default is 0.
- **Start High** and **Start Low** specify whether the pulse sets the signal to '0' or '1';
- **OK** sets up the pulse.
- **Cancel** closes the Pulse dialog box without affecting the trace.

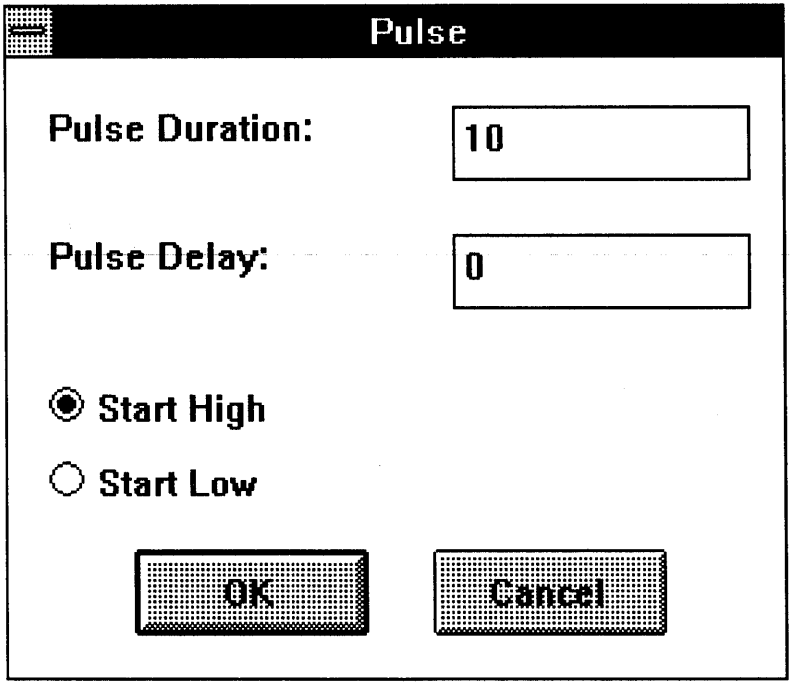


Figure 2-11. Pulse Dialog Box.

2.4. The Edit Menu

2.4.4. Nodes

A node is an area of a circuit containing one or more points at which you may wish to trace a signal. Nova lets you specify the exact point or points within a node at which to trace signal values. It also gives you facilities for setting the default value of a node and for forcing one or more positions in a node to known values.

To Nova, a node is:

1. any input to an array;
2. any output from an array;
3. any pin on the device;
4. any other electrical position that needs to be modeled, but doesn't meet the first three criteria.

Figure 2-12 diagrams a simple PLD to illustrate what constitutes a node. The pin to the left of the array meets criteria 1 and 3; the macrocell to the right of the array meets criterion 2; and the macrocell drawn below the array meets criterion 4.

For each node, you can:

- create a view node, i.e., specify one or more positions within a node from which to trace values;
- specify the means by which a node is assigned its value;
- force any position in a node to a known value. This is often useful for multi-segment simulations.

Each of these capabilities is discussed in greater detail on the following pages.

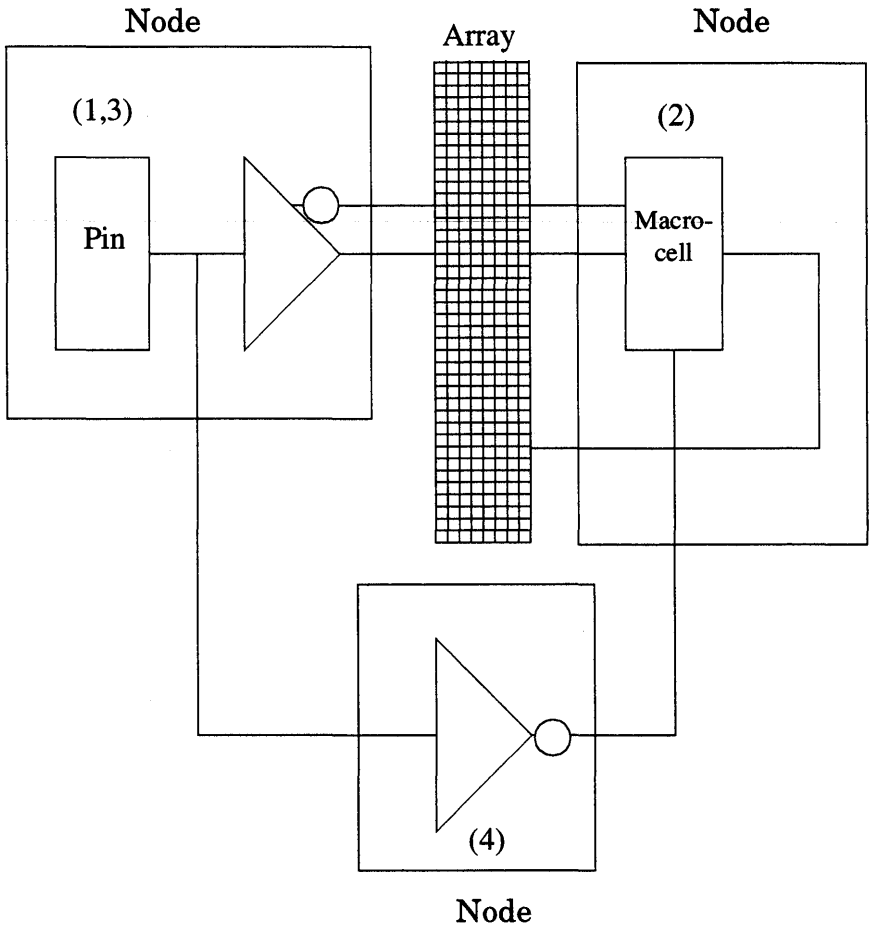


Figure 2-12. Logic Diagram of a Simple PLD, Showing Nodes.

2.4. The Edit Menu

2.4.4. Nodes

2.4.4.1. Selecting Node Points to View

Many nodes contain several points at which you can trace simulation values. **Create View Node** lets you select which of those points to view.

A view node allows you to see what is happening at various points inside a node. Selecting **Create View Node** brings up the Create Node View dialog box (Figure 2-13), which allows you to select points to view within a selected node. To bring up this dialog box, you must select a node with the current view set to FULL. (See Section 2.6., "The Views Menu", for information about changing views.)

The Create Node View dialog box displays the node name with the view node name to be created directly below it. Nova creates the view node name by taking the node number, followed by a '-' and an extension to represent the selected signal to be displayed.

The view node points that can be displayed depend upon the selected node. Examples of view node points that can be displayed include:

- Data from Array - This is the data at the output of an OR-XOR combination of gates. Extension is "ardat".
- Out value before OE - This is the data on the output pin if the output enable is asserted. This includes the output buffer inversion, if there is one. Extension is "b_oe".
- OE Value - This is the state of the output enable. If high, OE is asserted so the output is driven. Extension is "oe".
- Node Output - This is the data on the pin. This is the default view for output nodes. Extension is "out".

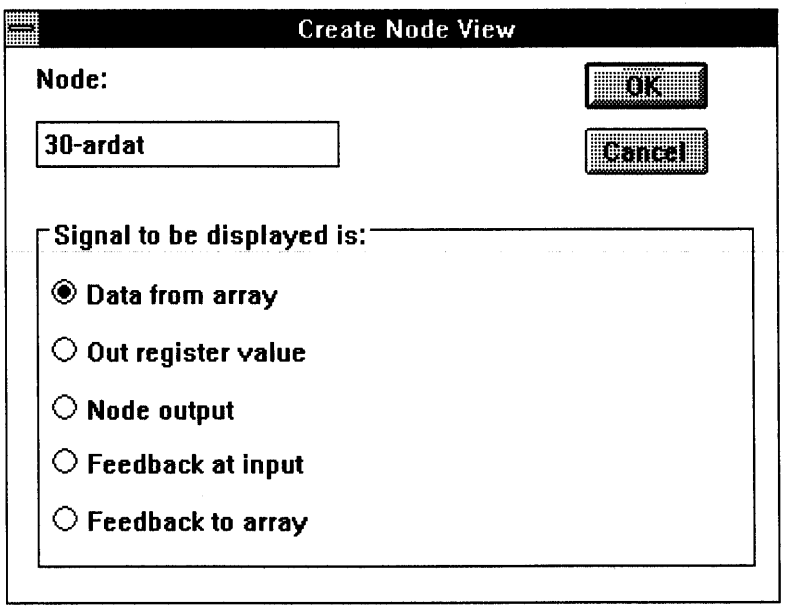


Figure 2-13. Create Node View Dialog Box.

- Feedback at input - This is the data at the D input of the input register, if there is one. If there is no input register, feedback at input and feedback to array are identical. Extension is "fbkin".

- Feedback to array - This is the data that is being fed to the array. It differs from feedback at input because it may be the other side of a register. Extension is "fbk_ar".

Selecting "OK" closes the Create Node View dialog box and creates a view node, displayed at the end of the node list. Selecting "Cancel" closes the Create Node View dialog box without creating the view node.

To delete a view node, select the view node to delete, then select **Delete View Node** from the Edit menu.

2.4.4.2. Setting Input Node Values

Node Defaults lets you specify the default source for the displayed value of a node.

Selecting **Node Defaults** brings up the Node Defaults dialog box (Figure 2-14).

You use the Change Default Input window of the Node Defaults dialog box to specify the high-impedance source for the value of an input node. The current setting is shown highlighted within this window.

There are four possible settings for each input. They are:

- High (1): tie the signal to Vcc.
- Low (0): tie the signal to Vss (ground).
- Use Simulation Record: use the value(s) in the simulation record (.sim file).
- Other Node Record: tie the signal to another node. Enter the node number on the line to the right of the Other Node Record button.

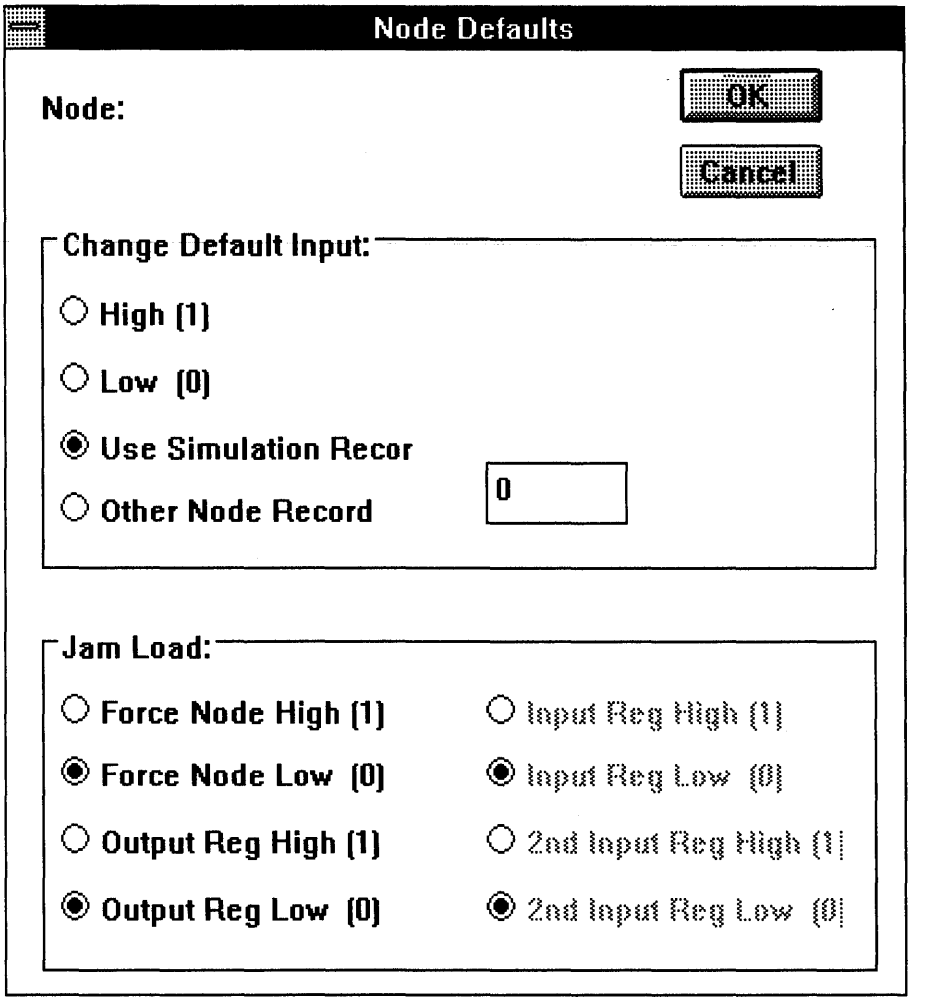


Figure 2-14. Node Defaults Dialog Box (1).

2.4.4.3. Forcing Output Node Values

Node Defaults lets you force the value of an output node at a specified point.

Selecting Node Defaults brings up the Node Defaults dialog box (Figure 2-15).

You use the Jam Load window of the Node Defaults dialog box to force an output node to a specified value. Values of these nodes rarely need to be touched for normal simulations. However, for multi-segment simulation (for long counters and other long-period design) or if there are problems in simulating the start-up of a circuit, the values may need to be changed. The current setting is shown highlighted.

Depending on the type of node, it may be possible to select from Force Node High(1), Force Node Low(0), Output Reg High(1), Output Reg Low (0), Input Reg High (1), Input Reg Low (0), 2nd Input Reg High (1), and 2nd Input Reg Low(0).

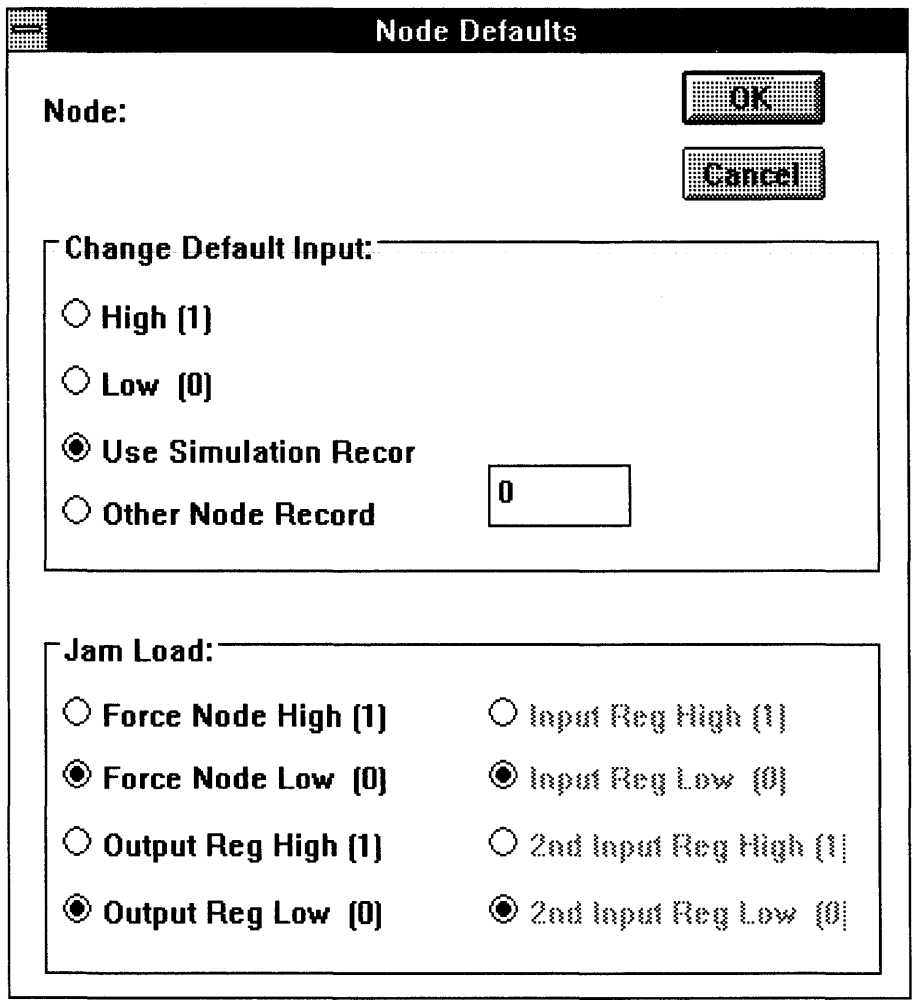


Figure 2-15. Node Defaults Dialog Box (2).

2.4. The Edit Menu

2.4.5. Working With Buses

Sometimes, it's more convenient to group several traces in a simulation and view them as a single trace. You can do this with the Create Bus, Delete Bus, and Edit Bus items in the Edit menu.

Selecting **Create Bus** brings up the Bus dialog box (Figure 2-16). This dialog box combines nodes into a user-named bus. The View list in the dialog box contains the names of all nodes in the current view. The Bus list holds the names of each node in the bus. A bus may be made up of any number of nodes.

Selecting OK closes the Bus dialog box and creates a bus with the specified bus name. Buses are placed at the top of the trace area. Selecting Cancel closes the Bus dialog box without changing the trace area. Bus values are not displayed unless a measuring cursor is present.

To add a node to the bus: select the node from the View list and select the "Add>>" button. Double-clicking on the node name also adds the selected node to the bus. The new node is added below the selected nodes of the bus.

NOTE

You can only add nodes in the current view to a bus in that view.

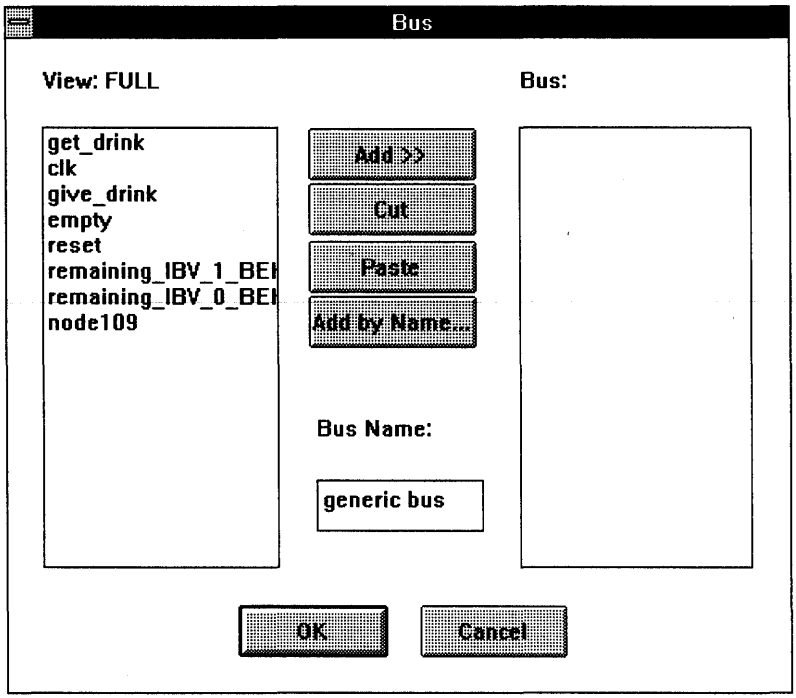


Figure 2-16. Bus Dialog Box.

Clicking on the Add-by-Name button brings up a dialog box that asks you to specify the name(s) of signals to add to the bus. The use of wild card characters is permitted. A "?" matches a single character; a "*" matches any string of characters. The construct *name*[*m*:*n*] denotes a range of signals, numbered from *m* through *n*, beginning with the characters *name*. For example, "input[0:3]" matches signals input0, input1, input2, and input3.

To remove a node from the bus: select the node to be removed and select the "Cut" button. Double-clicking on the node name in the Bus list also removes the node from the bus.

To change a node's position in the bus: Select the node, then click "Cut." Select another node, then click "Paste". The node that was previously cut will be inserted below the newly selected node.

To name the bus: click on the line below the words "Bus Name" and enter the name for the bus. If no name is provided, the bus is named "generic bus".

To delete a bus: Select a bus trace by clicking on the bus name button or the bus trace. After the bus is selected, selecting the **Delete Bus** item from the Edit menu brings up a dialog box with which you can remove the bus from the trace area.

Edit Bus brings up the same Bus dialog box used for creating the bus. The bus name line is filled in, and the nodes in the bus are displayed in the Bus list. Add and remove nodes from the bus in the same way as when you create a bus. You may also change the bus name when editing the bus.

After all changes have been completed, selecting OK closes the bus dialog box and applies the modifications to the selected bus. Selecting Cancel closes the dialog box without updating the bus.

Bus Radix brings up a submenu that allows you to choose how bus information is displayed. The three choices are binary, octal and hexadecimal. Hexadecimal is the default.

2.5. The Simulate Menu

The Simulate Menu has only one menu item: Execute.

Selecting **Execute** from the Simulate Menu (Figure 2-17) simulates the design's operation. The Nova screen is redrawn, and the resulting waveforms are displayed.

Figure 2-17. Simulate Menu.

2.6. The Views Menu

Items in the Views menu let you select the views (i.e., groupings of traces) that you see in the trace area.

The View menu (Figure 2-18) contains five items:

- **Edit Views**: lets you create and edit views.
- **Select View:** lets you select a view to display.
- **Delete View:** lets you remove one or more views from the list.
- **Zoom In (2X):** multiplies the displayed timescale resolution factor by two.
- **Zoom Out (1/2X):** divides the displayed timescale resolution factor by two.

Each of these items is discussed in greater detail in the following pages.

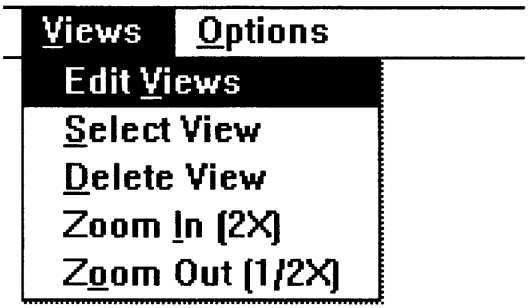


Figure 2-18. Views Menu.

2.6. The Views Menu

2.6.1. Editing Views

Edit Views lets you create new views, and add, remove, or exchange traces in existing views.

Three views are automatically created with each .JED file: full, pins-only, and pins & registers. The full view (default) lists all nodes in the design. This view cannot be edited. The pins-only view contains only nodes that are attached to pins. The pins & registers view contains all nodes attached to registers or pins.

Selecting **Edit Views** displays the Edit Views dialog box (Figure 2-19), used to edit the current view. The view list on the left displays the FULL view, which contains the default traces for all nodes. Use this list, along with appropriate buttons, to add or remove traces from the view list on the right.

To create a new view: click on New View. You will be prompted for a name, which is placed at the top of the right-hand view list.

To move between views: click on Next View or Previous View.

To add a trace to a view: select one or more traces from the left (Full) view window, then click Add>>. If a trace is also selected in the right window, the new traces are inserted after the selection; otherwise, the new traces are added to the end of the view.

To remove traces from a view: select the traces in the right window, then click on Cut.

To exchange (i.e., re-order) traces within a view: Select one or more traces from the right view window, then click Cut. Then, select another trace from the right view window and click Paste. The previously cut trace(s) are inserted after the selected trace.

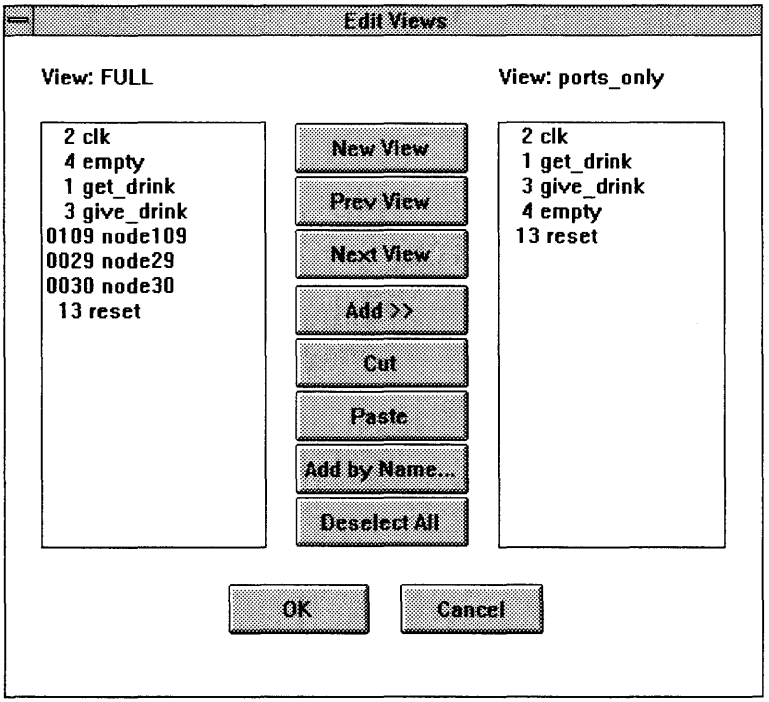


Figure 2-19. Edit Views Dialog Box.

Add-by-Name brings up a dialog box that asks you to specify the name(s) of traces to add. The use of wild card characters is permitted. A "?" matches a single character; a "*" matches any string of characters. The construct *name*[*m*:*n*] denotes a range of signals, numbered from *m* through *n*, beginning with the characters *name*. For example, "input[0:3]" matches signals input0, input1, input2, and input3. You can also use multiple expressions separated by spaces.

Deselect All unselects all selected traces in either window.

Selecting OK closes the Edit Views dialog box and updates the trace area to reflect changes made to the view. Selecting Cancel closes the Edit Views dialog box without making any changes to the view.

2.6. The Views Menu

2.6.2. Selecting and Deleting Views

Select View lets you change the active view. **Delete View** lets you remove a view from the list of available views.

Select View brings up the Select View dialog box (Figure 2-20). The View line gives the name of the current view. To change the current view, select the desired view from the list, then click OK or type a carriage return. Clicking Cancel closes the Select View dialog box without affecting the active view.

Delete View also brings up the Select View dialog box. Select the view to delete from the scrollable list. The FULL view may not be removed, and so it is not included in this list. Clicking OK or typing carriage return applies the change to the list of views. If you remove the current active view, the active view changes to FULL. There is no undo for Delete View, so be certain the view you are deleting is correct before clicking on OK or typing a carriage return. Cancel closes the Select View dialog box without deleting the selected view.

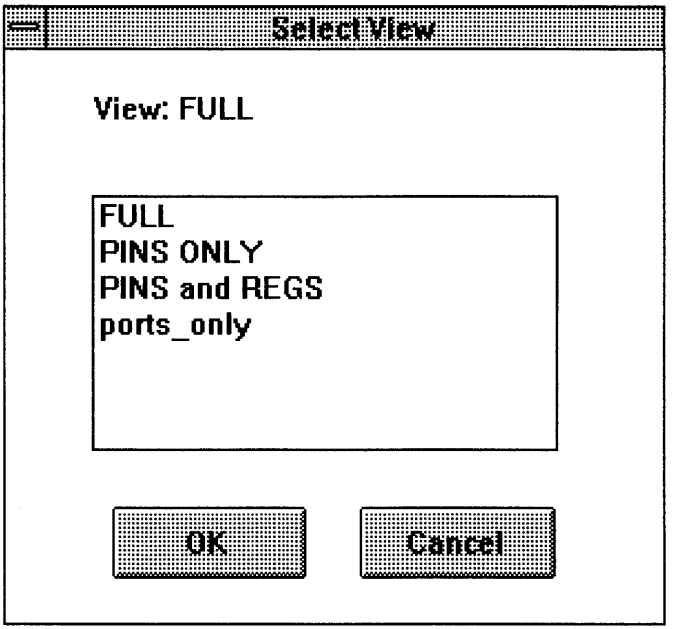


Figure 2-20. Select View Dialog Box.

2.6. The Views Menu

2.6.3. Zoom In, Zoom Out

Zoom In doubles the time scale resolution of the trace window. **Zoom Out** halves the time scale resolution of the trace window.

Zoom In doubles the time scale resolution of the trace window, i.e., it doubles the number of pixels in the X-axis used to display one tic of simulation time. The result is to "zoom in" the view of displayed traces.

Zoom Out does just the reverse.

The resolution setting must be 1 or greater. The default is 5. Attempting to set the time scale resolution lower than 1 has no effect.

2.7. The Options Menu

The Options Menu contains items that let you specify the simulation length, create or delete simulation segments, and specify the viewing resolution of the trace area.

The Options Menu (Figure 2-21) contains five items:

- **Simulation Length:** lets you set the length of the simulation.
- **Create Segment:** lets you create a segment, or "new-start-point", within the simulation.
- **Delete Segment:** lets you delete a previously created segment from the simulation.
- **Resolution:** lets you stretch and compress displayed traces.
- **Signal Name Size:** lets you specify the width in characters of Nova's signal name buttons.

Each of these items is described in greater detail in the following pages.

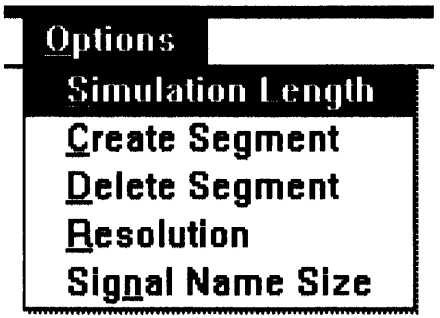


Figure 2-21. Options Menu.

2.7. The Options Menu

2.7.1. Simulation Length

Simulation Length lets you set the length of the simulation.

Selecting **Simulation Length** brings up the Simulation Length dialog box (Figure 2-22).

The minimum and default simulation length is 256 tics. The maximum simulation length is 9984. Clicking the up arrow adds 64 tics to the simulation length, to a maximum of 9984. Clicking the down arrow subtracts 64 tics from the simulation length, to a minimum of 256 tics.

You can also set the simulation length by typing a number on the line next to the up and down arrows. The number will be rounded downward to the nearest multiple of 64.

Clicking OK closes the Simulation Length dialog box and sets the simulation length, to be used on the next simulator run. Clicking Cancel closes the Simulation Length dialog box without affecting the simulation length.

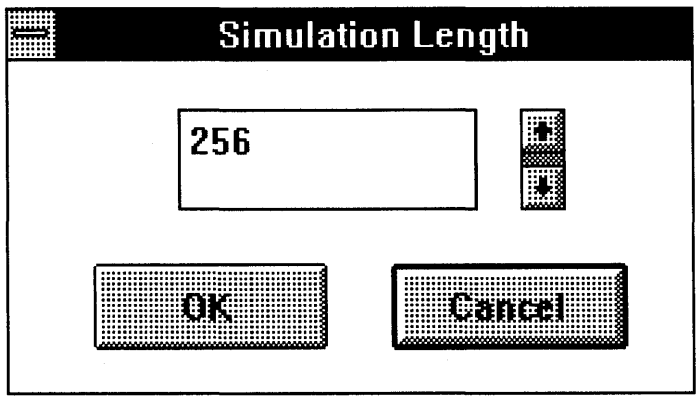


Figure 2-22. Simulation Length Dialog Box.

2.7. The Options Menu

2.7.2. Creating and Deleting Segments

Create Segment lets you create a segment, or "new start point," within the simulation. **Delete Segment** deletes a previously created start boundary.

A segment is a point in the simulation at which various nodes are reset to their "jam load" values (set through the Node Defaults dialog box).

To create a segment, position the leftmost measuring cursor where you would like the segment to start, then select **Create Segment** from the Options menu to bring up the Create Segment dialog box (Figure 2-23). The dialog box indicates the starting and ending boundaries of the segment. Selecting Yes closes the dialog box and creates the new simulation segment. Selecting No closes the dialog box without creating the segment. You may create up to 15 segments.

To delete a segment, position the leftmost measuring cursor within the segment to be deleted, then select **Delete Segment** to bring up the Delete Segment dialog box (Figure 2-24). The dialog box indicates the segment boundaries for the segment you wish to delete. Selecting Yes closes the dialog box and deletes the segment. Selecting No closes the dialog box without removing the segment.

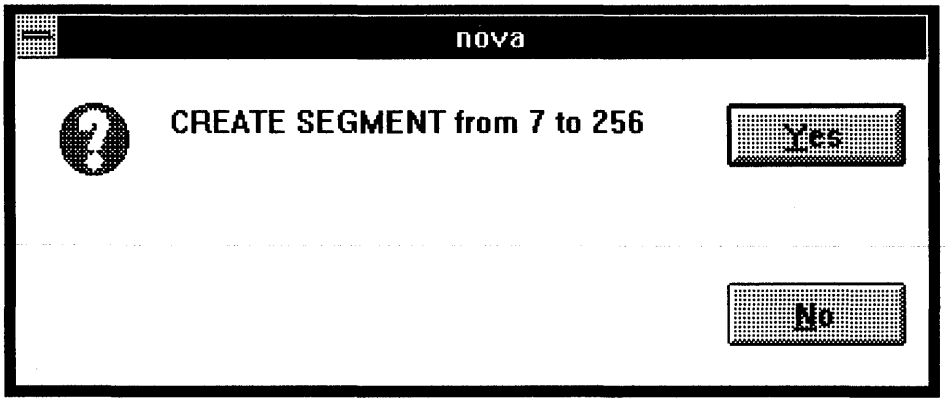


Figure 2-23. Create Segment Dialog Box.

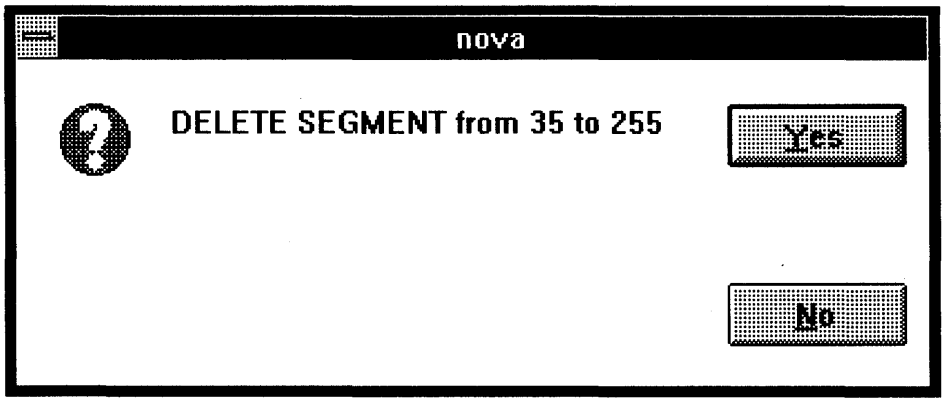


Figure 2-24. Delete Segment Dialog Box.

2.7. The Options Menu

2.7.3. Resolution

Resolution lets you stretch and compress displayed traces.

Selecting Resolution from the Options menu brings up the Resolution dialog box (Figure 2-25). This dialog box lets you set the number of screen pixels on the X-axis to be used per simulation tic. Varying this number effectively stretches or compresses the traces displayed on the screen.

The pixels-per-tic setting may be any number between 1 and 100. The default is 5. The larger the number, the more "stretched" the traces appear; the smaller the number, the more compressed the traces appear.

Selecting OK closes the Resolution dialog box and updates the trace display. Selecting Cancel closes the Resolution dialog box without updating the trace display.

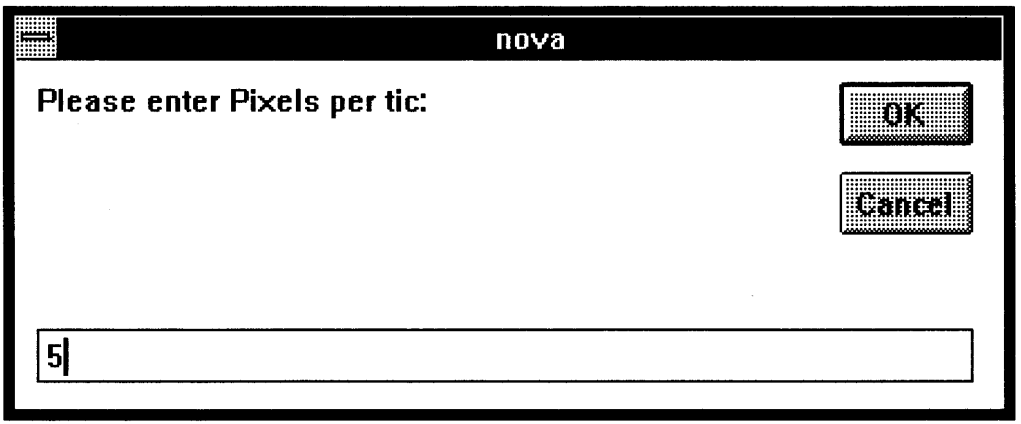


Figure 2-25. Resolution Dialog Box.

2.7. The Options Menu

2.7.4. Signal Name Size

Signal Name Size lets you specify the width in characters of Nova's signal name list buttons.

Selecting **Signal Name Size** from the Options menu brings up the Signal Name size dialog box (Figure 2-26). This dialog box lets you set the width in characters of the buttons in Nova's signal name list.

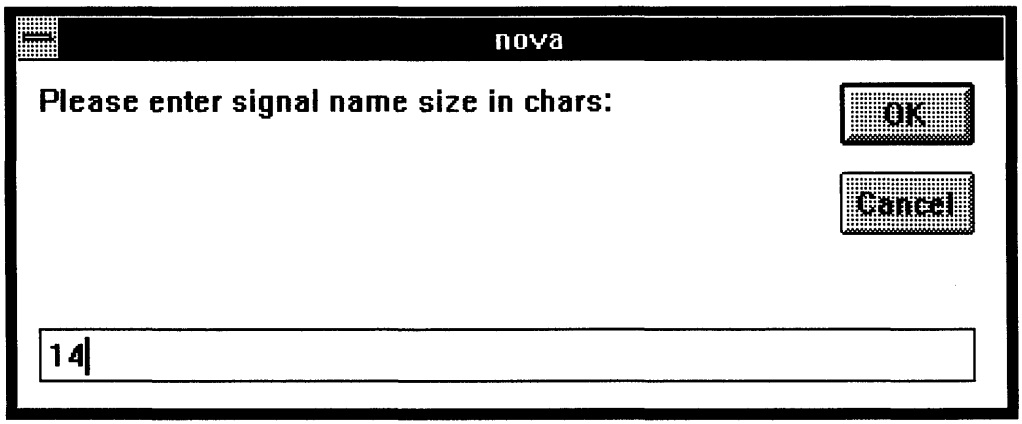


Figure 2-26. Signal Name Size Dialog Box.

Index

Warp3™

VHDL Development System

SpDE/Warp System User's Manual

Cypress Semiconductor
3901 North First Street
San Jose, CA 95134
(408)943-2600

JANUARY 1995

Cypress Software License Agreement

1. LICENSE. Cypress Semiconductor Corporation ("Cypress") hereby grants you, as a Customer and Licensee, a single-user, non-exclusive license to use the enclosed Cypress software program ("Program") on a single CPU at any given point in time. Cypress authorizes you to make archival copies of the software for the sole purpose of backing up your software and protecting your investment from loss.

2. TERM AND TERMINATION. This agreement is effective from the date the diskettes are received until this agreement is terminated. The unauthorized reproduction or use of the Program and/or documentation will immediately terminate this Agreement without notice. Upon termination you are to destroy both the Program and the documentation.

3. COPYRIGHT AND PROPRIETARY RIGHTS. The Program and documentation are protected by both United States Copyright Law and International Treaty provisions. This means that you must treat the documentation and Program just like a book, with the exception of making archival copies for the sole purpose of protecting your investment from loss. The Program may be used by any number of people, and may be moved from one computer to another, so long as there is **No Possibility** of its being used by two people at the same time.

4. DISCLAIMER. THIS PROGRAM AND DOCUMENTATION ARE LICENSED "AS-IS," WITHOUT WARRANTY AS TO PERFORMANCE. CYPRESS EXPRESSLY DISCLAIMS ALL WARRANTIES, EXPRESSED OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE IMPLIED WARRANTY OF MERCHANTABILITY OR

FITNESS OF THIS PROGRAM FOR A PARTICULAR PURPOSE.

5. LIMITED WARRANTY. The diskette on which this Program is recorded is guaranteed for 90 days from date of purchase. If a defect occurs within 90 days, contact the representative at the place of purchase to arrange for a replacement.

6. LIMITATION OF REMEDIES AND LIABILITY. IN NO EVENT SHALL CYPRESS BE LIABLE FOR INCIDENTAL OR CONSEQUENTIAL DAMAGES RESULTING FROM PROGRAM USE, EVEN IF CYPRESS HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. CYPRESS'S EXCLUSIVE LIABILITY AND YOUR EXCLUSIVE REMEDY WILL BE IN THE REPLACEMENT OF ANY DEFECTIVE DISKETTE AS PROVIDED ABOVE. IN NO EVENT SHALL CYPRESS'S LIABILITY HEREUNDER EXCEED THE PURCHASE PRICE OF THE SOFTWARE.

7. ENTIRE AGREEMENT. This agreement constitutes the sole and complete Agreement between Cypress and the Customer for use of the Program and documentation. Changes to this Agreement may be made only by written mutual consent.

8. GOVERNING LAW. This Agreement shall be governed by the laws of the State of California. Should you have any question concerning this agreement, please contact:

   Cypress Semiconductor Corporation
   Attn: Legal Counsel
   3901 N. First Street
   San Jose, CA 95134-1599

   408-943-2600

Table of Contents

Chapter 4 - Design Techniques

Appendix A - Error Messages

Chapter 1 Introduction

About This Chapter

This chapter introduces the purpose, elements, and design flow of the *Warp* system's SpDE tools.

1.1 Introduction

SpDE is the name of Cypress Semiconductor Corporation's pASIC processing toolkit.

This chapter tells you how to use the SpDE Toolkit to process pASIC designs. It assumes that you are already familiar with common user interface elements for your platform, such as scroll bars, menu buttons, windows, etc. The SpDE toolkit runs within a rich graphical environment that provides a consistent user interface and a unified pASIC processing environment.

The SpDE Toolkit gives you the following pASIC processing capabilities:

- Design verification
- Logic optimization
- Automatic placement and routing
- Physical viewing of pASIC layout
- Post-layout delay modeling and back-annotation
- Timing analysis
- Automatic test vector generation (ATVG)

The typical order of use of the SpDE tools in *Warp* is:

- import a .QDF file, using Import QDIF from the File menu;
- run all the SpDE tools in sequence, using Run All Tools from the Tools menu;
- export the .LOF file using Export/LOF from the File menu.

Design Verifier

The Design Verifier analyzes a user design for common design errors, architectural violations, and architectural warnings. In addition to catching errors prior to running the other SpDE tools, the Design Verifier alerts the user to such potential problems as excessive fanout.

Logic Optimizer

Logic optimization is the first step in automatic placement and routing. The Logic Optimizer utilizes two modules to efficiently map logic into the pASIC logic cell: the Packer and the Technology Mapper.

Starting with the user design, the Packer maps logic symbols (hard macros) into logic cells. As many as four macro symbols may be packed into a single logic cell. The Packer allows users to design with easy and convenient macros, while achieving high utilization of the pASIC resources.

The Technology Mapper provides automatic logic cell optimization. The Technology Mapper introduces and removes inversion bubbles in order to improve capacity and performance. In more general terms, the Logic Optimizer merges gates in order to achieve more efficient implementations.

Placer

Placement is the second step in automatic placement and routing (APR). Starting with the fully packed design, the Placer moves cells around the pASIC to minimize routing delays. This not only produces extremely fast pASICs, it allows completely automatic routing. The Placer offers fixed placement of I/O cells as well as registers. Fixed I/O cell placement lets you take circuit board considerations into account, while register placement offers precise control over internal routing delays.

Working closely with the Path Analyzer, the Placer offers timing-driven placement. Timing constraints are specified directly in the Path Analyzer; these are passed to the Placer to insure that critical timing goals are met automatically.

Router

The Router connects cells by routing nets within the pASIC matrix. The Router's optimization capabilities minimize routing delays and routing resources required.

Physical Viewer

The Physical Viewer displays the physical layout of a pASIC part. The viewer allows the user to inspect the results of APR for placement efficiency and routing congestion. Long signal paths can easily be identified.

Delay Modeler

The Delay Modeler performs precise post-layout delay calculations using state-of-the-art circuit analysis techniques. Processing the complete results of APR, the Delay Modeler analyzes packing, placement, and routing to determine intrinsic delays and routing delays for the entire design. These precisely calculated delays are written directly into the logic simulator netlist for timing-accurate results.

Path Analyzer

The Path Analyzer performs path analysis of the circuit delays from the Delay Modeler. The Path Analyzer offers automatic analysis of the complete design, as well as interactive analysis of user-specified portions of the design.

Automatic Test Vector Generator

The Automatic Test Vector Generator (ATVG) provides

convenient test coverage of programmed pASIC parts. Using the internal scan path circuitry built into every pASIC, the Automatic Test Vector Generator tests the programmed device. The user can specify test patterns to be included. ATVG provides both ideal and actual coverage statistics, allowing the design to be tuned for optimal testability.

Starting SpDE

To start SpDE, double-click on the "Place&Rte" icon in the Workview PLUS cockpit (on IBM PC's and compatibles) or the Powerview cockpit (on Sun workstations).

1.2 The File Menu

The File menu (Figure 1-1) includes commands relating to: creating, opening, and saving chip files: importing/exporting QDIF, EDIF, and LOF files; printing the physical view of the layout; and exiting SpDE.

SpDE operates primarily on chip files (file extension .CHP) and QDIF files (file extension .QDF). Chip files have a compact and efficient binary file format, while QDIF files have an open ASCII file format. Chip files are used within SpDE for reasons of efficiency, while QDIF files are used to import the output from *Warp* synthesis operations.

New clears any current design and initializes SpDE to operate on a new design.

Open loads an existing chip file (file extension *.CHP*). Selecting Open brings up a dialog box, from which you can select a chip file. Doing so re-initializes SpDE and loads the specified chip file.

Save saves the current design as a chip file.

Save As saves the current design as a chip file with a user-specified name. Selecting Save As brings up a dialog box, which allows you to specify the name for the chip file. This menu item gives you the capability to save backup or history copies of a design for later reference.

Import loads files saved in QDIF or EDIF format. Export LOF generates a file for use in programming devices. The Import and Export menu items are discussed in greater detail in Section 1.2.1, "Importing, Exporting".

Print Setup invokes the printer setup dialog box for the default printer driver. Print prints the current physical view using the parameters set in Print Setup.

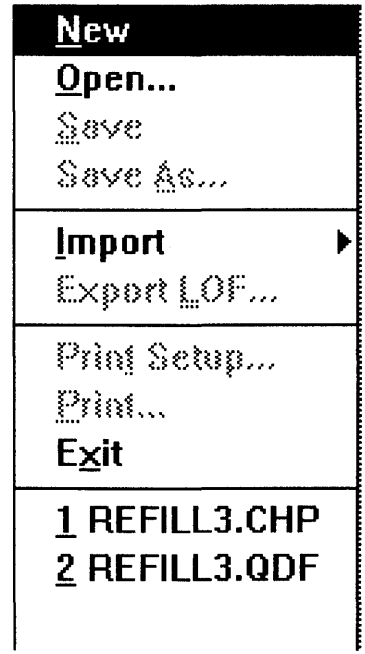


Figure 1-1. File Menu

Exit terminates SpDE, prompting you to save the design if it has been modified since the last save. On PC's and compatibles, double-clicking the Control-Menu box (in the upper-left corner of the SpDE window) is a shortcut for Exit.

Below the Exit menu item, the last five accessed files are listed. This provides a shortcut for the Open and Import commands. Clicking on a chip file from this list opens the file; clicking on another file type from this list imports the file.

1.2 The File Menu

1.2.1. Importing, Exporting

Import lets you import files written in QDIF or EDIF formats. Export LOF writes a file for use in programming devices.

The ***Import*** menu item features a pop-up sub-menu with QDIF and EDIF entries. Position the cursor on the Import menu item, click and hold the button, drag the cursor to select one of QDIF or EDIF. This bring up a dialog box, from which you can select a design. Doing so re-initializes SpDE and loads the specified file.

As the file is loaded, the Design Verifier analyzes it for common design errors, architectural violations, and architectural warnings. A dialog box lists any problems found.

Problems fall into one of four categories: Notice, Warning, Error, or Fatal Error:

- Notices and Warnings serve as alerts; they may or may not require action (e.g., excessive fanout).
- Errors prevent devices from being programmed, but do not prevent the toolkit from running (e.g., floating inputs).
- Fatal Errors prevent the toolkit from being run (e.g., net with multiple drivers).

The Design Verifier removes unused gates from the design; any gate that cannot affect an output is removed. Unused counter and register bits, for example, are automatically removed. "Stripped" gates are flagged with a Notice to ensure that their removal is sound.

Export LOF creates a Link Object Format file (file extension .LOF) with the specified name. This file contains all the data required to program and test pASIC devices. The file may be compressed at the user's option. Use this file to program the pASIC on a device programmer.

1.3 The View Menu

The View menu (Figure 1-2) includes commands for controlling the physical view and highlighting nets.

Zoom In "magnifies" the Physical View. The command supports two modes: "click" and "drag". To use click mode, select Zoom In, then click the mouse button once. This increases the scale factor by 1.25, and centers the view on the position of the cursor.

Drag mode allows an arbitrary rectangle to specify the desired view. To use drag mode, select Zoom In, and position the cursor on one corner of this rectangle. Then, click and hold the mouse button, drag the rectangle as desired, and release the mouse button. The view adjusts to fit the specified rectangle as closely as possible. The shortcut key for Zoom In is Ctrl-Z.

Zoom Out "de-magnifies" the view. The shortcut key for Zoom Out is Ctrl-X.

Full Fit modifies the scale factor to fit a view of the entire pASIC chip on the screen. The shortcut key for Full Fit is Ctrl-F.

Normal Fit sets the scale factor to its initial value, and centers the view on the selected position. The shortcut key for Normal Fit is Ctrl-N.

Preferences sets physical view options. These are discussed in greater detail in Section 1.3.1, "Preferences".

Hilight Net allows you to select nets to be highlighted in the physical view. This menu item is discussed in greater detail in Section 1.3.2, "Hilight Net".

Redraw erases and redraws the physical view.

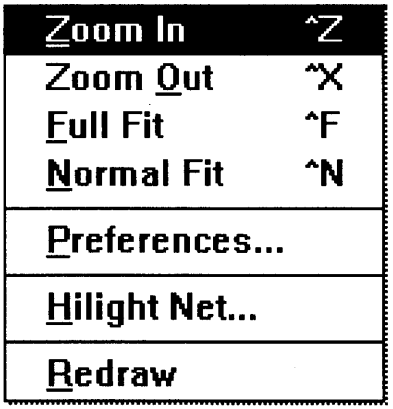


Figure 1-2. View Menu

1.3 The View Menu

1.3.1. Preferences

Preferences sets physical view options.

Selecting Preferences brings up the Preferences dialog box (Figure 1-3).

The Texting group of check boxes includes items relating to text in the physical view. Table 1-1 lists the check boxes and examples of the items they control. Text items can be turned off to increase redraw speed or simplify the physical view.

Table 1-1. Texting Options in the Physical View

Item	Example
Logic Cell Locations	A1, A2, B1, C1, H12
I/O Cell Numbers	3, 12, 24, 42
Flip-Flop Net Names	specified in schematic
I/O Cell Net Names	specified in schematic
Logic Cell Net Names	specified in schematic

The Flip-Flop Net Names option only includes the net names of logic cell flip-flop outputs. The Logic Cell Net Names option includes the net names of all logic cell inputs and outputs.

The Drawing Style group of radio buttons controls the use of color in classifying interconnect. In Black/White mode, all wires are shown in black. In Color mode, short wires are shown in blue and black, express wires are shown in green, I/O wires are shown in red, and clock traces are shown in mauve.

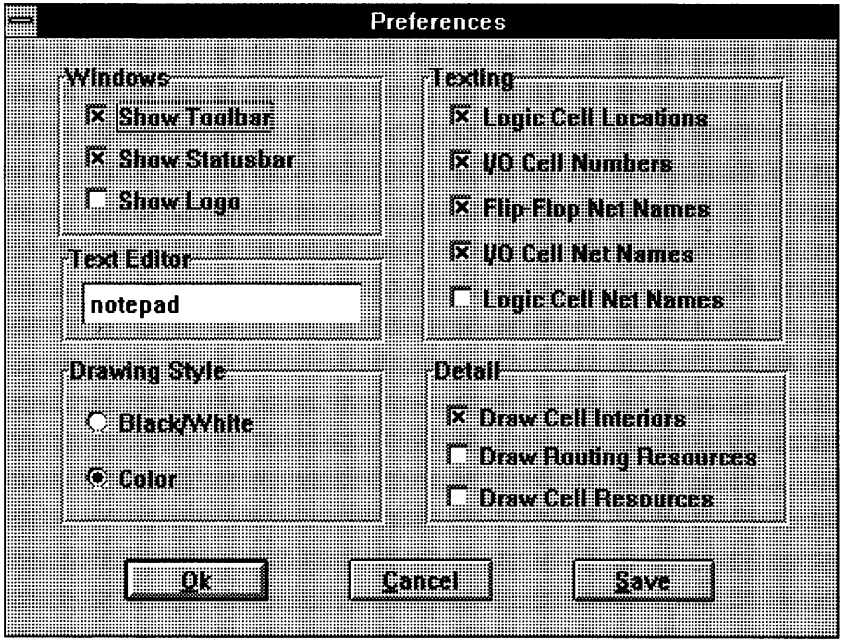


Figure 1-3. Preferences Dialog Box

The Detail group includes the Draw Cell Interiors check box. Deselecting this check box replaces the detailed logic cells with simple boxes, thereby increasing redraw speed.

Clicking OK accepts all preference settings for tools that are subsequently executed. Clicking the Save button does the same, and also records the preference settings. These settings are then used the next time SpDE is invoked.

1.3 The View Menu

1.3.2. Hilight Net

Hilight Net allows you to specify nets for SpDE to highlight in the physical view.

Selecting Hilight Net brings up the Highlight Net dialog box (Figure 1-4), and redraws the Physical View in light gray, which allows the highlighted nets to stand out. The net list box on the left of the dialog box contains the names of nets not currently highlighted, while the net list box on the right contains the names of nets that are currently highlighted.

To highlight nets, select one or more nets from the box on the left, or specify a net name in the "Wildcard Selection" field on the left side of the dialog box, and click on the right arrow button.

To remove nets from the highlight list, select one or more nets from the box on the right, or specify a net name in the "Wildcard Selection" field on the right side of the dialog box, and click on the left arrow button.

When using the "Wildcard Selection" fields, the wildcard characters "*" and "?" are accepted. The "*" character matches 0 or more occurrences of any character. The "?" character matches a single occurrence of a any character.

Double-clicking on a net name in either list moves it to the other list.

All nets can be removed from the highlight list by clicking on the ALL button.

Once in highlight net mode, the highlight status may be toggled by clicking directly on the desired nets in the physical view.

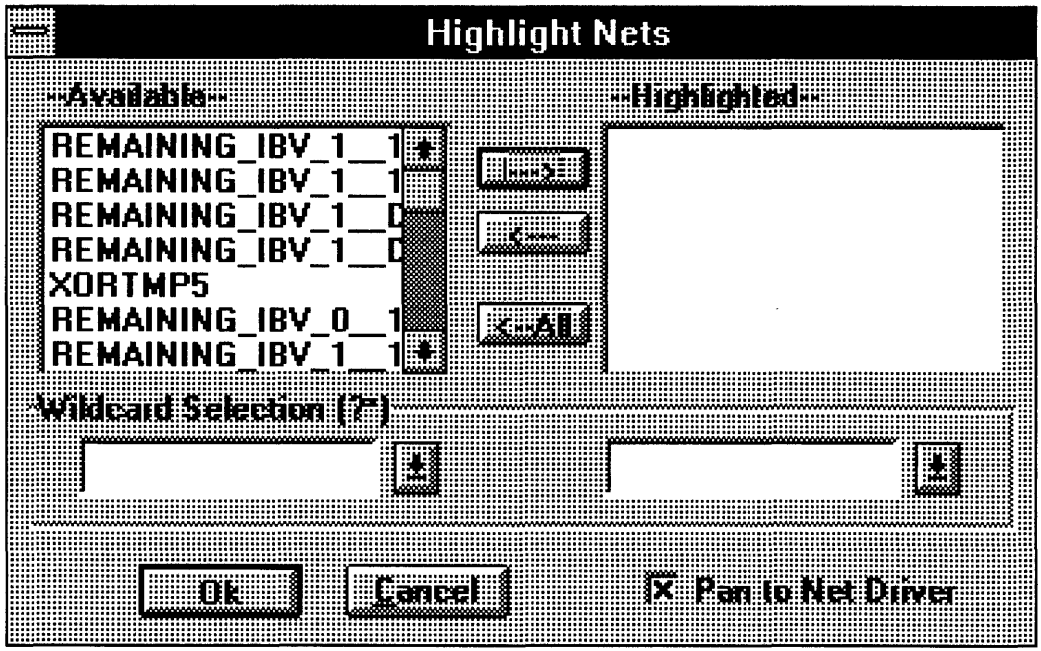


Figure 1-4. Highlight Net Dialog Box

To exit highlight net mode, click Cancel. The Physical View will be redrawn in the normal mode.

1.4 The Tools Menu

The Tools menu is used to set up and run the optimizing, placing, routing, sequencing, delay modeling, back annotation, and path analysis tools.

The Tools menu (Figure 1-5) contains three items: Run Tools, Path Analyzer, and Options.

Figure 1-5. Tools Menu.

Run Tools opens the ***Select Tools to Run*** window (Figure 1-6). This window is used to select which tools are to be run on the design. Two types of logic optimization can be run. This selection is made in the ***SpDE Tools Option*** window (Figure 1-7). Disabled tools are grayed out. Tools that have already been run are un-checked. Detailed descriptions of the Logic Optimizer, Placer, Router, Delay Modeling, Back Annotation, and ATVG tools can be found in Sections 2 and 3 of this manual. The Sequencer tool is only used to create data needed to program devices. It has no options, and so is not documented.

Path Analyzer can be used to determine operating frequency, setup and hold times, and clock skew. This menu item is discussed in greater detail in Section 3.2., "Path Analyzer".

Options lets you select General or Simulator options. General options are the options for all tools. Simulator options affect only the Back-Annotation tool. The General Tools Options and

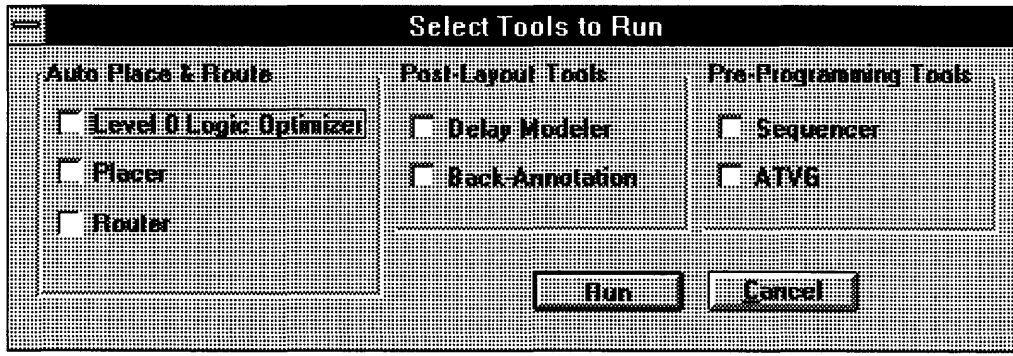


Figure 1-6. Select Tools to Run Dialog Box.

Simulator Options windows are shown in Figures 1-7 and 1-8, respectively.

The ***SpDE Tools Options*** window is used to configure each of the SpDE tools. A detailed description of each tool's options can be found in the section of this manual pertaining to that tool.

The ***Simulator Options*** window is used to select the simulator output that is desired. The ***Back Annotation*** tool uses this information to produce the proper timing netlist. When the ***Back Annotation*** tool is run, the file(s) appropriate to the selected simulation option are created. See the section on the Delay Modeler and Back Annotation for details.

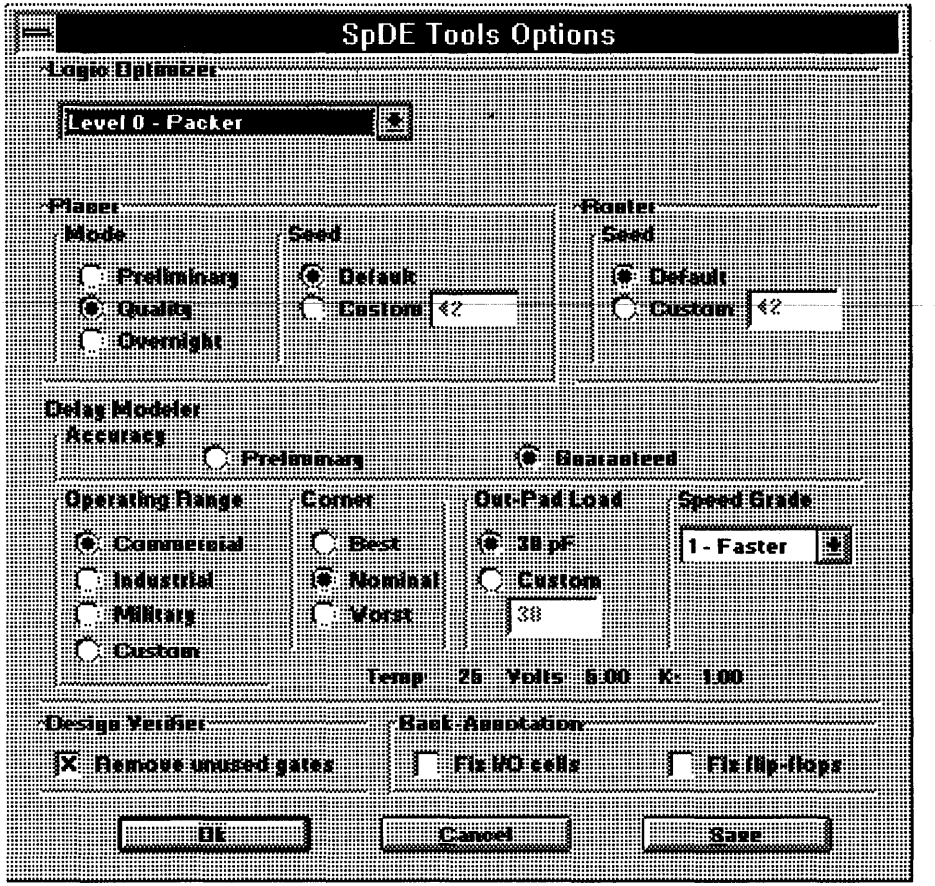


Figure 1-7. General Tools Options Window.

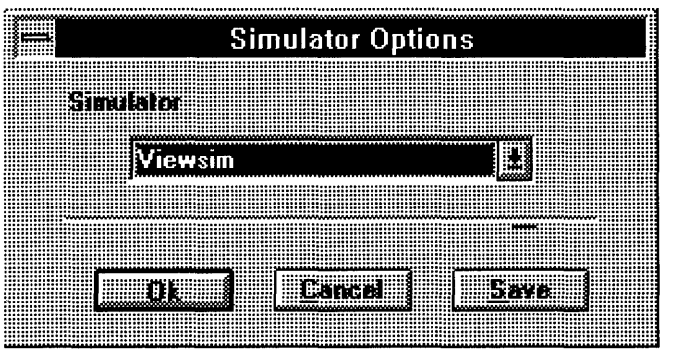


Figure 1-8. Simulator Options Window.

1.5 The Info Menu

The Info menu (Figure 1-7) includes items that provide statistics and other information about a design.

Cell Utilization reports on the number of cells of different types used in the design (Figure 1-8). It reports on the number of logic, input-only, clock-only, and bidirectional cells. Also included is a count of "partially-free" cells; these are logic cells with a free AND fragment. These cells can accept any macro that can be implemented in a single AND fragment. This information is provided to allow fine-tuning of high-utilization designs.

ATVG Coverage reports on the design's test coverage (Figure 1-9). These statistics may be used as guidelines to improve the design in order to increase test coverage. Of particular interest is the Fault Grading statistic, which indicates the quality of test coverage produced by ATVG.

Tool Versions is provided for diagnostic purposes. Tool Versions lists the tools that have been run on the current design, including the version number of each tool listed (Figure 1-10).

Report File displays the Cypress report file produced by the *Warp* compiler and appended to by the SpDE tools.

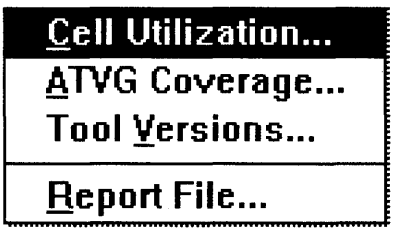


Figure 1-7. Info Menu

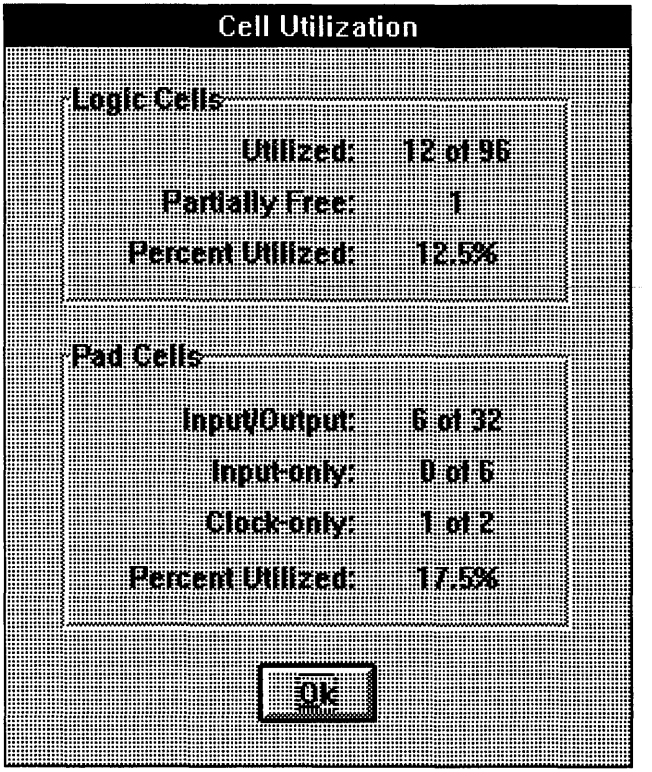


Figure 1-8. Typical Cell Utilization Dialog Box

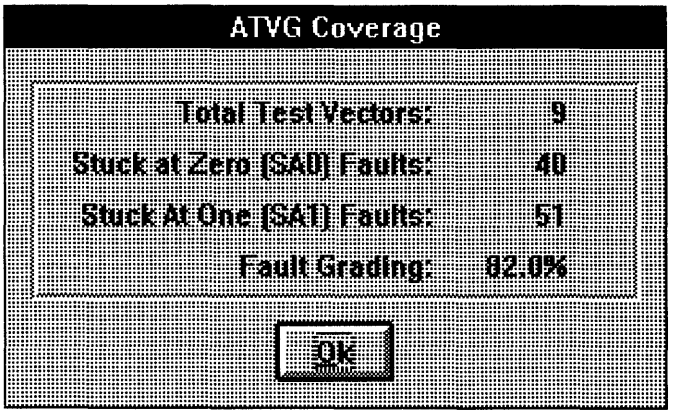


Figure 1-9. Typical ATVG Coverage Dialog Box

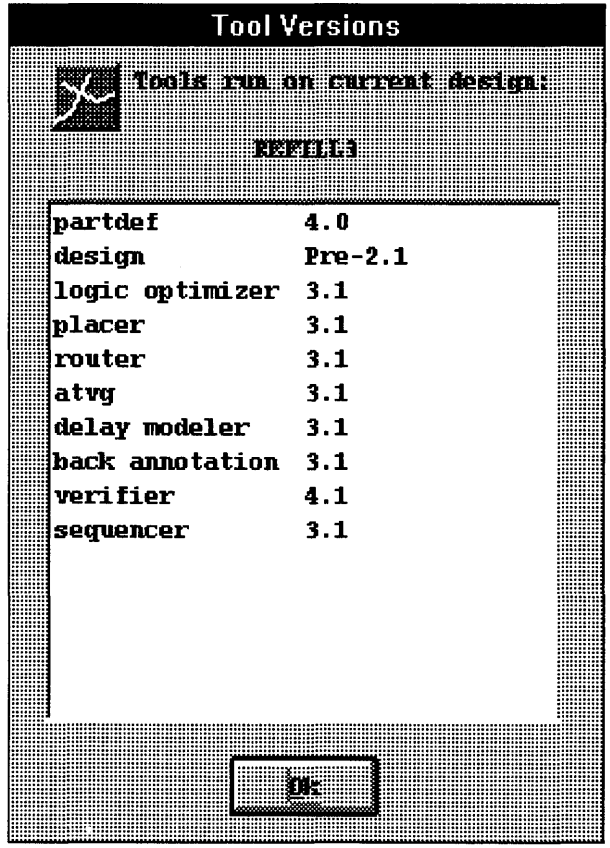


Figure 1-10. Typical Tool Versions Dialog Box

The figure above does not represent the most current releases of the software tools. This figure is for informational purposes only.

1.6 The Help Menu

The Help Menu (Figure 1-11) includes commands to provide on-line help on SpDE.

Index invokes Help with SpDE's on-line help.

Macro Library brings up help on using the macro library cells.

Using Help provides introductory information on using the Help facility itself.

About SpDE provides information on the SpDE Toolkit, including the revision number of each tool and the configuration.

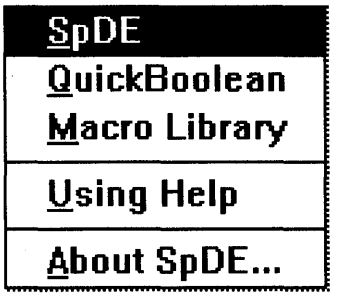


Figure 1-11. Help Menu.

NOTE

Cypress Semiconductor's *Warp* products do not support QuickBoolean.

Chapter 2

SpDE Design Tools

About This Chapter

This chapter describes the five SpDE design tools:

1. Logic Optimizer;
2. Placer;
3. Router;
4. Delay Modeler/Back Annotation;
5. Automatic Test Vector Generator.

2.1. Introduction

This chapter describes the five SpDE design tools: (1) Logic Optimizer; (2) Placer; (3) Router; (4) Delay Modeler/Back Annotation; and (5) Automatic Test Vector Generator.

The Logic Optimizer partitions designs into logic cells using sophisticated technology mapping algorithms.

The Placer takes the design from the Logic Optimizer and places the logic cells in optimal locations on the chip.

The Router connects I/O and logic cells, using the pASIC interconnect resources.

The Delay Modeler calculates delays for use by Path Analyzer, and writes the delays and delay scale to a file for use by the Viewsim simulator.

The Back Annotation tool writes pin numbers and fixed flip-flop numbers to a file that Cypback uses to back-annotate schematics.

The Automatic Test Vector Generator generates test vectors that can be used to test pASIC devices after they have been programmed.

2.2. Logic Optimizer

The Logic Optimizer is the first tool to be run after a design netlist has been loaded into SpDE. The Logic Optimizer uses sophisticated technology mapping algorithms to partition the design into logic cells.

There are two levels of optimization available in the Logic Optimizer: Level 0 - Packer, and Level 1 - Technology Mapper. The optimization levels can be selected from the SpDE Tools Options window (Figure 2-1), which comes up when you select Options/General from the Tools menu.

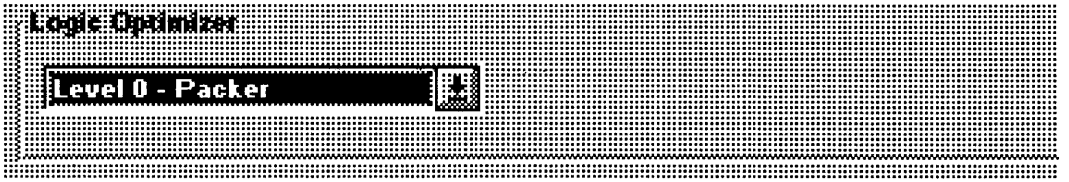


Figure 2-1. SpDE Tools Options Window: Optimizer Options.

Level 0 Optimization: The Packer

The Packer "packs" logic into logic cells, but always leaves the original nets in the design intact. While the Level 1 Optimizer (the Technology Mapper) is the preferred logic optimizer in almost every case, Level 0 optimization is provided for versatility and compatibility with old designs.

Level 1 Optimization: The Technology Mapper

The Technology Mapper uses a more sophisticated algorithm than the Packer. Consequently, the Technology Mapper sometimes takes longer to run. For example, the Packer never takes more than a few seconds to run, but the Technology Mapper can take from a few seconds to several minutes, depending on the complexity of the design.

WARNING Internal nets may be deleted as a result of Level 1 optimization.

Logic Optimization Modes

For Level 1 optimization, you may choose Preliminary, Quality, or Overnight mode from the SpDE Tools Options window. (Level 0 optimization is a simple, predictable algorithm that does not require different modes.)

Preliminary Level 1 optimization completes in half the time of the Quality mode.

Quality Level 1 optimization is recommended for high quality results.

Overnight (or Exhaustive) mode produces slightly better results than Quality mode on some designs, but with a significantly longer run time.

2.3. Placer

The Placer places cells output from the Logic Optimizer, using internal algorithms. For placing critical-path elements, use timing-driven placement from the Path Analyzer.

When run from the Path Analyzer, the Placer determines optimal locations by looking at the nets connecting logic cells, and by looking at timing constraints added by the designer. (See "Timing-Driven Placement," later in this section.)

Placer Options

Figure 2-2 shows the dialog box by which you select Placer options. Two kinds of options are available: you can select the seed value for the Placer, or the placement mode, or both.

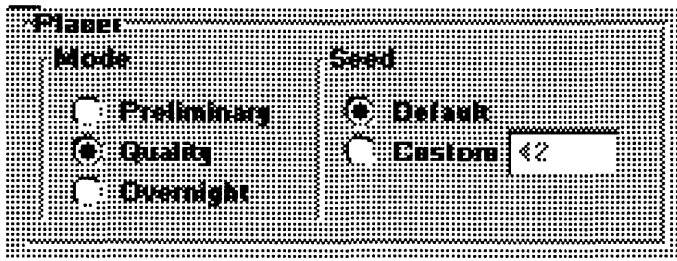


Figure 2-2. Placer Options.

Placer Seed - The placement seed initializes the placement process and sets a starting point for the decisions made during automatic placement. The seed for the Placer is an integer between 1 and 32767. You can use a custom seed, or the default seed value (42). Changing the seed value sets a different starting point for the placer, which can produce a slightly different (and possibly improved) placement.

Placement Mode - Three placement modes are available: Preliminary, Quality, and Overnight. These three modes have the following characteristics:

- **Preliminary** placement is faster than quality placement, but usually by only a few minutes. Results are not as predictable and usually not as good as Quality placement. Cypress recommends that you place designs using at least Quality mode before programming chips.
- **Quality** placement is the default placement mode. As its name implies, it produces high-quality placements.
- **Overnight** placement exists primarily for the curious, or just in case you're going home for the evening and want to keep your computer busy. Results of Overnight mode placement are usually about 4% better than Quality placement, but at a significant cost in run time (about 10X). Thus, a design that places in six minutes in Quality mode will take about an hour in Overnight placement mode.

Timing-Driven Placement

The Placer works closely with the Path Analyzer to provide timing-driven placement, an advanced technique that has been used in gate array placement to produce optimal results. User-specified constraints are fed from the Path Analyzer to the Placer. Paths not meeting the specified constraints are automatically boosted in priority until the constraint is met. Timing-driven placement allows the user to obtain peak performance without resorting to fixed placements.

Constraints can be entered for these paths directly in the Path Analyzer. Once the Path Analyzer has been run, paths not meeting the desired goal can be easily identified.

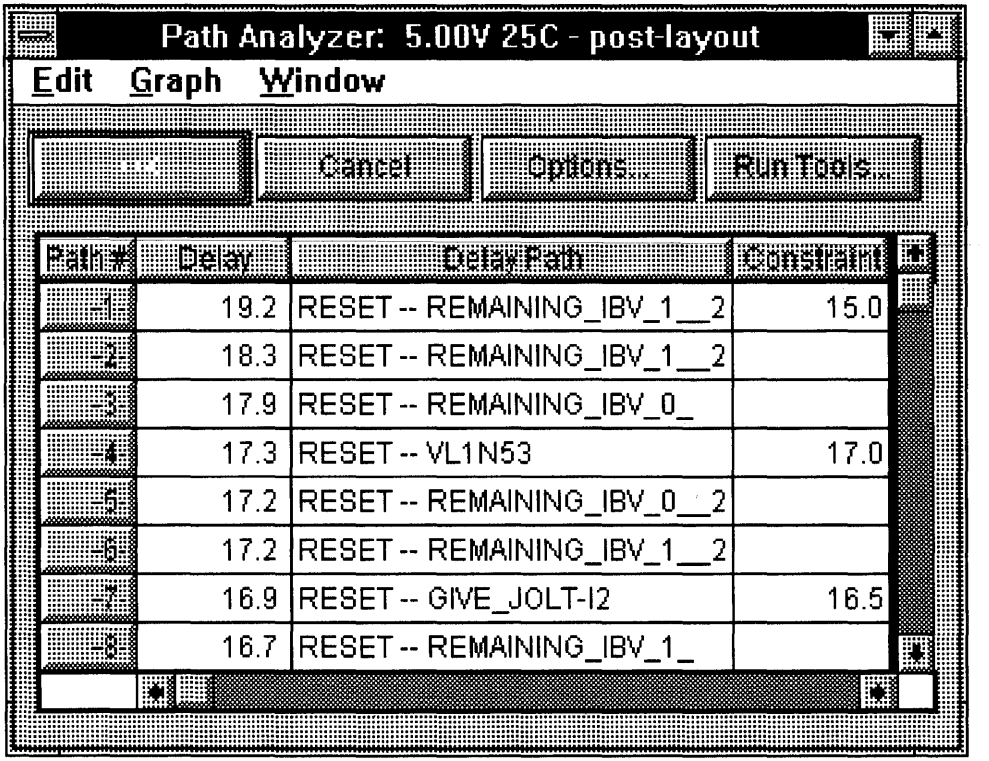
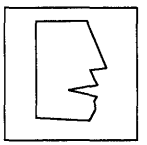
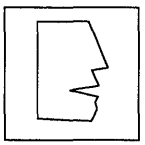


Figure 2-3. Using Timing Constraints in the Path Analyzer.



It is important to set the constraints realistically—set each constraint at or just slightly below the required value. One of the keys to timing-driven placement is the concept of "good enough." Once a critical path has met its constraint, the Placer boosts the priority elsewhere in order to optimize all critical paths.

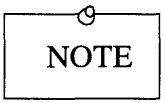
For each path with a constraint, the Placer estimates the delay throughout the placement process. If a constraint is met, the Placer continues to optimize the nets in the path normally. If a constraint is not met, the Placer boosts the priority of the nets in the paths; the boost in priority is proportional to the difference between the constraint and the estimated value. In other words, paths near their constraints are boosted less than paths far from their constraints



Add constraints only where required. The dynamic delay estimation mentioned above adds work to the placement process. Each constraint specified slows the placement process.

You may have to return to your design description and restructure the design in order to achieve the desired performance

Constraints are stored with the design database. Once the constraints have been specified, all subsequent Placer runs operate in timing-driven mode. This can be verified during placement from the **SpDE Status** window—under normal placement the heading is "Placer," while under timing-driven placement the heading is "Timing-Driven Placer."



Constraints are not saved to a separate file, so each time you import a QDIF file into SpDE, all constraints are cleared.

Fixed Placement

Although the Placer automatically determines placement for logic cells and I/O pads, it also supports fixed assignments of both I/O and flip-flops when required. As the name implies, fixed assignments made by the designer are not modified by the Placer.

Fixing the Placement of I/O Pads

Design constraints sometimes require some or all I/O cell locations to be fixed. For example, an existing printed circuit board (PCB) might dictate a precise pinout. Alternatively, a high-speed PCB might require fixing a small number of critical pins in order to limit skew. The SpDE Placer can handle these cases.

To fix I/O pad (i.e., pin) placements, use the pin_numbers attribute in *Warp* VHDL. (See the *Warp Synthesis Compiler Reference Manual* for syntax information about this attribute.)

Fixing the Placement of Flip-Flops

Design constraints rarely require logic cell locations to be fixed. However, to allow designers a greater degree of flexibility, the Placer allows some or all of a pASIC's flip-flop macros to be fixed. One scenario that would dictate fixed flip-flops would be where all the bits of an 8-bit register need to appear on the output pins with absolute minimum skew. The Placer, not realizing this design constraint, might sacrifice the skew on the outputs in order to produce a circuit that was faster overall. By manually fixing the flip-flops on logic cell locations adjacent to the output pins, the designer can meet the design constraint.

To fix flip-flop (i.e., internal) placements, use the fixed_ff attribute in *Warp* VHDL. (See the *Warp Synthesis Compiler Reference Manual* for syntax information about this attribute.)

NOTE

In order for placement to proceed correctly, you must apply the fixed_ff attribute to each element of a bus, and not to the bus as a whole.

Keep in mind that two flip-flop macros cannot be assigned to the same location. The naming convention for pASIC logic cells assigns a character to each column and a decimal number to each row. SpDE verifies the uniqueness of location assignments for fixed placement with the Design Verifier.

Locking Down a Previous Pin Assignment

It is sometimes necessary to "lock down" an I/O placement. This means that for all subsequent place and route runs, you do not want the I/O pins to change their locations.

Use the SpDE back annotation tool and CypBack to fix I/O placements, using the following steps:

1. Select the appropriate options for back-annotation in the SpDE Tools Options window and click OK.
2. Run the Back-Annotation Tool in SpDE (creates, writes out placement information to .ATR file).

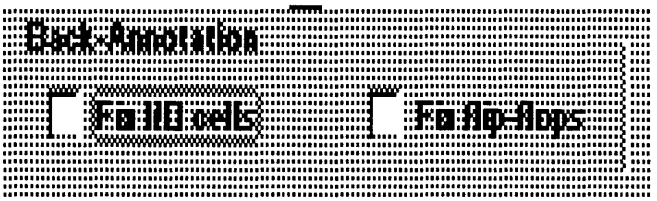
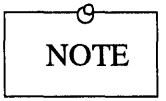


Figure 2-4. Back Annotation Options

3. Run CypBack after exiting SpDE (back-annotates placement information to schematic).



The back-annotator only writes to the schematic, not to the VHDL file.

WARNING The back-annotator cannot back-annotate pin information to buses.

2.4. Router

The Router employs highly optimized algorithms to connect I/O and logic cells using the pASIC interconnect resources. This finely tuned arrangement produces excellent performance with high utilization.

Seed Value

Figure 2-5 shows the area of the SpDE Tools Options window that allows you to set the seed value for the Router. The seed value may be an integer between 0 and 32767, inclusive. If the router seed is changed, the resulting route may be slightly different. This option is rarely needed, but is provided for versatility.

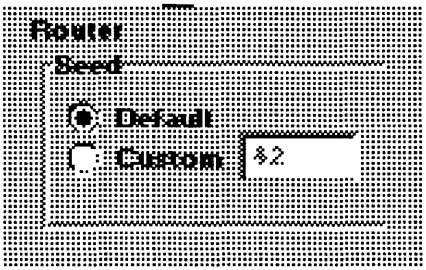


Figure 2-5. Router Options.

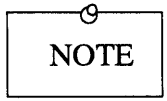
Interconnect Resources

The Router uses four different types of routing resources for fast and efficient connection between logic cells and I/O pads. These resources are **clock networks**, **express wires**, **quad wires**, and **segmented wires**.

Clock Networks: Two clock networks exist on each pASIC device. Both of these dedicated resources are capable of connecting to the clock, set, or reset of any flip-flop in the pASIC device. Each clock network must be driven by one of the clock cells (CKPADs) located at specific pins depending on the package used. (Notice the pins labeled CLK in the device pinout appendix.)

Segmented Wires: Segmented wires are the most abundant routing resource. These wires traverse the distance of one logic cell. High-drive pads cannot drive segmented wires, so the Router restricts nets on High-Drive pads to be routed on quad or express wires.

Quad Wires: Quad wires span four times the distance on the chip that segmented wires do (four logic cells). Quad wires may be used in routing any net in the design, including nets driven by High-Drive pads and parallel logic (see "Special Routing Cases," below).



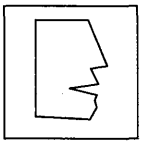
Quad wires are not available in all devices.

Express Wires: Express wires span the entire length or height of the pASIC device. They are used for high-fanout nets, or nets that need to travel across the device.

Special Routing Cases

High-Drive Pads: High-Drive Pads (HDPADs) must drive either quad wires or express wires. On devices that do not have quad wires, high-drive pads must drive express wires.

Parallel Logic: The pASIC architecture allows quad or express wires to be driven from higher-drive sources, such as HDPADs or parallel logic. Parallel logic is a logic configuration in which two identical gates (with the same inputs) have their output nets attached for higher drive capability. There is a restriction on the type of gates that can be tied in parallel. For more information, refer to the **Design Techniques** chapter of this manual and its discussion on **double-buffering**.



SpDE warns you if you use more than the recommended limit of high-drive nets (nets driven by high-drive pads or parallel logic). The router may have difficulty completing successfully in these cases.

2.5. Delay Modeler and Back Annotation

The Delay Modeler and Back Annotation tools are used to calculate the specific timing delays in the pASIC device and to send these timing numbers to a simulator for back-annotated simulation.

Delay Modeler

The Delay Modeler performs a comprehensive timing analysis, accounting for load, slew rate, signal propagation, and intrinsic delay. The tool uses a precise model of the pASIC device and calculates the effects of fanout, packing, placement, and routing.

The Delay Modeler can perform best-case, nominal, or worst-case analysis. The results of the worst-case analysis account for process variation, temperature, and voltage.

The Delay Modeler may be run in **Preliminary** or **Guaranteed** mode (see Figure 2-6). In Preliminary mode, the Delay Modeler uses statistical estimates for the impedance of ViaLinks in the device. In Guaranteed mode, the more accurate ViaLink impedances calculated by the Sequencer are used.

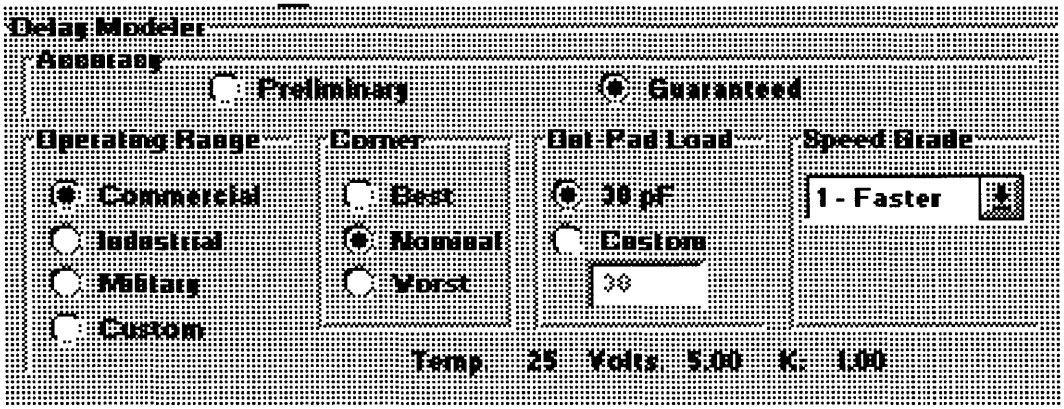
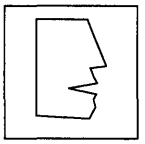


Figure 2-6. Delay Modeler Options.



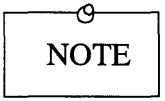
If your machine does not have a math co-processor, the run time of the Delay Modeler may be prohibitive to use while you are debugging your design. In that case, use the Preliminary setting. Once the design is stable, use the Guaranteed setting to ensure proper timing performance.

The **Operating Range** radio button group controls the voltage and temperature ranges used by the Delay Modeler. The default setting is **Commercial**. The **Custom** setting allows a user-specified temperature and voltage to be employed. See "Custom Temperature and Voltage," later in this section.

The **Corner** radio button group selects the corner of the selected operating range. The default is **Nominal** (25 degrees Celsius and 5 volts, regardless of the operating range selected). **Best** selects the lowest temperature and highest voltage in the selected operating range; **Worst** selects the highest temperature and lowest voltage in the operating range. Simulation should be performed at the **Worst** corner.

The **Speed Grade** radio button group selects the pASIC speed grade to be analyzed.

The Out-Pad Load radio button group selects the capacitive loading on the output pins. The default is 30 pF. The Custom setting allows a user-specified load in the range of 0 pF to 150 pF to be employed for all output pins.



The Delay Modeler has been tuned for peak accuracy within the recommended fanout ranges. High-fanout nets that produce fanout warnings are calculated to the highest accuracy possible, but these results are not guaranteed.

Custom Temperature and Voltage

To change the temperature and voltage setting for the Custom operating range, you must edit the spde.ini file located in *install_directory*\spde\data. For the SUN platform, this file should be named .spderc, and should be in your home directory. The following lines should be changed:

PC: spde.ini

```
[delay modeler]
...
CustomVCCBest=5.0
CustomVCCNominal=5.0
CustomVCCWorst=5.0
CustomTempBest=25.0
CustomTempNominal=25.0
CustomTempWorst=25.0
...
```

SUN: .spderc

```
...
delay modeler.customvccbest=5.0
delay modeler.customvccnominal=5.0
delay modeler.customvccworst=5.0
delay modeler.customtempbest=25.0
delay modeler.customtempnominal=25.0
delay modeler.customtempworst=25.0
...
```

The measurement units for CustomVCC variables are Volts. The measurement units for CustomTemp variables are Celsius degrees. The voltage range should not vary more than +/- 10% from nominal (5V). The Best, Nominal, and Worst extensions of these variables represent best, nominal, and worst process factors for the devices. SpDE selects which variable to use, based on the Corner setting from the SpDE Tools Options window.

Back Annotation

The Back Annotation tool produces files to send timing and placement information back to the design entry and simulation tools. A variety of simulation tools are supported for back-annotated simulation. (The simulator that ships with *Warp3* is ViewSim.) You can specify the simulator in SpDE by selecting the Options/Simulator items from the Tools menu (see Figure 2-7).

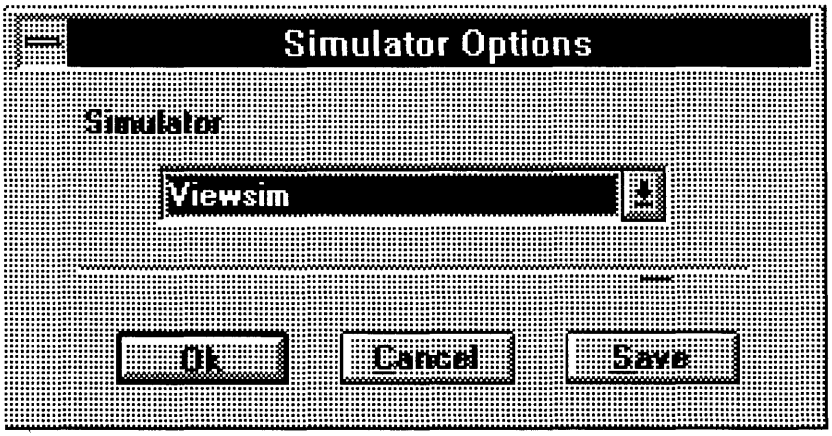


Figure 2-7. Simulator Options for Back Annotation.

If you change the default simulator by making a selection from the Simulator Options window, click on the Save button to write this information into the spde.ini file (.spderc for SUN workstations). Table 2-1 lists the files created by the Back Annotation tool for each of several simulator settings.

NOTE

After you change the simulator, you must still run the Back Annotation tool to create the simulation netlist.

The Verilog simulator also requires a primitive file, which describes the functionality of the primitive components specified in the *design*.v file. This primitive file is design-independent, and is shipped with SpDE. The filename for this primitive file is *pasic_directory*\spde\data\qlprim.v, where *pasic_directory* is the directory where the pASIC Toolkit is installed.

Table 2-1. Back-Annotation Files

Simulator Setting	Files Created	Function
Verilog	*design*.v *design*.sdf	verilog netlist delay back annotation file
ViewSim	*design*.vl *design*.dtb *design*.var	intermediate file for spde2vl delay back annotation file variable values for .dtb file
LMC EDIF	*design*.edo *design*.kf	LMC EDIF netlist device characteristics files
Intergraph EDIF	*design*.edo *design*.kf	Intergraph EDIF netlist device characteristics files

When back annotating to Viewsim, the spde2vl program must be run after back annotation to create the Viewlogic .vsm file needed for simulation.

The icon that runs the spde2vl program from the cockpit is labeled "pasic>vsm".

The Back Annotation tool also supports the back-annotation of pin placement information back to the source design. For information on this process, see Section2.3.

2.6. Automatic Test Vector Generator

The Automatic Test Vector Generator (ATVG) automatically generates vectors that can be used on devices after they have been programmed.

The ATVG tools makes use of an internal scan path in pASIC devices that allows values to be applied to and read from each flip-flop on the device. Once the ATVG tool has been run in SpDE, fault coverage can be ascertained by selecting ATVG Coverage from the Info menu.

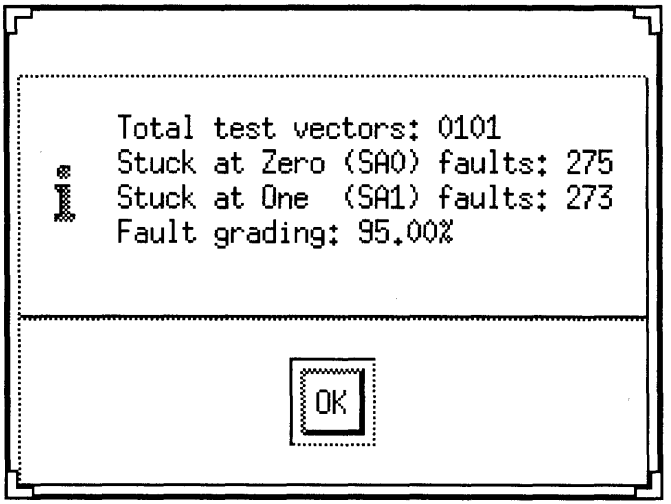


Figure 2-8. Sample ATVG Coverage Window (Sun version).

Testing Overview

A major problem encountered in testing FPGA's is the inability to directly access the vast majority of circuit nodes from the chip periphery. Internal faults, therefore, must be made observable at the output pins by creating a set of input stimuli that will exercise the appropriate path and cause faults to appear as invalid output level changes.

Stuck-At Faults

Stuck-at-0 (SA0) and stuck-at-1 (SA1) fault analysis is an effective means of evaluating test sequences for their ability to detect potential faults in circuits. SpDE's ATVG tool uses an advanced testing technique capable of detecting these faults.

A SA0 fault results from a condition that holds a given signal at a logical 0 regardless of the signal being asserted on that line. Similarly, a SA1 fault is a line that is held at logical 1 regardless of the asserted signal. To illustrate this concept, consider the simple case of the 3-input AND gate shown in Figure 2-9. The truth table for this function is given in Table 2-2.

Figure 2-9. Stuck-at-Fault Example.

There are eight potential faults in this circuit, representing each node (A, B, C, D) stuck-at-0 and stuck-at-1. It is apparent that if A or B or C were stuck at logical low, D would also assume logical low, indicating an incorrect state for the last state (vector 8) of the truth table, and hence a fault. Similarly, stuck-at-1 states for either A or B or C would show up as invalid outputs in vectors 4, 6, or 7 of the truth table, respectively.

By applying appropriate inputs to the circuit, the SA0 and SA1 faults may be detected at node D. Table 2-3 lists these inputs, and their expected responses. Sets of input conditions and resultant output states are commonly referred to as *test vectors*.

Table 2-2. Example Truth Table

A	B	C	D	Vector	Tests
0	0	0	0	1	D for SA1
0	0	1	0	2	D for SA1
0	1	0	0	3	D for SA1
0	1	1	0	4	A, D for SA1
1	0	0	0	5	D for SA1
1	0	1	0	6	B, D for SA1
1	1	0	0	7	C, D for SA1
1	1	1	1	8	A, B, C, D for SA0

Table 2-3 indicates the fault(s) detected for each test vector.

Table 2-3. Test Vector Table

A	B	C	D	Vector	Tests
0	1	1	0	1	A, D for SA1
1	0	1	0	2	B, D for SA1
1	1	0	0	3	C, D for SA1
1	1	1	1	4	A, B, C, D for SA0

As this example demonstrates, a total of four test vectors are required to find all the potential faults in this simple AND gate example. Actually, SA0 and SA1 faults for node D are indistinguishable from faults in the input signals A, B, and C. From a functionality point of view, this is not important since sufficient information is generated to confirm correct or incorrect operation of the circuit.

Fault Grading

Fault grading is a quantitative measure of the testability of a circuit, and is defined by the following expression:

$$FG = \frac{\text{SA0 faults detected + SA1 faults detected}}{\text{total number of detectable faults}}$$

In this example, the total number of detectable faults is six (the total number of detectable faults is simply two times the number of inputs). Note that faults at the output D are not counted because they are "covered" by faults at the inputs. The actual number of potential faults detected by these test vectors is also six, resulting in 100% fault coverage. Furthermore, these test vectors comprise an optimum set required to test the sample circuit, even though the truth table for it has the eight vectors shown in Table 2-2.

Design Considerations

The pASIC's testability features allow the designer to achieve a high degree of fault coverage. Testability can be further increased by designing with a few simple rules in mind.

1. Avoid combinatorial loops.

   Combinatorial loops, such as those shown in Figure 2-10, cannot be tested by the ATVG tool. Logic driven by or driving these loops may be untestable.

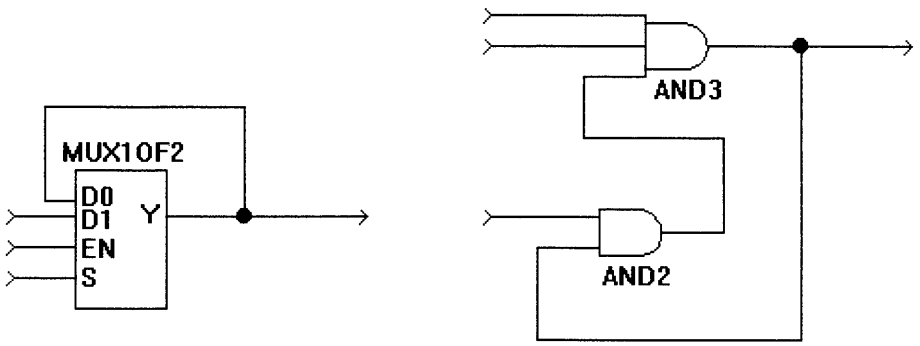


Figure 2-10. Examples of Combinatorial Loops (Feedback).

2. Using the output of a gate or flip-flop to clock, clear, or preset another flip-flop reduces testability.

   The ATVG tool tries to use all flip-flops in the pASIC for testing purposes. Clocking, setting, or clearing a flip-flop by internal logic renders the flip-flop useless to ATVG, reducing testability. Examples of logic structures that reduce testability in this way are shown in Figure 2-11.

   Although gated clocks reduce testability, a buffer or inverter in the path between an input pad and the clock of the flip-flop does NOT reduce testability. Notice also in Figure 2-11 that a flip-flop clock, clear, or set driven by a bi-directional pin (BiPAD) reduces fault coverage

3. Certain input-only pins (HDPADs) should not be used to drive logic that controls an asynchronous set or reset of a flip-flop, as doing so either disables ATVG or reduces coverage.

A subset of the input-only pins (HDPADs) should not be used to drive asynchronous sets or resets (of flip-flops) directly or through a logic path. If so, ATVG will be disabled or fault coverage will be reduced. Table 2-4 shows the pin numbers for different packages.

Table 2-4. Input pins with restrictions in setting and resetting

Package	Input-pins (HDPADs) with restrictions driving sets or resets.	
	Disables ATVG	Lowers Coverage
44 pin PLCC	11	10
68 pin PLCC	17	16
68 pin CPGA	A7	B7
84 pin PLCC	22	21, 66
84 pin CPGA	C6	B7
100 pin TQFP	12	11, 65
144 pin TQFP	18	17, 93
144 pin CPGA	C8	B8, P7

Because of the test mode requirements of pASIC devices, certain input-only (also known as high-drive) pins cannot be used in a multiple HDPAD configuration without disabling ATVG, unless they are used only to drive flip-flop clock inputs. Table 2-5 lists the high-drive pins that cannot be used in HD2PADs or HD3PADs without disabling ATVG.

Table 2-5. Input pins with restrictions on HDxPADs

Package	Pins not to include in HD2PADs or HD3PADs, unless used as clock only (disables ATVG)
44 pin PLCC	10, 36
68 pin PLCC	16, 54
68 pin CPGA	B7, K7
84 pin PLCC	21, 66
84 pin CPGA	B7, K7
100 pin TQFP	11, 65
144 pin TQFP	17, 93
144 pin CPGA	B8, P7

Chapter

3

SpDE Analysis Tools

About This Chapter

After a design has been placed and routed, it is often necessary to analyze timing paths carefully to determine the speed of the design. SpDE contains two tools for this purpose—**Highlight Net** and the **Path Analyzer**.

3.1. Highlight Net

Highlight Net mode helps you analyze a design by highlighting and unhighlighting nets. Highlight Net mode serves no purpose until a design has been placed and routed.

To open the Highlight Nets window (Figure 3-1), select the **Highlight Net...** item from the View menu in SpDE.

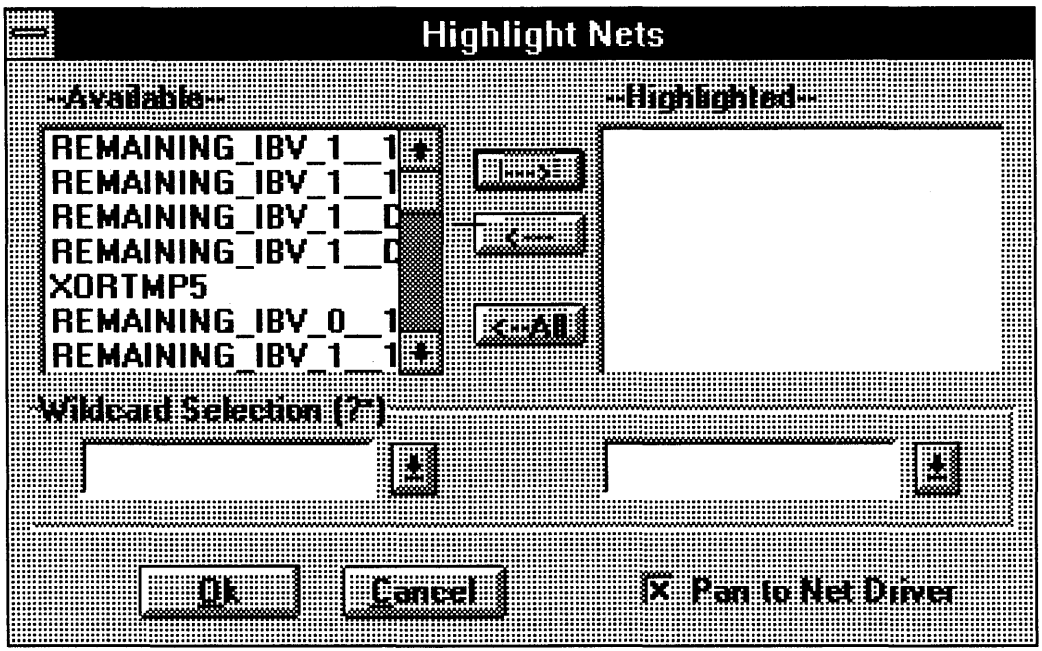


Figure 3-1. Highlight Nets Window.

To highlight nets, select one or more nets from the box on the left, or specify a net name in the "Wildcard Selection" field on the left side of the dialog box, and click on the right arrow button.

To remove nets from the highlight list, select one or more nets from the box on the right, or specify a net name in the "Wildcard Selection" field on the right side of the dialog box, and click on the left arrow button.

When using the "Wildcard Selection" fields, the wildcard characters "*" and "?" are accepted. The "*" character matches 0 or more occurrences of any character. The "?" character matches a single occurrence of a character.

Double-clicking on a net name in either list moves it to the other list.

All nets can be removed from the highlight list by clicking on the ALL button.

Once in highlight net mode, the highlight status may be toggled by clicking directly on the desired nets in the physical view.

To exit highlight net mode, click Cancel. The Physical View will be redrawn in the normal mode.

Double-clicking on a net in the Highlighted box un-highlights the net in the physical view. Clicking on a highlighted wire in the physical view un-highlights the net.

Pan to Net Driver

The Highlight Nets window contains a check box named **Pan to Net Driver**. When this box is checked, SpDE automatically pans to the driver of the net that is selected. This is true whether the net is selected by clicking on a wire in the physical view, or by selecting a net from the available list.

3.2. Path Analyzer

The Path Analyzer is a powerful static timing analyzer that can be used to determine operating frequency, setup and hold times, and clock skew. Working closely with the Physical Viewer, the Path Analyzer instantly identifies critical paths for optimization. Once the critical path has been identified, you can use the Timing-Driven Placer to optimize the placement in order to achieve specified operating constraints.

To run the Path Analyzer, select **Path Analyzer** from the Tools menu. Results are displayed in a four-column spreadsheet format (Figure 3-2).

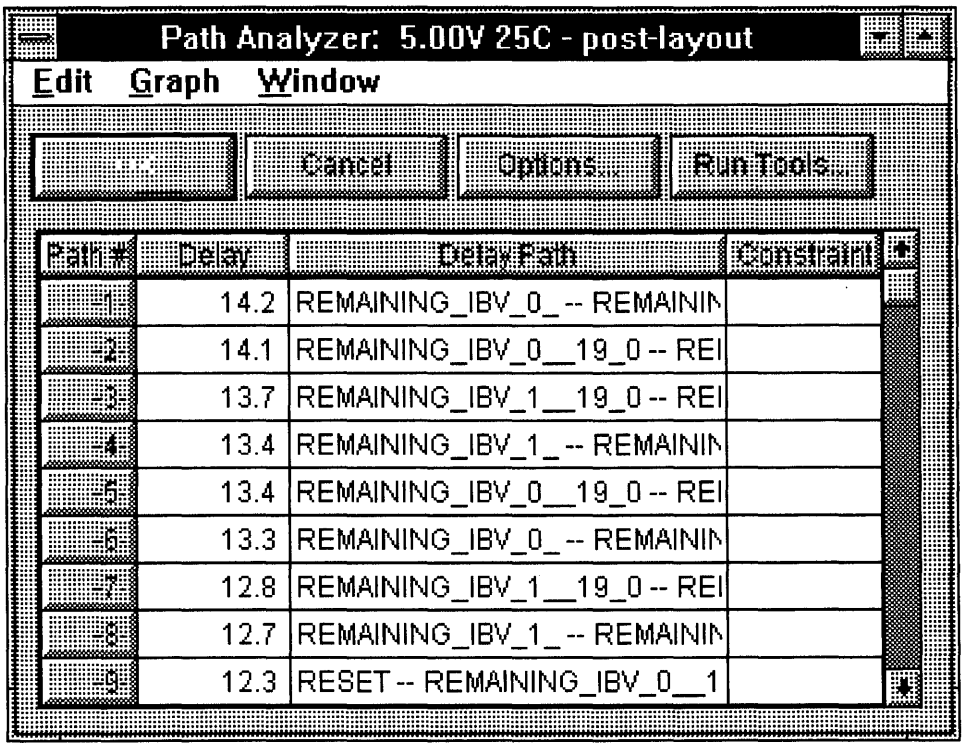


Figure 3-2. Path Analyzer window.

The **Path #** column is displayed in a push-button format, for reasons to be explained shortly. The **Delay** column indicates the delay in nanoseconds. For post-layout analysis, these delays are determined by the Delay Modeler. The **Delay Path** column displays the starting and ending nets of each path.

If the starting point is a pad, the net attached to the off-chip terminal on the pad will be used; if the starting point is a flip-flop, the net attached to the output of the flip-flop will be used. Likewise, if the ending point is a pad, the net attached to the off-chip terminal on the pad will be used; if the ending point is a flip-flop, the net attached to the output of the flip-flop will be used.

Expanding Paths

To expand a path into its component trails, position the cursor over the desired button in the **Path #** column and double-click. The Path # button changes from -26 to +26+ (assuming path number "26") to indicate that the path has been expanded. The component trails are indented and listed in blue to differentiate them from the other **Delay Path** rows. Each trail lists a delay value in nanoseconds, along with an R or F token to denote a rising- or falling-edge delay.

Path Analyzer Options

All Path Analyzer options are set from the **Path Analyzer Options** dialog box (Figure 3-3), which appears when you click the **Options** button in the Path Analyzer window.

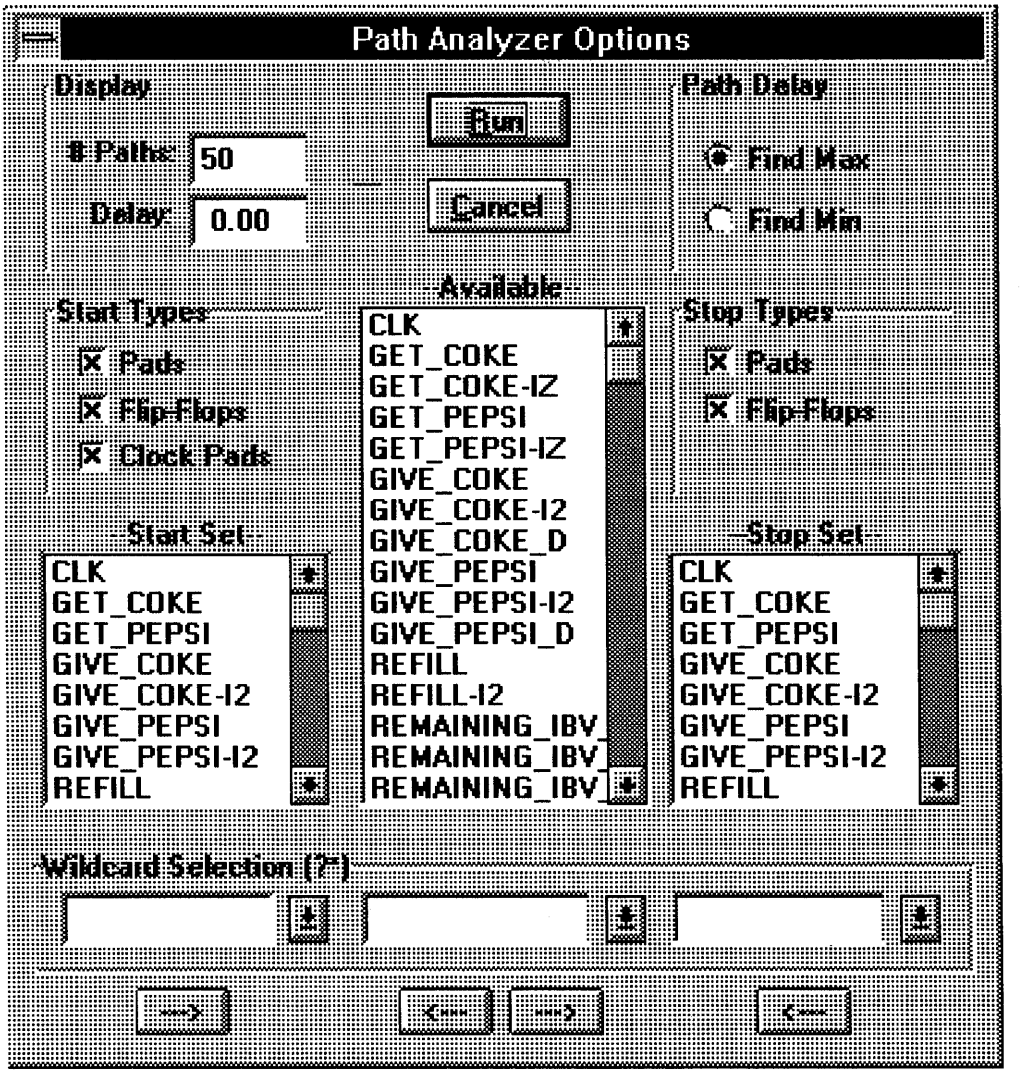


Figure 3-3. Path Analyzer Options Dialog Box.

The **Run** button at the top of the window re-runs the Path Analyzer with the newly specified options. The **Cancel** button returns to the Path Analyzer and discards any newly specified option selections.

The Path Delay group of radio buttons selects maximum or minimum path delays. Each trail along a given path includes a rising-edge delay and a falling-edge delay. If **Find Max** is selected, the Path Analyzer sums the larger of these edge delays at each trail; if **Find Min** is selected, the Path Analyzer sums the smaller of these edge delays. Note that this selection does not change the operating conditions. (In other words, it does not change worst-case commercial to best-case commercial.) **Find Max** lists signals in order of longest delay to shortest. **Find Min** lists signals in order of smallest delay to longest.

The **Display** group determines the number of paths calculated and listed in the path analyzer spreadsheet. The **# Paths** entry limits the number of paths to the specified value. The **Delay** entry is interpreted with regard to the **Path Delay** setting—if **Find Max** is selected, paths are listed if their delay is greater than or equal to the specified value; if **Find Min** is selected, paths are listed if their delay is less than or equal to the specified value.

The remaining lower sections of the dialog box are used to select the **Start Set** and **Stop Set** that specify the desired paths. The **Start Set** list box specifies the starting nets for path analysis, while the **Stop Set** list box specifies the ending nets for path analysis. Providing specific **Start Set** and **Stop Set** information limits the amount of data in the spreadsheet report, making it easier to interpret the results of the Path Analyzer.

The **Start Types** and **Stop Types** check boxes provide the easiest method for selecting the **Start Set** and **Stop Set** list box entries. By default, all of these check boxes are selected. The **Pads** check box selects all nets attached to the external terminals of all pads; this check box selects I/O pads, high-drive pads, and clock pads. The **Flip-Flops** check box selects all nets attached to the output terminals of all flip-flops. The **Clock Pads** check box selects all nets attached to the external terminals of any pad functioning as a clock (not only the internally buffered clocking networks).

Selecting one of these check boxes adds all of the appropriate nets to the desired set. De-selecting one of these check boxes removes all of the appropriate nets from the desired set. For example, assume none of the **Start Types** check boxes are selected. Selecting the **Pads** check box adds all pad nets to the **Start Set** list box. Selecting the **Clock Pads** check box results in no change, as all pad nets are already selected. De-selecting the **Clock Pads** check box, however, removes the clock pad nets from the **Start Set** list box, leaving only non-clock pad nets.

Nets can be selected manually using the **Available** list box in the center of the dialog box. Select a net or nets in this list box, then click on one of the arrow buttons below the **Available** list box. Clicking on the left-arrow button adds the selected nets to the Start Set list box, while clicking on the right-arrow button adds the selected nets to the **Stop Set** list box.

Likewise, the **Start Set** and **Stop Set** list boxes can be "pruned" by selecting a net or nets and clicking on the arrow button below the list box involved.

Groups of nets can be selected using the combo buttons below each list box. In Figure 3-3, for example, the bus IB[0:3] can be selected by clicking in the combo button just below the **Available** list box and typing IB* or IB[?] and pressing the **Tab** key. Once the desired nets are selected, they can be acted upon using the arrow buttons, as described previously.

NOTE

When entering text to select nets from the **Start Set**, **Available**, or **Stop Set** lists, wildcards can be used. An asterisk ("*") represents 1 or more characters; a question mark ("?") represents a single character (e.g., *addr** would select *addr[0]*, *addr[1]*, *addr[2]*, etc.

Graphing

The Path Analyzer provides essential information about the performance of the design. Occasionally, it is useful to view this information in graphical form. The Path Analyzer's Graph menu can be used to create two types of graphs: **Path vs Delay** and **Delay Histogram** graphs, shown in Figures 3-4 and 3-5.

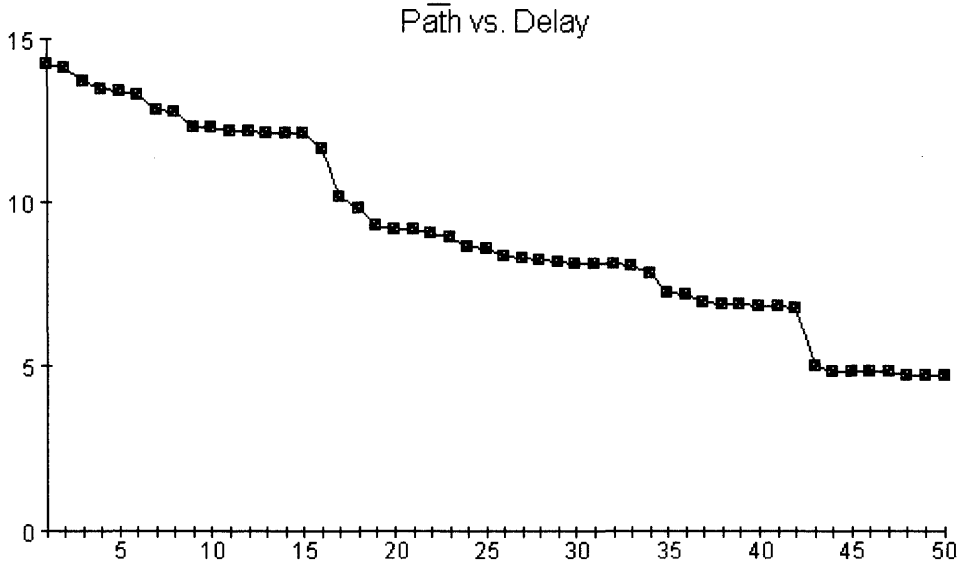


Figure 3-4. Path vs Delay Graph.

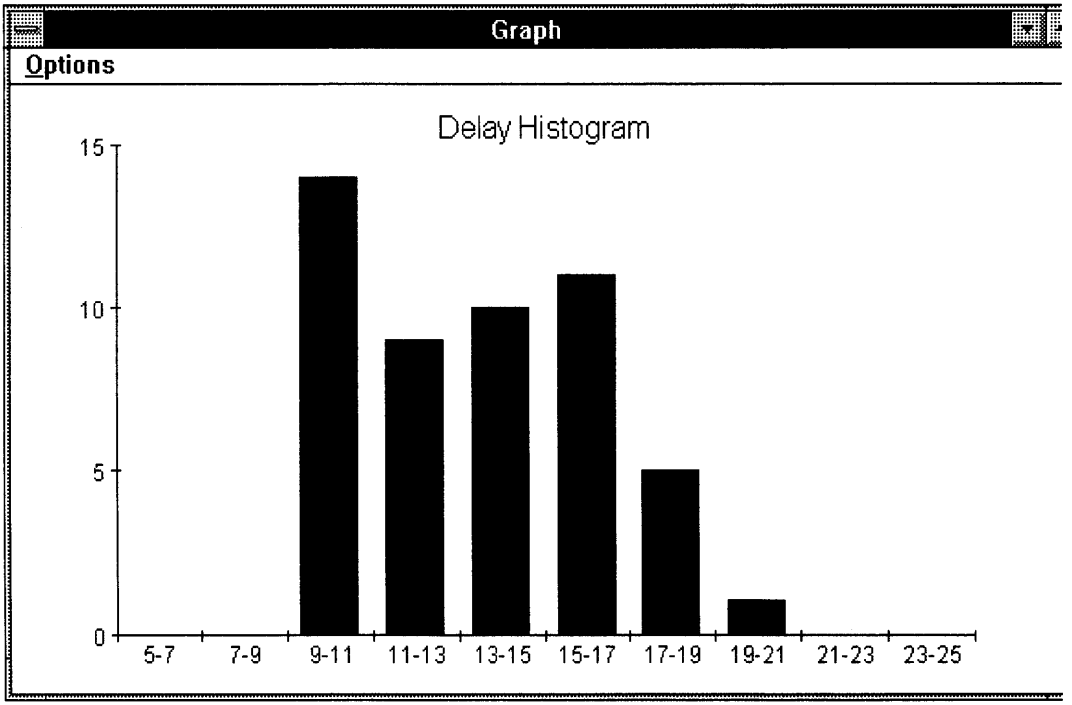


Figure 3-5. Delay Histogram Graph.

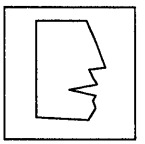
The Path vs Delay graph shows the path delays on the Y axis and the path numbers on the X axis. Double-clicking on the points in this graph has the same highlighting effect as double-clicking on the paths in the Path Analyzer.

The Delay Histogram graph uses a range of path delays as "buckets" on the X axis. The number of paths falling into a delay range "bucket" is shown as a Y value for each range.

Both graphs feature an options menu that provides the capability to copy the graph to the clipboard (as a bitmap) or to print the graph to the current printer. Also, each graph can be customized with the **Graph/Options** menu command from the main Path Analyzer window.

Key Calculations

Using the Path Analyzer, key information can be determined with simple arithmetic.



The results of these calculations will always be conservative, for reasons provided below. The calculations do, however, provide a quick and convenient means of determining worst-case design performance.

Clock Skew

Click the Path Analyzer's **Option** button to bring up the Path Analyzer Options dialog box.

1. Set the **Path Delay** radio button to **Find Max**.
2. In the **Start Types** check box group, activate only the **Clock Pad** check box. Note that you must be using the clock pad as a clock, not an input.
3. In the **End Types** check box group, activate only the **Flip-Flops** check box.
4. Click the **Run** button to execute the Path Analyzer.
5. Make a note of the first path listed and the last path listed. (Note that the # Paths setting must be high enough to list all specified paths; in this case, it's the number of flip-flops used in the design.) The Clock Skew is given by

$$\text{SKEW} = \textit{first\_path} - \textit{last\_path}$$

The clock skew calculation is always conservative, as the calculation ignores the fact that clock skew is meaningful only between flip-flops on a common path.

Operating Frequency

Click the Path Analyzer's **Option** button to bring up the Path Analyzer Options dialog box.

1. Set the **Path Delay** radio button to **Find Max**.
2. In the **Start Types** check box group, activate only the **Flip-Flops** check box.
3. In the **End Types** check box group, activate only the **Flip-Flops** check box.
4. Click the **Run** button to execute the Path Analyzer.
5. Note the delay of the critical path listed. The operating frequency is given by

$$F_{max} = 1/(critical\_path + \text{clock SKEW})$$

It is the designer's task to determine the critical path in his design. This will come from a knowledge of the circuit function and its implementation. The designer may have to use some analysis to determine which paths are the frequency determining paths. Many designs contain false paths, therefore the maximum delay path listed in the path analyzer may not be (and usually is not) the frequency determining path. False paths may include.

1. Data paths with multiplexing and various data sources and destinations where the longest path through the logic is never used.
2. Long paths purposely given to signals which arrive long before they are required. Such paths are encountered in counters where high order stages reach their state long before the low order stage finally triggers a toggle in the last stages of the counter.

Clock skew must be chosen for the critical path or paths identified. The largest skew identified is not necessarily the one to be used in the equation above. The number used must be the skew relevant to the critical delay path.

Setup Time

Click the Path Analyzer's **Option** button to bring up the Path Analyzer Options dialog box.

1. Set the **Path Delay** radio button to **Find Max**.
2. In the **Start Types** check box group, activate only the **Pads** check box.
3. In the **End Types** check box group, activate only the **Flip-Flops** check box.
4. Click the **Run** button to execute the Path Analyzer.
5. Make a note of the first path listed; call it *pads_to_ffs*.
6. Click the Path Analyzer's **Option** button to bring up the Path Analyzer Options dialog box.
7. Set the **Path Delay** radio button to **Find Min**.
8. In the **Start Types** check box group, activate only the **Clock Pad** check box.
9. In the **End Types** check box group, activate only the **Flip-Flops** check box.
10. Click the **Run** button to execute the Path Analyzer.
11. Note the delay of the first path listed; call it *clock_to_ffs*. The setup time is given by

$$t_{setup} = pads\_to\_ffs - clock\_to\_ffs$$

The setup time calculation is always conservative , because the two calculations often apply to different flip-flops.

HoldTime

Click the Path Analyzer's **Option** button to bring up the Path Analyzer Options dialog box.

1. Set the **Path Delay** radio button to **Find Min**.
2. In the **Start Types** check box group, activate only the **Pads** check box.
3. In the **End Types** check box group, activate only the **Flip-Flops** check box.
4. Click the **Run** button to execute the Path Analyzer.
5. Make a note of the first path listed; call it *pads_to_ffs*.
6. Click the Path Analyzer's **Option** button to bring up the Path Analyzer Options dialog box.
7. Set the **Path Delay** radio button to **Find Max**.
8. In the **Start Types** check box group, activate only the **Clock Pad** check box.
9. In the **End Types** check box group, activate only the **Flip-Flops** check box.
10. Click the **Run** button to execute the Path Analyzer.
11. Note the delay of the first path listed; call it *clock_to_ffs*. The hold time is given by

$$t_{hold} = clock\_to\_ffs - pads\_to\_ffs$$

This calculation will typically yield a negative number. The Hold Time calculation is always pessimistic, as the calculation ignores the fact that the two measurements are likely along different paths.

Chapter

4

Design Techniques

About This Chapter

This chapter describes several techniques for speeding up the performance of designs created by the *Warp* system's SpDE tools.

4.1. Speeding Up High-Fanout Nets

For high-fanout, timing-critical nets, designers should consider improving design performance using buffering techinques. In some cases, solutions such as paralleling or pipelining can be used.

Five techniques that can be used to improve circuit performance are described on the following pages:

1. double buffering
2. split buffering
3. selective buffering
4. paralleling
5. pipelining

4.1. Speeding Up High-Fanout Nets

4.1.1. Double Buffering

The pASIC architecture allows a net to be driven by two sources in specific cases. This is called double buffering. Using two gates to drive a high-fanout net speeds up the performance of the net dramatically.

Figure 4-1 is an example of double buffering in a schematic.

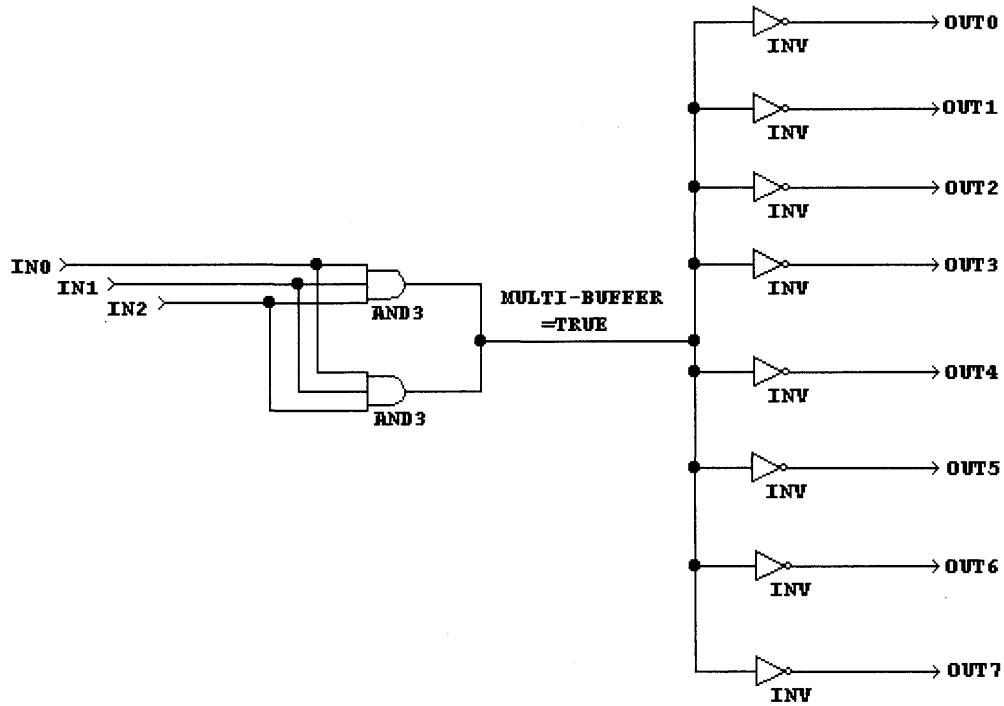


Figure 4-1. Double-Buffering Example.

Double buffering is legal as long as the two gates driving the high-fanout net are identical gates, with the same nets on the inputs and output. Each gate must fit into an AND-fragment (frag_a library element). Double buffering is an excellent performance solution, and offers the best skew and delay characteristics of all buffering solutions for fanouts of 8 to 16.

NOTE

Double buffering on an 8x12 (1000 usable gates) or 12x16 (2000 usable gates) device requires the use of express wires. These devices have limited express wire resources, so only a few double buffers should be used. Refer to the section on the Router for more information.

Using Double-Buffering With VHDL Source Files

In order to use double-buffering within VHDL source files, you must:

1. include the Cypress-provided package `resolutionpkg` in your VHDL source file, using the following statement:

```
USE work.resolutionpkg.all;
```

2. declare the name of the signal to be resolved (i.e., the signal to be driven by more than one driver), and declare it to be of sub-type `multi_buffer` and type `bit`.

The example in Figure 4-2 is a VHDL source file that implements the design shown in Figure 4-1.

```
-- Resolution function for wired-or.  Used to create
-- legal VHDL for double-buffering techniques
-- employed for pasic.
---------------------------------------------------
use work.resolutionpkg.all;
use work.GATESPKG.all;
use work.cypress.all;
use work.rtlpkg.all;

entity DOUBLEBUF is
    port(IN0: IN bit;
         IN1: IN bit;
         IN2: IN bit;
         OUT7: INOUT bit;
         OUT6: INOUT bit;
         OUT5: INOUT bit;
         OUT4: INOUT bit;
         OUT3: INOUT bit;
         OUT2: INOUT bit;
         OUT1: INOUT bit;
         OUT0: INOUT bit);
end DOUBLEBUF;

architecture archDOUBLEBUF of DOUBLEBUF is
  -- net to be resolved
  signal multiple_driver: multi_buffer bit;
begin
  multiple_driver <= IN0 AND IN1 AND IN2; -- driver #1
  multiple_driver <= IN0 AND IN1 AND IN2; -- driver #2
  OUT0 <= NOT multiple_driver;
  OUT1 <= NOT multiple_driver;
  OUT2 <= NOT multiple_driver;
  OUT3 <= NOT multiple_driver;
  OUT4 <= NOT multiple_driver;
  OUT5 <= NOT multiple_driver;
  OUT6 <= NOT multiple_driver;
  OUT7 <= NOT multiple_driver;
end archDOUBLEBUF;
```

Figure 4-2. Example of Double-Buffering in VHDL Source File

4.1. Speeding Up High-Fanout Nets

4.1.2. Split Buffering

Split buffering breaks a wide-fanout net into two or more nets.

Figure 4-3 is an example of split buffering. Without the buffers, the DFF drives a fanout of 8. As configured in the illustration, the DFF drives a fanout of 2, and each buffer drives a fanout of 4.

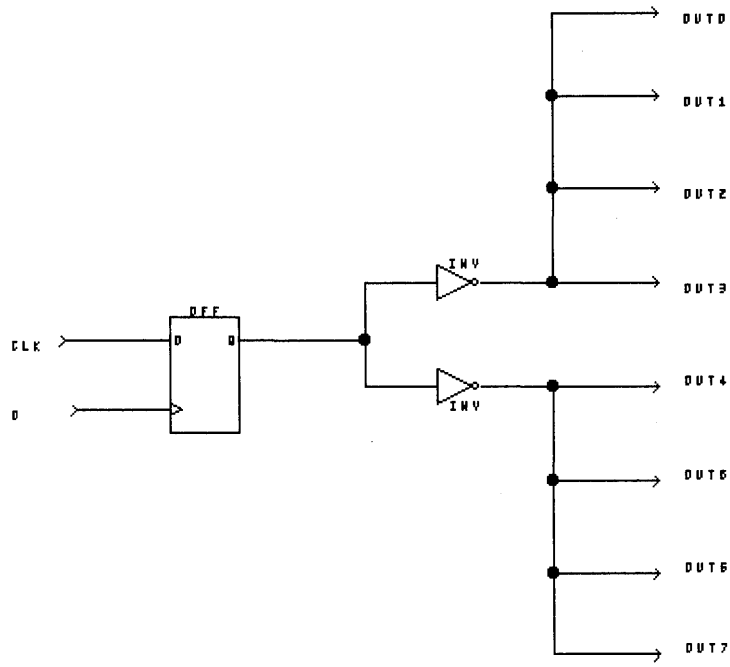


Figure 4-3. Circuit demonstrating split buffering.

NOTE

Adding buffers introduces a logic cell delay to the net. This added delay must be balanced against the gain in reducing the fanout. Simple split buffering (as demonstrated in Figure 4-3) is generally employed only with fanouts of 16 or greater.

4.1. Speeding Up High-Fanout Nets

4.1.3. Selective Buffering

Selective buffering is the selective use of buffers in situations where a high-fanout net has a small number of critical destinations, and a large number of less-critical ones.

Figure 4-4 is an example of selective buffering. The DFF drives a fanout of 8, but only one of the destinations is in the critical path of the circuit. Inserting a single buffer between the DFF output and the 7 non-critical destinations re-structures the circuit so that the DFF drives a fanout of two without adding any logic cell delay in the critical path.

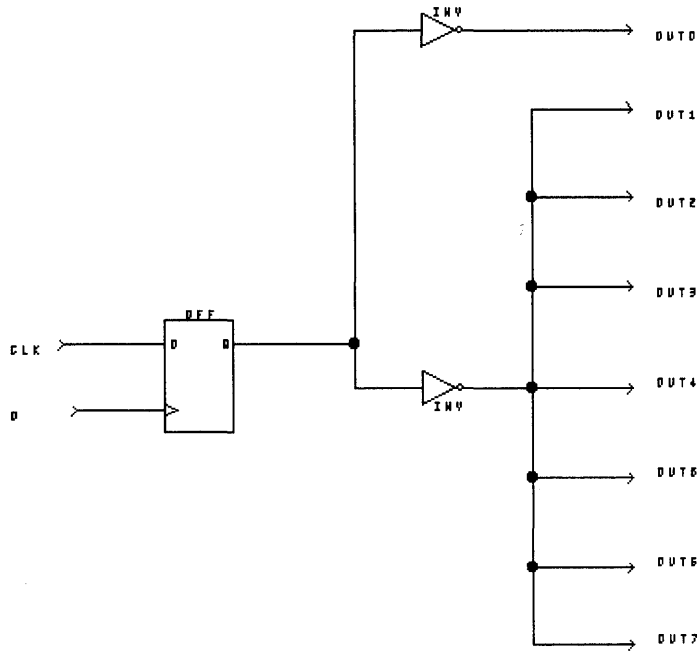
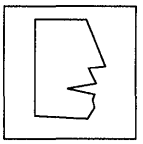


Figure 4-4. Circuit Demonstrating Selective Buffering.



Buffers should be introduced with care and skill. **Selective buffering** offers tremendous improvement in circumstances where the circuit has a few clearly identifiable critical paths.

4.1. Speeding Up High-Fanout Nets

4.1.4. Paralleling

Paralleling is a design technique that duplicates the logic driving a high-fanout load to reduce the effective fanout. Duplicating the logic avoids the delay introduced by adding buffers to the circuit.

Successful buffering must balance reduced fanout against the additional delay caused by the use of buffers. Paralleling is an alternative that does not introduce this added delay.

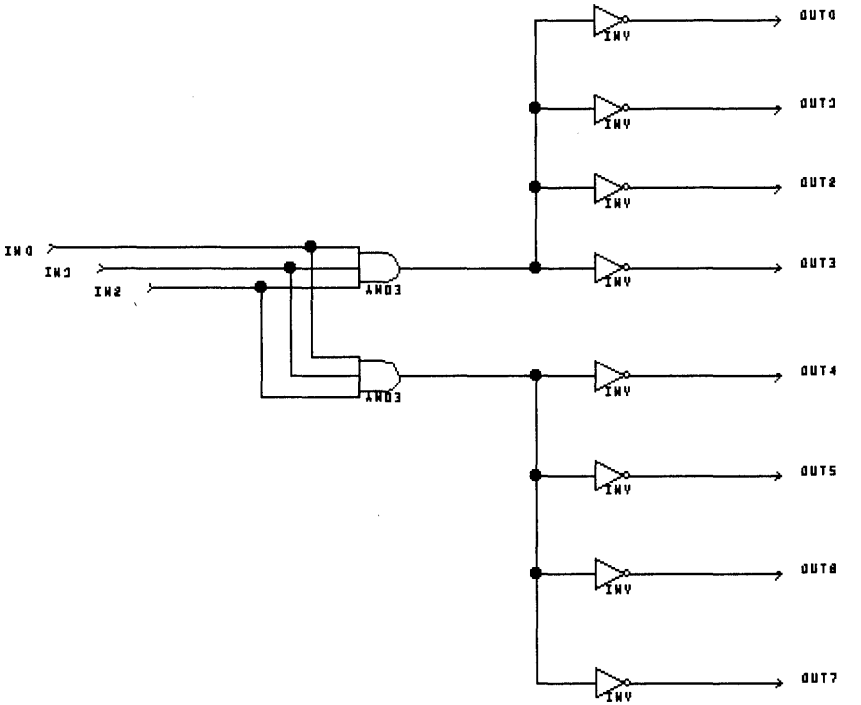


Figure 4-5. Circuit demonstrating paralleling.

Figure 4-5 is an example of paralleling. The AND gate has been duplicated, with each of its inputs tied to the corresponding input on the "twin" gate. Each AND gate drives a fanout of 8, effectively halving the fanout, without introducing the added delay associated with buffering. By duplicating the AND gate, however, the fanout on each of the input nets has been increased.

Notice that paralleling is similar to double buffering, except that the outputs are not tied together. Paralleling should be used instead of double buffering when:

1. skew is not critical;
2. too many express wires have already been used for high-drive inputs or double buffers (see the section on the Router); or
3. the logic to be replicated does not fit into an AND fragment of the larger cell (no larger than a PAfrag_a library element).

4.1. Speeding Up High-Fanout Nets

4.1.5. Pipelining

Pipelining is the technique of inserting registers in long combinatorial paths, effectively increasing the system clock rate.

Inserting registers in long combinatorial paths shortens the length of the critical path and allows operations to be overlapped, increasing the system clock rate. The pASIC architecture promotes pipelining, as each logic cell contains a D flip-flop. As a result, a design can be pipelined with little or no increase in the number of logic cells used.

Appendix A Error Messages

This appendix is a reference of all SpDE messages. You may get error messages after different actions using the SpDE toolkit.

1. **Import - QDIF** from the SpDE menu: refer to the section of this appendix titled **Import Design Verifier**.
2. All numbered error messages from SpDE: refer to the section of this appendix titled **User Errors**.
3. Error messages from other design tools: refer to the documentation for that tool.

A.1. Import Design Verifier

The Design Verifier, which runs when a design is loaded into SpDE, presents **Notes**, **Warnings**, and **Errors** in an interactive list box.

Notes

Notes are intended to bring a situation to the designer's attention. The situation is probably not a problem, but should be verified nevertheless.

Gate <*gate*> is not used, and is being removed.
The Design Verifier has determined that the gate is not being used. This "stripper" function can be deactivated from the SpDE Tools Options dialog box.

Warnings

Warnings alert the designer to a problematic situation, commonly associated with a real problem.

Exceeded recommended limit of high-drive nets.
Too many nets are sourced by HDPADs, CKDPADs, and/or double-buffers (parallel AND gates); the router may not be able to complete. Using fewer signals in tandem with these pads guarantees routability.

Gate <*gate*> cannot have a fixed placement.
The specified gate cannot have a fixed placement. Fixed placements can be applied to logic cells, which utilize the flip-flop in the logic cell.

Gate <*gate*> has no net on pad.

There is no external net defined for an input or an output of the design. The path analyzer will not be able to use this gate as a defined start or stop point in analysis. Add a net and a net name to the pad.

Net <*net*> drives no inputs.

The specified net has a fanout of zero. (The net has a driving gate, but no other connections.)

Net <*net*> has high I/O pad fanout of <*fanout*>.

The specified net has exceeded the recommended fanout limit for a bi-directional pad driver. If the net is speed-critical, employ buffering or paralleling techniques.

Net <*net*> has high input pad fanout of <*fanout*>.

The specified net has exceeded the recommended fanout limit for an input pad driver. If the net is speed-critical, employ buffering or paralleling techniques.

Net <*net*> has high logic cell fanout of <*fanout*>.

The specified net has exceeded the recommended fanout limit for a logic cell driver. If the net is speed-critical, employ buffering or paralleling techniques.

Pin <*pin#*> (<*gate*>) drives set or reset, disabling ATVG.

One of the restricted testing pins (labeled I/SCLK or I/SM in pinout diagrams) is driving a set or reset, directly or indirectly. These pads are restricted testing pins, and require that ATVG be disabled.

Pin <*pin#*> (<*gate*>) paralleled, disabling ATVG.
One of the restricted testing pins (labeled I/SCLK or I/SM in pinout diagrams) is wired in parallel with another pin. These pads are restricted testing pins, and require that ATVG be disabled.

Errors

Errors flag genuine error conditions that would prevent parts from being programmed. However, the tools can still be run for experimental purposes and examination.

Gate <*gate*> has floating input.
The specified gate has one or more unconnected inputs. Floating inputs are not allowed.

Net <*net*> driven by multiple I/O pads.
The specified net is driven by more than one I/O pad. A net cannot be driven by multiple I/O pads.

Net <*net*> has fanout of <24, but >2 drivers.
The specified net has too many high-drive pads. Remove one high-drive pad and re-try.

Net <*net*> has no driver.
The specified net does not have a driving cell; thus, the inputs of the attached cells are floating.

Fatal Errors

Fatal errors flag serious error conditions that prevent the tools from being run..

Clock net <*net*> has multiple drivers.
The specified net is driven by more than one clock pad. Clock nets must be driven by one and only one clock pad.

Dual drive gate <*gate*> is illegally connected.
You have tried to use double-buffering, but incorrectly, OR you illegally tied the outputs of two gates together.

Gate has illegally connected outputs.
Two gates have their outputs tied together illegally. (You may have double-buffered them incorrectly.)

Gate <*gate*> is placed on incompatible cell.
The specified gate has an invalid fixed placement. A bi-directional pad macro may have been placed on an input cell, or vice versa.

Gates <*gate*> and <*gate*> are placed on the same cell.
Two gates cannot be placed on the same cell.

High-drive net <*net*> has opposing pads in a corner.
A net, driven by a high-drive pad, cannot drive a pair of bi/tripads that are at a 90-degree angle to each other in a corner of the chip (e.g., one on the top, one on the right side). Move one of the pads away from the corner and re-try.

High-drive net <*net*> has pads on top and bottom.

Multiple high-drive pads (HD2PAD, HD3PAD, HD4PAD) must have fixed placements. Multiple high-drive pads must be placed on the same side of the chip (e.g., all on the top or all on the bottom of the chip). This error also occurs if you are driving tri-state enables directly from HDPADs. In this case, you cannot fix a pin driven from the HDPAD on the opposite side of the chip from the HDPAD.

Net <*net*> uses clock pad to drive logic inputs.

The clock output tree of the CKPAD cannot be used to drive any logic except for clock pins, asynchronous presets, and clears. Consider using one of the two high-drive outputs of the CKPAD (IN or IZ).

Net <*net*> driven by more than two logic outputs.

The specified net is driven by more than two logic cells. A net can be driven by two logic cells in the case of double-buffering, but a net can never be driven by more than two logic cells.

Net <*net*> driven by multiple sources.

The specified net has an illegal configuration of multiple drivers. The only valid configuration of multiple drivers is two, three, or four high-drive pads.

Net <*net*> is on both sides of an I/O pad.

The specified net has been wired both inside and outside the boundary of a single pASIC. Often, a net attached outside the chip (to a pad, for example) will be named accidentally with a name already used inside the chip.

Net <*net*> uses clock pad to drive logic inputs.

A clock net is being used to drive logic cells. The dedicated clocking structures (CKPADs) may only drive clocks, sets, or resets of logic cells.

Pad on net <*net*> must be pre-placed.

When using HDPADs to drive the enables of more than 16 tri/bipads, the tri/bipads must be pre-placed either on the same side or adjacent sides of the HDPAD placement. The tri/bipads may not be located on the opposite side of the HDPAD. If there are 16 or fewer pads driven from an HDPAD, the Design Verifier performs the placement automatically.

Used <*number*> bi-directional pads with <*max*> available.

You have used more general I/O pads than are available on the chosen device. Remember that some pin positions require special pads, such as input-only pads or clock pads.

Used <*number*> clock pads with <*max*> available.

You have used more general I/O pads than are available on the chosen device. There are only two clock pads (CKPADs) on each pASIC device.

Used <*number*> flip-flops with <*max*> available.

You have used more flip-flops in your design than are available in the chosen device.

Used <*number*> input-only pads with <*max*> available.

You have used more HDPADs in your design than are available on the chosen device. There are six HDPADs available on all pASIC devices.

A.2. User Errors

SpDE reports user errors using an Error dialog box. These errors represent design or system errors that can be fixed by the user. The list below is organized by tool code; the first two letters of the error code indicate the tool.

XX—(starting with any two letters)

xx0100-xx0199	**Out of memory.** SpDE has requested more memory than Windows currently has available. Try closing other applications and re-running SpDE. If the problem persists, try re-starting Windows. Many memory problems can be solved by creating a larger Windows swap file. Windows offers very efficient memory management; refer to the **Microsoft Windows User's Guide** for complete details.

CH—Chip file to QDIF file converter (loads old design files)

CH0001-CH0002	**Error loading binary file:<*filename*>.** **Cannot save QDIF file: <*filename*>.** The converter software is having trouble loading the source design or saving to the destination. This could be due to a full disk, or to a lack of read or write access to the files.

DB—the SpDE Database Module

DB0001- DB0002	**Invalid package type.** An invalid package topic has been chosen for the pA-SIC chip being targeted.

ED—EDIF Netlist Reader

ED0002- ED0003	**Syntax error on line <*line number*>.** Illegal syntax has been used at line <*line number*> in the EDIF file.

ET—EDIF Netlist Reader (EDIF to SpDE Translator)

ET0006	**Unknown package type: <package>** A package that SpDE does not recognize is specified in the EDIF file.
ET0007	**Package has incorrect pin bonding** A pin that does not exist (or is not bonded out) on the selected package is used in the EDIF file. Either the pin number or package type are incorrect.

GP—Graphing Package

GP0001- GP0002	**Error opening clipboard** **Error opening picture** The Grapher could not properly open the picture ~~of~~ or clipboard with Windows calls. Try re-booting your computer.

GP0003- GP0005	**Error closing picture Error closing clipboard** The Grapher could not properly close the picture or clipboard with Windows calls. Try re-booting your computer.
GP0004	**Error putting picture onto clipboard** The Grapher could not complete the operation of copying the graph to the clipboard. You may be low on memory, or Windows could be unstable. Try re-booting your computer.

JE—LOF Netlister

JE0001	**Could not open file <*filename*>** <*filename*> specified by the user either does not exist, or does not have a read attribute.
JE0002	**No LOF support for part <*part*>** The device used for the current design (<*part*>) is currently not supported by the LOF Netlister.

LS—Load and Save Files

LS0001- LS0004	**Could not open binary file <*filename*>** <*filename*> specified by the user either does not exist, or does not have a read attribute.
LS0002- LS0005	**Wrong part file DB version in file <*file*>** An old version of the specified part file exists in the SpDE data directory. Check your WIN.INI file to ensure that the ini-path entry in the [SpDE] section has been properly set.

LS0003-LS0006

Unknown part name *<part>*

The part specified in the design file does not exist, or does not have an associated part file. Check your WIN.INI file to ensure that the ini-path entry has been properly set.

LS0007-LS0010

Part File Errors

This error occurs if SpDE cannot find a current, valid part file. If this error occurs, you may want to re-install SpDE.

LS0011

Unknown package type: *<package>*

A package that SpDE does not recognize is specified in the QDIF file.

LS0200

<error>* at approximately line *<line number>

The parsing error *<error>* occurred while reading line *<line number>* of the QDIF file.

PA—Path Analyzer

PA000x

Clipboard Errors

These errors indicate that the Path Analyzer could not use the Windows clipboard properly. Try re-booting your computer.

PK—Packer (Level 0 Optimizer)

PK0000 **PK0001** **PK0002** **PK0003**	**Cannot pack - too many logic cells** **Too many HDPADs (input-only pads) used** **Too many I/O pads used** **Too many CKPADs (clock pads) used**
	The design requires more of the specified resources than are available in the selected pASIC device. Use fewer of the specified components, or select a larger device.

PK0004	**Illegal fixed I/O location**
	An I/O cell has been assigned to an incompatible pin location. For example, a high-drive pad was placed on a bi-directional pin. Move the fixed placement to an appropriate location.

RT—Router

RT0000 **RT0001** **RT0002**	**Could not complete routing** **Could not complete clock routing** **Could not complete hi-drive routing**
	The router does not have enough resources to complete routing. In the case of hi-drive routing, refer to "Special Routing Cases" in Section 2.4., "Router". Otherwise, try re-placing after changing the placer seed.

RT0003

Out of express wires in channel <x><y>. Re-run placer with another seed.
The router requires more express wires than are available in the specified channel. This problem is most often caused by an excess of signals attached to the high-drive input pads. Employing four or fewer signals in tandem with these pads guarantees routability of these signals.

SD—SDF Writer

SD0001

Cannot open file: *<filename>*
The SDF writer cannot open the SDF file that it needs to create. This could be due to a full disk, or a write-protected file or directory.

SP—SpDE

SP0004

SPDE.INI is read-only or does not exist.
SpDE could not find its initialization file SPDE.INI for saving defaults. This could mean that the file has been erased or that the file is read-only.

SQ—Sequencer

SQ0000

Sequencer could not complete. Re-run Router with a different seed.
The sequencer could not determine an order in which to program the Via-Links in the part. Re-running the placer and/or the router with different seeds should correct the problem.

TM—Technology Mapper (Level 1 Optimizer)

TM0001 **TM0002** **TM0003** **TM0004**	**Cannot pack - too many high-drive pads** **Cannot pack - too many I/O pads** **Cannot pack - too many clock pads** **Cannot pack - too many logic cells**
	The Technology Mapper has determined that the design needs more resources than are available. You may need to select a larger device, or change the design accordingly.

UI—User Interface

UI0001	**There is (are) <*number*> dll(s) not in SpDE's path.**
	SpDE has detected <*number*> DLL's that it needs that are not in the current spDE directory.
UI0002	**Cannot convert chip file <*filename*>**
	SpDE could not convert the chosen chip (.CHP) file to the latest version. Possibly a non-chip file or a chip file from a very old version of SpDE has been selected.
UI0003	**Unable to complete command <*command*> successfully.**
	SpDE tried to execute the command <*command*>, without success.
UI0004	**Invalid directory: <*directory*>**
	The chosen directory cannot be accessed. This may happen if the chosen directory is a DOS drive that has been "joined" to a network directory. Also, the directory may not exist.

UI0005

Can't change to specified directory

Unable to successfully change to chosen directory. This will happen if the chosen directory is a DOS drive that has been "joined" to a network directory.

UI0006
UI0023
UI0024

SPDE.INI is read-only. Cannot save options

SpDE could not read and/or modify the SPDE.INI file. Make sure this file is not in a read-only directory. You may also check the WIN.INI file under [SpDE] to see that the ini-path points to the directory where this file exists. (.spderc in home directory for SUN users).

UI0008

PKZIP.EXE was not found in path

The LOF file cannot be properly compressed unless PKZIP version 1.01 is in the DOS path. Either put PKZIP in the path, or ZIP the file manually using PKZIP 1.01.

UI0009

Unable to run command: *<command>*
Reason: *<reason>*

SpDE could not run a Windows application because of *<reason>*. This could indicate an improper configuration.

UI0010

No printer connected

SpDE could not detect a printer device under Windows. Check Printer Setup in the Windows Control Panel.

UI0011

Printer not set up

SpDE could not print to the default device. Check Printer Setup in the Windows Control Panel.

UI0015

File *<filename>* is from a later version of SpDE...
You have chosen to open a file that was created from a later version of SpDE than is running currently. If you did not intend this, check your configuration, or re-install the latest SpDE tools.

UI0016

Unable to convert *<filename>*
SpDE could not properly convert *<filename>* to the current version of SpDE.

UI0017

Can't initialize gang programmers
You may have a configuration problem with the gang programmers, or a problem with your serial card.

UI0018

No gang programmers found
Check to make sure all gang programmers are connected and plugged in, and that the correct COM port has been chosen.

UI0019

No automatic place and route tools were run - Check Place and Route option settings.
You have chosen to Run All Tools after all the tools have already been run. If you wish to iterate: change seeds in the Tools Options, then re-run tools with Run Selected Tools.

UI0020
UI0021

Cannot process SPDE.INI file
The SPDE.INI file has been corrupted.

UI0022

Error opening report file <*filename*>
SpDE could not open the report file it has created. This could happen if you were too low on memory to load the chosen editor, or if the chosen editor could not be loaded properly. Change the chosen editor from View/ Preferences.

UI0034

Cannot load QDIF file <*filename*>
An error was detected while reading a QDIF file. The file may have a syntax error, or the file may have been damaged.

UI0037

Ini-path not found in win.ini. **Using c:\pasic\spde\data**
SpDE expects to find the variable ini-path under the heading [SpDE] in the win.ini file. The installation program will do this automatically. Check the win.ini file, or re-install SpDE.

UI005x

Save Error
A DOS error was detected while trying to save a file. This may be caused by a write-protect violation or insufficient disk space.

UI006x

Load Error
A DOS error was detected while trying to load a file. This may be caused by choosing the wrong file type to load, or trying to load a file without a read attribute.

VE—SpDE Physical Viewer

VE0007
VE0010
VE0012

Value must be between <*min*> and <*max*>
The value you have entered is out of the allowable range. Enter a value between <*min*> and <*max*>.

VE0009
VE0011

Bad (unsigned) integer value
The value you have entered does not represent a proper integer value. If SpDE is expecting an unsigned integer, make sure the number is positive. Always make sure integers do not have decimal points.

VG—Verilog Netlister

VG0001

Error: cannot open file: <*filename*>
The Verilog netlister cannot open the output file it is trying to create. This could be due to a full disk or a read-only directory.

VL—Viewlogic Netlister

VL0000-
VL0004,
VL0006

Error: cannot open file: <*filename*>
The Viewlogic netlister could not access the specified file. Check Viewlogic environment variables and write-access of specified directory; also check to be sure specified file exists.

VL0005
VL0007

Cannot write to file: <*filename*>
The Viewlogic netlister could not write to the specified file. Check available disk space and write permission on the specified directory and file.

A.3. Internal Errors

These errors are indicated by the title bar **Internal Error**, as well as message of a (usually) cryptic nature. These error should not occur—they indicate an inconsistency in SpDE's data structures.

If you encounter one of these errors, record the text of the error completely and contact Cypress Semiconductor Corporation.

Index

Warp3 Installation

For PC's:

Installing *Warp3* on your IBM PC or compatible computer requires about 55 Megabytes of hard disk space. The installation procedure is as follows:

1. Write down the serial number found on the first disk; you will be asked for it later.
2. Insert disk 1 into a floppy disk drive.
3. Select File/Run from the Windows program manager.
4. Type `a:install` or `b:install`, as appropriate, and select OK.
5. Answer questions from the installation script. We recommend that you install Warp3 in an empty directory.
6. Place your copy of the "license.wv" file in the first directory of the WDIR path. This is usually C:\WARP\WARPSTD.

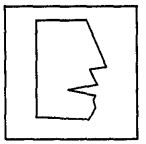
If you choose the option to create example DOS configuration files, the files are created with a ".exm" extension, e.g., the example file for "autoexec.bat" would be "autoexec.exm". The three files that will be modified are autoexec.bat, system.ini and win.ini. The latter two files are found in your windows directory.

Note that the example file only contains the difference between the old and new files, not a complete copy. Because of this, Cypress recommends that you simply choose the option to go ahead and modify the originals. The installation procedure copies the original file to a ".bak" extension.

NOTE

IMPORTANT DOS REMINDER...

Your PATH must be shorter than 128 characters. This is a DOS limitation.



IMPORTANT WINDOWS TIPS...

1. The Optimizing Windows chapter of the Microsoft Windows User's Guide includes numerous ideas to increase performance. We especially recommend creating a SMARTdrive of 1 to 2 Megabytes. We also recommend a Windows 3.1 virtual memory size of 16 Megabytes.

2. Some PC clones utilize the upper portion of the first megabyte of memory in a manner that conflicts with Windows. If any *Warp3* (or other) application "hangs," try adding the following line to your SYSTEM.INI file (found in the Windows directory) in the section marked as indicated:

```
[386Enh]
        EMMexclude=A000-FFFF
```

After adding this line, re-start Windows. If the problem persists, consult your PC manufacturer for additional assistance.

For Sun:

Installing *Warp3* on your Sun workstation requires about 38 Megabytes of hard disk space. The installation procedure is as follows:

1. Install Viewlogic software from tapes. The following are required Viewlogic software packages to install for Warp3. Please read the file "install.pkg" (found on the Viewlogic Powerview tape) for other options.

   ```
   VHDL Developer Package
   Standard, Plotting & Admin
   Solution Packages->Digital Design Entry Soln Package
   Individual Prods->Design Analysis Prods->ViewSim
   Individual Prods->Design Analysis Prods->Export 1076
   ```

2. To install the *Warp3* software, create the directory where you want the software to be located.

3. For installation from a tape, tar the contents of the tape, run the install script and answer the questions:

   ```
   tiger% mkdir /usr/local/warp
   tiger% cd /usr/local/warp
   tiger% tar xvf /dev/rst8
   tiger% ./install
   ```

   For installation from disks, insert the first disk and bar the contents before running the install script:

   ```
   tiger% mkdir /usr/local/warp
   tiger% cd /usr/local/warp
   tiger% bar xvfZ /dev/rfd0
   tiger% ./install
   ```

4. When asked for the serial number, copy the number from the tape (or first disk) label.

5. In your .cshrc or .login file, set a variable called CYPRESS_DIR pointing to your install directory. Make sure your path includes */your_Cypress_directory/bin* and */your_Powerview_directory*.

   For example:

```
setenv CYPRESS_DIR /usr/local/warp
set path=($path $CYPRESS_DIR/bin)
```

6. Set your WDIR environment variable as follows:

   1) a writable directory for Powerview to place the /vf directory;

   2) /your_Cypress_directory/warpstd, and

   3) /your_Powerview_directory/standard

   For example (should be on one line):

```
setenv WDIR $HOME/powerview:$CYPRESS_DIR/warpstd:/
usr/local/Powerview/standard
```

7. Set your pASIC environment variables in your .cshrc file as follows:

```
setenv SPDE_ROOT $CYPRESS_DIR/spde
setenv XVTPATH $SPDE_ROOT/print
setenv UIDPATH $SPDE_ROOT/%N.uid
set path=($path $SPDE_ROOT)
```

8. Copy the pASIC configuration file into your home directory:

```
cp $CYPRESS_DIR/spde/.spderc $HOME
```

9. By default on the Sun Sparc, the Galaxy and Nova interfaces use the OpenLook toolkit. If the user wishes to use the Motif interface instead, set the environment variable CYPRESS_MOTIF to TRUE.

For example:

```
setenv CYPRESS_MOTIF TRUE
```

Adding this command to your .login or .cshrc files will make this interface change permanent.

10. Either install the ViewLogic software directory into a directory called /usr/local/Powerview, or have a symbolic link located in /usr/local/Powerview that points to the location of the ViewLogic directory. This allows Galaxy to automatically call the VHDL analyzer when it finishes creating the ViewSim Model. To do this, become root and enter the following command:

    ln -s */your_Powerview_directory* /usr/local/Powerview

11. Place your copy of the "license.wv" file in the first directory of the WDIR path.

Note that the Sun Sparc documentation for the ViewLogic software is on-line (after you install it). Hard copies of the manuals can be purchased from ViewLogic. Call ViewLogic at (508)480-0881 for more information.

Debugging Common Sun License File Problems

There are two types of licenses available on the Sun: node-locked and floating.

The following paragraph applies to floating licenses only:

You will need to add the line "LICENSE_SERVER hostname" at the end of the workview.ini file which is located in a subdirectory under powerview. The hostname of your machine can be obtained by typing "hostname" at a UNIX prompt. You will also need to run the license administrator called "ladmin". Type "ladmin" then enter "start" and "quit". This will launch the required vnsd process. You may want the license server to start automatically when the machine reboots. See the powerview documentation for instructions on adding the appropriate commands to the Sun startup file.

The date on the Sun must fall between the start and expiration date in your license file. Type "date" to verify this. If date is ok, then focus on the license file. Check that the hostid of the server matches that of the license file. Type "hostid" to get the hostid.

Many problems are the result of an incorrectly entered license file. Carefully check for differences from the original file such as missing products or missing dashes, misspellings or differences in start and stop dates from version 4.1 to 5.1. Even minor differences in the file will cause the message "no valid license found" to be displayed.

The WDIR environment variable (usually set in the .cshrc file) tells powerview the location of the license.wv file. Check to make sure that only one license file exists, and it is correctly placed in the WDIR path.

If you have any additional questions or problems regarding your license file please contact Cypress applications at 1-800-419-1481 or (408) 943-2821 or refer to the ViewLogic documentation.