PLXMon

User's Guide

Version 1.0

January 31, 1997

Product Sales: 1-800-759-3735
Fax: 1-408-774-2169
Email: info@plxtech.com
Web and FTP Site: //www.plxtech.com

Table of Contents

Chapter 1

Chapter 2

Chapter 3

Chapter 4

Chapter 5

Preface

This guide provides all the information you need to use the PLXMon program.

Document Organization

This guide is organized into the following chapters for easy reference:

- Chapter 1, “Introduction to PLXMon,” introduces the PLXMon program.
- Chapter 2, “Getting Started with PLXMon,” discusses how to start using the PLXMon and PLXMon95 programs.
- Chapter 3, “PLXMon Basics,” discusses the basic features of PLXMon.
- Chapter 4, “PLXMon Command Set,” discusses the PLXMon command set, by category.
- Chapter 5, “Examples,” provides examples of how to use the PLXMon commands.

Style Conventions

The following styles are used in this manual:

- Commands, file names, keyboard keys, text you manually enter, and URL paths are in `Courier type` (for example, `DEBUG.EXE` or `http://www.plxtech.com`).
- Variables and parameters are in ***`Courier italic type`*** (for example, ***`hbuf`*** or ***`x`***).
- Emphasized text, names of diskettes, and names of publications are in *italic type* (for example, *Read these notes!*).
- Screen references are in **Arial Bold type** (for example, “Choose **OK**”).

Contacting PLX

We want to hear from you! If you have comments, corrections or suggestions, please let us know.

PLX Technology, Inc.
390 Potrero Ave
Sunnyvale, CA 94086

Phone	(408) 774-9060
FAX	(408) 774-2169
Email	apps@plxtech.com
Website/FTP site	http://www.plxtech.com

Chapter 1

Introduction to PLXMon

This chapter introduces the PLXMon program.

PLXMon, and its Windows 95 variation, PLXMon95, are powerful, user-interactive programs designed for engineers working with the PCI bus, PCI devices, the PLX family of PCI devices, memory, and I/O.

PLXMon and PLXMon95 are identical except for low-level differences demanded by the different operating system architectures. This document uses the name “PLXMon” to describe both PLXMon and PLXMon95. Any differences between the programs are clearly labeled.

PLXMon uses a simple, yet versatile, command-line architecture.

General PCI, I/O, and memory manipulation commands include the following:

- Select any PCI device on any PCI bus
- Examine and modify PCI Configuration Registers of a device
- Display, modify, copy, fill, compare and search memory (flat 32-bit addressing)
- Input and output (16-bit addressing)

and more.

PLX device family commands include the following:

- Examine and modify Local Configuration Registers
- Examine and modify Run-Time Registers
- Examine and modify Local DMA Registers (PLX PCI 9060 Rev3 and PLX PCI 9060SD)
- Program chained and nonchained DMA (PLX PCI 9060 Rev3 and PLX PCI 9060SD)
- Read and write EEPROM

and more.

Commands are generally short and mnemonic, and one line can contain multiple commands.

Command lines can be repeated indefinitely or for a user-specified number of iterations, using the unique repeat command in PLXMon. Sections of a command line can be repeated—even sections within sections!

PLXMon supports user-defined macros, user-defined variables, and an extensive set of expression evaluation operators. Macros and variables can be saved and restored on files. These are text files that can be examined and modified with any text editor.

Who Should Use PLXMon?

PLXMon is a PCI engineering tool. People that are becoming familiar with PCI, as well as experienced PCI engineers, will find PLXMon to be a helpful program.

Engineers using the PLX family of PCI interface products will enjoy the PLXMon commands that are made specifically for PLX chips and evaluation boards.

PLXMon has proved to be useful in tracking down PCI-related problems.

The PLXMon syntax, however, may be too terse to be useful as a marketing or sales tool.

Previous Incarnation of PLXMon—`HOST.EXE`

The most recent predecessor of PLXMon is `HOST.EXE`. PLXMon has the same “look and feel” of Host, but there are major differences.

Host is the companion software for the PLX PCI 9060 Evaluation Board (PCI 9060EB). It has special software to communicate with the Intel I960 processor of the PCI 9060EB by way of the PCI 9060 mailboxes.

In contrast, PLXMon is a general-purpose PCI tool designed for use with PLX and other manufacturer’s PCI devices. PLXMon makes few assumptions about the local side of a PCI device so it can be easily applied to your PCI boards.

Similarities between PLXMon and `DEBUG.EXE`

If you have used DOS `DEBUG.EXE` to manipulate memory and I/O, you will quickly feel comfortable with PLXMon. While PLXMon may have the “look and feel” of the Debug memory and I/O commands, it also has extensions for word and long word data values (refer to the sections, “Memory Manipulation” and “Input and Output,” for additional information).

PLXMon does not have any assembler/disassembler or breakpoint commands.

“Versions” of PLXMon

PLXMon is continually being enhanced and improved to make it work with all PLX PCI products and for all PLXMon users. As a result, there are many versions of PLXMon in existence. You should be aware that two different versions may appear to act the same, but they could be doing subtly different things. Most of the time, however, newer versions simply include new commands, user enhancements, or bug fixes. Most commands, such as memory and I/O manipulation commands, are stable and will not change.

PLXMon prints a date-coded version number when it first starts. Use this number to compare different versions. In general, you will want to use the most recent version.

Be aware that any documentation (including this document) could easily be out-of-date and misleading! The most up-to-date (yet brief) documentation is in the PLXMon online help (refer to the section, “`help, h, ?`,” for additional information).

PLXMon License Agreement

The PLXMon LICENSE AGREEMENT is built into the program. You should review the license agreement by running the `license` command (refer to the section, “View PLXMon License Agreement: `license`,” for additional information).

By using PLXMon, you indicate that you agree to all terms of the agreement.

DO NOT USE THE PLXMon PROGRAM UNLESS YOU AGREE TO, AND ABIDE BY, ALL TERMS OF THE LICENSE AGREEMENT.

Programmers’ Statement

The PLXMon programmers chose to apply KISS (Keep It Simple, Stupid) principles to this program. Rather than making PLXMon “smart” and full of prohibitions, warnings and error messages, it is (hopefully) “stupid.” That is, if you do something nonsensical, such as writing to a read-only register, PLXMon will obediently carry out your wishes.

With the KISS style of programming, you can verify things that do not work, as well as things that do. Your system may crash, but our philosophy is, “You asked for it!”

Late Breaking News

Be sure to read any addendums and errata to PLXMon. These documents discuss important changes, corrections and improvements to the program. The PLX website and FTP site may contain important late breaking news as well.

Chapter 2

Getting Started with PLXMon

This chapter discusses how to start using the PLXMon and PLXMon95 programs.

Running PLXMon (DOS)

The DOS version of PLXMon runs on a PC running DOS with a 386 (or higher).

♦ To run PLXMon for the first time:

1. Do a "clean" DOS boot with a "clean" `CONFIG.SYS` and `AUTOEXEC.BAT`.

Note: This step may not be necessary, but it helps if you are having any problems. It helps a lot, especially if PLXMon appears to be partially functional. PLXMon uses a DPMI that may conflict with other programs (such as Windows, `EMM386.EXE`, and others).

2. If your PLXMon package is zipped, unzip it into an empty directory.

PLXMon is a single executable DOS file, `PLXMON.EXE`. It does not require any other files to make it run.

3. Type `PLXMON` at the DOS prompt, then press `ENTER`.

Note: PLXMon does not acknowledge any parameters from the DOS command line.

PLXMon starts up, and is now ready for your input. Read Chapter 3, "PLXMon Basics," to learn how to use PLXMon to its full potential.

Running PLXMon95 (Windows 95)

The Windows version of PLXMon, PLXMon95, runs on a PC running Windows 95.

◆ **To run PLXMon for the first time:**

1. If your PLXMon95 package is zipped, unzip it into an empty directory.

The package should contain the executable console application `PLXMON95.EXE` and the driver `PHYSACC.VXD`.

2. Start PLXMon95, using your choice of methods (such as finding the file with the Windows Explorer and double-clicking the PLXMon95 icon).

PLXMon95 starts up, and it is ready for your input. Read Chapter 3, “PLXMon Basics,” to learn how to use PLXMon95 to its full potential.

Chapter 3

PLXMon Basics

This chapter discusses the basic features of PLXMon.

Command Line

The PLXMon command line is similar to DOS, DEBUG and other command line programs—you type in commands, then press `ENTER`.

Commands are case-sensitive.

One difference between the PLXMon command line and command lines in other programs is that you can enter as many commands as you want (up to 80 characters) on one line. If there is any ambiguity between commands and parameters, enter a semicolon (`;`) between the commands.

An ambiguity may arise when a numeric parameter could be confused with a command. In hexadecimal, a digit includes characters "0" through "f". A numeric parameter, such "*`db`*," could be confused with the `db` "display bytes" command.

For example, to display a block of memory at 100, then input from port 200, you can enter

```
db 100;i 200
```

The semicolon explicitly tells PLXMon that “`i 200`” is a new command and not a numeric parameter for the display command, which can accept zero, one or two numeric parameters (refer to the section, “Display: `d, db, dw, dl, dd`,” for additional information). In this case, however, the semicolon is not necessary because “`i`” cannot be interpreted as a number. Therefore, entering

```
db 100 i 200
```

would be interpreted exactly the same as the first example.

The semicolon is required in this example:

```
db 100; db 200
```

because the second “`db`” could be interpreted as a numeric parameter for the first command.

Scrolling and Editing Command Lines

PLXMon allows you to scroll to previous command lines you have entered, the same as you do with the standard DOS `DOSKEY` and `NDOSEDIT` commands. To scroll, use the up (🡅) and down (🡇) arrow keys.

You can edit any line by using the left (🡄) and right (🡆) arrow keys, `BACKSPACE`, and `ESC`. PLXMon defaults to Overwrite mode. Press the `INSERT` key to toggle between Overwrite and Insert modes.

User-Defined Variables

User-defined variables can be used as simple substitutes for obscure numbers, or they can be combined with expressions, numbers, and commands to create complex and powerful command sequences.

Create (or modify) a variable by typing a *variable name*, followed by the equals sign (=) and a *value*. For example, the value can be any legal combination of numbers, variables or expressions:

```
x = 1

x = x+1
```

You can also eliminate a variable by leaving the right-hand side empty:

```
x =
```

You can use virtually any name of any length for your variables.

Note: Avoid duplicate variable names. Also, be aware that commands and macros have priority over variables.

To view all variables, use the `vars` command (refer to the section, “Show Variables: `vars`,” for additional information).

The following example illustrates the usefulness of variables. The example outputs an incrementing value to a 16-bit port:

```
x = 0 [ow 100 x; x = x + 1; r 10]
```

(This example gets ahead of itself by showing commands and concepts that are covered in the remainder of this guide; however, it is important to see what variables can do!)

Refer to the section, “Expressions,” for additional information about expressions.

Expressions

Expressions are legal (and intuitive) arrangements of numbers, variables and operators (refer to the section, “User-Defined Variables,” for information about variables). PLXMon enables you to work with expressions “on-the-fly”—anywhere you enter numbers, you can enter expressions.

Expressions can even be evaluated—without a command—right at the command line!

Expressions are evaluated left to right, but can be overridden with parenthesis.

The PLXMon expression operators include monadic and dyadic operators, as described in the following sections.

Monadic Operators

Monadic operators include one’s complement, two’s complement, and so forth, as listed in Table Chapter 3 -1:

Table Chapter 3 -1. Monadic Operators (where N represents a number, expression or variable)

Syntax	Operation	Example (Hex)	Result
~N	Inverse of N	~1	Returns 0xfffffffe (One’s complement)
-N	0 minus N	-1	Returns 0xffffffff (Two’s complement)
+N	0 plus N	+1	Returns 0x00000001 (does not do anything!)
*N	Dereference Pointer N	*12345678	Returns the value pointed to by 12345678

Dyadic Operators

Dyadic operators include addition, subtraction, and so forth, as listed in Table Chapter 3 -2:

Table Chapter 3 -2. Dyadic Operators (where N and M represent numbers, expressions or variables)

Syntax	Operation	Example (Hex)	Result
N+M	N plus M	4+1	Returns 0x00000005
N-M	N minus M	4-1	Returns 0x00000003
N*M	N multiplied by M	7*3	Returns 0x00000015
N/M	N divided by M	7/3	Returns 0x00000002
N%M	N modulo M	7%3	Returns 0x00000001
N&M	N AND M	7&3	Returns 0x00000003
N|M	N OR M	7|3	Returns 0x00000007
N^M	N XOR M	7^3	Returns 0x00000004

Macros

Macros enable you to create a complex sequence of commands, and call it by a name specified by you. To execute the complex sequence of commands, use the macro name instead. Macros are convenient if you find yourself frequently using the same complex command sequence.

Refer to the section, “Define a Macro: `define`,” for information about creating and using macros.

Macros, as well as variables, can be saved to a file, then restored in a future session. Refer to the sections, “`save`” and “`read`,” for information about the `save` and `read` commands.

Ranges

Several PLXMon commands require a full or partial range to be specified. The range includes a starting address and a byte count. For example, the byte count can be implied with an ending address or explicit:

```
db 4000 4020
```

which implies a count of 20 (4020-4000); whereas

```
db 4000 l 20
```

explicitly specifies a count of 20.

Note: “`l`” must be lowercase.

Both examples yield identical results.

Commands that accept partial ranges use the starting address you type and substitute a default count. Some commands, such as the display commands, substitute both the starting address and count if you do not provide them.

Repeating Command Lines (Looping)

A unique feature of PLXMon is the repeat (`r`) command. The `r` command enables you to repeat all or part of a command line for a number of iterations you specify, or indefinitely. Further, you can nest your repeat loops by using nesting tokens, “`[`” and “`]`” (refer to the section, “Repeat: `r`,” for additional information).

Online Help

PLXMon provides built-in help messages for every command (refer to the section, “`help, h, ?`,” for additional information).

PLXMon Batch Files

PLXMon can read batch files, which are text files that list several consecutive lines of PLXMon commands. Use the `read` command to submit the file for PLXMon to execute (refer to the section, "`read`," for additional information).

It is recommended that you add descriptive comments to your batch file, as it may resemble gibberish to someone else. Use the PLXMon `echo` command to add comments (refer to the section, "`echo`," for additional information).

MON Environment Variable

If you find yourself initializing your system with the same set of commands at the start of every PLXMon session, consider using the PLXMon DOS environment variable.

If this variable is set when PLXMon starts up, PLXMon reads the text string following the variable as its first command line. The PLXMon environment variable name is ***`MON`***. You can set up the "`set MON=`" line in your `AUTOEXEC.BAT` file, as in

```
set MON=ow 100 abcd; ow 102 1234
```

If a single PLXMon command line is not enough for you, consider putting your command lines in a text file (that is, a batch file, as described in the section, "PLXMon Batch Files"), then you can apply the `read` command in the environment variable. For example:

```
set MON=read MYFILE.MON
```

Refer to the section, "`read`," for information about the `read` command.

When PLXMon Starts…

PLXMon takes the following actions when it starts. Differences between PLXMon and PLXMon95 are specifically indicated.

1. **PLXMon (DOS)**—Rational Systems DPMI sets up to run in Protected mode, and displays its sign-on banner and a reference to the license agreement command.

PLXMon95 (Windows 95)—Displays its sign-on banner and a reference to the license agreement command.

2. Displays release notes, if any exist. *Read these notes!* They may contain late-breaking changes to the program.
3. **PLXMon (DOS)**—Checks for PCI BIOS. Displays a message if there is a problem.

4. Internally registers all PCI devices on all PCI buses. Displays a message if no PCI devices are found.

This step has a command-line equivalent, `regdevs` (refer to the section, “Register PCI Devices: `regdevs`,” for additional information).

5. Selects a PCI device. Searches the registered devices for the PLX vendor ID (0x10b5). Selects the first PLX device found, if any exist. User-defined variables *`s0`*, *`s1`*, *`s2`*, and *`s3`* are set to the values found in PCI Configuration Registers 0x18, 0x1c, 0x20, and 0x24 of the selected device, respectively. For PLX devices, these variables refer to PCI base addresses for the local address spaces of the device. This step has a command-line equivalent, `dev` (refer to the section, “Show or Select a Device: `dev`,” for additional information).
6. Allocates a 64 KB buffer for you to “play” in. Sets a user-defined variable, *`hbuf`*, that refers to the buffer.
7. ***PLX PCI 9060 Rev 3 only:*** *This step is important for accessing the DMA registers from the PCI side.* Sets a user-defined variable, *`regbase`*, that refers to the memory-mapped address of the Local Configuration Registers on the local side of the PLX PCI 9060EB. If you are using a PLX PCI 9060 Rev 3 on your own board, and you want to access DMA registers from the PCI side, you may need to change the value of this variable to match your hardware (refer to the section, “User-Defined Variables,” for additional information).
8. ***PLX PCI 9060 Rev 3 and PLX PCI 9060SD only:*** *This step is important for programming chained DMA transfers from the PCI side.* Sets a user-defined variable, *`membase`*, that refers to an address in the local memory of the PLX PCI 9060EB that is available to load chained DMA descriptors. If you are using a PLX PCI 9060 Rev 3 or a PLX PCI 9060SD on your own board, and you want to load chained DMA descriptors from the PCI side, you may need to change the value of this variable to match the local memory map of your board.
9. Displays a list of all devices on all buses, including the Vendor ID, Device ID, and a few selected PCI Configuration Registers of each device. The currently selected device is marked with an asterisk (“`*`”).
10. If the environment variable *`MON`* is set, it executes the data following “*`MON=`*” as a command line (refer to the section, “MON Environment Variable,” for additional information).
11. Displays the PLXMon prompt (`&`).
12. PLXMon is ready for user input. Type commands, variables or expressions, then press `ENTER`.

Chapter 4

PLXMon Command Set

This chapter discusses the PLXMon command set, by category.

Parameters listed with an *`expr`* substring, such as *`valexpr`*, can be numbers, variables, or expressions. Refer to the sections, “User-Defined Variables” and “Expressions,” for additional information.

Parameters listed within square braces (“`[`” and “`]`”) are optional parameters; otherwise, the parameter is necessary.

Parameters listed with the vertical bar (`|`) indicate that you must provide *one or the other* parameter, but *not both*, as in

```
command xexpr | yexpr
```

Parameters listed with ellipses (…) indicate that you can extend the command with more parameters of the same type.

```
command expr […]
```

Memory Manipulation

Memory manipulation commands enable you to display, modify, fill, search and compare memory in several intuitive and generally accepted ways. These commands are similar to the memory manipulation commands in DOS `DEBUG.EXE`.

From your perspective, all memory is referenced as a flat, nonsegmented, 4 GB space.

Caution: There are no restrictions on memory access (so you should realize that you can easily crash your system).

Most of the memory manipulation commands accept range parameters. Ranges control the address limits of the command (refer to the section, “Ranges,” for additional information).

Display, enter and move have variations for byte, word and long word sizes. PLXMon uses the appropriate byte, word, or long word assembler instruction (hence, the appropriate bus cycle) for each variation.

The `d` and `e` commands adopt the size (byte, word, or long word) of the most recently used display or enter command (that is, if you just used the `el` command, the `d` command displays long words).

Display: d, db, dw, `dl`, dd

```
d [range]

db [range]

dw [range]

dl [range]

dd [range]
```

Table Chapter 4 -1. Display (d, db, dw, `dl`, dd) Command Input Parameter

Parameter	Description
`range`	Memory address and count (refer to the section, “Ranges”)

This group of commands displays a range of memory. You can display in byte, word, long word, or double format. (Long word is the same as double.)

Display commands accept two, one, or zero parameters:

Table Chapter 4 -2. Display Command Input Parameter Information

No.	Parameter	Description
two	*`range`* (refer to the section, “Ranges”)	Displays the specified memory range
one	*`address only`*	Displays memory starting at the address specified for the default count (0x80 bytes)
zero	—	Displays memory starting *after* the last address for the default count (0x80 bytes)

PLXMon displays the ASCII values of displayed memory in a separate column. This column displays in byte-order, regardless of the display size (byte, word, long word, or double.)

Enter: e, eb, ew, el, ed

```
e addrexpr [INC incexpr] [valexpr …]

eb addrexpr [INC incexpr] [valexpr …]

ew addrexpr [INC incexpr] [valexpr …]

el addrexpr [INC incexpr] [valexpr …]

ed addrexpr [INC incexpr] [valexpr …]
```

Table Chapter 4 -3. Enter (e, eb, ew, el, ed) Command Input Parameters

Parameter	Description
addrexpr	Memory address to start entering
incexpr	Address increment
valexpr	Value to enter

Use the Enter commands to modify—or to interactively examine and modify—memory. You can specify bytes, words, long words or doubles. (Long word is the same as double.)

Enter commands require an *`addrexpr`* parameter. If *`valexpr`* is not specified, PLXMon enters Interactive-Enter mode. Interactive-Enter mode displays the value at the starting address and allows you type in a new value. You then have one of three choices:

1. Type a new value, then press `ENTER` or the `SPACEBAR` (the new value is actually written to memory when you press `ENTER` or the `SPACEBAR`).
2. Press the `SPACEBAR`.
3. Press `ENTER`.

Pressing the `SPACEBAR` continues Interactive-Enter mode at the next address, while pressing `ENTER` ends Interactive-Enter mode.

Pass at least one *`valexpr`* on the command line to have values entered in Noninteractive mode.

Use *`INC incexpr`* when you want to enter values at noncontiguous addresses. For example, you may want to enter new values every 100 bytes, as in

```
e 123400 INC 100 a b c d
```

The *`INC`* specifier must be capitalized, and *`incexpr`* must be supplied. This unique feature can be applied in both Interactive-Enter and Noninteractive-Enter modes.

Fill: `f`

```
f range fillexpr […]
```

Table Chapter 4 -4. Fill (`f`) Command Input Parameters

Parameter	Description
`range`	Fill address and count (refer to the section, "Ranges")
`fillexpr`	Fill values (byte)
…	More fill values (bytes)

You can use the `f` command to prepare a large portion of memory with data patterns. The pattern can be one or more bytes long. The fill values are repeated throughout the entire range.

Move: `m`, `mb`, `mw`, `ml`, `md`

```
m range destexpr

mb range destexpr

mw range destexpr

ml range destexpr

md range destexpr
```

Table Chapter 4 -5. Move (`m`, `mb`, `mw`, `ml`, `md`) Command Input Parameters

Parameter	Description
`range`	Source address and count (refer to the section, "Ranges")
`destexpr`	Destination address

Use Move commands to move blocks of memory from one place to another. You can specify byte, word, long word, or double variations. (Long word is the same as double.)

The byte variations (`m` and `mb`) are appropriate for most needs; the other variations are provided because they perform 16 and 32 bit data transfers at the assembler level (refer to the section, "Memory Manipulation," for additional information).

Unlike the `d` and `e` commands, the `m` command always moves bytes.

Note: Moves use CPU "`MOV`" and other related instructions—DMA is not used; therefore, do not expect bursting on the PCI bus.

Compare: c

```
c range addrexpr
```

Table Chapter 4 -6. Compare (c) Command Input Parameters

Parameter	Description
range	Left address and count (refer to the section, "Ranges")
addrexpr	Right address

This command is useful for checking the result of a move or DMA. It performs a byte-by-byte comparison of data, starting at the left and right address locations. PLXMon displays both addresses and both data bytes of any miscomparisons.

Search: s

```
s range [expr […]] | ["text"]
```

Table Chapter 4 -7. Search (s) Command Input Parameters

Parameter	Description
range	Search address and count (refer to the section, "Ranges")
expr	First byte of search pattern
…	Subsequent bytes of search pattern
"text"	Text pattern being searched

This command looks for a data pattern. You can search for an ASCII string or a data pattern that is one or more bytes in length. The address of any matching pattern within the range is printed.

Text must be typed between quote (`" "`) characters. The search is case-sensitive.

Input and Output

The PLXMon input and output commands provide unrestricted port I/O. Variations for byte, word, or long word port I/O are provided. The assembler instruction used for the port transfer is appropriate for the I/O width (byte, word, or long word) you specify.

Input: `i`, `ib`, `iw`, `il`, `id`

```
i [range]

ib [range]

iw [range]

il [range]

id [range]
```

Table Chapter 4 -8. Input (`i`, `ib`, `iw`, `il`, `id`) Command Input Parameter

Parameter	Description
`range`	Port address and count (refer to the section, “Ranges”)

The Input group of commands reads and displays from a range of port addresses. You can display in byte, word, long word, or double format. (Long word is the same as double.)

Input commands accept two, one, or zero parameters:

Table Chapter 4 -9. Input Command Input Parameter Information

No.	Parameter	Description
two	*`range`* (refer to the section, “Ranges”)	Specified range of ports is read and displayed
one	*`address only`*	Port address specified is read and displayed
zero	—	Port address starting *after* the last port address read is read and displayed

The most popular usage is to simply specify the port address (one parameter), as in

```
iw 100
```

The `i` command adopts the size (byte, word, or long word) of the most recently used input command (that is, if you just used the `il` command, the `i` command reads and displays long words).

PLXMon displays the ASCII values of the port data in a separate column. This column displays in byte-order, regardless of the size specified (byte, word, long word, or double).

Output: `ob`, `ow`, `ol`, `od`

```
ob portexpr valexpr

ow portexpr valexpr

ol portexpr valexpr

od portexpr valexpr
```

Table Chapter 4 -10. Output (ob, ow, ol, od) Command Input Parameters

Parameter	Description
portexpr	Port address
valexpr	Value to be written

Use an Output command to write data to ports. You can write bytes, words, long words, or doubles. (Long word is the same as double.)

PCI Commands

PCI commands apply to all PCI devices—not just PLX PCI devices.

Register PCI Devices: `regdevs`

```
regdevs
```

This command registers and displays all PCI devices found on all PCI buses.

The registration list associates a logical device number with physical PCI devices.

Note: The device registration list only stores the PCI bus number and PCI BIOS device number of a device; the list does not store the register values of a device. As a result, PLXMon cannot reference "stale" copies of registers—they are read "on-the-fly." Some of the PLXMon user-defined variables are an exception (refer to the section, "Show or Select a Device: `dev`," for additional information).

PLXMon (DOS)

The DOS version of PLXMon uses the PCI BIOS to select every PCI device location (ignoring multifunction devices) and reads the Device ID and Vendor ID of the location. If both values read 0xffff, PLXMon assumes there is no device at that location. Otherwise, the device is added to the list of registered devices.

PLXMon95 (Windows 95)

PLXMon95 searches the registry (`HKEY_DYN_DATA\ConfigManager\Enum`) and adds any PCI devices it finds to the list of registered devices.

Show or Select a Device: dev

```
dev [devnum]
```

Table Chapter 4 -11. Show or Select a Device (`dev`) Command Input Parameter

Parameter	Description
`devnum`	Logical device number

If no parameter is specified, this command displays a list of all devices on all buses found in the list of registered devices (refer to the section, “Register PCI Devices: `regdevs`,” for additional information). Each line displays the logical device number, Vendor ID, Device ID, and a few PCI Configuration Registers of the device. The currently selected device is marked with an asterisk (“`*`”).

If *`devnum`* is specified, and it is in the list of registered devices, the associated PCI device is selected. User-defined variables *`s0`*, *`s1`*, *`s2`*, and *`s3`* are set to the values found in the PCI Configuration Registers 0x18, 0x1c, 0x20, and 0x24 of the selected device, respectively. These variables refer to PCI base addresses for the various local address spaces of the device.

The user-defined variables *`s0`*, *`s1`*, *`s2`*, and *`s3`*, combined with the PLXMon (“`+`”) operator, make it easy to access the memory or I/O of the current device. For example, to display a portion of the memory of the current device at offset 1000, you could use

```
d s0+1000
```

Note: You may have to adjust the remap register of the device to access a specific portion of the local memory.

PLX Device Commands

PLX device commands are designed for use with the PLX family of PCI devices.

Note: PLX device commands are the most likely to change from one version to the next. *As such, this section of the documentation may not be current.* Be sure to read any addendums and errata to PLXMon. These documents discuss important changes, corrections and improvements to the program. The PLX website and FTP site (`http://www.plxtech.com`) may contain important late-breaking news as well.

PCI Configuration Registers: `pcr`

```
pcr [regnum [regval]]
```

Table Chapter 4 -12. PCI Configuration Registers (`pcr`) Command Input Parameters

Parameter	Description
`regnum`	Register number (long-word aligned)
`regval`	Register value (long word)

If no parameter is specified, this command displays all the PCI Configuration Registers of the currently selected device. When a legal *`regnum`* is specified, one register is displayed. Apply the *`regval`* parameter to set a PCI Configuration Register to a specific value.

Note: `pcr` does not prohibit writing to unwritable registers.

Local Configuration Registers: `lcr`

```
lcr [regnum [regval]]
```

Table Chapter 4 -13. Local Configuration Registers (`lcr`) Command Input Parameters

Parameter	Description
`regnum`	Register number (long-word aligned)
`regval`	Register value (long word)

If no parameter is specified, this command displays all the Local Configuration Registers of the currently selected device. When a legal ***`regnum`*** is specified, one register is displayed. Apply the ***`regval`*** parameter to set a Local Configuration Register to a specific value.

Note: `lcr` does not prohibit writing to unwritable registers.

Run-Time Registers: `rtr`

```
rtr [regnum [regval]]
```

Table Chapter 4 -14. Run-Time Registers (`rtr`) Command Input Parameters

Parameter	Description
`regnum`	Register number (long-word aligned)
`regval`	Register value (long word)

If no parameter is specified, this command displays all the Run-Time Registers of the currently selected device. When a legal ***`regnum`*** is specified, one register is displayed. Apply the ***`regval`*** parameter to set a Run-Time Register to a specific value.

Note: `rtr` does not prohibit writing to unwritable registers.

Local DMA Registers: `ldr`

```
ldr [regnum [regval]]
```

Table Chapter 4 -15. Local DMA Registers (`ldr`) Command Input Parameters

Parameter	Description
regnum	Register number (long-word aligned)
regval	Register value (long word)

If no parameter is specified, this command displays all the Local DMA Registers of the currently selected device. When a legal *regnum* is specified, one register is displayed. Apply the *regval* parameter to set a Local DMA Register to a specific value.

Note: This command applies only to PLX devices with DMA registers that are accessible from the PCI side. Accessing other devices will probably yield garbage.

Note: `ldr` does not prohibit writing to unwritable registers.

Read and Write EEPROM: eep

```
eep [regnum [regval]]
```

Table Chapter 4 -16. Read and Write EEPROM (`eep`) Command Input Parameters

Parameter	Description
`regnum`	Register number (long-word aligned)
`regval`	Register value (long word)

If no parameter is specified, this command displays all the EEPROM of the currently selected device. When a legal *`regnum`* is specified, one register is displayed. Apply the *`regval`* parameter to set an EEPROM location to a specific value.

Note: This command applies only to PLX devices that have EEPROMs connected. Accessing other devices will probably yield garbage.

This command is written specifically for National Semiconductor NMC93CS46 EEPROM (and compatible) devices. Refer to the EEPROM manufacturer's documentation for device details.

Writing to EEPROM could fail for many reasons. The NMC93CS46 includes features to prevent accidental writes, including the ability to permanently prevent writes to a portion of its memory.

Before writing a value to the EEPROM, this command issues a `WREN` (Write Enable) instruction to the EEPROM. After writing the value, it then issues a `WDS` (Write Disable) instruction.

Note: It is recommended that you verify any writes to the EEPROM.

Refer to the sections, “Read EEPROM: `re`” and “Write EEPROM: `we`,” for information about related commands.

Interactive DMA programming: `idma`

```
idma
```

Interactive DMA programming helps you through the arduous process of programming the DMA. (You can get the same results using register and memory manipulation commands.)

Note: The `idma` command applies only to PLX devices that allow their DMA registers to be programmed from the PCI side (such as the PLX PCI 9060SD and the PLX PCI 9060 Rev 3).

The command uses a question and answer session that is closely associated to the questions and answers a live programmer would have. When you use `idma`, it is recommended that you have the PLX DMA register description at hand. The command asks the following questions:

```
DMA channel (0 or 1)?
Chained mode (yes or no)?
Local to PCI (yes or no)?
PCI Address?
Local Address?
Count?
```

If you choose Chained mode, you also will be asked the following questions:

```
9060SD: Descriptor in PCI space (yes or no)?
Descriptor address?
End of chain (yes or no)?
```

Note: The `idma` command is generic for all PLX DMA implementations, so it may ask questions that are not appropriate for your device.

Each question displays the current setting. Press `ENTER` to accept the setting, or type in a new setting, then press `ENTER`. Unfortunately, `idma` does not let you to go backwards through the questions. If you must change the response to a previous question, press `CTRL-C`, then `ENTER` to stop the command. You may now execute the `idma` command.

Note: Remember that you are responsible for all memory management! This can get confusing, especially in Chained mode, because you must manage memory for chain descriptors and data buffers, in at least two physical memory spaces!

Chained mode—`idma` helps you manage descriptors by suggesting addresses in which to store the descriptors. The suggested addresses are based on user-defined variables:

- Suggested PCI-side descriptor storage variable—*`hbuf`*
- Suggested local-side descriptor storage variable—*`membase`*

You can change these variables to match your system (refer to the section, “User-Defined Variables,” for additional information).

Chained mode—When you finish entering the last descriptor, `idma` displays all the descriptors in the chain. It is recommended that you review the transfer details and descriptor locations of the chain before starting the DMA transfer.

Chained mode tip: You can make the DMA chain of descriptors loop indefinitely by making the last descriptor refer to the first descriptor, and using memory manipulation to reset the End-of-Chain bit in the last descriptor. Start DMA by executing the `dma` command (refer to the section, “DMA Programming: `dma`,” for additional information).

DMA Programming: `dma`

Noninteractive DMA programming. This command simply starts a DMA transfer—chain descriptors, mode, address and count registers are used *as is*.

You can use Interactive DMA programming (refer to the section, “Interactive DMA programming: `idma`”) to set up and test a DMA transfer, then use `dma` to quickly repeat the transfer.

Read EEPROM: `re`

```
re [range]
```

Table Chapter 4 -17. Read EEPROM (`re`) Command Input Parameter

Parameter	Description
`range`	Destination range (defaults to user-defined variable *`hbuf`*, for `0x40` words)

This command reads the EEPROM of the selected PCI device into a specified range (refer to the section, “Ranges”) of PCI memory.

If no parameter is specified, this commands reads 0x40 words (0x80 bytes) into PCI memory, specified by the *`hbuf`* user-defined variable. If *`range`* includes the destination address, but not the length, the length defaults to 0x40 words.

This command is written specifically for National Semiconductor NMC93CS46 EEPROM (and compatible) devices. Refer to the EEPROM manufacturer’s documentation for device details.

Unlike other commands, the range always specifies words.

Note: PLXMon does not restrict use of this command on inappropriate PCI devices.

Note: This command does not test the EEPROM Present bit.

Refer to the section, “Read and Write EEPROM: `eep`,” for an alternative method of reading and writing the EEPROM.

Write EEPROM: `we`

`we range`

Table Chapter 4 -18. Write EEPROM (`we`) Command Input Parameter

Parameter	Description
range	Source range

This command writes the specified range (refer to the section, "Ranges") of PCI memory to the EEPROM of the selected PCI device.

Note: This command applies only to PLX devices that have EEPROMs connected. Accessing other devices will probably yield garbage.

This command is written specifically for National Semiconductor NMC93CS46 EEPROM (and compatible) devices. Refer to the EEPROM manufacturer's documentation for device details.

Unlike other commands, the range always specifies words.

Unlike the EEPROM `re` command, this command requires the source range to be specified. The length must resolve to 0x40 words or less.

This command could fail for many reasons. The NMC93CS46 includes features to prevent accidental writes, including the ability to permanently prevent writes to a portion of its memory.

Before writing a value to the EEPROM, this command issues a `WREN` (Write Enable) instruction to the EEPROM. After writing the value, it then issues a `WDS` (Write Disable) instruction.

To read, modify and write the EEPROM, use the Read EEPROM command (refer to the section, "Read EEPROM: `re`"), then use the memory manipulation functions, followed by this command.

Refer to the section, "Write EEPROM: `we`," for an alternative method of reading and writing the EEPROM.

PLXMon Internal Commands

The following sections discuss the PLXMon internal commands.

View PLXMon License Agreement: `license`

```
license
```

To view the PLXMon License Agreement, type

```
license<Enter>
```

at the PLXMon prompt (&).

PLXMon displays a comprehensive LICENSE AGREEMENT. This agreement spans several screens of text. After reading one screen, press any key other than “`q`” to continue to the next screen.

Refer to the section, “PLXMon License Agreement,” for important information about the license agreement.

Show Variables: `vars`

```
vars
```

This command displays all variables. Refer to the section, “User-Defined Variables,” for information about defining variables.

Expressions: `expr`

```
expr valexpr
```

Table Chapter 4 -19. Expressions (`expr`) Command Input Parameter

Parameter	Description
`valexpr`	Expression to be evaluated

This command evaluates *`valexpr`* and displays the result in hexadecimal and signed-decimal format.

Note: Expressions can be evaluated without this (or any) command (refer to the section, “Expressions,” for additional information).

Define a Macro: `define`

```
define macname [macbody]
```

Table Chapter 4 -20. Define a Macro (`define`) Command Input Parameters

Parameter	Description
macname	Macro name
macbody	Macro body

This command defines (or deletes) a macro. The first parameter names the macro, while the remainder of the line is stored as the macro body. The macro body can include commands, variables and other macros.

Note: The macro body is *not* checked for validity.

Caution: Referencing the same macro within a macro works; however, this will probably overflow the PLXMon stack!

To execute a macro, type the macro name anywhere you would normally type a command.

Once a macro is defined, you can delete it by using the `define` command, with the macro name as the only parameter. A macro can be redefined by entering the macro name, followed by a new macro body.

PLXMon scans the built-in command list before the macro list. For example, if your macro name matches a built-in command, the command, rather than your macro, will be executed (that is, PLXMon will ignore your macro).

There is room for about 25 macros, at 80 characters per macro.

Macros can be saved to a file for future use (refer to the section, “`save`,” for additional information).

Show Macros: `macs`

```
macs
```

This command displays all macros. Refer to the section, “Define a Macro: `define`,” for information about defining macros.

Repeat: `r`

```
r [countexpr]
```

Table Chapter 4 -21. Repeat (`r`) Command Input Parameter

Parameter	Description
countexpr	Iteration count

This command repeats all or part of a command line, indefinitely or for a specific number of iterations.

This is one of the more useful commands in PLXMon. In its simplest application, you append the `r` command to the end of the command line; PLXMon repeats the entire command line indefinitely until you press a key to stop the process:

```
ob 100 ab; r
```

Adding a *countexpr* variable limits the number of iterations:

```
ob 100 ab; r 10
```

You can also use nesting tokens, “`[`” and “`]`”, to repeat part of a command line:

```
ib 200 [ob 100 ab; r 10]
```

This example inputs one byte from port 200, then outputs `ab` to port 100 10 times. (The absence of nesting tokens tells PLXMon to repeat, starting at the beginning of the line.)

There is no limit to the number of nesting tokens you can apply. For example:

```
d 1000 [ib 200 [ob 100 ab; r 10]]r 5
```

Note: The `r` command can be used in a macro body; however, it only makes sense if you also specify a *countexpr* variable.

echo

```
echo text
```

Table Chapter 4 -22. echo Command Input Parameter

Parameter	Description
`text`	Any text up to the point where you press `ENTER`

This command echoes the remainder of the command line. Use this command to document a PLXMon command file (refer to the section, “PLXMon Batch Files,” for additional information).

wait

```
wait countexpr
```

Table Chapter 4 -23. `wait` Command Input Parameter

Parameter	Description
`countexpr`	Number of cycles to wait

This command waits while the processor down-counts *`countexpr`*. The wait loop is uncalibrated; PLXMon simply “sits” in a software loop until *`countexpr`* reaches zero.

The longest wait is obtained by passing “0” as the parameter. This is equivalent to wait 100000000.

Note: There is no way to break out of a wait, so be careful using a large *`countexpr`* value.

base

```
base [baseexpr]
```

Table Chapter 4 -24. base Command Input Parameter

Parameter	Description
baseexpr	New radix (base 16 max)

This command sets the PLXMon radix to a new base. If ***baseexpr*** is a number (not a variable or an expression), it must be in base 10. The highest base (and its default base) of PLXMon is base 16.

Entering numeric parameters to a command using digits greater than the radix (but less than F) generates an error message. However, the command will continue, and the offending digits are truncated to zero.

Use `base`, with no parameters specified, to determine the current radix of PLXMon.

save

```
save [filename]
```

Table Chapter 4 -25. save Command Input Parameter

Parameter	Description
filename	Name of command file

This command saves current variables and macros to a specific filename. The file is saved as simple, editable text. The variables and macros are saved as PLXMon commands—you can edit the file, even add other PLXMon commands, then execute the file at any time using the `read` command (refer to the section, “`read`,” for additional information).

If no parameter is specified, `save` uses `SAVE.MON` as the default filename.

`read`

```
read [filename]
```

Table Chapter 4 -26. `read` Command Input Parameter

Parameter	Description
`filename`	Name of command file

This command reads and executes a PLXMon command file. PLXMon reads characters from the file as if they are typed-in command lines.

If no parameter is specified, `read` uses `SAVE.MON` as the default filename.

The companion to `read` is the `save` command. The `save` command saves the PLXMon macros and variables in a format ready for the `read` command (refer to the section, "`save`," for additional information).

`range`

```
range
```

This command does nothing! It is simply a placeholder for the PLXMon online help for using ranges. Refer to the section, "Ranges," for information about specifying ranges.

`environment`

```
environment
```

This command does nothing! It is simply a placeholder for the PLXMon online help for the environment variable. Refer to the section, "MON Environment Variable," for information about setting the environment variable.

`cmd_edit`

```
cmd_edit
```

This command does nothing! It is simply a placeholder for the PLXMon online help for editing command lines. Refer to the section, “Scrolling and Editing Command Lines,” for information about editing command lines.

`quit, q`

This command quits PLXMon.

`help, h, ?`

```
help [command]

h [command]

? [command]
```

Table Chapter 4 -27. `help`, h, ? Command Input Parameter

Parameter	Description
command	Command name

These commands display online help for PLXMon commands. If no parameter is specified, these commands print a complete list of PLXMon commands. Enter a PLXMon command as a parameter to display the online help message for that command.

Enter an asterisk (“`*`”) as a parameter to display the online help messages for all commands.

The help text of a command may display a parameter within square braces (“`[`” and “`]`”). Such a parameter is optional for the command.

The help text of a command may display ellipses (…) following a parameter. This indicates that more parameters of the same type can follow the first parameter.

Chapter 5

Examples

This chapter provides examples of how to use the PLXMon commands. Most PLXMon commands are simple and straightforward. Combining functions on a command line can yield useful results, as shown in these examples.

Write Values to Port and Check Result

These examples are popular for bringing up new hardware for the first time. To make the examples realistic, they use the serial port that is typically used for the mouse (0x3f8).

The first example repeatedly reads all eight registers of the serial chip used with the mouse. Try it—as you move the mouse, you should see the data port register change:

```
ib 3f8 l 8; r
```

The second example writes an incrementing pattern to the data port, then reads all eight registers of the serial chip—one set of reads for each write:

```
x = 0 [ib 3f8 l 8; ob 3f8 x; x = x + 1; r 100]
```

Caution: Running this example with PLXMon95 can cause strange button presses to occur!

Fill Memory with a Pattern

This example writes 0x100 bytes of an incrementing byte pattern to the host buffer:

```
x = 0 [e (hbuf + x) x; x = x + 1; r 100]
```

The PLXMon fill command (refer to the section, "Fill: `f`"), however, may suffice for your needs.

Program EEPROM from File

Use any text editor (such as Notepad or WordPad) to create a batch file of `eep` commands:

```
eep 0 905010b5

eep 4 06800001

…
```

Then, use the `read` command to execute the file. For example, if your text file is named `EEP.MON`, execute the following:

```
read eep.mon
```

Setting a Variable with the Contents of Memory

You can use the dereferencing operator ("`*`") to set a variable to a value found at a specific memory location. For example, you want to set a variable ***p*** to the port address of the first serial chip on your PC. The BIOS keeps a list of first four serial chips at memory location 0x400:

```
p = (*400) &ffff
```

Variables are always long words, and ports are words; therefore, &ffff is added to the command to force the upper bits to zero.

Notes to Programmers

You may want to use the PLXMon source code for reference or to build your own version of PLXMon. If you build your own version, you can extend the command set for your hardware design. Extending the command set is fairly easy, as most of the work (both the user front end, and the PLX/PCI back end) is already written and tested.

Contact PLX for information about obtaining PLXMon source code.

PLXMon Development Environment (DOS and Windows 95 Versions)

PLXMon, the DOS version of PLXMon, is compiled and linked with Microsoft MSVC, version 1.51. The DPMI libraries are provided by DOS/16M from Rational Systems. After linking, the program is extended to run in Protected mode, using Splicer/Loader, also from Rational Systems.

PLXMon95, the Windows 95 version of PLXMon, and `PHYSACC.VXD` are compiled and linked with Microsoft MSVC, version 4.00.

PLXMon DPMI (DOS Only)

If you choose to rebuild the DOS version of PLXMon, you will find that supporting the DPMI will probably be the biggest obstacle. Due to license agreements, PLX cannot include the DPMI libraries or Splicer/Loader—you must supply the DPMI interface and support.

PLXMon was developed by PLX using DOS/16M DPMI from Rational Systems. The code is structured with the expectation of another DPMI being applied; however, as of this writing, no other DPMI provider has been tested.

PLX Family of PCI Devices

Because PLXMon supports the complete family of PLX PCI devices, and you will probably only be interested in one of them, the device files are modularized according to family members. You will find one file that has useful functions for any PLX PCI device, and a set of extension files for specific PLX PCI devices.

PCI BIOS Support—PLXMon (DOS Version)

The DOS version of PLXMon makes most of the PCI BIOS functions available as “C” functions. The PCI BIOS is called using int 1A.

Mechanism 2 is no longer being tested.

PCI BIOS Support—PLXMon95 (Windows 95 Version)

PLXMon95 does not use the PCI BIOS directly. To find all the PCI devices, PLXMon95 searches the registry (`HKEY_DYN_DATA\ConfigManager\Enum`) for all PCI devices. For reading and writing PCI configuration registers, `PHYSACC.VXD` includes a function that calls the configuration manager function (`CM_Call_Enumerator_Function`).

PLX Technology, Inc.
390 Potrero Ave.
Sunnyvale, CA. 94086 USA
Tel: 1-800-759-3735
Fax: 1-408-774-2169
Email: info@plxtech.com
Web and FTP Site:
www.plxtech.com

Order Number: PLXMon-M-001