Cupl™

User's Guide

91-10128-5

Contents

Copyright ii

Introduction iii

USER GUIDE iii

Chapter U1
Introduction iii

Chapter U2
Installation iv

Chapter U3
Getting Started iv

Chapter U4
CUPL Operation iv

Chapter U5
Design Examples iv

REFERENCE iv

Chapter 1
CUPL Language iv

Chapter 2
Using CSIM iv

Chapter 3
Using CBLD v

Chapter 4
Using PTOC v

APPENDICES v

Appendix A
Error Messages v

Appendix B
Device Usage Notes v

Appendix C
Download Formats v

Appendix D
Node Numbering v
Appendix E
Trouble Shooting v
Index v

Conventions Used vi

Introduction to CUPL 1
CUPL OVERVIEW 1
CUPL DATA FLOW 3

Installing CUPL™ 6
MS-DOS INSTALLATION 6
Specifying the Configuration 9
The CUPL Menu System 10
The EZ Edit Editor 13
UNIX INSTALLATION 15
Installing the Software on Sun Workstation 15
Setting Up the Environment 16
Installing the Software on Apollo Workstation 18
Setting Up the Environment 19
VAX/VMS INSTALLATION 22
Installing the Software 22
Setting Up The Environment 23

Getting Started 25
First Steps 25
Simple Logic Design 26
Simple Gates Example 28
Compiling the Source File 32
Simulating a Design 32

CUPL Operation 36
CUPL OPERATION 36
Input 36

Output 38
Running CUPL From the Command Line 39
Running CUPL Using the Menus 46

Design Examples 49
PART A. SAMPLE DESIGN SESSION 50
EXAMINING THE DESIGN TASK 50
CREATING THE CUPL SOURCE FILE 54
FORMULATING THE EQUATIONS ... 56
CHOOSING A TARGET DEVICE 59
MAKING THE PIN ASSIGNMENTS ... 61
RUNNING CUPL 63
CREATING THE CSIM SOURCE FILE 72
RUNNING CSIM 76
SUMMARY 81
PART B. SAMPLE PLD FILES 82
EXAMPLE 1. SIMPLE GATES 85
EXAMPLE 2. CONVERTING A TTL DESIGN TO PLDs 88
EXAMPLE 3. TWO-BIT COUNTER 95
EXAMPLE 4. DECADE UP/DOWN COUNTER 97
EXAMPLE 5. SEVEN-SEGMENT DISPLAY DECODER 102

CUPL Language 107
LANGUAGE ELEMENTS 107
Variables 107
Indexed Variables 108
Reserved Words and Symbols 109
Numbers 110
Comments 112
List Notation 112
Template File 114

Header Information 116
Pin Declaration Statements 119
Node Declaration Statements 122
Bit Field Declaration Statements 125
MIN Declaration Statements 126
FUSE Statement 127
Preprocessor Commands 128
$DEFINE 128
$UNDEF 130
$INCLUDE 130
$IFDEF 131
$IFNDEF 132
$ENDIF 133
$ELSE 134
$REPEAT 135
$REPEND 137
$MACRO 137
$MEND 139
LANGUAGE SYNTAX 140
Logical Operators 140
Arithmetic Operators 141
Arithmetic Function 142
Extensions 142
Feedback Extensions Usage 147
Multiplexer Extension Usage 149
Boolean Logic Review 151
Expressions 152
Logic Equations 153
APPEND Statements 155
Set Operations 166
Equality Operations 168
Range Operations 172
Truth Tables 169
State-Machine Syntax 171
State-Machine Model 171

Inputs 172
Combinatorial Logic 172
State Bits 173
Storage Registers 173
Nonregistered Outputs 173
Registered Outputs 173
Syntax 174
Unconditional NEXT Statement 176
Conditional NEXT Statement 178
Unconditional Synchronous Output Statement 183
Conditional Synchronous Output Statement 186
Unconditional Asynchronous Output Statement 192
Conditional Asynchronous Output Statement 194
Sample State-Machine Syntax File 197
Condition Syntax 199
User-Defined Functions 201
CUPL to XILINX XNF Interface 205
Getting an XNF File 206
Translating an Existing PLD File. 207
Using a PALASM Device 208
Source Files for XILINX Devices 208

Using CSIM 210
INPUT 210
OUTPUT 212
RUNNING CSIM 213
Simulator Option Flags 214
Viewing Waveform(MS-DOS) 216

Change Signal Order 216
Group Signals into Bus 218
Create Waveform Hardcopy 219
Help Menu 220
TEST SPECIFICATION FILE 220
Header Information 221
Comments 222
Statements 222
ORDER Statement 222
BASE Statement 224
VECTORS Statement 226
Preload 228
Clocks 229
Asynchronous Vectors 230
Simulator Directives 232
$MSG 233
$REPEAT 233
$TRACE 234
$EXIT 237
$SIMOFF 237
$SIMON 237
Fault Simulation 238

Using CBLD 239
RUNNING CBLD 239
LISTING THE CONTENTS OF A LIBRARY 241
LISTING ALLOWABLE EXTENSIONS 242
LISTING ALLOWABLE MACROS 243
BUILDING DEVICE LIBRARIES 245

Using PTOC 248
RUNNING PTOC 248
PALASM SOURCE FILE FORMAT 249
PTOC .PLD OUTPUT FILE 250
Header Information 251
Pin List 251

Equations 252
PTOC .SI OUTPUT FILE 253
Translation Ambiguities 254

Error Messages 256
CUPL ERROR MESSAGES 257
CUPL Module Error Messages 258
CUPLX Module Error Messages 259
CUPLA Module Error Messages 262
CUPLB Module Error Messages 269
CUPLM Module Error Messages 273
CUPLC Module Error Messages 275
CSIM ERROR MESSAGES 278
CSIM Module Error Messages 278
CSIMA Module Error Messages 279
CBLD ERROR MESSAGES 283
CBLD Module Error Messages 283
PTOC ERROR MESSAGES 286
PTOC Module Error Messages 286
WCSIM Error Messages 287

Device Usage Notes 289

Download Formats 304
DOWNLOADABLE FILE FORMATS 304
JEDEC Format 304
ASCII-Hex Format 309
HL Formats 301
DOCUMENTATION FILE FORMAT 323
PDIF FILE FORMAT 330
BERKELEY PLA FILE FORMAT 331

Node Numbering 335
Advanced Micro Devices 336
Altera 338
Atmel 338
Cypress 339
Intel 341

Lattice 342
Monolithic Memories 343
PLX Technology 343
Signetics 344
Texas Instruments 351

Trouble Shooting 353
HOW TO GET SUPPORT 353

Copyright

Copyright © 1983, 1990 by Logical Devices, Inc.(LDI)

All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means —electronic, mechanical, photocopying, recording, or otherwise — without the written permission of LDI.

Logical Devices, Inc. provides this manual "as is" without warranty of any kind, either expressed or implied, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. LDI may make improvements and/or changes in the product(s) and/or program(s) described in this manual without notice.

Although LDI has gone to great effort to verify the integrity of the information herein, this publication could contain technical inaccuracies or typographical errors. Changes are periodically made to the information herein. These changes will be incorporated in new editions of this publication.

TRADEMARKS

CUPL, NX-CUPL and ON-CUPL are trademarks of Logical Devices, Inc. Schema is a trademark of Omation. Wintek and HiWIRE are registered trademarks of Wintek Corporation.

Logical Devices, Inc. 1201 NW 65th Place, Ft. Lauderdale, FL. 33309 USA Technical Support Telephone: (305) 974-0975

Introduction

This book is designed to serve as a learning aid and as a reference manual for CUPL, the programmable logic compiler from Logical Devices, Inc. It is divided into three sections: the User Guide section, the Reference section and the Appendices. The User Guide provides information on getting started with the CUPL package. The Reference section provides specific information about the programs that make up the CUPL package. The Appendices contain a variety of information, including error messages, important information about specific devices, node numbering and download formats.

Be sure to read the Installation Chapter in the User Guide section to find out what to do before using the CUPL software on a system. Since improvements are made to CUPL quite frequently, additional information is provided in the form of an addendum. Be certain to read the addendum, which provides information on the latest enhancements to the CUPL software.

USER GUIDE

The USER GUIDE provides information on installing and using CUPL. There is also a section that provides several examples. These examples are provided on the distribution disks.

Chapter U1: Introduction
Gives a brief overview of CUPL and includes some notes on recent enhancements to CUPL. Also included is a section on getting technical support.

Chapter U2: Installation
Provides information necessary to get CUPL to run on a system.

Chapter U3: Getting Started
Is a step-by-step guide on how to create a source file, and compile and simulate the design using a simple example.

Chapter U4: CUPL Operation
Is a description of the use of CUPL. This includes running **CUPL**.

Chapter U5: Design Examples
Provides instructions on using **CUPL** and **CSIM** and examples of CUPL designs. Part A, Sample Design Session, goes step-by-step through a sample design session using **CUPL** and **CSIM**. Part B, Sample PLD files, illustrates some typical designs that can be created using **CUPL**.

REFERENCE

The REFERENCE section is provided for easy access to random facts related to the CUPL language and related utilities.

Chapter 1: CUPL Language
Describes CUPL, a programmable logic compiler. This chapter provides an overview of CUPL, and describes the input and output and how to run the program. It also describes the elements of the CUPL design language, and the syntax of the CUPL design language.

Chapter 2: Using CSIM
Describes **CSIM**, a logic simulation program. This chapter explains how to use **CSIM**, including input and output and running the program, and how to create test specification files to verify a **CUPL** design.

Chapter 3: Using CBLD
Describes **CBLD**, a utility program for managing device libraries.

Chapter 4: Using PTOC
Describes **PTOC**, a utility program for converting PALASM source files to CUPL format.

APPENDICES

Appendix A: Error Messages
Lists error messages that may appear during operation of any of the CUPL programs. The messages are listed by program name and module name within each program.

Appendix B: Device Usage Notes
Provides information about specific devices and how they require special attention with CUPL.

Appendix C: Download Formats
Describes the format of the CUPL downloadable files and the documentation file.

Appendix D: Node Numbering
Lists the devices which contain internal nodes supported by CUPL.

Appendix E: Trouble Shooting

Lists some common problems and their respective solutions.

Index
Is an index of the entire book.

Conventions Used In This Manual

This manual gives step-by-step procedures and examples. To make it easy to follow these procedures, the following conventions.are used.

Note

LDI software is not case sensitive. It doesn't matter whether upper or lower case.characters are typed.

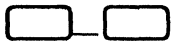
Screen icons indicate a prompt or response on the screen. For example:

```
Name of List Device?  PRN
```

Return

Return is the key that must be pressed to execute a command or accept an option. This key is called different names on different systems. For example:

Enter↵, Enter, ←┘, ENTER, RETURN



Connected keys indicate the keys must be pressed simultaneously. For example:

Press Ctrl — Alt — Del

\ A backward slash separates main menu and sub-menu commands. For example:

FILE\LOAD

It also begins a keyboard command. For example:

\EXE

* An asterisk in a filename indicates any characters can occupy that position and all remaining positions. For more information, refer to the DOS manual.

Select Select means use Space or the cursor keys to cycle through options or settings, and press Return to select the desired option or setting.

Boldface Boldface is used for two purposes. First, it is used to highlight menu or file names within text, and, second, it indicates characters that must be typed from the keyboard.These characters are usually designated as "Enter the following:" or are set aside by line spacing. For example:

del pcprint.cfg

Italics — Italics represent variable names. For example:

***filename*.SCH**

< > — When shown on the screen, variable names are indicated by angle brackets. For example:

<filename>.SCH

[ ] — Square brackets indicate the enclosed item is optional. For example:

prepack filename.fil [filename.lib]

When shown on the screen, square brackets indicate the name of a key. For example:

Press [Return] to accept

Introduction to CUPL U1

❑ INTRODUCTION

Welcome to CUPL Universal Compiler for Programmable Logic. CUPL is a very versatile and powerful logic compiler, and can be used to create very sophisticated logic designs for PLDs. This section will introduce the user to CUPL.

❑ CUPL OVERVIEW

The CUPL package consists of the following programs: **CUPL** (Universal Compiler for Programmable Logic), **CSIM** (CUPL Simulator), **CBLD** (CUPL Build), and **PTOC** (PALASM to CUPL Translator).

Using CUPL, logic descriptions may be written and compiled. The logic descriptions can be assigned to specific programmable logic devices (PLDs). CUPL searches its library and creates a file which can be downloaded to a device programmer and, from there, the PLD can be programmed.

CSIM permits a design to be simulated before it is put into production. A file is created, describing the expected functionality of the PLD in terms of input and output values. **CSIM** compares the expected values to actual values calculated during CUPL operation. The DOS version of **CSIM** will also allow the simulation to be viewed in waveform.

CBLD allows the manipulation of the device libraries containing descriptions of the PLDs supported by CUPL.

PTOC converts PALASM designs to CUPL format.

The key features of the CUPL package include:

- **Universal applicability.** CUPL supports products from all manufacturers of PLDs, enabling a user to put the same functional logic into physically different parts, to create a second source at the socket. CUPL produces a standard type of file called JEDEC. This is a download file that is compatible with any logic programmer that uses JEDEC files. The CUPL language translator (**PTOC**) converts PALASM designs to CUPL, enabling the use of any PALASM designs.

- **A high-level language.** Expression substitution for equations, shorthand notation for lists, address ranges, and bit fields are available to save design time.

  CUPL simplifies Boolean operations by the distributive property and DeMorgan's Theorem.

  State machine syntax provides a powerful means of implementing any synchronous application using either Mealy or Moore state machine models.

  Truth table syntax provides a way to clearly express certain logic descriptions.

  User-defined functions allow the creation of keywords for use by CUPL.

- **Flexible documentation.** CUPL provides a template file for standard "fill in the blanks" documentation and allows the placement of free-form comments throughout a design.

CUPL's comprehensive error-checking capability generates detailed error messages designed to lead to the source of any problems.

➢ **Powerful minimizer and simulation programs.** CUPL contains the fastest and most powerful minimizer offered for programmable logic equation reduction, featuring four levels of minimization.

The CUPL simulation program enables logic to be simulated prior to using a PLD. This feature prevents blown devices and helps debug system-level problems. Test vectors verified by **CSIM** can be downloaded to a logic programmer.

❑ CUPL DATA FLOW

This section describes the CUPL data flow.

First, a logic description source file is created, using the CUPL language to describe the logic that will be assigned to a programmable logic device.

Then, run **CUPL** to compile the source file to create a fusemap file for downloading to a device programmer. On execution of **CUPL**, the '**-a**' flag may be used to cause **CUPL** to generate an '**.ABS**' file for later use by **CSIM**.

Optionally, a test specification file may be created to verify the design. **CSIM** is executed to compare the expected values in the test file to the actual values in the absolute file created by **CUPL**. When simulation is complete without any errors, the verified test vectors can be appended to the download file generated by **CUPL**.

At this point, the verified fusemap file can be transferred to a device programmer.

Figure U1-1 shows the CUPL data flow.

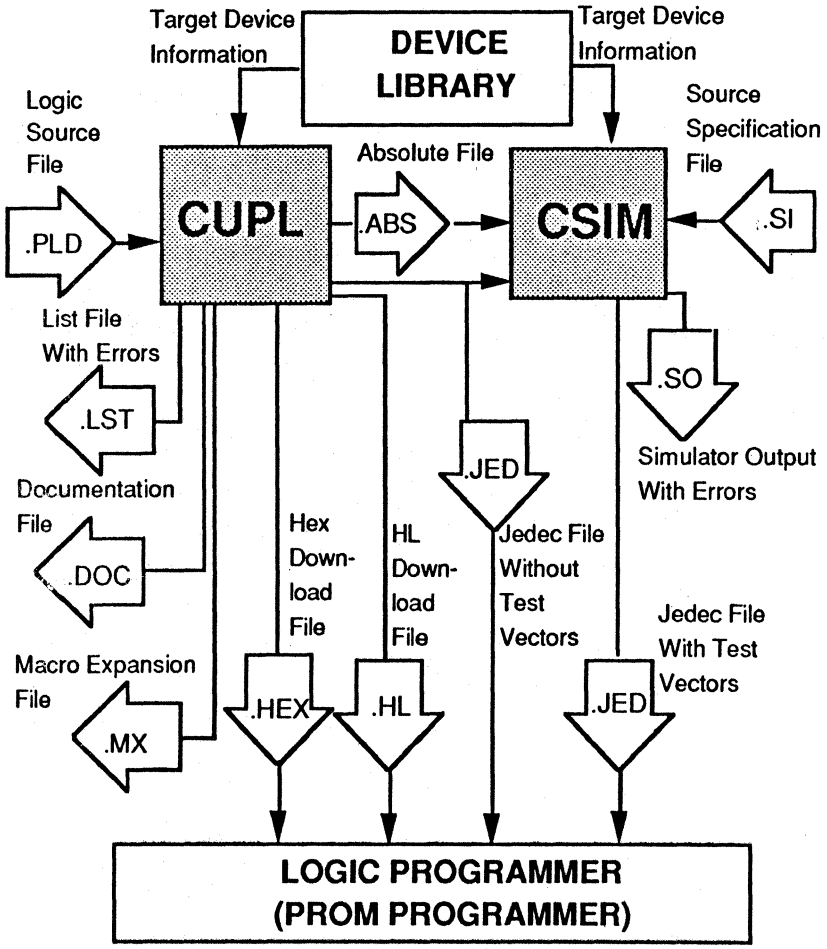


Figure U1-1. CUPL Data Flow

❑ INSTALLING CUPL

The following are the requirements and steps for installing CUPL on a system.

❑ MS-DOS INSTALLATION

The following are the minimum requirements in order to install and run CUPL properly:

- An IBM PC, PC/XT, or PC/AT (or equivalent) with at least 512K of RAM.
- PC-DOS version 2.0 or higher (or MS-DOS 2.0).
- A hard disk
- At least one double-sided floppy disk drive.
- Network operators need to be notified of the need for total access to the root directory and directory where CUPL will be installed. Users will need read access to the root directory and directory where CUPL will be installed.

Building the libraries requires additional hard disk space. Before running **build**, there should be an additional 1 Mb of free space on the hard disk. To build the SMT libraries, an additional .7 Mb hard disk space is required. To install CUPL

and all the libraries, therefore, requires 4.2Mb hard disk space. After building the libraries, the partial library files may be deleted and CUPL will occupy only 2.5 Mb of disk space.

Installing CUPL requires four steps.

- Run the **INSTALL** program contained on the diskette labelled "**DISK 1** ".
- Build the library.
- Set the 'Configuration' of the computer so that CUPL will have the setup that it needs.
- Set the 'Environment'of the computer so that the system will function properly.

The **INSTALL** program has been provided to quickly and easily install the CUPL software. It creates the appropriate directory and copies the original diskettes onto the hard drive.

To run the **INSTALL** program, follow the steps below:

1. Place the "**DISK 1**" diskette in drive A. Change the current drive to drive A by typing

   a: Return

2. Type

   install Return

3. Follow the prompts on the screen. The program suggests default values at each prompt; for

example, it supplies **CUPL** as the directory name. Press [Return] to accept the default name or type a different name and then press [Return].

4. CUPL needs to access its device library **CUPL.DL**. This file must be created on the hard disk. To do this, run **build**. This is a batch file which creates the library and puts the required information into it. A batch file is simply a text file which contains a series of DOS commands which can be executed by typing in the batch file name. Before running **build**, make sure that there is 1 Mb of disk space available.

   Type **build** [Return]

   If SMT parts are to be used, then an additional step is required to add these to the library. Before running **buildsmt**, make sure that there is .7 Mb free space on the disk.

   Type **buildsmt** [Return]

5. When a DOS computer starts up, it executes commands that are contained in a file called **AUTOEXEC.BAT** if it is present on the disk. Some additional commands must be added to the **AUTOEXEC.BAT** file for CUPL to function most efficiently. If there is no **AUTOEXEC.BAT** file on the computer, then create one and add the following information to it:

   path c:\cupl

   set LIBCUPL=c:\cupl\cupl.dl

   This lets the computer know where to find the CUPL Program and Library.

The computer may already have an **AUTOEXEC.BAT** file. In that case, it should be modified as follows.

- Add **CUPL** to the path command.

  If the current path command reads:

  path c:\dos;c:\bin

  Change it to:

  path c:\dos;c:\bin;c:\cupl

- Add this set command to the **AUTOEXEC.BAT** file:

  set LIBCUPL=c:\cupl\cupl.dl

This example assumes that CUPL is installed in a directory called **CUPL** on Drive C of the computer.

Specifying the Configuration

Each time the system is booted, the disk operating system (DOS) configures the system for a specified number of buffers and files. A buffer is a block of memory that DOS uses to hold data being read from or written to disk when the amount of data being transferred is not an exact multiple of the sector size. The number of files for which the system is configured determines how many files can be open at one time during program operation. A CONFIG.SYS file in the root directory instructs DOS on how to configure the system. If there is no CONFIG.SYS file, DOS sets default values for the configuration. CUPL requires 20 buffers and 15 files to ensure proper operation. Verify that the system has a CONFIG.SYS file in the root directory and that it contains the following two commands:

```
buffers=20
files=15
```

If the system does not have a CONFIG.SYS file, create one using the a editor in non-document mode. Type in the following:

```
BUFFERS=20
FILES=15
```

If the system has a CONFIG.SYS file, but the buffers are less than 20 or the files are less than 15, change the values to 20 and 15 respectively. If the CONFIG.SYS file on the system specifies buffers greater than 20 and files greater than 15, leave the file intact. Other programs that are running may require more buffers or files than CUPL does. If a CONFIG.SYS file has been created or altered, or if the AUTOEXEC.BAT file was changed according to the instructions in the previous section, press Control-Alternate-Delete to reboot the system and put the new commands into effect.

The CUPL Menu System

There are two ways to run the CUPL program. The program may be executed using the DOS command line, or the optional menu system may be used.

The CUPL menu system is called **MCUPL**. It is entirely menu driven and can be learned in a few minutes. This section will describe **MCUPL**.

Note: **MCUPL** is available for MS-DOS computers only.

Note: Although it may not be explicitly stated on the screens, the **Escape** key can be used to cancel a choice or to go back to the previous screen or menu.

MENU CONFIGURATION

The menu system uses information stored in a file called "**MCUPL.CFG**". This file permits the following information to be specified:

Machine ID: This can be set to "**PC**" or "**AT**". If using a computer with an 8088/8086 processor, then select "**PC**". If using an 80286 or better, then select "**AT**". The specification is done with the following syntax:

"set mid PC" or "set mid AT".

The editor being used must be specified. If this is not done, then **MCUPL** will not function properly. Specifying the editor is done as follows:

"set editor me"	if **"me.exe"** is the editor being used.
"set editor ws"	if **"ws.exe"** is the editor being used.

MCUPL uses a working directory where it reads and writes files. Any file created by the user or by CUPL should be stored here. Set the working directory with the following syntax: "**set wdpath *pathname***", where ***pathname*** is a directory path name.

Example: **"set wdpath c:\cupl\files".**

MCUPL.CFG contains specifications for screen colors of the menus. Use "**set color *area color***", where ***area*** is the area of the screen, such as **MCFG** (Message Center Text), **MCBG** (Message Center Background), **MCBC** (Message Center Border), **UIFG** (User Input Text), **UIBG** (User Input Background), **UIBC** (User Input Border), **MMFG** (Main Menu Text), **MMBC** (Main Menu Border) and **MMBG** (Main Menu Background).

The ***color*** attribute determines what color the area will have, such as Black, Blue, Green, Cyan, Red, Magenta, Brown, Yellow, Darkgrey, Lightgrey, Lightblue, Lightgreen, Lightcyan, Lightred, Lightmagenta and White. Example "**set**

color MCFG Lightblue" will make the message center text appear light blue.

The **function keys** F6 to F10 can be user-defined. Since the menus can be operated either by using the arrow keys or by typing the first key of the desired menu selection, the menus can be used normally or a function key can be defined to perform a specific action by having it type a series of keys. The following example defines F10 (F0) to quit **MCUPL** by typing 'Q' to select quit and then 'Y' to confirm the action.

Example: **"def F0 QY"**

Start the menu system as follows. Start up the computer. At the prompt, type:

mcupl Return

The startup screen will appear, followed by the **main menu** with 11 items on it.

* Edit Design Entry	Edit or convert a design file
* Compile CUPL file	Prompts for a **.PLD** file then compiles it.
* Look at DOC file	View the Documentation file.
* JEDEC file editor	View or edit a JEDEC file.
* Input simulation file (.SI)	Create or edit a simulation input file
* Simulate CUPL file **.SI** file.	Simulate the PLD design using the

* View simulation results View simulation output file (.SO).

* Device Selection Choose a device from a list, giving device type and manufacturer.

* Help The online quick reference guide for CUPL

* Tutorial for PLDs Executes the Programmable Logic User's guide demonstration.

* Quit Return to DOS.

Only one menu item can be highlighted at a time. While an item is highlighted, a description of the action performed is displayed in the message center window. To initiate the action associated with the menu item, press [Return]. The action may also be initiated by pressing the first letter of the menu item.

The keyboard can be used to select from the menu by typing the first letter of the menu item name. This has the same effect as using the arrow keys and pressing the return key for that item.

Some menu selections simply cause another menu to appear. Selecting items from sub-menus is the same as selecting from the main menu. To return to the previous menu, press the **ESC** key.

The EZ Edit Editor

The EZ Edit editor is the stand-alone editor used by MCUPL, unless a different editor is specified in **MCUPL.CFG**. EZ Edit is a simple text editor.

When using EZ Edit, the insert key is used to alternate between the insert and overwrite modes. The Alternate-X [Alt-X] key is used to

exit the editor. When exiting, the editor will ask for the filename to which changes should be saved (only if there were changes).

The following are the keystrokes and commands available when using the EZ Edit editor:

Key	Action
Up Arrow	Cursor up
Down Arrow	Cursor down
Left Arrow	Cursor left
Right Arrow	Cursor right
Home	Beginning of line
End	End of line
PgUp	Page up
PgDn	Page down
Ctrl-Home	Top of screen
Ctrl-End	Bottom of screen
Ctrl-PgUp	Top of file
Ctrl-PgDn	Bottom of file
Backspace	Deletes a character
Ctrl-Backspace	Deletes a line
Alt-X	Exits EZ Edit

❑ UNIX INSTALLATION

CUPL for the UNIX operating system is supplied on either 1/4" cartridge tape with **device = rst0** or 1/2" 9 track tape with **device = rmt0**. Both tapes are written in **TAR** format.

The following are required in order to run CUPL successfully on a UNIX System:

➢ CUPL is available in versions compatible with UNIX Version 7, System V and Berkeley 4.2 BSD. Make sure that the correct version is being used.

➢ At least 1000 (512 byte) blocks of disk space available.

➢ A text editor to create CUPL source files. Any text editor will work once it can create a standard ASCII text file.

➢ An RS-232 port for downloading fusemap files to the device programmer.

Installing the Software on Sun Workstation

CUPL requires SunOS release 4.0.3 or higher.

The program files should be placed in a directory which will be accessible to CUPL users. for example, the **/usr/bin** or **/usr/local** directories. A new directory **/usr/cupl** may be created, if having such a directory is more suitable.

If a new directory **/usr/cupl** is being created, then do the following:

mkdir /usr/cupl
chown root /usr/cupl
chmod 755 /usr/cupl

These make the directory **/usr/cupl** owned by **root**. It is read-write-execute by the owner and read-execute for all other groups and users.

Change the default directory to the new **/usr/cupl**.

cd /usr/cupl

Place the CUPL tape into the tape drive and put the drive online. Then start the installation by typing:

tar xv

Note: The above command assumes that the tape is loaded on the default device for **tar** (usually **/dev/rst0**). The command copies all the files from the tape to the current directory, therefore, the **cupl** directory should be set to the current directory. If the system uses a different tape device, consult the system administrator for assistance.

Remove the distribution tape and put it into a proper storage rack for safe keeping.

Setting Up the Environment

CUPL uses the environment to locate the device library file **CUPL.DL**. The environment is a series of names and parameters which is made available to all UNIX commands and application programs. The user may display the current contents at any time by typing the command:

printenv

The display might look something like this, assuming the user login name is "**designer**":

HOME=/usr/designer
PATH=.:/bin:/usr/bin

This shows the user that the "**home directory**" (the directory first entered after login) is the **/usr/designer** directory. The path to be searched for all application programs is the current directory specified by the period and then **/bin** and **/usr/bin** directories.

CUPL takes advantage of the hierarchical directory structure of UNIX so that the user can execute the compiler from a directory other than the one where the CUPL application is stored. In order for this to work, the user must set the UNIX path parameter in the environment to search for the executable files in the appropriate directories. The name of the directory where **CUPL** is stored should be added to the path parameter of each CUPL user.

CUPL searches the environment for the string name **LIBCUPL**, and uses the supplied parameter for the directory and filename of the default device library file. In order to properly access this file, the user must add the string **LIBCUPL** to the environment. If **CUPL** was installed in a directory called **/usr/cupl**, then use the following commands.

setenv PATH .:/bin:/usr/bin:/usr/cupl
setenv LIBCUPL /usr/cupl/cupl.dl

These new parameters are assigned during login or by the following command in the home directory:

source .cshrc

Note: The above procedure applies only when using the UNIX **C-shell** command interpreter.

For those installations running the **Bourne shell**, the following commands must be placed in the **.profile** file of the home directory:

```
PATH=.:/bin:/usr/bin:/usr/cupl
LIBCUPL=/usr/cupl/cupl.dl
export PATH LIBCUPL
```

These new parameters will take effect at login time or by executing the following command in the home directory:

```
source .profile
```

If it is not known which command interpreter is being used on the system, consult the system administrator.

Installing the Software on Apollo Workstation

CUPL requires DOMAIN/IX rev 9.7.5 or higher.

The program files should be placed in a directory which will be accessible to CUPL users, for example, the **/usr/bin** or **/usr/local** directories. A new directory **/usr/cupl** may be created if desired.

When creating a new directory /usr/cupl, do the following:

```
mkdir /usr/cupl
chown root /usr/cupl
chmod 755 /usr/cupl
```

These make the directory **/usr/cupl** owned by **root**. It is read-write-execute by the owner and read-execute for all other groups and users.

Change the default directory to the new **/usr/cupl**.

cd /usr/cupl

Place the CUPL tape into the tape drive and put the drive online.

If using cartridge tape, it must be retensioned before use. Type:

/com/rbak -dev c -reten

Then start the installation by typing:

tar xvf /dev/rct8

Note: The above command assumes that the tape is loaded on the device **rct8**. If the system uses a different tape device, consult the system administrator for assistance.

Remove the distribution tape and put it into a proper storage rack for safe keeping.

Setting Up the Environment

CUPL uses the environment to locate the device library file **CUPL.DL**. The environment is a series of names and parameters which is made available to all UNIX commands and application programs. The user may display the current contents at any time by typing the command:

printenv or

set

The display might look something like this, assuming the user login name is "**designer**":

HOME=/usr/designer
PATH=.:/bin:/usr/bin
TERM=apollo_15_color

This shows the user that the **home directory** (the directory first entered after login) is the **/usr/designer** directory. The path to be searched for all application programs is the current directory specified by the period and then **/bin** and **/usr/bin** directories.

CUPL takes advantage of the hierarchical directory structure of UNIX so that the user can execute the compiler from a directory other than the one where the **CUPL** application is stored. In order for this to work, the UNIX path parameter in the environment must be set to search for the executable files in the appropriate directories. The name of the directory where **CUPL** is stored should be added to the path parameter of each CUPL user.

CUPL searches the environment for the string name **LIBCUPL** and uses the supplied parameter for the directory and filename of the default device library file. In order to properly access this file, the user must add the string **LIBCUPL** to the environment. If **CUPL** was installed in a directory called **/usr/cupl**, then use the following commands:

setenv PATH (.:/bin:/usr/bin:/usr/cupl)
setenv LIBCUPL /usr/cupl/cupl.dl

These new parameters are assigned during login or by the following command in the home directory:

source .cshrc

Note: The above procedure applies only when using the UNIX **C-shell** command interpreter.

To use the **AEGIS** shell, the following statements must appear in the **user_data/sh/startup** file:

```
SETVAR LIBCUPL /usr/cupl/cupl.dl

export LIBCUPL

SETVAR PATH ((PATH:/usr/cupl))

csr ./bin: /usr/bin : /usr/cupl
```

For those installations running the **Bourne shell**, the following commands must be placed in the **.profile** file of the home directory:

```
PATH=.:/bin:/usr/bin:/usr/cupl
LIBCUPL=/usr/cupl/cupl.dl
export PATH LIBCUPL
```

These new parameters will take effect at login time or by executing the following command in the home directory:

```
source .profile
```

If it is not known which command interpreter is being used on the system, consult the system administrator.

❑ VAX/VMS INSTALLATION

The following are necessary to use CUPL on a VAX system:

➢ A VAX Computer capable of running VMS operating system 3.6 or later.

➢ A minimum of 512K of RAM.

➢ At least 6250 (512 byte) blocks of disk space available.

➢ A text editor to create CUPL source files. Any text editor will work once it can create a standard ASCII text file.

➢ An RS-232 port for downloading fusemap files to the device programmer.

Installing the Software

CUPL for the VAX is normally supplied on a nine-track magnetic tape, written at 1600 BPI density using the VAX/VMS **BACKUP** command.

All files are contained in a single save set called "**CUPL.BCK**". The program and device library files should be placed so that all CUPL users may have access. This may be the **SYS$SYSTEM** directory, or a new directory **USR_CUPL** may be created for **CUPL** only.

If CUPL is to be installed in **USR_CUPL** do the following:

Have the system administrator create the directory **USR_CUPL** by typing:

```
CREATE/DIR/PROT=(S:RWED,0:RWED,G:RE,
W:RE) DISK1: [USR_CUPL]
```

This creates the new directory on the device **DISK1** and gives full access privileges to the system and owner. The group and world categories are allowed only read and execute privileges. Change the name of the device to agree with the installation being used, as necessary.

Change the default to the new **USR_CUPL** directory:

```
SET/DEFAULT DISK1:[USR_CUPL]
```

Place the distribution tape into the system tape drive. Load the tape and put the tape **ON-LINE**. To read the tape onto the system, type the following:

```
MOUNT/FOR   MTA0
BACKUP/REW/LOG   MTA0:CUPL.BCK *.*
DISMOUNT   MTA0
```

Remove the distribution tape and put it into the proper storage rack for safe keeping.

Setting Up The Environment

CUPL uses logical names to locate the device library file (**CUPL.DL**). The file **Login.com** contains a series of names and parameters which are made available to all VAX commands and application programs. The user may display the current logical names at any time by using the command:

```
show logical/process
```

The display might look like this:

```
"SYS$CUPL" = "SYS$VAX1:[USR.DISTR]"
```

"SYS$LIBCUPL" = "SYS$CUPL:CUPL.DL"

Note: **SYS$LIBCUPL must be in uppercase letters.**

This shows how CUPL finds the device library. Within **Login.com** are other global variables that allow CUPL to be run from any directory or sub-directory within the login environment. The contents of **Login.com** must be added to the existing **Login.com** to be able to use CUPL properly. The disk and directory names in the **ASSIGN** statements should be changed to reflect the system that is being used.

This chapter assumes that the user has had some experience with programming devices. A basic understanding of logic and logic gates, and how they relate to the design of programmed devices, is required.

❑ GETTING STARTED

First Steps

Examine the Design Task

➢ Take a close look at the design that is needed. Remember that state-machine, Boolean equations, or truth tables are available for the design. Try to determine which type of syntax best suits the design task that is being worked on.

Create the CUPL Source File

➢ Use the template file provided and remove the sections that do not apply. Remember to edit the header to reflect the new file that is being created.

Formulate the Equations

➢ Equations must be written in CUPL syntax to specify the logic desired. This can be in Boolean, state-machine, or truth table format.

Choosing a Target Device

➢ Make sure that there is a sufficient number of input pins.

- Check that the number of registered and non-registered output pins is sufficient for the design.
- Ensure that the device has three-state output control.
- Check that the device can adequately handle the number of product terms required.

Making Pin Assignments

Assign the inputs and outputs of the design to the pins of the device. Make sure that the device is being used properly by consulting the reference material available from the manufacturer.

Running CUPL

Decide which file formats will be needed for downloading and simulation. A choice of four minimizers is available (**M1–M4**). CUPL will use **M1** minimization if none is specified. See Chapter 1 "CUPL Language" for details.

The device library which CUPL will use may be specified when the compiler is invoked. CUPL will use the library defined in the environment variable '**LIBCUPL**' if none is specified. See Chapter U2 "Installation" for more information on LIBCUPL.

Simple Logic Design

The Subway Turnstile controller is the simplest state machine design. This controller waits for a signal that a coin has been deposited. It then changes its state from locked to open. In the open phase, it waits for someone to walk through the turnstile, then it changes from open to locked. This two-state design cycles between open and locked using a coin detector and a walk-through detector as inputs. The following diagram shows the states and the pulses that change the device from one state to the next.

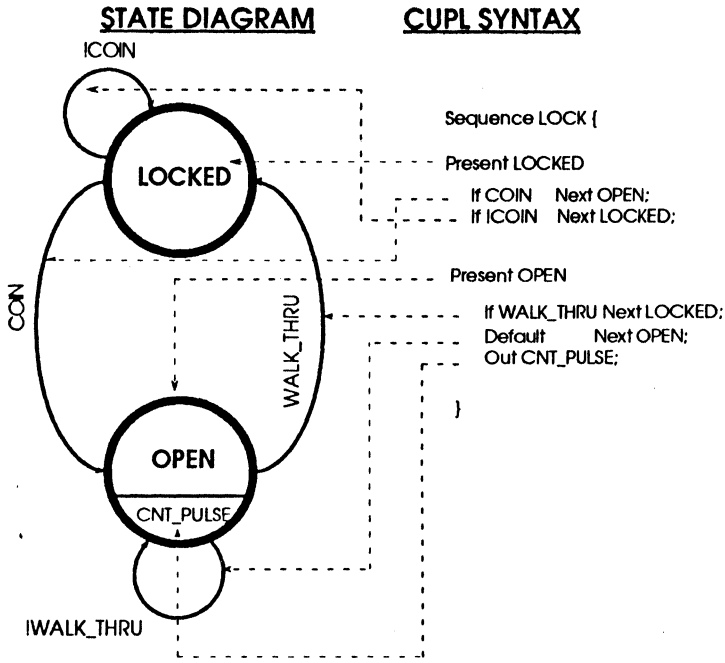


Figure U3-1 Subway Turnstile Example

The corresponding CUPL state machine code is displayed on the right, so that the relationship between the code and the design concept is easily understood.

```
                        TURNSTIL.PLD
Name            Turnstyl;
Partno          FL00001;
Date            03/06/89;
Designer        R. Teixeira;
Company         LDI;
Location        D21;
Assembly        Example

/*************************************************/
/*                                               */
/* This is an example to demonstrate how CUPL    */
/* compiles a subway turnstile controller        */
/*                                               */
/*************************************************/
/*   Target Devices: P16R4                       */
/*************************************************/

/* Inputs:   define inputs to Turnstile controller   */
Pin 1 = CLK;
Pin 2 = WALK_THRU;
Pin 3 = COIN;

/* Outputs:   define outputs as active HI levels

   ...............................                */

Pin 15 = LOCK;

/* Logic:  Subway Turnstile example expressed in CUPL */

Sequence  LOCK{

Present LOCKED
     if COIN      Next OPEN;
     if !COIN     Next LOCKED;

Present OPEN

     if WALK_THRU Next LOCKED;
     Default      Next OPEN;
     Out CNT_PULSE;

}
```

Figure U3-2 TURNSTIL.PLD

Simple Gates Example

This section will explain in detail the creation of a simple gates program for a PLD. The following diagram shows what design will be implemented. This design is taken from the Chapter U5 "Design Examples".

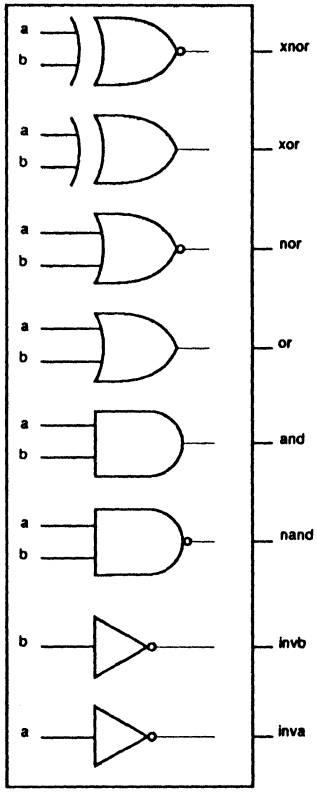


Figure U3-3 Simple Gates

This design gives a simple set of inputs and generates output simulating some simple gates. The outputs are labeled to reflect the function of their gate; for example, the AND gate has an output labeled AND.

Figure U3-4 shows the **CUPL** source file (**GATES.PLD** provided in the CUPL package) that describes the design.

```
                              GATES.PLD
Name            Gates;
Partno          CA0001;
Date            07/16/87;
Designer        G Woolheiser;
Company         ATI;
Location        San Jose, CA.;
Assembly        Example

/*************************************************/
/*                                               */
/* This is an example to demonstrate how CUPL    */
/* compiles simple gates                         */
/*                                               */
/*************************************************/
/*   Target Devices: P16L8, P16P8, EP300, and 82S153  */
/*************************************************/

/* Inputs:   define inputs to build simple gates      */
Pin 1 = a;
Pin 2 = b;

/* Outputs:   define outputs as active HI levels

   For PAL16L8 and PAL16LD8, De Morgan's Theorem is
   applied to invert all outputs due to fixed
   inverting buffer in the device.                    */

Pin 12 = inva;
Pin 13 = invb;
Pin 14 = and;
Pin 15 = nand;
Pin 16 = or;
Pin 17 = nor;
Pin 18 = xor;
Pin 19 = xnor;

/* Logic:   examples of simple gates expressed in CUPL */

inva = !a;              /* inverters              */
invb = !b;
and  = a & b;           /* and gate               */
nand = !(a & b);        /* nand gate              */
or   = a # b;           /* or gate                */
xor  = a $ b;           /* nor gate               */
xnor = !(a $ b);        /* exclusive nor gate     */
```

Figure U3-4. Simple Gates Source File (GATES.PLD)

The first part of the file provides archival information and a description of the intended function of the design, including compatible PLDs. First, there is the '**Name**' line, which **CUPL** uses to name the output files by adding extensions. '**Partno**' specifies the Company's proprietary part number, issued by manufacturing, for a particular PLD design. The part number is not the type of the target PLD. '**Date**' is used to specify the date of compilation. The date should be changed to the current date as a good documentation practice. '**Designer**' should be the designer's name or the name of the design team.

'Assembly' is used to specify the assembly name or number of the PC board on which the PLD will be used. Use the abbreviation **ASSY** if desired. **'Location'** is supposed to be used to indicate the PC board reference or coordinate where the PLD is located. The abbreviation **LOC** may also be used. This may be used for other purposes.

Pin declarations are made corresponding to the inputs and outputs in the design diagram. The gates in this example require two inputs, which are passed through the gates as necessary. 'a' and 'b' are names for the input pins. Next, names are assigned to the output pins. The names chosen are descriptive of the function being performed. The use of descriptive names is encouraged, as it makes files easier to debug and update at a later time.

In the **"Logic"** section of the file, equations describe each of the gates in the design. Boolean syntax is used to specify each output pin as a function of the input pins 'a' and 'b'.

For the PAL16L8 and PAL16LD8 devices, which contain fixed inverting buffers, **CUPL** applies DeMorgan's Theorem to invert all outputs because they were all declared active-HI in the pin list. For example, during compilation, **CUPL** converts the following equation for an OR gate, on an output pin that has been declared as active-HI:

```
or = a # b;
```

to the following single expanded product term (as shown in the documentation file):

```
or => !a & !b
```

The devices chosen for this design were selected because they fit the criteria, as specified earlier, for choosing a device. They have an adequate number of pins, both input and output. They have tri-state control. The number of registered and

non-registered pins fits our design, and the device can handle the number of product terms.

Compiling the Source File

At the system prompt, type:

cupl -x P16L8 gates

This will execute **CUPL**. The '**-x**' means to generate a documentation file (**GATES.DOC**) so that the expanded listing generated by CUPL may be viewed. The target device is a PAL16L8 and the source file is **GATES.PLD**.

After running **CUPL**, open the file **GATES.DOC**, which has the expanded listing generated by CUPL. This shows how CUPL expands the logic equations when it compiles the design for the device chosen.

In order to see how CUPL reports errors, edit the source file **GATES.PLD** and remove the semicolon at the end of one of the assignment statements. Now run **CUPL** as follows:

cupl -l P16L8 gates.pld

After it has finished, take a look at the file **GATES.LST** to see how CUPL reports errors. Notice that an error line appears next to the error, and there are line numbers at the beginning of each line.

Simulating a Design

Now the design will be simulated using **CSIM**. Test vectors must be created for the simulator to function. Test vectors specify the expected functional operation of a PLD by defining the outputs as a function of the inputs. Test vectors are also used to do functional testing of a device once it has been programmed, to see if the device functions as expected.

CSIM needs a test specification source file. For this example, it will be called **GATES.SI**. This file has a description of the function of the device in the circuit.

CSIM compares the input pin and output pin test values in the **GATES.SI** file, as shown in figure U3-5, with the actual values that are calculated from the logic equations in the absolute file **GATES.ABS** created by **CUPL** when the '**-a**' option is used. **CSIM** will not work unless the **GATES.ABS** file is created. To create the file **GATES.ABS**, type:

CUPL -aj p16l8 gates Return

```
Name            Gates;
Partno          000000;
Revision        03;
Date            9/12/83;
Designer        CUPL Engineering;
Company         Logical Devices, Inc.;
Location        None;
Assembly        None;

/*****************************************************************/
/*                                                               */
/*      This is a example to demonstrate how CUPL                */
/*      compiles simple gates.                                   */
/*                                                               */
/*****************************************************************/
/*      Taget Devices:  P16L8, P16LD8, P16P8, EP300, and 82S153  */
/*****************************************************************/


/*
 * Order:  define order, polarity, and output
 * spacing of stimulus and response values
 */

Order:  a, %2, b, %4, inva, %3, invb, %5, and, %8,
        nand, %7, or, %8, nor, %7, xor, %8, xnor;

/*
 * Vectors:  define stimulus and response values, with header
 *           and intermediate messages for the simulator listing.
 *
 * Note: Don't Care state (X) on inputs is reflected in outputs
 *       where appropriate.
 */

Vectors:

$msg "";
$msg "                              Simple Gates Simulation";
$msg "";
$msg "             inverters  and      nand     or       nor      xor       xnor";

$msg "      a  b   !a  !b   a & b  !(a & b)  a # b  !(a # b)  a $ b  !(a $ b)";
$msg "      -  -   --  --   -----  --------  -----  --------  -----  --------";

00 HHLHLHLH
01 HLLHHLHL
10 LHLHHLHL
11 LLHLHLLH
1X LXXXHLXX
X1 XLXXHLXX
0X HXLHXXXX
X0 XHLHXXXX
XX XXXXXXXX
```

Figure U3-5 Gates Simulator Input File (GATES.SI)

To run the simulator for the simple gates example, type the following:

CSIM -w P16L8 gates Return

The waveform output is available for the DOS version of CUPL only. For UNIX, VMS and all other versions, the only output

available is the simulator output file. Figure U3-6 shows the simulator output file. The inputs are listed with the corresponding outputs.

```
 1:Name           Gates;
 2:Partno         000000;
 3:Revision       03;
 4:Date           9/12/83;
 5:Designer       CUPL Engineering;
 6:Company        Logical Devices, Inc.;
 7:Location       None;
 8:Assembly       None;
 9:
10:/*************************************************************/
11:/*                                                           */
12:/*      This is a example to demonstrate how CUPL            */
13:/*      compiles simple gates.                               */
14:/*                                                           */
15:/*************************************************************/
16:/*      Taget Devices:  P16L8, P16LD8, P16P8, EP300, and 82S153 */
17:/*************************************************************/
18:
19:
20:/*
21: * Order:  define order, polarity, and output
22: * spacing of stimulus and response values
23: */
24:
25:Order:  a, %2, b, %4, inva, %3, invb, %5, and, %8,
26:        nand, %7, or, %8, nor, %7, xor, %8, xnor;
27:
28:/*
29: * Vectors:  define stimulus and response values, with header
30: *           and intermediate messages for the simulator listing.
31: *
32: * Note: Don't Care state (X) on inputs is reflected in outputs
33: *       where appropriate.
34: */
35:
```

Simulation Results

Simple Gates Simple Simulation

	a	a	inverters !a	!b	and a & b	namd !(a & b)	or a # b	nor !(a # b)	xor a $ b	xnor !(a $ b)
0001:	0	0	H	H	L	H	L	H	L	H
0002:	0	1	H	L	L	H	H	L	H	L
0003:	1	0	L	H	L	H	H	L	H	L
0004:	1	1	L	L	H	L	H	L	L	H
0005:	1	X	L	X	X	X	H	L	X	X
0006:	X	1	X	L	X	X	H	L	X	X
0007:	0	X	H	X	L	H	X	X	X	X
0008:	X	0	X	H	L	H	X	X	X	X
0009:	X	X	X	X	X	X	X	X	X	X

Figure U4-6 Gates Simulator Output File (GATES.SO)

CUPL Operation *U4*

This chapter explains how to use the various components of the CUPL package. It includes **CUPL, CSIM, CBLD** and **PTOC.**

❑ CUPL OPERATION

This section describes **CUPL** input and output and explains how to run **CUPL** using the command line options or the Menu system. This is a condensed version of the information given in the User Guide section "Using The CUPL System".

❑ Input

A logic description source file (*filename*.PLD) is the input to **CUPL.** This file describes the logical functionality to assign to a specified target device.

The source file is created using a standard text editor. There are a wide variety of text editors available and the choice depends entirely upon personal preference. The only requirement is that it be able to produce a standard text file.

The **CUPL** compiler must be able to access the device library file **(CUPL.DL)**, which contains a description of each of the target devices supported in the current version of CUPL. The library describes the physical characteristics of each device, including internal architecture, number of pins, and valid input and output pins, and also describes the logical characteristics, including registered and non-registered pins, number of product terms, fuse map information, and download format information.

The target device is referenced using device mnemonics. The mnemonic is composed of a device family prefix and industry standard part number suffix. Table U4-1 lists the device mnemonic prefixes.

Table U4-1. CUPL Device Mnemonic Prefixes

Symbol	Meaning
EP	Erasable Programmable Logic Device (EPLD)
G	Generic Array Logic (GAL)
F	Field Programmable Logic Array (FPLA)
F	Field Programmable Gate Array (FPGA)
F	Field Programmable Logic Sequencer (FPLS)
F	Field Programmable Sequence Generator (FPSG)
P	Programmable Logic Array (PAL)
P	Programmable Logic Device (PLD)
P	Programmable Electrically Erasable Logic (PEEL)
PLD	Pseudo Logical Device
RA	Bipolar Programmable Read Only Memory (PROM)

For example, the device mnemonic for a PAL10L8 is P10L8; for an 82S100 the device mnemonic is F100. For bipolar PROMs, the suffix is the array size; for example, the device mnemonic for a 1024 x 8 bipolar PROM is RA10P8, since there are 10 address input pins and 8 data output pins.

❑ Output

CUPL can output the files described below.

- **A JEDEC-compatible ASCII download file (*filename*.JED)** for input to a device programmer.
- **An ASCII Hex download file (*filename*.HEX)** available for PROMs.
- **An HL download file (*filename*.HL)** available for Signetics IFL devices.
- **An absolute file (*filename*.ABS)** for use by **CSIM**, the CUPL logic simulation program.
- **An error listing file (*filename*.LST)** that lists errors in the original source file.
- **A documentation file (*filename*.DOC)** that contains expanded logic equations, a variable symbol table, product term utilization, and fusemap information.
- **P-CAD PDIF file (*filename*.PDF)** that can be translated by PDIFIN into a PC-CAPS symbol representing the pinouts of the programmable logic device.
- **A Berkeley PLA file (*filename*.PLA)** for use by the Berkeley PLA tools.

❑ Running CUPL From the Command Line

Run **CUPL** using the following command line format:

cupl [-flags] [library] [device] source

where

-flags is the following set of compiler options:

-j	JEDEC download format
-h	ASCII-HEX download format
-i	HL download format
-n	use input filename for output file
-a	create absolute file
-l	create listing file
-e	create expanded macro definition file
-x	create expanded product-terms in documentation file
-f	create fuse plot/chip diagram in documentation file
-p	create PDIF database interchange format file
-b	create Berkeley PLA format file
-c	create PALASM format file
-d	deactivate unused OR terms
-r	disable product term merging
-g	program security fuse
-u	use specified library for compilation
-s	perform logic simulation after compilation
-w	perform simulation with waveform output (MS-DOS only)
-m0	no minimization
-m1	quick minimization (default)
-m2	Quine McCluskey
-m3	Presto
-m4	Expresso

library is the path name and library name used with the **-u** flag to specify a library other than the default library.

device is the device mnemonic for the type of part to be used in the compilation. Use the **CBLD** program to list available devices (see Chapter 3, "Using CBLD").

source is the user-created ASCII logic description file (***filename*.PLD**). The **.PLD** extension is assumed for the source file and may be omitted when giving the **CUPL** command.

Note

The square brackets indicate optional items.

Multiple option flags can be specified when running **CUPL**. A hyphen must be typed before the first flag entered, but is optional for additional flags. Spaces also can be put between the option flags. For example, the following two **CUPL** command lines are equivalent:

cupl -a -l -j p16r4 waitgen Return

cupl -alj p16r4 waitgen Return

Type **CUPL** without any flags to see the command line format and a list of the option flags. Table U4-2 lists descriptions of the **CUPL** option flags and output files. An introductory example will be presented in the next chapter.

Table U4-2. Compiler Option Flags

Option Flag	Description
j	Generates a JEDEC-compatible ASCII download file (**filename.JED**). The filename is not necessarily the same as the logic description filename input to **CUPL**. The **NAME** statement in the header information section of the logic description file determines the download filename (see the subtopic, Header Information in this chapter).
c	This can be used to create an input file for other logic design tools and gate array fitters such as PDS2XNF from XILINX.
h	Generates an ASCII-hex download file (**filename.HEX**). This format is available only for PROMs. The filename is not necessarily the same as the logic description filename input to **CUPL**. The **NAME** statement in the header information section of the logic description file determines the download filename (see the subtopic, Header Information in this chapter).
i	Generates an HL download file (**filename.HL**). This format is available only for the Signetics IFL devices. The filename is not necessarily the same as the logic description filename input to **CUPL**. The **NAME** statement in the header information section of the logic description file determines the download filename (see the subtopic, Header Information in this chapter).
a	Generates an absolute file (**filename.ABS**) for use by the **CSIM** logic simulation program.

n Allows the source filename to be used as the JEDEC filename instead of using the name in the **NAME** field of the source file.

l Generates an error listing file (**filename.LST**). Each line in the original source file is numbered. Error messages are listed at the end of the file and use the line numbers for reference.

x Generates a documentation file (**filename.DOC**) which contains an expanded listing of the logic terms in sum-of-products format and a symbol table of all variables used in the source file. It includes the total number of product terms and the number available for each output.

f Generates a fuse plot in the documentation file. For PAL devices, each output pin is listed and the associated product term rows are shown with the starting JEDEC fuse number. Fuses present are denoted with "**x**". Fuses blown are denoted with "**-**". For IFL devices, the HL download format is used, showing JEDEC fuse numbers with input terms denoted as "**H**," "**L**," "**0**," or "**-**".

p Generates a PDIF (P-CAD Database Interchange Format) file (**filename.PDF**) which can be translated by the **PDIFIN** program into a symbol for the PC-CAPS (P-CAD Schematic Capture) program. The generated symbol will contain packaging information for the PLD.

b Generates a Berkeley PLA file (**filename.PLA**) for use by the Berkeley PLA tools, such as PLEASURE, or other PLA layout tools which use the Berkeley PLA format.

d In IFL devices, the OR-gate output array is driven by each of the AND-gate product terms. Normally, unused OR-gate inputs are left connected to the product term array so that new terms may be added. However, with this option, the unused OR-gate inputs are removed (deactivated) from the product term array. The result is reduced propagation delay from input to output.

r In IFL devices, each product term from the AND- gate array may be shared among any number of OR- gate outputs. This option defeats this capability, forcing identical product terms to be generated for each output OR-array when required. The result is reduced propagation delay from input to output. This option will also force minimization to be performed on each output individually (as opposed to minimization on all outputs at once) when level **m2** or **m4** minimization is chosen.

g Adds the necessary code in the JEDEC download file to automatically allow the device programmer to blow the security fuse when programming. Not all programmers support this option.

u Overrides the default device library specified in the environment. Specify the complete path and filename for the library. Use this option on systems that may have special libraries created for unique or custom devices.

s Creates the absolute file and automatically runs the **CSIM** logic simulator. **CSIM** is run with the **-l** option that creates a list file. If the **-j** flag was specified for **CUPL**, it will be passed to **CSIM**, creating a JEDEC download file with test vectors.

e

Generates an expanded macro definition file (**filename.MX**) which contains an expanded listing of all macros used in the source file. It also contains the expanded expressions that use the **REPEAT** command.

w

(MS-DOS only) Creates the absolute file and automatically runs the **CSIM** logic simulator with waveform output. **CSIM** is executed with the **-w** option that displays the output in wave form.

m0

Defeats all logic minimization during a **CUPL** compilation. It is useful when working with PROMs, to keep contained product terms from being eliminated.

m1 - m4

CUPL provides four minimization levels: **-m1**, **-m2**, **-m3**, and **-m4**. The default minimization level is **m1**. Figure U4-1 shows the relative memory usage, speed, and efficiency of the four minimization levels. Minimization levels **m2** and **m4** will perform multiple output minimization in IFL devices. This maximizes product term sharing in these types of devices.

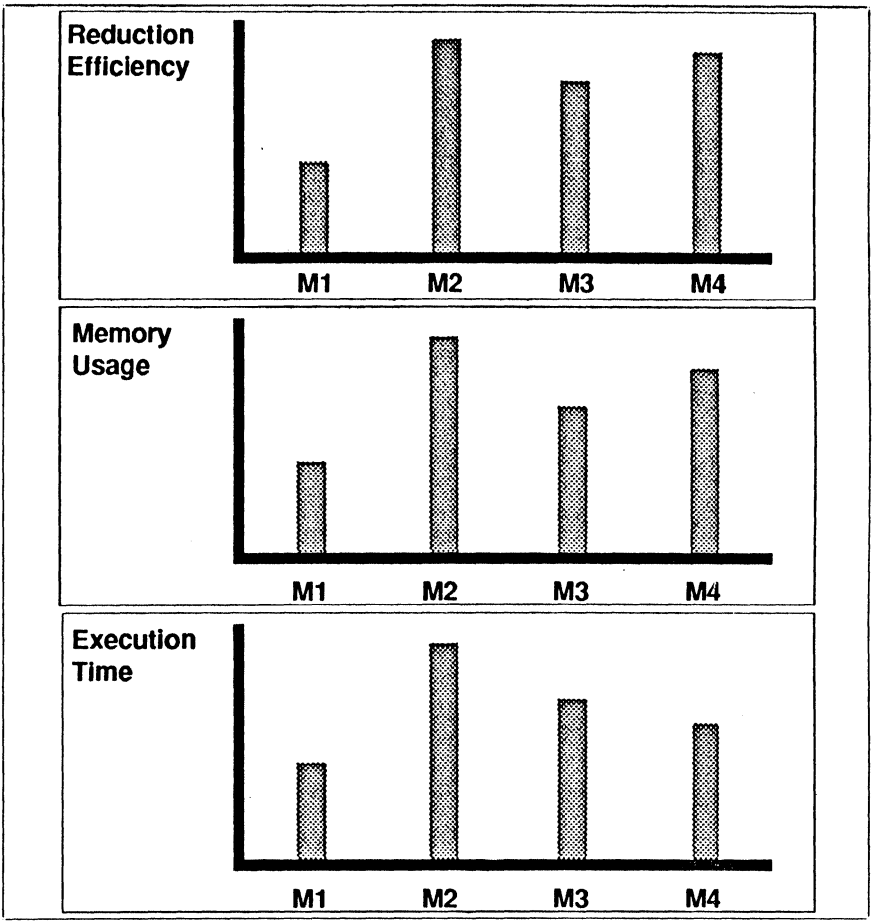


Figure U4-1. Logic Minimization Levels

Flag	Minimization Description
M1	Quick Minimization
M2	Quine-McCluskey Minimization
M3	Presto Minimization
M4	Expresso Minimization

❑ Running CUPL Using the Menus

The MS-DOS version of CUPL contains a front-end menu system. This can be used instead of the command line for working more quickly with CUPL. An explanation of the menu system follows.

The CUPL menu program is called **MCUPL**. It is a front-end program, meaning that it executes the other CUPL programs just as a user would, but the user never sees this step. All that is necessary is to select a choice from a menu and press return.

The Installation Chapter contains a description of how to install **MCUPL**. The **MCUPL** files **MCUPL.EXE** and **MCUPL.CFG** must be copied to the hard disk. **MCUPL.CFG** must be modified, so that it knows how the system has been set up. This is not difficult. **MCUPL.CFG** contains information such as the location of a working directory where work files are stored, what colors to use on the screen, the name of the editor being used and the class of computer that is being used. The installation chapter gives more information on this.

MCUPL is used by pressing keys on the keyboard to make selections from the menus. The arrow keys can be used to move up and down the menu. When the desired choice is selected, simply press the return key. This will either cause another menu to appear with more specific choices or it will cause an action to occur. It is also possible to press the first letter of the menu item to select it. For example, from the main menu, [**Q**] can be pressed to select the quit item. Type a '**Y**' to confirm the choice. This will quit the program back to DOS.

Often, an action will require input from the user. The message center will display a prompt detailing what information is required, such as a file name to be compiled. After entering the information, the user then presses the return key on the keyboard, and the action proceeds.

If a sub-menu is displayed, the escape key [**Esc**] can be used to display the previous menu.

The following is a description of the main menu of **MCUPL**.

* Edit Design Entry	Edit or convert a design file.
* Compile CUPL file	Prompts for a **.PLD** file then compiles it.
* Look at DOC file	View the Documentation file.
* Review error list file	View the error listing file.
* JEDEC file editor	View or edit a JEDEC file.
* Input simulation file	Create or edit the simulation input file (**.SI**).
* Simulate CUPL file	Simulate the PLD design using the **.SI** file.
* View simulation results	View simulation output file (**.SO**).
* Device Selection	Choose a device from a list giving device type and manufacturer.
* Help	The online quick reference guide for CUPL.
* Tutorial for PLDs	This executes the Programmable Logic User's guide demonstration.
* Quit	Return to DOS.

Defining Function keys

MCUPL allows for the custom definition of Function Keys F6 to F10. This involves modifying **MCUPL.CFG**. The function key definitions simply simulate the typing of a sequence of keys.

To define F6 to quit, add the following to **MCUPL.CFG**:

```
def F6 QY
```

This will type the letter '**Q**' then '**Y**' for the response to the confirm prompt.

Boolean Logic

Table U4-3 shows the Boolean Logic rules for eliminating excess product terms from the expanded equations, used by the logic reduction algorithms built into the **CUPL** compiler.

Table U4-3. Boolean Logic Rules

Expression		Result
!0	=	1
!1	=	0
A & 0	=	0
A & 1	=	A
A & A	=	A
A & !A	=	0
A # 0	=	A
A # 1	=	1
A # A	=	A
A # !A	=	1
A & (A # B)	=	A
A # (A & B)	=	A

Design Examples U5

This chapter provides examples of using **CUPL** and **CSIM**. It is divided into two parts.

Part A provides step-by-step instructions through a sample design session. Part B describes some of the designs that can be implemented with the logic description files provided with the CUPL package.

PART A. SAMPLE DESIGN SESSION

This part provides step-by-step instructions through a sample design session using **CUPL** and **CSIM**. The steps in the process are:

Step 1. Examining the Design Task
Step 2. Creating the **CUPL** Source File
Step 3. Formulating the Equations
Step 4. Choosing a Target Device
Step 5. Making the Pin Assignments
Step 6. Running **CUPL**
Step 7 Creating the **CSIM** Source File
Step 8 Running **CSIM**

❑ STEP 1. EXAMINING THE DESIGN TASK

The system in this programmable logic device (PLD) design example is microprocessor-based, with the CPU interfacing with ROM and RAM. Figure U5-1 shows a diagram of the system.

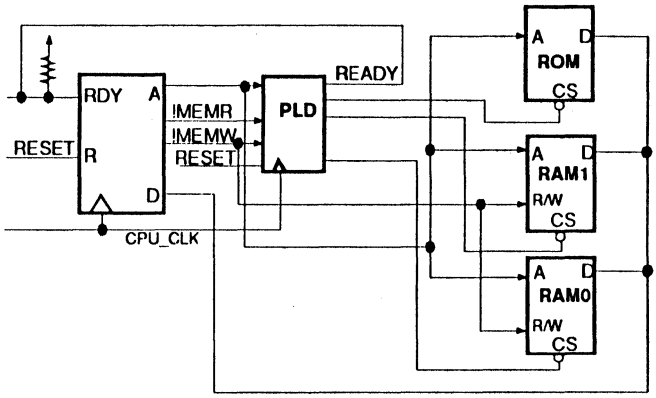


Figure U5-1. Microprocessor-Based System

A PLD provides a flexible interface between the CPU and peripherals by performing address decoding and timing control functions. As the diagram shows, a ROM (or PROM) is used for system control and two static RAMs are used for scratch pad and auxiliary memory functions.

In this sample session, a PLD will be designed that decodes the CPU's address using a memory map, and creates chip select signals for the ROM and RAM chips based upon CPU address and CPU data strobes.

The memory map in Figure U5-2 shows where the ROM and two RAM chips reside in the CPU's addressing space.

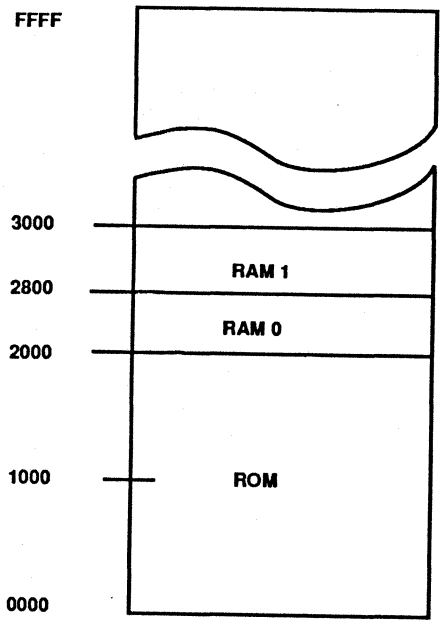


Figure U5-2. Memory Map

Addresses are marked and shown in hexadecimal in the memory map. Use this memory map when designing the logic for the PLD.

Because the ROM chip is slow, the PLD must be designed to perform a wait state generation that adds at least one CPU clock period to the ROM access. The worm arrows on the timing diagram in Figure U5-3 show signals affected or created by other signals.

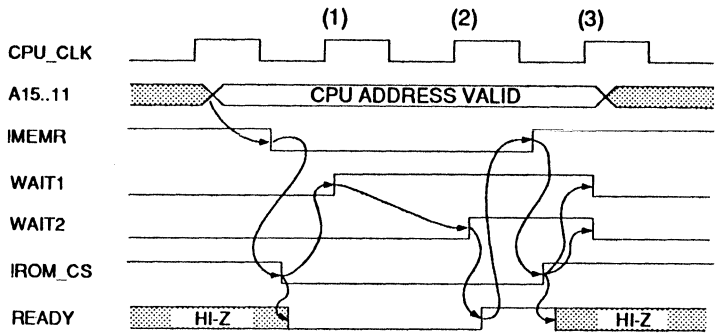


Figure U5-3. Wait State Generator Timing Diagram

A description of the operation of the timing diagram follows. The numbers in parentheses indicate the rising edge of the **CLOCK** signal.

A wait state sequence starts with the CPU address becoming valid prior to the memory read strobe. Only the **!MEMR** signal needs to be considered, because the wait state is generated only for the ROM.

When the **!MEMR** strobe is active for an address corresponding to the ROM, the **!ROM_CS** signal is asserted and turns on the three-state buffer, driving the **CPU READY** signal LO, (indicating not ready, or wait). The next rising edge of the CPU clock (1) after **!ROM_CS** becomes active and sets the **WAIT1** signal. After one CPU clock period passes, the **WAIT2** signal is asserted (2); the wait state period (one CPU clock) is completed, causing the **CPU READY** signal to be driven HI, which causes the CPU to continue its read cycle and remove the **!MEMR** strobe at the appropriate time. The **!ROM_CS** signal is negated, disabling the three-state buffer driving the ready signal and, at the next rising edge of the CPU clock (3), causing **WAIT1** and **WAIT2** to be reset. The

wait state generator is now prepared for the next CPU access time.

❑ STEP 2. CREATING THE CUPL SOURCE FILE

In this step, a logic description file will be created to describe the design for the PLD. The logic description file serves as input to **CUPL**, which compiles the design for downloading to a device programmer.

To make it easy to set up the required format for the logic description file, **CUPL** provides a template file, **TMPL.PLD**, that can be copied into the file being used. First, choose a name for the file that reflects the use being designed for the PLD. Since this is a sample session, use the name **SAMPLE.PLD**. Copy **TMPL.PLD** to **SAMPLE.PLD**, by typing:

copy tmpl.pld sample.pld Return

Note

To move more quickly through this design example, it is not necessary to actually create and edit the **SAMPLE.PLD** file. The CUPL package provides a sample file, **WAITGEN.PLD**, that can be used instead.

Use a text editor in non-document mode to open **SAMPLE.PLD**. Figure U5-4 shows the template information that has been copied into the file.

```
                        TEMPLATE FILE

Name                    XXXXX;
Partno                  XXXXX;
Date                    XX/XX/XX;
Revision                XX;
Designer                XXXXX;
Company                 XXXXX;
Assembly                XXXXX;
Location                XXXXX;

/*************************************************************/
/*                                                           */
/*                                                           */
/*                                                           */
/*************************************************************/
/*  Allowable Target Device                                  */
/*************************************************************/

/**   Inputs  **/

Pin         -           ;       /*                            */
Pin         -           ;       /*                            */
Pin         -           ;       /*                            */
Pin         -           ;       /*                            */
Pin         -           ;       /*                            */
Pin         -           ;       /*                            */
Pin         -           ;       /*                            */
Pin         -           ;       /*                            */
Pin         -           ;       /*                            */
Pin         -           ;       /*                            */
Pin         -           ;       /*                            */
Pin         -           ;       /*                            */
Pin         -           ;       /*                            */
Pin         -           ;       /*                            */

/**  Outputs  **/
Pin         -           ;       /*                            */
Pin         -           ;       /*                            */
Pin         -           ;       /*                            */
Pin         -           ;       /*                            */
Pin         -           ;       /*                            */
Pin         -           ;       /*                            */
Pin         -           ;       /*                            */
Pin         -           ;       /*                            */

/**   Declarations and Intermediate Variable Definitions  **/
```

Figure U5-4. SAMPLE.PLD Template Information

The file can be edited, in order to enter specific header and title information, specify the input and output pins, and write the intermediate and logic equations.

In the header section, replace the XXXs with specific information referring to the company and the PLD being designed. Since this is a sample design, use the information provided (as shown in Figure U5-5) or any other desired information.

Below the header section is a title block with comment symbols (/* and */). In the title block, type in information describing the design, as shown in Figure U5-5.

```
                        SAMPLE.PLD

Name                    Sample;
Partno                  P9000183;
Date                    07/16/87;
Revision                02;
Designer                Osann;
Company                 ATI;
Assembly                PC Memory;
Location                U106;


/*******************************************************/
/* This device generates chip select signals for one  */
/* 8Kx8 ROM and two 2Kx8 static RAMs. It also drives  */
/* the system READY line to insert a wait-state of at */
/* least one cpu clock for ROM accesses               */
/*******************************************************/
```

Figure U5-5. SAMPLE.PLD Header and Title Block

❑ STEP 3. FORMULATING THE EQUATIONS

To make it easier to enter the specific equations for address decoding and wait state generation, first enter equations for intermediate variables. Intermediate variables are arbitrary names; that is, they do not represent specific pins. Enter the intermediate equations in the space provided in the **SAMPLE.PLD** file for "Declarations and Intermediate Variable Definitions."

The first intermediate equation to enter is a bit field declaration to define the address bus. Use the name **MEMADR** (memory address) to represent the address, and type the equation as follows:

FIELD MEMADR = [A15..11] ;

In the system diagram in Figure U5-1, notice that the chip select signals for the static RAMs are not dependent solely

upon address but must be asserted for either the **MEMW** or **MEMR** data strobes.

To simplify the equations for the static RAM chip select signals, create a signal called **MEMREQ** (memory request). Type the following:

```
MEMREQ = MEMW # MEMR ;
```

Whenever **MEMREQ** is used in other equations, CUPL substitutes **MEMW # MEMR** when it compiles.

Notice in the timing diagram in Figure U5-3 that the decoding of the address corresponding to the ROM combines with the **!MEMR** strobe to produce the ROM chip select (**ROM_CS**), and to initiate the wait state sequence.

Create an intermediate variable, called **SELECT_ROM**, representing the combination of the **!MEMR** strobe and the specific address decoding for the ROM's address space, by typing the following:

```
SELECT_ROM = MEMR & MEMADR :
    [0000..1FFF] ;
```

After entering the above intermediate equations, the specific equations for address decoding and wait state generation may be entered.

If the signal **ROM_CS**, which feeds back into the array, is being used to initiate the wait state timing, an additional pass delay is incurred through the PLD. Because the clock rate is relatively slow (4-8 MHz), in this example the additional delay is not a problem. However, at higher clock rates it is better to recreate the same logic (using the **SELECT_ROM** intermediate) in the registered logic equations.

Create the ROM chip select (**ROM_CS**) using the intermediate variable **SELECT_ROM**, by typing:

```
ROM_CS = SELECT_ROM;
```

The chip-selects for the two RAMs, **RAM_CS0** and **RAM_CS1**, are dependent on **MEMREQ** and the address bus being within the hexadecimal boundaries taken from the memory map. Use the CUPL range operation with the lower and upper boundaries of the desired address range to decode these signals. Type the following:

```
RAM_CS0 = MEMREQ & MEMADR :
    [2000..27FF];

RAM_CS1 = MEMREQ & MEMADR :
    [2800..2FFF];
```

Next, create the equations that relate to the wait state timing and generation. First, as shown in the timing diagram (Figure U5- 3), a signal called **WAIT1** is required that responds to the selection of the ROM chip by being set at the next rising edge of the CPU clock. According to the rules for a D-type flip-flop, the logic level at the D input is transferred to the Q output after the clock. Enter the equation for this signal, where **WAIT.D** represents the signal at the D input of the flip-flop within the PLD, by typing the following:

```
WAIT.D = SELECT_ROM & !RESET ;
```

Notice that in the equation for **WAIT1.D**, the **!RESET** signal has been ANDed with the rest of the equation to perform a synchronous reset when the **RESET** signal is asserted.

Next, create the signal **WAIT2** at the next clock edge following the one that causes **WAIT1** to set, by making the equation for **WAIT2.D** dependent on the signal **WAIT1**. Since **WAIT2.D** must reset at the next clock edge following the removal of the CPU's access of the ROM, AND the variable, and then **SELECT_ROM** into this equation by typing the following:

```
WAIT2.D = SELECT_ROM & WAIT1 ;
```

This creates the signal **SELECT_ROM** in accordance with the timing diagram (Figure U5-3) to indicate that the three-state buffer should be turned on while the ROM is being decoded and the **MEMR** data strobe is active. Therefore, enter the equation for the three-state output by typing the following:

READY.OE = SELECT_ROM ;

While this equation determines when the three-state buffer actually drives its output and leaves the high impedance state, it does not determine which logic level the signal is driven to. The equation for **READY** determines the logic level to which the signal is driven; the signal should remain inactive at **READY** until the completion of a wait state period equal to one full CPU clock cycle. As this condition does not occur until **WAIT2** becomes set, type the equation for **READY** as follows:

READY = WAIT2 ;

❑ STEP 4. CHOOSING A TARGET DEVICE

After the equations are completed, the next step is to identify a compatible, commercially available PLD. Points to consider when choosing a target device are:

- The number of input pins required.
- The relative number of registered and non-registered output pins.
- Three-state output control.
- The number of product terms required to implement the logic function for each equation.

The PLD package diagram in Figure U5-6 shows pin assignments configured to match up with a device similar to a PAL16R4 or an 82S155 IFL.

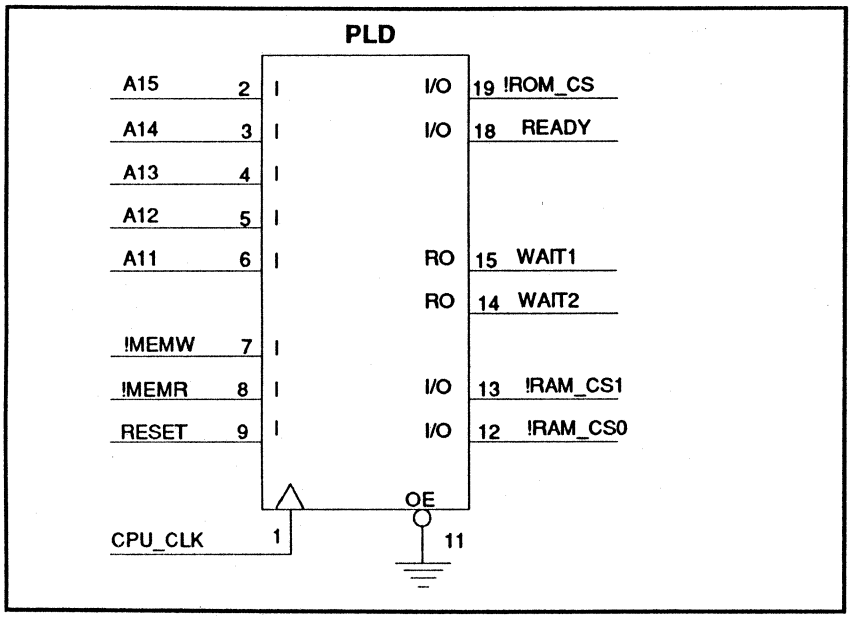


Figure U5-6. Sample Pin Configuration

In the pin configuration in Figure U5-6, the three chip select signals are assigned to I/O type pins that should always be in the output drive mode. The **READY** pin, attached to the **READY** signal on the CPU bus, is used in a controllable three-state mode. The two flip-flops that are needed to implement the wait state generator have been assigned to output pins that are internally connected to registers.

One of the registered outputs could be used to drive the **READY** signal directly, since the logical function of **READY** is the same as that of the signal **WAIT2**. However, use of the dedicated three-state output enable signal connected to pin 11 of the target device would be required. Since pin 11 controls the

three state outputs of all four pins connected to internal registers, this defeats the ability to use the other two registered output pins for any purpose other than wait state generation.

It is better to keep options open by not using the dedicated three-state control, since it is difficult to predict all the changes that might be made during the evolution of a design. Therefore, pin 11 is tied to ground, which always enables the three-state outputs coming from the registers.

The PAL16R4 has at least seven product terms available on all outputs, which is an adequate number for this application. The IFL 82S155, which is a second source for this socket position, has a total of thirty-two product terms available for all outputs combined, which is also an adequate number for this application.

The PAL16R4 devices have only D-type flip-flops, whereas the 82S155 devices may be configured for either D or JK types. **CUPL** automatically chooses a D-type flip flop configuration because the equations entered for **WAIT1** and **WAIT2** in step 3 specified the **.D** extension.

❑ STEP 5. MAKING THE PIN ASSIGNMENTS

Match the pin assignments to the pins in Figure U5-6 for a PAL16R4 or 82S155 device. First, in **SAMPLE.PLD** in the comment space labeled "Allowable Target Device Types," type:

pal16r4, 82s155 Return

To ensure consistent documentation when making the pin assignments, be certain that the signal polarities (signal levels) assigned are the same as those in the logic schematic (see Figure U5-1). Make the pin assignments as shown in Figure U5- 7.

```
                    SAMPLE PIN ASSIGNMENTS

/**   Inputs  **/

Pin 1        = cpu_clk    ;       /*  CPU clock                    */
Pin [2..6]   = [a15..11] ;       /*  CPU Address Bus              */
Pin [7,8]    = ![memw,memr]  ;   /*  Memory Data Strobes          */
Pin 9        = reset      ;      /*  System Reset                 */
Pin 11       = !oe        ;      /* Output Enable                 */

/**  Outputs  **/
Pin 19       = !rom_cs    ;       /* ROM Chip select              */
Pin 18       = ready      ;       /* CPU Ready signal             */
Pin 15       = wait1      ;       /* Start wait state             */
Pin 14       = wait2      ;       /* End wait state               */
Pin [13,12] = ![ram_cs1..0]  ;    /* RAM Chip selects             */
```

Figure U5-7. SAMPLE.PLD Pin Assignments

After making all the pin assignments, delete the extra "**pin = ;**" lines provided by the template file.

Figure U5-8 shows the completed logic description file, **SAMPLE.PLD**.

```
                        SAMPLE.PLD

Name                    Sample;
Partno                  P9000183;
Date                    07/16/87;
Revision                02;
Designer                Osann;
Company                 ATI;
Assembly                PC Memory;
Location                U106;

/*****************************************************/
/* This device generates chip select signals for one  */
/* 8Kx8 ROM and two 2Kx8 static RAMs. It also drives  */
/* the system READY line to insert a wait-state of at */
/* least one cpu clock for ROM accesses               */
/*****************************************************/
/*****************************************************/
/** Allowable Target Device Types : PAL16R4, 82S155  **/
/*****************************************************/
/**   Inputs  **/

Pin 1       = cpu_clk   ;       /*  CPU clock                 */
Pin [2..6]  = [a15..11] ;       /*  CPU Address Bus           */
Pin [7,8]   = ![memw,memr]  ;   /*  Memory Data Strobes       */
Pin 9       = reset     ;       /*  System Reset              */
Pin 11      = !oe       ;       /* Output Enable              */

/**  Outputs  **/

Pin 19      = !rom_cs     ;     /*                            */
Pin 18      = ready       ;     /*                            */
Pin 15      = wait1       ;     /*                            */
Pin 14      = wait2       ;     /*                            */
Pin [13,12] = ![ram_cs1..0] ;   /*                            */

/** Declarations and Intermediate Variable Definitions      */

Field memadr = [a15..11]  ;     /* Give the address bus       */
                                /* the Name "memadr"          */

memreq = memw # memr ;          /* Create the intermediate    */
                                /* variable "memreq"          */
select_rom = memr & memadr:[0000..1FFF] ;     /*  = rom_cs    */

/**  Logic Equations  **/

rom_cs = select_rom;
ram_cs0 = memreq & memadr:[2000..27FF]  ;
ram_cs1 = memreq & memadr:[2800..2FFF]  ;

wait1.d = select_rom & !reset  ;
                                /* Synchronous Reset          */
wait2.d = select_rom & wait1 ;  /* wait1 delayed               */

ready.oe = select_rom  ;   /* Turn Buffer off */

ready = wait2  ; /* end wait */
```

Figure U5-8. SAMPLE.PLD File

❑ STEP 6. RUNNING CUPL

When running **CUPL**, specify the target PLD, the source logic description file, and option flags to enable specific output files. In this step, compile the logic description file **SAMPLE.PLD**

for the target device PAL16R4, and create the following output files:

SAMPLE.ABS (-a flag) - This is the absolute file for later use by **CSIM**, the **CUPL** simulator (This file is needed for step 7). It contains a condensed representation of the logical function to be programmed into a device. **CSIM** compares this representation to test vectors in a user-created input file to determine whether the response vectors in the input file are a correct response to the stimulus vectors.

SAMPLE.DOC (-x, and -f flags) - This is the documentation file. It provides fully expanded product terms for both intermediate and output pin variables, and a fuse plot and chip diagram.

SAMPLE.LST (-l flag) - This is the list file. It is a recreation of the description file, except line numbers have been added and any error messages generated during compilation are appended at the end of the file.

SAMPLE.JED (-j flag) - This is a JEDEC file for downloading to a device programmer. It contains a fuse pattern but no test vectors.

Note

The **SAMPLE.JED** filename is determined by the **NAME** field in the header information section of the logic description file. When only one device is described in the file, be certain to use the same name (in this case, SAMPLE) as the filename.

To begin to compile and create the files described above, type the following command line:

cupl -jaxfl p16r4 sample Return

Note

If **SAMPLE.PLD** has not been created, type WAITGEN instead of SAMPLE to specify the sample file, **WAITGEN.PLD**, provided by CUPL. The filename for all output files created by CUPL is WAITGEN instead of SAMPLE.

The following messages appear on the screen, indicating how much time each CUPL module takes for completion. The actual time will vary depending on the system being used.

```
CUPL: Universal Compiler for Programmable Logic
Version 3.XX Serial # XX-XXX-XXXX
Copyright (C) 1983, 1990 Logical Devices, Inc.

cuplx
time:   2 secs
cupla
time    2 secs
cuplb
time:   2 secs
cuplm
time:   1 secs
cuplc
time:   5 secs
total time:      12 secs
```

When the prompt appears, compilation is complete. **SAMPLE.LST** and **SAMPLE.DOC** are ASCII files, so it is possible to display them on the screen, open them with a text editing program, or print a hard copy of their contents.

The list file, **SAMPLE.LST**, is essentially a recreation of the source file with line numbers inserted and any error messages attached to the end. The line numbers facilitate the quick locating of error sources, if any are detected by **CUPL**.

Figure U5-9 shows the contents of **SAMPLE.LST**.

```
                        SAMPLE.LST

CUPL  Version 3.XX  Serial # XX-XXX-XXXX
Copyright (C) 1983,1990 Logical Devices, Inc.
CREATED Thur Jan 14 08:42:12 1990

LISTING FOR LOGIC DESCRIPTION FILE: sample.pld;

1:Name                  Sample;
2:Partno                P9000103;
3:Date                  07/16/87;
4:Revision              02;
5:Designer              Osann;
6:Company               ATI;
7:Assembly              PC Memory;
8:Location              U106;
9:
10:/*************************************************/
11:/* This device generates chip select signals for one  */
12:/* 8Kx8 ROM and two 2Kx8 static RAMs. It also drives  */
13:/* the system READY line to insert a wait-state of at */
14:/* least one cpu clock for ROM accesses               */
15:/*************************************************/
16:/*************************************************/
17:/** Allowable Target Device Types : PAL16R4, 82S155  **/
18:/*************************************************/
19:/**   Inputs  **/
20:
21:Pin 1       = cpu_clk   ;      /*  CPU clock                  */
22:Pin [2..6]  = [a15..11] ;      /*  CPU Address Bus            */
23:Pin [7,8]   = ![memw,memr]  ;  /*  Memory Data Strobes        */
24:Pin 9       = reset     ;      /*  System Reset               */
25:Pin 11      = !oe       ;      /* Output Enable               */
26:
27:/**  Outputs  **/
28:
29:Pin 19      = !rom_cs    ;     /*                             */
30:Pin 18      = ready      ;     /*                             */
31:Pin 15      = wait1      ;     /*                             */
32:Pin 14      = wait2      ;     /*                             */
33:Pin [13,12] = ![ram_cs1..0]  ;   /*                          */
34:
35:/** Declarations and Intermediate Variable Definitions      */
36:
37:Field memadr = [a15..11]  ;     /* Give the address bus       */
38:                               /* the Name "memadr"          */
39:
40:memreq = memw # memr ;          /* Create the intermediate    */
41:                               /* variable "memreq"          */
42:
43:select_rom = memr & memadr:[0000..1FFF] ;      /*  = rom_cs     */
44:
45:/**  Logic Equations  **/
46:
47:rom_cs = select_rom;
48:ram_cs0 = memreq & memadr:[2000..27FF]  ;
49:ram_cs1 = memreq & memadr:[2800..2FFF]  ;
50:wait1.d = select_rom & !reset  ;
51:                              /* Synchronous Reset          */
52:wait2.d = select_rom & wait1 ;   /* wait1 delayed               */
53:ready.oe = select_rom  ;   /* Turn Buffer off */
54:ready = wait2  ; /* end wait */

Jedec Fuse Checksum       (4D50)
Jedec Transmit Checksum   (E88F)
```

Figure U5-9. SAMPLE.LST

Figure U5-10 shows the documentation file, **SAMPLE.DOC**, created by **CUPL**.

```
                                    SAMPLE.DOC
*******************************************************************************
                                    Sample
*******************************************************************************

CUPL                    4.XX Serial# XX-XXX-XXXX
Device                  p16r4 Library DLIB-d-26-11
Created                 Mon Aug 20 10:48:32 1990
Name                    Sample;
Partno                  P9000183;
Date                    04/1/90;
Revision                02;
Designer                Osann;
Company                 ATI;
Assembly                PC Memory;
Location                U106;

-------------------------------------------------------------------------------
                        Expanded Product Terms
-------------------------------------------------------------------------------

wait1.d  =>
     !memr
     # a15
     # a14
     # a13
     # reset

select_rom =>
       !a13 & !a14 & !a15 & memr

wait2.d  =>
     !memr
     # a15
     # a14
     # a13
     !wait

memadr  =>
       a15,a14,a13,a12,a11

ready  =>
       !wait2

ready.oe  =>
       !a13 & !a14 & !a15 & memr
rom_cs  =>
       !a13 & !a14 & !a15 & memr
```

Figure U5-10. SAMPLE.DOC Sheet 1 of 4

```
memreq  =>
   memw
   # memr

ram_cs0  =>
   !a11 & !a12 & !a13 & !a14 & !a15 & memw
   # !a11 & !a12 & !a13 & !a14 & !a15 & memr

ram_cs1  =>
   a11 & !a12 & a13 & !a14 & !a15 & memw
   # a11 & !a12 & a13 & !a14 & !a15 % memr

rom_cs.oe  =>
    1

ram_cs0.oe  =>
    1

ram_cs1.oe  =>
    1

--------------------------------------------------------------------------------
                        Symbol Table
--------------------------------------------------------------------------------
Pin Variable                                                  Pterms    Max     Min
Pol   Name                      Ext      Pin      Type        Used     Pterms  Level
--- --------                    ---      ---      ----        ------   ------  ------
    wait1                               15       V             -        -       -
    wait1                        d      15       X             5        8       1
    a11                                 6        V             -        -       -
    select_rom                          0        I             1        -       -
    wait2                               14       V             -        -       -
    wait2                        d      14       X             5        8       1
    a12                                 5        V             -        -       -
    a13                                 4        V             -        -       -
    a14                                 3        V             -        -       -
    a15                                 2        V             -        -       -
 !  oe                                  11       V             -        -       -
 !  memr                                8        V             -        -       -
    memadr                              0        F             -        -       -
    ready                               18       V             1        7       1
    ready                        oe     18       X             1        1       1
 !  memw                                7        V             -        -       -
    cpu_clk                             1        V             -        -       -
 !  rom_cs                              19       V             1        7       1
    reset                               9        V             -        -       -
    memreq                              0        I             2        -       -
 !  ram_cs0                             12       V             2        7       1
 !  ram_cs1                             13       V             2        7       1
    rom_cs                       oe     19       D             1        1       0
    ram_cs0                      oe     12       D             1        1       0
    ram_cs1                      oe     13       D             1        1       0

LEGEND      F : field      D : default                  M : extended node
            N : node       I : Intermediate variable    T : function
            V : variable   X : extended variable        U : undefined
```

Figure -10. Sheet 2 of 4

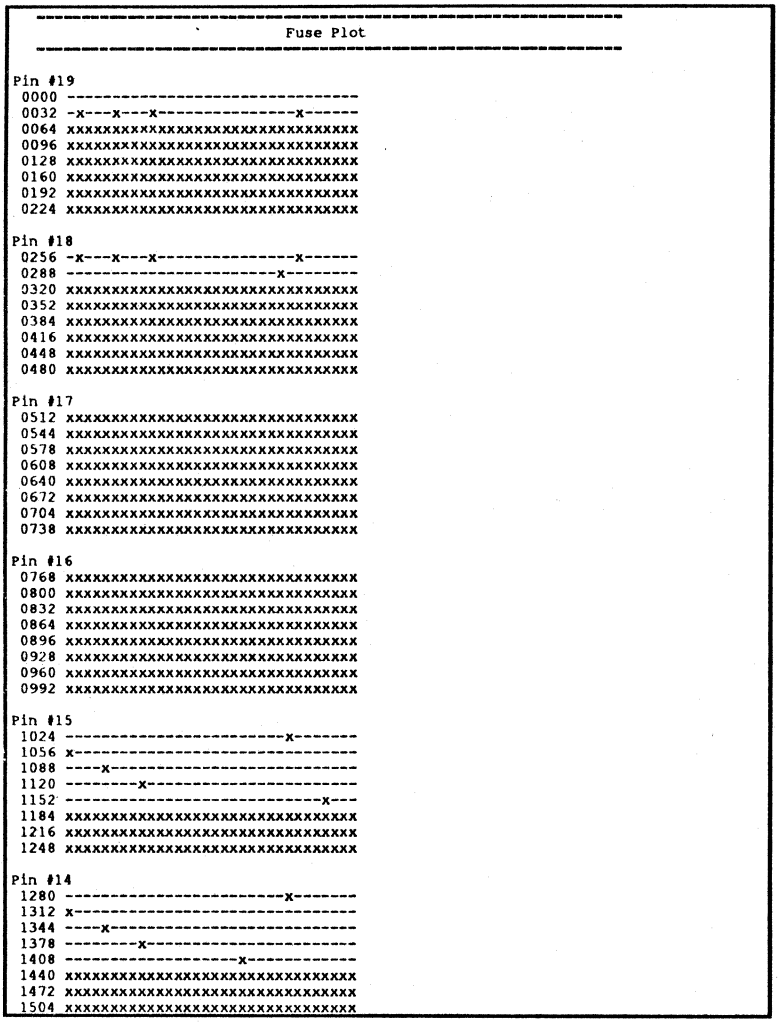


Figure U5-10. Sheet 3 of 4

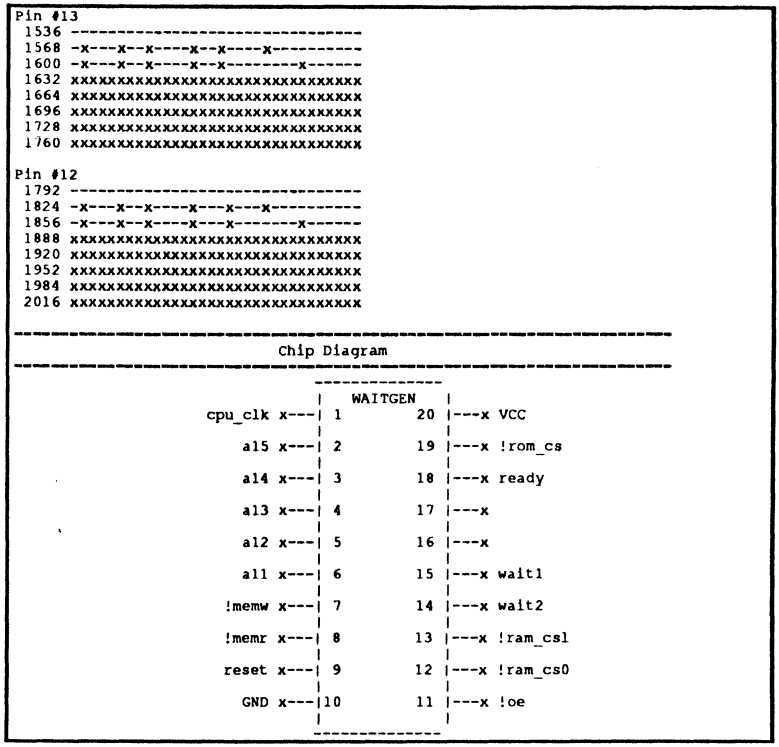


Figure U5-10. Sheet 4 of 4

The expanded product terms for **WAIT1.D** and **WAIT2.D** show five product terms, because the fixed inverting output buffer (active-LO architecture) in the PAL16R4 causes CUPL to perform DeMorgan's Theorem on equations when an output variable has been declared as active-HI in the pin list for this particular device.

❑ STEP 7. CREATING THE CSIM SOURCE FILE

In this step, a simulation will be performed to verify the compiled design for the PAL16R4 device. Performing this step before downloading to a logic programmer decreases the probability of programming a device with incorrect logic.

Create a source specification file, **SAMPLE.SI**, containing test vectors for input to **CSIM**. **CSIM** compares the test vector inputs and expected outputs to the actual values contained in the **SAMPLE.ABS** file that was created during **CUPL** operation, and flags any discrepancies.

Figure U5-11 shows the contents of a sample source specification file.

```
Name                    Sample;
Partno                  P9000183;
Date                    07/16/87;
Revision                02;
Designer                Osann;
Company                 ATI;
Assembly                PC Memory;
Location                U106;

/*******************************************************/
/* This device generates chip select signals for one   */
/* 8Kx8 ROM and two 2Kx8 static RAMs. It also drives    */
/* the system READY line to insert a wait-state of at  */
/* least one cpu clock for ROM accesses                */
/*******************************************************/

ORDER:
     cpu_clk, %2, al5, %2, al4, %2,
     al3, %2, al2, %2, all, %2,
     !memw, %2, !memr, %2, reset, %2, !oe,
     %4, !ram_cs1, %2, !ram_cs0, %2, !rom_cs, %2,
     wait1, %2, wait2, %2, ready;

VECTORS:
/* 123456-leave six blanks to allow for numbers in .SO file */
$msg "                                          !  !          ";
$msg "      c                                   r  r  !       ";
$msg "      p                                   a  a  r       ";
$msg "      u                    !  !  r        m  m  o  w  w  r";
$msg "                           m  m  e        _  _  m  a  a  e";
$msg "      c  a  a  a  a  a  e  e  e  s  !     c  c  _  i  i  a";
$msg "      l  1  1  1  1  1  m  m  e  o        s  s  c  t  t  d";
$msg "      k  5  4  3  2  1  w  r  t  e        1  0  s  1  2  y";
$msg "                                                        ";
$msg "      Power On Reset                                    ";
      O  X  X  X  X  X  1  1  1  0     H  H  H  *  *  Z
$msg "      Reset Flip Flops                                  ";
      C  X  X  X  X  X  1  1  0  0     H  H  H  L  L  Z
$msg "      Write RAM0                                        ";
      0  0  0  1  0  0  0  1  0  0     H  L  H  L  L  Z
$msg "      Read RAM0                                         ";
      0  0  0  1  0  0  1  0  0  0     H  L  H  L  L  Z
$msg "      Write RAM1                                        ";
      0  0  0  1  0  1  0  1  0  0     L  H  H  L  L  Z
$msg "      Read RAM1                                         ";
      0  0  0  1  0  1  1  0  0  0     L  H  H  L  L  Z
$msg "      Begin ROM read                                    ";
      0  0  0  0  0  0  1  0  0  0     H  H  L  L  L  L
$msg " Two clocks for wait state, Then drive READY High       ";
$repeat2;
      C  0  0  0  0  0  1  0  0  0     H  H  L  *  *  *
$msg "      End ROM Read                                      ";
      0  0  0  0  0  0  1  1  0  0     H  H  H  H  H  Z
$msg "      End ROM Read                                      ";
      C  0  0  0  0  0  1  1  0  0     H  H  H  L  L  Z
```

Figure U5-11. SAMPLE.SI

The source specification file contains three major parts: header information and title block, an **ORDER** statement, and a **VECTORS** statement.

SAMPLE.SI must have the same header information as **SAMPLE.PLD** to ensure that the proper files, including current revision level, are being compared against each other. Therefore, first copy **SAMPLE.PLD** to **SAMPLE.SI** and then

use a text editor to delete everything in **SAMPLE.SI**, except the header and title block. Figure U5-12 shows the result.

```
Name                    Sample;
Partno                  P9000183;
Date                    07/16/87;
Revision                02;
Designer                Osann;
Company                 ATI;
Assembly                PC Memory;
Location                U106;

/*************************************************/
/* This device generates chip select signals for one  */
/* 8Kx8 ROM and two 2Kx8 static RAMs. It also drives  */
/* the system READY line to insert a wait-state of at */
/* least one cpu clock for ROM accesses               */
/*************************************************/
```

Figure U5-12. SAMPLE.SI Header Information

In the **ORDER** statement, list the input and output variables from **SAMPLE.PLD** to be included in test vectors. List the variables in the order in which they will be used in test variables; that is, put the clock variable, **CPU_CLK**, first, followed by the other input variables. Put the output variables to the right. Separate variables with commas. Use the **%** symbol to insert spaces between the variables; put two spaces between each variable, and four spaces between the last input variable in the list, **!OE**, and the first output variable, **!RAM_CS1**. Type the **ORDER** statement as follows:

```
ORDER:
    CPU_CLK, %2, A15, %2, A14, %2,
    A13, %2,A12, %2, ALL, %2,
    !MEMW, %2, !MEMR, %2, RESET, %2, !OE,
    %4, !RAM_CS1, %2, !RAM_CSO, %2, !ROM_CS, %2
    WAIT1, %2, READY;
```

Following the **ORDER** statement, enter a **VECTORS** statement that creates a function table containing eleven test vectors (see Figure U5-11). First, to make the vectors easier to enter and understand, use the **$MSG** command to create a heading for

the function table. List the variable names in vertical columns in the same order and with the same spacing as specified in the **ORDER** statement, by typing the information in Figure U5-13.

```
/* 123456-leave six blanks to allow for numbers in .SO file */
$msg "                                          ! !          ";
$msg "      c                                   r r !        ";
$msg "      p                                   a a r        ";
$msg "      u                       ! ! r       m m o w w r";
$msg "                              m m e         m a a e";
$msg "      c a a a a a e e s !     c c   i i a";
$msg "      l 1 1 1 1 1 m m e o     s s c t t d";
$msg "      k 5 4 3 2 1 w r t e     1 0 s 1 2 y";
$msg "      ____________________    ____________";
```

Figure U5-13. Vectors Table Header

Now enter the test vectors. Create the vectors by assigning a value to each of the input variables and an expected value to each of the output variables. Refer to Table 4-4 in Chapter 4, "Using CSIM", for allowable values to use for test vectors. Use the **$MSG** directive to describe the device function tested by the function. The **ORDER** statement above specifies the spacing when creating the test vectors. For example, create the first vector, Power On Reset, by typing:

```
$msg " Power On Reset      ";
0XXXXX1110HHH**Z
```

Return

Note that the output value (*) has been used for **WAIT1** and **WAIT2** to instruct **CSIM** to calculate the power-on state of the registers, since some devices power-on to X and some to H or L. Using the asterisk gives a more universal simulation file. Type in the rest of the test vectors, as shown in Figure U5-14.

```
$msg "     Power On Reset                                          ";
            O  X  X  X  X  X  1  1  1  0    H  H  H  *  *  Z
$msg "     Reset Flip Flops                                        ";
            C  X  X  X  X  X  1  1  0  0    H  H  H  L  L  Z
$msg "     Write RAM0                                              ";
            0  0  0  1  0  0  0  1  0  0    H  L  H  L  L  Z
$msg "     Read RAM0                                               ";
            0  0  0  1  0  0  1  0  0  0    H  L  H  L  L  Z
$msg "     Write RAM1                                              ";
            0  0  0  1  0  1  0  1  0  0    L  H  H  L  L  Z
$msg "     Read RAM1                                               ";
            0  0  0  1  0  1  1  0  0  0    L  H  H  L  L  Z
$msg "     Begin ROM read                                          ";
            0  0  0  0  0  0  1  0  0  0    H  H  L  L  L  L
$msg " Two clocks for wait state, Then drive READY High            ";
$repeat2;
            C  0  0  0  0  0  1  0  0  0    H  H  L  *  *  *
$msg "     End ROM Read                                            ";
            0  0  0  0  0  0  1  1  0  0    H  H  H  H  H  Z
$msg "     End ROM Read                                            ";
            C  0  0  0  0  0  1  1  0  0    H  H  H  L  L  Z
```

Figure U5-14. Test Vectors

The **$REPEAT** directive in the test vectors causes the eighth vector to be repeated twice. The asterisks in the eighth vector for **WAIT1**, **WAIT2**, and **READY** tell **CSIM** to compute the output based on the inputs and place the results in the output file.

The value of the clock variable, **CPU_CLK**, is 0 in some vectors and C in others. A value of 0 causes no clocking to occur. A value of C causes **CSIM** to examine the input values in the vector and also look back to the previous vector for any registered outputs that would be fed back internally prior to the clock. Then, after a clock is applied, **CSIM** computes the appropriate expected outputs for registered and non-registered variables.

After putting in the **VECTORS** statement, save the file. The next step is to run **CSIM** to perform the simulation.

❑ STEP 8. RUNNING CSIM

When **CSIM** is run, it creates **SAMPLE.SO**, which contains the result of the simulation. Specify the **-l** flag to list any errors that might be generated.

To run **CSIM**, type:

csim -1 p16r4 sample Return

Note

If **WAITGEN.PLD** was used to run **CUPL** in step 6, specify WAITGEN instead of SAMPLE when running **CSIM**.

CSIM displays the amount of time to perform the simulation, as follows:

```
CSIM: CUPL Simulation Program
Version 3.XX Serial # XX-XXX-XXXX
Copyright (C) 1983, 1990 Logical Devices, Inc.

csima
time: 4 secs
total time: 4 secs
```

When the prompt reappears, the simulation is complete. **SAMPLE.SO** is an ASCII file, so it is possible to display it on the screen, print a hardcopy of it, or open it with a text editor.

Figure U5-15 shows the contents of **SAMPLE.SO**.

```
                        SAMPLE.SO

CSIM:    CUPL Simulation Program
Version 4.XX Serial # XX-XXX-XXXX
copyright (c) 1983,1990 Logical Devices, Inc.
CREATED Thur Aug 20 09:34:16 1990

 1: Name                     Sample;
 2: Partno                   P9000183;
 3: Date                     07/16/87;
 4: Revision                 02;
 5: Designer                 Osann;
 6: Company                  ATI;
 7: Assembly                 PC Memory;
 8: Location                 U106;
 9:
10: /*****************************************************/
11: /* This device generates chip select signals for one  */
12: /* 8Kx8 ROM and two 2Kx8 static RAMs. It also drives  */
13: /* the system READY line to insert a wait-state of at */
14: /* least one cpu clock for ROM accesses               */
15: /*****************************************************/
16:
17: ORDER:
18:      cpu_clk, %2, a15, %2, a14, %2,
19:      a13, %2, a12, %2, a11, %2,
20:      !memw, %2, !memr, %2, reset, %2, !oe,
21:      %4, !ram_cs1, %2, !ram_cs0, %2, !rom_cs, %2,
22:      wait1, %2, wait2, %2, ready;
23:
-------------------------------------------------------------
                      Simulation Results
-------------------------------------------------------------
                                                    !   !
      c                                       r  r  !
      p                                       a  a  r
      u                          !  !  r      m  m  o  w  w  r
      _                          m  m  e      _  _  m  a  a  e
      c  a  a  a  a  a  e  e  s  !   c  c  _  i  i  a
      l  1  1  1  1  1  m  m  e  o   s  s  c  t  t  d
      k  5  4  3  2  1  w  r  t  e   1  0  s  1  2  y
     ------------------------------   ------------------
     Power On Reset
0001: O  X  X  X  X  X  1  1  1  0    H  H  H  X  X  Z
     Reset Flip Flops
0002: C  X  X  X  X  X  1  1  0  0    H  H  H  L  L  Z
     Write RAM0
0003: 0  0  0  1  0  0  0  1  0  0    H  L  H  L  L  Z
     Read RAM0
0004: 0  0  0  1  0  0  1  0  0  0    H  L  H  L  L  Z
     Write RAM1
0005: 0  0  0  1  0  1  0  1  0  0    L  H  H  L  L  Z
     Read RAM1
0006: 0  0  0  1  0  1  1  0  0  0    L  H  H  L  L  Z
     Begin ROM read
0007: 0  0  0  0  0  0  1  0  0  0    H  H  L  L  L  L
     Two clocks for wait state, Then drive READY High
0008: C  0  0  0  0  0  1  0  0  0    H  H  L  H  L  L
0009: C  0  0  0  0  0  1  0  0  0    H  H  L  H  H  H
     End ROM Read
0010: 0  0  0  0  0  0  1  1  0  0    H  H  H  H  H  Z
     End ROM Read
0011: C  0  0  0  0  0  1  1  0  0    H  H  H  L  L  Z
```

Figure U5-15. SAMPLE.SO

Compare **SAMPLE.SO** to the **SAMPLE.SI** file in Figure U5-11. Note that vectors 8 and 9 were created as a result of the **$REPEAT** directive, and that **CSIM** has replaced the asterisks

from **SAMPLE.SI** with the appropriate logic levels (H and L) for the **WAIT1, WAIT2** and **READY** signals.

Now that a successful simulation has been completed, test vectors can be added to the JEDEC file created while running **CUPL** (in step 6). Run the simulation again with the **-j** option flag, by typing:

csim -j p16r4 sample Return

If **WAITGEN.SI** was used to perform the simulation, specify it here instead of **SAMPLE.SI**.

Figure U5-16 shows the contents of **SAMPLE.JED**, which now contains both programming and testing information.

```
                                    SAMPLE.JED

CUPL                4.XX   Serial# XX-XXX-XXXX
Device              p16r4  Library DLIB-d-26-11
Created             Thur Aug 20 09:52:02 1990
Name                Sample
Partno              P9000183
Revision            02
Date                12/16/89
Designer            Osann
Company             ATI
Assembly            PC Memory;
Location            U106;
*QP20
*QF2048
*G0
*F0
*L00000 11111111111111111111111111111111
*L00032 10111011101111111111111110111111
*L00256 10111011101111111111111110111111
*L00288 11111111111111111111111011111111
*L01024 11111111111111111111111101111111
*L01056 01111111111111111111111111111111
*L01088 11110111111111111111111111111111
*L01120 11111111011111111111111111111111
*L01152 11111111111111111111111111110111
*L01280 11111111111111111111111101111111
*L01312 01111111111111111111111111111111
*L01344 11110111111111111111111111111111
*L01376 11111111011111111111111111111111
*L01408 11111111111111111111101111111111
*L01536 11111111111111111111111111111111
*L01568 10111011011110110111101111111111
*L01600 10111011011110110111111110111111
*L01792 11111111111111111111111111111111
*L01824 10111011011110110111011111111111
*L01856 10111011011110110111111110111111
*C4D50
*V0001 0XXXXX111N0HHXXXXZHN
*V0002 CXXXXX110N0HHLLXXZHN
*V0003 000100010N0LHLLXXZHN
*V0004 000100100N0LHLLXXZHN
*V0005 000101010N0HLLLXXZHN
*V0006 000101100N0HLLLXXZHN
*V0007 000000100N0HHLLXXLLN
*V0008 C00000100N0HHLHXXLLN
*V0009 C00000100N0HHHHXXHLN
*V0010 000000110N0HHHHXXZHN
*V0011 C00000110N1HHLLXXZHN
*3152
```

Figure U5-16. SAMPLE.JED with Test Vectors

❑ SUMMARY

This part provided the opportunities to create and compile a **CUPL** source file and a **CSIM** test specification file. The important points were how to:

- ➢ Use the template file.
- ➢ Choose a PLD and make pin assignments.
- ➢ Write intermediate and logic equations to describe the design.
- ➢ Run **CUPL** to compile a file.
- ➢ Create and compile a test specification file to verify the design.
- ➢ Run **CSIM** to simulate a logic design.

The design examples in Part B do not provide the same step-by-step instruction, but are provided as familiarization with how CUPL can be used to describe different types of designs.

PART B. SAMPLE PLD FILES

This section lists the logic description files that are included in the CUPL package to illustrate how **CUPL** and **CSIM** implement various designs.

FILE: GATES.PLD
DEVICES: PAL16L8, PAL16P8 , 82S153
Simple use of NOT, AND, OR, and XOR gates

FILE: FLOPS.PLD
DEVICES: PAL16R8, PAL16RP8, 82S159
Using D-type flip-flops to create a 2-bit counter (four ways)

FILE: SHFTCNT.PLD
DEVICES: 82S105, 82S159
4-bit counter/shifter using SR SHFTCNT4.PLD and JK-type flip-flops

FILE: SHFTCNT6.PLD
DEVICES: 82S167
4-bit counter/shifter

FILE: BARREL22.PLD
DEVICES: PAL22V10
8-bit barrel shifter

FILE: HEXDISP.PLD
DEVICES: 32x8 PROM
7-segment decoder

FILE: COUNT8.PLD
DEVICES: PAL20X8
8-bit counter using XOR capability

FILE: COUNT8A.PLD
DEVICES: PAL20X8
8-bit counter using set notation

FILE: IODECODE.PLD
DEVICES: PAL12L6 , PAL12P6, 82S153
I/O decoder

FILE: MDECODE.PLD
DEVICES: PAL16L8, PAL16P8 , 82S153
Memory decoder

FILE: RIPPLE8.PLD
DEVICES: PAL20RA10
8-bit ripple counter with asynchronous load

FILE: COUNT13.PLD
DEVICES: PAL32R16
13-bit counter using set notation with load hold and clear

FILE: IOPORT.PLD
DEVICES: PAL20RA10
7-bit parallel I/O port with hand-shake and data ready flag

FILE: DECADE.PLD
DEVICES: 82S157
Decade counter uses state-machine syntax and complement array

FILE: ADDER.PLD
DEVICES: PAL16L8, PAL16P8, 82S153
4-bit adder slice uses CUPL function calls

FILE: LOOKUP.PLD
DEVICES: 256 x 8 EPROM
Arithmetic lookup table for PROM circle perimeter

FILE: COUNT10.PLD
DEVICES: PAL16RP4, GAL16V8
Synchronous up/down counter with clear, uses state-machine syntax

FILE: BUSARB.PLD

DEVICES: 82S105
Multiprocessor bus arbiter having two machines in one part

FILE: TCOUNTER.PLD
DEVICES: EP600
16-bit synchronous counter using toggle flip-flops

FILE: PRIORITY.PLD
DEVICES: PALR19L8
68000 priority interrupt encoder with registered inputs

FILE: DATASEP.PLD
DEVICES: EP600
Single density 8" floppy disk data separator

FILE: KEYBOARD.PLD
DEVICES: 82S100
Keyboard encoder converts rows and columns to ASCII

FILE: STEPPER.PLD
DEVICES: PALT19R6
Stepper motor controller interface for 8048 microprocessor

FILE: MULTIBUS.PLD
DEVICES: PAL23S8
Simple MULTIBUS arbiter supports parallel and serial priority

FILE: ADDER_TT.PLD
DEVICES: PAL16L8, PAL16P8, 82S153

FILE: TTL.PLD
DEVICES: PAL16L8
TTL chip representation using macros

FILE: CYP_CNT.PLD
DEVICES: CY7C330
Up/Down counter with preloadable upper and lower limits.

Any of these logic description files can be viewed or printed out, or they can be input to **CUPL** to generate documentation or download files. A corresponding test specification file (***filename*.SI**) is also provided for each logic description file, so that **CSIM** can be run to verify the designs.

The following examples describe key points of the following designs (the logic description file for each design is shown in parentheses):

- Simple gates (GATES.PLD)
- TTL conversion (WGTTL.PLD)
- Two-bit counter (FLOPS.PLD)
- Decade up/down counter using state-machine syntax (COUNT10.PLD)
- Seven-segment display decoder (HEXDISP.PLD)

EXAMPLE 1. SIMPLE GATES

This example describes a design containing simple gates. Figure U5-17 shows the design.

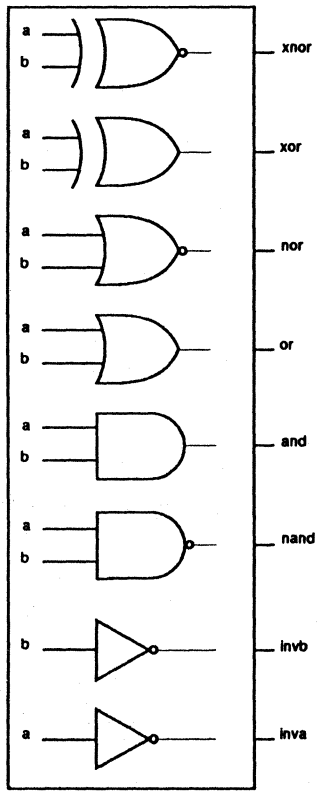


Figure U5-17. Design with Simple Gates

The outputs are labeled to reflect the function of their gate; for example, the AND gate has an output labeled AND.

Figure U5-18 shows the **CUPL** source file (**GATES.PLD** provided in the CUPL package) that describes the design.

```
                              GATES.PLD
Name            Gates;
Partno          CA0001;
Date            07/16/87;
Designer        G Woolheiser;
Company         ATI;
Location        San Jose, CA.;
Assembly        Example;

/*************************************************/
/*                                               */
/* This is an example to demonstrate how CUPL    */
/* compiles simple gates                         */
/*                                               */
/*************************************************/
/*   Target Devices: P16L8, P16P8, EP300, and 82S153  */
/*************************************************/

/* Inputs:   define inputs to build simple gates     */
Pin 1 = a;
Pin 2 = b;

/* Outputs:   define outputs as active HI levels

   For PAL16L8 and PAL16LD8, De Morgan's Theorem is
   applied to invert all outputs due to fixed
   inverting buffer in the device.                   */

Pin 12 = inva;
Pin 13 = invb;
Pin 14 = and;
Pin 15 = nand;
Pin 16 = or;
Pin 17 = nor;
Pin 18 = xor;
Pin 19 = xnor;

/* Logic:   examples of simple gates expressed in CUPL */

inva = !a;              /* inverters              */
invb = !b;
and  = a & b;           /* and gate               */
nand = !(a & b);        /* nand gate              */
or   = a # b;           /* or gate                */
xor  = a $ b;           /* nor gate               */
xnor = !(a $ b);        /* exclusive nor gate     */
```

Figure U5-18. Simple Gates Source File (GATES.PLD)

The first part of the file provides archival information and a description of the intended function of the design, including compatible PLDs.

Pin declarations are made corresponding to the inputs and outputs in the design diagram.

In the "**Logic**" section of the file, equations describe each of the gates in the design.

For the PAL16L8 and PAL16LD8 devices, which contain fixed inverting buffers, **CUPL** applies DeMorgan's Theorem to invert all outputs because they were all declared active-HI in the pin list. For example, during compilation, **CUPL** converts the following equation for an OR gate, on an output pin that has been declared as active high:

```
or = a # b;
```

to the following single expanded product term (as shown in the documentation file):

```
or => !a & !b
```

EXAMPLE 2. CONVERTING A TTL DESIGN TO PLDs

This example shows how to use a PLD to replace existing TTL circuitry. The conversion requires translating the gates of a TTL logic design into equivalent Boolean logic equations, which can then be compiled by **CUPL** and assigned to a PLD.

Figure U5-19 shows the TTL gate representations used in designing logic systems and the corresponding Boolean equation for each gate.

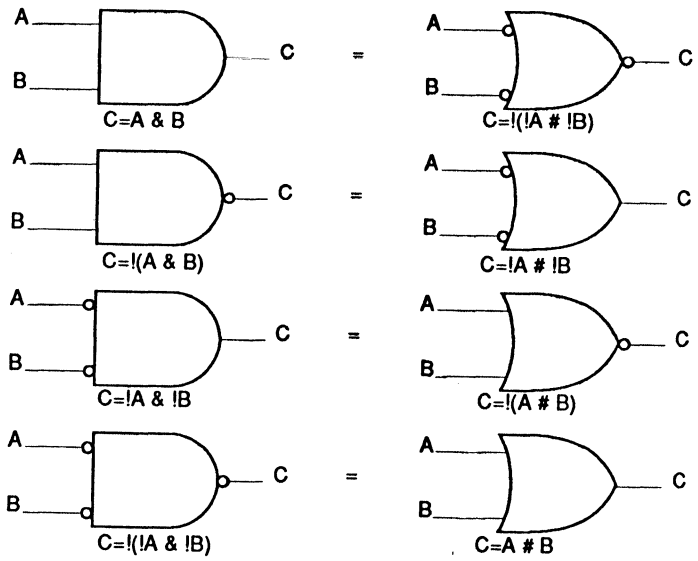


Figure U5-19. TTL Gate Representations and Boolean Equations

The basic conversion rules shown in Figure U5-19 are sufficient to write equations for each gate within a system of TTL gates when converting the logic to a PLD equivalent. CUPL uses an expression substitution process to build larger equations from the smaller expressions representing each gate in the TTL schematic. Expression substitution permits approaching a schematic one gate at a time.

Figure U5-20 shows the schematic for the TTL logic that is converted in Example 1.

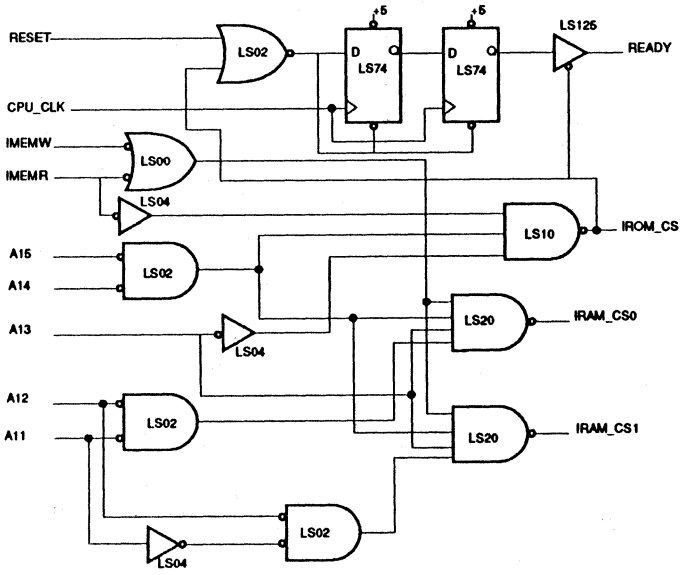


Figure U5-20. TTL Schematic

The TTL logic shown in Figure U5-20 performs the same address decoding and wait state generation as the **WAITGEN.PLD** file contained in the CUPL distribution package. The **SAMPLE.PLD** file created in the sample design session (see Part A of this chapter) is identical to **WAITGEN.PLD**.

The PLD equivalent of this TTL circuit replaces five to six packages with one device.

The first step in the conversion process is to determine from the TTL schematic the logic that is to be placed in the PLD. Figure U5-21 shows a PLD diagram equivalent to the TTL schematic with a box around the logic, and PLD pin number assignments.

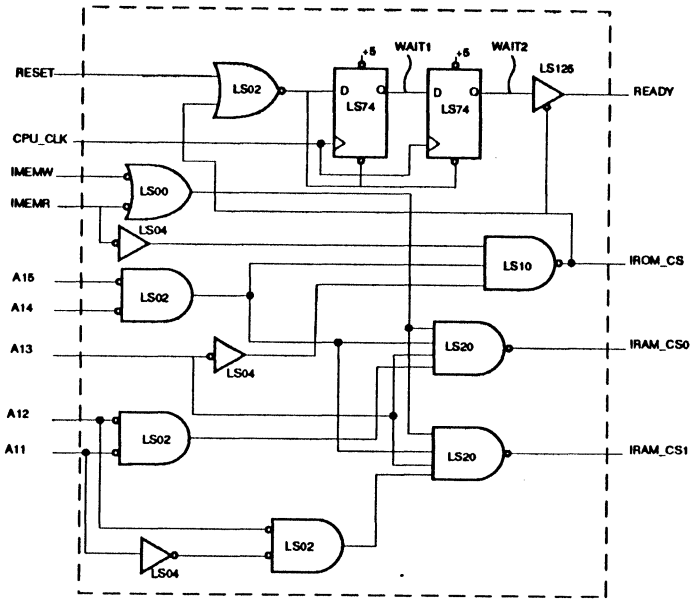


Figure U5-21. PLD Equivalent Diagram

Note that the outputs of the internal gates (those that do not connect to the PLD output pins) are arbitrarily labeled with the variable names, A-H, to aid in entering equations in the logic description file.

The logic description file used to convert this design is named **WGTTL.PLD** because it performs wait state generation and is based on a TTL design. Figure U5-22 shows the contents of **WGTTL.PLD**.

```
                WGTTL.PLD

Name                    Sample;
Partno                  P9000183;
Date                    07/16/87;
Revision                02;
Designer                Osann;
Company                 ATI;
Assembly                PC Memory;
Location                U106;

/*******************************************************/
/* This device generates chip select signals for one   */
/* 8Kx8 ROM and two 2Kx8 static RAMs. It also drives   */
/* the system READY line to insert a wait-state of at  */
/* least one cpu clock for ROM accesses                */
/*******************************************************/
/** Inputs **/

PIN 1           = cpu_clk       ;  /* CPU clock          */
PIN [2..6]      = [a15..11]     ;  /* CPU Address Bus    */
PIN 9           = ![memw,memr]  ;  /* Memory Data Strobes*/
PIN 11          = !oe           ;  /* Output enable      */

/** Outputs **/

PIN 19          = !rom_cs       ;  /* ROM Chip Select    */
PIN 18          = ready         ;  /* CPU ready signal   */
PIN 15          = wait1         ;  /* Start wait state   */
PIN 14          = wait2         ;  /* End wait state     */
PIN [13,12]     = ![ram_cs1..0] ;  /* RAM chip selects   */

/** Declarations and Intermediate Variable Definitions **/

a = !(!memw) # !(!memr) ;
b = !a15 & !a14 ;
c = !a13 ;
d = !a12 & !a11 ;
e = !a11 ;
f = !a12 & !oe ;
g = !(!rom_cs # reset) ;
h = !(!memr) ;

/** Logic Equations **/

!rom_cs = !(h & b & c);

!ram_cs0 = !(a & b & a13 & d) ;

!ram_cs1 = !(a & b & a13 & f) ;

wait1.d = g ;

wait2.d = wait1 & g ;

ready.oe = !(!(h & b & c)) ;

ready = wait2 ;
```

Figure U5-22. WGTTL.PLD

The header information is identical to that in **WAITGEN.PLD** (and **SAMPLE.PLD**; see Part A), because the functionality is the same.

The pin assignments match the schematic in Figure U5-21.

The logic equations for the internal gates are placed in the section of the file for "Declarations and Intermediate Variable Definitions." The equations in this section use the output variable names, A-H, assigned in the schematic in Figure U5-21. For example, the AND gate LS02 is described by the following equation:

```
d = !a12 & !a11 ;
```

The equations in this section can be simplified. For example, the double negations can be eliminated in the following equation:

```
a = !(!memw) # !(!memr) ;
```

by entering:

```
a = memw # memr ;
```

The section of the file "Logic Equations" contains equations that describe the output signals of the PLD. These equations are written in terms of the intermediate equations that describe the outputs of the internal gates. For example, the AND gate, LS10, has **!ROM_CS** as its output signal, and the signals H, B and C as inputs. Therefore, enter the following equation to describe LS10:

```
!rom_cs = !(h & b & c) ;
```

WGTTL.PLD to **WAITGEN.PLD** are not exactly alike, because the internal gates have been defined differently. When compiled, however, they perform the same function. This can be verified by simulating each logic description in turn with an identical simulation source file (**WGTTL.SI** and **WAITGEN.SI** provided in the CUPL package).

When converting a TTL design to a PLD, some slight changes in functionality are required. The asynchronous reset capability found on TTL flip-flops like the LS74 is not found in many of the commonly available PLDs. However, the same reset capability can be achieved by including a **RESET** variable in all product terms to ensure a synchronous reset at the clock.

Therefore, **WGTTL.PLD** incorporates **!RESET** into the equation for G, which is used in the equations for both **WAIT1** (wait1.d = g;) and **WAIT2** (wait2.d = wait1.d & g). Although the functional nature of the synchronous reset does differ in timing from that of the asynchronous reset, the synchronous reset is sufficient for proper function of the device.

The simple methodology described in Example 2 allows the conversion of many TTL designs, especially those consisting of simple gates, to a PLD equivalent, regardless of the number of gate delays of TTL (logic) in the original design. In most cases, the only difference between the TTL design and the PLD is the total propagation delay through the circuit.

❑ EXAMPLE 3. TWO-BIT COUNTER

This example demonstrates the implementation of a two-bit counter for a D-type flip-flop.

Figure U5-23 shows the timing diagram for the counter.

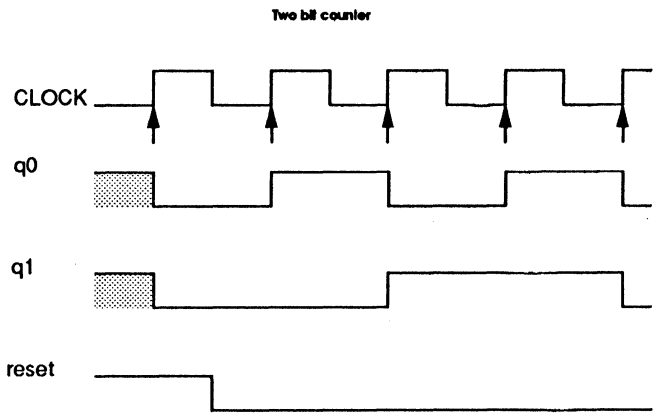


Figure U5-23. Two-Bit Counter Timing Diagram

As indicated by the arrows, the registers are clocked on the rising edge of the clock signal.

Figure U5-24 shows the **CUPL** source file (**FLOPS.PLD**, provided in the CUPL package) to describe the two-bit counter design.

```
                        HEXDISP.PLD

Name                    Flops;
Partno                  CA0002;
Revision                02;
Date                    07/16/87;
Designer                G. Woolheiser;
Company                 ATI;
Location                None;
Assembly                None;

/*************************************************************/
/*                                                           */
/*  This example demonstrates the use of D-type flip-flop    */
/*  to implement a two bit counter using the following       */
/*  timing diagram.                                          */
/*                                                           */
/*  clock  |__|  |__|  |__|  |__|  |__|  |__|  |__|          */
/*                                                           */
/*  q0     ¯///¯|_____|     |_____|     |_____|  |_         */
/*                                                           */
/*  q1     ¯///¯|___________|           |_____|  |_         */
/*                                                           */
/*                                                           */
/*                                                           */
/*  reset  ¯¯¯¯¯¯¯|_____________________________________     */
/*                                                           */
/*************************************************************/
/*      Target Devices: PAL16R8, PAL16RP8, GAL16V8           */
/*************************************************************/

Pin 1 = clock;
Pin 2 = reset;

/*  Outputs: define outputs and output active levels   */

Pin 17 = q0;
Pin 16 = q1;

/*  Logic:  two bit counter using expanded exclusive
ors with d-type flip-flop       */
q0.d = !reset & (!q0 & !q1
               # !q0 & q1);
q1.d = !reset & (!q0 & q1
               # q0 & !q1);
/* ANDed !reset defines a synchronous register reset */
```

Figure U5-24. Two-Bit Counter Source File (FLOPS.PLD)

The first part of the file provides archival information and a description of the intended function of the design, including compatible PLDs.

Pin declarations are made corresponding to the inputs and outputs in the design diagram.

In the "**Logic**" section of the file, equations are written to implement the counter. The equation for q0 is written to define when q0 asserts; that is, it defines the situation immediately before the rising clock edge.

The **!reset** term is used in the equations for both q0 and q1 to initialize the circuit, providing a synchronous reset. At power-on, the registers can be either high or low, as indicated by the DON'T CARE slashes in the timing diagram (see Figure U5-23); the reset signal is initially asserted. By ANDing **!reset** into the equation for each variable, the conditions are not met at power-on, so the registers do not set. Because the reset signal returns LO (false) after the power-on process is complete, **!reset** is then true and does not affect the value of the registers at any other point in the circuit.

The **.d** extension in the equations specifies a D-type flip-flop. However, when an output is used as feedback, the **.d** extension is dropped. For example, if q0 is fed back to q1, an equation could be written as:

```
q1.d = q0 & !reset;
```

not as:

```
q1.d = q0.d & !reset;
```

or:

```
q1.d = q0.dq & !reset;
```

❑ EXAMPLE 4. DECADE UP/DOWN COUNTER

This example describes a four-bit up/down decade counter with a synchronous clear capacity. The counter also provides an asynchronous ripple carry output for cascading multiple devices. The source file to implement the counter uses CUPL state machine syntax.

Figure U5-25 shows the counter design and its state.

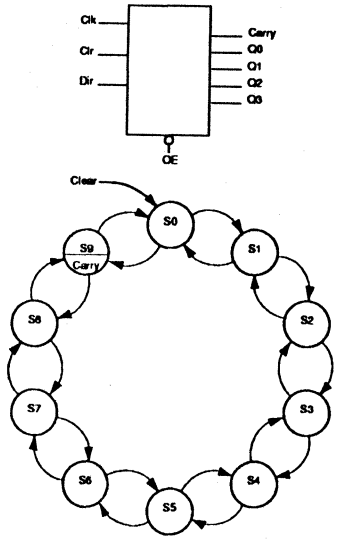


Figure U5-25. Up/Down Counter Diagram

The input signal **dir** determines the direction of the count. When **dir** is high, the count goes down one on each clock; when **dir** is low, the count goes up one on each clock. The **clr** signal performs a synchronous reset.

Figure U5-26 shows the **CUPL** source file (**COUNT10.PLD**, provided in the CUPL package) that implements the design.

```
                COUNT10.PLD

Name                Count10;
Partno              CA0018;
Revision            02;
Date                07/16/87;
Designer            Kahl;
Company             ATI;
Location            None;
Assembly            None;
Device              p16rp4;

/*************************************************************/
/*                                                           */
/*                  Decade Counter                           */
/* This is a 4-bit up/down decade counter with               */
/* synchronous clear capability. An asynchronous             */
/* ripple carry output is provided for cascading             */
/* multiple devices. CUPL state machine syntax               */
/* is used                                                   */
/*************************************************************/
/* Allowable Target Device Types: PAL16RP4, GAL16V8, EP300 */
/*************************************************************/
/**  Inputs  **/
Pin 1 = clk;                    /* counter clock             */
Pin 2 = clr;                    /* counter clear input       */
Pin 3 = dir;                    /* counter direction input   */
Pin 11 = !oe;                   /* Register output enable    */

/*  Outputs                                                  */

Pin [14..17] = [Q3..0];         /* counter outputs           */
Pin 18 = carry;                 /* ripple carry out          */

/*  Declarations and Intermediate Variable Definitions       */
field count = [Q3..0];          /* declare counter bit field */
$define S0 'b'0000
$define S1 'b'0001
$define S2 'b'0010
$define S3 'b'0011
$define S4 'b'0100
$define S5 'b'0101
$define S6 'b'0110
$define S7 'b'0111
$define S8 'b'1000
$define S9 'b'1001
field node = [clr,dir];         /* declare filed node control */
up = mode:0;                    /* define count up mode       */
down = mode:1;                  /* define count down mode     */
clear = mode:2..3];             /* define count clear mode    */
```

Figure U5-26. Up/Down Counter Source File (COUNT10.PLD) Sheet 1 of 2

```
/* Logic Equations */
sequence count {      /*     free running counter     */

present S0          if up          next S1;
                    if down        next S9;
                    if clear       next S0;
present S1          if up          next S2;
                    if down        next S0;
                    if clear       next S0;
present S2          if up          next S3;
                    if down        next S1;
                    if clear       next S0;
present S3          if up          next S4;
                    if down        next S2;
                    if clear       next S0;
present S4          if up          next S5;
                    if down        next S3;
                    if clear       next S0;
present S5          if up          next S6;
                    if down        next S4;
                    if clear       next S0;
present S6          if up          next S7;
                    if down        next S5;
                    if clear       next S0;
present S7          if up          next S8;
                    if down        next S6;
                    if clear       next S0;
present S8          if up          next S9;
                    if down        next S7;
                    if clear       next S0;
present S9          if up          next S0;
                    if down        next S8;
                    if clear       next S0;
                    out            carry;          /* assert carry output */
```

Figure U5-26. Sheet 2 of 2

The first part of the file provides archival information and a description of the intended function of the design, including compatible PLDs.

Pin declarations are made corresponding to the inputs and outputs in the design diagram.

The "Declarations and Intermediate Variable Definitions" section contains declarations that simplify the notation.

The name **"count"** is assigned to the output variables Q3, Q2, Q1, and Q0.

The **$DEFINE** command is used to assign names to ten binary states representing the state machine output. The state name can then be used in the logic equations to represent the corresponding binary number.

The **FIELD** keyword is used to combine the **clr** and **dir** inputs into a set called **mode. Mode** is defined by the following equations:

```
up = mode:0;
down = mode:1;
clear = mode [2..3];
```

Mode represents the inputs **clr** and **dir**, so the three equations above are equivalent to the following equations:

```
up = !clr & !dir ;
down = !clr & dir ;
clear = (clr & !dir) # (clr & dir) ;
```

The three modes are defined as follows:

up - Both the **dir** and **clr** inputs are not asserted.

down - The **dir** input is asserted and **clr** is not asserted.

clear - The **clr** input is asserted and **dir** is either asserted or not asserted.

The **"Logic Equations"** section contains the state machine syntax that specifies the states in the counter. In the first line, the **SEQUENCE** keyword identifies count (that is, Q3, Q2, Q1, and Q0) as the outputs to which the state values apply.

Conditional statements have been written to specify the transition from each possible present state to a next state, for each of the three modes. For example, when the present state is S4, if the mode is up, the counter goes to S5; if the mode is down the counter goes to S3; or if the mode is clear, the counter goes to S0. As example 4 shows, one advantage of the state machine syntax is that it clearly documents the operation of the design.

In Example 4, state 0 (binary value 0000) is defined, because it is the result of the **clr** signal. It is recommended that all designs have a valid 0000 defined to avoid being stuck at state 0. For example, in this design, if a state that hasn't been

defined occurs at power-on, such as hexadecimal A-F, none of the conditions described in the equations is met, so the state goes to state 0 (hex value 0000). If 0000 has not been defined as a valid state, the counter stays at state 0.

❑ EXAMPLE 5. SEVEN-SEGMENT DISPLAY DECODER

This example shows a hexadecimal-to-seven-segment decoder for driving common-anode LEDs. The design incorporates both a ripple-blanking input to inhibit the display of leading zeroes, and a ripple-blanking output for easy cascading of digits

Figure U5-27 shows the segment display decoder.

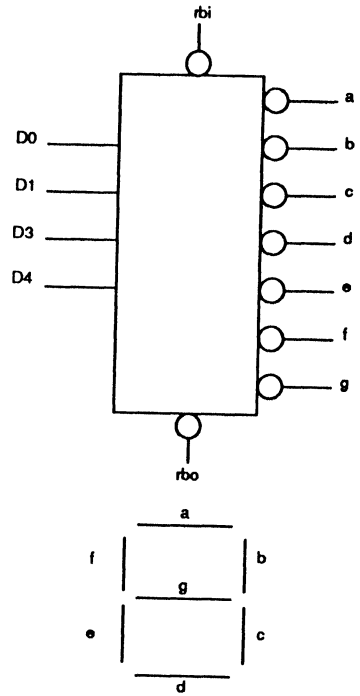


Figure U5-27. Seven-Segment Display Decoder

The segments in the display, labeled a-g, correspond to the outputs in the diagram.

Figure U5-28 shows the source file (**HEXDISP.PLD**, provided with the CUPL package).

```
                HEXDISP.PLD

Name                Hexdisp;
Partno              CA0007;
Revision            02;
Date                07/16/87;
Designer            T. Kahl;
Company             ATI;
Location            None;
Assembly            None;

/*************************************************************************/
/*                                                                       */
/*                                                a                      */
/* This is a hexadecimal-to-seven-segment          -----                 */
/* decoder capable of driving  common-anode       |     |                */
/* LEDs. It incorporates both a ripple-          f|     |b               */
/* blanking input (to inhibit displaying          |  g  |                */
/* leading zeroes) and a ripple blanking output    -----                 */
/* to allow for easy cascading of digits          |     |                */
/*                                               e|     |c               */
/*                                                |     |                */
/*                                                 -----                 */
/*                                                  d                    */
/*                                                                       */
/*************************************************************************/
/* Allowable Target Device Types: 32 x 8 PROM (82S123 or
   equivalent                                                            */
/*************************************************************************/
/**  Input group (Note this is only a comment)                          **/

pin [10..13] = [D0..3];          /* data input lines to display          */
pin 14 = !rbi;                   /* ripple blanking input                */

/** Output Group  ( Note this is only a comment )                       **/

pin [7..1] = ![a,b,c,d,e,f,g];   /* Segment output lines                 */
pin 9      = !rbo;               /* Ripple Blanking output               */

/**  Declarations and Intermediate Variable Definitions                 */
field data = [D3..0];            /* hexadecimal input field              */
field segment=[abcdefg];         /* Display segment field                */
$define ON  'b'1                 /* segment lit when logically    "ON" */
$define OFF 'b'0                 /* segment dark when logically "OFF"  */
```

Figure U5-28. Display Decoder Source File (HEXDISP.PLD) Sheet 1 of 2

```
/**  Logic Equations   **/
      /*       a     b     c     d     e     f     g  */
segment =
/* 0 */      [ ON,   ON,   ON,   ON,   ON,   ON,  OFF] & data:0 & !rbi
/* 1 */    # [OFF,   ON,   ON,  OFF,  OFF,  OFF,  OFF] & data:1
/* 2 */    # [ ON,   ON,  OFF,   ON,   ON,  OFF,   ON] & data:2
/* 3 */    # [ ON,   ON,   ON,   ON,  OFF,  OFF,   ON] & data:3
/* 4 */    # [OFF,   ON,   ON,  OFF,  OFF,   ON,   ON] & data:4
/* 5 */    # [ ON,  OFF,   ON,   ON,  OFF,   ON,   ON] & data:5
/* 6 */    # [ ON,  OFF,   ON,   ON,   ON,   ON,   ON] & data:6
/* 7 */    # [ ON,   ON,   ON,  OFF,  OFF,  OFF,   ON] & data:7
/* 8 */    # [ ON,   ON,   ON,   ON,   ON,   ON,  OFF] & data:8
/* 9 */    # [ ON,   ON,   ON,   ON,  OFF,   ON,   ON] & data:9
/* A */    # [ ON,   ON,   ON,  OFF,   ON,   ON,   ON] & data:A
/* B */    # [OFF,  OFF,   ON,   ON,   ON,   ON,   ON] & data:B
/* C */    # [ ON,  OFF,  OFF,   ON,   ON,   ON,  OFF] & data:C
/* D */    # [OFF,   ON,   ON,   ON,   ON,  OFF,   ON] & data:D
/* E */    # [ ON,  OFF,  OFF,   ON,   ON,   ON,   ON] & data:E
/* F */    # [ ON,  OFF,  OFF,  OFF,   ON,   ON,   ON] & data:F;

rbo = rbi & data:0;
```

Figure U5-28. Sheet 2 of 2

The first part of the file provides archival information and a description of the intended function of the design, including compatible PLDs.

Pin declarations are made corresponding to the inputs and outputs in the design diagram.

In the "Declarations and Intermediate Variables" section, field assignments are made to group the input pins into a set named data and the output pins into a set named segment. ON and OFF are defined respectively as binary 1 and binary 0.

The logic equations are set up as a function table to describe the segments that are lit up by each input pattern. Comments create a header for the function table, listing the output segments across the top and the input numbers vertically down the side.

Each line of the table describes a decoded hex value and the segments of the display that the hex value turns on or off. For example, the line for an input value of 4 is written as follows:

[OFF, ON, ON, OFF, OFF, ON, ON] & data:4

The function table format expresses the intent of the design more clearly than equations; that is, the example above shows that an input value of 4 turns segment a off, segment b on, segment c on, and so on.

CUPL Language 1

This chapter explains CUPL language elements and CUPL language syntax.

❑ LANGUAGE ELEMENTS

This section describes the elements that comprise the CUPL logic description language.

❑ Variables

Variables are strings of alphanumeric characters that specify device pins, internal nodes, constants, input signals, output signals, intermediate signals, or sets of signals. This section explains the rules for creating variables.

Variables can start with a numeric digit, alphabet character, or underscore, but must contain at least one alphabet character.

Variables are case sensitive; that is, they distinguish between uppercase and lowercase letters.

Do not use spaces within a variable name. Use the underscore character to separate words.

Variables can contain up to 31 characters. Longer variables are truncated to 31 characters.

Variables cannot contain any of the CUPL reserved symbols (see Table 1-2).

Variables cannot be the same as a CUPL reserved keyword (see Table 1-1).

Examples of some valid variable names are:

a0
A0
8250_ENABLE
Real_time_clock_interrupt
_address

Note how the use of the underscore in the above examples makes the variable names easier to read. Also, note the difference between uppercase and lowercase variable names. The variable **A0** is not the same as **a0**.

Examples of some invalid variable names are:

99	does not contain an alpha character
I/O enable	contains a special character (/)
out 6a	contains a space; the system reads it as two separate variables
tbl-2	contains a dash; the system reads it as two variables.

❑ Indexed Variables

Variable names can be used to represent a group of address lines, data lines, or other sequentially numbered items. For example, the following variable names could be assigned to the eight LO-order address lines of a microprocessor:

A0 A1 A2 A3 A4 A5 A6 A7

Variable names that end in a number, as shown above, are referred to as indexed variables.

The index numbers are always decimal numbers between 0 and 31. When used in bit field operations (see the subtopic, Bit Field Declaration Statements in this chapter) the variable with index number 0 is always the lowest order bit.

Examples of some valid indexed variable names are as follows:

a23
D07
D7
counter_bit_3

Note the difference between index variables with leading zeroes; the variable **D07** is not the same as **D7**.

Examples of some invalid indexed variable names are as follows:

D0F	**index number is not decimal**
a36	**index number out of range**

❑ Reserved Words and Symbols

CUPL uses certain character strings with predefined meanings called **keywords**. These keywords cannot be used as names in CUPL. Table 1-1 lists these keywords.

Table 1-1. CUPL Reserved Keywords

APPEND	FORMAT	PARTNO
ASSEMBLY	FUNCTION	PIN
ASSY	FUSE	PINNODE
COMPANY	IF	PRESENT
CONDITION	JUMP	REV
DATE	LOC	REVISION
DEFAULT	LOCATION	SEQUENCE
DESIGNER	MACRO	SEQUENCED
DEVICE	MIN	SEQUENCEJK
ELSE	NAME	SEQUENCERS
FIELD	NODE	SEQUENCET
FLD	OUT	TABLE

CUPL also reserves certain **symbols** for its use that cannot be used in variable names. Table 1-2 lists these reserved symbols.

Table 1-2. CUPL Reserved Symbols

&	#	( )	-
*	+	[ ]	/
:	.	..	/* */
;	,	!	' =
@	$	^	

❑ Numbers

All operations involving numbers in the CUPL compiler are done with 32-bit accuracy. Therefore, the numbers may have a

value from 0 to 2^{32} -1. Numbers may be represented in any one of the four common bases: binary, octal, decimal, or hexadecimal. The default base for all numbers used in the source file is hexadecimal, except for device pin numbers and indexed variables, which are always decimal. Numbers for a different base may be used by preceding them with a prefix listed in Table 1-3.

Table 1-3. Number Base Prefixes

Base Name	Base	Prefix
Binary	2	'b'
Octal	8	'o'
Decimal	10	'd'
Hexadecimal	16	'h'

The base letter is enclosed in single quotes and can be either uppercase or lowercase. Some examples of valid number specifications are listed in Table 1-4.

Table 1-4. Sample Base Conversions

Number	Base	Decimal Value
'b'0	Binary	0
'B'1101	Binary	13
'O'663	Octal	435
'D'92	Decimal	92
'h'BA	Hexadecimal	186
'O'[300..477]	Octal (range)	192..314

Binary, octal, and hexadecimal numbers can have don't-care values ("X") and numerical values. Some examples of valid number specifications with don't-care values are listed in Table 1-5.

Table 1-5. Sample Don't-Care Numbers

Number	Base

'b'1X11	Binary
'O'0X6	Octal
'H'[3FXX..7FFF]	Hexadecimal (range)

❑ Comments

Comments are an important part of the logic description file. They improve the readability of the code and document the intentions, but do not significantly affect the compile time, as they are removed by the preprocessor before any syntax checking is done. Use the symbols /* and */ to enclose comments; the program ignores everything between these symbols.

Comments may span multiple lines and are not terminated by the end of a line. Comments cannot be nested. Some examples of valid comments are shown in Figure 1-1.

```
/*****************************************/
/*  This is one way to create a title or  */
/*         an information block           */
/*****************************************/

/*
This is another way to create an information block
*/

out1=in1 # in2;        /* A Simple OR Function  */
out2=in1 & in2;        /* A Simple AND Function */
out3=in1 $ in2;        /* A Simple XOR Function  */
```

Figure 1-1. Sample Comments

❑ List Notation

Shorthand notations are an important feature of the CUPL language.

The most frequently used shorthand notation is the list. It is commonly used in pin and node declarations, bit field declarations, logic equations, and set operations. The list format is as follows:

[variable, variable, ... variable]

where

[ ] are brackets used to delimit items in the list as a set of variables.

Two examples of the list notation are as follows:

[UP, DOWN, LEFT, RIGHT]
[A0, A1, A2, A3, A4, A5, A6, A7]

When all the variable names are sequentially numbered, either from lowest to highest or vice versa, the following format may be used:

[variablem..n]

where

m is the first index number in the list of variables.

n is the last number in the list of variables; n can be written without the variable name.

For example, the second line from the example above could be written as follows:

[A0..7]

Index numbers are assumed to be decimal and contiguous. Any leading zeros in the variable index are removed from the variable name that is created. For example:

[A00..07]

is shorthand for:

[A0, A1, A2, A3, A4, A5, A6, A7]

not for:

[A00, A01, A02, A03, A04, A05, A06, A07]

The two forms for the list notation may be mixed in any combination. For example, the following two list notations are equivalent:

[A0..2, A3, A4, A5..7]

[A0, A1, A2, A3, A4, A5, A6, A7]

❑ Template File

When a logic description source file is created using the CUPL language, certain information must be entered, such as header information, pin declarations, and logic equations. As an assist, CUPL provides a template file that contains the proper structure for the source file. Figure 1-2 shows the contents of the template file.

```
Name        XXXXX;
Partno      XXXXX;
Date
Revision
Designer
Company
Assembly
Location
/*************************************************************/
/*                                                           */
/*                                                           */
/*                                                           */
/*************************************************************/
/*    Allowable Target Device                                */
/*************************************************************/
/* Inputs */
Pin             =           ;   /*                           */
Pin             =           ;   /*                           */
Pin             =           ;   /*                           */
Pin             =           ;   /*                           */
Pin             =           ;   /*                           */
Pin             =           ;   /*                           */
Pin             =           ;   /*                           */
Pin             =           ;   /*                           */
Pin             =           ;   /*                           */
Pin             =           ;   /*                           */
Pin             =           ;   /*                           */
Pin             =           ;   /*                           */
Pin             =           ;   /*                           */
Pin             =           ;   /*                           */
Pin             =           ;   /*                           */

/* outputs */
Pin             =           ;   /*                           */
Pin             =           ;   /*                           */
Pin             =           ;   /*                           */
Pin             =           ;   /*                           */
Pin             =           ;   /*                           */
Pin             =           ;   /*                           */
Pin             =           ;   /*                           */
Pin             =           ;   /*                           */
```

Figure 1-2. Template File

The template file provides the following sections:

Header Information - Keywords followed by XXXs that are replaced with text to identify the file for archival and revision purposes.

Title Block - Comment symbols that enclose space for describing the function of the design and allowable target devices.

Pin Declaration - Keywords and operators in the proper format for input and output pin declarations and comment space to describe the pin assignments. After pin declarations are made, remove any extra "**pin = ;**" lines. Otherwise, a syntax error will occur during compilation.

The **/* Inputs */** and **/* Outputs */** are comments that provide groupings for readability only. Assign any pin type in any order, no matter how it is used in the logic description file.

Declaration and Intermediate Variable - Space for making declarations, such as bit field declarations (see the subtopics, Bit Field Declaration Statements and Node Declaration Statements in this chapter) and for writing intermediate equations (see the subtopic, Logic Equations in this chapter).

Logic Equation - Space for writing logic equations describing the function of the device (see the subtopic, Logic Equations in this chapter).

Header Information

The header information section of the source file identifies the file for revision and archival purposes. Normally place it at the beginning of the file. CUPL provides 10 keywords to use in header information statements. Begin each statement with a keyword which can be followed by any valid ASCII characters,

including spaces and special characters. End each statement with a semicolon. Table 1-6 lists the CUPL header keywords and the information to provide with each keyword.

Table 1-6. Header Information Keywords

Keyword	Information
NAME	Normally use the source logic description filename. Use only character strings that are valid for the operating system. The name specified here determines the name for any JEDEC, ASCII - hex, or HL download files. The NAME field accommodates filenames up to 32 characters long. When using systems such as DOS which allow filenames of only eight characters, the filename will be truncated.
PARTNO	Specify a company's proprietary part number (usually issued by manufacturing) for a particular PLD design. The part number is not the type of target PLD. For GAL devices, the first eight characters are encoded using seven-bit ASCII in the User Signature Fuses of the devices' fuse map.
REVISION	Begin with 01 when first creating a file and increment each time a file is altered. REV can be used for an abbreviation.
DATE	Change to the current date each time a source file is altered.
DESIGNER	Specify the designer's name.
COMPANY	Specify the company's name for proper documentation practice and because specifications may be sent to semiconductor manufacturers for high volume PLD orders.
ASSEMBLY	Give the assembly name or number of the PC board on which the PLD will be used. The abbreviation ASSY can be used.
LOCATION	Indicate the PC board reference or coordinate where the PLD is located. The abbreviation LOC can be used.

DEVICE — Set the default device type for the compilation. A device type specified on the command line overrides all device types set in the source file. For multi-device source files, DEVICE must be used with each section if the device types are different.

FORMAT — Set a download output format override for the current logic description section. The valid values to use for the output format are:

h produce ASCII-hex output
i produce Signetics HL output
j produce JEDEC output

FORMAT overrides any option flag on the command line. It is useful in multi-device source files where different parts have incompatible output formats. More than one format value at a time may be specified to produce more than one type of output. The format value must be a lowercase letter.

The template file provides all the header keywords except **DEVICE** and **FORMAT**. An example of proper CUPL header information is as follows:

```
Name        WAITGEN ;
Partno      P9000183;
Revision    02;
Date        1/11/89;
Designer    Osann ;
Company     Logical Devices, Inc. ;
Assembly    PC Memory Board ;
Location    U106;
Device      F155;
Format      ij ;
```

If any header information is omitted, CUPL issues a warning message, but continues with compilation.

Pin Declaration Statements

Pin declaration statements declare the pin numbers and assign them symbolic variable names. The format for a pin declaration is as follows:

PIN pin_n=[!]var ;

where

PIN is a keyword to declare the pin numbers and assign them variable names.

pin_n is a decimal pin number or a list of pin numbers grouped using the list notation; that is,

[pin_n 1, pin_n 2 ... pin_nn]

! is an optional exclamation point to define the polarity of the input or output signal.

= is the assignment operator.

var is a single variable name or a list of variables grouped using the list notation; that is,

[var, var ... var]

; is a semicolon to mark the end of the pin declaration statement.

The template file provides a section for entering the pin variables individually or in groups using the list notation.

Use the exclamation point (!) to define the polarity of an input or output signal. If an input signal is active-level LO (that is, the asserted TTL signal voltage level is 0 volts), put an exclamation point before the variable name in the pin declaration. The exclamation point informs the compiler to

choose the inverted sense of the signal when it is listed as active in the logic equations.

Similarly, if an output signal is active-level LO, define the variable with an exclamation point in the pin declaration and write the logic equation in a logically true form. The exclamation point permits declaring pins without regard to the limitations of the type of target device.

If a pin declaration specifying an active-level HI output is compiled for a target device (such as a PAL16L8) that has only inverting outputs, CUPL automatically performs DeMorgan's Theorem on the logic equation to fit the function into the device.

Consider the following example. The logic description file is written for a PAL16L8 device. All output pins are declared as active-HI. The following equation has been written to specify an OR function:

```
c = a # b;
```

However, because the PAL16L8 contains a fixed inverting buffer, CUPL must perform a DeMorgan to fit the logic to the device. CUPL generates the following product term in the documentation file (see Documentation File Formats in Appendix C):

```
c => !a & !b
```

Figure 1-3 shows the process described above.

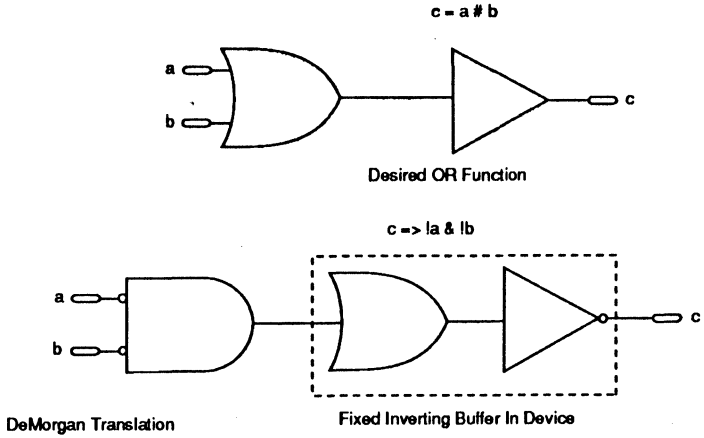


Figure 1-3. Active-HI Pin Declaration for Inverting Buffer

If excess product terms are generated, CUPL displays an error message and the compilation stops. The documentation file (***filename*.DOC**) lists the number of product terms required to implement the logic function and the number of product terms the device physically has for the particular output pin.

Some examples of valid pin declarations are:

```
pin 1       =       clock;      /* Register Clock  */
pin 2       =       !enable;    /* Enable I/O Port */
pin [3,4]   =       ![stop,go]; /* Control Signals  */
in [5..7]   =       [a0..2];    /* (Address Bit 0-2 */
```

The last two lines in the example above are shorthand notations for the following:

```
pin 3   =       !stop;  /* Control Signal */
pin 4   =       !go;    /* Control Signal */
pin 5   =       a0;     /* Address Bit 0  */
pin 6   =       a1;     /* Address Bit 1  */
pin 7   =       a2;     /* Address Bit 2  */
```

The input, output, or bi-directional nature of a device pin is not specified in the pin declaration. The compiler infers the nature of a pin from the way the pin variable name is used in the logic specification. If the logic specification and the physical characteristics of the target device are incompatible, CUPL displays an error message denoting the improper use of the pin.

Node Declaration Statements

Some devices contain functions that are not available on external pins, but logic equations must be written for these capabilities. For example, the 82S105 contains both buried state registers (flip-flops) and a mechanism for inverting any transition term through a complement array. Before writing equations for these flip-flops (or complement arrays), they must be assigned variable names. Since there are no pins associated with these functions, the **PIN** keyword cannot be used. Use the **NODE** keyword to declare variable names for buried functions.

The format for node declarations is as follows:

NODE [!] var ;

where

NODE is a keyword to declare a variable name for a buried function.

! is an optional exclamation point to define the polarity of the internal signal.

var is a single variable name or list of variables grouped using the list notation.

; is a semicolon to mark the end of the statement.

Place node declarations in the "Declarations and Intermediate Variables Definitions" section of the source file provided by the template file.

Most internal nodes are active-level HI; therefore, the exclamation point should not be used to define the polarity of an internal signal as active-level LO. Using the exclamation point almost always causes the compiler to generate a significantly greater number of product terms. An exception is the complement array node, which, by definition, is an active-level LO signal.

Although no pin numbers are given in the declaration statement, CUPL assigns the variable name to an internal pseudo-pin number. The assignment is automatic and determined by usage (flip-flop, complement array, and so on), so variable order is not a concern. However, once a node variable is declared, a logic equation must be created for the variable, or a compilation error results.

CUPL uses the node declaration to distinguish between a logic equation for a buried function and an intermediate expression.

Examples of the use of the **NODE** keyword are:

```
NODE [State0..5]; /* Internal State Bits        */
NODE !Invert;     /* For Complement Array      */
```

An alternative for assigning buried functions instead of allowing CUPL to automatically assign them via the **NODE** keyword, is to use the **PINNODE** keyword. The **PINNODE** keyword is used for explicitly defining buried nodes by assigning a node number to a symbolic variable name. This is similar to the way the pin declaration statements work. The format for a pinnode declaration is as follows:

```
PINNODE node_n = [!]var;
```

where

PINNODE is a keyword to declare the node numbers and assign them variable names.

node_n is a decimal node number or a list of node numbers grouped using the list notation; that is,

[node_n1,node_n2 ... node_nn]

! is an optional exclamation point to define the polarity of the internal signal.

= is the assignment operator.

var is a single variable name or list of variables grouped using the list notation; that is,

[var,var ... var]

; is a semicolon used to mark the end of the statement.

Place pinnode declarations in the "Declarations and Intermediate Variables Definitions" section of the source file provided by the template file.

As with node declarations, most internal nodes are active-level HI; therefore, the exclamation point should not be used to define the polarity of an internal signal as active level LO. Using the exclamation point almost always causes the compiler to generate a significantly greater number of product terms. An exception is the complement array node, which by definition is an active-level LO signal.

A list of node numbers for all devices containing internal nodes is included in Appendix D. Please reference these node numbers for pinnode declarations.

Examples of the use of the **PINNODE** keyword are:

```
PINNODE [29..34] = [State0..5]; /* Internal State Bits */
PINNODE 35 = !Invert;           /* For Complement Array */
PINNODE 25 = Buried;            /* For Buried register part */
                                /* of an I/O macrocell with */
                                /* multiple feedback paths */
```

Bit Field Declaration Statements

A bit field declaration assigns a single variable name to a group of bits. The format is as follows:

FIELD var = [var, var, ... var] ;

where

FIELD is a keyword.

var is any valid variable name.

[var, var, ... var] is a list of variable names in list notation.

= is the assignment operator.

; is a semicolon used to mark the end of the statement.

Note

The square brackets do not indicate optional items but are used to delimit items in a list.

Place bit field declarations in the "Declarations and Intermediate Variable Definitions" section of the source file provided by the template file.

After assigning a variable name to a group of bits, the name can be used in an expression; the operation specified in the expression is applied to each bit in the group. See the subtopic, Set Operations in this chapter for a description of the operations allowed for **FIELD** statements. The example below shows two ways to reference the eight address input bits (A0 through A7) of an I/O decoder as the single variable named ADDRESS.

FIELD ADDRESS = [A7,A6,A5,A4,A3,A2,A1,A0] ;

or

FIELD ADDRESS = [A7..0] ;

MIN Declaration Statements

The **MIN** declaration statement overrides, for specified variables, the minimization level specified on the command line when running CUPL. The format is as follows:

MIN var [.ext] = level ;

where

MIN is a keyword to override the command line minimization level.

var is a single variable declared in the file or a list of variables grouped using the list notation; that is,

[var, var, ... var]

.ext is an optional extension that identifies the function of the variable.

level is an integer between 0 and 4.

; is a semicolon to mark the end of the statement.

The levels 0 to 4 correspond to the option flags on the command line, **-m0** through **-m4**.

The **MIN** declaration permits specifying different levels for different outputs in the same design, such as no reduction for outputs requiring redundant or contained product terms (to avoid asynchronous hazard conditions), and maximum reduction for a state machine application.

The following are examples of valid **MIN** declarations.

```
MIN async_out      = 0;   /* no reduction      */
MIN [outa, outb]   = 2;   /* level 2 reduction */
MIN count.d        = 4;   /* level 4 reduction */
```

Note that the last declaration in the example above uses the **.d** extension to specify that the registered output variable is the one to be reduced.

FUSE Statement

The **FUSE** statement provides for special cases where it is necessary to blow TURBO or MISER bits. This statement should be used with utmost care, as it can lead to unpredictable results if used incorrectly.

FUSE (fusenumber, x)

where **fusenumber** is the fuse number corresponding to the MISER Bit or TURBO Bit that must be blown, and x is either 0 or 1. Specify 0 if the bit must not be blown. Specify 1 to blow the bit. **Use this statement with extreme caution.**

In this example, fuse 101 is a MISER Bit or TURBO Bit. This blows fuse number 101.

example:

FUSE(101,1)

DO NOT ATTEMPT TO USE THIS STATEMENT TO BLOW ARBITRARY FUSES!

The fuse statement was designed to blow MISER bits and TURBO Bits only. The exact fuse number for the TURBO or MISER Bit must be specified. Every time this statement is used, CUPL will generate a warning. This is a reminder to double check that the fuse number specified is correct. If a wrong fuse number is specified, disastrous results can occur. Be very careful using this statement. If the **FUSE** statement is

used in a design and strange results occur, check the fuse number specified and make sure that it is a MISER or TURBO Bit.

❑ Preprocessor Commands

The preprocessor portion of CUPL operates on the source file before it is passed to the parser and other sections of the compiler. The preprocessor commands add file inclusion, conditional compilation, and string substitution capabilities to the source processing features of CUPL. Table 1-7 lists the available preprocessor commands. Each command is described in detail in this section.

Table 1-7. Preprocessor Commands

$DEFINE	$IFDEF	$UNDEF
$ELSE	$IFNDEF	$REPEAT
$ENDIF	$INCLUDE	$REPEND
$MACRO	$MEND	

The dollar sign ($) is the first character in all preprocessor commands and must be used in column one of the line. Any combination of uppercase or lowercase letters may be used to type these commands.

$DEFINE

This command replaces a character string by another specified operator, number, or symbol. The format is as follows:

$DEFINE argument1 argument2

where

argument1 is a variable name or special ASCII character.

argument2 is a valid operator, a number, or a variable name.

"Argument1" is replaced by "argument2" at all locations in the source specification after the **$DEFINE** command is given (or until the preprocessor encounters an **$UNDEF** command). The replacement is a literal string substitution made on the input file before being processed by the CUPL compiler. Note that no semicolon or equal sign is used for this command.

The **$DEFINE** command allows numbers or constants to be replaced with symbolic names, for example:

```
$DEFINE ON    'b'1
$DEFINE OFF   'b'0
$DEFINE PORTC   'h'3F0
```

The **$DEFINE** command also allows creation of a personal set of logical operators. For example, the following define an alternate set of operators for logic specification:

```
$DEFINE     {     /*  Alternate Start Comment
$DEFINE     }     */  Alternate End Comment
$DEFINE     /     !   Alternate Negation
$DEFINE     *     &   Alternate AND
$DEFINE     +     #   Alternate OR
$DEFINE     :+:   $   Alternate XOR
```

Note

The above definitions are contained in the **PALASM.OPR** file included with the CUPL software package. This file may be included in the source file (see **$INCLUDE** command) to allow logic equations using the PALASM set of logical operator symbols, as well as the standard CUPL operator symbols.

$UNDEF

This command reverses a **$DEFINE** command. The format is as follows:

$UNDEF argument

where

argument is an argument previously used in a **$DEFINE** command.

Before redefining a character string or symbol defined with the **$DEFINE** command, use the **$UNDEF** command to undo the previous definition.

$INCLUDE

This command includes a specified file in the source to be processed by CUPL. The format is as follows:

$INCLUDE filename

where

filename is the name of a file in the current directory.

File inclusion allows standardizing a portion of a commonly used specification. It is also useful for keeping a separate parameter file that defines constants that are commonly used in many source specifications. The files that are included may also contain **$INCLUDE** commands, allowing for "nested" include files. The named file is included at the location of the **$INCLUDE** command.

For example, the following command includes the **PALASM.OPR** file in a source file.

$INCLUDE PALASM.OPR

PALASM.OPR is included with the CUPL software and contains **$DEFINE** commands that specify the following alternate set of logical operators.

```
$DEFINE     /     !   Alternate Negation
$DEFINE     *     &   Alternate AND
$DEFINE     +     #   Alternate OR
$DEFINE     :+:   $   Alternate XOR
$DEFINE     [     /*  Alternate Start Comment
$DEFINE     ]     */  Alternate End Comment
```

$IFDEF

This command conditionally compiles sections of a source file. The format is as follows:

$IFDEF argument

where

argument may or may not have previously been defined with a **$DEFINE** command.

When the argument has previously been defined, the source statements following the **$IFDEF** command are compiled until the occurrence of an **$ELSE** or **$ENDIF** command.

When the argument has not previously been defined, the source statements following the **$IFDEF** command are

ignored. No additional source statements are compiled until the occurrence of an **$ELSE** or **$ENDIF** command.

One use of **$IFDEF** is to temporarily remove source equations containing comments from the file. It is not possible to "comment out" the equations because comments do not nest. The following example illustrates this technique. NEVER is an undefined argument.

```
$IFDEF NEVER
out1=in1 & in2;   /* A Simple AND Function */
out2=in3 # in4;   /* A Simple OR Function */
$ENDIF
```

Because NEVER is undefined, the equations are ignored during compilation; that is, they function as comments.

$IFNDEF

This command sets conditions for compiling sections of the source file.

$IFNDEF argument

where

argument may or may not have previously been defined with a **$DEFINE** command.

The **$IFNDEF** command works in the opposite manner of the **$IFDEF** command. When the argument has not previously been defined, the source statements following the **$IFNDEF** command are compiled until the occurrence of an **$ELSE** or **$ENDIF** command.

If the argument has previously been defined, the source statements following the **$IFNDEF** command are ignored. No

additional source statements are compiled until the occurrence of an **$ELSE** or **$ENDIF** command.

One use of **$IFNDEF** is to create a single source file containing two mutually exclusive sets of equations. Using an **$IFNDEF** and **$ENDIF** command to set off one of the sets of equations, quick toggling is possible between the two sets of equations by defining or not defining the argument specified in the **$IFNDEF** command.

For example, some devices contain common output enable pins that directly control all the tri-state buffers, whereas other devices contain single product terms to enable each tri-state buffer individually. In the following example, the argument, **COMMON_OE** has not been defined, so the equations that follow are compiled. Any equations following **$ENDIF** are not compiled.

```
$IFNDEF COMMON_OE
pin 11 = !enable;              /* input pin for OE     */
[q3,q2,q1,q0].oe = enable;     /* assign tri-state     */
                               /* equation for 4       */
                               /* outputs              */
$ENDIF
```

If the device has common output enables, no equations are required to describe it. Therefore, in the above example, for a device with common output enables, define **COMMON_OE** so the compiler skips the equations between **$IFNDEF** and **$ENDIF**.

$ENDIF

This command ends a conditional compilation started with the **$IFDEF** or **$IFNDEF** commands. The format is as follows:

```
$ENDIF
```

The statements following the **$ENDIF** command are compiled in the same way as the statements preceding the **$IFDEF** or **$IFNDEF** commands. Conditional compilation may be nested, and for each level of nesting of the **$IFDEF** or **$IFNDEF** command, an associated **$ENDIF** must be used.

The following example illustrates the use of **$ENDIF** with multiple levels of nesting.

```
$IFDEF    prototype_1
pin 1     = set;        /* Set on pin 1      */
pin 2     = reset;      /* Reset on pin 2    */
$IFDEF    prototype_2
pin 3     = enable;     /* Enable on pin 3   */
pin 4     = disable;    /* Disable on pin 4  */
$ENDIF
pin 5     = run;        /* Run on pin 5      */
pin 6     = halt;       /* Halt on pin 6     */
$ENDIF
```

$ELSE

This command reverses the state of conditional compilation as defined with **$IFDEF** or **$IFNDEF**. The format is as follows:

```
$ELSE
```

If the tested condition of the **$IFDEF** or **$IFNDEF** commands is true (that is, the statements following the command are compiled), then any source statements between an **$ELSE** and **$ENDIF** command are ignored.

If the tested condition is false, then any source statements between the **$IFDEF** or **$IFNDEF** and **$ELSE** command are ignored, and statements following **$ELSE** are compiled.

For example, many times the production printed circuit board uses a different pinout than does the wire-wrap prototype. In the following example, since Prototype has been defined, the source statements following **$IFDEF** are compiled and the statements following **$ELSE** are ignored.

```
$DEFINE Prototype X              /* define Prototype      */
$IFDEF Prototype
pin 1          = memreq;         /* memory request on*/
                                 /* pin 1 of prototype    */
pin 2          = ioreq;          /* I/O request on        */
                                 /* pin 2 of prototype    */
$ELSE
pin 1          = ioreq;          /* I/O request on        */
                                 /* pin 1 of PCB          */
pin 2          = memreq;         /* memory request on*/
                                 /* pin 2 of PCB          */
$ENDIF
```

To compile the statements following **$ELSE**, remove the definition of Prototype.

$REPEAT

This command is similar to the FOR statement in C language and DO statements in FORTRAN language. It allows the user to duplicate repeat body by index. The format is as follows:

```
$REPEAT index=[number1,number2,...numbern]
    repeat body
$REPEND
```

where **n** can be any number in the range 0 to 1023

In preprocessing, the repeat body will be duplicated from $number_1$ to $number_n$. The index number can be written in short form as [number1..numbern] if the number is consecutive. The repeat body can be any CUPL statement.

Arithmetic operations can be performed in the repeat body. The arithmetic expression must be enclosed by braces { }.

For example, design a three to eight decoder.

```
FIELD sel = [in2..0]
$REPEAT i = [0..7]
    !out{i} = sel:'h'{i} &enable;
$REPEND
```

Where index variable i goes from 0 to 7, so the statement "out{i} = sel:'h'{i} &enable;" will be repeated during preprocessing and create the following statements:

```
FIELD sel = [in2..0];
    !out0 = sel:'h'0 &enable;
    !out1 = sel:'h'1 &enable;
    !out2 = sel:'h'2 &enable;
    !out3 = sel:'h'3 &enable;
    !out4 = sel:'h'4 &enable;
    !out5 = sel:'h'5 &enable;
    !out6 = sel:'h'6 &enable;
    !out7 = sel:'h'7 &enable;
```

The following example shows how the arithmetic operation addition (+) and modulus (%) are used in the repeat body.

Design a five bit counter with a control signal advance, if advance is high, counter is increased by one.

```
FIELD count[out4..0]
SEQUENCE count {
$REPEAT i = [0..31]
    PRESENT S{i} IF advance & !reset NEXT
    S{(i+1)%(32)};
    IF reset NEXT S{0};
    DEFAULT NEXT S{i};
$REPEND
```

$REPEND

This command ends a repeat body that was started with **$REPEAT**. The format is as follows:

$REPEND

The statements following the **$REPEND** command are compiled in the same way as the statements preceding the **$REPEAT** command. For each **$REPEAT** command, an associated **$REPEND** command must be used.

$MACRO

This command creates user-defined macros. The format is as follows:

$MACRO name argument$_1$ argument$_2$...argument$_n$
macro function body
$MEND

When macros are invoked, the keyword **NC** is used to represent no connection. Because NC is a keyword, the letters NC should not be used as a variable elsewhere in CUPL.

The macro function body will not be compiled until the macro name is called. The function is called by stating function name and passing the parameters to the function.

Like the **$REPEAT** command, the arithmetic operation can be used inside the macro function body and must be enclosed in braces.

The following example illustrates how to use the **$MACRO** command.

Use the **$MACRO** command to define a decoder function with an arbitrary number of bits. This example places the macro definition and call in the same file.

```
$MACRO decoder bits X Y E;
    FIELD select = [y{bits-1}..0];
    $REPEAT i = [0..{2**(bits-1)}]
            !X{i} = select:'h'{i} & E;
    $REPEND
$MEND

.../* Other statements */
decoder(3, out, in, enable); /*macro function call*/
```

Calling function decoder will create the following statements by macro expansion.

```
FIELD sel = [in2..0];
    !out0 = sel:'h'0 &enable;
    !out1 = sel:'h'1 &enable;
    !out2 = sel:'h'2 &enable;
    !out3 = sel:'h'3 &enable;
    !out4 = sel:'h'4 &enable;
    !out5 = sel:'h'5 &enable;
    !out6 = sel:'h'6 &enable;
    !out7 = sel:'h'7 &enable;
```

Use the flag **-e** when compiling the PLD file. CUPL will create an expanded file with the same name as the PLD file, with an extension **".mx"**.

The macro definition can be stored in a separate file with a **".m"** extension. Using the **$INCLUDE** command, specify the file. All the macro functions in that file will then be accessible. The following example shows the macro definition and calling statement stored in different files.

The macro definition of decoder is stored in the file "macrolib.m"

```
$INCLUDE macrolib.m /*specify the macro library */
.../* other statements */
decoder(4, out, in enable);
.../* other statements */
```

More examples can be found in the example files provided on diskette.

$MEND

This command ends a macro function body started with **$MACRO**. The format is as follows:

$MEND

The statements following the **$MEND** command are compiled in the same way as the statements preceding the **$MACRO** command. For each **$MACRO** command, an associated **$MEND** command.must be used.

❑ LANGUAGE SYNTAX

This section describes the CUPL language syntax. It explains how to use logic equations, truth tables, state machine syntax, condition syntax and user-defined functions to create a PLD design.

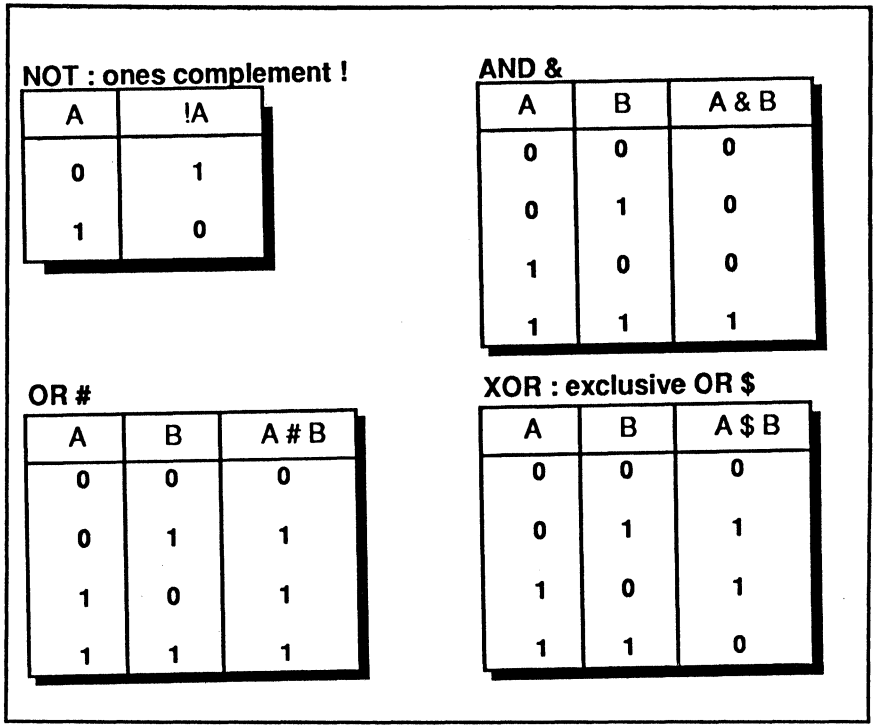
❑ Logical Operators

CUPL supports the four standard logical operators used for boolean expressions. Table 1-8 lists these operators and their order of precedence, from highest to lowest.

Table 1-8. Precedence of Logical Operators

Operator	Example	Description	Precedence
!	!A	NOT	1
&	A & B	AND	2
#	A # B	OR	3
$	A$B	XOR	4

The truth tables in Figure 1-4 list the Boolean Logic rules for each operator.



NOT : ones complement !

A	!A
0	1
1	0

AND &

A	B	A & B
0	0	0
0	1	0
1	0	0
1	1	1

OR #

A	B	A # B
0	0	0
0	1	1
1	0	1
1	1	1

XOR : exclusive OR $

A	B	A $ B
0	0	0
0	1	1
1	0	1
1	1	0

Figure 1-4. Truth Tables

❑ Arithmetic Operators

CUPL supports six standard arithmetic operators used for arithmetic expressions. The arithmetic expressions can only be used in the **$REPEAT** and **$MACRO** commands. Arithmetic expressions must appear in braces { }. Table 1-9 lists these operators and their order of precedence, from highest to lowest.

Table 1-9 Precedence of Arithmetic Operators

Operator	Example	Description	Precedence
* *	2**3	Exponentiation	1
*	2*i	Multiplication	2
/	4/2	Division	2
%	9%8	Modulus	2
+	2+4	Addition	3
-	4-i	Subtraction	3

❑ Arithmetic Function

CUPL supports one arithmetic function used for arithmetic expressions. The arithmetic expressions can only be used in the **$REPEAT** and **$MACRO** commands. Table 1-10 lists the function.

Table 1-10 Arithmetic Function

Function	Base
LOG2	Binary
LOG8	Octal
LOG16	Hexadecimal
LOG	Decimal

The LOG function returns a rounded integer value. For example:

LOG2(32) = 5 <==> 2**5 = 32

LOG2(33) = round(5.0444) = 6 <==> 2**6 = 64

❑ Extensions

Extensions can be added to variable names to indicate specific functions associated with the major nodes inside a programmable device, including such capabilities as flip-flop description and programmable three-state enables. Table 1-11 lists the extensions that are supported by CUPL and on which side of the equal sign (=) they are used. The compiler checks

the usage of the extension to determine whether it is valid for the specified device.

Table 1-11. Extensions

Extension	Side Used	Description
.D	L	D input of D-type flip-flop
.L	L	D input of transparent latch
.J	L	J input of JK-type flip-flop
.K	L	K input of JK-type flip-flop
.S	L	S input of SR-type flip-flop
.R	L	R input of SR-type flip-flop
.T	L	T input of toggle flip-flop
.DQ	R	Q output of D-type flip-flop
.LQ	R	Q output of transparent latch
.AP	L	Asynchronous preset of flip-flop
.AR	L	Asynchronous reset of flip-flop
.SP	L	Synchronous preset of flip-flop
.SR	L	Synchronous reset of flip-flop
.CK	L	Programmable clock of flip-flop
.OE	L	Programmable output enable
.CA	L	Complement array
.PR	L	Programmable preload
.CE	L	CE input of enabled D-CE type flip-flop
.LE	L	Programmable latch enable
.OBS	L	Programmable observability of buried nodes
.BYP	L	Programmable register bypass
.DFB	R	D registered feedback path selection
.LFB	R	D latched feedback path selection
.TFB	R	T registered feedback path selection
.IO	R	Pin feedback path selection
.INT	R	Internal feedback path selection

.CKMUX	L	Clock multiplexer selection
.OEMUX	L	Tri-state multiplexer selection
.TEC	L	Technology-dependent fuse selection
.IMUX	L	Input multiplexer selection of two pins
.T1	L	T1 input of 2-T flip-flop
.T2	L	T2 input of 2-T flip-flop
.IOD	R	Pin feedback path through D register
.IOL	R	Pin feedback path through latch
.IOCK	L	Clock for pin feedback register
.IOAR	L	Asynchronous reset for pin feedback register
.IOAP	L	Asynchronous preset for pin feedback register
.IOSR	L	Synchronous reset for pin feedback register
.IOSP	L	Synchronous preset for pin feedback register
.ARMUX	L	Asynchronous reset multiplexer selection
.APMUX	L	Asynchronous preset multiplexer selection
.LEMUX	L	Latch enable multiplexer selection

Each extension provides access to a specific function. For example, to specify an equation for output enable (on a device that has the capability) use the .OE extension. The equation will look as follows:

```
PIN  2  =  A;
PIN  3  =  B;
PIN  4  =  C;
PIN  15  =  VARNAME;
VARNAME.OE  =  A&B;
```

Note that the compiler supports only the flip-flop capabilities that are physically implemented in the device. For example, the compiler does not attempt to emulate a JK-type flip-flop in a device that only has D-type registers. Any attempt to use

capabilities not present in a device will cause the compiler to report an error.

For those devices containing bi-directional I/O pins with programmable output enables, CUPL automatically generates the output enable expression according to the usage of the pin. If the variable name is used on the left side of an equation, the pin is assumed to be an output and is assigned binary value 1; that is, the output enable expression is defaulted to the following:

```
PIN_NAME.OE='b'1;    /* Tri-state buffer    */
                     /* Always ON           */
```

Those pins that are used only as inputs (that is, the variable name appears only on the right side of an equation) are assigned binary value 0; the output enable expression is defaulted to the following:

```
PIN_NAME.OE = 'b'0;  /* Tri-state buffer
                        Always OFF   */
```

When the I/O pin is to be used as both an input and output, any new output enable expression that the user specifies overrides the default to enable the tri-state buffer at the desired time.

When using a JK or SR-type flip-flop, an equation must be written for both the J and K (or S and R) inputs. If the design does not require an equation for one of the inputs, use the following construct to turn off the input:

```
COUNT0.J='b'0 ; /* J input not used      */
```

Control functions such as asynchronous resets and presets are commonly connected to a group (or all) of the registers in a device. When an equation is written for one of these control functions, it is actually being written for all of the registers in the group. For documentation purposes, CUPL checks for the presence of such an equation for each register in the group and generates a warning message for any member of the group that does not have an identical equation. If all the control

functions for a given group are defined with different equations, the compiler will generate an error since it cannot decide which equation is the correct one. Remember that this is a device specific issue and it is a good idea to understand the capability of the device being used.

Figure 1-5 shows the use of extensions. Note that this figure does not represent an actual circuit, but shows how to use extensions to write equations for different functions in a circuit.

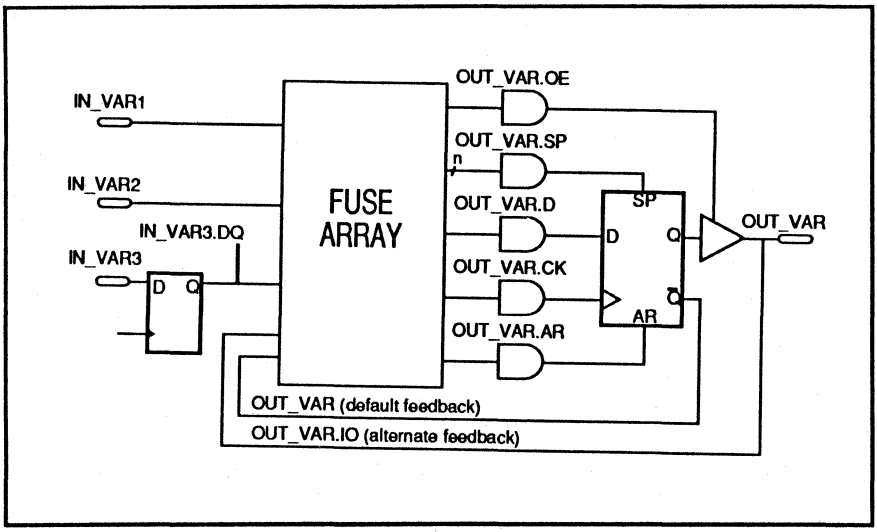


Figure 1-5. Circuit Illustrating Extensions

The figure shows an equation with a **.D** extension that has been written for the output to specify it as a registered output. Note that when feedback (**OUT_VAR**) is used in an equation, it does not have an extension.

The **.DQ** extension is used for input pins only.

Additional equations can be written to specify other types of controls and control points. For example, an equation for the output enable can be written as follows:

OUT_VAR.OE = IN_VAR1 # IN_VAR2

❑ Feedback Extensions Usage

Certain devices can program the feedback path. For example, the EP300 contains a multiplexer for each output that allows the feedback path to be selected as internal, registered, or pin feedback. Figure 1-6 shows the EP300 programmable feedback capability.

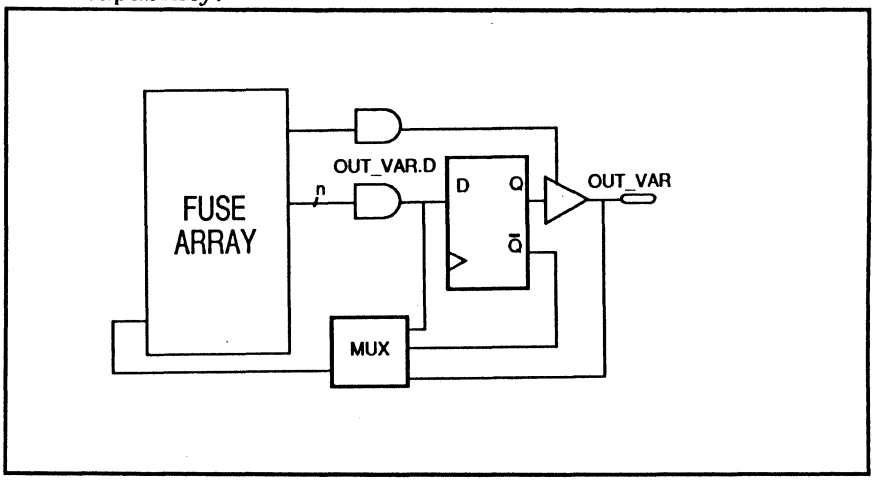


Figure 1-6. Programmable Feedback

CUPL automatically chooses a default feedback path according to the usage of the output. For example, if the output is used as a registered output, then the default feedback path will be registered, as in Figure 1-7. This default can be overridden by adding an extension to the feedback variables. For example, by adding the **.IO** extension to the feedback variables of a registered output, CUPL will select the pin feedback path. Figure 1-7 shows a registered output with pin feedback.

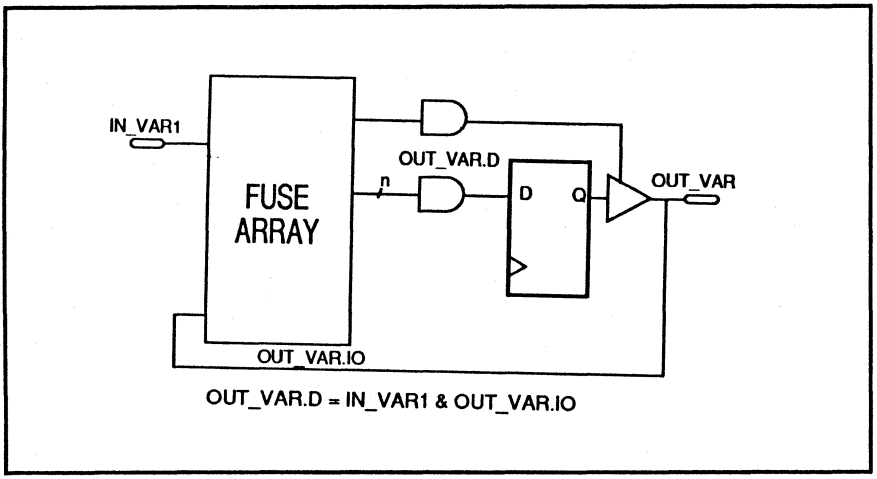


Figure 1-7. Programmable Pin (I/O) Feedback

Figure 1-8 shows a combinatorial output with registered feedback.

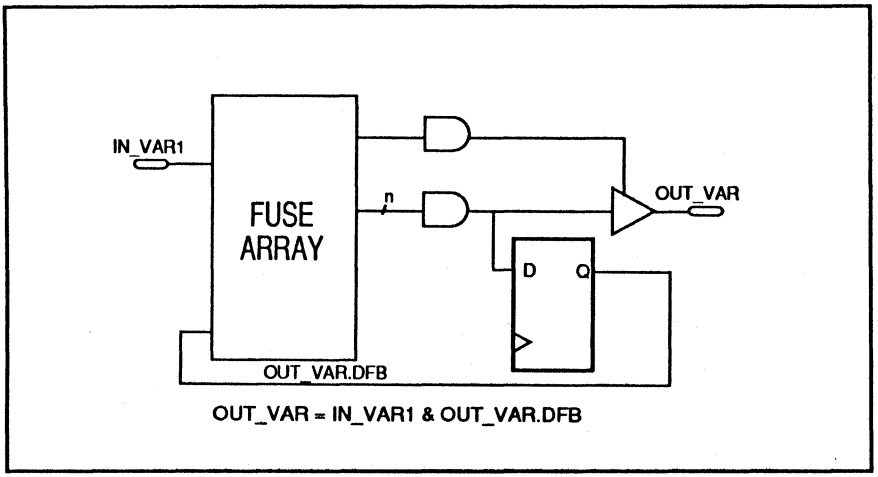


Figure 1-8. Programmable Registered Feedback

Figure 1-9 shows a registered output with internal feedback.

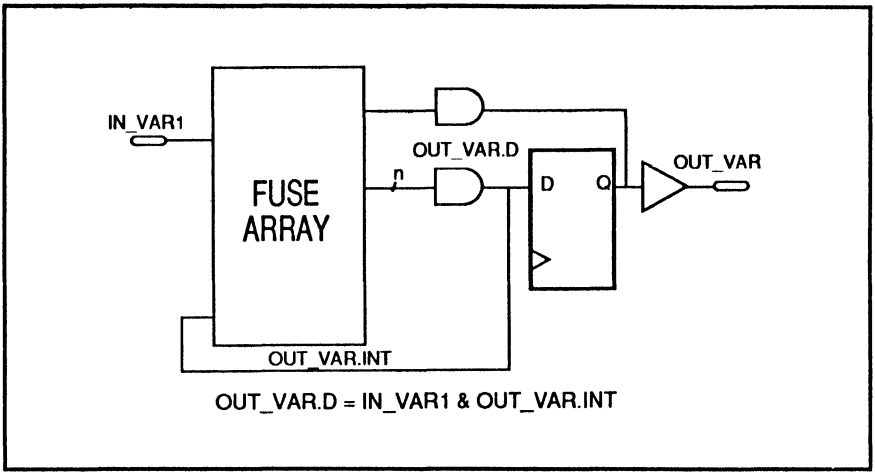


Figure 1-9. Programmable Internal Feedback

❑ Multiplexer Extension Usage

Certain devices allow selection between programmable and common control functions. For example, for each output, the P29MA16 contains multiplexers for selecting between common and product term clocks and output enables. Figure 1-10 shows the P29MA16 programmable clock and output enable capability.

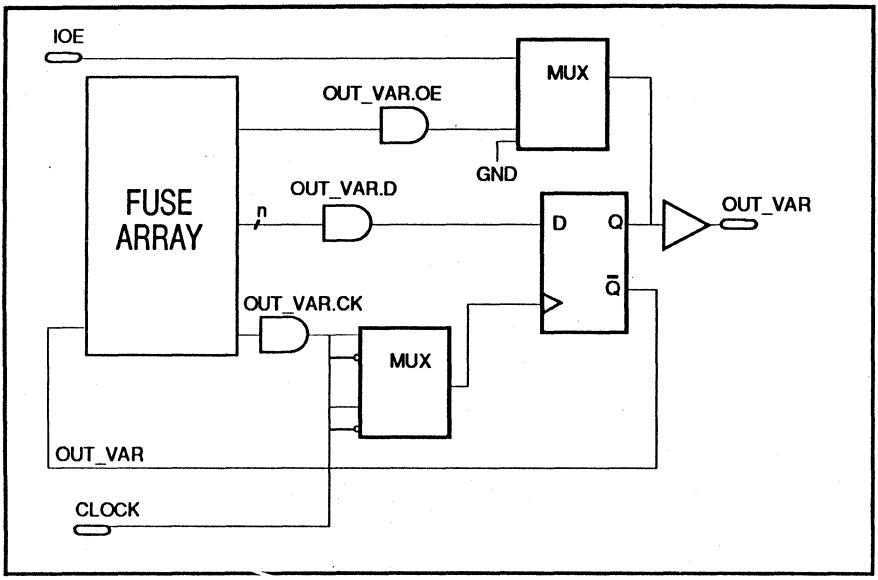


Figure 1-10. Output with Output Enable and Clock Multiplexers

If expressions are written for the **.OE** and **.CK** extensions, the multiplexer outputs are selected as product term output enable and clock, respectively. Otherwise, if expressions are written for the **.OEMUX** and **.CKMUX** extensions, the multiplexer outputs are selected as common output enable and clock, respectively.

Expressions written for the **.OEMUX** and **.CKMUX** extensions can have only one variable and be operated on only by the negation operator, **!**. This is because their inputs are not from the fuse array, but from a common source, such as a clock pin. This is in contrast with expressions written for the **.OE** and **.CK** extensions, which take their inputs from the fuse array.

Figure 1-11 shows a registered output with the output enable multiplexer output selected as Vcc, output enable always enabled, and the clock multiplexer output selected as the common clock pin inverted, negative-edge clock.

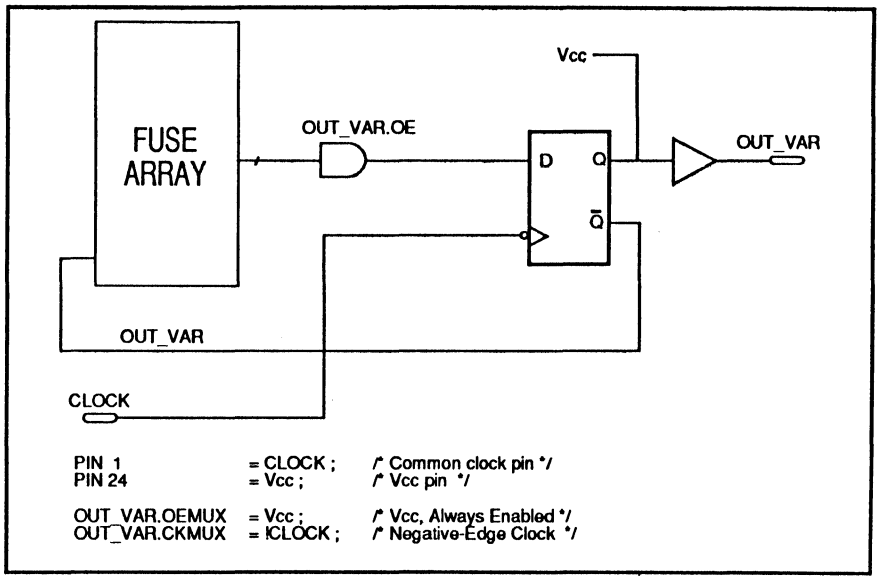


Figure 1-11. Output with Output Enable and Clock Multiplexers Selected

Expressions for the **.OE** and **.OEMUX** extensions are mutually exclusive; that is, only one may be written for each output. Likewise, expressions for the **.CK** and **.CKMUX** extensions are mutually exclusive.

❑ Boolean Logic Review

Table 1-12 lists the rules that the CUPL compiler uses for evaluating logic expressions. These basic rules are listed for reference purposes only.

Table 1-12. Logic Evaluation Rules

Commutative Property:
A & B = B & A
A # B = B # A
Associative Property:

A & (B & C) = (A & B) & C
A # (B # C) = (A # B) # C
Distributive Property:
A & (B # C) = (A & B) # (A & C)
A # (B & C) = (A # B) & (A # C)
Absorptive Property:
A & (A # B) = A
A # (A & B) = A
DeMorgan's Theorem:
!(A & B & C) = !A # !B # !C
!(A # B # C) = !A & !B & !C
XOR Identity:
A $ B = (!A & B) # (A & !B)
!(A $ B) = A $!B = !A $ B
= (!A & !B) # (A& B)
Theorems:
A & 0 = 0 A & 1 = A
A # 0 = A A # 1 = 1
A & A = A A & !A = 0
A # A = A A # !A = 1

❑ Expressions

Expressions are combinations of variables and operators that produce a single result when evaluated. An expression may be composed of any number of sub-expressions.

Expressions are evaluated according to the precedence of the particular operators involved. When operators with the same precedence appear in an expression, evaluation order is taken from left to right. Parentheses may be used to change the order of evaluation; the expression within the innermost set of parentheses is evaluated first.

In Table 1-13, note how the order of evaluation and use of parentheses affect the value of the expression.

Table 1-13. Sample Expressions

Expression	Result	Comments
A # B & C	A # B & C	
(A # B) & C	A & C # B & C	Parentheses change order
!A & B	!A & B	
!(A & B)	!A # !B	DeMorgan's Theorem
A # B & C # D	A # D # B & C	
A # B & (C # D)	A # B & C # B & D	Parentheses change order

❑ Logic Equations

Logic equations are the building blocks of the CUPL language. The form for logic equations is as follows:

[!] var [.ext] = exp ;

where

var is a single variable or a list of indexed or non-indexed variables defined according to the rules for the list notation (see the subtopic, List Notation in this chapter). When a variable list is used, the expression is assigned to each variable in the list.

.ext is an optional extension to assign a function to the major nodes inside a programmable device (see Table 1-11).

exp is an expression; that is, a combination of variables and operators (see "Expressions" in this chapter).

= is the assignment operator; it assigns the value of an expression to a variable or set of variables.

! is the complement operator.

The complement operator can be used to express the logic equation in negative true logic. The operator directly precedes the variable name (no spaces) and denotes that the expression on the right side is to be complemented before it is assigned to the variable name. Use of the complement operator on the left side is provided solely as a convenience. The equation may just as easily be written by complementing the entire expression on the right side.

Older logic design software that did not provide the automatic DeMorgan capability (output polarity assigned according to the pin variable declaration) required the use of the complement operator when using devices with inverting buffers.

Place logic equations in the "Logic Equation" section of the source file provided by the template file.

Logic equations are not limited solely to pin (or node) variables, but may be written for any arbitrary variable name. A variable defined in this manner is an intermediate variable. An intermediate variable name can be used in other expressions to generate logic equations or additional intermediate variables. Writing logic equations in this "top down" manner yields a logic description file that is generally easier to read and comprehend.

Place intermediate variables in the "Declarations and Intermediate Variable Definitions" section of the source file.

The following are some examples of logic equations:

```
SEL_0=A15 & !A14;          /* A simple, decoded output pin */
Q0.D=Q1 & Q2 & Q3;         /* Output pin w/ D flip-flop    */
```

```
Q1.J = Q2 # Q3;          /* Output pin w/ JK flip-flop    */
Q1.K = Q2 & !Q3;
MREQ=READ # WRITE;       /* Intermediate Variable         */
SEL_1=MREQ & A15;        /* Output intermediate var       */
[D0..3] = 'h'FF;         /* Data bits assigned to constant
                         */
[D0..3].oe = read;       /* Data bits assigned to variable
                         */
```

APPEND Statements

In standard logic equations, normally only one expression is assigned to a variable. The **APPEND** statement enables multiple expressions to be assigned to a single variable. The format is as follows.

APPEND [!]var[.ext] = expr ;

where

! is the complement operator to optionally define the polarity of the variable.

var is a single variable or a list of indexed or non-indexed variables in standard list format.

.ext is an optional extension that defines the function of the variable.

= is the assignment operator.

expr is a valid expression.

; is a semicolon to mark the end of the statement.

The expression that results from multiple **APPEND** statements is the logical OR of all the **APPEND** statements. If an expression has not already been assigned to the variable, the first **APPEND** statement is treated as the first assignment.

The following example shows several APPEND statements.

```
APPEND Y = A0 & A1 ;
```

```
APPEND Y = B0 & B1 ;
APPEND Y = C0 & C1 ;
```

The three statements above are equivalent to the following equation.

```
Y = (A0 & A1) # (B0 & B1) # (C0 & C1) ;
```

The **APPEND** statement is useful in adding additional terms (such as reset) to state-machine variables or constructing user-defined functions (see the subtopics, State Machine Syntax and User-Defined Functions in this chapter).

❑ Set Operations

All operations that are performed on a single bit of information, for example, an input pin, a register, or an output pin, may be applied to multiple bits of information grouped into sets. Set operations can be performed between a set and a variable or expression, or between two sets.

The result of an operation between a set and a single variable (or expression) is a new set in which the operation is performed between each element of the set and the variable (or expression). For example

```
[D0, D1, D2, D3] & read
```

evaluates to:

```
[D0 & read, D1 & read, D2 & read, D3 & read]
```

When an operation is performed on two sets, the sets must be the same size (that is, contain the same number of elements). The result of an operation between two sets is a new set in which the operation is performed between elements of each set.

For example

```
[A0, A1, A2, A3] & [B0, B1, B2, B3]
```

evaluates to:

[A0 & B0, A1 & B1, A2 & B2, A3 & B3]

Bit field statements (see the subtopic, Bit Field Declaration Statements in this chapter) may be used to group variables into a set that can be referenced by a single variable name. For example, group the two sets of variables in the above operation as follows:

FIELD a_inputs = [A0, A1, A2 A3] ;

FIELD b_inputs = [B0, B1, B2, B3] ;

Then perform a set operation between the two sets, for example, an AND operation, as follows:

a_inputs & b_inputs

When numbers are used in set operations, they are treated as sets of binary digits. A single octal number represents a set of three binary digits, and a single decimal or hexadecimal number represents a set of four binary digits. Table 1-14 lists the representation of numbers as sets.

Table 1-14. Equivalent Binary Sets

Number	Equivalent Binary Set	Number	Equivalent Binary Set
'O'X	[X, X, X]	'H'X	[X,X,X,X]
'O'0	[0, 0, 0]	'H'0	[0,0,0,0]
'O'1	[0, 0, 1]	'H'1	[0,0,0,1]
'O'2	[0, 1, 0]	'H'2	[0,0,1,0]
'O'3	[0, 1, 1]	'H'3	[0,0,1,1]
'O'4	[1, 0, 0]	'H'4	[0,1,0,0]
'O'5	[1, 0, 1]	'H'5	[0,1,0,1]
'O'6	[1, 1, 0]	'H'6	[0,1,1,0]
'O'7	[1, 1, 1]	'H'7	[0,1,1,1]
'D'0	[0,0,0,0]	'H'8	[1,0,0,0]
'D'1	[0,0,0,1]	'H'9	[1,0,0,1]
'D'2	[0,0,1,0]	'H'A	[1,0,1,0]
'D'3	[0,0,1,1]	'H'B	[1,0,1,1]
'D'4	[0,1,0,0]	'H'C	[1,1,0,0]

'D'5	[0,1,0,1]	'H'D	[1,1,0,1]
'D'6	[0,1,1,0]	'H'E	[1,1,1,0]
'D'7	[0,1,1,1]	'H'F	[1,1,1,1]
'D'8	[1,0,0,0]		
'D'9	[1,0,0,1]		

Numbers may be effectively used as "bit masks" in logic equations using sets. An example of this application is the following 4-bit counter.

```
field count = [Q3, Q2, Q1, Q0];
count.d = 'b' 0001 & (!Q0)
    # 'b' 0010 & (Q1 $ Q0)
    # 'b' 0100 & (Q2 $ Q1 & Q0)
    # 'b' 1000 & (Q3 $ Q2 & Q1 & Q0);
```

The equivalent logic equations written without set notation are as follows:

```
Q0.d = !Q0;
Q1.d = Q1 $ Q0;
Q2.d = Q2 $ Q1 & Q0;
Q3.d = Q3 $ Q2 & Q1 & Q0;
```

Equality Operations

Unlike other set operations, the equality operation evaluates to a single Boolean expression. It checks for bit equality between a set of variables and a constant. The format for the equality operation is as follows:

1. **[var, var, ... var]: constant ;**

2. **bit_field_var:constant ;**

where

[var, var, ... var] is a list of variables in shorthand notation.

constant is a number (hexadecimal by default).

bit_field_var is a variable defined using a bit field statement.

: is the equality operator.

; is a semicolon used to mark the statement end.

Note

Square brackets do not indicate optional items, but delimit variables in a list.

Format 1 is used between a list of variables and a constant value. Format 2 is used between a bit field variable and a constant value.

The bit positions of the constant number are checked against the corresponding positions in the set. Where the bit position is a binary 1, the set element is unchanged. Where the bit position is a binary 0, the set element is negated. Where the bit position is a binary X, the set element is removed. The resulting elements are then ANDed together to create a single expression. In the following example, hexadecimal D (binary 1101) is checked against A3, A2, A1, and A0.

```
select = [A3..0]:D ;
```

The set elements A3, A2, and A0 remain unchanged because the corresponding bit position is positive. Set element A1 is negated because its corresponding bit position is negative. Therefore, the above expression is equivalent to the following expression:

```
select = A3 & A2 & !A1 & A0 ;
```

In the following example, binary 1X0X is checked against A3, A2, A1, A0.

```
select = [A3..0]:'B'1X0X ;
```

The set element A3 remains unchanged because the corresponding bit position is positive. Set element A1 is negated because its corresponding bit position is negative. Set elements A2 and A0 are removed from the expression because the corresponding bit positions are "don't-cared." Therefore, the above expression is equivalent to the following expression:

```
select = A3 & !A1 ;
```

In addition to address decoding, the equality operator can be used to specify a counter or state machine. For example, a 4-bit counter can be specified using the following notation:

```
FIELD count = [Q0..3];
Q0.J      =     count:0 # count:2 # count:4 # count:6
   # count:8 # count:A # count:C # count:E ;
Q0.K      =     count:1 # count:3 # count:5 # count:7
          # count:9 # count:B # count:D # count:F ;
Q1.J      =     count:1 # count:5 # count:9 # count:D
;
Q1.K      =     count:3 # count:7 # count:B # count:F
;
Q2.J      =     count:3 # count:B ;
Q2.K      =     count:7 # count:F ;
Q3.J      =     count:7 ;
Q3.K      =     count:F ;
```

The equality operator can also be used with a set of variables that are to be operated upon identically. The following syntax can be used as a time-saving convenience:

[var, var, ... , var]:op

which is equivalent to:

var op var op ... var

where

op is the &, # or $ operator (or its equivalent if an alternate set of operators has been defined).

var is any variable name.

For example, the following three expressions

[A3,A2,A1,A0]:&

[B3,B2,B1,B0]:#

[C3,C2,C1,C0]:$

are equivalent respectively to:

A3 & A2 & A1 & A0

B3 # B2 # B1 # B0

C3 $ C2 $ C1 $ C0

The equality operation can be used with an equivalent binary set to create a function table description of the output values. For example, in the following Binary-to-BCD code converter, output values are assigned by using the equality operation to define the inputs, and equivalent binary sets to group the output.

```
$DEFINE L 'b'0
$DEFINE H 'b'1
FIELD input = [in3..0];
FIELD output = [E, D, C, B, A];
/* in3..0 ->EDCBA
*/
output =    input:0  & [L, L, L, L, L]
          # input:1  & [L, L, L, L, H]
```

```
# input:2  & [L  L,  L,  H,  L]
# input:3  & [L,  L,  L,  H,  H]
# input:4  & [L,  L,  H,  L,  L]
# input:5  & [L,  L,  H,  L,  H]
# input:6  & [L,  L,  H,  H,  L]
# input:7  & [L,  L,  H,  H,  H]
# input:8  & [L,  H,  L,  L,  L]
# input:9  & [L,  H,  L,  L,  H]
# input:A  & [H,  L,  L,  L,  L]
# input:B  & [H,  L,  L,  L,  H]
# input:C  & [H,  L,  L,  H,  L]
# input:D  & [H,  L,  L,  H,  H]
# input:E  & [H,  L,  H,  L,  L]
# input:F  & [H,  L,  H,  L,  H];
```

Range Operations

The range operation is similar to the equality operation except that the constant field is a range of values instead of a single value. The check for bit equality is made for each constant value in the range. The format for the range operation is as follows:

1. **[var, var, ... var]:[constant_lo..constant_hi] ;**

2. **bit_field_var:[constant_lo..constant_hi] ;**

where:

[var, var, ... var] is a list of variables in shorthand notation.

bit_field_var is a variable that has been defined using a bit field statement.

: is the equality operator.

; is a semicolon used to end the statement.

[constant_lo constant_hi] are numbers (hexadecimal by default) that define the range operation.

Note

Square brackets do not indicate optional items, but delimit items in a list

---

Format 1 specifies the range operation between a list of variables and a range of constant values. Format 2 specifies a range operation between a bit field variable and a range of constant values.

All numbers greater than or equal to **constant_lo** and less than or equal to **constant_hi** are used to create ANDed expressions as in the equality operation. The sub-expressions are then ORed together to create the final evaluated expression. For example, the **RANGE** notation can be used to look for a decoded hex value between 1100 and 1111 on an address bus containing A3, A2, A1, and A0. First, define the address bus, as follows:

```
FIELD address = [A3..A0]
```

Then write the **RANGE** equation:

```
select = address:[C..F] ;
```

This is equivalent to the following equation:

```
select = address:C # address:D
  # address:E # address:F ;
```

This equation expands to:

```
select    =  A3     &  A2  &  !A1 &  !A0
          #  A3     &  A2  &  !A1 &   A0
          #  A3     &  A2  &   A1 &  !A0
          #  A3     &  A2  &   A1 &   A0;
```

The logic minimization capabilities within CUPL reduce the previous equation into a single product term equivalent. The minimization works as follows. First, lines one and two are combined and lines three and four are combined to produce the following equation:

```
select =    A3 & A2 & !A1 & (!A0 # A0)
          # A3 & A2 &  A1 & (!A0 # A0) ;
```

Since the expression (!A0 # A0) is always true, it can be removed from the equation, and the equation reduces to:

```
select = A3 & A2 & !A1
       # A3 & A2 &  A1 ;
```

By the same process, the equation reduces to the following:

```
select = A3 & A2 & (!A1 # A1) ;
```

Since the expression (!A1 # A1) is always true, removing it reduces the equation to the single product term:

```
select = A3 & A2 ;
```

When either the equality or range operations are used with indexed variables, the **CONSTANT** field must contain the same number of significant bit locations as the highest index

number in the variable list. Index positions not in the pin list or field declaration are DON'T CAREd in the operation.

In the following example, pin assignments are made, an address bus is declared, and a decoded output is asserted over the hexadecimal memory address range 8000 through BFFF.

```
PIN   [1..4] = [A15..12] ;
FIELD address = [A15..12] ;
chip_select = address:[8000..BFFF] ;
```

Although the variables A15, A14, A13, and A12 are the only address inputs to the device, a full 16-bit address is used in the range expression. The most significant bit, A15, determines that the field is a 16-bit field. The lower order address bits (A0 through A11) are effectively DON'T CAREd in the equation, because the variable index numbers are used to determine bit position. Even though the lower order bits are not present in the device, the constant value is written as though they did exist, generating a more meaningful expression in terms of documentation.

Consider, for example, the following application that decodes a microprocessor address for an I/O port:

```
PIN [3..6] = [A7..10] ;
FIELD ioaddr = [A7..10]; /*       order of field
                declaration is not
                important when using
                indexed variables */
io_port = ioaddr:[400..6FF] ;
```

Since the most significant bit is A10, an 11-bit constant field is required (although three hex digits form a 12-bit address, the bit position for A11 is ignored).

Address bits A0 through A6 are DON'T CAREd in the expression. Without the bit position justification, the range equation would be written as

```
io_port = ioaddr:[8..D];
```

This expression doesn't clearly document the actual I/O address range that is desired.

The original equation without the range operation could be written as follows:

```
io_port =   A10 & !A9 & !A8 & !A7
        #   A10 & !A9 & !A8 &  A7
        #   A10 & !A9 &  A8 & !A7
        #   A10 & !A9 &  A8 &  A7
        #   A10 &  A9 & !A8 & !A7
        #   A10 &  A9 & !A8 &  A7;
```

CUPL reduces this equation to the following:

```
io_port = A10 & !A9  # A10 &  A9 & !A8;
```

Note

Careless use of the range feature may result in the generation of huge numbers of product terms, particularly when fields are composed of variables with large index numbers. The algorithm for the range does a bit-by-bit comparison of the two constant values given in the range operation, starting with index variable 0 (whether it exists in the field or not). If the value of the bit position for **constant_lo** is less than that for **constant_hi**, the variable for that bit position is not used in the generation of the ANDed expressions. When the value of the bit position for

constant_lo is equal to or greater than that for **constant_hi**, an ANDed expression is created for all constant values between this new value and the original **constant_hi** value.

For example, consider the following logic equation that uses the range function on a 16-bit address field.

```
field address = [A15..12];
board_select = address:[A000..DFFF];
```

Figure 1-12 shows how the CUPL algorithm treats this equation.

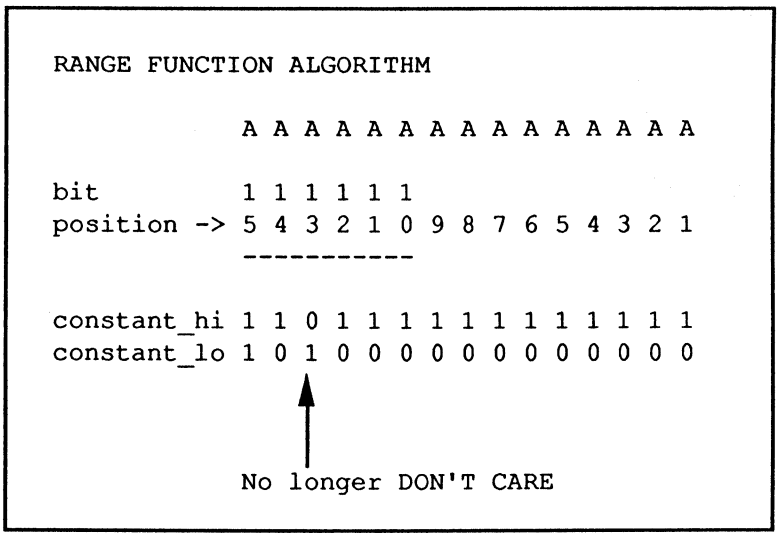


Figure 1-12. Range Function Algorithm

The algorithm ignores all bit positions lower than position 13, because for these positions **constant_lo** is less than **constant_hi**. Figure 1-13 shows the result.

```
RANGE FUNCTION ALGORITHM

              A A A A A A A A A A A A A A A A

bit           1 1 1 1 1 1
position ->   5 4 3 2 1 0 9 8 7 6 5 4 3 2 1 0
              -----------

constant_hi   1 1 0 x x x x x x x x x x x x x x
constant_lo   1 0 1 x x x x x x x x x x x x x x
```

Figure 1-13. Range Function Results

The following two product terms are generated as a result of the range function in Figure 1-13.

A15 & A14 & !A13

A15 & !A14 & A13

The following equation is another example using the range function.

board_select = address:[A000..D000] ;

Because the values of **constant_lo** and **constant_hi** match for the least significant bits, the algorithm generates product terms as follows:

1010 0000 0000 0000

1010 0000 0000 0001

1010 0000 0000 0010

1010 0000 0000 0011

•

1100 1111 1111 1111

1101 0000 0000 0000

The number of product terms generated is over twelve thousand (4096 x 3 + 1). This number of product terms would probably produce an "out of memory" error message because CUPL cannot hold this many product terms in memory at one time.

❑ Truth Tables

Sometimes the clearest way to express logic descriptions is in tables of information. CUPL provides the **TABLE** keyword to create tables of information. The format for using **TABLE** is as follows:

```
TABLE var_list_1 => var_list_2 {
    input_n => output_n ;
        .
        .
    input_n => output_n ;
    }
```

where

var_list_1 defines the input variables.

var_list_2 defines the output variables.

input_n is a decoded value (hex by default) or a list of decoded values of **var_list_1**.

output_n is a decoded value (hex by default) of **var_list_2**.

{ } are braces to begin and end the assignment block.

=> specifies a one-to-one assignment between variable lists, and between input and output numbers.

First, define relevant input and output variable lists, and then specify one-to-one assignments between decoded values of the

input and output variable lists. Don't-care values are supported for the input decode value, but not for the output decode value.

A list of input values can be specified to make multiple assignments in a single statement. The following block describes a simple binary-to-BCD code converter:

```
FIELD input = [in3..0];
FIELD output = [E, D, C, B, A];
TABLE input => output {
0=>00;    1=>01;    2=>02;    3=>03;
4=>04;    5=>05;    6=>06;    7=>07;
8=>08;    9=>09;    A=>10;    B=>11;
C=>12;    D=>13;    E=>14;    F=>15;
}
```

The following example illustrates the use of a list of input numbers to do address decoding for various-sized RAM, ROM, and I/O devices. The address range is decoded according to the rules (in terms of indexed variable usage) for the range operation (see the subtopic, Range Operations in this chapter).

```
PIN [1..4] = [a12..15];      /* Upper 4 address      */
PIN 12 = !RAM_sel;           /* 8K x 8 RAM           */
PIN 13 = !ROM_sel;           /* 32K x 8 ROM          */
PIN 14 = !timer_sel;         /* 8253 Timer           */
FIELD address = [a15..12];
FIELD decodes = [RAM_sel,ROM_sel,timer_sel];
TABLE address => decodes {
[1000..2FFF] => 'b'100;   /* select RAM */
[5000..CFFF] => 'b'010;   /* select ROM */
F000 => 'b'001;     /* select timer         */
}
```

❑ State-Machine Syntax

This section describes the CUPL state machine syntax, providing a brief overview of its use, a definition of a state machine, and explaining in detail the CUPL state machine syntax.

The state-machine approach used with the CUPL compiler-based PLD language permits bypassing the gate and equation level stage in logic design and to move directly from a system-level description to a PLD implementation. Additionally, unlike assembler-based approaches, the state-machine approach allows clear documentation of design, for future users.

❑ State-Machine Model

A synchronous state machine is a logic circuit with flip-flops. Because its output can be fed back to its own or some other flip-flop's input, a flip-flop's input value may depend on both its own output and that of other flip-flops; consequently, its final output value depends on its own previous values, as well as those of other flip-flops.

The CUPL state-machine model, as shown in Figure 1-14, uses six components: inputs, combinatorial logic, storage registers, state bits, registered outputs, and non-registered outputs.

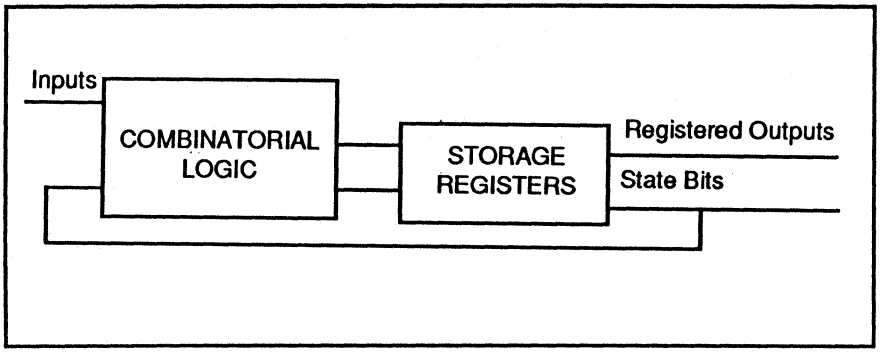


Figure 1-14. State Machine Model

Figure 1-15 shows the timing relationships between the state machine components.

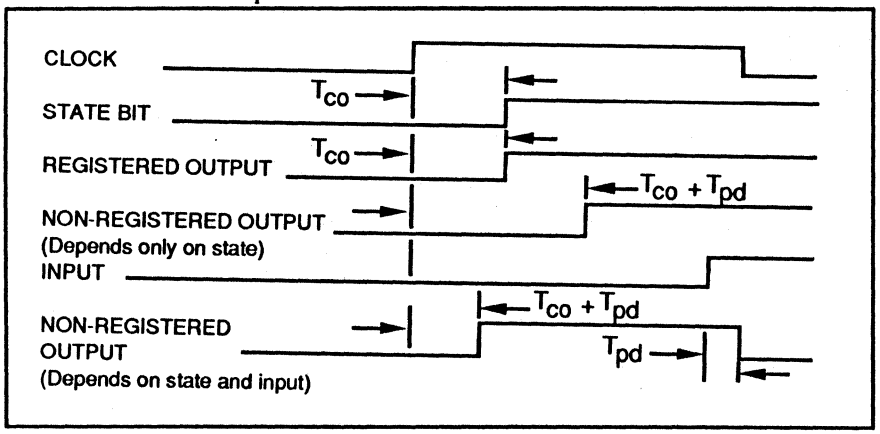


Figure 1-15. State Machine Timing Diagram

The following definitions refer to the timing diagram in Figure 1-15.

Inputs - are signals entering the device that originate in some other device.

Combinatorial Logic - is any combination of logic gates (usually AND-OR) that produces an output signal that is valid Tpd (propagation delay time) nsec after any of the signals that drive these gates changes. Tpd is the delay between the initiation of an input or feedback event and the occurrence of a non-registered output.

State Bits - are storage register outputs that are fed back to drive the combinatorial logic. They contain the present-state information.

Storage Registers - are any flip-flop elements that receive their inputs from the state machine's combinatorial logic. Some registers are used for state bits: others are used for registered outputs. The registered output is valid Tco (clock to out time) nsec after the clock pulse occurs. Tco is the time delay between the initiation of a clock signal and the occurrence of a valid flip-flop output.

For the system to operate properly, the PLD's requirements for setup and hold times must be met. For most PLDs, the setup time (Tsu) usually includes both the propagation delay of the combinatorial logic and the actual setup time of the flip-flops. Tsu is the time it takes for the result of either feedback or an input event to appear at the input to a flip-flop. A subsequent clock input cannot be applied until this result becomes valid at the flip-flop's input. The flip-flops can be either D, D-CE, J- K, S-R, or T types.

Non-registered Outputs - are outputs that come directly from the combinatorial logic gates. They may be functions of the state bits and the input signals (and have asynchronous timing), or they may be purely dependent on the current state-bit values, in which case they become valid Tco + Tpd nsec after an active clock edge occurs.

Registered Outputs - are outputs that come from the storage registers but are not included in the actual state-bit field (that is, a bit field composed of all the state bits). State- machine

theory requires that the setting or resetting of these registered outputs depends on the transition from a present state to a next state. This allows a registered output to be either set or reset in a given state depending upon how the machine came to be in that state. Thus, a registered output can assume a hold operation mode. In the hold mode, the registered output will remain at its last value as long as the current state transition does not specify an operation on that registered output.

Note

This hold mode of operation is available only for devices which use D-CE, J-K, or S-R type flip-flops.

❑ Syntax

To implement the state machine model, CUPL supplies a syntax that allows the describing of any function in the state machine.

The **SEQUENCE** keyword identifies the outputs of a state machine and is followed by statements that define the function of the state machine. The format for the **SEQUENCE** syntax is as follows:

```
SEQUENCE state_var_list {
PRESENT state_n0 statements;
    .
    .
    .
PRESENT state_nn statements;
}
```

where

state_var_list is a list of the state bit variables used in the state machine block. The variable list can be represented by a field variable.

state_n is a decoded value (hex by default) of the **state_variable_list** and must be unique for each **PRESENT** statement.

statements are any of the conditional, next, or output statements described in the following subsections of this section.

; is a semicolon used to mark the end of a statement.

{ } are braces to mark the beginning and end of the state machine description.

Symbolic names defined with the **$DEFINE** command may be used to represent **state_numbers**.

The **SEQUENCE** keyword causes the storage registers and registered output types generated to be the default type for the target device. For example, by using the **SEQUENCE** keyword in a design with a P16R8 target device, the state storage registers and registered outputs will be generated as D-type flip-flops.

The storage registers for certain devices can be programmed as more than one type. In the case of the F159 (Signetics PLS159), they can be either D or J-K type flip-flops. By default, using the **SEQUENCE** statement with a design for the F159 will cause the state storage registers and registered outputs to be generated as J-K type flip-flops. To override this default, the **SEQUENCED** keyword would be used in place of the **SEQUENCE** keyword. This would cause the state registers and registered outputs to be generated as D-type flip-flops.

Along with the **SEQUENCE** and **SEQUENCED** keywords are the **SEQUENCEJK**, **SEQUENCERS**, and **SEQUENCET** keywords. Respectively, they cause the state registers and registered outputs to be generated as J-K, S-R, and T-type flip-flops.

The subsections that follow describe the types of statements that can be written in the state-machine syntax. Statements use the **IF**, **NEXT**, **OUT** and **DEFAULT** keywords.

Unconditional NEXT Statement

This statement describes the transition from the present state to a specified next state. The format is:

```
PRESENT state_n
NEXT state_n ;
```

where

state_n is a decoded value (default hex) of the state bit variables that are the output of the state machine.

A symbolic name can be assigned with the **$DEFINE** command to represent **state_n**.

Because the statement is unconditional (that is, it describes the transition to a specific next state), there can be only one **NEXT** statement for each **PRESENT** statement.

The following example specifies the transition from binary state 01 to binary state 10.

```
PRESENT 'b'01
NEXT 'b'10 ;
```

Figure 1-16 shows the transition described in the example above.

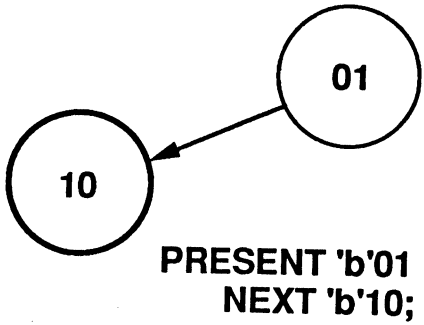


Figure 1-16. Unconditional NEXT Statement Diagram

For the transition described in the example and figure above, CUPL generates the following equations, depending on the type of flip-flop that is specified:

D-Type Flip-Flop

```
APPEND Q1.D = !Q1 & Q0;
APPEND Q0.D = 'b'0;     /* implicitly resets */
```

J-K-Type Flip-Flop

```
APPEND Q1.J = !Q1 & Q0;
APPEND Q1.K = 'b'0;
APPEND Q0.J = 'b'0;
APPEND Q0.K = !Q1 & Q0;
```

S-R-Type Flip-Flop

```
APPEND Q1.S = !Q1 & Q0;
APPEND Q1.R = 'b'0;
APPEND Q0.S = 'b'0;
APPEND Q0.R = !Q1 & Q0;
```

D-CE-Type Flip-Flop

```
APPEND Q1.D  = !Q1 & Q0;
APPEND Q1.CE = !Q1 & Q0;
APPEND Q0.D  = 'b'0;
APPEND Q0.CE = !Q1 & Q0;
```

T-Type Flip-Flop

```
APPEND Q1.T  = !Q1 & Q0;
APPEND Q0.T  = !Q1 & Q0;
```

See the subtopic, **APPEND** Statements in this chapter for a description of the **APPEND** command.

Conditional NEXT Statement

This statement describes the transition from the present state to a next state if the conditions in a specified input expression are met. The format is as follows.

```
PRESENT state_n
IF expr NEXT state_n;
   .
   .
   .
IF expr NEXT state_n;
[DEFAULT NEXT state_n;]
```

where

state_n is a decoded value (default hex) of the state bit variables that are the output of the state machine.

expr is any valid expression (see the subtopic, Expressions in this chapter).

; is a semicolon used to mark the end of a statement.

Note

The square brackets indicate optional items.

The value for each state number must be unique.

More than one conditional statement can be specified for each **PRESENT** statement.

The **DEFAULT** statement is optional. It describes the transition from the present state to a next state if none of the conditions in the specified conditional statements are met. In other words, it describes the condition that is the complement of the sum of all the conditional statements.

Note

Be careful when using the **DEFAULT** statement. Because it is the complement of all the conditional statements, the **DEFAULT** statement can generate an expression complex enough to greatly slow CUPL operation. In most applications, one or two conditional statements can be specified instead of the **DEFAULT** statement.

The following is an example of two conditional **NEXT** statements without a **DEFAULT** statement.

```
PRESENT 'b'01
    IF INA NEXT 'b'10;
    IF !INA NEXT 'b'11;
```

Figure 1-17 shows the transitions described by the above example.

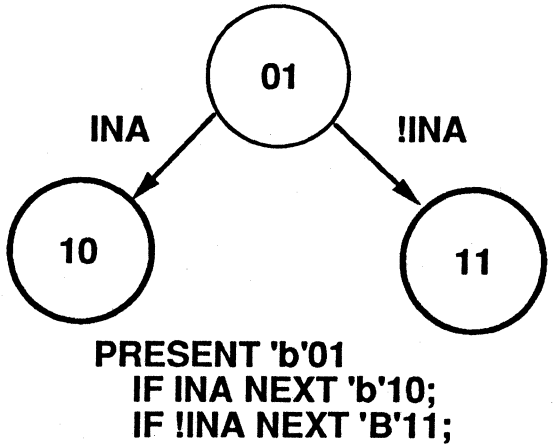


Figure 1-17. Conditional NEXT Statement Diagram

For the transitions described in the above example and figure, CUPL generates the following equations, depending on the type of flip-flop that is specified:

D-Type Flip-Flop

```
APPEND Q1.D = !Q1 & Q0;
APPEND Q0.D = !Q1 & Q0 & !INA;
```

D-CE-Type Flip-Flop

```
APPEND Q1.D  = !Q1 & Q0;
APPEND Q1.CE = !Q1 & Q0;
APPEND Q0.D  = !Q1 & Q0 & !INA;
```

```
APPEND Q0.CE = !Q1 & Q0 & INA;
```

J-K-Type Flip-Flop

```
APPEND Q1.J = !Q1 & Q0;
APPEND Q1.K = 'b'0;
APPEND Q0.J = 'b'0;
APPEND Q0.K = !Q1 & Q0 & INA;
```

S-R-Type Flip-Flop

```
APPEND Q1.S = !Q1 & Q0;
APPEND Q1.R = 'b'0;
APPEND Q0.S = 'b'0;
APPEND Q0.R = !Q1 & Q0 & INA;
```

T-Type Flip-Flop

```
APPEND Q1.T = !Q1 & Q0;
APPEND Q0.T = !Q1 & Q0 & INA;
```

The following is an example of two conditional statements with a **DEFAULT** statement.

```
PRESENT 'b'01
    IF INA & INB NEXT 'b'10';
    IF INA & !INB NEXT 'b'11;
        DEFAULT NEXT 'b'00;
```

Figure 1-18 shows the transitions described by the above example. Note the equation generated by the **DEFAULT** statement.

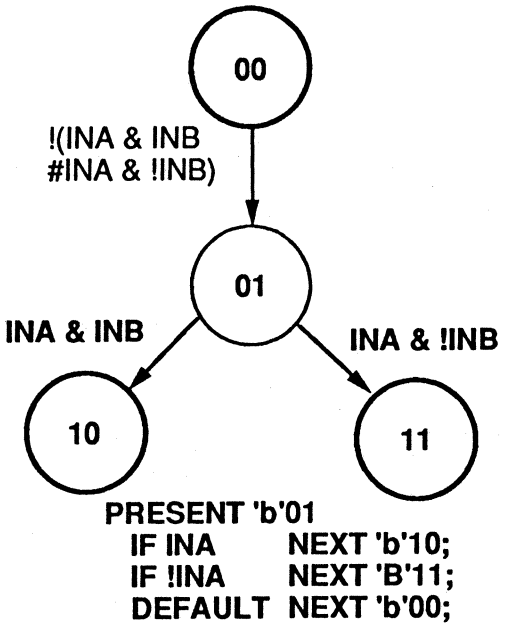


Figure 1-18. Conditional NEXT Statement with Default Diagram

For the transitions described in the above example and figure, CUPL generates the following equations, depending on the type of flip-flop that is specified.

D-Type Flip-Flop

```
APPEND Q1.D = !Q1 & Q0 & INA;
APPEND Q0.D = !Q1 & Q0 & INA & !INB;
```

D-CE-Type Flip-Flop

```
APPEND Q1.D  = !Q1 & Q0 & INA;
APPEND Q1.CE = !Q1 & Q0 & INA;
```

```
APPEND Q0.D  = 'b'0;
APPEND Q0.CE = !Q1 & Q0 & !INA
             # !Q1 & Q0 & INA & INB;
```

J-K-Type Flip-Flop

```
APPEND Q1.J = !Q1 & Q0 & INA;
APPEND Q1.K = 'b'0;
APPEND Q0.J = 'b'0;
APPEND Q0.K = !Q1 & Q0 & INA & INB
            # !Q1 & Q0 & !INA;
```

S-R-Type Flip-Flop

```
APPEND Q1.S = !Q1 & Q0 & INA;
APPEND Q1.R = 'b'0;
APPEND Q0.S = 'b'0;
APPEND Q0.R = !Q1 & Q0 & INA & INB
            # !Q1 & Q0 & !INA;
```

T-Type Flip-Flop

```
APPEND Q1.T  = !Q1 & Q0 & INA;
APPEND Q0.T  = !Q1 & Q0 & !INA
             # !Q1 & Q0 & INA & INB;
```

Unconditional Synchronous Output Statement

This statement describes a transition from the present state to a next state, specifies a variable for the registered (synchronous) outputs associated with the transition, and defines whether the variable is logically asserted. The format is as follows:

```
PRESENT state_n
   NEXT state_n  OUT [!]var... OUT [!]var;
```

where

state_n is a decoded value (default hex) of the state bit variables that are the output of the state machine.

var is a variable name declared in the pin declarations. It is not a variable from the **SEQUENCE state_var_list.**

! is the complement operator; use it to logically negate the variable, or omit it to logically assert the variable.

; is a semicolon used to mark the end of a statement.

Note

The square brackets indicate optional items.

The **PIN** declaration statement (see the subtopic, Pin Declaration Statements in this chapter) determines whether the variable, when asserted, is active-HI or active-LO. For example, if the variable has the negation symbol (!var) in the pin declaration, when it is asserted in the **OUT** statement, its value is active-LO.

Note

Use the negation mode only for D-CE, J-K, T or S-R type flip-flops; D-type flip-flops implicitly reset when assertion is not specified.

The following is an example of an unconditional synchronous output statement.

```
PRESENT 'b'01
    NEXT 'b'10 OUT Y OUT !Z ;
```

Figure 1-19 shows the transition and output variable definition described in the example above.

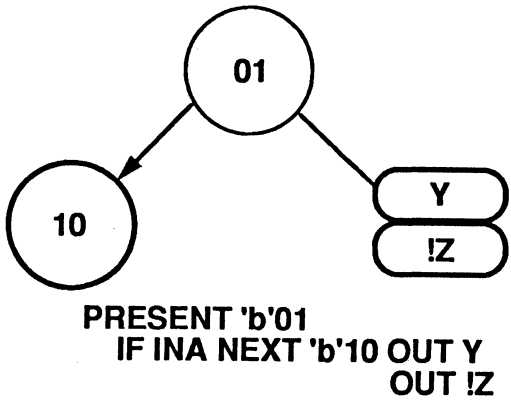


Figure 1-19. Unconditional Synchronous Output Diagram

For the synchronous output definitions in the example and figure above, CUPL generates the following equations, depending on the type of flip-flop that is specified.

D-Type Flip-Flop

```
APPEND Y.D = !Q1 & Q0;
```

(not defined for Z output)

D-CE Type Flip-Flop

```
APPEND Y.D    =    !Q1 & Q0;
APPEND Y.CE   =    !Q1 & Q0;
APPEND Z.D    =    'b'0;
APPEND Z.CE   =    !Q1 ^ Q0;
```

J-K-Type Flip-Flop

```
APPEND Y.J  =  !Q1 & Q0;
APPEND Y.K  =  'b'0;
APPEND Z.J  =  'b'0;
APPEND Z.K  =  !Q1 & Q0;
```

S-R-Type Flip-Flop

```
APPEND Y.S  =  !Q1 & Q0;
APPEND Y.R  =  'b'0;
APPEND Z.S  =  'b'0;
APPEND Z.R  =  !Q1 & Q0;
```

T-Type Flip-Flop

```
APPEND Y.T  =  !Q1 & Q0;
APPEND Z.T  =  !Q1 & Q0;
```

Conditional Synchronous Output Statement

This statement describes a transition from the present state to a next state, specifies a variable for the registered (synchronous) outputs associated with the transition, and defines whether the variable is logically asserted if the conditions specified in an input expression are met. The format is as follows:

```
PRESENT state_n
    IF expr NEXT state_n OUT [!]var...OUT [!] var;
                .
                .
    IF expr NEXT state_n OUT [!]var...OUT [!]var;
          [DEFAULT] NEXT state_n OUT [!]var;]
```

where

state_n is a decoded value (default hex) of the state bit variables that are the output of the state machine.

var is a variable name declared in the pin declarations. It is not a variable from the **SEQUENCE state_variable_list.**

! is the complement operator; use it to logically negate the variable, or omit it to logically assert the variable.

; is a semicolon used to mark the end of a statement.

expr is any valid expression.

Note

The square brackets indicate optional items.

The **PIN** declaration statement (see the subtopic, Pin Declaration Statements in this chapter) determines whether the variable, when asserted, is active-HI or active-LO. For example, if the variable has the negation symbol (!var) in the pin declaration, when it is asserted in the **OUT** statement, its value is active-LO.

Note

Use the negation mode only for J-K or S-R-type flip-flops; D-type flip-flops implicitly reset when assertion is not specified.

The **DEFAULT** statement is optional. It describes the transition from the present state to a next state, and defines the output variable, if none of the conditions in the specified conditional statements are met. In other words, it describes the condition that is the complement of the sum of all the conditional statements.

Note

Be careful when using the **DEFAULT** statement. Because it is the complement of all the conditional statements, the **DEFAULT** statement can generate an expression complex enough to greatly slow CUPL operation. In most applications, one or two conditional statements can be specified instead of the **DEFAULT** statement.

The following is an example of conditional synchronous output statements without a **DEFAULT** statement.

```
PRESENT 'b'01
    IF INA NEXT 'b'10 OUT Y;
    IF !INA NEXT 'b'11 OUT Z;
```

Figure 1-20 shows the transitions and outputs defined by the statements in the example above.

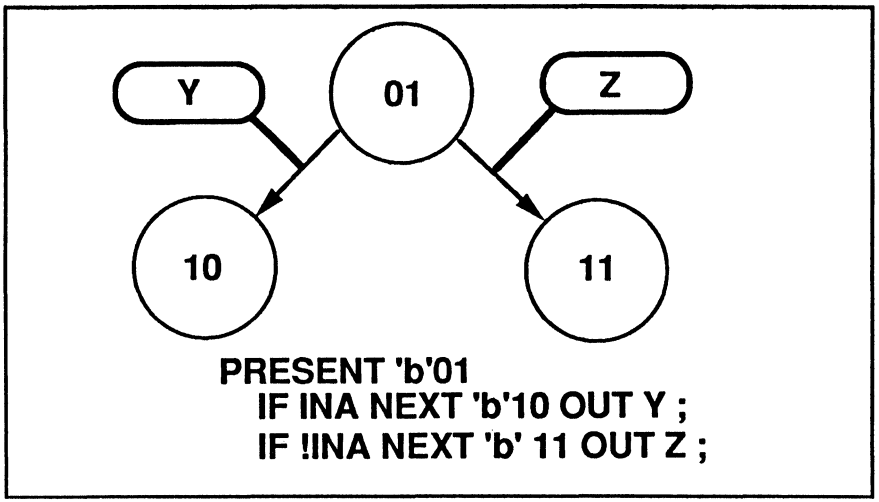


Figure 1-20. Synchronous Conditional Output Diagram

For the synchronous output definitions in the example and figure above, CUPL generates the following equations, depending on the type of flip-flop specified:

D-Type Flip-Flop

```
APPEND Y.D  =  !Q1 & Q0 & INA;
APPEND Z.D  =  !Q1 & Q0 & !INA;
```

D-CE-Type Flip-Flop

```
APPEND Y.D  =  !Q1 & Q0 & INA;
APPEND Y.CE =  !Q1 & Q0 & INA;
APPEND Z.D  =  !Q1 & Q0 & !INA;
APPEND Z.CE =  !Q1 & Q0 & !INA;
```

J-K-Type Flip-Flop

```
APPEND Y.J  =  !Q1 & Q0 & INA;
APPEND Y.K  =  'b'0;
APPEND Z.J  =  !Q1 & Q0 & !INA;
```

```
APPEND Z.K      =      'b'0;
```

S-R-Type Flip Flop

```
APPEND Y.S      =      !Q1 & Q0 & INA;
APPEND Y.R      =      'b'0;
APPEND Z.S      =      !Q1 & Q0 & !INA;
APPEND Z.R      =      'b'0;
```

T-Type Flip-Flop

```
APPEND Y.T      =      !Q1 & Q0 & INA;
APPEND Z.T      =      !Q1 & Q0 & !INA;
```

The following is an example of conditional output statements with a **DEFAULT** statement.

```
PRESENT 'b'01
    IF INA & INB NEXT 'b'10;
    IF INA & !INB NEXT 'b'11;
      DEFAULT NEXT 'b'00 OUT Y
          OUT !Z;
```

Figure 1-21 shows the transitions described by the above example. Note the equation generated by the **DEFAULT** statement.

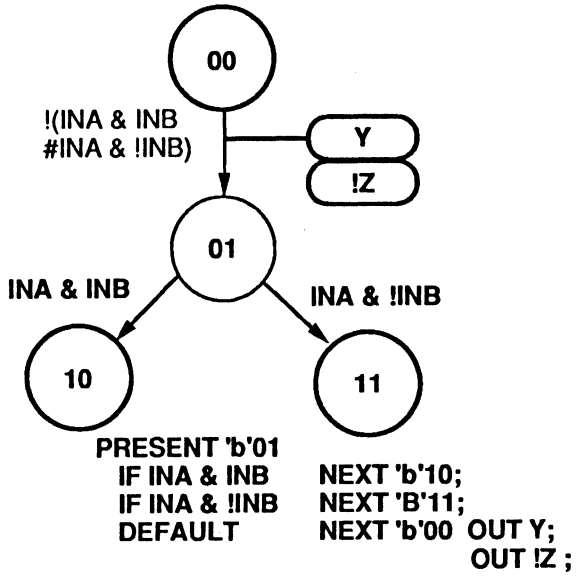


Figure 1-21. Synchronous Conditional Output with Default Diagram

For the transitions described in the above example and figure, CUPL generates the following equations, depending on the type of flip-flop that is specified.

D-Type Flip-Flop

APPEND Y.D = !Q1 & Q0 & !INA;

(not defined for Z output)

D-CE-Type Flip-Flop

APPEND Y.D = !Q1 & Q0 & !INA;

APPEND Y.CE = !Q1 & Q0 & !INA;

```
APPEND Z.D      =    'b'0;
APPEND Z.CE     =    !Q1 & Q0 & INA;
```

J-K-Type Flip-Flop

```
APPEND Y.J      =    !Q1 & Q0 & !INA;
APPEND Y.K      =    'b'0;
APPEND Z.J      =    'b'0;
APPEND Z.K      =    !Q1 & Q0 & !INA;
```

S-R-Type Flip-Flop

```
APPEND Y.S      =    !Q1 & Q0 & !INA;
APPEND Y.R      =    'b'0;
APPEND Z.S      =    'b'0;
APPEND Z.R      =    !Q1 & Q0 & !INA;
```

T-Type Flip-Flop

```
APPEND Y.T      =    !Q1 & Q0 & !INA
APPEND Z.T      =    !Q1 & Q0 & INA;
```

Unconditional Asynchronous Output Statement

This statement specifies variables for the non-registered (asynchronous) outputs associated with a given present state, and defines when the variable is logically asserted. The format is as follows:

```
PRESENT state_n
    OUT var ... OUT var;
```

where:

state_n is a decoded value (default hex) of the state bit variables that are the output of the state machine.

var is a variable name declared in the pin declarations. It is not a variable from the **SEQUENCE state_var_list.**

; is a semicolon used to mark the end of a statement.

The **PIN** declaration statement (see the subtopic, Pin Declaration Statements in this chapter) determines whether the variable, when asserted, is active-HI or active-LO. For example, if the variable has the negation symbol (!var) in the pin declaration, when it is asserted in the **OUT** statement, its value is active-LO.

Negating the variable (with the complement operator) is not a valid format for this statement.

Only one output statement can be written for each present state. However, multiple variables can be defined using more than one **OUT** keyword.

The following is an example of an unconditional asynchronous output statement.

```
PRESENT 'b'01
    OUT Y OUT Z;
```

Figure 1-22 shows the outputs defined by the statements in the example above.

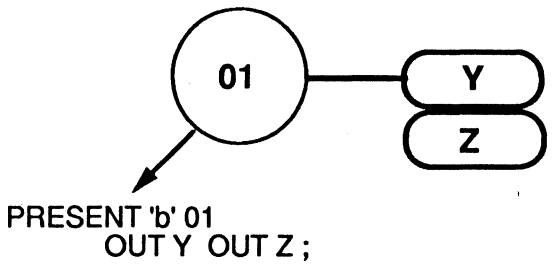


Figure 1-22. Unconditional Asynchronous Output Diagram

For the asynchronous output definitions in the example and figure above, CUPL generates the following equations:

```
APPEND Y = !Q1 & Q0;
```

APPEND Z = !Q1 & Q0;

Conditional Asynchronous Output Statement

This statement specifies variables for the non-registered (asynchronous) outputs associated with a given present state, and defines when the variables are logically asserted, if the conditions in an input expression are met. The format is as follows:

PRESENT state_n
IF expr OUT var ... OUT var;
.
.
IF expr OUT var ... OUT var;
[DEFAULT OUT var ... OUT var;]

where

state_n is a decoded value (default hex) of the state bit variables that are the output of the state machine.

var is a variable name declared in the pin declarations. It is not a variable from the **SEQUENCE** statement.

expr is any valid expression.

; is a semicolon used to mark the end of a statement.

Note

The square brackets indicate optional items.

The **PIN** declaration statement determines whether the variable, when asserted, is active-HI or active-LO. For

example, if the variable has the negation symbol (!var) in the pin declaration, when it is asserted in the **OUT** statement, its value is active-LO.

Negating the variable (with the complement operator) is not a valid format for this statement. Multiple output statements can be written for each present state, and define multiple variables using the **OUT** keyword.

The **DEFAULT** statement is optional. It defines the output variable if none of the conditions in the specified conditional statements are met. In other words, it describes the condition that is the complement of the sum of all the conditional statements.

Note

Be careful when using the **DEFAULT** statement. Because it is the complement of all the conditional statements, the **DEFAULT** statement can generate an expression complex enough to greatly slow CUPL operation. In most applications, one or two conditional statements can be specified instead of the **DEFAULT** statement.

The following is an example of conditional asynchronous output statements without a default statement.

```
PRESENT 'b'01
    IF INA OUT Y;
    IF !INA OUT Z;
```

Figure 1-23 shows the outputs defined by the statements in the above example.

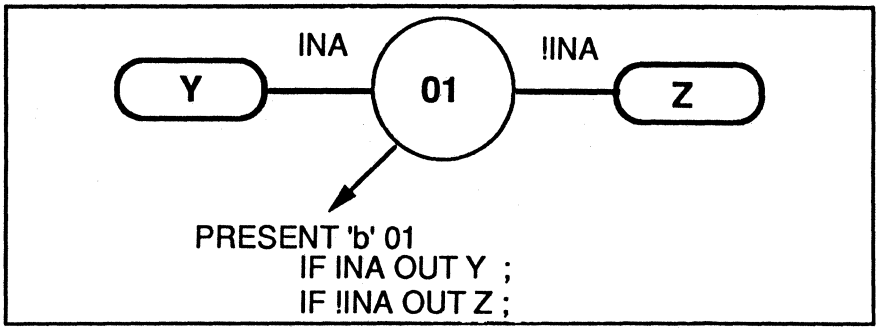


Figure 1-23. Conditional Asynchronous Output Diagram

For the asynchronous output definitions in the example and figure above, CUPL generates the following equations:

```
APPEND Y = !Q1 & Q0 & INA;
APPEND Z = !Q1 & Q0 & !INA;
```

The following is an example of conditional asynchronous output statements with a **DEFAULT** statement.

```
PRESENT 'b'01
    IF INA & INB OUT X;
    IF INA & !INB OUT Y;
    DEFAULT OUT Z;
```

Figure 1-24 shows the transitions described by the above example. Note the equation generated by the **DEFAULT** statement.

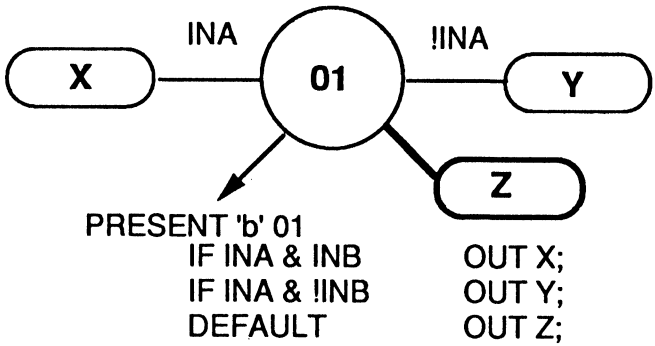


Figure 1-24. Conditional Asynchronous Output with Default Diagram

For the transitions described in the above example and figure, CUPL generates the following equations, depending on the type of flip-flop that is specified.

```
APPEND X = !Q1 & Q0 & INA & !INB;
APPEND Y = !Q1 & Q0 & INA & INB;
APPEND Z = !Q1 & Q0 & !INA;
```

Sample State-Machine Syntax File

This section provides an example of a simple two-bit counter implemented with state-machine syntax.

Figure 1-25 shows a diagram of the counter operation.

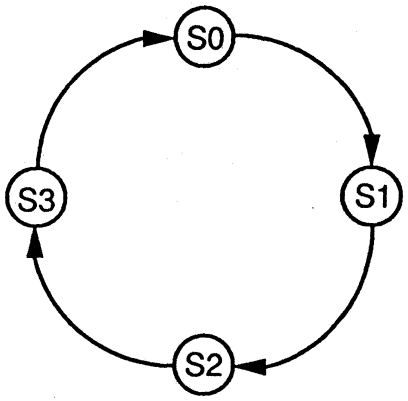


Figure 1-25. Simple 2-Bit Counter Diagram

The **$DEFINE** command assigns symbolic names to the states of the counter, and the **SEQUENCE** statement defines the transitions between states.

```
$DEFINE S0 0     /* assign symbolic names */
$DEFINE S1 1     /* to states       */
$DEFINE S2 2
$DEFINE S3 3
FIELD count = [Q1, Q0];
/* assign field variable to statebits */
SEQUENCE count {
    PRESENT S0  NEXT  S1;
    PRESENT S1  NEXT  S2;
    PRESENT S2  NEXT  S3;
    PRESENT S3  NEXT  S0;
}
```

See the example, Decade Up/Down Counter, in Chapter U5 for another illustration of a state machine implementation.

❑ Condition Syntax

The **CONDITION** syntax provides a higher-level approach to specifying logic functions than does writing standard Boolean logic equations for combinatorial logic. The format is as follows:

```
CONDITION {
    IF expr0 OUT var;
      .
      .
    IF exprn OUT var;
    DEFAULT OUT var;
    }
```

where

expr is any valid expression.

var is a variable name declared in the pin declaration. It can also be a list of indexed or non-indexed variables in list notation.

; is a semicolon used to mark the end of a statement.

The **CONDITION** syntax is equivalent to the asynchronous conditional output statements of the state machine syntax, except that there is no reference to any particular state. The variable is logically asserted whenever the expression or **DEFAULT** condition is met.

The variable cannot be logically negated in this format.

Note

Be careful when using the **DEFAULT** statement. Because it is the complement of all the conditional statements, the **DEFAULT** statement can generate an expression complex enough to greatly slow CUPL operation. In most applications, one or two conditional statements may be specified instead of the **DEFAULT** statement.

The following is an example of a 2 to 4 line decoder for the **CONDITION** syntax. The two data inputs, A and B, select one of four decoded outputs, Y0 through Y3, whenever the **ENABLE** signal is asserted. The **NO_MATCH** output is asserted if none of the other four outputs are true.

```
PIN [1,2] = [A,B] ;  /* Data Inputs */
PIN 3 = !enable ;    /* Enable Input */
PIN [12..15] = [Y0..3] ;     /* Decoded Outputs */
PIN 14 = no_match ;          /* Match Output */
CONDITION {
    IF enable & !B & !A   out Y0;
    IF enable & !B & A    out Y1;
    IF enable & B & !A    out Y2;
    IF enable & B & A     out Y3;
    }
```

The **DEFAULT** expression of the above example is equivalent to the following logic equation

```
no_match = !( enable & !B & !A)
# enable  &     !B     &     A
# enable  &     B      &     !A
# enable  &     B      &     A     ;
```

which reduces to the following:

```
no_match = !enable;
```

❑ User-Defined Functions

The **FUNCTION** keyword permits the creating of personal keywords by encapsulating some logic as a function and giving it a name. This name can then be used in a logic equation to represent the function. The format for user-defined functions is as follows:

FUNCTION name ([parameter$_0$,....,parameter$_n$])

{ body }

where

name is any group of valid symbols used to reference the function. Do not use any of the CUPL reserved keywords.

parameter is an optional variable used to reference variables when the function is used in a logic equation. It cannot be an expression.

body is any combination of logic equations, truth tables, state-machine syntax, condition syntax, or user function.

( ) are parentheses used to enclose the parameter list.

{ } are braces used to enclose the body of the function.

Note

The square brackets indicate optional items.

The statements in the body may assign an expression to the function, or may be unrelated equations.

When using optional parameters, the number of parameters in the function definition and in the reference must be identical. The parameters defined in the body of the function are substituted for the parameters referenced in the logic equation.

For example, the following defines an exclusive OR function:

```
FUNCTION xor(in1, in2) {
/* in1 and in2 are parameters */
xor = in1 & in2 # !in1 & in2;
}
```

An xor can be used in an equation with the inputs A and B passed as parameters, as follows:

```
Y = xor(A,B);
```

The result is the following logic equation assignment for the output variable Y:

```
Y = A & !B # !A & B;
```

When a function variable is referenced in an expression, the compiler takes the following action:

1. A special function invocation variable is assigned for the function name and its arguments. This variable name is not user accessible.

2. The rest of the expression is evaluated.

3. The function body, with the invocation parameters substituted, is evaluated.

4. The function invocation variable is assigned an expression according to the body of the function. If no assignment is made in the body statements, the function invocation variable is assigned the value of 'h'o.

Note

Functions must be defined before they may be referenced. Functions are not recursive; that is, a function body may not include a reference of the function being defined.

The following example shows a user-defined function to construct state-machine-type transitions for non-registered devices without internal feedback (such as PROMs).

```
FUNCTION  TRANSITION(present_state,
                     next_state,
                     input_conditions ) {
APPEND  state_out = state_in:present_state &
                     input_condition &
```

```
        next_state;
}
```

The function defined in the example above is used in the following example to implement a simple up/down counter as a series of **TRANSITION** function references:

```
PIN [10,11] = [Qin0..1];    /* Registered PROM */
    /*      output feed             */
    /*      back externally         */
    /*      on input pins           */
PIN [12,13] = [count0..1] ; /*      Count Control  */
PIN [1,2] = [Q0..1] ;       /* PROM Outputs */
FIELD state_in    =         [Qin0..1] ;
FIELD state_out   =         [Q0..1] ;
count_up = !count1 & !count0 ; /* count up          */
count_dn = !count1 & count0 ; /* count down         */
hold_cnt = count1;          /*      hold count     */
$DEFINE STATE0 'b'00
$DEFINE STATE1 'b'01
$DEFINE STATE2 'b'10
$DEFINE STATE3 'b'11
/* (transition function definition made here)     */
TRANSITION(STATE0, STATE1,      count_up) ;
TRANSITION(STATE1, STATE2,      count_up) ;
TRANSITION(STATE2, STATE3,      count_up) ;
TRANSITION(STATE3, STATE0,      count_up ;
TRANSITION(STATE0, STATE3,      count_dn) ;
TRANSITION(STATE1, STATE0,      count_dn) ;
```

```
TRANSITION(STATE2, STATE1,    count_dn);
TRANSITION(STATE3, STATE2,    count_dn);
TRANSITION(STATE0, STATE0,    hold_cnt);
TRANSITION(STATE1, STATE1,    hold_cnt);
TRANSITION(STATE2, STATE2,    hold_cnt);
TRANSITION(STATE3, STATE3,    hold_cnt);
```

❑ CUPL to XILINX XNF Interface

This section describes how a CUPL source file can be translated into an XNF netlist file. CUPL generates an equivalent PALASM file, using the **c** flag, which can be used as input to PDS2XNF. PDS2XNF will translate the PALASM file into an XNF file to be used by the remaining XILINX XACT™ integrated software package.

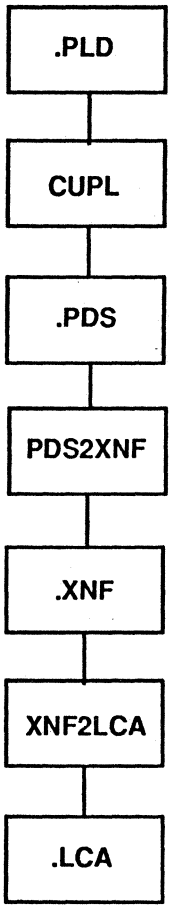


Figure 1-26. CUPL Design to LCA Translation Data Flow.

Getting an XNF File.

Just as CUPL is run to compile a PLD design, it can be run using the **-c** flag to produce an equivalent PALASM format file. For example, entering the following:

CUPL -c EXAMPLE.PLD

will create the PALASM file **EXAMPLE.PDS.**

Translating an Existing PLD File.

Any PLD which can be compiled by CUPL can be translated into PALASM format. Because a device must be specified for PLD design, a device must also be specified when translating a CUPL file to a PALASM format. The **VIRTUAL** device should be specified when a source file is for XILINX devices. In order to preserve the original use of the PLD source file, do not change the device name; CUPL will translate the file based on the devices that PALASM recognizes. A recognizable device is translated into the PDS file using the names given in Table 1-15. An unrecognizable or **VIRTUAL** device is translated into the PDS file as **USER.**

Table 1-15. PALASM Device Listing

PAL6L16	PAL10H8	PAL10L8
PAL10H20G8	PAL10H20P8	PAL12H6
PAL12L6	PAL12L10	PAL14H4
PAL14L4	PAL14L8	PAL16C1
PAL16H2	PAL16L2	PAL16L6
PAL16L8	PAL16P8	PAL16R4
PAL16R6	PAL16R8	PAL16RA8

PAL16RP4	PAL16RP6	PAL16RP8
PAL18L4	PAL20C1	PAL20L2
PAL20L8	PAL20L10	PAL20R4
PAL20R6	PAL20R8	PAL20RA10
PAL20RS4	PAL20RS10	PAL20S10
PAL20X4	PAL20X8	PAL20X10
PAL22RX8	PAL22V10	PAL32R16
PAL32VX10	PAL64R32	PMS14R21

Using a PALASM Device

When a PALASM device is selected, CUPL will generate a functionally-equivalent PALASM file, which can then be translated into an XNF file.

When a device is selected that PALASM does not recognize, CUPL will insert **USER** as the device name in the PALASM file. PDS2XNF recognizes this **USER** device. Some PLD macro configurations which are not supported by PALASM will be ignored or translated into a PALASM equivalent. This will generate a warning message. For example, if the extension **CKMUX** is used, it is considered to be a **CK** extension, and a warning message is generated. If the extension **DFB** is used, it is ignored and a warning message is generated.

Source Files for XILINX Devices.

The following discusses how to create a new CUPL source file for XILINX devices.

When a new CUPL source file is created, any device on the CUPL device list can be specified. The imaginary **PLD9000** or the limitless **VIRTUAL** device can also be specified. Refer to the device support section of the CUPL manual for details on these devices.

The CUPL source file remains the same except for the following few differences:

Pin numbers are not needed when using the **VIRTUAL** device. The **PIN** keyword is still used, however, to assign names to the pins of a device that will be used later. For example:

```
pin = Input0;
```

The keyword **NODE** is not used with the **VIRTUAL** device. The keyword **PINNODE** is, however, still used.

Using CSIM 2

This chapter explains how to use **CSIM** program to create test vectors for the programmable logic device under design. Test vectors specify the expected functional operation of a PLD by defining the outputs as a function of the inputs. Test vectors are used both for simulation of the device logic before programming and for functional testing of the device once it has been programmed. **CSIM** can generate JEDEC-compatible downloadable test vectors.

❑ INPUT

A test specification source file (***filename*.SI**) is the input to **CSIM**. It contains a functional description of the requirements of the device in the circuit.

The source file may be created using a standard text editor like DOS EDLIN or WordStar in non-document mode.

The input pin stimuli and output pin test values entered in the source file are compared to the actual values calculated from the logic equations in the CUPL source file. These calculated values are contained in the absolute file (***filename*.ABS**), which is created during CUPL operation when the **-a** flag on the command line is specified. The absolute file must be created during CUPL operation before running **CSIM**.

CSIM must also be able to access the device library file, **CUPL.DL**, which contains a description of each of the target devices supported in the current version of **CSIM**.

The library describes the physical characteristics of each device, including internal architecture, number of pins, and

type of registers available, and the logical characteristics, including registered and non-registered pins, feedback capabilities, register power-on state and register control features.

Reference the target device using device mnemonics. Each mnemonic is composed of a device family prefix and industry-standard part number suffix. Table 2-1 lists the device mnemonic prefixes.

Table 2-1. CSIM Device Mnemonic Prefixes

Prefix	Device Family
EP	Erasable Programmable Logic Device (EPLD)
G	Generic Array Logic (GAL)
F	Field Programmable Logic Array (FPLA)
F	Field Programmable Gate Array (FPGA)
F	Field Programmable Logic Sequencer (FPLS)
F	Field Programmable Sequence Generator (FPSG)
P	Programmable Logic Array (PAL)
P	Programmable Logic Device (PLD)
P	Programmable Electrically Erasable Logic (PEEL)
PLD	P-CAD Logical Device
RA	Bipolar Programmable Read-Only Memory (PROM)

For example, the device mnemonic for a PAL10L8 is P10L8; for an 82S100 the device mnemonic is F100. For bipolar PROMs, the suffix is the array size. For example, the device mnemonic for a 1024 x 8 bipolar PROM is RA10P8, since there are 10 address input pins and 8 data output pins.

❑ OUTPUT

The simulator output is the following two files: a simulation listing file and an optional JEDEC downloadable fuse link file.

A simulation listing file (***filename*.SO**) contains the results of the simulation. It has the same filename as the input test specification file.

All header information is displayed in the listing file with any header errors marked appropriately. Each complete vector is assigned a number. Any output tests that failed are flagged with the actual (simulator-determined) output value displayed. Each variable in error is listed along with the expected (user-supplied) value. Any invalid or unexpected test values are listed along with an appropriate error message.

The simulator output listing can also be output to the screen (using the **-v** option on the command line).

An optional JEDEC downloadable fuse link file (***filename*.JED**) contains structured test vectors. **CSIM** appends the test vectors to an existing ***filename*.JED** created during CUPL operation.

Note

CSIM does not support multi-device files as does CUPL. **CSIM** only simulates the first device of a multi-device file.

❑ RUNNING CSIM

Run **CSIM** using the following command line format:

csim [-flags] [library] [device] source

where

-flags is the following set of simulator options:

-l create listing file.

-j append test vectors to JEDEC file.

-n use source filename for JEDEC file.

-v display simulation results to terminal.

-u use specified library for simulation.

-w (MS-DOS only) simulate and display output file in waveform.

-d (MS-DOS only) display an existing simulation output file in waveform.

library is the library name and path name if the **-u** flag is being used to specify a library other than the default library.

device must be the same device mnemonic as was used in the CUPL compilation. Specifying the device is optional; if a device is not specified, **CSIM** uses the device CUPL compiled (contained in the **.ABS** file).

source is the user-created ASCII test specification file (***filename*.SI**). The extension **.SI** is assumed for the source file and may be omitted when giving the **CSIM** command.

Note

The square brackets indicate optional items.

❑ Simulator Option Flags

Multiple option flags can be specified when running **CSIM**. A hyphen must be used before the first flag entered, but can be omitted for subsequent flags. Spaces may also be placed between the flags. For example, the following two **CSIM** command lines are equivalent:

csim -l -v -j p16r4 waitgen Return

csim -lvj p16r4 waitgen Return

CSIM can be typed without any flags, to see the command line format and a list of the option flags.

Table 2-2 lists descriptions of the **CSIM** option flags.

Table 2-2. Simulator Option Flags

Option Flag	Description
-j	Appends the structured test vectors generated by the simulation onto the existing JEDEC download file.
-l	Generates a simulation listing file (*filename*.SO.) The input and output values for each variable are listed. Error messages are listed following each vector, with the signal name in error displayed.

-n	Allows the source filename to be used as the JEDEC filename instead of using the name in the NAME field of the source file.
-v	Displays the contents of the listing file to the screen. When the simulation data begins to appear on the screen, type Ctrl-S to stop the display (and any key to start it again) or Ctrl-C to cancel the simulation.
-u	Overrides the default device library specified in the environment. Specify the complete path and library name. This option is of particular use on systems that have special libraries created for unique or custom devices.
-d	(MS-DOS only) Displays an existing simulation output file in waveform. Type F1, F2, F5, F6, ↑, ↓, ←, → keys to view the waveform output. Type F4 to exit.
-w	(MS-DOS only). Generates a simulation listing file and displays the output in waveform. Use the F1, F2, F5, F6, ↑, ↓, ←, → keys to view the waveform output. Type F4 to exit.

❑ Viewing Waveform(MS-DOS)

Running CSIM with the -w or -d flag generates waveform output on the screen. The view of the waveform can be changed by using the following keys:

→	Scroll right
←	Scroll left
↓	Scroll down
↑	Scroll up
F1	Decrease scale horizontally
F2	Enlarge scale horizontally
F4	Exit to DOS
F5	Shift screen left
F6	Shift screen right
F7	Change signal orders
F8	Group signals into bus
F9	Create Waveform Hardcopy
F10	Waveform Legend

Change Signal Order

The CSIM waveform display allows signal orders to be changed. To change signals, press the F7 key. The cursor appears on the signal window instead of the waveform window. Position the cursor over the desired signal, and then press [Return] to select the signal. The selected signal is indicated as <Sel: signal-name> on the lower right portion of the screen (Refer to Figure 1). Move the cursor to the

desired position and press the [Insert] key. The selected signal is inserted into the cursor position, and the signals below the cursor are shifted one position down. As an example, refer to Figure 1.

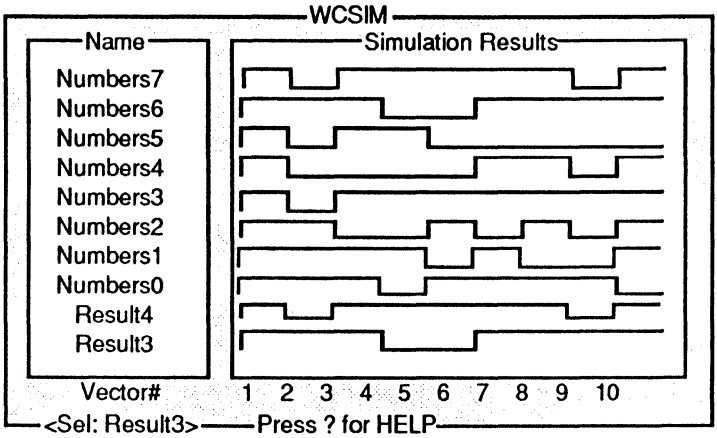


Figure 1. Moving a Signal

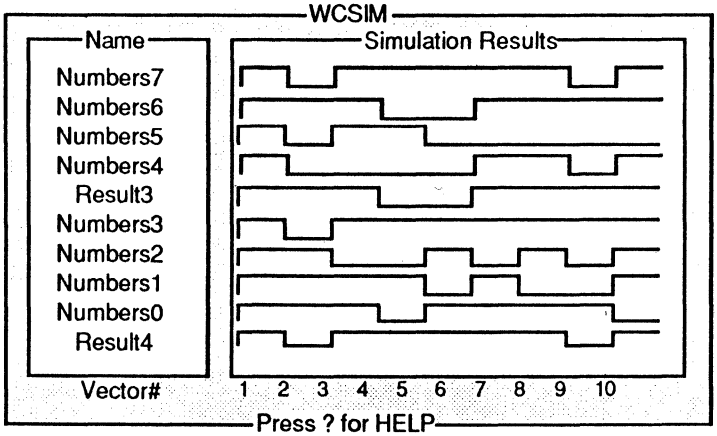


Figure 2. Signal After Being Moved

In this figure, Result3 is selected. Result3 is to be placed between Numbers3 and Numbers4. Move the cursor to Numbers4 and press

[Insert]. Figure 2 shows the result. To quit the change signal mode, press [Escape]. The cursor returns to the waveform window.

Group Signals into Bus

Grouping signals into bus is another useful feature. This feature allows the grouping of up to eight signals into a bus, and the hex value can be displayed on the screen. Figure 4 shows the grouping of Numbers0 to Numbers7 into a bus called INPUT_BUS.

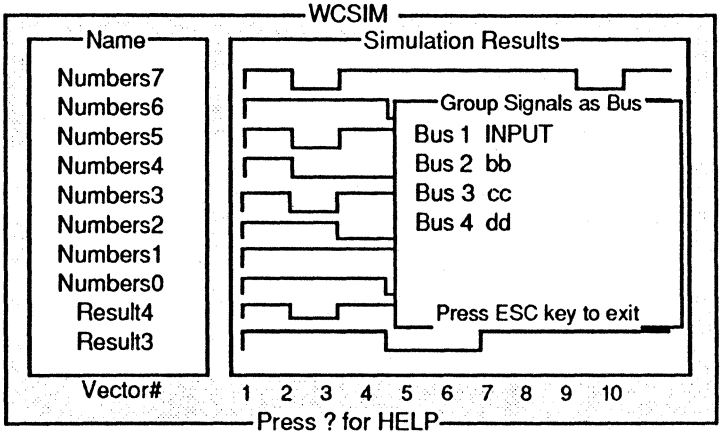


Figure 3. Making a Bus

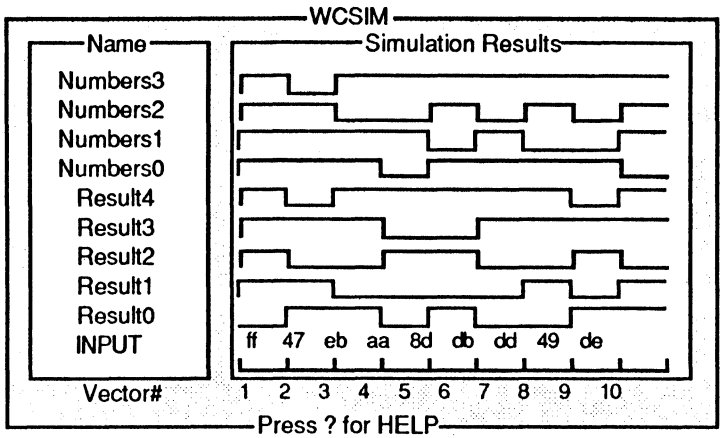


Figure 4. Moving a Bus

To group signals, first press F8, and a bus window pops up onto the screen (see Figure 3). Type the bus name INPUT_BUS at Bus1 and press [Return]. The cursor moves to the signal window. Move the cursor to Numbers0 and press [Return]. This signal is grouped into bus INPUT_BUS as the least significant bit. Now move the cursor to Numbers1 and press [Return]. This becomes the second least significant bit of INPUT_BUS. Continue this procedure until the eighth signal, Numbers7, is selected. After all signals are selected, a new bus-type signal called INPUT_BUS is placed after the last signal, as shown in Figure 4.

If fewer than eight signals are to be grouped into a bus, press [Escape] after all the desired signals have been selected. The new bus-type signal is placed after the last signal.

The maximum number of busses that can be created if four, and the maximum number of signals that can be grouped into a bus is eight. A sixteen-signal bus can be created as two busses of high-order and low-order.

Create Waveform Hardcopy

The F9 function key does not cause an immediate print-out. When activated, F9 creates a file with an extension **.prt**. This file can then be sent to the printer. The printer must be capable of handling extended ASCII characters.

Help Menu

The question mark (**?**) key can be used to bring up the help menu. This menu provides a description of the function keys.

The following figure defines the waveforms that may appear during view waveform. The waveform legend screen appears when the F10 key is depressed.

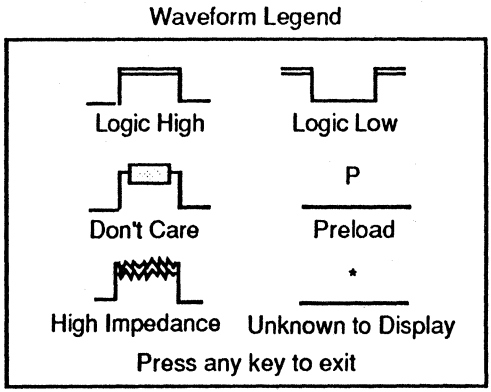


Figure. Waveform Legend

❑ TEST SPECIFICATION FILE

The test specification file (***filename*.SI**) may be created using a text editing program. The filename is the same as the corresponding **CUPL** logic description source file. Put the following information into the test specification file:

➢ Header information

- Comments
- Variable ordering
- Base sets
- Test vectors
- Simulator directives

❑ Header Information

Header information which is entered must be identical to the information in the corresponding **CUPL** logic description file. If any header information is different, a warning message appears, stating that the status of the logic equations could be inconsistent with the current test vectors in the test specification file. Table 2-3 lists the keywords used for header information (see the subtopic, Header Information in Chapter 1):

Table 2-3. CSIM Header Keywords

PARTNO	**NAME**
REVISION	**DATE**
DESIGNER	**COMPANY**
ASSEMBLY	**LOCATION**
DEVICE	**FORMAT**

When creating a test specification file, begin by copying the contents of the corresponding **CUPL** source file to the test specification file, to assure proper header information. Then delete everything except the header information from the test specification file.

❑ Comments

Comments can be placed anywhere within the test specification file. Comments can be used to explain the contents of the specification file or the function of certain test vectors. A comment begins with a slash-asterisk (/*) and ends with an asterisk-slash (*/). Comments can span multiple lines and are not terminated by the end of a line. However, comments cannot be nested.

❑ Statements

CSIM provides the keywords, **ORDER**, **BASE**, and **VECTORS** to write statements in the source file that determine the simulation output and how it is displayed. The following sections describe how to write statements with the CUPL keywords.

ORDER Statement

Use the **ORDER** keyword to list the variables to be used in the simulation table, and to define how they are displayed. Typically, the variable names are the same as those in the corresponding **CUPL** logic description file.

Place a colon after **ORDER**, separate each variable in the list with a comma, and terminate the list with a semicolon. The following is an example of an **ORDER** statement:

ORDER: inputA, inputB, output;

Only those variables that are actually used in the simulation must be listed.

The polarity of the variable name can be different than was declared in the **CUPL** logic description file, allowing simulation of active-LO outputs with an active-HI simulation vector. The variable names can be entered in any order; **CSIM**

automatically creates the proper order and polarity of the resulting vector to match the requirements of the JEDEC download format for the device.

When indexed variables are used in the **ORDER** statement, they can be expressed in list notation format. However, since the **ORDER** statement is already in list form, square brackets are not needed to delimit the **ORDER** set. The following is an example of two equivalent **ORDER** statements; the first statement lists all the variables, and the second is written in list form.

```
ORDER: A0, A1, A2, A3, SELECT, !OUT0, !OUT1;
ORDER: A0..3, SELECT, !OUT0..1 ;
```

In list notation format, the polarity of the first indexed variable (**!OUT0** in the above example) determines the polarity for the entire list.

Bit fields that are declared in the **CUPL** logic description file can be referenced by their single variable name. Bit fields can also be declared in the test specification file for **CSIM**, using **FIELD** declaration statements (see Bit Field Declaration Statements in Chapter 2). The **FIELD** statement must appear before the **ORDER** statement.

The **ORDER** statement can be used to specify the format of the vector results in the simulator listing file (or on the screen if screen output is specified.) By default, variable values are displayed without spaces between columns. For example, the following **ORDER** statement

```
ORDER: clock, input, output ;
```

generates the following display in the output file (using sample values):

0001: C0H

0002: C1L

Spaces can be inserted between columns by using the % symbol and a decimal value between 1 and 80. For example, the following **ORDER** statement

ORDER: clock, %2, input, %4, output ;

generates the following display in the output file:

0001: C 0 H

0002: C 1 L

Text can be inserted into the output file by putting a character string, enclosed by double quotes (" "), into the **ORDER** statement. For example, the following **ORDER** statement

ORDER: "Clock is ", clock,
" and input is ", input,
" output goes ", output ;

produces the following result in the output file:

0001: Clock is C and input is 0 output goes H

0002: Clock is C and input is 1 output goes L

BASE Statement

In most cases, each variable in the **ORDER** statement (except for **FIELD** variables) has a corresponding single character test value that appears in the test vector table of the output file. Multiple test vector values can be represented with quoted numbers. Use single quotes for input values and double quotes for output values. Enter a **BASE** statement to specify how each

quoted number is expanded. The format for the **BASE** statement is:

BASE: name

where

name is either octal, decimal or hex.

Follow **BASE** with a colon.

The default base for quoted test values is hexadecimal. The **BASE** statement must appear in the file before the **ORDER** statement.

If the base is decimal or hexadecimal, quoted numbers expand to four digits; if the base is octal, they expand to three digits. For example, a test vector entered as '7' is interpreted as follows:

```
111        Base is octal
```

or

```
0111       Base is decimal
```

or

```
0111       Base is hex
```

More than one hexadecimal or octal digit may be entered between quotes. For example, '563' expands to the following:

```
101 110 011          Base is octal
```

or

```
0101 0110 0011       Base is decimal
```

or

```
0101 0110 0011       Base is hex
```

Quoted values may also be used with all other test values. For example, if the base is set to octal

"XX"	expands to X X X X X X
"LL"	expands to L L L L L L
"45"	expands to H L L H L H

Test values for **FIELD** variables can be expressed either individually (for example, 001, HHLL) or with quoted values (for example, '1', "C"). When quoted values are used, the value is automatically expanded to the number of variables in the field. For example, for the following address field

FIELD address = [A0..5] ;

A test value of

```
/*
   A     A     A     A     A     A
   5     4     3     2     1     0
   ---------------------------------*/
   1     1     1     0     0     1
```

could be written using single test values, or

'39'

using quoted test values.

VECTORS Statement

Use the **VECTORS** keyword to prefix the test vector table. Following the keyword, include test vectors made up of single test values or quoted test values (see the subtopic, Base Statement in this chapter). Each vector must be contained on a single line. No semicolons follow the vector. Table 2-4 lists allowable test vector values.

Table 2-4. Test Vector Values

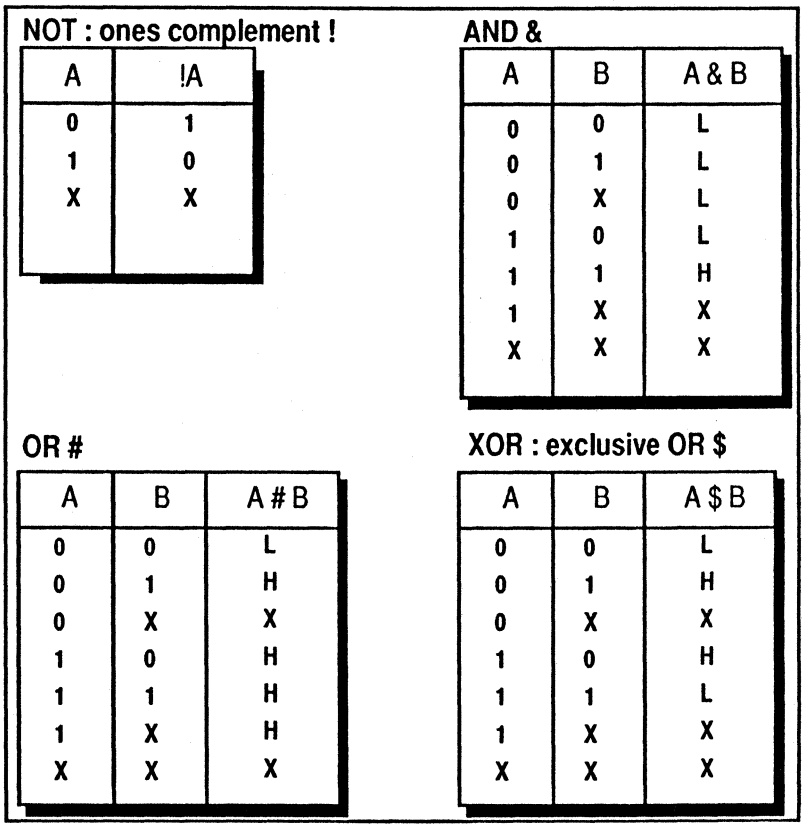
Test Value	Description
0	Drive input LO (0 volts) (negate active-HI input)
1	Drive input HI (+5 volts) (assert active-HI input)
C	Drive (clock) input LO, HI, LO
K	Drive (clock) input HI, LO, HI
L	Test output LO (0 volts) (active-HI output negated)
H	Test output HI (+5 volts) (active-HI output asserted)
Z	Test output for high impedance
X	Input HI or LO, output HI or LO. NOTE: Not all device programmers treat X on inputs the same; some put it to 0, some allow input to be pulled to 1, and some leave it at the previous value.
N	Output not tested
P	Preload internal registers (value is applied to !Q output)
*	Outputs only -simulator determines test value and substitutes in vector
' '	Enclose input values to be expanded to a specified BASE (octal, decimal, or hex). Valid values are 0-F and X.
" "	Enclose output values to be expanded to a specified BASE (octal, decimal, or hex.) Valid values are 0-F, H, L, Z, and X.

The following is an example of a test vector table:

```
VECTORS:
0 0 1 1 1 'F' Z "H"  /* test outputs HI      */
0 1 1 0 0 '0' Z "L"  /* test outputs LO      */
```

Unlike many other simulators, **CSIM** treats the DON'T-CARE (state X) as any other value. State X is not assumed to be 0 on

input and N on the output as with PALASM. The X state allows specific determination of which inputs affect the output value, according to the rules listed in the truth tables in Figure 2-1.



NOT : ones complement !

A	!A
0	1
1	0
X	X

AND &

A	B	A & B
0	0	L
0	1	L
0	X	L
1	0	L
1	1	H
1	X	X
X	X	X

OR #

A	B	A # B
0	0	L
0	1	H
0	X	X
1	0	H
1	1	H
1	X	H
X	X	X

XOR : exclusive OR $

A	B	A $ B
0	0	L
0	1	H
0	X	X
1	0	H
1	1	L
1	X	X
X	X	X

Figure 2-1. Vector Truth Tables

Preload

Use the **P** test value on the clock pin of a registered device to preload internal registers of a state machine or counter design to a known state, if the device does not have a dedicated TTL-level preload pin. The device programmer uses a supervoltage to actually load the registers. All input pins to the device are

ignored and hence should be defined as X. The values that appear for registered variables are loaded into the **!Q** output of the register. These values (0 or 1) are absolute levels and are not affected by output polarity nor inverting buffers. The following is an example of a preload sequence for an active-LO output variable in a device with an inverting buffer between the register Q output and device pin:

```
ORDER: clock, input1, input2, !output;
VECTORS:
PXX1    /* reset flip-flop */
        /* !Q goes to 1 */
        /* Q goes to 0 */
0XXH    /* output is HI due to */
        /* inverting buffer */
```

Note

Although **CSIM** can simulate and generate preload test vectors, not all PLDs are capable of preload using a supervoltage. Some devices have dedicated preload pins to use for this purpose. **CSIM** does not verify whether the device under simulation is actually capable of preload because parts from different manufacturers exhibit different characteristics. Before using the preload capability, determine whether the device being tested is physically capable of supervoltage preloading.

Clocks

Most synchronous devices (devices containing registers with a common clock tied to an output pin) use an active-HI (positive edge triggered) clock. To assure proper **CSIM** operation for these devices, always use a **C** test value (not a 1 or 0) on the clock pin. For synchronous devices with an active-LO (negative edge triggered) clock, use the **K** test value on the clock pin.

Asynchronous Vectors

When writing test vectors for a circuit with asynchronous feedback, changing two test values at once can create a spike condition that produces anomalous results. (See Figure 2-2. It shows the diagram for a circuit with three inputs [A, B, and C] and an output at Y that feeds back.)

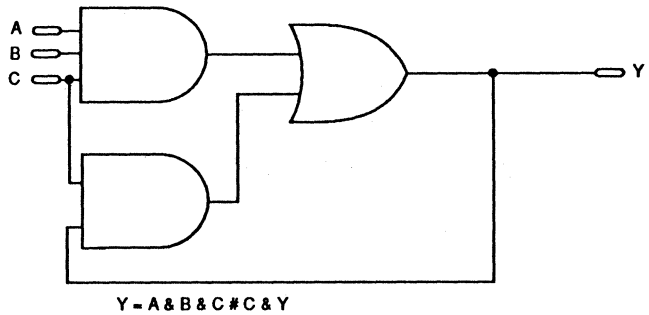


Figure 2-2. Circuit with Feedback

The equation for the output at Y is as follows:

Y = A & B & C # C & Y

The vectors table in Figure 2-3 shows an expected low output at Y based on the specified input values.

	A	B	C	Y
0001	0	0	0	L
0002	0	1	1	L
0003	1	0	1	L

Figure 2-3. Vectors Table for Circuit with Feedback

Because one of the inputs is 0 in each of the vectors, the AND gate defined by A, B, and C produces a low output. The low value feeding back from the Y output keeps the other AND gate low also. Therefore, the OR gate (driven by the output of the two AND gates) and consequently the output at Y remain low for the specified test vectors.

However, when the programmer operates on the test vectors, it applies values serially, beginning with the first pin. Because two test values change between vectors, the programmer creates intermediate results (labeled "a" in Figure 2-4).

	A	B	C	Y
0001	0	0	0	L
0001a	0	1	0	L
0002	0	1	1	L
0002a	1	1	1	H
0003	1	0	1	H

Figure 2-4. Vectors Table with Intermediate Results

The intermediate result, [0002a], produces a high value for the output at Y. This high value feeds back and combines with the "1" value specified for input C in vector [0003] to produce a high output for the AND gate and consequently for the OR gate and for the output at Y. This high value conflicts with the expected low value specified in the third test vector, and the result is a spike condition.

By taking care to always change only one value between test vectors, the spike condition described above can be avoided. Also, in the source specification file, it is possible to specify a **TRACE** value of 1, 2, or 3 (rather than the default value of 0) that instructs **CSIM** to display intermediate results in the output file (see "**TRACE**" in the following section, Simulator Directives).

Simulator Directives

CSIM provides four directives that can be placed on any row of the file after the **VECTOR** statement. All directive names begin with a dollar sign and each directive statement must end with a semicolon. Table 2-5 lists the **CSIM** directives.

Table 2-5. CSIM Directives

$MSG	**$REPEAT**	**$TRACE**
$SIMOFF	**$SIMON**	**$EXIT**

$MSG

Use the **$MSG** directive to place documentation messages or formatting information into the simulator output file. For example, a header for the simulator function table, listing the variable names, may be created. The format is as follows:

$MSG "any text string";

In the output table, the text string appears without the double quotes.

Blank lines can be inserted into the output, for example, between vectors, by using the following format:

$MSG "";

$REPEAT

The **$REPEAT** directive causes a vector to be repeated a specified number of times. Its format is:

$REPEAT n;

where

n is a decimal value between 1 and 9999.

The vector following the **$REPEAT** directive is repeated the specified number of times.

The **$REPEAT** directive is particularly useful for testing counters and state transitions. Use the asterisk (*) to represent output test values supplied by **CSIM**. The following example

shows a a 2-bit counter from a CUPL source file, and a **VECTORS** statement using the **$REPEAT** directive to test it.

From CUPL:

```
Q0.d = !Q0 ;
Q1.d = !Q1 & Q0 # Q1 & !Q0 ;
```

In CSIM:

```
ORDER: clock, input, Q1, Q0 ;
VECTORS:
00XX      /* power-on condition          */
PX11      /* reset the flip-flops   */
00HH
$REPEAT 4 ;       /* clock 4 times          */
C0**
```

The above file generates the following test vectors:

```
00XX
PX11
00HH
C0LL
C0LH
C0HL
C0HH
```

CSIM supplies four sets of vector values.

$TRACE

Use the **$TRACE** directive to set the amount of information that CSIM prints for the vectors during simulation. The format is

```
$TRACE n ;
```

where

n is a decimal value between 0 and 4.

Trace level 0 (the default) turns off any additional information and only the resulting test vectors are printed.

When non-registered feedback is used in a design, the value for the output feeding back is unknown for the first evaluation pass of the vector. If the new feedback value changes any output value, the vector is evaluated again. All outputs must be identical for two passes before the vector is determined to be stable.

Trace level 1 prints the intermediate results for any vector that requires more than one evaluation pass to become stable. Any vector that requires more than twenty evaluation passes is considered unstable.

Trace level 2 identifies three phases of simulation for designs using registers. The first phase is "Before the Clock," where intermediate vectors using non-registered feedback are resolved. The second phase is "At the Clock," where the values of the registers are given immediately after the clock. The third phase is "After the Clock," where the outputs utilizing feedback are resolved as in trace level 1.

Trace level 3 provides the highest level of display information possible from **CSIM**. Each simulation phase of "Before Clock," "At Clock," and "After Clock" is printed and the individual product term for each variable is listed. The output value for the AND gate is listed along with the value of the inputs to the AND array.

Trace level 4 provides the ability to watch the logical value before the output buffer. Using **$TRACE 4, CSIM** only reports the true output pin values, and assigns a "?" to inputs and buried nodes. For combinatorial output, trace level 4 displays the results of the OR term. For registered outputs, trace level 4 shows the Q output of the register.

The following example uses a p22v10:

```
pin 1 = CLK;
pin 2 = IN2;
pin 3 = IN3;
....
pin 14 = OUT14;
pin 15 = OUT15;
....
OUT 14.D = IN2;
OUT 14.AR = IN3;
OUT 14.OE = IN4;
....
```

Figure 2-5. Using P22V10.

Figure 2-6 shows the simulation result file:

```
order  CLK, IN2, IN3, IN4, . OUT14, OUT15 . ;
******before output buffer******
     ???? .. LL ...
0001:0011 .. HH ...
.....
******before output buffer******
     ????   HH...
0004 C100...ZZ
.....
```

Figure 2-6. Simulation File.

Figure 2-7 shows the virtual observation points when using trace level 4 with either a combinatorial configuration or a register configuration.

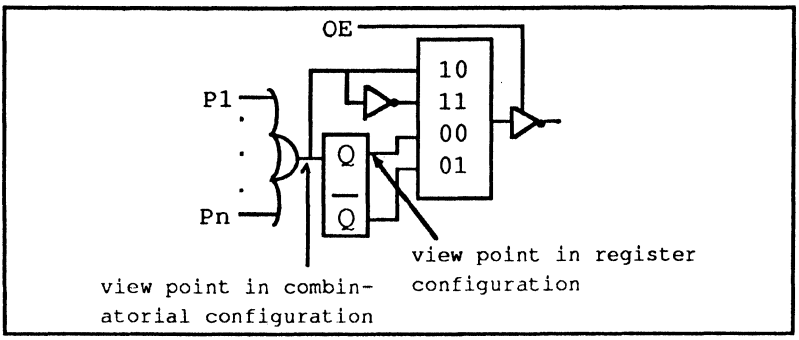


Figure 2-7. Observation Points Using Trace Level 4.

$EXIT

Use the **$EXIT** directive to abort the simulation at any point. Test vectors appearing after the **$EXIT** directive are ignored. This directive is useful in debugging registered designs in which a false transition in one vector causes an error in every vector thereafter.

Placing a **$EXIT** command after the vector in error directs attention to the true problem, instead of to the many false errors caused by the incorrect transition.

$SIMOFF

Use the **$SIMOFF** simulator directive to turn off test vector evaluation. Test vectors appearing after the **$SIMOFF** directive are only evaluated for invalid test values and the correct number of test values. This directive is useful in testing asynchronously clocked designs in which **CSIM** is unable to correctly evaluate registered outputs.

$SIMON

Use the **$SIMON** simulator directive to cancel the effects of the **$SIMOFF** directive. Test vectors appearing after the **$SIMON** directive are evaluated fully.

❑ Fault Simulation

An internal fault can be simulated for any product term, to determine fault coverage for the test vectors. The format for this option is as follows:

STUCKL n ;

or

STUCKH n ;

where

n is the JEDEC fuse number for the first fuse in the product term.

The documentation file (***filename*.DOC**) fuse map lists the fuse numbers for the first fuse in each product term in the device.

Format 1 forces the product term to be stuck-at-0.

Format 2 forces the product term to be stuck-at-1. The **STUCK** command must be placed between the **ORDER** and **VECTORS** statements.

Using CBLD 3

CBLD permits the maintaining and personalizing of CUPL device libraries. It can list the contents of a library and build new libraries from existing ones.

A CUPL device library is a file containing descriptions of one or more PLDs. **CUPL** and **CSIM** use device libraries to determine the characteristics of target devices. CUPL provides device libraries to describe all the PLDs instead of supplying individual files for each PLD. The advantage of device libraries is that they are easier to keep track of than multiple device files and they enable quicker program operation.

The use of device libraries also resolves compatibility problems between the compiler and device descriptions. Each version of CUPL and its corresponding device library is given a matching key. If a library is used that has a different key than the CUPL compiler, an error message is generated during compilation and the compilation is aborted.

The library that is created when the BUILD batch file is executed, **CUPL.DL**, contains descriptions for each device that is supported by the current version of CUPL.

❑ RUNNING CBLD

Run **CBLD** with the following command line format:

cbld [-flags] [build] [library] [devices]

where

-flags is the following set of options:

-b generate library using build file.

-e list allowable extensions.

-l list long contents of library.

-m list allowable macros by pin.

-t list short contents of library.

-u use specified library for listings.

build is the name of a file to be used with the **-b** option flag

library is a device library name and path name to be used with the **-u** option flag.

devices is one or more device names to be used with the **-t** or **-l** options.

Note

The square brackets indicate optional items.

CBLD can be typed without any flags to see the command line format and a list of the option flags.

CBLD provides two distinct functions: listing the contents of a library (the **-t, -l** and **-u** options) and building a new library (the **-b** option). These two functions are described in the following sections.

LISTING THE CONTENTS OF A LIBRARY

To list the contents of a device library, use one of the following command line formats:

cbld -t [-u library] [devices]

cbld -l [-u library] [devices]

where

library is a device library name and path name.

devices is one or more device names.

Note

The square brackets indicate optional items.

Use the **-t** flag to specify the short form, which lists only the device names and the revision number of the library.

Use the **-l** flag to specify the long form, which lists the revision number of the library and the names of the devices, and provides the revision number, the number of pins, the number of fuses, and the number of product terms for each device.

By default, **CBLD** lists the entire contents of the library; however, certain devices can be specified to be listed. For example, by typing:

cbld -l p16h8 p16l8 p16hd8 p16ld8 Return

CBLD lists the characteristics of the p16h8, p16l8, p16hd8 and p16ld8 devices in the **CUPL.DL** library as follows:

Device	Rev	Pins	Fuses	Pterms
pl6h8	08	20	2048	64
pl6l8	08	20	2048	64
pl6hd8	06	20	2048	64
pl6ld8	06	20	2048	64

Use the **-u** flag and a library name with either the **-l** or **-t** flag to list the contents of a library other than the default library specified in the environment file.

For example, to see the contents of a library named **small.dl**, type:

cbld -l -u small.dl Return

❑ LISTING ALLOWABLE EXTENSIONS

The **-e** flag allows the listing of the valid extensions for devices. For example:

cbld -e p22v10

The screen will display:

Device	Extension
p22v10	OE D AR SP

This shows that the valid extensions for the P22V10 are:

OE	output enable control
D	D-type register
AR	asynchronous reset
SP	synchronous reset

If more than one device type must be listed, specify multiple device names. For example:

cbld -e p22v10 ep600

The screen will display:

Device	Extensions
ep600	OE D T AR CK IO DFB TFB
p22v10	OE D AR SP

If a particular device is not specified, **CBLD** will, by default, list all the devices of the library.

❑ LISTING ALLOWABLE MACROS

When defining device logic using schematic capture, only certain component symbols (macros) can be used on any given device pin.

To list the macros that can be used on a device pin, use the following command line format:

cbld -m [-u library] [devices]

where

library is a device library name and path name.

devices is one or more device names.

Note

The square brackets indicate optional items.

The **-m** flag lists, for each device, the pin number and the macro number.

By default, **CBLD** lists the entire contents of the library; however, certain devices can be specified to be listed. For example, by typing:

cbld -m p16L8 Return

CBLD lists the macros for the p16L8 device in the **CUPL.DL** library as follows:

Device	Pin	Macros
P16L8	1	0
	2	0
	3	0
	4	0
	5	0
	6	0
	7	0
	8	0
	9	0
	10	0
	11	0
	12	1 2 4 6
	13	0 1 2 3 4 5 6 16 17
	14	0 1 2 3 4 5 6 16 17
	15	0 1 2 3 4 5 6 16 17
	16	0 1 2 3 4 5 6 16 17
	17	0 1 2 3 4 5 6 16 17
	18	0 1 2 3 4 5 6 16 17
	19	1 2 4 6

20

❑ BUILDING DEVICE LIBRARIES

To build a new library from an existing library, first create an ASCII build file (***filename*.BLD**) that specifies the name of the new library, the name of the source library, and the devices to use from it. Any text editor can be used, once it can produce a standard ASCII Text file.

Note

The vertical line (|) separates items in the syntax that are mutually exclusive. Do not type it in.

The build file uses the following syntax:

```
TARGET  library
SOURCE  library1
devices | *
SOURCE  library2
devices | *
.                    .
.                    .
.                    .
```

where

TARGET is a keyword to identify the new library.

SOURCE is a keyword to identify the source libraries.

library is a target library name.

library1 and **library2** are source library names.

devices are device names contained in the source libraries.

* is a "wildcard" representing all devices in the specified library.

When building a new library, devices from more than one source library can be specified by using more than one **SOURCE** keyword.

Comments may be used in the build file. Enclose comments between the /* and */ symbols.

To compile the build file, use the following command line format:

cbld -b *filename*.bld

Note

There is no standard extension for build files. The **.BLD** extension is for mnemonic convenience only. When running **CBLD**, the entire filename, including the extension, must be typed in.

The following example shows the contents of a build specification file, **small.bld**. **Small.bld** specifies that a new library, **smallib.dl**, be built containing the devices p22v10, f105, and p16rp8 from the **cupl.dl** library, and all the devices in the **biglib.dl**

```
TARGET smallib.dl

SOURCE cupl.dl
    p22v10
    f105
    p16rp8
SOURCE biglib.dl
    *
```

To run **CBLD** and build the **smallib.dl** library specified in the sample file above, type

cbld -b small.bld Return

Using PTOC 4

PTOC converts PALASM source files into **CUPL** and **CSIM** source files in order to maintain existing designs and develop new ones while using only one universal language.

Use **PTOC** on files that have assembled successfully with a PALASM assembler. The output of **PTOC** is directly compatible with CUPL syntax and may be compiled without modifications.

❑ RUNNING PTOC

Run **PTOC** with the following format:

ptoc palasm_file1 palasm_file2 ...

where

palasm_file1, palasm_file2 are PALASM source files to be translated.

There is no standard file extension for PALASM source files; specify the full filename and extension when running **PTOC**.

For each input PALASM file, **PTOC** generates a CUPL logic description file, ***filename*.PLD**; the filename is the same as the input PALASM filename.

If the PALASM file contains a function table (that is, test vector information) **PTOC** also generates a test specification file, ***filename*.SI**, that can be input to **CSIM**. It is possible to specify more than one file to be translated with a single **PTOC** command line.

For example, to translate the files **COUNT2** and **DECADE**, type the following:

```
ptoc count2 decade
```

PTOC generates the files **COUNT2.PLD** and **DECADE.PLD**. If the PALASM files contain test vectors, **PTOC** also generates **COUNT2.SI** and **DECADE.SI**.

PALASM SOURCE FILE FORMAT

Because of variations in PALASM assemblers, **PTOC** follows the standard set by Monolithic Memories in the PAL Handbook (Third Edition). Figure 4-1 shows a sample PALASM source file in the format required by **PTOC**.

```
PAL16R8                          Line 1, col 1 - PAL part #
PART NO 1234B                    Line 2 - User's part #
SAMPLE                           Line 3 - Name of device
GYRO, SAN JOSE, CA               Line 4 - Company, city, state
CLK I0 I1 I2 I3 I4 I5 I6 I7      Line 5 - Pin list (must start on
GND /OE O1 O2 O3 NC NC           line 5), consists of 20 or 24
NC NC NC VCC                     symbolic names that are
                                 consecutively assigned to pins 1
                                 through 20 (24).
/O1 := /I0*/I1*/I2 + /I1*/I2*/I3     Equations
/O2 := /I0*/I1*/I2 + /I1*/I2*/I4
/O3 := /I0*/I1*/I2 + /I1*/I2*/I5
FUNCTION TABLE                   Keyword (must start in column 1)

I0 I1 I2 I3 I4 I5 CLK /OE 01 02 03   Function table pin list
;I0-I5 CLK /OE 01-03 COMMENTS  Optional comments

--------------------------------     Dashed line (length optional)

LLLLLL C L LLL LOAD ZEROS        Function table vectors,
HHHHHH C L HHH LOAD ONES         one vector per line
HHHHHL C L HHH WHATEVER          followed by optional comments
HHHHLL C L HHH
HHHLLL C L HHH

--------------------------------     Dashed line(length optional)
DESCRIPTION                          Keyword, optional if following
                                     function table (must start
THIS IS A SAMPLE PALASM FILE     in column 1) followed by
FOR A PAL16R8 DEVICE                 comments
```

Figure 4-1. Sample PALASM File with Explanations

The following sections refer back to this sample file to show how **PTOC** converts PALASM files to **CUPL** format.

❑ PTOC .PLD OUTPUT FILE

PTOC generates a **CUPL** source file (**palasm_file.PLD**) for each PALASM file that it translates. The **CUPL** source file is organized into header information, pin assignments, and equations.

❑ Header Information

PALASM header information cannot be directly translated into **CUPL** header information. **PTOC** places dummy header information into the **CUPL** source file, and puts the PALASM header information within comments. Use a text editor to replace the dummy header information with valid information to maintain consistent documentation. Figure 4-2 shows the header information created by translating the PALASM sample file in Figure 4-1.

```
NAME            XXXXX;
PARTNO          XXXXX;
REV             XXXXX;
DATE            XXXXX;
DESIGNER        XXXXX;
COMPANY         XXXXX;
ASSEMBLY        XXXXX;
LOCATION        XXXXX;
/* PAL16R8 */
/* PART NO 1234B */
/* SAMPLE */
/* GYRO, SAN JOSE, CA */
```

Figure 4-2. Header Information Translation

❑ Pin List

PTOC translates the PALASM pin list into CUPL pin assignments beginning at line five in the PALASM source file. CUPL does not translate GND, NC and VCC pins. Figure 4-3 shows the pin list translation from the sample PALASM file in Figure 4-1.

```
PIN 1 = CLK;
PIN 2 = I0;
PIN 3 = I1;
PIN 4 = I2;
PIN 5 = I3;
PIN 6 = I4;
PIN 7 = I5;
PIN 11 = !OE;
PIN 12 = O1;
PIN 13 = O2;
PIN 14 = O3;
```

Figure 4-3. Pin List Translation

❑ Equations

PTOC translates PALASM equations into **CUPL** equations, replacing symbols, and adding parentheses and variable extensions as necessary. Figure 6-4 shows the equation translation for the PALASM sample file shown in Figure 4-1.

```
!O1.D = !I0 & !I1 & !I2 # !I1 & !I2 & !I3 ;
!O2.D = !I0 & !I1 & !I2 # !I1 & !I2 & !I4 ;
!O3.D = !I0 & !I1 & !I2 # !I1 & !I2 & !I5 ;
```

Figure 4-4. Equation Translation

PTOC replaces the *, +, and / symbols in the PALASM file with the CUPL &, # and ! symbols respectively. **PTOC** translates the assignment operator for registered outputs, :=, by adding the .D extension to the variable name on the left side of the assignment statement (see the subtopic, Extensions in Chapter 1).

❑ PTOC .SI OUTPUT FILE

When **PTOC** translates a PALASM file containing a function table, it generates a **CSIM** source file, (**palasm_file.SI**). This **CSIM** source file is organized into header information, pin order, and vectors. The header information is translated in the same manner as the **CUPL** source file header information (see the subtopic, Header Information in this chapter).

PTOC translates the PALASM function table pin list into a **CSIM** pin order list. Figure 4-5 shows the function table pin list translation for the PALASM sample file shown in Figure 4-1.

```
ORDER:
I0, I1, I2, I3, I4, I5, CLK, !OE, O1, O2, O3;
```

Figure 4-5. CSIM Pin Order List

PTOC translates the PALASM function table vectors into a **CSIM** vectors list. Figure 4-6 shows the function table vectors translation for the PALASM sample file shown in Figure 4-1.

```
VECTORS:
000000    C      0      LLL    /* LOAD ZEROS */
111111    C      0      HHH    /* LOAD ONES */
111110    C      0      HHH    /* WHATEVER */
111100    C      0      HHH
111000    C      0      HHH
```

Figure 4-6. Vectors List

❑ Translation Ambiguities

Because of inherent ambiguities in PALASM syntax, certain conditions cannot be accurately translated into CUPL syntax.

Many PALASM assemblers do not verify that the correct assignment statement, "=" for combinatorial outputs and ":=" for registered outputs, is being used. For example, source assemblers treat the following equations in the same manner:

OUT = INA * INB
OUT := INA * INB

PTOC translates them respectively as follows:

OUT = INA & INB
OUT.D = INA & INB

Another ambiguity concerns the use of exclusive OR devices. **PTOC** requires the use of the PALASM exclusive OR symbol ":+:" to explicitly define the position of the exclusive OR in the equation. However, some PALASM assemblers allow an exclusive OR output to be organized by position without explicitly defining the exclusive OR. For example, some PALASM assemblers treat the following two equations in the same manner:

```
OUT := INA * INB      + INC * IND   + INE * INF
OUT := INA * INB      + INC * IND   :+: INE * INF
```

PTOC translates them respectively as follows:

```
OUT.D = INA & INB     # INC & IND   # INE & INF;
OUT.D = INA & INB     # INC & IND   $ INE & INF;
```

The most troublesome ambiguity is in the function table of source files with bi-directional pins. Since the same values are used by PALASM to indicate inputs and outputs during simulation, **PTOC** cannot accurately determine whether a bidirectional pin should be treated as an input or an output. If a PALASM source file has bidirectional pins, check the vector table translation in the source specification file to determine whether **PTOC's** interpretation is correct.

Error Messages A

CUPL error messages are intended to be self-explanatory. This appendix provides additional information describing them.

Some of the CUPL programs, such as **CUPL** and **CSIM**, are composed of individual modules. Error messages are numbered and listed according to the program and module in which they occur. The suffix to the error message number identifies the program and module.

Table A-1. Error Message Module Suffixes

Module	Suffix
CUPL processor	ck
CUPLX preprocessor	cx
CUPLA source file parser	ca
CUPLB equation fitter	cb
CUPLM minimizer	cm
CUPLC fusemap generator	cc
CSIM processor	sk
CSIMA logic simulator	sa
CBLD device library manager	ba
PTOC PALASM translator	pt

This appendix lists the error messages by modules in the same order as they appear in Table A-1 above. The error messages within each module are listed in numerical order.

CUPL provides three levels of error messages: warnings, errors, and fatals.

warnings — do not prevent CUPL from continuing, but indicate a problem that should be corrected.

errors — allow CUPL to continue but must be corrected before future compiles.

fatals — prevent CUPL from continuing and must be corrected.

Note

Error messages with indexes greater than 1000 are program errors. This section does not individually list program errors. Possible causes for program errors are bad data in a source file caused by disk errors or word processors in document mode; or previous errors continuing to propagate unexpected circumstances. If the cause of a program error cannot be determined, gather as much information as possible on the conditions in effect when the error occurred, then call CUPL support.

Error messages report the line number on which the error was detected; however, the cause of the error may be on a previous line. If the message doesn't seem to apply to the reported line, look at preceeding lines for the source of the error.

❑ CUPL ERROR MESSAGES

This section describes the errors for the **CUPL, CUPLX, CUPLA, CUPLB, CUPLM,** and **CUPLC** modules.

❑ CUPL Module Error Messages

0001ck could not open: "filename"

Fatal. CUPL cannot continue because of the failure to open the indicated file. Be sure the file exists if it is an input.

0002ck could not execute program: "program name"

Fatal. CUPL is unable to perform the next step in the compilation. Be sure that all of the CUPL program files exist on the same directory or disk.

0003ck could not find PATH in ENVIRONMENT

Fatal. The PATH assignment has not been made in the ENVIRONMENT.

0004ck could not find LIBCUPL in ENVIRONMENT

Fatal. The LIBCUPL assignment has not been made in the ENVIRONMENT.

0005ck could not find program: "program name"

Fatal. CUPL is unable to locate the CUPL programs using the PATH in the ENVIRONMENT.

0006ck insufficient memory to execute program: "filename"

Fatal. Not enough program storage available to load and execute the program. Refer to Chapter 1, "Introduction," for the minimum memory requirements for the configuration being used.

0007ck invalid flag: "option flag"

Fatal. The option flag specified is not one of the allowable compilation flags. Verify proper command line flags and syntax as discussed in Chapter 2, "Using CUPL."

0008ck out of memory: "condition"

Fatal. CUPL has used all available RAM memory which has been allocated by the operating system. Check for the existence of print spoolers, RAM disks, or other memory-resident programs which may decrease the amount of memory available to the CUPL application.

10xxck program error: "specifics"

Fatal. An operating system interface problem is suspected. Contact Logical Devices, Customer Support.

❑ CUPLX Module Error Messages

0001cx could not open: "filename"

Fatal. CUPLX cannot continue because of the failure to open the indicated file. Be sure the file exists if it is an input.

0002cx could not execute program: "program name"

Fatal. CUPLX is unable to perform the next step in the compilation. Be sure that all of the CUPL program files exist on the same directory or disk.

0003cx no label given for command

Error. One of the preprocessor commands, $DEFINE, $UNDEF, $IFDEF, or $IFNDEF, was used without a succeeding label.

0004cx already defined: "label"

Error. The label was previously defined using $DEFINE. To redefine the label, first use $UNDEF to undefine the label, and then use $DEFINE to redefine it.

0005cx string error

Fatal. All preprocessor label string space has been used.

0006cx $else without $ifdef

Error. An $ELSE preprocessor command was used without being preceded by an $IFDEF or $IFNDEF command.

0007cx $endif without $ifdef

Error. An $ENDIF preprocessor command was used without being preceded by an $IFDEF or $IFNDEF command.

0008cx $ifdef nesting too deep

Error. The level of $IFDEF nesting exceeded twelve.

0009cx missing $endif

Error. An $IFDEF preprocessor command was used without being succeeded by an $ENDIF command.

0010cx invalid preprocessor command: "$command"

Error. The preprocessor command is unknown. Refer to Preprocessor Commands in Chapter 2 for a list of valid commands.

0011cx disk write error: "filename"

Fatal. CUPLX encountered an I/O error trying to write the indicated file. This error usually occurs when there is insufficient disk space.

0012cx out of memory: "condition"

Fatal. CUPLX has used all the available RAM memory allocated by the operating system.

0013cx illegal character: "hex value"

Error. CUPLX has encountered an illegal ASCII value in the source file. Make sure the file was created in nondocument mode on the word processor. This error

can also be caused by files which were created over a serial modem upload/download link.

0014cx unexpected symbol:"symbol"

Fatal. CUPLX encountered a symbol that it was not expecting. This occurs when certain symbols are expected in a particular order and are either incorrect, misplaced or misspelled.

0015cx Repeat nesting too deep

Fatal. The level of Repeat nesting exceeded two.

0016cx duplicate Macro function name:"function"

Error. The Macro function name has already been previously defined. A duplicate Macro name will cause confusion when they are called.

0017cx missing Macro name

Fatal. A Macro was defined without a name. This macro will never be accessed.

0018cx incorrect number of parameters

Fatal. The number of parameters defined in the Macro function did not equal the number of parameters in the macro call. All parameters defined in the Macro function must be defined in the Macro call.

0019cx out of range

Fatal. The index number exceeded 1023. Valid index numbers are 0 - 1023.

0020cx internal stack overflow

Fatal. A mathematical expression was too complex for CUPLX to handle. The expression can be reduced by eliminating as many parenthetical expressions as possible. Expressions are evaluated from left to right

using standard precedence. The user should take advantage of this.

0021cx expression contains undefined symbol:"symbol"

Fatal. A symbol appearing in the expression has not been defined in the source file or predefined by CUPL.

0022cx invalid library access key

Fatal. The version of CUPLX is not compatible with the version of the device library file. This occurs when either CUPLX or the device library, but not both, has been updated.

0023cx invalid library interface

Fatal. Either the device library was not created using the CUPL library manager, CBLD, or CUPLX and the device library are not compatible.

0024cx bad library file: "library"

Fatal. Either the device library does not exist or the contents of the device library have been damaged.

10xxcx program error: "specifics"

Fatal. An operating system interface problem is suspected. Contact Logical Devices customer support.

❑ CUPLA Module Error Messages

0001ca could not open: "filename"

Fatal. CUPLA cannot continue because of the failure to open the indicated file. Be sure the file exists if it is an input.

0002ca invalid number: "number"

Error. Either the number is used improperly, or a previous syntax error caused the number to be used improperly.

0003ca invalid library access key

Fatal. The version of CUPLA is not compatible with the version of the device library file. This occurs when either CUPLA or the device library, but not both, has been updated.

0004ca invalid library interface

Fatal. Either the device library was not created using the CUPL library manager, CBLD, or CUPLA and the device library are not compatible.

0005ca bad library file: "library"

Fatal. Either the device library does not exist or the contents of the device library have been damaged.

0006ca device not in library: "device"

Fatal. Either the specified target device does not exist or an entry has not been made in the device library for the device.

0007ca invalid syntax: "symbol"

Error. Either the symbol is used improperly, or a previous syntax error caused the symbol to be used improperly.

0008ca too many errors

Fatal. CUPLA has encountered more than 30 errors.

0009ca missing: "symbol"

Error. The missing symbol is required to make the specified statement valid.

0010ca vector too wide

Fatal. A variable list has more than 50 members.

0011ca expression already assigned to: "variable"

Error. The variable (either an intermediate or output variable) was previously assigned an expression. Use APPEND to make multiple expression assignments for the same variable.

0012ca vector size mismatch

Error. The number of members in the variable list on the left side of the equation does not match the number of variables on the right side.

0013ca undefined function: "function"

Error. The variable name used as a function reference has no corresponding function definition. Functions must be defined before they can be referenced.

0014ca variable already declared: "variable"

Error. The variable which was previously assigned an expression cannot be reassigned.

0015ca out of memory: "condition"

Fatal. CUPLA has used all available RAM memory which has been allocated by the operating system. Decrease the number of intermediate variables, fields, or numbers in order to reduce the size of the symbol table.

Note

This error is not a result of insufficient product terms in the device to implement a particular expression.

0016ca invalid number of function arguments: "number"

Error. The user has attempted to pass an incorrect number of arguments to the user-defined function. The number of arguments for the function reference does not match the number in the function definition.

0017ca disk write error: "filename"

Fatal. CUPLA encountered an I/O error trying to write the indicated file. This error usually occurs when there is insufficient disk space.

0018ca intermediate var not assigned an expression: "variable"

Error. The intermediate variable was used as an input in an expression without having been assigned an expression. This error often occurs when a pin or intermediate variable in a logic expression is misspelled.

0019ca indexed and non-indexed vars in range or match expression

Warning. A list (or field variable) in a range or match expression contains both indexed (variable names ending in a number) and nonindexed variables. This type of operation cannot produce the expected results because of inability to hold relative bit positions in the field. It is recommended to use all non-indexed variables in a field for portability to future versions of CUPL.

0020ca index too large for range or match operation

Error. The index of a variable in a list or field exceeds the range or match values.

0021ca header item already declared

Error. One of the header statements was duplicated.

0022ca missing header item(s)

Warning. At least one of the header statements is missing.

0023ca invalid range arguments: always true (in range)

Error. A range has been specified which will always be true and is therefore not an actual range. CUPLA attempts to minimize range functions and does not allow a NULL range such as this. This happens with ranges such as [0000..FFFF] for a 16-bit address. This error can also be given if non-indexed list variables are used in a range expression.

0024ca range or match number larger than variable list

Warning. The range or match number exceeds the width of the bit field it is being applied to. Values exceeding the width of the bit field will be ignored.

0025ca range minimization error

Error. The range reduces to always false, that is, none of the bits in the range are active.

0026ca invalid table statement

Error. Input numbers cannot be mapped into more than one output number.

0027ca invalid present state number

Error. The present state number specified is not valid. This error can occur whenever the present state has not been properly defined as a number using the $DEFINE command.

0028ca invalid next state number

Error. The next state number specified is not valid. This error can occur whenever the next state has not been

properly defined as a number using the $DEFINE command.

0029ca invalid flip-flop type for sequence statement: "type"

Error. The flip-flop type for this device cannot be used for building the requested sequential state machine.

0030ca intermediate dependent on itself: "variable"

Error. The intermediate variable was used in the expression defining the same intermediate variable. This error often occurs when an intermediate variable is misspelled or an output pin expression is being defined using feedback without declaring the output variable as a pin.

0031ca invalid minimization level: "level"

Error. The minimization level specified is invalid. Refer to "Running CUPL" in Chapter 2 for valid minimization levels.

0032ca invalid next state: "hex number"

Error. The next state value is invalid. This error can occur whenever the next state has not been properly defined as a number using the $DEFINE command or has not been identified as a present state using the present command.

0033ca multiple asynchronous defaults for state: "hex number"

Error. By definition, only one asynchronous default expression can be assigned for any one state. The resulting expression is the complement of all previous conditional (if) asynchronous expressions.

0034ca multiple synchronous defaults for state: "hex number"

Error. By definition, only one synchronous default expression can be assigned for any one state. The

resulting expression is the complement of all previous conditional (if) synchronous expressions.

0035ca multiple unconditional statements for state: "hex number"

Error. By definition, only one unconditional synchronous statement can be given for any one state.

0036ca device does not support synchronous state machines

Fatal. The device specified for compilation cannot be used with the sequence statement since it does not support registered operations.

0037ca duplicate present state: "hex number"

Error. The present state number was identified in more than one PRESENT command. This can occur when symbolic state names are used to refer to states, but the $DEFINE command, used to define states, assigned the same number to more than one symbolic name.

0038ca target device not specified

Fatal. The user did not specify a target device on the command line and the source file did not contain a DEVICE assignment in the header information.

0039ca line exceeds maximum length

Error. The statement is greater than 256 characters long. Break the line up into shorter statements.

0040ca invalid or duplicate header name: "name"

Fatal. The NAME field in the header information must not be NULL. When more than one device is being defined in a logic description file, the NAME field in the header information must be unique.

0041ca don't care(s) not allowed for decimal number, treated as 0

Warning. "Don't-care" values, "X", are valid only for binary, octal, and hexadecimal numbers.

0042ca range or match list completely don't cared, decoded as 0

Warning. The variable list in a range or match operation has been completley "don't-cared," leaving an empty variable list. The empty variable list will be decoded into a 0.

10xxca program error: "specifics"

Fatal. An operating system interface problem is suspected. Contact Logical Devices, customer support.

❑ CUPLB Module Error Messages

0001cb could not open: "filename"

Fatal. CUPLB cannot continue because of the failure to open the indicated file. Be sure the file exists if it is an input.

0002cb could not execute program: "program name"

Fatal. CUPLB is unable to perform the next step in the compilation. Be sure that all of the CUPL program files exist on the same directory or disk.

0003cb invalid file: "filename"

Warning. The file was not created by the current version of CUPL.

0004cb missing or mismatched parentheses:

Error. The number of open parentheses [(] and close parentheses [)] in the specified statement does not match.

0005cb invalid library access key

Fatal. The version of CUPLB is not compatible with the version of the device library file. This occurs when

either CUPLB or the device library, but not both, has been updated.

0006cb invalid library interface

Fatal. Either the device library was not created using the CUPL library manager, CBLD, or CUPLB and the device library are not compatible.

0007cb bad library file: "library"

Fatal. Either the device library does not exist or the contents of the device library have been damaged.

0008cb device not in library: "device"

Fatal. Either the specified target device does not exist or an entry has not been made in the device library for the device.

0009cb pin/node "number" redeclared: "variable"

Error. The same pin number or variable name was used more than once in a pin declaration statement.

0010cb pin/node "number" invalid output: "variable"

Error. The variable being assigned an output expression was previously declared for an input-only pin.

0011cb unknown extension: "extension"

Error. The extension is unknown or invalid for the particular device. Refer to "Extensions" in Chapter 2 for a list of valid extensions. Check to make sure the device has the capability required.

0012cb pin/node "number" invalid usage: "variable"

Fatal. The pin number assigned to the variable is invalid for the target device specified.

0013cb pin/node "number" invalid output extension or usage: "variable"

Error. Either the extension is used improperly or it is not valid for the assigned pin/node.

0014cb invalid input: "variable" or pin/node "number" invalid input: "variable"

Error. The variable used as an input was previously assigned to an output that is neither bidirectional nor feeds back into the input array.

0015cb device not yet fully supported: "device"

Fatal. There is an entry for the device in the device library, but the device is not fully supported by the current version of CUPL.

0016cb no expression assigned to: "variable"

Warning. The variable requires an output expression assignment. This warning message is commonly given when all outputs in a bank have the same capability (reset, preset, and so on) and not all the variables have been assigned the same expression. It is given to remind the user that all outputs will be affected.

Note

This warning may be suppressed by assigning the variable to 'b'0 or 'b'1 as appropriate.

0017cb out of memory: "conditions"

Fatal. CUPLB has used all available RAM memory that has been allocated by the operating system, typically as a result of performing a DeMorgan or expansion operation on a large expression. If using fixed polarity devices, check to make sure that the pin variable declaration

matches the polarity of the device. Also check whether an intermediate variable which has been expressed in sum-of-product form is being complemented.

Note

This error does not result from insufficient product terms in the device to implement a particular expression.

0018cb missing flip-flop expression for: "variable"

Error. The matching flip-flop expression for a J-K or S-R type flip-flop is missing. Both inputs must have expressions assigned to them. An input may be assigned to 'b'0 or 'b'1 as appropriate.

0019cb DeMorgan's theorem invoked for: "variable"

Warning. DeMorgan's Theorem has been applied to the expression assigned to the variable. Unlike D- or T-type flip-flops, meaningful results are not guaranteed when a DeMorgan equivalent expression is applied to the logic input.

0020cb invalid mix of banked outputs: "variable"

Error. All outputs in a banked group must be used in the same manner. An attempt was made to mix registered and nonregistered output types.

0021cb no expression allowed for: "variable"

Error. Logic expressions are not allowed for reset and preset nodes when the output has been specified as asynchronous. CUPL will generate the proper defaults.

0022cb pin/node "number" conflicting input architectures: "variable"

Error. A fuse-assigned input architecture must be used consistently in all expressions. An attempt was made to specify both fuse options in different expressions.

0023cb disk write error: "filename"

Fatal. CUPLB encountered an I/O error trying to write the indicated file. This error usually occurs when there is insufficient disk space.

0024cb output defined for node which does not exist: "variable"

Error. Variable is defined for a pin or node number which does not exist.

0025cb output mutually excluded by previous output: "variable"

Error. Variable usage is mutually excluded by a previous usage or other output. A shared product term or terms has been defined more than once.

0026cb disk read error, unexpected end of file: "filename"

Fatal. CUPLB encountered an I/O error trying to read the indicated file. This error usually occurs when the file is being read from damaged media.

10xxcb program error: "specifics"

Fatal. An operating system interface problem is suspected. Contact LDI customer support.

❑ CUPLM Module Error Messages

0001cm could not open: "filename"

Fatal. CUPLM cannot continue because of the failure to open the indicated file. Be sure the file exists if it is an input.

0002cm could not execute program: "program name"

Fatal. CUPLM is unable to perform the next step in the compilation. Be sure that all of the CUPL program files exist on the same directory or disk.

0003cm invalid file: "filename"

Warning. The file was not created by the current version of CUPL.

0004cm out of memory: "conditions"

Fatal. CUPLM has used all available RAM memory which has been allocated by the operating system while performing logic reduction.

Note

This error does not result from insufficient product terms in the device to implement a particular expression.

0005cm disk write error: "filename"

Fatal. CUPLM encountered an I/O error trying to write the indicated file. This error usually occurs when there is insufficient disk space.

0006cm invalid library access key

Fatal. The version of CUPLM is not compatible with the version of the device library. This occurs when either CUPLM or the device library, but not both, has been updated.

0007cm invalid library interface

Fatal. Either the device library was not created using the CUPL library manager, CBLD or CUPLM and the device library are not compatible.

0008cm bad library file: "library"

Fatal. Either the device library does not exist or the contents of the device library have been damaged.

0009cm device is not in library: "device"

Fatal. Either the specified target device does not exist or an entry has not been made in the device libary for the device.

00010cm design too complex for this minimization level

Fatal. CUPLM has exceeded the array size allowed on this machine while reducing a particular expression. Specify a more efficient minimization level.

00011cm disk read error, unexpected end of file: "filename"

Fatal. CUPLM encountered an I/O error trying to read the indicated file. This error usually occurs when the file is being read from damaged media.

10xxcm program error: "specifics"

Fatal. An operating system interface problem is suspected. Contact LDI customer support.

❑ CUPLC Module Error Messages

0001cc could not open: "filename"

Fatal. CUPLC cannot continue because of the failure to open the indicated file. Be sure the file exists if it is an input.

0002cc invalid file: "filename"

Warning. The file was not created by the current version of CUPL.

0003cc invalid library access key

Fatal. The version of CUPLC is not compatible with the version of the device library. This occurs when either CUPLC or the device library, but not both, has been updated.

0004cc invalid library interface

Fatal. Either the device library was not created using the CUPL library manager, CBLD, or CUPLC and the device library are not compatible.

0005cc bad library file: "library"

Fatal. Either the device library does not exist or the contents of the device library have been damaged.

0006cc excessive number of product terms: "variable"

Error. The number of product terms needed to implement the logic expression for the given variable exceeds the capacity of the output pin for which it was declared.

0007cc invalid download format(s)

Warning. At least one of the download formats specified is not available for the target device. For example, the HL download format is not available for PALs or PROMs.

0008cc pin can not be used as input: "variable"

Error. The pin to which the variable is assigned provides no input or feedback capability.

0009cc header name undefined, using no_name

Error. The NAME field in the header information is missing. Since CUPLC uses this name to generate download files, the desired file will be created as "no_name" along with the appropriate extension.

0010cc disk write error: "filename"

Fatal. CUPLC encountered an I/O error trying to write the indicated file. This error usually occurs when there is insufficient disk space.

0011cc out of memory: "conditions"

Fatal. CUPLC has used all the available RAM memory allocated by the operating system.

Note

This error does not result from insufficient product terms in the device to implement a particular expression.

0012cc disk read error, unexpected end of file: "filename"

Fatal. CUPLC encountered an I/O error trying to read the indicated file. This error usually occurs when the file is being read from damaged media.

0013cc conflicting usage of pinnode:'variable"

Error. Variable usage is mutually excluded by a previous usage of the pin or pinnode. A shared product term or terms has been defined more than once.

10xxcc program error: "specifics"

Fatal. An operating system interface problem is suspected. Contact Logical Devices customer support.

❑ CSIM ERROR MESSAGES

This section describes the error messages for the CSIM and CSIMA modules.

❑ CSIM Module Error Messages

0001sk could not open: "filename"

Fatal. CSIM cannot continue because of the failure to open the indicated file. Be sure the file exists if it is an input.

0002sk could not execute program: "program name"

Fatal. CSIM is unable to perform the next step in the simulation. Be sure that all of the CSIM program files exist on the same directory or disk.

0003sk could not find PATH in ENVIRONMENT

Fatal. The PATH assignment has not been made in the ENVIRONMENT.

0004sk could not find LIBCUPL in ENVIRONMENT

Fatal. The LIBCUPL assignment has not been made in the ENVIRONMENT.

0005sk could not find program: "program name"

Fatal. CSIM is unable to locate the CSIM program using the PATH in the ENVIRONMENT.

0006sk insufficient memory to execute program: "filename"

Fatal. Not enough program storage available to load and execute the program. Refer to the System Overview for

the minimum memory requirements for the configuration being used.

0007sk invalid flag: "flag"

Fatal. The specified flag is not a valid option flag. Execute CSIM without arguments to get a listing of valid option flags.

008sk out of memory: "condition"

Fatal. CSIM has used all the available RAM memory allocated by the operating system. Check for the existence of print spoolers, RAM disks, or other memory-resident programs which may decrease the amount of memory available to the CUPL application program.

10xxsk program error: "specifies"

Fatal. An operating system interface problem is suspected. Contact LDI customer support.

❑ CSIMA Module Error Messages

0001sa could not open: "filename"

Fatal. CSIM cannot continue because of the failure to open the indicated file. Be sure the file exists if it is an input.

0002sa invalid number: "number"

Error. Either the number is used improperly, or a previous syntax error has caused the number to be used improperly.

0003sa invalid file format: "filename"

Warning. The file was not created by a compatible version of CUPL.

0004sa invalid library access key

Fatal. The version of CSIMA is not compatible with the version of the device library used in the simulation. This occurs when either CSIMA or the device library, but not both, has been updated.

0005sa invalid library interface

Fatal. Either the device library was not created using the CUPL library manager, CBLD, or CSIMA and the device library are not compatible.

0006sa bad library file: "library"

Fatal. Either the device library does not exist or the contents of the device library have been damaged.

0007sa device not in library: "device"

Fatal. Either the specified target device does not exist or an entry has not been made in the device library for the device.

0008sa invalid output format: "format"

Warning. The download format is not available for the target device; for example, the JEDEC download format is not available for PROMS.

0009sa invalid syntax: "symbol"

Error. Either the symbol is used improperly, or a previous syntax error has caused the symbol to be used improperly.

0010sa expecting device: "device"

Fatal. The target device is not the same as used when CUPL created the absolute file.

0011sa unknown symbol: "symbol"

Error. The symbol, used in the order statement, was not previously defined in the CSIM or CUPL source files.

0012sa too many symbols:

Fatal. The number of symbols in the order statement exceeds the number of symbols previously defined in the CSIM and CUPL source files.

0013sa excessive test value "value"

Error. The test vector value is greater than the maximum possible value defined in the order statement. This error will occur when there are too many test values.

0014sa insufficient test values

Fatal. The test vector value is less than the minimum possible value defined in the order statement. This error will occur when there are too few test values.

0015sa field already defined: "field"

Error. The field name was previously used in either the CSIM or CUPL source files.

0016sa too many errors

Fatal. CSIM has encountered too many errors to continue.

0017sa missing symbol "symbol"

Error. CSIM expected a keyword.

0018sa out of memory: "condition"

Fatal. CSIM has used all the available RAM memory allocated by the operating system.

0019sa user expected (value) for: "variable"

Error. The test value expected by the user in the .SI file did not match the actual value computed by CSIM.

0020sa unstable output: "variable"

Error. The output variable did not have the same test value for two continuous evaluation passes after the maximum twenty passes were attempted. Check the logic equation for an untestable design.

0021sa invalid test value: "value"

Error. Either the test value is an invalid test vector symbol or the test value is used improperly; that is, a test value of 0 is used for an output.

0022sa bad fault id: "jedec number"

Error. The JEDEC number, given as the fault ID, is not the address of the beginning of a product term.

0023sa could not read file: "filename"

Fatal. CSIM could not read from the specified file. This occurs when the contents of the file have been corrupted.

0024sa could not write file: "filename"

Fatal. CSIM could not write to the specified file. This occurs when the file is write protected or there is no room left on the disk.

0025sa inconsistent header information

Warning. The header information in the CSIM source file does not match the header information in the CUPL source file used to create the absolute file.

0026sa missing header item(s)

Warning. At least one of the header statements is missing.

0027sa old absolute file format for "filename"

Fatal. The absolute file was created by an incompatible version of CUPL.

0028sa statement too long

Fatal. The statement exceeds 256 characters.

0029sa invalid trace level: "number"

Error. The trace level must be a decimal number in the range of 0 through 4.

0030sa invalid character: "hex value"

Error. CSIMA has encountered an illegal ASCII value in the source file. Make sure the file was created in nondocument mode on the word processor. This error can also be caused by files which were created over a serial modem upload/download link.

0031sa disk read error, unexpected end of file: "filename"

Fatal. CSIMA encountered an I/O error trying to read the indicated file. This error usually occurs when the file is being read from damaged media.

10xxsa program error: "specifies"

Fatal. An operating system interface problem is suspected. Contact Logical Devices customer support.

❑ CBLD ERROR MESSAGES

This section describes the error messages for the single CBLD program module.

❑ CBLD Module Error Messages

0001ba invalid syntax: "statement line number"

Fatal. The specified statement contains a syntax error.

0002ba could not open: "filename"

Fatal. CBLD cannot continue because of the failure to open the indicated file. Be sure the file exists if it is an input.

0003ba invalid flag: "flag"

Warning. CBLD will warn that the invalid flag cannot be evaluated, and then continue operation. Invoke CBLD without arguments for a list of valid flags.

0004ba invalid library access key

Warning. The version number of CBLD does not match the version number of the device library. This occurs when either CBLD or the device library, but not both, has been updated. To prevent incompatibility problems, it is important that the version numbers match.

0005ba invalid library interface

Fatal. Either the device library was not created using CBLD, or the library and CBLD have different version numbers causing incompatibility problems.

0006ba bad library file: "library"

Fatal. Either the device library does not exist or the contents have been damaged.

0007ba device not in library: "device"

Warning. Either the specified target device does not exist or an entry has not been made in the device library for the device.

0008ba inconsistent library access key

Warning. During a build operation, the target library and a source library have different access keys.

0009ba inconsistent library rev

Warning. During a build operation, the target library and a source library have different revision numbers.

0010ba too many devices for target library: "library"

Fatal. The device library directory is full.

0011ba device not in library: "device"

Error. Either the specified device does not exist or an entry has not been made in the device library.

0012ba could not find LIBCUPL in ENVIRONMENT

Fatal. The LIBCUPL assignment has not been made in the ENVIRONMENT.

0013ba too many devices on command line

Error. The user has specified too many devices on the command for CBLD to proceed. The maximum allowable is determined by the command line limit of the operating system.

0014ba disk write error: "filename"

Fatal. CBLD encountered an I/O error in trying to write the indicated file. This error usually occurs when there is insufficient disk space.

0015ba out of memory: "condition"

Fatal. CBLD has used all the available RAM memory allocated by the operating system. Check for the existence of print spoolers, RAM disks, or other memory-resident programs which may decrease the amount of memory available to the CUPL application progam.

10xxba program error: "specifies"

Fatal. An operating system interface problem is suspected. Contact Logical Devices customer support.

❑ PTOC ERROR MESSAGES

This section describes the error messages for the single **PTOC** program module.

❑ PTOC Module Error Messages

0001pt invalid syntax: "statement"

Error. The specified statement contains a PALASM syntax error.

0002pt could not open: "filename"

Fatal. PTOC cannot continue because of the failure to open the indicated file. Be sure the file exists if it is an input.

0003pt string too large: "string"

Error. The character string exceeds the maximum allowed number of characters (256).

0004pt unrecognized symbol: "symbol"

Error. The symbol is not a valid PALASM symbol.

0005pt unknown variable: "variable"

Error. The variable used in the function table has not been defined in the pin list.

0006pt unknown part: "part"

Fatal. The part is either not a valid PALASM part number or the part number does not start at line 1, column 1.

0007pt missing dashed line

Error. The dashed line designating the beginning or the end of the PALASM function table is missing.

0008pt disk write error: "filename"

Fatal. PTOC encountered an I/O error in trying to write the indicated file. This error usually occurs when there is insufficient disk space.

0009pt out of memory: "condition"

Fatal. PTOC has used all the available RAM memory allocated by the operating system. Check for the existence of print spoolers, RAM disks, or other memory-resident programs which may decrease the amount of memory available to the CUPL application program.

10xxpt program error: "specifies"

Fatal. An operating system interface problem is suspected. Contact LDI customer support.

❑ WCSIM Error Messages

0001sw could not open: "filename"

Fatal. WCSIM cannot continue because of the failure to open the indicated file. Be sure the file exists if it is an input.

0002sw out of memory: "condition"

Fatal. WCSIM has used all the available RAM memory allocated by the operating system. Check for the existence of print spoolers, RAM disks, or other memory-resident programs which may decrease the amount of memory available to the CUPL application program.

0003sw too many symbols

Fatal. The number of symbols in the order statement exceeds the number of symbols previously defined in the CSIM source file.

0004sw index variable not matched

Fatal. A variable list was defined using incorrect list notation. Refer to List Notation in the Using CUPL chapter.

0005sw too many test vectors

Fatal. The number of test vectors exceeded 1024.

Device Usage Notes B

This appendix lists the programmable logic devices (PLDs) that require special attention when being used with **CUPL** and **CSIM.**

Device Mnemonic: P20X4, P20X8, P20X10

1. When writing logic equations for devices containing an XOR gate, the $ operator may not be included inside any parentheses that change the evaluation order of the expression.

2. When applying DeMorgan's Theorem to an equation involving the XOR gate, the expression written first is the one negated.

Device Mnemonic: F155, F157, F159, F179

1. Registers may be used as either D-type or JK-type, but not both (no dynamic conversion).

2. The output enable buffer for all registers is always controlled by pin 11 (pin 13 for F179).

3. The load control term (for loading registers from the output pins) is not supported.

4. The product term that drives the register control buffer is fixed and may not be accessed to drive the complement array.

Device Mnemonic: EP300, P18CV8

1. The registered, internal combinatorial and I/O feedback paths can be selected by using the **.DFB**, **.INT** and **.IO** extensions respectively. If the feedback type is the same as the output (registered feedback for registered output), a feedback extension "**.IO**" is required.

Device Mnemonic: EP600, EP900

1. Support for registered mode of the macrocell consists only of D-type and T-type flip-flops. J-K and S-R flip-flops are not supported because they don't physically exist in the device. They must be emulated with exclusive-or equations.

2. The D-type registered, T-type registered, and I/O feedback paths can be selected by using the **.DFB**, **.TFB** and **.IO** extensions respectively. If the feedback type is the same as the output (D-type registered feedback for D-type registered output), then a feedback extension is not required.

Device Mnemonic: F501, F502

1. Both f501 and f502 contain NAND array architecture. The NAND gates must be defined as complement array nodes.

2. Although there is only one product term for each output pin in f501 and f502, multiple

product term output can be implemented using DeMorgan algorithm and NAND node.

For example, to implement the following logical function in f502:

```
X = (a & b)
  # (c & d)
  # (e & f);
```

Define as follows:

```
Pin 19 = X;

Pinnode 81 = !X1;
Pinnode 82 = !X2;
Pinnode 83 = !X3;

X = !(!X1 & !X2 & !X3);
X1.ca = a & b;
X2.ca = c & d;
X3.ca = e & f;
```

Device Mnemonic: F506

1. The registered clock polarity is set by writing one **.CKMUX** expression for all registers used in the design. By default, the clock is treated as positive-edge triggered.

Device Mnemonic: F507

1. The register and counter clock polarity are set by writing one **.CKMUX** expression for all

registers and counter inputs used in the design. By default, the clock is treated as positive-edge triggered.

2. The built-in counter input nodes are defined using the **NODE** or **PINNODE** statements, and the **.CNT** extension. The counter clear and hold controls are defined using the **NODE** or **PINNODE** statements and writing either combinatorial or S-R registered expressions.

3. **CSIM** generates the counter inputs automatically, based on the counter control logic.

Device Mnemonic: F16V8, F18V8Z, F20V8

1. These devices emulate two different PAL architectures with their flexible output macro configuration. If the F16V8, F18V8Z or F20V8 device mnemonic is used, the device parameters for the proper sub-mode are automatically selected according to the following:

A. **Registered Mode**
Specifying any output pin as registered invokes the registered mode (D). Specifying any output enable term for a nonregistered pin invokes the registered mode (D).

Mnemonic:	**F16V8D**
	F18V8ZD
	F20V8D

B. **Small Mode**
If neither of the above conditions are met, the

device type defaults to the small mode (S).

Mnemonic: **F16V8S**
F18V8S
F20V8S

2. Either the automatic selection mechanism or the device mnemonic for the specific sub-mode may be used.

Device Mnemonic: G16V8, G20V8

1. These devices emulate three different PAL architectures with their flexible output macro configuration. If the G16V8 or G20V8 device mnemonic is used, the device parameters for the proper sub-mode are automatically selected according to the following:

A. **Medium Synchronous Mode**
This mode is automatically chosen when the PLD source file has registered output. In the medium synchronous mode, specifying an output enable term for a registered output pin is not flagged as an error by the compiler or simulator. In this mode, the output enable control for registered pins is common to pin 11 (GAL16V8) or pin 13 (GAL20V8).

Mnemonic: **G16V8MS**

Input only	Output only	Input/Output
2, 3, 4,		12, 13, 14,
5, 6, 7,		15, 16, 17,
8, 9		18, 19

Pin 1 = common clock
Pin 11 = common output enable

Mnemonic: **G20V8MS**

Input only	Output only	Input/Output
2, 3, 4,		15, 16, 17,
5, 6, 7,		18, 19, 20,
8, 9, 10,		21, 22
11, 14, 23		

Pin 1 = common clock
Pin 13 = common output enable

B. **Medium Asynchronous Mode**
This mode is automatically chosen when the PLD source file has an output enable term for a non-registered pin and/or.

Mnemonic: **G16V8MA**

Input only	Output only	Input/Output
1, 2, 3,	12, 19	13, 14, 15,
4, 5, 6,		16, 17, 18
7, 8, 9,		
11		

Mnemonic: **G20V8MA**

Input only	Output only	Input/Output
1, 2, 3,	15, 22	16, 17, 18,
4, 5, 6,		19, 20, 21
7, 8, 9,		
10, 11, 13,		
14, 23		

C. **Small Mode (Default)**
If none of the above are met, the device type defaults to the small mode. In this mode, the Input/Output pins are configured as either

Input Only or Output only (that is, no feedback can occur) .

Mnemonic:		**G16V8S**
Input only	Output only	Input/Output
1, 2, 3,	15, 16	12, 13, 14,
4, 5, 6,		17, 18, 19
7, 8, 9,		
11		

Mnemonic:		**G20V8S**
Input only	Output only	Input/Output
1, 2, 3,	18, 19	15, 16, 17,
4, 5, 6,		20, 21, 22
7, 8, 9,		
10, 11, 13,		
14, 23		

2. Either the automatic selection mechanism or the device mnemonic for the specific sub-mode desired can be used.

Device Mnemonic: G6001

1. The output logic macrocell feedback paths can be selected as internal only, I/O only, or both, via the .INT and .IO extensions. If the feedback type is the same as the output (internal feedback for registered output), then a feedback extension is not required.

2. The output logic macrocells can be treated as buried register or combinatorial nodes, allowing the pins to be treated as inputs. The buried nodes must be defined in NODE or

PINNODE statements and the input pins defined in PIN statements.

3. All input pins can be configured as registered or latched inputs via the .DQ and .LQ extensions, respectively. Each input pin in a group must be treated in the same manner.

Device Mnemonic: RA10P8, RA11P8, RA12P8

1. Active-HI chip enables are simulated as Active-LO.

Device Mnemonic: P10P8V, P12P6V, P14P4V, P16P2V

1. The output macrocell for these devices is always configured for the OR path, utilizing double the product terms of a standard PAL. The default bypass path (lower power) and XOR path are not supported.

Device Mnemonic: P16P8V, P16RP4V, P16RP6V, P16RP8V

1. The output macrocell for these devices is always configured for the default bypass path. The OR path (double product terms at the expense of an output pin) and XOR path are not supported.

Device Mnemonic: P22CV10, P22VP10, P23S8

1. The registered and I/O feedback paths can be selected by using the **.DFB** and **.IO**

extensions respectively. If the feedback type is the same as the output (registered feedback for registered output), then a feedback extension is not required.

Device Mnemonic: P29M16

1. The output feedback paths can be selected as registered only, latched only, I/O only, registered and I/O or latched and I/O via the **.DFB**, **.LFB** and **.IO** extensions. If the feedback type is the same as the output (registered feedback or registered feedback for registered output), then a feedback extension is not required.

2. The dual feedback outputs can be treated as buried register or latched nodes, allowing the pins to be treated as inputs. The buried nodes must be defined in **NODE** or **PINNODE** statements and the input pins defined in **PIN** statements.

3. The dual feedback outputs can be treated as registered or latched inputs, via the **.DQ** and **.LQ** extensions.

4. Individual clock control is set by writing a **.CKMUX** expression. By default, the clock control is set to clock/latch enable pin1, positive edge triggered.

5. Individual output enable control is set by writing either a **.OEMUX** expression for common control, or a **.OE** expression for banked product term control. By default, the output enable control is set to common output enable pin 11.

Device Mnemonic: P29MA16

1. The output feedback paths can be selected as registered only, latched only, I/O only, registered and I/O or latched and I/O via the **.DFB**, **.LFB** and **.IO** extensions. If the feedback type is the same as the output (registered feedback for registered output), then a feedback extension is not required.

2. The dual feedback outputs can be treated as buried register or latched nodes, allowing the pins to be treated as inputs. The buried nodes must be defined in **NODE** or **PINNODE** statements and the input pins defined in **PIN** statements.

3. The dual feedback outputs can be treated as registered or latched inputs via the **.DQ** and **.LQ** extensions.

4. Individual clock or latch enable control is set by writing either a **.CKMUX** expression for synchronous control or a **.CK** expression for asynchronous control. By default, the clock control is set to clock/latch enable pin 1, positive edge triggered.

5. Individual output enable control is set by writing either a **.OEMUX** expression or common control, or by a **.OE** expression for banked product term control. By default, the output enable control is set to common output enable pin 11.

Device Mnemonic: P32VX10

1. Combinatorial outputs have active low polarity, whereas registered outputs have programmable polarity.

2. The output feedback paths can be selected as registered only, I/O only, or both, via the **.DFB** and **.IO** extensions. If the feedback type is the same as the output (registered feedback for registered output), then a feedback extension is not required.

3. The outputs can be treated as buried register nodes allowing the pins to be treated as inputs. The buried nodes must be defined in **NODE** or **PINNODE** statements and the input pins defined in **PIN** statements.

Device Mnemonic: PLX448

1. The output feedback paths can be selected as internal only, I/O only, or both, via the **.INT** and **.IO** extensions. If the feedback type is the same as the output (internal feedback for registered output), then a feedback extension is not required.

2. The outputs can be treated as a buried register or combinatorial nodes, allowing the pins to be treated as inputs. The buried nodes must be defined in **NODE** or **PINNODE** statements and the input pins defined in **PIN** statements.

3. The 48mA output pins can be programmed to behave as open collector outputs, by writing **.TEC** expressions.

Device Mnemonic: V750

1. The output feedback paths can be selected as registered only, I/O only, or both, via the **.DFB** and **.IO** extensions. If the feedback type is the same as the output (registered feedback for registered output), then a feedback extension is not required.

2. The outputs can be treated as buried register nodes allowing the pins to be treated as inputs. The buried nodes must be defined in the **NODE** or **PINNODE** statements and the input pins defined in **PIN** statements.

3. When an output requires more product terms than its sum term provides, CUPL uses the sum term pairing capability to increase the number of available product terms. This will cause an incorrect fuse map to be generated if the associated buried register is also being used.

Device Mnemonic V2500

1. The output feedback paths can be selected as registered with I/O or as combinatorial. When a registered output is specified, the **.IO** extension is required for the I/O feedback.

2. The outputs can be treated as buried register nodes allowing the pins to be treated as inputs. The buried nodes must be defined in

the **NODE** or **PINNODE** statements and the input pins defined in **PIN** statements.

3. When an output requires more product terms than its sum term provides, CUPL uses the sum term pairing capability to increase the number of available product terms. This will cause an incorrect fuse map to be generated if the associated buried register is also being used.

Device Mnemonic: PLD9000

This is an imaginary device designed to represent a generalized PLA. It is intended to be used for designs which are not targeted for any particular device.

The device contains 100 pins, 45 inputs, 50 outputs, and 5 clocks. Each output has individual output enable control and can be configured in one of four modes.: combinatorial/active low, combinatorial/active high, D-registered/active low, or D-registered/active high. There are no power and ground pins. The pin organization is as follows:

1. Pin organization:

Pin 1 - A common register clock input for output pins 51 – 100.

Pin 2 - A secondary common register clock input for output pins 61 – 70.

Pin 3 - A secondary common register clock input for output pins 71 – 80.

Pin 4 - A secondary common register clock input for output pins 81 – 90.

Pin 5 - A secondary common register clock input for output pins 91 – 100.

Pins 6 – 50 - Input pins.

Pins 51 – 100 Input-Output pins.

2. Banked clock control is set by writing **.CKMUX** expressions. By default the clock control is set to clock pin 1.

3. There are 200 product terms organized in a PLA format, so each is available to any output.

Download Formats C

This appendix describes downloadable file formats and the **.DOC** file.

❑ DOWNLOADABLE FILE FORMATS

This section describes the JEDEC, ASCII-hex, and HL standards for data transfer to a device programmer.

❑ JEDEC Format

The JEDEC JC-42.1 standard consists of a transmission that begins with an ASCII Start-of-Text (STX) character, followed by various fields of information: an ASCII End-of-Text (ETX) character, and a transmission checksum. The allowed legal characters consist of printable ASCII characters (hex 20 through 7E) and the four control characters listed in table C-1.

Table C-1. Control Characters

STX	Start-of-Text	hex 02
ETX	End-of-Text	hex 03
LF	Line Feed	hex 0A
CR	Carriage Return	hex 0D

Figure C-1 shows a sample JEDEC file created by using **CUPL** and **CSIM.**

```
<STX>
Cupl                3.0  Serial # 0-00000-000
Device              p16r4  Library DLIB-h-24-11
```

```
Created             Tue Jul 07 15:22:33 1987
Name                WAITGEN
Partno              P9000183
Revision            02
Date                03/14/85
Designer            Osann
Company             P-CAD
Assembly            PC Memory
Location            U106
*QP20
*QF2048
*G0
*F0
*L00000 10110101110111111100111000110111
*C0307
*QV
*P 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
*V0001 CXXXXX110N0HHLLZXXHN
*<<ETX>6AA1
```

Figure C-1. Sample JEDEC File

The rest of this section describes the fields in the sample file in Figure C-1.

The design specification is the first field in the format. It includes all information between the STX and the first asterisk (*). This information is for documentation purposes only, and consists of the header information from the CUPL source file along with the version number of the compiler and device library.

Each of the fields after the design specification field begins with one of the single character identifiers shown in Table C-2.

Table C-2. Field Identifiers

A - *	N - *
B - *	O - *
C - Fuse check sum	P - Pin Order
D - Device type	Q - Value
E - *	R - *
F - Default fuse state	S - *
G - Security fuse	T - *
H - *	U - *
I - *	V - Test vector
J - *	W - *
K - *	X - *
L - Fuse link data	Y - *
M - *	Z - *

* - indicates reserved for future use

Characters that have not been defined are reserved for future use.

Fields can be identified by multiple characters; for example, **QF** to indicate a value default fuse state.

The device field (**D**) is no longer supported. In CUPL 1.0 the device field contained a four-digit code that identified the device to be programmed. However, lack of consistent support by the device programmer manufacturers led to this field being dropped from the JEDEC file format for CUPL 2.0 and higher.

A value field, **QP**, describes the number of pins for the device; another value field, **QF**, describes the total number of programmable fuses in the device. Both values are decimal numbers.

The security fuse field (G) instructs the programmer to disable (G0) or enable (G1) the programming of the security fuse on the devices that contain this option. A single space follows the number for compatibility with certain manufacturers' equipment.

The default fuse state field (F) defines the state of the fuses that are not explicitly defined in the L field. Since CUPL 3.0 does not transmit all fuse states (to speed data transmission on large designs), this field must be recognized by the device programmer.

The fuse link field (L) contains the actual data. Each device fuse link is assigned a decimal number, starting with 0000. Each numbered fuse has two possible states: binary 0 specifies a low resistance link (**FUSE INTACT**) and binary 1 specifies a high resistance link (**FUSE BLOWN**).

Note

Some manufacturers specify "test fuses" for purposes of running AC parameter tests on the device before programming. These fuses are not part of the fuse link data.

The **L** identifier begins the field and is followed by the number of the first fuse being defined in the field. When more than one binary value is specified, the additional values are assigned to fuses numbered consecutively from the first fuse number.

The next field is a fuse checksum (C) field. The checksum is a 16-bit hexadecimal value which is computed by adding 8-bit words formed from the specified state of each fuse link in the

device. Link number 0 is the least-significant bit (lsb) and link number 7 is the most significant bit (msb) of word 0. Unspecified bits in the final 8-bit word are set to zero before computing the checksum. In the Figure C-1 the first thirty-two fuses generate four 8-bit words as follows:

	msb							lsb		
word 00	1	0	1	0	1	1	0	1	-->	AD
word 01	1	1	1	1	1	0	1	1	-->	FB
word 02	0	1	1	1	0	0	1	1	-->	73
word 03	1	1	1	0	1	1	0	0	-->	EC

										0307

The optional test vector field (V) is created by running **CSIM** with the **-j** option flag. It contains functional test information for each device pin. The **QV** value field defines the number of test vectors which the file contains. Test vectors are numbered starting with 0001 and applied in numerical order to the device being tested. Table C-3 lists the valid conditions for any pin.

Table C-3. Test Conditions

0	-	Drive input LO (0 volts)
1	-	Drive input HI (+5 volts)
C	-	Drive input LO, HI, LO
K	-	Drive input HI, LO, HI
L	-	Test output LO (0 volts)
H	-	Test output HI (+5 volts)
Z	-	Test output for high impedance
X	-	Input undefined, Output untested
N	-	Power pins and Outputs not tested
P	-	Preload registers
		Value given applied to $\overline{Q}$ of register

The test conditions, as they appear in the vector, are applied to the device pins according to the sequence given in the pin order (P) field. In this example (figure C-1), the first condition is applied to pin 1 and the last to pin 20 of a 20 pin device. The **C** and **K** driving signals are presented after all other inputs are stable. The **L, H,** and **Z** conditions are tested after all inputs have been stabilized, including **C** and **K.**

The **P** driving signal on the clock pin is valid only for those devices capable of preloading registers with a supervoltage. Devices which use dedicated TTL-level preload pins must use the **C** or **K** driving signals on these pins to preload the registers.

The end of transmission is signified with a non-printing ASCII ETX character followed immediately by a transmission checksum (sum-check) of four ASCII hex characters. The checksum is the 16-bit sum of the ASCII values of all the transmitted characters between, and including, the starting STX and ending ETX characters. In the sample file (figure C-1), the transmission checksum calculates to 46C9, when taking into account a non-printing carriage return and line feed at the end of every line.

❑ ASCII-Hex Format

The ASCII-hex format is generated for PROMs only. Data in this format is organized in sequential bytes separated by the execute character (space). Characters immediately preceding the execute character are interpreted as data bytes. The format may express the data bytes as either a single hex digit (x4 PROMs) or two hex digits (x8 PROM's)

An ASCII STX [Ctrl]-[B] character starts the transmission. A four-digit hexadecimal address, preceded by a $, A , and comma (**$A,**) starts each line of 16 data bytes. An ASCII ETX [Ctrl]-[C] ends the data portion of the transmission. It is followed by forty spaces. Figure C-2 shows a sample hex file.

```
^B
$A0000,00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
$A0010,10 11 12 13 14 15 16 17 18 19 1A 1B 1C 1D 1E 1F
$A0020,20 21 22 23 24 25 26 27 28 29 2A 2B 2C 2D 2E 2F
$A0030,30 31 32 33 34 35 36 37 38 39 3A 3B 3C 3D 3E 3F
^C
$S07E0,
```

Figure C-2. Sample Hex File

❑ HL Formats

The HL Download Format is generated for Signetics IFL devices only. Each device has its own unique format. All formats begin with STX [Ctrl]-[B] and end with ETX [Ctrl]-[C]. The following sections describe the format for each type of IFL device.

82S100/101 FPLA

The active-level identifier ***A** is followed by the states of the active levels for F7 to F0, where H denotes active-HI and L denotes active-LO. The product terms are described by the ***P** identifier followed by a space and the P-term number. The input variable identifier ***I** is followed by the input variables I15 to I0 and then the output function identifier ***F** followed by F7 to F0. Figure C-3 shows a sample file in this format.

```
^B
*A LHLHLHLH
*P 00 *I HHHHLLLLHHHHLLLL *F A.A.A.A.
 .
 .
```

```
*P 47 *I LLLLLLLLLLLLLLL *F ....AAAA
^C
```

Figure C-3. Sample 82S100/101 FPLA File

82S103 FPGA

The product terms start with the ***G** identifier followed by a space and the term number. This is followed by the active-level identifier ***A** and the active-level data. The input variable identifier ***I** is followed by the input variables I15 to I0. Figure C-4 shows a sample file in this format.

```
^B
*G 00 *A L *I HHHHLLLLHHHHLLLL
 .
 .
*G 47 *A H *I LLLLLLLLLLLLLLL
^C
```

Figure C-4. Sample 82S103 FPGA File

82S105 FPLS

The preset / output enable option is entered using the ***A** identifier followed by H (preset) or L (output enable). The transition terms are described by the term identifier ***T** followed by the term number. The complement array identifier ***C** is followed by the value of this term. The input variables for the term are given with the input variable identifier ***I** followed by I15 to I0. The present state of the flip-flops is given with ***P** followed by P5 to P0. The next state values follow with ***N** and N5 to N0. The output function is described as ***F** followed by F7 to F0. Figure C-5 shows a sample file in this format.

```
^B
*AL
```

```
*T 00 *C.*ILLLLHHHHLLLLHHH*PHHHLLL
          *NHHHLLL *F HHHHHHHH
 .
 .
*T 47 *C A *I LL------- *P LLLLLL
          *N HHHHHH *F HHHLLL-
^C
```

Figure C-5. Sample 82S105 FPLS File

82S151 FPGA

The direction of the I/O pins follows the ***DIR** identifier, and the output polarity follows the ***POL** identifier. The product terms are described by the ***P** identifier followed by a space and the P-term number.

Control term numbers start with a ***D** and then the term number. The input variable identifier ***I** is followed by the input variables I5 to I0 and then the I/O feedback identifier ***B** followed by B11 to B0. Figure C-6 shows a sample file in this format.

```
^B
*DIR HHLLHHLLHLHL *POL HHHHLLLLHHLL
*P 00 *I HHLLLL           *B HL--HL--LLHH
 .
 .
*P 11 *I LLLLLL *B HHHHLLLLHHLL
*D 02 *I ----HH    *B ----LLLLLL-
 .
 .
*D 00 *I LLLL-- *B HHHHHHLLLL-
^C
```

Figure C-6. Sample 82S151 FPGA File

82S153 FPLA

The output polarity identifier ***POL** is followed by the states of the active levels for outputs B9 to B0, where H denotes active-HI and L denotes active-LO. The product terms are described by the ***P** identifier followed by a space and the P-term number. Control term numbers start with a D and then the term number. The input variable identifier ***I** is followed by the input variables I7 to I0. The feedback variables B9 to B0 follow the ***BI** identifier and the output functions B9 to B0 follow the ***BO** identifier. Figure C-7 shows a sample file in this format.

```
^B
*POL HHLLHHLLHH
*P 00 *I --HH--LL *BI --HL------ *BO A..A..A..A
 .
 .
*P 31 *I ------HH *BI HLHLHLHLHL *BO ....AA....
*P D9 *I --HHHHHH *BI ----HHHHLL
 .
 .
*P D0 *I LLLLLLLL *BI ---------
^C
```

Figure C-7. Sample 82S153 FPLA File

82S155 FPLS

The output enable modes for groups A and B follow the ***E** identifier. The flip-flop mode for each register follows an ***F/F** identifier. The polarity for the output pins follows an ***POL** identifier. The transition terms are described by the term identifier ***T** followed by the term number. The complement array identifier ***C** is followed by the value of this term. The input variables for the term are given with the input variable identifier ***I** followed by I3 to I0. The I/O feedback data follows the ***B** identifier. The present state of the flip-flops is given with ***QP** followed by Q3 to Q0. The next state values follow with ***ON** and O3 to O0. The preset terms for

groups PB and PA follow the preset identifier ***P**. The reset terms for groups RB and RA follow the reset identifier ***R**. The output function is described as ***BO** followed by B7 to B0. The terms for flip-flop control, reset, preset, load, and output enable follow. Figure C-8 shows a sample file in this format.

```
^B
*E AA *F/F A.A. *POL HLHLLHLH
*T 00 *C . *I HHLL *BI HL--HLHL *QP LH-
          *QN LLHH *P .. *R .. *BO .A.A.A.A
 .
 .
*T 31 *C A *I LLHH *BI ----HLHL *QP HHLL
          *QNHHHH *P .A *R .A *BO ..A.AA.A
*T FC *C . *I LLLL *BI LLLLHHHH *QP LLHH
*T LB *C . *I HLLL *BI --LL--LL *QP HHHH
*T LA *C . *I LL-- *BI LLLLHHHH *QP --LL
*T D3 *C . *I LLLL *BI LLLLLLL- *QP LLHH
 .
 .
*T D0 *C . *I LLLL *BI LLHHHHLL *QP HLLH
^C
```

Figure C-8. Sample 82S155 FPLS File

82S157 FPLS

The output enable modes for groups A and B follow the ***E** identifier. The flip-flop mode for each register follows a ***F/F** identifier. The polarity for the output pins follows a ***POL** identifier. The transition terms are described by the term identifier ***T** followed by the term number. The complement array identifier ***C** is followed by the value of this term. The input variables for the term are given with the the input variable identifier ***I** followed by I3 to I0. The I/O feedback data follows the ***BI** identifier. The present state of the flip-

flops is given with ***QP** followed by Q5 to Q0. The next state values follow with ***QN** and Q5 to Q0. The preset terms for group PA follows the preset identifier ***P**. The reset terms for group PA follows the reset identifier ***R**. The output function is described as ***BO** followed by B5 to B0. The terms for Flip-Flop control, reset, preset, load and output enable follow. Figure C-9 shows a sample file in this format.

```
^B
*E AA *F/F A.A. *POL HLHLLHLH
*T 00 *C . *I HHLL *BI HL--HL *QP LH--HL
          *QN LLHHHL *P . *R . *BO .A.A.A
 .
 .
*T 31 *C A *I LLHH *BI --HLHL *QP HHLLHH
          *QNHHHHLL *P A *R A *BO ..AA.A
*T FC *C . *I LLLL *BI LLLLHH *QP LLHHHH
*T PB *C . *I ---- *BI ----HH *QP ----LL
*T RB *C . *I HHHL *BI HHLLLL *QP HLLLHH
*T LB *C . *I HLLL *BI --L-LL *QP HHHHLL
*T LA *C . *I LL-- *BI LLLHHH *QP --LLHH
*T D3 *C . *I LLLL *BI LLLLL- *QP LLHH-
 .
 .
*T D0 *C . *I LLLL *BI LLHHLL *QP HLLHLL
^C
```

Figure C-9. Sample 82S157 FPLS File

82S159 FPLS

The output enable modes for groups A and B follow the ***E** identifier. The flip-flop mode for each register follows a ***F/F** identifier. The polarity for the output pins follows a ***POL** identifier. The transition terms are described by the term identifier. ***T** followed by the term number. The complement array identifier ***C** is followed by the value of this term. The

input variables for the term are given with the the input variable identifier ***I** followed by I3 to I0. The I/O feedback data follows the ***BI** identifier. The present state of the flip-flops is given with ***QP** followed by Q7 to Q0. The next state values follow with ***QN** and Q7 to Q0. The output function is described as ***BO** followed by B7 to B0. The terms for Flip-Flop control, reset, preset, load and output enable follow . Figure C-10 shows a sample file in this format.

```
^B
*E AA *F/F A.A.A.A. *POL LHLH
*T 00 *C . *I HHLL *BI HL-- *QP HHLLHH-
           *QN LLHHLLHH *BO .A.A
 .
 .
*T 31 *C A *I LLHH *BI ---- *QP --HHHHLL
           *QN LLLLHHHH *BO ...A
*T FC *C . *I LLLL *BI LLLL *QP LLLLHHHH
*T PB *C . *I LLLL *BI LLLL *QP LLLLHHHH
*T RB *C . *I LLLL *BI LLLL *QP LLLLHHHH
*T LB *C . *I LLLL *BI LLLL *QP LLLLHHHH
*T PA *C . *I LLLL *BI LLLL *QP LLLLHHHH
*T RA *C . *I LLLL *BI LLLL *QP LLLLHHHH
*T LA *C . *I LLLL *BI LLLL *QP LLLLHHHH
*T D3 *C . *I LLLL *BI LLLL *QP LLLLHHHH
 .
 .
*T D0 *C . *I LLLL *BI LLLL *QP LLLLHHHH
^C
```

Figure C-10. Sample 82S159 FPLS File

82S161 FPLA

The active level identifier ***A** is followed by the states of the active levels for F7 to F0, where H denotes active-HI and L denotes active-LO. The product terms are described by the ***P**

identifier followed by a space and the P-term number. The input variable identifier ***I** is then followed by the input variables I11 to I0 and the output function identifier ***F** followed by F7 to F0. Figure C-11 shows a sample file in this format.

```
^B
*A LHLHLHLH
*P 00 *I LLLLHHHHLLLL *F A.A.A.A.
 .
 .
*P 47 *I LLLLLLLLLLL *F ....AAAA
^C
```

Figure C-11. Sample 82S161 FPLA File

82S162 FPGA

The output polarity identifier ***POL** is followed by the states of the active levels for outputs F4 to F0. The product terms start with the ***G** identifier followed by a space and the term number. The input variable identifier ***I** is followed by the input variables I15 to I0. Figure C-12 shows a sample file in this format.

```
^B
*POL HHLL
*G 00 *I HHHHLLLLHHHHLLLL
 .
 .
*G 04 *I LLLLLLLLLLLLLLLL
^C
```

Figure C-12. Sample 82S162 File

82S163 FPGA

The output polarity identifier ***POL** is followed by the states of the active levels for outputs F8 to F0. The product terms start with the ***G** identifier followed by a space and the term number. The input variable identifier ***I** is followed by the input variables I11 to I0. Figure C-13 shows a sample file in this format.

```
^B
*G 00 *I HLLLHHHHLLLL
 .
 .
*G 08 *I LLLLLLLLLLL
^C
```

Figure C-13. Sample 82S163 FPGA File

82S167 FPLS

The preset / output enable option is entered using the ***A** identifier followed by H (preset) or L (output enable). The transition terms are described by the term identifier ***T** followed by the term number. The complement array identifier ***C** is followed by the value of this term. The input variables for the term are given with the input variable identifier ***I** followed by I13 to I0. The present state of the flip-flops is given with ***P** followed by P7 to P0. The next state values follow with ***N** and N7 to N0. The output function is then described as ***F** followed by F3 to F0. Figure C-14 shows a sample file in this format.

```
^B
*AL
*T 00 *C .*ILLHHHHLLLLHHHH*PHHHLLLHH
                    *NHHHLLLLL *F HHHH
 .
 .
*T 47 * A *I LL------------ *P --LLLLL
*N HHLLHHHH *F HL-
```

```
^C
```

Figure C-14. Sample 82S167 FPLS File

82S168 FPLS

The preset/output enable option is entered using the *A identifier followed by H (preset) or L (output enable). The transition terms are described by the term identifier *T followed by the term number. The complement array identifier *C is followed by the value of this term. The input variables for the term are given with the input identifier *I followed by I11 to I0. The present state of the flip-flops is given with *P followed by P9 to P0. The next state values follow with *N and N9 to N0. The output function is described as *F followed by F3 to F0. Figure C-15 shows a sample file in this format.

```
^B
*AL
*T 00 *C.*ILLHHHLLLHHHH*PHHHLLLHHLH
                    *NHHLHHLLLLL *F HHHH
 .
 .
*T 47 *C A *I LL------------ *P --LLLLLL-
*N HHLLHHHH-L *F HL-
^C
```

Figure C-15. Sample 82S168 FPLS File

82S173 FPLA

The output polarity identifier *POL is followed by the states of the active levels for outputs B9 to B0, where H denotes active-HI and L denotes active-LO. The product terms are described by the *P identifier followed by a space and the P-term number. Control term numbers start with D and then the term number. The input variable identifier *I is followed by the input variables I11 to I0. The feedback variables B9 to B0 follow the *B0 identifier. Figure C-16 shows a sample file in this format.

```
^B
*POL HHLLHHLLHH
*P 00 *I --HH--LLHH *BI --HL------ *BO A..A..A..A
  .
  .
*P 31 *I LL------HH *BI HLHLHLHLHL *BO ....AA....
*P D9 *I LL--HHHHHH *BI ----HHHHLL
  .
  .
*P D0 *I --LLLLLLLL *BI ---------
^C
```

Figure C-16. Sample 82S173 FPLA File

82S179 FPLS

The output enable modes for groups A and B follow the ***E** identifier. The flip-flop mode for each register follows a ***F/F** identifier. The polarity for the output pins follows a *POL identifier. The transition terms are described by the term identifier ***T** followed by the term number. The complement array identifier ***C** is followed by the value of this term. The input variables for the term are given with the input variable identifier ***I** followed by I7 to I0. The I/O feedback data follows the ***BI** identifier. The present state of the flip-flops is given with ***QP** followed by Q7 to Q0. The next state values follow with ***QN** and Q7 to Q0. The output function is described as ***BO** followed by B7 to B0. The terms for flip-flop control, reset, preset, load, and output enable follow. Figure C-17 shows a sample file in this format.

```
^B
*E AA *F/F A.A.A.A. *POL LHLH
*T 00 *C . *I HHLLHHLH *BI HL-- *QP HHLLHH-
           *QN LLHHLLHH *BO .A.A
  .
  .
*T 31 *C *1 LLH--LHH *BI LLLL *QP --HHHHLL
           *QN     LLLLHHHH *BO ...A
*T FC *C . *1      LLLHHHHL *BI LLLL *QP LLLLHHHH
```

```
*T PB *C . *I      LLLHLHLL *BI LLLL *QP LLLLHHHH
*T RB *C . *I      LLLLLLHL *BI LLLL *QP LLLLHHHH
*T LB *C . *I      LLLHHHHL *BI LLLL *QP LLLLHHHH
*T PA *C . *I      LLLLHLHL *BI LLLL *QP LLLLHHHH
*T RA *C . *I      LLLL---- *BI LLLL *QP
*T LA *C . *I      LLLL---H *BI LLLL *QP LLLLHHHH
*T D3 *C . *I      LLLLLLLL *BI LLLL *QP LLLLHHHH
  .
  .
*T DO *C . *I HHHHHHHH *BI LLLL *QP LLLLHHHH
^C
```

Figure C-17. Sample 82S179 FPLS File

DOCUMENTATION FILE FORMAT

This section describes the format for the documentation file (***filename*** **.DOC**), including fuse plot information. A documentation file can be generated by specifying the **-x** option flag when running **CUPL**. Specifying the **-f** option generates a fuse plot in the documentation file.

Figure C-18 shows a sample documentation file.

```
                        WAITGEN.DOC

************************************************************
                          Waitgen
************************************************************
CUPL                3.0 Serial# 9-99999-999
Device
Created
Name
Partno
Revision
Date
Designer
Company
Assembly
Location
============================================================
                    Expanded Product Terms
============================================================
wait1.d  =>
   !memr
   # a15
   # a14
   # a13
   # reset

select_rom =>
     !a13 & !a14 & !a15 & memr

wait2.d  =>
   !memr
   # a15
   # a14
   # a13
   # !wait1

memadr  =>
     a15 , a14 , a13 , a12 , a11

ready  =>
     !wait2

ready.oe  =>
     !a13 & !a14 & !a15 & memr
rom_os  =>
     !a13 & !a14 & !a15 & memr

memreq  =>
     memw
   # memr
```

Figure C-18. Sample Documentation File Sheet 1 of 5

```
  ram_cs0  =>
           !a11 & !a12 & a13 & !a14 & !a15 & memw
  #        !a11 & !a12 & a13 & !a14 & !a15 & memr

  ram_cs1  =>
           a11 & !a12 & a13 & !a14 & !a15 & memw
           a11 & !a12 & a13 & !a14 & !a15 & memr

  rom_cs.oe  =>
           1

  ram_cs0.oe  =>
           1

  ram_cs1.oe  =>
           1
================================================================
                        Symbol Table
================================================================
Pin         Variable                                    Pterms    Max
Min
Pol         Name      Ext      Pin      Type     Used    Pterms   Level
---         --------           ---      ---     -----    ------   ------
  ------
            wait1              15        V        -        -        -
            wait1     d        15        X        5        8        1
            all                6         V        -        -        -
            select_rom                   0        I        1        -
-
            wait2              14        V        -        -        -
            wait2     d        14        X        5        8        1
            a12                5         V        -        -        -
            a13                4         V        -        -        -
            a14                3         V        -        -        -
            a15                2         V        -        -        -
!           oe                 11        V        -        -        -
!           memr               8         V        -        -        -
            memadr             0         F        -        -        -
            ready              18        V        1        7        1
            ready     oe       18        X        1        1        1
!           memw               7         V        -        -        -
            cpu_clk            1         V        -        -        -
!           rom_cs             19        V        1        7        1
            reset              9         V        -        -        -
            memreq             0         I        2        -        -
!           ram_cs0            12        V        2        7        1
!           ram_cs1            13        V        2        7        1
            rom_cs    oe       19        D        1        1        0
            rom_cs0   oe       12        D        1        1        0
            rom_cs1   oe       13        D        1        1        0

LEGEND      F:field     D:default variable       W:extended node
                      N:node      I:intermediate variable  T:function
                      V:variable  X:extended variable      U:undefined
```

Figure C-18. Sheet 2 of 5

```
================================================
                    Fuse Plot
================================================
Pin #19
   0000 --------------------------------
   0032 -x---x------------------x------
   0064 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
   0096 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
   0128 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
   0160 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
   0192 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
   0224 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
Pin #18
   0256 -x---x---x--------------x------
   0258 ----------------------x--------
   0320 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
   0352 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
   0384 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
   0416 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
   0448 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
   0480 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
Pin #17
   0512 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
   0544 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
   0576 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
   0608 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
   0640 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
   0672 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
   0704 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
   0738 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
Pin#16
   0788 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
   0800 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
   0832 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
   0864 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
   0898 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
   0928 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
   0980 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
   0992 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
Pin #15
   1024 ----------------------x--------
   1056 x--------------------------------
   1088 ----x----------------------------
   1120 --------x------------------------
   1152 ----------------------------x---
   1184 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
   1216 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
   1248 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
```

Figure C-18. Sheet 3 of 5

```
Pin #14
  1280 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
  1312 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
  1344 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
  1378 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
  1408 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
  1440 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
  1472 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
  1504 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
Pin#13
  1536 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
  1568 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
  1600 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
  1632 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
  1664 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
  1696 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
  1728 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
  1760 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
Pin #12
  1792 --------------------------------
  1824 -x---x--x----x---x---x----------
  1856 -x---x--x----x---x-------x------
  1888 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
  1920 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
  1952 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
  1984 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
  2016 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

LEGEND      X : fuse not blown
            - : fuse blown
```

Figure C-18. Sheet 4 of 5

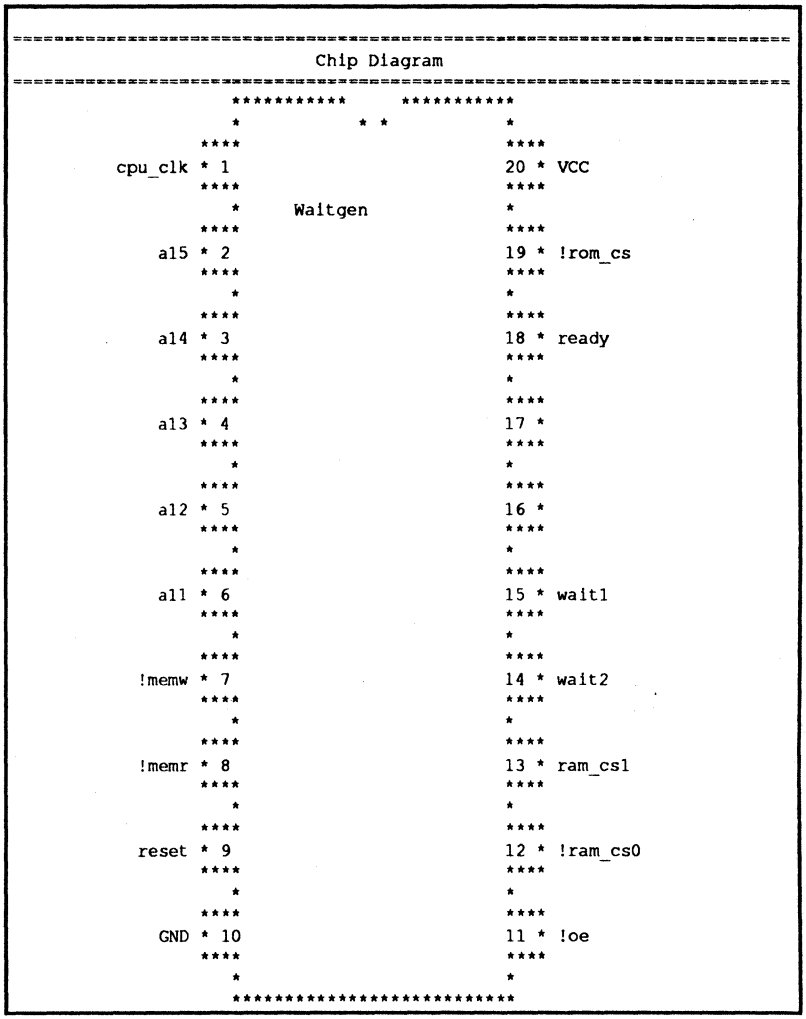


Figure C-18. Sheet 5 of 5

The first part of the file contains archival and revision information that is identical to the header information in the corresponding **CUPL** source file. The first part also contains

version information about the device library and the CUPL program, and the date and time the file was created.

The next section of the file, Expanded Product Terms, contains the product terms generated by CUPL from the equations contained in the logic description file. **WAITGEN.PLD**, contained in the CUPL package, is the source logic description file for the sample documentation file in Figure C-18. Its contents can be viewed in order to compare the original logic equations with the product terms generated by CUPL.

CUPL generates product terms for the devices specified on the command line when running **CUPL** or with the **DEVICE** keyword in the logic description file. For example, some devices, such as PAL16L8s, contain fixed inverting buffers. In certain cases, to fit the logic to the device, CUPL performs DeMorgan'sTheorem. For example, the logic description file is written for a PAL16L8 device; in the pin list, all outputs have been declared as active-HI. The following equation is written to specify an OR function.

```
c = a # b;
```

However, the PAL16L8 contains a fixed inverting buffer. Because the inverting buffer cannot be changed, CUPL fits the logic to the device by performing a DeMorgan on the OR equation, which generates the following product term:

```
c => !a & !b
```

See Pin Declaration Statements in Chapter 2 for further information on how CUPL generates product terms for devices with fixed inverting buffers, when the output pins are asserted in the pin declaration statement.

The next section of the file, Symbol Table, provides information about each variable in the logic description file, including the pin number, extension, type of variable, number of product terms available, number of product terms used, and the minimization level used by CUPL.

If the maximum available product terms for a device is exceeded, CUPL displays an error message during compilation, naming the pin. However, the message doesn't indicate how much the limit was exceeded. The product term availability and use information in the Symbol Table (see Figure C-18) indicates if the number of available product terms was grossly exceeded, or was just slightly over the limit.

The next section, Fuse Plot, is generated by specifying the **-f** option flag when running **CUPL**. This section provides more detailed fuse information than that contained in the JEDEC file. The four-digit beginning JEDEC number for each product term is the number to use to reference **STUCK H** and **STUCK L** (see Fault Simulation in Chapter 4)

The last section, Chip Diagram, provides a diagram of the device showing the location of each variable name.

❑ PDIF FILE FORMAT

This section describes the use of the PDIF (P-CAD Database Interchange Format) file (<F59>filename.PDF) generated by CUPL. For a detailed description of the PDIF-format file, see the PDIF User's Manual. A PDIF-format file can be generated by specifying the **-p** option flag when running **CUPL**.

The PDIF format is used as an interface to the P-CAD schematic capture program PC-CAPS. This is accomplished by translating the CUPL-generated PDIF-format file into a PC-CAPS symbol using the **PDIF-IN** program. The resulting symbol represents the logical representation of the PLD design. This includes pin packaging information, printed circuit board reference designator, PLD type, and design name.

Figure C-19 shows an example of a PC-CAPS symbol generated by **PDIF-IN**.

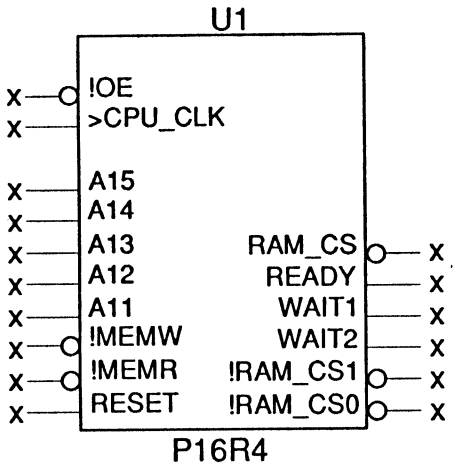


Figure C-19. PC-CAPS Symbol Generated by PDIF-IN

Refer to the *PDIF User's Manual* and the *PC-CAPS User's Manual* for instructions on running the **PDIF-IN** and PC-CAPS programs, respectively.

❑ BERKELEY PLA FILE FORMAT

This section describes the format for the Berkeley PLA file (***filename*.PLA**). The Berkeley PLA format is used as an interface format for PLA logic synthesis tools, such as the Berkeley PLA tools. A Berkeley PLA-format file can be generated by specifying the **-b** option flag when running **CUPL**.

Figure C-20 shows a sample Berkeley PLA-format file.

```
# Berkeley PLA format generated using
#
# CUPL                    3.0 Serial# 9-99999-999
# Device                  pl6rp4 Library DLIB-g-24-15
# Created                 Thu Feb 26 13:45:23 1987
# Name                    Count10
# Partno                  CA0018
# Revision                01
# Date                    07/16/87
# Designer                Kahl
# Company                 Assisted Technology
# Assembly                None
# Location                None
#
# Inputs 1 Q0 Q1 Q2
#        Q3 clr dir
# Outputs Q0.d Q1.d Q2.d Q3.d
#         carry carry.oe
.i 7
.o 6
.p 18
-00010- 100000
-0--00- 100000
-000101 010000
-10-000 010000
-01-000 010000
-11-001 010000
-001001 010000
-000101 001000
-110000 001000
--01000 001000
-1-1001 001000
-01100- 001000
-100101 000100
-000001 000100
-111000 000100
-000100 000100
-1001-- 000010
1------ 000001
.end
```

Figure C-20. Sample Berkeley PLA-Format File

The first part of the file contains archival and revision information. The # character indicates these are comments. This information is identical to the header information in the corresponding **CUPL** source file. There is also version information about the device library and CUPL program, the

date and time the file was created, and a list of the input and outputs to the PLA.

The next section consists of a PLA description generated by CUPL from the equations contained in the logic description file, **COUNT10.PLD** (contained in the CUPL package). Its contents can be viewed to compare the original logic equations with the PLA description generated by CUPL.

The PLA description consists of fields to define the number of inputs '**.i**', outputs '**.o**', product terms '**.p**', and a description of the AND and OR planes of the PLA with one line per product term. Connections in the AND plane are represented with a '1' for connection to the non-inverted input line and a '0' for connection to the inverted input line. No connection to an input line is indicated with a '-'. Connections in the OR plane are indicated by a '1'. No connections are indicated with a '0'. The end of the PLA description is indicated with a'**.end**'.

Node Numbering D

This appendix lists the devices which contain internal nodes supported by CUPL. Each entry contains a node number and a reference. For example, in the case of the AmPAL29M16, the pin number of the I/O macrocell for which the node is contained is given along with the corresponding node number. In the case of the AmPAL23S8, the pin on which the node output would be seen if observability were used is referenced. In the case of Signetics and TI FPLA's, the logical names used in "Signetics Programmable Logic Data Manual 1986" are used for reference.

Table D-1. Node Numbering

Device Manufacturer	Dip Package		LCC Package	
Advanced Micro Devices				
	pin	node	pin	node
AmPAL29M/MA16	3	25	4	29
	4	26	5	30
	9	27	11	31
	10	28	12	32
	15	29	18	33
	16	30	19	34
	21	31	25	35
	22	32	26	36
	obs pin	node		
AmPAL23S8	13	26		
	14	24		
	15	22		
	16	21		
	17	23		
	18	25		
	name	node		
PLS30K12	Q0	29		
	Q1	30		
	Q2	31		
	Q3	32		
	Q4	33		
	Q5	34		
	Q6	35		
	Q7	36		
	HOLD	37		
	LOAD	38		

Table D-1. Node Numbering

Device Manufacturer	Dip Package		LCC Package
	C	39	
	C	40	
	name	node	
PLS30S16	Q1	29	
	Q2	30	
	Q3	31	
	Q4	32	
	C	33	
	C	34	
	name	node	
PMS14R21	A5	25	
	A6	26	
	A0	27	
	A1	28	
	A2	29	
	A3	30	
	A4	31	
	CS0	32	
	CS1	33	
	CS2	34	
	CS3	35	
	CS4	36	
	CS5	37	
	XF0	38	
	XF1	39	

Table D-1. Node Numbering

Device Manufacturer	Dip Package		LCC Package	
Altera				
	pin	node		
EP1800	10	69		
	11	70		
	12	71		
	13	72		
	23	73		
	24	74		
	25	75		
	26	76		
	44	77		
	45	78		
	46	79		
	47	80		
	57	81		
	58	82		
	59	83		
	60	84		
Atmel				
	pin	node	pin	node
V750	14	25,35	17	29,39
	15	26,36	18	30,40
	16	27,27	19	31,41
	17	28,38	20	32,42
	18	29,39	21	33,43
	19	30,40	23	34,44
	20	31,41	24	35,45
	21	32,42	25	36,46
	22	33,43	26	37,47
	23	34,44	27	38,49

Table D-1. Node Numbering

Device Manufacturer	Dip Package		LCC Package	
	pin	node	pin	node
		Q2,Q1		Q2,Q1
V2500	4	41,65	5	45,69
	5	42,66	6	46,70
	6	43,67	7	47,71
	7	44,68	8	48,72
	8	45,69	9	49,73
	9	46,70	10	50,74
	11	47,71	13	51,75
	12	48,72	14	52,76
	13	49,73	15	53,77
	14	50,74	16	54,78
	15	51,75	17	55,79
	16	52,76	18	56,80
	24	53,77	27	57,81
	25	54,78	28	58,82
	26	55,79	29	59,83
	27	56,80	30	60,84
	28	57,81	31	61,85
	29	58,82	32	62,86
	31	59,83	35	63,87
	32	60,84	36	64,88
	33	61,85	37	65,89
	34	62,86	38	66,90
	35	63,87	39	67,91
	36	64,88	40	68,92
Cypress				
	pin	node		
P7C330	BR0	29		

Table D-1. Node Numbering

Device Manufacturer	Dip Package		LCC Package
	BR1	30	
	BR2	31	
	BR3	32	
	28	33	
	27	34	
	26	35	
	25	36	
	24	37	
	23	38	
	20	39	
	19	40	
	18	41	
	17	42	
	16	43	
	15	44	
	* IMUX1	45	
	IMUX2	46	
	IMUX3	47	
	IMUX4	48	
	IMUX5	49	
	IMUX6	50	
	pin	node	
P7C331	28	29	
	27	30	
	26	31	
	25	32	
	24	33	
	23	34	
	20	35	
	19	36	
	18	37	
	17	38	

Table D-1. Node Numbering

Device Manufacturer	Dip Package	LCC Package
	16	39
	15	40
	IMUX1	41
	IMUX2	42
	IMUX3	43
	IMUX4	44
	IMUX5	45
	IMUX6	46

* IMUX1 is Input mux between pin 28 and pin 27, IMUX2 is Input mux between pin 26 and 25, etc.

Intel	pin	node
5AC312	2	25
	11	26
	14	27
	15	28
	16	29
	17	30
	18	31
	19	32
	20	33
	21	34
	22	35
	23	36
	pin	node
5AC324	4	41
	5	42
	6	43
	7	44

Table D-1. Node Numbering

Device Manufacturer	Dip Package		LCC Package	
	9	45		
	10	46		
	11	47		
	12	48		
	14	49		
	15	50		
	16	51		
	17	52		
	24	53		
	25	54		
	26	55		
	27	56		
	29	57		
	30	58		
	31	59		
	32	60		
	33	61		
	34	62		
	35	63		
	36	64		
Lattice				
	pin	node	pin	node
GAL6001	slmc0	25	slmc0	29
	slmc1	26	slmc1	30
	slmc2	27	slmc2	31
	slmc3	28	slmc3	32
	slmc4	29	slmc4	33
	slmc5	30	slmc5	34
	slmc6	31	slmc6	35
	slmc7	32	slmc7	36
	14	33	17	37

Table D-1. Node Numbering

Device Manufacturer	Dip Package		LCC Package	
	15	34	18	38
	16	35	19	39
	17	36	20	40
	18	37	21	41
	19	38	22	42
	20	39	23	43
	21	40	24	44
	22	41	25	45
	23	42	26	46
Monolithic Memories				
	pin	node	pin	node
PAL32VX10	14	25	17	29
	15	26	18	30
	16	27	20	31
	17	28	21	32
	18	29	22	33
	19	30	23	34
	20	31	24	35
	21	32	25	36
	22	33	26	37
	23	34	2	738
PLX Technology				
	pin	node		
PLX448	13	25		
	15	26		
	16	27		
	17	28		
	19	29		
	21	30		
	22	31		

Table D-1. Node Numbering

Device Manufacturer	Dip Package	LCC Package
	23	32
Signetics		
	pin	node
PLC415	c0	37
	c1	38
	p0	29
	p1	30
	p2	31
	p3	32
	p4	33
	p5	34
	p6	35
	p7	36
	pin	node
PLC42VA12	14	25
	15	26
	16	27
	17	28
	18	29
	19	30
	20	31
	21	32
	22	33
	23	34
	name	node
PLHS501	T0	53
	T1	54
	T2	55

Table D-1. Node Numbering

Device Manufacturer	Dip Package	LCC Package
	T3	56
	T4	57
	T5	58
	T6	59
	T7	60
	T8	61
	T9	62
	T10	63
	T11	64
	T12	65
	T13	66
	T14	67
	T15	68
	T16	69
	T17	70
	T18	71
	T19	72
	T20	73
	T21	74
	T22	75
	T23	76
	T24	77
	T25	78
	T26	79
	T27	80
	T28	81
	T29	82
	T30	83
	T31	84
	T32	85
	T33	86
	T34	87
	T35	88
	T36	89

Table D-1. Node Numbering

Device Manufacturer	Dip Package	LCC Package
	T37	90
	T38	91
	T39	92
	T40	93
	T41	94
	T42	95
	T43	96
	T44	97
	T45	98
	T46	99
	T47	100
	T48	101
	T49	102
	T50	103
	T51	104
	T52	105
	T53	106
	T54	107
	T55	108
	T56	109
	T57	110
	T58	111
	T59	112
	T60	113
	T61	114
	T62	115
	T63	116
	T64	117
	T65	118
	T66	119
	T67	120
	T68	121
	T69	122
	T70	123

Table D-1. Node Numbering

Device Manufacturer	Dip Package		LCC Package	
	T71	124		
	name	node	name	node
PLHS502	Q0	69	Q0	65
	Q1	70	Q1	66
	Q2	71	Q2	67
	Q3	72	Q3	68
	Q4	73	Q4	69
	Q5	74	Q5	70
	Q6	75	Q6	71
	Q7	76	Q7	72
	Q8	77	Q8	73
	Q9	78	Q9	74
	Q10	79	Q10	75
	Q11	80	Q11	76
	Q12	81	Q12	77
	Q13	82	Q13	78
	Q14	83	Q14	79
	Q15	84	Q15	80
	FB0	85	FB0	81
	FB1	86	FB1	82
	FB2	87	FB2	83
	FB3	88	FB3	84
	FB4	89	FB4	85
	FB5	90	FB5	86
	FB6	91	FB6	87
	FB7	92	FB7	88
	FB8	93	FB8	89
	FB9	94	FB9	90
	FB10	95	FB10	91
	FB11	96	FB11	92
	FB12	97	FB12	93
	FB13	98	FB13	94

Table D-1. Node Numbering

Device Manufacturer	Dip Package		LCC Package	
	FB14	99	FB14	95
	FB15	100	FB15	96
	FB16	101	FB16	97
	FB17	102	FB17	98
	FB18	103	FB18	99
	FB19	104	FB19	100
	FB20	105	FB20	101
	FB21	106	FB21	102
	FB22	107	FB22	103
	FB23	108	FB23	104
	FB24	109	FB24	105
	FB25	110	FB25	106
	FB26	111	FB26	107
	FB27	112	FB27	108
	FB28	113	FB28	109
	FB29	114	FB29	110
	FB30	115	FB30	111
	FB31	116	FB31	112
	FB32	117	FB32	113
	FB33	118	FB33	114
	FB34	119	FB34	115
	FB35	120	FB35	116
	FB36	121	FB36	117
	FB37	122	FB37	118
	FB38	123	FB38	119
	FB39	124	FB39	120
	FB40	125	FB40	121
	FB41	126	FB41	122
	FB42	127	FB42	123
	FB43	128	FB43	124
	FB44	129	FB44	125
	FB45	130	FB45	126
	FB46	131	FB46	127
	FB47	132	FB47	128

Table D-1. Node Numbering

Device Manufacturer	Dip Package		LCC Package	
	FB48	133	FB48	129
	FB49	134	FB49	130
	FB50	135	FB50	131
	FB51	136	FB51	132
	FB52	137	FB52	133
	FB53	138	FB53	134
	FB54	139	FB54	135
	FB55	140	FB55	136
	FB56	141	FB56	137
	FB57	142	FB57	138
	FB58	143	FB58	139
	FB59	144	FB59	140
	FB60	145	FB60	141
	FB61	146	FB61	142
	FB62	147	FB62	143
	FB63	148	FB63	144
	CKFB0	149	CKFB0	145
	CKFB1	150	CKFB1	146
	CKFB2	151	CKFB2	147
	CKFB3	152	CKFB3	148
	name	node		
PLS105	c	35		
	p0	29		
	p1	30		
	p2	31		
	p3	32		
	p4	33		
	p5	34		
	name	node		
PLS155	c	21		

Table D-1. Node Numbering

Device Manufacturer	Dip Package		LCC Package	
	name	node		
PLS157	c	21		
	name	node		
PLS159	c	21		
	name	node	name	node
PLS167	c	31	c	35
	p2	25	p2	29
	p3	26	p3	30
	p4	27	p4	31
	p5	28	p5	32
	p6	29	p6	33
	p7	30	p7	34
	name	node	name	node
PLS168	c	31	c	35
	p4	30	p4	34
	p5	29	p5	33
	p6	28	p6	32
	p7	27	p7	31
	p8	26	p8	30
	p9	25	p9	29
	name	node	name	node
PLS179	c	25	c	29

Table D-1. Node Numbering

Device Manufacturer	Dip Package		LCC Package	
	name	node		
PLS405	c0	37		
	c1	38		
	p0	29		
	p1	30		
	p2	31		
	p3	32		
	p4	33		
	p5	34		
	p6	35		
	p7	36		
Texas Instruments				
	name	node	name	node
FPLS506	c0	41	c0	45
	c1	42	c14	6
	p0	27	p0	31
	p1	28	p1	32
	p2	29	p2	33
	p3	30	p3	34
	p4	31	p4	35
	p5	32	p5	36
	p6	33	p6	37
	p7	34	p7	38
	p8	35	p8	39
	p9	36	p9	40
	p10	37	p10	41
	p11	38	p11	42
	p12	39	p12	43
	p13	40	p13	44
	p14	26	p14	30

Table D-1. Node Numbering

Device Manufacturer	Dip Package		LCC Package	
	p15	25	p15	29
	name	node	name	node
FPLS507	c0	33	c0	37
	c1	34	c1	38
	c2	35	c2	39
	c3	36	c3	40
	c4	37	c4	41
	c5	38	c5	42
	sclr0	39	sclr0	43
	sclr1	40	sclr1	44
	!cnt/hold0	41	!cnt/hold0	45
	!cnt/hold1	42	!cnt/hold1	46
	p0	25	p0	29
	p1	26		p130
	p2	27	p2	31
	p3	28	p3	32
	p4	29	p4	33
	p5	30	p5	34
	p6	31	p6	35
	p7	32	p7	36

Trouble Shooting E

❑ HOW TO GET SUPPORT

The user of the CUPL product may need to have some questions answered or get help with the package in some way. If problems are encountered, LDI customer support may be contacted. There are several ways to do this.

By Telephone (305) 974-0967

By FAX (305) 974-8531

Via Customer Support Electronic Bulletin Board Service (BBS) (305) 974-0612

The BBS requires that a modem be attached to a terminal or a computer. The BBS can be dialed, and messages may be left for the customer support staff. Additionally, files may be uploaded for review at LDI.

Baud:	300, 1200, 2400
Parity:	None
Data Bits:	8
Stop Bits:	1
Password:	user-defined

Special Interest Groups (SIGS) are also available; these can be a great source of information.

If assistance is neeeded with using CUPL, it is a good idea to contact your local dealer or distributor first, for their suggestions as to how to proceed.

❑ Contacting Customer Support

Before contacting Customer Support, make sure to collect the following information:

Make sure that you have a semicolon at the end of each statement. You would be surprised at the number of files we get that contain only this problem. Examine the header section of the PLD file in particular since a missing semicolon in this area will often cause strange results.

Check to make sure that all comment blocks are closed. Many times designers start a comment with /* but forget to close it with */. What happens is that the compiler continues reading until it finds an end of comment marker */. Everything read is considered a comment and is therefore invisible to the compiler.

The CUPL serial number: This enables us to know if there is a known problem with your version.

The CUPL version number.

The device mnemonic that you were using when you encountered the problem.

Fax a copy of the PLD file or upload the file to the BBS. Uploading the file is better because we can test it. Make sure to send the entire file. Sometimes, a problem will seem to be caused by one thing but it is actually caused by something else.

Index

! symbol 119
$DEFINE 128
$ELSE 134
$ENDIF 133
$EXIT 237
$IFDEF 131
$IFNDEF 132
$INCLUDE 130
$MACRO 137
$MEND 139
$MSG 233
$REPEAT 135, 233
$REPEND 137
$SIMOFF 237
$SIMON 238
$TRACE 234
$UNDEF 130
/* symbol 116
absolute file 38
alternate operators 129
Apollo work station 18
APPEND 155
arithmetic
    function 142
    operators 141
ASCII format 38, 309
ASSEMBLY 117
ASSY 117
asynchronous output
    conditional 194
    unconditional 192
AUTOEXEC.BAT 8
base 225
    prefixes 111
BASE keyword 222, 224
Berkeley
    PLA file 38
    PLA format 331
binary sets 157
bit
    masks 158
    positions 159
bit field statements 125, 156
    equality operations 158
    range operations 162
Boolean
    expressions 152
    logic 140, 151
    review 151
    rules 48
build 6, 8, 240
    syntax 245
buildsmt 8
bulletin board 353
buried function 122
C-shell 18, 21
CBLD 239
    allowable macros 243
    building libraries 245
    error messages 283
    extensions 242
    flags 240
    library 241
    running 239

CK 151
CKMUX 150, 151
clock 149, 150, 230
color 11
combinatorial
  logic 173
  output 148
command line
  options - see flag
comments 112, 222
  in test spec file 222
  sample 112
  with $IFDEF 132
COMPANY 117
complement operator 154
CONDITION 199
conditional
  asynchronous output 194, 197
  NEXT 178
  synchronous output 186, 191
configuration 9
  buffers 9
  files 9
constant number bit positions 159
conventions used vi
conversion, base 111
counters 95, 97
CSIM 210
  design example 50
  design simulation 32
  error messages 278
  example running 76
  example source file 72
  flags 214
  input 210
  mnemonic 211
  output 212
  running 213
  simulator directives 232
  test specification 220
CUPL
  assistance 353
  command options 39
  compiler 36
  data flow 3
  design example 50
  device mnemonic 37
  error messages 256
  example running 63
  input 36
  installation 6
  introduction 1
  key features 2
  language 107
  operation 36
  output 38
  overview 1
  running 32
  running using menus 46
  syntax 140
  to XNF 205
CUPL.BCK 22
CUPL.DL 8
data flow 3
DATE 117
decade up/down example 97
DEFAULT 179, 188, 195
default equations 181, 190, 196, 200
DeMorgan's Theorem 152
design examples 49
DESIGNER 117
DEVICE 118
  EP300 290
  EP600 290
  EP900 290
  F155 289
  F157 289
  F159 289

F16V8 292
F179 289
F18V8Z 292
F20V8 292
F501 290
F502 290
F506 291
F507 291
G16V8 293
G20V8 293
G6001 295
P10P8V 296
P12P6V 296
P14P4V 296
P16P2V 296
P16P8V 296
P16RP4V 296
P16RP6V 296
P16RP8V 296
P18CV8 290
P20X10 289
P20X4 289
P20X8 289
P22CV10 296
P22VP10 296
P23S8 296
P29M16 297
P29MA16 298
P32VX10 299
PLD9000 301
PLX448 299
RA10P8 296
RA11P8 296
RA12P8 296
V2500 300
V750 300
devices 240
choosing 59
extensions 142
library also see library
usage notes 289
directives 232
DOC 323
documentation
file 38
format 323
DOS
installation 6
requirements 6
download formats 304
equality operations 158
bit field 158
counter 160
function table 161
equations
address decoding 56
example 56
logic 153
PTOC 252
wait state 56
error
CBLD module 283
CSIM module 278
CSIMA module 279
CUPL module 258
CUPLA module 262
CUPLB module 269
CUPLC module 275
CUPLM module 273
CUPLX module 259
list of messages 256, 257
message suffix 256
messages in list file 38
PTOC module 286
errors message 257
examples
design 49
summary 81

expressions 152
extensions 142
  CBLD 242
  example use 146
  feedback 147
  multiplexer 149
  table of 143
EZ editor 13
fatals 257
fault simulation 238
feedback
  extensions 147
  internal 149
  pin 148
  programmable 147
  registered 148
  test vectors 230
FIELD 125
fields 306
  identifiers 306
file, template 114
flag
  CBLD 240
  compiler option 39, 41
  CSIM 213, 214
  CUPL 39
  multiple option 40
  PTOC 248
  simulator option 214
flip-flops 142, 171
format
  ASCII 304, 309
  Berkeley PLA file 331
  documentation file 323
  download 304
  HL 304, 310
  JEDEC 304
  PDIF file 330
FORMAT keyword 118
function
  arithmetic 142
  buried 122
  control 145
  extensions 142
  table 169
  user-defined 201
  writing equations for 146
function keys 12
  defining 48
FUNCTION keyword 201
FUSE 127
gates, designing 88
GATES.ABS 33
GATES.PLD 29
header information 116
  CSIM 221
  CUPL 116
  in source spec file 72
  keywords 116
  PTOC 251
  template file 114
hex download format 309
HL download file 38
HL format 310
IF 178
imaginary device 301
indexed variables 108
input, CSIM 210
INSTALL 7
installing
  CUPL on Apollo 18
  CUPL on DOS 6
  CUPL on Sun 15
  CUPL on UNIX 15
  CUPL on VAX/VMS 22
  procedure 7
intermediate variable 116, 154
internal nodes 335

introduction iii
  to CUPL 1
JEDEC
  field identifiers 306
  file 38
  format 304
  test vectors 308
keywords
  CSIM 222
  CUPL reserved 109
  header 116, 117
  preprocessor 128
  user-defined 201
language
  elements 107
  syntax 140
LIBCUPL 8
library 240, 241
  CBLD 245
  CSIM 210
  description of 210
  listing contents of 241
list notation 112, 113
LOCATION 117
logic
  description file 36
  evaluation rules 151
  minimization 126
  minimization example 164
  reduction 126
logic equation 116, 153
  complement operator 154
  intermediate variable 154
  with APPEND 155
logical operators 140
  precedence 140
machine ID 11
macros
  CBLD 243
  listing 243
main menu 12
MCUPL 10, 46
MCUPL.CFG 11, 46, 48
MCUPL.EXE 46
memory map, PLD 52
menu
  MCUPL described 47
  running CUPL using 46
  screen 12
  system for MCUPL 10
MIN 126
MIN declaration 126
  examples 126
minimization 126
  flags 44
  levels 45
MISER bit 127
mnemonic
  CSIM prefix 211
  CUPL prefix 37
modulus % symbol 136
multiplexer extension usage 149
NAME 117
negation 187
  conditional 187
  pin declaration 184
  symbol 119, 129
  unconditional 184
NEXT
  conditional 178
  unconditional 176
node
  declaration 122
  numbering 335
NODE keyword 122
non-registered outputs 173
notation, list 112
numbers 110

base conversion 111
base prefix 111
index 113
value range 110
OE 151
OEMUX 150, 151
operation, CUPL 36
operators
alternate 129, 131
arithmetic 141
arithmetic example 135, 136
complement 154
logic rules 107, 140
modulus example 136
precedence 140
option flag - see flag
ORDER 222, 224
statement in source spec file 73
OUT 146
output
CSIM 212
CUPL 38
enable 150
PALASM 208
ambiguities 254
devices 207
format 207
in PTOC 248
operators 129
source file format 249
parentheses 152
in parameter list 201
PARTNO 117
PCAD PDIF format 330
PDIF format 330
pin
assignments 26
assignments example 61
configuration example 60
declaration 116, 119
PIN keyword 119
pin list
PTOC 251
PINNODE 123, 124
PLA
with Berkeley format 331
PLD
design example 51
file example 82
file to PALASM 207
from TTL example 88
gates example 28
output file 250
usage - see device
polarity 119
of internal signal 123
with DeMorgan 154
precedence of operators 140
preload 228
preprocessor commands 128
PRESENT 174
product terms 330
PTOC 248
equations 252
error messages 286
header information 251
pin list 251
PLD output file 250
running 248
SI output file 253
translation ambiguities 254
range
address decoding 58
function 167
operations 162
registered outputs 173
requirements

DOS 6
UNIX 15
VAX/VMS 22
reserved
symbols 110
words 109
REVISION 117
SAMPLE files 64
screen color 11
SEQUENCE 174, 176
SEQUENCED 176
SEQUENCEJK 176
SEQUENCERS 176
SEQUENCET 176
set operations 156
binary equivalent 157
bit field 157
equality 158
seven segment decoder example 102
shorthand notation 112
SI output files 253
signal polarity 119
simple gates example 85
simulator
directives 232
flags 214
small.bld 246
SMT parts 8
SOURCE 245
source file 32
example 54
example creating CSIM 72
format 249
header 74
state bits 173
state machine
combinatorial logic 173
model 171
sample 197
syntax 171, 174
storage registers 173
STUCK 238
subway turnstile example 26
Sun work station 15
symbols, CUPL reserved 110
synchronous output
conditional 186
unconditional 183
syntax 140
arithmetic function 142
arithmetic operators 141
condition 199
extensions 142
logical operators 140
state machine 171, 174
state machine sample 197
TABLE 169
tar 16
target 25
example choosing 59
mnemonic 37
TARGET keyword 245
template 25
template file 114, 116
header information 116
pin declaration 116
title block 116
test conditions 308
test specification file 33
test vector
see also vector
translated by PTOC 254
using * in 75
values 226
text editor 36
timing diagram example 95
title block 116

trace levels 235
translation ambiguities 254
truth tables
    CUPL 141
    variable list 169
TTL to PLD example 88
TURBO bit 127
two bit counter example 95
unconditional
    asynchronous output 192
    NEXT 176
    synchronous output 183
UNIX
    environment 16, 19
    installation 15
    requirements 15
user-defined functions 201
variables 107
    extensions 142
    indexed 108
    intermediate 116, 154
VAX/VMS
    environment 23
    installation 22
vector
    asynchronous 230
    clock 230
    creating 75
    preload 228
    see also test vector
    tables 231
    values 226
VECTORS 222, 226
    in source spec file 73
warnings 257
waveform output 34
XILINX devices 209
XNF 205

Installing CUPL MS-DOS

Note: If the install procedure is interrupted, resume install with the disk that was due to be installed next. For example, if disk 3 was completely installed, resume by typing **INSTALL 4**. If disk 3 was partially installed, resume by typing **INSTALL 3**.

1. Place the diskette labelled **Disk 1** into drive A and type **A:** [Return] and then **INSTALL** [Return].

2. Follow the prompts as they appear on the screen. If a one-character response is requested, type in only the character; do not press the [Return] key.

3. If the computer being used has more than one floppy disk drive, a screen will appear asking which drive should be used.

4. When requested, input the name of the destination drive and the directory name. Use the defaults as a guide.

5. After the source and target drives are selected, copying will begin. The **Message** window will indicate when the next diskette should be inserted.

6. If you chose not to install the example files and later wish to use them, copy EXAMPLES.ARC from Disk 4 to your CUPL directory and type **ARCE EXAMPLES.ARC** .

7. After copying is complete, a screen will appear asking if the **SET LIBCUPL=** command should be added to the **AUTOEXEC.BAT** file. Respond with either a **Y** for yes or an **N** for no.
   The following screen explains what is needed to add the **PATH** statement to the AUTOEXEC.BAT file. Make certain that the environment is updated by either re-booting the computer or by running the AUTOEXEC.BAT file.
   For further details, refer to page U2-8 in the CUPL Manual.

8. Once the installation is complete, type **BUILD** [Return]. This will combine the device libraries into one device library named **CUPL.DL**. To add the **SMT** library, type **BUILDSMT** [Return] after completing BUILD.
   For further details, refer to page U2-8 in the CUPL Manual.

9. If the CUPL front-end menu system **MCUPL** is to be used, edit the MCUPL configuration file **MCUPL.CFG**, and configure it as desired.
   For further details, refer to page U2-1 in the CUPL Manual.

What's New in CUPL 4.0a

The latest version of CUPL, the Universal Compiler for Programmable Logic, is 4.0a. This version encompasses Software Improvements, New Architecture Features, New Device Architectures, menu support for New Devices using existing mnemonics, and mnemonic changes and revisions.

❑ *Software Improvements*

CUPL Executables

Increased operation speeds.

Installation

Installation has been modified to recognize the difference between an update and a brand new copy. When the package is an update, the install queries for original serial number. The install also queries for company name which is displayed during compilation.

Menu

Menu operation has been modified to improve the method of moving around the directory structure. If the directory mask is set to *.*, the directories are displayed along with the files. It is possible to change to any of those directories simply by highlighting and selecting the desired directory.

Menu operation now expects to find the INCLUDE and individual user library files in the current working directory.

EZEDIT

The exiting procedure for EZEDIT has been improved to allow for either highlighting the SAVE or QUIT option or typing 'S' or 'Q'. If

the quit option is selected, EZEDIT queries whether or not to save the file.

Simulation

The simulation waveform displayer now displays 10 waveforms. It can also allow the user to arbitrarily change the order of the waveforms and combine up to 8 waveforms to form a BUS with hex output.

The waveform hardcopy supports additional printers.

CSIM recognizes the symbol '.' as a valid path symbol.

DOC File

The DOC file will now show the total number of product terms used in a FPLA device.

Syntax

The OUT equations generated from a D-CE type state machine have been corrected. This affects the GAL6001.

OUTPUT Formats

Interfacing to PCAD 4.5 is supported by CUPL PDIF files.

Virtual device supports PLA output format.

❑ *New Architecture Features*

Automatic Product Term Allocation

Automatic product term allocation is now supported. Intel's 5AC312 and 5AC324, and Altera's EP512 have this capability.

Sharable Product Terms

Some devices have sharable product terms that can be isolated for single use by a buried register. This makes the product term unusable by anything else. CUPL determines when these types of product terms are being used by more than one storage component

and generates an error message. The best example of this is the ATV2500.

❑ *New Device Architectures*

Manufacturer	Device Name	Device Mnemonic
Cypress	CY10E301/100E301	P16P8C, P16P8CLCC
	CY10E302/100E302	P16P4C, P16P4CLCC
	PLDC18G8	P18G8
Exel	XL78C800	XL78C800
Intel	85C224	PC224, PC224LCC
National	PAL1016C4/10016C4	P1016C4
	PAL1016PE8/10016PE8	P1016PE8
	PAL1020RP4/10020RP4	P1020RP4
Signetics	PHD48N22	F48N22

❑ *New Devices*

Manufacturer	Device Name	Device Mnemonic
Altera	EP330	EP320
	EP630	EP600
AMD/MMI	PALCE630	EP600
Cypress	PAL22V10C	P22V10
	PAL22VP10C	P22VP10
Intel	85C060	EP600
	85C090	EP900
Signetics	10020EV8	P1020EV8
	10H20EV8	P1020EV8

Texas Inst.	EP330	EP320
	TIBPAL1602	P16L8

❑ *Mnemonic Changes*

SGS-Thomson changed the name of GAL36V18 to GAL6001.

The mnemonic **PC220** was removed. Use EP320 as the mnemonic for the Intel 85C220.

The incorrectly listed **P20RP10** mnemonic was removed from the menu.

The **P1016** family can now correctly simulate the active high common reset pin. It previously was simulated using an active low reset pin.

The **G20V8A LCC** and **G16Z8 LCC** mnemonics were added.

PA7024 support will be available on CUPL-386.

❑ *Mnemonic Revision Changes*

Mnemonic	Rev.	Description
ep1800	09	Corrected AR simulation for buried nodes. synchronous Added INT for global macrocells. Defaulted buried registers to use synchronous Clock.Corrected polarity definition of buried registers.
ep312	05	Corrected pin 16 combinatorial feedback column number.
f18v8z	04	Corrected configuration bits.
f20v8	03	Improved mode selection.
f30s16	02	Added automatic defaulting of OE terms to 'b'1.
f273	02	Corrected total number of fuses.

f415	03	Corrected AR and AP default. Corrected buried node feedback.
g16v8	09	Changed device to power-up to zero in the registers.
g20v8	03	Changed device to power-up to zero in the registers.
g22v10	02	Corrected common clock pin. Corrected DOC file fuse map.
g6001	13	Corrected feedback polarity of output registers defined as buried nodes.
p1016rm4	02	Corrected clocking of registers using all clock pins. Corrected DOC file fuse map.
p20cg10	02	Corrected definition of input only on an I/O pin.
p20l8lcc	09	Corrected DOC file fuse map.
p20r6lcc	11	Corrected clocking of registers.
p20r8lcc	10	Corrected DOC file fuse map.
p24l10	02	Changed to new specifications.
p24r10	02	Changed to new specifications.
p26v12	03	Exchanged pins 7(VCC) and 21(GND).
p29ma16	11	Corrected AP definition for node 31.
p32vx10	10	Corrected pin 16 registered feedback column number.
p7c330	05	Improved simulation for IOD.
p7c331	04	Corrected total number of fuses.
v750lcc	05	Corrected buried node numbers. Corrected DOC file fuse map.
v2500	07	Corrected fuse array number for pin nodes 51, 55, 71, 75, 76. Corrected configuration of I/O pins for dedicated input.
v2500lcc	08	Corrected pin 89 AR array number.

DEVICE CODE	DEVICE MNEMONIC	# OF PINS	# OF FUSES	# OF PTERMS	REC. VER. LEVEL
ALTERA					
EP1200	EP1200	40	15146	236	3.2b
EP1210	EP1200	40	15146	236	3.2b
EP1800	EP1800	68	42490	480	4.0a
EP1810	EP1800	68	42490	480	4.0a
EP300	EP300	20	2720	74	
EP310	EP300	20	2720	74	
EP320	EP320	20	2916	72	2.15a
EP330	EP320	20	2916	72	2.15a
EP512	EP312	24	13713	200	4.0a
EP600	EP600	24	6482	160	3.0a
EP610	EP600	24	6482	160	3.0a
EP630	EP600	24	6482	160	3.0a
EP900	EP900	40	17402	240	3.2b
EP910	EP900	40	17402	240	3.2b
AMD/MMI					
AMPAL16H8	P16H8	20	2048	64	
AMPAL16HD8	P16HD8	20	2048	64	
AMPAL16L8	P16L8	20	2048	64	1.01a
AMPAL16LD8	P16LD8	20	2048	64	
AMPAL16R4	P16R4	20	2048	64	1.01a
AMPAL16R6	P16R6	20	2048	64	1.01a
AMPAL16R8	P16R8	20	2048	64	1.01a
AMPAL18P8	P18P8	20	2600	72	2.01a
AMPAL20L10	P20L10	24	1600	40	1.01a
AMPAL20L8	P20L8	24	2560	64	2.00a
AMPAL20R4	P20R4	24	2560	64	2.00a
AMPAL20R6	P20R6	24	2560	64	2.00a
AMPAL20R8	P20R8	24	2560	64	2.00a
AMPAL20RP10	P20RP10A	24	3210	80	2.11b
AMPAL20RP4	P20RP4A	24	3450	86	2.11b
AMPAL20RP6	P20RP6A	24	3370	84	2.11b
AMPAL20RP8	P20RP8A	24	3290	82	2.11b
AMPAL20XRP10	P20XRP10	24	3210	80	2.11c
AMPAL20XRP4	P20XRP4	24	3450	86	2.11c
AMPAL20XRP6	P20XRP6	24	3370	84	2.11c
AMPAL20XRP8	P20XRP8	24	3290	82	2.11c
AMPAL22P10	P22P10A	24	3970	90	2.11b
AMPAL22V10	P22V10	24	5828	132	2.11a
AMPAL22XP10	P22XP10	24	3970	90	2.11c
AMPAL23S8	P23S8	20	6234	135	2.11a
PAL10H/10020EG8	P1020EG8	24	3616	80	2.11b
PAL10H/10020EV8	P1020EV8	24	3616	80	3.0a
PAL10H20G8	P1020G8	24	1352	32	2.11b
PAL10H20P8	P1020P8	24	1352	32	
PAL10H8	P10H8	20	320	16	
PAL10H8-2	P10H8	20	320	16	
PAL10L8	P10L8	20	320	16	
PAL10L8-2	P10L8	20	320	16	
PAL12H6	P12H6	20	384	16	
PAL12H6-2	P12H6	20	384	16	
PAL12L10	P12L10	24	480	20	1.01a
PAL12L6	P12L6	20	384	16	
PAL12L6-2	P12L6	20	384	16	
PAL14H4	P14H4	20	448	16	
PAL14H4-2	P14H4	20	448	16	
PAL14L4	P14L4	20	448	16	
PAL14L4-2	P14L4	20	448	16	
PAL14L8	P14L8	24	560	20	
PAL16C1	P16C1	20	512	16	1.01a
PAL16C1-2	P16C1	20	512	16	1.01a
PAL16H2	P16H2	20	512	16	
PAL16H2-2	P16H2	20	512	16	
PAL16L2	P16L2	20	512	16	
PAL16L2-2	P16L2	20	512	16	

DEVICE CODE	DEVICE MNEMONIC	# OF PINS	# OF FUSES	# OF PTERMS	REC. VER. LEVEL
AMD/MMI					
PAL16L6	P16L6	24	640	20	
PAL16L8	P16L8	20	2048	64	1.01a
PAL16L8A	P16L8	20	2048	64	1.01a
PAL16L8A-2	P16L8	20	2048	64	1.01a
PAL16L8A-4	P16L8	20	2048	64	1.01a
PAL16L8B	P16L8	20	2048	64	1.01a
PAL16L8B-2	P16L8	20	2048	64	1.01a
PAL16L8B-4	P16L8	20	2048	64	1.01a
PAL16L8BP	P16L8	20	2048	64	1.01a
PAL16L8D	P16L8	20	2048	64	1.01a
PAL16P8A	P16P8	20	2056	64	
PAL16P8B	P16P8	20	2056	64	
PAL16R4	P16R4	20	2048	64	1.01a
PAL16R4A	P16R4	20	2048	64	1.01a
PAL16R4A-2	P16R4	20	2048	64	1.01a
PAL16R4A-4	P16R4	20	2048	64	1.01a
PAL16R4B	P16R4	20	2048	64	1.01a
PAL16R4B-2	P16R4	20	2048	64	1.01a
PAL16R4B-4	P16R4	20	2048	64	1.01a
PAL16R4BP	P16R4	20	2048	64	1.01a
PAL16R4D	P16R4	20	2048	64	1.01a
PAL16R6	P16R6	20	2048	64	1.01a
PAL16R6A	P16R6	20	2048	64	1.01a
PAL16R6A-2	P16R6	20	2048	64	1.01a
PAL16R6A-4	P16R6	20	2048	64	1.01a
PAL16R6B	P16R6	20	2048	64	1.01a
PAL16R6B-2	P16R6	20	2048	64	1.01a
PAL16R6B-4	P16R6	20	2048	64	1.01a
PAL16R6BP	P16R6	20	2048	64	1.01a
PAL16R6D	P16R6	20	2048	64	1.01a
PAL16R8	P16R8	20	2048	64	1.01a
PAL16R8A	P16R8	20	2048	64	1.01a
PAL16R8A-2	P16R8	20	2048	64	1.01a
PAL16R8A-4	P16R8	20	2048	64	1.01a
PAL16R8B	P16R8	20	2048	64	1.01a
PAL16R8B-2	P16R8	20	2048	64	1.01a
PAL16R8B-4	P16R8	20	2048	64	1.01a
PAL16R8BP	P16R8	20	2048	64	1.01a
PAL16R8D	P16R8	20	2048	64	1.01a
PAL16RA8	P16RA8	20	2056	64	
PAL16RP4A	P16RP4	20	2056	64	
PAL16RP6A	P16RP6	20	2056	64	
PAL16RP8A	P16RP8	20	2056	64	
PAL18L4	P18L4	24	720	20	
PAL20C1	P20C1	24	640	16	
PAL20L10	P20L10	24	1600	40	1.01a
PAL20L10A	P20L10	24	1600	40	1.01a
PAL20L2	P20L2	24	640	16	
PAL20L8	P20L8	24	2560	64	2.00a
PAL20L8A	P20L8	24	2560	64	2.00a
PAL20L8A-2	P20L8	24	2560	64	2.00a
PAL20L8B	P20L8	24	2560	64	2.00a
PAL20R4	P20R4	24	2560	64	2.00a
PAL20R4A	P20R4	24	2560	64	2.00a
PAL20R4A-2	P20R4	24	2560	64	2.00a
PAL20R4B	P20R4	24	2560	64	2.00a
PAL20R6	P20R6	24	2560	64	2.00a
PAL20R6A	P20R6	24	2560	64	2.00a
PAL20R6A-2	P20R6	24	2560	64	2.00a
PAL20R6B	P20R6	24	2560	64	2.00a
PAL20R8	P20R8	24	2560	64	2.00a
PAL20R8A	P20R8	24	2560	64	2.00a
PAL20R8A-2	P20R8	24	2560	64	2.00a
PAL20R8B	P20R8	24	2560	64	2.00a
PAL20RA10	P20RA10	24	3210	80	2.02a

DEVICE CODE	DEVICE MNEMONIC	# OF PINS	# OF FUSES	# OF PTERMS	REC. VER. LEVEL
AMD/MMI					
PAL20RS10	P20RS10	24	3338	80	
PAL20RS4	P20RS4	24	3330	80	
PAL20RS8	P20RS8	24	3338	80	
PAL20S10	P20S10	24	3322	80	
PAL20X10	P20X10	24	1600	40	2.00a
PAL20X10A	P20X10	24	1600	40	2.00a
PAL20X4	P20X4	24	1600	40	2.00a
PAL20X4A	P20X4	24	1600	40	2.00a
PAL20X8	P20X8	24	1600	40	2.00a
PAL20X8A	P20X8	24	1600	40	2.00a
PAL22IP6-35	P22IP6	24	3294	72	3.0
PAL22RX8	P22RX8	24	3616	82	2.15a
PAL22RX8A	P22RX8	24	3616	82	2.15a
PAL24L10	P24L10	28	3840	80	4.0a
PAL24R10	P24R10	28	3840	80	4.0a
PAL32R16	P32R16	40	8466	128	
PAL32VX10	P32VX10	24	9738	152	4.0a
PAL32VX10A	P32VX10	24	9738	152	4.0a
PAL64R32	P64R32	84	33316	256	
PAL6L16A	P6L16	24	192	16	2.01a
PAL8L14A	P8L14	24	224	16	2.01a
PALC16L8Q	P16L8	20	2048	64	1.01a
PALC16L8Z	P16L8	20	2048	64	1.01a
PALC16R4Q	P16R4	20	2048	64	1.01a
PALC16R4Z	P16R4	20	2048	64	1.01a
PALC16R6Q	P16R6	20	2048	64	1.01a
PALC16R6Z	P16R6	20	2048	64	1.01a
PALC16R8Q	P16R8	20	2048	64	1.01a
PALC16R8Z	P16R8	20	2048	64	1.01a
PALC18U8	P18U8	20	2688	72	3.0
PALC20L8Z	P20L8	24	2560	64	2.00a
PALC20R4Z	P20R4	24	2560	64	2.00a
PALC20R6Z	P20R6	24	2560	64	2.00a
PALC20R8Z	P20R8	24	2560	64	2.00a
PALC22V10H	P22V10	24	5828	132	2.11a
PALC22V10Q	P22V10	24	5828	132	2.11a
PALCE16V8	G16V8	20	2194	64	4.0a
PALCE20RA10	P20RA10	24	3210	80	2.02a
PALCE20V8	G20V8	24	560	20	4.0a
PALCE22V10	P22V10	24	5828	132	2.11a
PALCE22V10Z	P22V10	24	5828	132	2.11a
PALCE26V12H	P26V12	28	7848	150	3.0
PALCE29M16	P29M16	24	11040	188	3.2b
PALCE29MA16	P29MA16	24	11460	188	4.0a
PALCE630	EP600	24	6482	160	3.0a
PLS105	F105	28	3553	48	2.00a
PLS167A/B	F167	24	3361	48	2.00a
PLS168A/B	F168	24	3553	48	2.10a
PLS30K12	F30K12	28	7424	72	3.2b
PLS30S16	F30S16	28	7236	71	4.0a
PMS14R21	P14R21	24	3137	86	3.0
ATMEL					
AT22V10	P22V10	24	5828	132	2.11a
ATV2500	V2500	40	71644	416	4.0a
ATV750	V750	24	14394	171	4.0a
CYPRESS					
CY100E301	P16P8C	24	2056	32	4.0a
CY100E302	P16P4C	24	2056	32	4.0a
CY10E301	P16P8C	24	2056	32	4.0a
CY10E302	P16P4C	24	2056	32	4.0a
CY7C330	P7C330	28	17082	258	4.0a
CY7C331	P7C331	28	11934	216	4.0a
CY7C332	P7C332	28	9902	194	3.2b

DEVICE CODE	DEVICE MNEMONIC	# OF PINS	# OF FUSES	# OF PTERMS	REC. VER. LEVEL
CYPRESS					
PAL16L8A	P16L8	20	2048	64	1.01a
PAL16L8A-2	P16L8	20	2048	64	1.01a
PAL16R4A	P16R4	20	2048	64	1.01a
PAL16R4A-2	P16R4	20	2048	64	1.01a
PAL16R6A	P16R6	20	2048	64	1.01a
PAL16R6A-2	P16R6	20	2048	64	1.01a
PAL16R8A	P16R8	20	2048	64	1.01a
PAL16R8A-2	P16R8	20	2048	64	1.01a
PAL22V10C	P22V10	24	5828	132	2.11a
PAL22VP10C	P22VP10	24	5828	132	2.11a
PALC12L10	P12L10	24	480	20	1.01a
PALC14L8	P14L8	24	560	20	
PALC16L6	P16L6	24	640	20	
PALC16L8	P16L8	20	2048	64	1.01a
PALC16R4	P16R4	20	2048	64	1.01a
PALC16R6	P16R6	20	2048	64	1.01a
PALC16R8	P16R8	20	2048	64	1.01a
PALC18L4	P18L4	24	720	20	
PALC20L10	P20L10	24	1600	40	1.01a
PALC20L2	P20L2	24	640	16	
PALC20L8	P20L8	24	2560	64	2.00a
PALC20R4	P20R4	24	2560	64	2.00a
PALC20R6	P20R6	24	2560	64	2.00a
PALC20R8	P20R8	24	2560	64	2.00a
PALC22V10	P22V10	24	5828	132	2.11a
PLDC18G8	P18G8	20	2624	72	4.0a
PLDC20G10	P20G10	24	3990	90	
PLDC20RA10	P20RA10	24	3210	80	2.02a
EXEL					
XL78C800	XL78C800	24	6400	66	4.0a
FAIRCHILD					
93Z458	F100	28	1928	48	2.00a
93Z459	F100	28	1928	48	2.00a
F16L8	P16L8	20	2048	64	1.01a
F16P8	P16P8	20	2056	64	
F16R4	P16R4	20	2048	64	1.01a
F16R6	P16R6	20	2048	64	1.01a
F16R8	P16R8	20	2048	64	1.01a
F16RP4	P16RP4	20	2056	64	
F16RP6	P16RP6	20	2056	64	
F16RP8	P16RP8	20	2056	64	
F20P8	P20P8	24	2568	64	2.15a
F20RP4	P20RP4	24	2568	64	
F20RP6	P20RP6	24	2568	64	
F20RP8	P20RP8	24	2568	64	
GAZELLE					
GA22V10	P22V10	24	5828	132	2.11a
GA22VP10	P22VP10	24	5828	132	2.11a
GA23S8	P23S8	20	6234	135	2.11a
GA23SV8	P23SV8	20	6242	135	3.0
GOULD					
PEEL153	F153	20	1842	42	2.15a
PEEL173	F173	24	2178	42	2.10a
PEEL18CV8	P18CV8	20	2696	74	2.15a
PEEL22CV10	P22V10	24	5828	132	2.11a
PEEL22CV10Z	P22CV10Z	24	5873	132	3.2b
PEEL253	F253	20	2378	42	3.0
PEEL273	F273	24	2714	42	4.0a
HARRIS					
HPL16H8	P16H8	20	2048	64	

DEVICE CODE	DEVICE MNEMONIC	# OF PINS	# OF FUSES	# OF PTERMS	REC. VER. LEVEL
HARRIS					
HPL16L8	P16L8	20	2048	64	1.01a
HPL16LC8	P16L8	20	2048	64	1.01a
HPL16P8	P16P8H	20	2056	64	
HPL16R4	P16R4	20	2048	64	1.01a
HPL16R6	P16R6	20	2048	64	1.01a
HPL16R8	P16R8	20	2048	64	1.01a
HPL16RC4	P16RP4	20	2056	64	
HPL16RC6	P16RP6	20	2056	64	
HPL16RC8	P16RP8	20	2056	64	
HPL77153	F153	20	1842	42	2.15a
HPL77209	P16L8	20	2048	64	1.01a
HPL77216	P16P8H	20	2056	64	
HPL82S153	F153	20	1842	42	2.15a
ICT					
PEEL153	F153	20	1842	42	2.15a
PEEL173	F173	24	2178	42	2.10a
PEEL18CV8	P18CV8	20	2696	74	2.15a
PEEL20CG10	P20CG10	24	4088	92	4.0a
PEEL22CV10	P22V10	24	5828	132	2.11a
PEEL22CV10Z	P22CV10Z	24	5873	132	3.2b
PEEL253	F253	20	2378	42	3.0
PEEL273	F273	24	2714	42	4.0a
INTEL					
5AC312	EP312	24	13713	200	4.0a
5AC324	EP324	40	47493	394	4.0a
5C031	EP300	20	2720	74	
5C032	EP320	20	2916	72	2.15a
5C060	EP600	24	6482	160	3.0a
5C090	EP900	40	17402	240	3.2b
5C180	EP1800	68	42490	480	4.0a
85C060	EP600	24	6482	160	3.0a
85C090	EP900	40	17402	240	3.2b
85C220	EP320	20	2916	72	2.15a
85C224	PC224	24	3204	72	4.0a
85C508	PC508	28	256	8	3.2a
LATTICE					
GAL16V8	G16V8	20	2194	64	4.0a
GAL16V8A	G16V8A	20	2194	64	4.0a
GAL16Z8	G16Z8	24	2195	64	
GAL18V10	G18V10	20	3540	96	3.2a
GAL20RA10	G20RA10	24	3274	80	3.2a
GAL20V8	G20V8	24	560	20	4.0a
GAL20V8A	G20V8A	24	2706	64	4.0a
GAL22V10	G22V10	24	5892	132	4.0a
GAL26CV12	G26CV12	28	6432	120	3.2a
GAL6001	G6001	24	8294	75	4.0a
RAL10H8	P10H8	20	320	16	
RAL10L8	P10L8	20	320	16	
RAL10P8	P10P8	20	328	16	
RAL12H6	P12H6	20	384	16	
RAL12L6	P12L6	20	384	16	
RAL12P6	P12P6	20	390	16	
RAL14H4	P14H4	20	448	16	
RAL14H8	G20V8	24	560	20	4.0a
RAL14L4	P14L4	20	448	16	
RAL14L8	P14L8	24	560	20	
RAL14P4	P14P4	20	452	16	
RAL14P8	P14P8	24	568	20	
RAL16C1	P16C1	20	512	16	1.01a
RAL16H2	P16H2	20	512	16	
RAL16H6	G20V8	24	560	20	4.0a
RAL16H8	P16H8	20	2048	64	

DEVICE CODE	DEVICE MNEMONIC	# OF PINS	# OF FUSES	# OF PTERMS	REC. VER. LEVEL
LATTICE					
RAL16L2	P16L2	20	512	16	
RAL16L6	P16L6	24	640	20	
RAL16L8	P16L8	20	2048	64	1.01a
RAL16P2	P16P2	20	514	16	
RAL16P6	P16P6	24	646	20	
RAL16P8	P16P8	20	2056	64	
RAL16R4	P16R4	20	2048	64	1.01a
RAL16R6	P16R6	20	2048	64	1.01a
RAL16R8	P16R8	20	2048	64	1.01a
RAL16RP4	P16RP4	20	2056	64	
RAL16RP6	P16RP6	20	2056	64	
RAL16RP8	P16RP8	20	2056	64	
RAL18H4	G20V8	24	560	20	4.0a
RAL18L4	P18L4	24	720	20	
RAL18P4	P18P4	24	724	20	
RAL20H2	G20V8	24	560	20	4.0a
RAL20H8	G20V8	24	560	20	4.0a
RAL20L2	P20L2	24	640	16	
RAL20L8	P20L8	24	2560	64	2.00a
RAL20P2	P20P2	24	642	16	
RAL20P8	P20P8	24	2568	64	2.15a
RAL20R4	P20R4	24	2560	64	2.00a
RAL20R6	P20R6	24	2560	64	2.00a
RAL20R8	P20R8	24	2560	64	2.00a
RAL20RP4	P20RP4	24	2568	64	
RAL20RP6	P20RP6	24	2568	64	
RAL20RP8	P20RP8	24	2568	64	
NATIONAL					
87X839	F839	24	1094	32	2.00a
87X840	F839	24	1094	32	2.00a
GAL16V8	G16V8	20	2194	64	4.0a
GAL16V8A	G16V8A	20	2194	64	4.0a
GAL20RA10	G20RA10	24	3274	80	3.2a
GAL20V8	G20V8	24	560	20	4.0a
GAL20V8A	G20V8A	24	2706	64	4.0a
GAL22V10	G22V10	24	5892	132	4.0a
GAL6001	G6001	24	8294	75	4.0a
PAL10012C4	P1012C4	24	2056	32	3.2a
PAL10016C4	P1016C4	28	2056	32	4.0a
PAL10016LD4	P1016LD4	24	2056	64	4.0a
PAL10016LD8	P1016LD8	24	2056	64	4.0a
PAL10016LM4	P1016LM4	24	2056	32	4.0a
PAL10016P4	P1016P4	24	2056	32	3.0
PAL10016P8	P1016P8	24	2056	64	2.10a
PAL10016PE8	P1016PE8	28	2056	64	4.0a
PAL10016RD4	P1016RD4	24	2056	64	4.0a
PAL10016RD8	P1016RD8	24	2056	64	4.0a
PAL10016RM4	P1016RM4	24	2056	32	4.0a
PAL10020RP4	P1020RP4	28	2568	32	4.0a
PAL1012C4	P1012C4	24	2056	32	3.2a
PAL1016C4	P1016C4	28	2056	32	4.0a
PAL1016LD4	P1016LD4	24	2056	64	4.0a
PAL1016LD8	P1016LD8	24	2056	64	4.0a
PAL1016LM4	P1016LM4	24	2056	32	4.0a
PAL1016P4	P1016P4	24	2056	32	3.0
PAL1016P8	P1016P8	24	2056	64	2.10a
PAL1016PE8	P1016PE8	28	2056	64	4.0a
PAL1016RD4	P1016RD4	24	2056	64	4.0a
PAL1016RD8	P1016RD8	24	2056	64	4.0a
PAL1016RM4	P1016RM4	24	2056	32	4.0a
PAL1020RP4	P1020RP4	28	2568	32	4.0a
PAL10H8	P10H8	20	320	16	
PAL10H8/16V8	P10H8	20	320	16	
PAL10H8A	P10H8	20	320	16	

DEVICE CODE	DEVICE MNEMONIC	# OF PINS	# OF FUSES	# OF PTERMS	REC. VER. LEVEL
NATIONAL					
PAL10H8A2	P10H8	20	320	16	
PAL10L8	P10L8	20	320	16	
PAL10L8/16V8	P10L8	20	320	16	
PAL10L8A	P10L8	20	320	16	
PAL10L8A2	P10L8	20	320	16	
PAL10P8/16V8	P10P8	20	328	16	
PAL12H6	P12H6	20	384	16	
PAL12H6/16V8	P12H6	20	384	16	
PAL12H6A	P12H6	20	384	16	
PAL12H6A2	P12H6	20	384	16	
PAL12L10	P12L10	24	480	20	1.01a
PAL12L6	P12L6	20	384	16	
PAL12L6/16V8	P12L6	20	384	16	
PAL12L6A	P12L6	20	384	16	
PAL12L6A2	P12L6	20	384	16	
PAL12P6/16V8	P12P6	20	390	16	
PAL14H4	P14H4	20	448	16	
PAL14H4/16V8	P14H4	20	448	16	
PAL14H4A	P14H4	20	448	16	
PAL14H4A2	P14H4	20	448	16	
PAL14H8/20V8	G20V8	24	560	20	4.0a
PAL14L4	P14L4	20	448	16	
PAL14L4/16V8	P14L4	20	448	16	
PAL14L4A	P14L4	20	448	16	
PAL14L4A2	P14L4	20	448	16	
PAL14L8	P14L8	24	560	20	
PAL14L8/20V8	P14L8	24	560	20	
PAL14P4/16V8	P14P4	20	452	16	
PAL14P8/20V8	P14P8	24	568	20	
PAL16C1	P16C1	20	512	16	1.01a
PAL16C1A	P16C1	20	512	16	1.01a
PAL16C1A2	P16C1	20	512	16	1.01a
PAL16H2	P16H2	20	512	16	
PAL16H2/16V8	P16H2	20	512	16	
PAL16H2A	P16H2	20	512	16	
PAL16H2A2	P16H2	20	512	16	
PAL16H6/20V8	G20V8	24	560	20	4.0a
PAL16H8/16V8	P16H8	20	2048	64	
PAL16L2	P16L2	20	512	16	
PAL16L2/16V8	P16L2	20	512	16	
PAL16L2A	P16L2	20	512	16	
PAL16L2A2	P16L2	20	512	16	
PAL16L6	P16L6	24	640	20	
PAL16L6/20V8	P16L6	24	640	20	
PAL16L8	P16L8	20	2048	64	1.01a
PAL16L8/16V8	P16L8	20	2048	64	1.01a
PAL16L8A	P16L8	20	2048	64	1.01a
PAL16L8A2	P16L8	20	2048	64	1.01a
PAL16L8B	P16L8	20	2048	64	1.01a
PAL16L8B2	P16L8	20	2048	64	1.01a
PAL16L8D	P16L8	20	2048	64	1.01a
PAL16P2/16V8	P16P2	20	514	16	
PAL16P6/20V8	P16P6	24	646	20	
PAL16P8	P16P8	20	2056	64	
PAL16P8/16V8	P16P8	20	2056	64	
PAL16R4	P16R4	20	2048	64	1.01a
PAL16R4/16V8	P16R4	20	2048	64	1.01a
PAL16R4A	P16R4	20	2048	64	1.01a
PAL16R4A2	P16R4	20	2048	64	1.01a
PAL16R4B	P16R4	20	2048	64	1.01a
PAL16R4B2	P16R4	20	2048	64	1.01a
PAL16R4D	P16R4	20	2048	64	1.01a
PAL16R6	P16R6	20	2048	64	1.01a
PAL16R6/16V8	P16R6	20	2048	64	1.01a
PAL16R6A	P16R6	20	2048	64	1.01a

DEVICE CODE	DEVICE MNEMONIC	# OF PINS	# OF FUSES	# OF PTERMS	REC. VER. LEVEL
NATIONAL					
PAL16R6A2	P16R6	20	2048	64	1.01a
PAL16R6B	P16R6	20	2048	64	1.01a
PAL16R6B2	P16R6	20	2048	64	1.01a
PAL16R6D	P16R6	20	2048	64	1.01a
PAL16R8	P16R8	20	2048	64	1.01a
PAL16R8/16V8	P16R8	20	2048	64	1.01a
PAL16R8A	P16R8	20	2048	64	1.01a
PAL16R8A2	P16R8	20	2048	64	1.01a
PAL16R8B	P16R8	20	2048	64	1.01a
PAL16R8B2	P16R8	20	2048	64	1.01a
PAL16R8D	P16R8	20	2048	64	1.01a
PAL16RA8	P16RA8	20	2056	64	
PAL16RP4	P16RP4	20	2056	64	
PAL16RP4/16V8	P16RP4	20	2056	64	
PAL16RP6	P16RP6	20	2056	64	
PAL16RP6/16V8	P16RP6	20	2056	64	
PAL16RP8	P16RP8	20	2056	64	
PAL16RP8/16V8	P16RP8	20	2056	64	
PAL18H4/20V8	G20V8	24	560	20	4.0a
PAL18L4	P18L4	24	720	20	
PAL18L4/20V8	P18L4	24	720	20	
PAL18P4/20V8	P18P4	24	724	20	
PAL20C1	P20C1	24	640	16	
PAL20H2/20V8	G20V8	24	560	20	4.0a
PAL20H8/20V8	G20V8	24	560	20	4.0a
PAL20L10	P20L10	24	1600	40	1.01a
PAL20L10A	P20L10	24	1600	40	1.01a
PAL20L2	P20L2	24	640	16	
PAL20L2/20V8	P20L2	24	640	16	
PAL20L8	P20L8	24	2560	64	2.00a
PAL20L8/20V8	P20L8	24	2560	64	2.00a
PAL20L8A	P20L8	24	2560	64	2.00a
PAL20L8B	P20L8	24	2560	64	2.00a
PAL20L8D	P20L8	24	2560	64	2.00a
PAL20P2/20V8	P20P2	24	642	16	
PAL20P8/20V8	P20P8	24	2568	64	2.15a
PAL20R4	P20R4	24	2560	64	2.00a
PAL20R4/20V8	P20R4	24	2560	64	2.00a
PAL20R4A	P20R4	24	2560	64	2.00a
PAL20R4B	P20R4	24	2560	64	2.00a
PAL20R4D	P20R4	24	2560	64	2.00a
PAL20R6	P20R6	24	2560	64	2.00a
PAL20R6/20V8	P20R6	24	2560	64	2.00a
PAL20R6A	P20R6	24	2560	64	2.00a
PAL20R6B	P20R6	24	2560	64	2.00a
PAL20R6D	P20R6	24	2560	64	2.00a
PAL20R8	P20R8	24	2560	64	2.00a
PAL20R8/20V8	P20R8	24	2560	64	2.00a
PAL20R8A	P20R8	24	2560	64	2.00a
PAL20R8B	P20R8	24	2560	64	2.00a
PAL20R8D	P20R8	24	2560	64	2.00a
PAL20RA10	P20RA10	24	3210	80	2.02a
PAL20RP4/20V8	P20RP4	24	2568	64	
PAL20RP6/20V8	P20RP6	24	2568	64	
PAL20RP8/20V8	P20RP8	24	2568	64	
PAL20X10	P20X10	24	1600	40	2.00a
PAL20X10A	P20X10	24	1600	40	2.00a
PAL20X4	P20X4	24	1600	40	2.00a
PAL20X4A	P20X4	24	1600	40	2.00a
PAL20X8	P20X8	24	1600	40	2.00a
PAL20X8A	P20X8	24	1600	40	2.00a
PLX TECH.					
PLX448	PLX448	24	5116	98	3.2b
PLX464	PLX448	24	5116	98	3.2b

DEVICE CODE	DEVICE MNEMONIC	# OF PINS	# OF FUSES	# OF PTERMS	REC. VER. LEVEL
RICOH					
EPL10P8A	P10P8V	20	664	32	2.10a
EPL10P8B	P10P8V	20	664	32	2.10a
EPL12P6A	P12P6V	20	786	32	2.10a
EPL12P6B	P12P6V	20	786	32	2.10a
EPL14P4A	P14P4V	20	908	32	
EPL14P4B	P14P4V	20	908	32	
EPL16P2A	P16P2V	20	1030	32	2.10a
EPL16P2B	P16P2V	20	1030	32	2.10a
EPL16P8B	P16P8V	20	2072	64	2.10a
EPL16RP4B	P16RP4V	20	2072	64	2.10a
EPL16RP6B	P16RP6V	20	2072	64	2.10a
EPL16RP8B	P16RP8V	20	2072	64	2.10a
SAMSUNG					
CPL16L8	P16L8	20	2048	64	1.01a
CPL16L8L	P16L8	20	2048	64	1.01a
CPL16R4	P16R4	20	2048	64	1.01a
CPL16R4L	P16R4	20	2048	64	1.01a
CPL16R6	P16R6	20	2048	64	1.01a
CPL16R6L	P16R6	20	2048	64	1.01a
CPL16R8	P16R8	20	2048	64	1.01a
CPL16R8L	P16R8	20	2048	64	1.01a
CPL20L10	P20L10	24	1600	40	1.01a
CPL20L10L	P20L10	24	1600	40	1.01a
CPL20L8	P20L8	24	2560	64	2.00a
CPL20L8L	P20L8	24	2560	64	2.00a
CPL20R4	P20R4	24	2560	64	2.00a
CPL20R4L	P20R4	24	2560	64	2.00a
CPL20R6	P20R6	24	2560	64	2.00a
CPL20R6L	P20R6	24	2560	64	2.00a
CPL20R8	P20R8	24	2560	64	2.00a
CPL20R8L	P20R8	24	2560	64	2.00a
SEEQ					
20RA10Z	P20RA10	24	3210	80	2.02a
26V12H	P26V12	28	7848	150	3.0
SGS-THOM.					
GAL16V8	G16V8	20	2194	64	4.0a
GAL16V8A	G16V8A	20	2194	64	4.0a
GAL16Z8	G16Z8	24	2195	64	
GAL20V8	G20V8	24	560	20	4.0a
GAL20V8A	G20V8A	24	2706	64	4.0a
GAL6001	G6001	24	8294	75	4.0a
RAL10H8	P10H8	20	320	16	
RAL10L8	P10L8	20	320	16	
RAL10P8	P10P8	20	328	16	
RAL12H6	P12H6	20	384	16	
RAL12L6	P12L6	20	384	16	
RAL12P6	P12P6	20	390	16	
RAL14H4	P14H4	20	448	16	
RAL14H8	G20V8	24	560	20	4.0a
RAL14L4	P14L4	20	448	16	
RAL14L8	P14L8	24	560	20	
RAL14P4	P14P4	20	452	16	
RAL14P8	P14P8	24	568	20	
RAL16H2	P16H2	20	512	16	
RAL16H6	G20V8	24	560	20	4.0a
RAL16H8	P16H8	20	2048	64	
RAL16L2	P16L2	20	512	16	
RAL16L6	P16L6	24	640	20	
RAL16L8	P16L8	20	2048	64	1.01a
RAL16P2	P16P2	20	514	16	
RAL16P6	P16P6	24	646	20	
RAL16P8	P16P8	20	2056	64	

DEVICE CODE	DEVICE MNEMONIC	# OF PINS	# OF FUSES	# OF PTERMS	REC. VER. LEVEL
SGS-THOM.					
RAL16R4	P16R4	20	2048	64	1.01a
RAL16R6	P16R6	20	2048	64	1.01a
RAL16R8	P16R8	20	2048	64	1.01a
RAL16RP4	P16RP4	20	2056	64	
RAL16RP6	P16RP6	20	2056	64	
RAL16RP8	P16RP8	20	2056	64	
RAL18H4	G20V8	24	560	20	4.0a
RAL18L4	P18L4	24	720	20	
RAL18P4	P18P4	24	724	20	
RAL20H2	G20V8	24	560	20	4.0a
RAL20H8	G20V8	24	560	20	4.0a
RAL20L2	P20L2	24	640	16	
RAL20L8	P20L8	24	2560	64	2.00a
RAL20P2	P20P2	24	642	16	
RAL20P8	P20P8	24	2568	64	2.15a
RAL20R4	P20R4	24	2560	64	2.00a
RAL20R6	P20R6	24	2560	64	2.00a
RAL20R8	P20R8	24	2560	64	2.00a
RAL20RP4	P20RP4	24	2568	64	
RAL20RP6	P20RP6	24	2568	64	
RAL20RP8	P20RP8	24	2568	64	
SIGNETICS					
10020EV8	P1020EV8	24	3616	80	3.0a
10H20EV8	P1020EV8	24	3616	80	3.0a
82S100	F100	28	1928	48	2.00a
82S101	F100	28	1928	48	2.00a
82S103	F103	28	297	9	2.00a
82S105	F105	28	3553	48	2.00a
82S105A	F105	28	3553	48	2.00a
82S151	F151	20	564	15	2.10a
82S153	F153	20	1842	42	2.15a
82S153A	F153	20	1842	42	2.15a
82S155	F155	20	2108	43	2.00a
82S157	F157	20	2108	43	2.00a
82S159	F159	20	2108	43	2.00a
82S161	F161	24	1544	48	2.00a
82S162	F162	24	165	5	2.00a
82S163	F163	24	225	9	2.00a
82S167	F167	24	3361	48	2.00a
82S167A	F167	24	3361	48	2.00a
82S168	F168	24	3553	48	2.10a
82S173	F173	24	2178	42	2.10a
82S179	F179	24	2452	43	3.0a
PHD16N8	P16N8	20	512	16	2.50a
PHD48N22	F48N22	68	7008	73	4.0a
PLC153	F153	20	1842	42	2.15a
PLC16V8	F16V8	20	2617	72	3.2b
PLC18V8Z	F18V8Z	20	2689	72	4.0a
PLC20V8	F20V8	24	3193	72	4.0a
PLC415	F415	28	5751	68	4.0a
PLC42VA12	F42VA12	24	8994	10	3.2b
PLC473	F473	24	1499	24	2.15a
PLHS153	F153	20	1842	42	2.15a
PLHS16L8A	P16L8	20	2048	64	1.01a
PLHS16L8B	P16L8	20	2048	64	1.01a
PLHS18P8A	P18P8	20	2600	72	2.01a
PLHS18P8B	P18P8	20	2600	72	2.01a
PLHS473	F473	24	1499	24	2.15a
PLHS501	F501	52	15780	112	3.2b
PLHS502	F502	64	23464	144	3.2a
PLS100	F100	28	1928	48	2.00a
PLS101	F100	28	1928	48	2.00a
PLS103	F103	28	297	9	2.00a
PLS105	F105	28	3553	48	2.00a

DEVICE CODE	DEVICE MNEMONIC	# OF PINS	# OF FUSES	# OF PTERMS	REC. VER. LEVEL
SIGNETICS					
PLS105A	F105	28	3553	48	2.00a
PLS151	F151	20	564	15	2.10a
PLS152	F153	20	1842	42	2.15a
PLS153	F153	20	1842	42	2.15a
PLS153A	F153	20	1842	42	2.15a
PLS155	F155	20	2108	43	2.00a
PLS157	F157	20	2108	43	2.00a
PLS159	F159	20	2108	43	2.00a
PLS159A	F159	20	2108	43	2.00a
PLS161	F161	24	1544	48	2.00a
PLS162	F162	24	165	5	2.00a
PLS163	F163	24	225	9	2.00a
PLS167	F167	24	3361	48	2.00a
PLS167A	F167	24	3361	48	2.00a
PLS168	F168	24	3553	48	2.10a
PLS168A	F168	24	3553	48	2.10a
PLS173	F173	24	2178	42	2.10a
PLS179	F179	24	2452	43	3.0a
PLUS105	F105	28	3553	48	2.00a
PLUS153B/D	F153	20	1842	42	2.15a
PLUS16L8D/-7	P16L8	20	2048	64	1.01a
PLUS16R4D/-7	P16R4	20	2048	64	1.01a
PLUS16R6D/-7	P16R6	20	2048	64	1.01a
PLUS16R8D/-7	P16R8	20	2048	64	1.01a
PLUS173B/D	F173	24	2178	42	2.10a
PLUS20L8D/-7	P20L8	24	2560	64	2.00a
PLUS20R4D/-7	P20R4	24	2560	64	2.00a
PLUS20R6D/-7	P20R6	24	2560	64	2.00a
PLUS20R8D/-7	P20R8	24	2560	64	2.00a
PLUS405	F405	28	5410	64	3.0a
SPRAGUE					
SPL16LC8	P16L8	20	2048	64	1.01a
SPL16RC4	P16R4	20	2048	64	1.01a
SPL16RC6	P16R6	20	2048	64	1.01a
SPL16RC8	P16R8	20	2048	64	1.01a
SPL20LC8	P20L8	24	2560	64	2.00a
SPL20XC8	P20X8	24	1600	40	2.00a
TI					
EP1810	EP1800	68	42490	480	4.0a
EP330	EP320	20	2916	72	2.15a
EP610	EP600	24	6482	160	3.0a
EP630	EP600	24	6482	160	3.0a
EP910	EP900	40	17402	240	3.2b
N82S105A	F105	28	3553	48	2.00a
N82S167A	F167	24	3361	48	2.00a
PAL16L8A /-2	P16L8	20	2048	64	1.01a
PAL16R4A /-2	P16R4	20	2048	64	1.01a
PAL16R6A /-2	P16R6	20	2048	64	1.01a
PAL16R8A /-2	P16R8	20	2048	64	1.01a
PAL20L8A	P20L8	24	2560	64	2.00a
PAL20R4A	P20R4	24	2560	64	2.00a
PAL20R6A	P20R6	24	2560	64	2.00a
PAL20R8A	P20R8	24	2560	64	2.00a
TIB82S105A	F105	28	3553	48	2.00a
TIB82S105B	F105	28	3553	48	2.00a
TIB82S167B	F167	24	3361	48	2.00a
TIBFPGA529	F529	20	128	8	2.10a
TIBPAD16N8-7.5	P16N8	20	512	16	2.50a
TIBPAD18N8-6	P18N8	20	304	8	3.0
TIBPAL16H8	P16H8	20	2048	64	
TIBPAL16HD8	P16HD8	20	2048	64	
TIBPAL16L8-10	P16L8	20	2048	64	1.01a
TIBPAL16L8-12/15/25	P16L8	20	2048	64	1.01a

DEVICE CODE	DEVICE MNEMONIC	# OF PINS	# OF FUSES	# OF PTERMS	REC. VER. LEVEL
TI					
TIBPAL16L8-5/7	P16L8	20	2048	64	1.01a
TIBPAL16LD8	P16LD8	20	2048	64	
TIBPAL16O2	P16L8	20	2048	64	1.01a
TIBPAL16R4-10	P16R4	20	2048	64	1.01a
TIBPAL16R4-12/15/25	P16R4	20	2048	64	1.01a
TIBPAL16R4-5/7	P16R4	20	2048	64	1.01a
TIBPAL16R6-10	P16R6	20	2048	64	1.01a
TIBPAL16R6-12/15/25	P16R6	20	2048	64	1.01a
TIBPAL16R6-5/7	P16R6	20	2048	64	1.01a
TIBPAL16R8-10	P16R8	20	2048	64	1.01a
TIBPAL16R8-12/15/25	P16R8	20	2048	64	1.01a
TIBPAL16R8-5/7	P16R8	20	2048	64	1.01a
TIBPAL20L10	P20L10	24	1600	40	1.01a
TIBPAL20L8-15/25	P20L8	24	2560	64	2.00a
TIBPAL20R4-15/25	P20R4	24	2560	64	2.00a
TIBPAL20R6-15/25	P20R6	24	2560	64	2.00a
TIBPAL20R8-15/25	P20R8	24	2560	64	2.00a
TIBPAL20RSP4	P20RSP4	24	2688	64	2.11a
TIBPAL20RSP6	P20RSP6	24	2692	64	2.11a
TIBPAL20RSP8	P20RSP8	24	2696	64	2.11a
TIBPAL20SP8	P20SP8	24	2680	64	
TIBPAL20X10	P20X10	24	1600	40	2.00a
TIBPAL20X4	P20X4	24	1600	40	2.00a
TIBPAL20X8	P20X8	24	1600	40	2.00a
TIBPAL22V10 /A	P22V10	24	5828	132	2.11a
TIBPAL22VP10	P22VP10	24	5828	132	2.11a
TIBPALR19L8	P19L8R	24	2443	64	2.10a
TIBPALR19R4	P19R4R	24	2443	64	2.10a
TIBPALR19R6	P19R6R	24	2443	64	2.10a
TIBPALR19R8	P19R8R	24	2443	64	2.10a
TIBPALT19L8	P19L8T	24	2443	64	2.11a
TIBPALT19R4	P19R4T	24	2443	64	2.11a
TIBPALT19R6	P19R6T	24	2443	64	2.11a
TIBPALT19R8	P19R8T	24	2443	64	2.11a
TIBPLS506A	F506	24	10680	98	2.15a
TIBPSG507A	F507	24	7370	80	2.15a
TICPAL16L8-55	P16L8	20	2048	64	1.01a
TICPAL16R4-55	P16R4	20	2048	64	1.01a
TICPAL16R6-55	P16R6	20	2048	64	1.01a
TICPAL16R8-55	P16R8	20	2048	64	1.01a
TICPAL16RSP4	P16RSP4	20	2176	64	2.11b
TICPAL16RSP6	P16RSP6	20	2180	64	2.11b
TICPAL16RSP8	P16RSP8	20	2184	64	2.11b
TICPAL16SP8	P16SP8	20	2168	64	2.11b
TICPAL18V8	P18V8	20	2752	74	3.0
TICPAL22V10	P22V10	24	5828	132	2.11a
TIEPAL10016ET6	P1016ET6	24	1542	48	3.2a
TIEPAL10016P8-6	P1016P8	24	2056	64	2.10a
TIEPAL10016TE6	P1016ET6	24	1542	48	3.2a
TIEPAL10H16ET6	P1016ET6	24	1542	48	3.2a
TIEPAL10H16P8-6	P1016P8	24	2056	64	2.10a
TIEPAL10H16TE6	P1016ET6	24	1542	48	3.2a
TIFPLA839	F839	24	1094	32	2.00a
TIFPLA840	F839	24	1094	32	2.00a
TOSHIBA					
TC9800P	F9800	20	1830	45	3.2a
TC9801P	F9800	20	1830	45	3.2a
VLSI					
VP10P8	P10P8	20	328	16	
VP12P6	P12P6	20	390	16	
VP14P4	P14P4	20	452	16	
VP16P2	P16P2	20	514	16	
VP16P8	P16P8	20	2056	64	

DEVICE CODE	DEVICE MNEMONIC	# OF PINS	# OF FUSES	# OF PTERMS	REC. VER. LEVEL
VLSI					
VP16RP4	P16RP4	20	2056	64	
VP16RP6	P16RP6	20	2056	64	
VP16RP8	P16RP8	20	2056	64	
VP16V8E	G16V8	20	2194	64	4.0a
VP20V8E	G20V8	24	560	20	4.0a

SUPPORT NOTES:

XILINX LCA devices are supported by CUPL in conjunction with XACT.

DEVICE CODE	DEVICE MNEMONIC	NO. OF PINS	ARRAY SIZE
AMD/MMI			
53/6300	RA8P4	16	256 X 4
53/6301	RA8P4	16	256 X 4
53/6305	RA9P4	16	512 X 4
53/6306	RA9P4	16	512 X 4
53/6308	RA8P8	20	256 X 8
53/6309	RA8P8	20	256 X 8
53/6330	RA5P8	16	32 X 8
53/6331	RA5P8	16	32 X 8
53/6348	RA9P8	20	512 X 8
53/6349	RA9P8	20	512 X 8
53/6352	RA10P4	18	1K X 4
53/6353	RA10P4	18	1K X 4
53/6380	RA10P8	24	1K X 8
53/6380JS	RA10P8	24	1K X 8
53/6380S	RA10P8	24	1K X 8
53/6381	RA10P8	24	1K X 8
53/6381JS	RA10P8	24	1K X 8
53/6381S	RA10P8	24	1K X 8
53/6388	RA11P4	18	2K X 4
53/6389	RA11P4	18	2K X 4
53/63S1641	RA12P4	20	4K X 4
53/63S1641A	RA12P4	20	4K X 4
53/63S1681	RA11P8	24	2K X 8
53/63S1681A	RA11P8	24	2K X 8
53/63S3281	RA12P8	24	4K X 8
53/63S3281A	RA12P8	24	4K X 8
AM27S12	RA9P4	16	512 X 4
AM27S13	RA9P4	16	512 X 4
AM27S18	RA5P8	16	32 X 8
AM27S180	RA10P8	24	1K X 8
AM27S181	RA10P8	24	1K X 8
AM27S184	RA11P4	18	2K X 4
AM27S185	RA11P4	18	2K X 4
AM27S19	RA5P8	16	32 X 8
AM27S190	RA11P8	24	2K X 8
AM27S191	RA11P8	24	2K X 8
AM27S20	RA8P4	16	256 X 4
AM27S21	RA8P4	16	256 X 4
AM27S28	RA9P8	20	512 X 8
AM27S29	RA9P8	20	512 X 8
AM27S32	RA10P4	18	1K X 4
AM27S33	RA10P4	18	1K X 4
AM27S37	RA10P8	24	1K X 8
AM27S40	RA12P4	20	4K X 4
AM27S41	RA12P4	20	4K X 4
AM27S43	RA12P8	24	4K X 8
AM27S49	RA13P8	24	8K X 8
CYPRESS			
CY7C264	RA13P8	24	8K X 8
CY7C282	RA10P8	24	1K X 8
CY7C292	RA11P8	24	2K X 8
HARRIS			
HM7602	RA5P8	16	32 X 8
HM7603	RA5P8	16	32 X 8
HM7610	RA8P4	16	256 X 4
HM7610A	RA8P4	16	256 X 4
HM7610B	RA8P4	16	256 X 4
HM7611	RA8P4	16	256 X 4
HM7611A	RA8P4	16	256 X 4
HM7611B	RA8P4	16	256 X 4
HM76160	RA11P8	24	2K X 8
HM76161	RA11P8	24	2K X 8
HM76161A	RA11P8	24	2K X 8

DEVICE CODE	DEVICE MNEMONIC	NO. OF PINS	ARRAY SIZE
HARRIS			
HM76161B	RA11P8	24	2K X 8
HM76164	RA12P4	20	4K X 4
HM76165	RA12P4	20	4K X 4
HM7620	RA9P4	16	512 X 4
HM7620A	RA9P4	16	512 X 4
HM7620B	RA9P4	16	512 X 4
HM7621	RA9P4	16	512 X 4
HM7621A	RA9P4	16	512 X 4
HM7621B	RA9P4	16	512 X 4
HM76321	RA12P8	24	4K X 8
HM7642	RA10P4	18	1K X 4
HM7642A	RA10P4	18	1K X 4
HM7642B	RA10P4	18	1K X 4
HM7643	RA10P4	18	1K X 4
HM7643A	RA10P4	18	1K X 4
HM7643B	RA10P4	18	1K X 4
HM7648	RA9P8	20	512 X 8
HM7649	RA9P8	20	512 X 8
HM7649A	RA9P8	20	512 X 8
HM76641	RA13P8	24	8K X 8
HM76641A	RA13P8	24	8K X 8
HM7680	RA10P8	24	1K X 8
HM7680P	RA10P8	24	1K X 8
HM7680R	RA10P8	24	1K X 8
HM7680RP	RA10P8	24	1K X 8
HM7681	RA10P8	24	1K X 8
HM7681A	RA10P8	24	1K X 8
HM7681P	RA10P8	24	1K X 8
HM7681R	RA10P8	24	1K X 8
HM7681RP	RA10P8	24	1K X 8
HM7684	RA11P4	18	2K X 4
HM7684P	RA11P4	18	2K X 4
HM7685	RA11P4	18	2K X 4
HM7685A	RA11P4	18	2K X 4
HM7685P	RA11P4	18	2K X 4
NATIONAL			
DM74LS471	RA8P8	20	256 X 8
DM74S188	RA5P8	16	32 X 8
DM74S287	RA8P4	16	256 X 4
DM74S288	RA5P8	16	32 X 8
DM74S387	RA8P4	16	256 X 4
DM74S472	RA9P8	20	512 X 8
DM74S472A	RA9P8	20	512 X 8
DM74S472B	RA9P8	20	512 X 8
DM74S473	RA9P8	20	512 X 8
DM74S473A	RA9P8	20	512 X 8
DM74S570	RA9P4	16	512 X 4
DM74S570A	RA9P4	16	512 X 4
DM74S571	RA9P4	16	512 X 4
DM74S571A	RA9P4	16	512 X 4
DM74S571B	RA9P4	16	512 X 4
DM74S572	RA10P4	18	1K X 4
DM74S572A	RA10P4	18	1K X 4
DM74S573	RA10P4	18	1K X 4
DM74S573A	RA10P4	18	1K X 4
DM74S573B	RA10P4	18	1K X 4
DM87S180	RA10P8	24	1K X 8
DM87S181	RA10P8	24	1K X 8
DM87S181A	RA10P8	24	1K X 8
DM87S184	RA11P4	18	2K X 4
DM87S185	RA11P4	18	2K X 4
DM87S185A	RA11P4	18	2K X 4
DM87S185B	RA11P4	18	2K X 4
DM87S190	RA11P8	24	2K X 8

DEVICE CODE	DEVICE MNEMONIC	NO. OF PINS	ARRAY SIZE
NATIONAL			
DM87S190A	RA11P8	24	2K X 8
DM87S190B	RA11P8	24	2K X 8
DM87S191	RA11P8	24	2K X 8
DM87S191A	RA11P8	24	2K X 8
DM87S191B	RA11P8	24	2K X 8
DM87S195	RA12P4	20	4K X 4
DM87S195A	RA12P4	20	4K X 4
DM87S195B	RA12P4	20	4K X 4
DM87S321	RA12P8	24	4K X 8
DM87S321A	RA12P8	24	4K X 8
SIGNETICS			
82S123	RA5P8	16	32 X 8
82S123A	RA5P8	16	32 X 8
82S126	RA8P4	16	256 X 4
82S126A	RA8P4	16	256 X 8
82S129	RA8P4	16	256 X 4
82S129A	RA8P4	16	256 X 8
82S130	RA9P4	16	512 X 4
82S130A	RA9P4	16	512 X 4
82S131	RA9P4	16	512 X 4
82S131A	RA9P4	16	512 X 4
82S135	RA8P8	20	256 X 8
82S137	RA10P4	18	1K X 4
82S137A	RA10P4	18	1K X 4
82S137B	RA10P4	18	1K X 4
82S147	RA9P8	20	512 X 8
82S147A	RA9P8	20	512 X 8
82S180	RA10P8	24	1K X 8
82S181	RA10P8	24	1K X 8
82S181A	RA10P8	24	1K X 8
82S181B	RA10P8	24	1K X 8
82S184	RA11P4	18	2K X 4
82S185	RA11P4	18	2K X 4
82S185A	RA11P4	18	2K X 4
82S185B	RA11P4	18	2K X 4
82S191	RA11P8	24	2K X 8
82S191A	RA11P8	24	2K X 8
82S191B	RA11P8	24	2K X 8
82S195	RA12P4	20	4K X 4
82S23	RA5P8	16	32 X 8
82S23A	RA5P8	16	32 X 8
82S321	RA12P8	24	4K X 8
82S641	RA13P8	24	8K X 8
TI			
TBP18S030	RA5P8	16	32 X 8
TBP18S22	RA8P8	20	256 X 8
TBP18SA030	RA5P8	16	32 X 8
TBP18SA22	RA8P8	20	256 X 8
TBP24S10	RA8P4	16	256 X 4
TBP24S166	RA12P8	24	4K X 4
TBP24S41	RA10P4	18	1K X 4
TBP24S81	RA11P4	18	2K X 4
TBP24SA10	RA8P4	16	256 X 4
TBP24SA166	RA11P8	24	4K X 4
TBP24SA41	RA10P4	18	1K X 4
TBP24SA81	RA11P4	18	2K X 4
TBP28S166	RA11P8	24	2K X 8
TBP28S42	RA9P8	20	512 X 8
TBP28S86A	RA10P8	24	1K X 8
TBP28SA166	RA11P8	24	2K X 8
TBP28SA42	RA9P8	20	512 X 8
TBP28SA86A	RA10P8	24	1K X 8