

# PCI SDK User's Manual

Release 2.1, initial publishing November 23, 1998.

Copyright © 1998, PLX Technology, Inc.. All rights reserved.

This document contains proprietary and confidential information of PLX Technology Inc. (PLX). The contents of this document may not be copied nor duplicated in any form, in whole or in part, without prior written consent from PLX.

PLX provides the information and data included in this document for your benefit, but it is not possible for us to entirely verify and test all of this information in all circumstances, particularly information relating to non-PLX manufactured products. PLX makes no warranties or representations relating to the quality, content or adequacy of this information. Every effort has been made to ensure the accuracy of this manual, however, PLX assumes no responsibility for any errors or omissions in this document. PLX shall not be liable for any errors or for incidental or consequential damages in connection with the furnishing, performance, or use of this manual or the examples herein. PLX assumes no responsibility for any damage or loss resulting from the use of this manual; for any loss or claims by third parties which may arise through the use of this SDK; for any loss or claims by third parties which may arise through the use of this SDK; and for any damage or loss caused by deletion of data as a result of malfunction or repair. The information in this document is subject to change without notice.

Product and Company names are trademarks or registered trademarks of their respective owners.

Document number: PCISDK-MAN-210.doc



# Table of Contents

<b>1. INTRODUCTION</b>	<b>1-1</b>
1.1 About This Manual .....	1-1
1.2 PCI SDK Features.....	1-1
1.3 Where To Go From Here.....	1-1
1.4 Other PCI SDK Manuals .....	1-1
1.5 Conventions .....	1-2
1.5.1 Windows Programming Conventions .....	1-2
1.6 Terminology .....	1-2
1.7 Development Tools Needed .....	1-2
1.8 Customer Support .....	1-3
<b>2. GETTING STARTED</b>	<b>2-1</b>
2.1 PCI SDK Installation .....	2-1
2.1.1 Unpacking .....	2-1
2.1.2 Minimum System Requirements.....	2-1
2.1.3 Development Requirements.....	2-1
2.1.4 Software Installation .....	2-1
2.1.4.1 Windows NT Installation Procedures.....	2-1
2.1.4.1.1 Windows NT Device Driver installation .....	2-2
2.1.4.2 Windows98 Installation Procedures .....	2-2
2.1.4.2.1 Windows98 Software Installation.....	2-2
2.1.4.2.2 Windows98 Device Driver installation.....	2-3
2.1.4.3 Removing All Software .....	2-4
2.1.4.4 PCI SDK v2.1 Compatibility.....	2-4
2.1.4.5 Troubleshooting.....	2-4
2.1.4.5.1 Increasing Available System Pages In WinNT.....	2-4
2.1.4.5.2 Driver Interrupt Sharing.....	2-5
2.2 Understanding The PCI SDK .....	2-5
2.2.1 IOP Software.....	2-5
2.2.1.1 Introduction .....	2-5
2.2.1.2 IOP Applications .....	2-5
2.2.1.2.1 MiniRom Application .....	2-6
2.2.2 Windows Based Software .....	2-6
2.2.3 Introduction.....	2-6
2.2.3.1 Windows NT Device Drivers .....	2-7
2.2.3.1.1 Starting And Stopping.....	2-7
2.2.3.1.2 Event Logging.....	2-8

2.2.3.1.3	Registry Configuration.....	2-9
2.2.3.1.4	Driver Configuration.....	2-10
2.2.3.2	Windows98 Device Drivers.....	2-11
2.2.3.2.1	Starting And Stopping.....	2-11
2.2.3.2.2	Event Logging.....	2-11
2.2.3.2.3	Registry Configuration.....	2-11
2.2.3.2.4	Known problems with the Windows98 device drivers. ....	2-11
2.3	Using The PCI SDK With A New Board .....	2-12
2.4	Using PCI SDK API Libraries With Other Operating Systems And Compilers .....	2-12
2.5	RTOS Support .....	2-13
2.6	Source Code Availability.....	2-13
<b>3.</b>	<b>PCI SDK SOFTWARE ARCHITECTURE OVERVIEW</b>	<b>3-1</b>
3.1	Assumptions .....	3-1
3.1.1	PCI SDK Assumptions .....	3-1
3.1.2	IOP API And IOP Software Assumptions .....	3-1
3.1.3	PCI API and Win32 Software Assumptions .....	3-1
3.2	Overview.....	3-2
3.3	Software Architecture.....	3-3
3.4	IOP Software Architecture .....	3-3
3.4.1	Board Support Package (BSP) Library .....	3-4
3.4.1.1	Microprocessor Initialization Module .....	3-5
3.4.1.1.1	Microprocessor Boot Code .....	3-5
3.4.1.1.2	Interrupt Service Routine.....	3-5
3.4.1.2	Board Initialization Module.....	3-5
3.4.1.2.1	Board Initialization Routine.....	3-6
3.4.1.3	The Main() And AppMain() Functions .....	3-6
3.4.2	IOP API Library.....	3-8
3.4.2.1	DMA Resource Manager.....	3-8
3.4.3	Back-End Monitor .....	3-12
3.4.3.1	Back-End Monitor Serial Protocol .....	3-12
3.4.3.1.1	Reset IOP Microprocessor Command.....	3-13
3.4.3.1.2	IOP Memory Read Command.....	3-13
3.4.3.1.3	IOP Memory Write Command.....	3-13
3.4.4	Methods For Debugging IOP Applications.....	3-14
3.4.4.1	Operation Of The Back-End Monitor In A System.....	3-14
3.4.5	IOP Applications.....	3-14
3.4.5.1	IOP Memory And IOP Applications .....	3-15
3.4.5.2	MiniRom Application.....	3-16

3.4.6	Porting The PCI SDK To New Platforms .....	3-16
3.4.7	Support For Multiple PCI ICs On One Board.....	3-17
3.5	Host Win32 Software Architecture.....	3-17
3.5.1	PLXMon98 .....	3-17
3.5.1.1	Serial Communication .....	3-18
3.5.1.2	PCI API/Device Driver Communication .....	3-18
3.5.1.2.1	PCI API Library .....	3-18
3.5.1.2.2	Win32 Device Driver .....	3-18
3.5.2	Win32 Applications And The PCI SDK .....	3-19
3.5.3	Win32 Device Driver Overview .....	3-19
3.5.3.1	PCI IC Device Driver Module .....	3-19
3.5.3.2	PCI IC Services Module .....	3-20
3.5.4	Creating A New Driver .....	3-20
3.5.5	Device Driver Features .....	3-20

**APPENDIX A. SERIAL EEPROM SETTINGS**

**A-1**

# List of Figures

Figure 2-1 Components of the PCI SDK .....	2-1
Figure 2-2 Windows Host software Layout for PCI SDK v2.1 .....	2-6
Figure 2-3 The Devices Utility Window. ....	2-7
Figure 2-4 The Event Viewer Window.....	2-8
Figure 2-5 The Event Detail Window.....	2-8
Figure 2-8 The PCI SDK Device Driver Wizard.....	2-10
Figure 3-1 The PCI SDK Software Architecture .....	3-2
Figure 3-2 The IOP Software Architecture.....	3-4
Figure 3-3 The Data Stream Flow Diagram. ....	3-7
Figure 3-4 Scatter-Gather DMA Flow Diagram.....	3-9
Figure 3-5 Block DMA Transfer Flow Diagram .....	3-10
Figure 3-6 The Shuttle DMA Flow Diagram.....	3-11
Figure 3-7 IOP Memory Diagram.....	3-15
Figure 3-8 The Host Software Architecture.....	3-17
Figure 3-9 The PLX Device Driver Layout.....	3-19



**PLX SOFTWARE LICENSE AGREEMENT**

THIS SOFTWARE DESIGN KIT INCLUDES PLX SOFTWARE THAT IS LICENSED TO YOU UNDER SPECIFIC TERMS AND CONDITIONS. CAREFULLY READ THE TERMS AND CONDITIONS PRIOR TO USING THIS DESIGN KIT. BY OPENING THIS PACKAGE OR INITIAL USE OF THIS SOFTWARE DESIGN KIT INDICATES YOUR ACCEPTANCE OF THE TERMS AND CONDITIONS. IF YOU DO NOT AGREE WITH THEM, YOU SHOULD RETURN THE ENTIRE SOFTWARE DESIGN KIT TO PLX.

**LICENSE** Copyright (c) 1998 PLX Technology, Inc.

This PLX Software License agreement is a legal agreement between you and PLX Technology, Inc. for the PLX Software Design Kit (SOFTWARE PRODUCT.) which is provided on the enclosed PLX diskettes, or may be recorded on other media included in this Software Design Kit. PLX Technology owns this SOFTWARE PRODUCT. The SOFTWARE PRODUCT is protected by copyright laws and international copyright treaties, as well as other intellectual property laws and treaties, and is licensed, not sold. If you are a rightful possessor of the Software Design Kit, PLX grants you a license to use the SOFTWARE PRODUCT as part of or in conjunction with a PLX chip on a per project basis. PLX grants this permission provided that the above copyright notice appears in all copies and derivatives of the SOFTWARE PRODUCT. Use of any supplied runtime object modules or derivatives from the included source code in any product without a PLX Technology, Inc. chip is strictly prohibited. You obtain no rights other than those granted to you under this license. You may copy the SOFTWARE PRODUCT for backup or archival purposes. You are not authorized to use, merge, copy, display, adapt, modify, execute, distribute or transfer, reverse assemble, reverse compile, decode, or translate the SOFTWARE PRODUCT except to the extent permitted by law.

**GENERAL**

If you do not agree to the terms and conditions of this PLX Software License Agreement, do not install or use the Software Design Kit and promptly return the entire unused SOFTWARE PRODUCT to PLX Technology, Inc. You may terminate your license at any time. PLX Technology may terminate your license if you fail to comply with the terms and conditions of this License Agreement. In either event, you must destroy all your copies of this SOFTWARE PRODUCT. Any attempt to sub-license, rent, lease, assign or to transfer the Software Design Kit except as expressly provided by this license, is hereby rendered null and void.

**WARRANTY**

PLX Technology, Inc. provides this SOFTWARE PRODUCT AS IS, WITHOUT ANY WARRANTY, EXPRESS OR IMPLIED, INCLUDING WITHOUT LIMITATION, ANY WARRANTY OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. PLX makes no guarantee or representations regarding the use of, or the results based on the use of the software and documentation in terms of correctness, or otherwise; and that you rely on the software, documentation, and results solely at your own risk. In no event shall PLX be liable for any loss of use, loss of business, loss of profits, incidental, special or, consequential damages of any kind. In no event shall PLX's total liability exceed the sum paid to PLX for the product licensed hereunder.



## **PLX Copyright Message Guidelines**

The following copyright message must appear in all software products generated and distributed by PLX customers:

**“Copyright (c) 1998 PLX Technology, Inc.”**

### **Requirements:**

- Arial font,
- Font size 12
- Bold type
- Must appear as shown above in the first section or the so called “Introduction Section” of all manuals

Must also appear as shown above in the beginning of source code as a comment

# 1. Introduction

## 1.1 About This Manual

The PLX family of embedded bridge ICs bridge the PCI bus to Intel, Power PC processors and other processors. They provide full I<sub>2</sub>O compatibility. The PCI SDK provides a powerful IOP API, and Windows software that are used to control PLX devices. We are confident that through the use of the PCI SDK, your PLX designs will be brought to market faster and more efficiently.

This manual provides information about the functionality of the PCI SDK. Customers have the choice of using the PCI SDK with any PLX Reference Design Kit (RDK), or a generic device that uses a PLX IC. Users should consult this manual when installing the PCI SDK and for general information.

## 1.2 PCI SDK Features

The PCI SDK includes the following features:

- A feature based IOP API, with support for a variety of PLX PCI ICs;
- Board Support Package (BSP) that allows customization of the PCI SDK;
- A Back-End Monitor application used for debugging;
- IOP DMA Resource Manager that supports three modes of operation;
- A PCI API and device drivers compatible with Windows NT/98; and,
- PLXMon98, A Windows GUI application used to configure and modify PLX PCI devices.

## 1.3 Where To Go From Here

The following is a brief summary of the chapters to help guide your reading of this manual:

Chapter 2, Getting Started, discusses how to start using the PCI SDK and some of the applications provided.

Chapter 3, PCI SDK Software Architecture Overview, describes the layout of the PCI SDK software.

Chapter 3, section 3.4, IOP Software Architecture, provides a brief explanation of the IOP software, specifically the Board Support Package (BSP), the IOP API, and the Back-End Monitors (BEM).

Chapter 3, section 3.5, Host Win32 Software Architecture, provides a brief explanation of the Win32 software, specifically the PCI API and the device driver.

## 1.4 Other PCI SDK Manuals

The PCI SDK includes the following manuals which users should consult for design details:

Programmer's Reference Manual: This manual covers all software design issues regarding the device drivers, Application Programmer's Interface (API) and user applications.



PLXMon 98 User's Manual: This manual describes the usage of the PLXMon98 application.

PCI IC Data Sheets & Application Notes (CD-ROM): This CD covers all the functionality of the PCI IC.

## 1.5 Conventions

Please note:

- *italics* are used to represent variables, function names, and program names;
- `courier` is used to represent source code given as examples.

### 1.5.1 Windows Programming Conventions

Some designers may not be familiar with Windows programming conventions. Therefore, a few conventions have been noted below:

- *PU32 data* is analogous to *U32 \*data* or *unsigned long \*data*; and
- *IN* and *OUT* are used to distinguish between parameters that are being passed into API functions and parameters that are being returned by API functions.

## 1.6 Terminology

All references to Windows NT assume Windows NT 4.0 or higher and may be denoted as WinNT.

All references to Windows98 may be denoted as Win98.

All references to Windows95 may be denoted as Win95.

Win32 references are used throughout this manual to mean any application that is compatible with the Windows 32-bit environment.

All references to IOP (I/O Platform) throughout this manual denote the embedded hardware and all references to IOP software denote the embedded software.

## 1.7 Development Tools Needed

Development tools needed for the PCI SDK that are not supplied include:

- Microsoft Visual C++ 5.0, with Microsoft Developer Studio;
- Microsoft Platform Software Development Kit (SDK);
- Microsoft Windows NT Device Driver Kit (DDK), version 4.0;
- Microsoft Windows98 Device Driver Kit (DDK);
- Diab Data, Inc. Compiler and Linker for the MPC860, version 4.0b;
- IBM High C/C++ PowerPC Cross-Compiler, version 1.0 (7/31/96); and,
- 401 EVB Software Support Package, version 1.6.4 (4/1/97).



## 1.8 Customer Support

Prior to contacting customer support, please ensure you have the following information:

1. You are situated close to the computer that has the PCI SDK installed;
2. Serial Numbers of the PLX PCI RDKs (if there is any in use with the PCI SDK);
3. Type of processor on the PLX PCI RDK;
4. Operating System version and type; and
5. Description of problem.

You may contact PLX customer support at:

Address: PLX Technology, Inc.  
390 Potrero Avenue  
Sunnyvale, CA 94086

Phone: 408-774-9060  
Fax: 408-774-2169  
Web: <http://www.plxtech.com>

You may send email to one of the following addresses:

[west-apps@plxtech.com](mailto:west-apps@plxtech.com)  
[mid-apps@plxtech.com](mailto:mid-apps@plxtech.com)  
[east-apps@plxtech.com](mailto:east-apps@plxtech.com)  
[euro-apps@plxtech.com](mailto:euro-apps@plxtech.com)  
[asia-apps@plxtech.com](mailto:asia-apps@plxtech.com)



## 2. Getting Started

### 2.1 PCI SDK Installation

#### 2.1.1 Unpacking

The PCI SDK comes complete with the following items (see Figure 2-1):

- User's Manual (this document);
- Programmer's Reference Manual;
- PLXMon98 User's Manual; and
- 1 CD-ROM.

Please take the time now to verify your PCI SDK is complete. If not, please contact Customer Support.

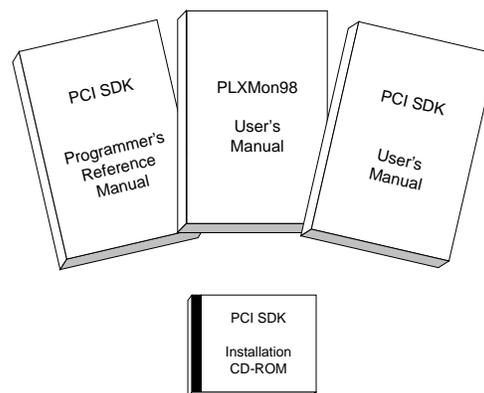


Figure 2-1 Components of the PCI SDK

#### 2.1.2 Minimum System Requirements

Minimum host system requirements for the PCI SDK are as follows:

- Windows NT 4.0 with Service Pack 3, or Windows98;
- 32MB RAM (when used with only one PLX PCI RDK. Additional memory may be required if more than one PLX PCI RDK are in the system);
- 20MB hard drive space; and,
- 1 RS 232 port.

#### 2.1.3 Development Requirements

The PCI SDK was developed using Window NT 4.0 and Windows98 operating systems.

The PCI API was developed using Microsoft Developer Studio, supplied with Microsoft Visual C++ 5.0 and the Microsoft Platform Software Development Kit.

The WinNT device drivers were developed using the Microsoft Windows NT DDK, version 4.0 and Microsoft Visual C++ 5.0.

The WDM Win98 device drivers were developed using the Microsoft Windows98 DDK and Microsoft Visual C++ 5.0.

#### 2.1.4 Software Installation

##### 2.1.4.1 Windows NT Installation Procedures

To install the PCI SDK Support Software, complete the following:



*Note: All previous PCI SDK versions located on the computer must be removed before installing a PCI SDK update. Refer to section 2.1.4.3 for more details.*

1. Insert the CD-ROM into the appropriate CD-ROM drive.
2. Run "D:\Install.exe" either by typing it at a command prompt or by choosing the Run option of the Start Menu (where "D:" is the drive letter for the CD-ROM Drive).
3. This will launch the Install Wizard application that will ask you to select the PLX RDK that you are using. The appropriate PCI SDK version will then be installed.
4. Reboot the computer.

*Note: For proper WinNT installation, the PCI SDK should be installed by a user with "administrator" user rights.*

The default installation directory may be changed from default path (C:\Plx\PciSdk) to any drive and path that is desired. This document uses "<INSTALLPATH>" to denote the installation directory.

This completes the PCI SDK software installation.

#### **2.1.4.1.1 Windows NT Device Driver installation**

Unlike the Win98 installation wizard the Windows NT installation wizard takes care of the device driver installation.

#### **2.1.4.2 Windows98 Installation Procedures**

The installation of the PCI SDK Support Software onto a Win98 system requires two steps: Install the PCI SDK files, and, then install the Win98 device drivers. The following two sections describe how to completely install the PCI SDK Support Software for Win98.

##### **2.1.4.2.1 Windows98 Software Installation**

To install the PCI SDK Support Software, complete the following:

*Note: All previous PCI SDK versions located on the computer must be removed before installing a PCI SDK update. Refer to section 2.1.4.3 for more details.*

1. Ensure no PLX engineering boards are installed in your computer.
2. Insert the CD-ROM into the appropriate CD-ROM drive.
2. Run "D:\Install.exe" either by typing it at a command prompt or by choosing the Run option of the Start Menu (where "D:" is the drive letter for the CD-ROM Drive). The interactive installation program will install all files.
3. This will launch the Install Wizard application that will ask you to select the PLX RDK that you are using. The appropriate PCI SDK version will then be installed.

The default installation directory may be changed from default path (C:\Plx\PciSdk) to any drive and path that is desired. This document uses "<INSTALLPATH>" to denote the installation directory.

This completes the PCI SDK software installation. Proceed to the next section to install the Win98 device driver.

### 2.1.4.2.2 Windows98 Device Driver installation

A device driver is necessary for the PCI SDK software to communicate to the PLX engineering board. PCI SDK applications cannot communicate with any engineering board through the PCI interface without a PCI SDK device driver installed. The installation script used to install the PCI SDK can not entirely install the PCI SDK. To install the device driver in Win98, complete the following:

1. After installing the PCI SDK successfully (see the previous section), shutdown the computer.
2. Insert a PLX engineering board into a free PCI slot.
3. Reboot the computer. Windows98 should first detect the new hardware device with a “New Hardware Found” message box. Acknowledge this message box.
4. Windows98 displays the “Add New Hardware” Wizard. Windows98 displays the following message: “This wizard searches for new drivers for:” with the corresponding board name following it. If you are using a PLX engineering board proceed to step 5. If you are using a custom engineering board with a PLX device then proceed to step 10.

#### Driver Installation for PLX Engineering Boards:

5. Click on the “Next” button. Once Windows98 has completed its search the following prompt is displayed: “What do you want Windows to do?” At the prompt select “Search for the best driver for your device”. This is the default option. Click on “Next” to continue.
6. The installation wizard asks “Windows will search [...] in any of the following selected locations.” Check none of the items in the list and click on “Next” to continue.
7. The device driver is ready to be installed when the installation wizard displays the following message: “Windows is now ready to install the best driver for this device.” Click on “Next” to continue.
8. The device driver installation is complete when Windows98 displays the following message: “Windows has finished installing the software that your new hardware device requires”.
9. Click on “Finish” to complete the Win 98 device driver installation.

#### Driver Installation for Custom Engineering Boards with PLX devices:

10. The installation wizard will detect your custom device as a “PCI Bridge”. Click on the “Next” button and then choose the option “Display a list of all the drivers in a specific location.” Click “Next”.
11. Select “Other Devices”. Click “Next”.
12. Now select the column that says “PLX Technology, Inc.” and choose “Unknown PCI 9054 board” if using a PCI 9054 device. Otherwise choose “Unknown PCI 9080 board” if using a PCI 9080 device. The install wizard will warn you that this device driver is not specific for your device. Ignore this warning by choosing “Yes”. Click “Next”.
13. Click “Finish” to complete the Win 98 device driver installation.

*Note: If you change the device from one slot to another, Windows98 will treat it as a “New Hardware” and will display the same dialog.*

Once the Win98 device driver installation is complete it is ready to run without having to reboot the system.



### 2.1.4.3 Removing All Software

To remove all PCI SDK Software, including device drivers, complete the following:

1. Stop all PLX applications;
2. Open the Windows Control Panel;
3. Double click on the Add/Remove Programs icon in the Control Panel window;
4. Choose the PCI SDK package from the item list; and
5. Click the Add/Remove... button.

*Note: This only removes the files that were originally installed by the PCI SDK installation program. For proper removal in WinNT, the PCI SDK should be removed by a user with "administrator" user rights.*

This completes the PCI SDK software removal.

### 2.1.4.4 PCI SDK v2.1 Compatibility

*The PCI SDK v2.1 host code is NOT compatible with previous PCI SDK IOP code. Therefore, before using any PCI SDK software, you must upgrade the FLASH and configuration EEPROM images on your PLX engineering board. If this step is not done, the PCI SDK will operate unpredictably.*

To upgrade your FLASH image follow the steps below:

- User's of the PCI9054RDK-860 may use PLXMon98, as described in the PLXMon98 User Manual, to download the "helloworld" sample to the PCI 9054RDK-860. The image should be programmed at FLASH offset 0x00000.
- User's of the PCI 9080RDK-401B may use PLXMon98, as described in the PLXMon98 User Manual, to download the "helloworld" sample to the PCI 9080RDK-401B. The image should be programmed at FLASH offset 0x60000.

To upgrade your serial EEPROM, please refer to Appendix A for more details.

### 2.1.4.5 Troubleshooting

If you experience difficulty during the installation of the PCI SDK software please:

- Verify that there is enough hard drive space for all software; and
- Verify that WinNT or Win98 operated properly before the PCI SDK installation.

*Warning: User may experience difficulties when using the PCI SDK with Windows NT with low memory and multiple PLX engineering boards. It is recommended that users increase the amount of available system pages in their System Registry. For more information, refer to section 2.1.4.5.1.*

#### 2.1.4.5.1 Increasing Available System Pages In WinNT

To change number of system pages in the System Registry:

1. From a command prompt type: regedt32. This will bring up the Registry Editor window. (This editor looks similar to the Windows Explorer application.)

2. Select the HKEY\_LOCAL\_MACHINE on Local Machine window from within the Registry Editor.
3. Open the SYSTEM folder.
4. From the SYSTEM folder, open the CurrentControlSet folder.
5. From the CurrentControlSet folder, open the Control folder.
6. From the Control folder, open the Session Manager folder.
7. From the Session Manager folder, open the Memory Management folder.
8. From the Memory Management folder, change the value of the SystemPages key from 0x0 to 0x13880.

If problems persist, please contact Customer Support.

### **2.1.4.5.2 Driver Interrupt Sharing**

The PCI SDK device drivers have interrupt sharing enabled. This allows PLX devices to share the same interrupt line as other devices. However, in order to share interrupts with non-PLX devices the device driver for the non-PLX device must also support sharing. Because many device drivers do not support interrupt sharing the PCI SDK can only be guaranteed to function properly with other PLX devices.

If a PCI SDK device driver will not start, a possible cause is the computer's BIOS is assigning an interrupt to the PLX device that is already being used by a device that doesn't support interrupt sharing. A possible work around for this condition is to manually configure the BIOS to assign a free interrupt to the PLX device.

## **2.2 Understanding The PCI SDK**

### **2.2.1 IOP Software**

#### **2.2.1.1 Introduction**

The PCI SDK includes several samples of IOP applications. Their purpose is to demonstrate how designers can interact with the PCI IC from IOP software. The IOP applications are user-interactive and require PLXMon98 with a serial cable link.

The IOP applications are designed specifically to run on a PLX engineering board. However, the IOP applications can be used as a good starting point for designers using their own hardware device.

#### **2.2.1.2 IOP Applications**

The PCI SDK contains several sample applications. By default all PLX engineering boards contain the "helloworld" application preprogrammed in FLASH memory. This application blinks an LED and prints "Hello World" from the serial port. Complete source code for this application is provided in the PCI SDK. Please refer to the PLXMon98 User's Manual for more information on how to communicate to the PLX engineering boards IOP applications.

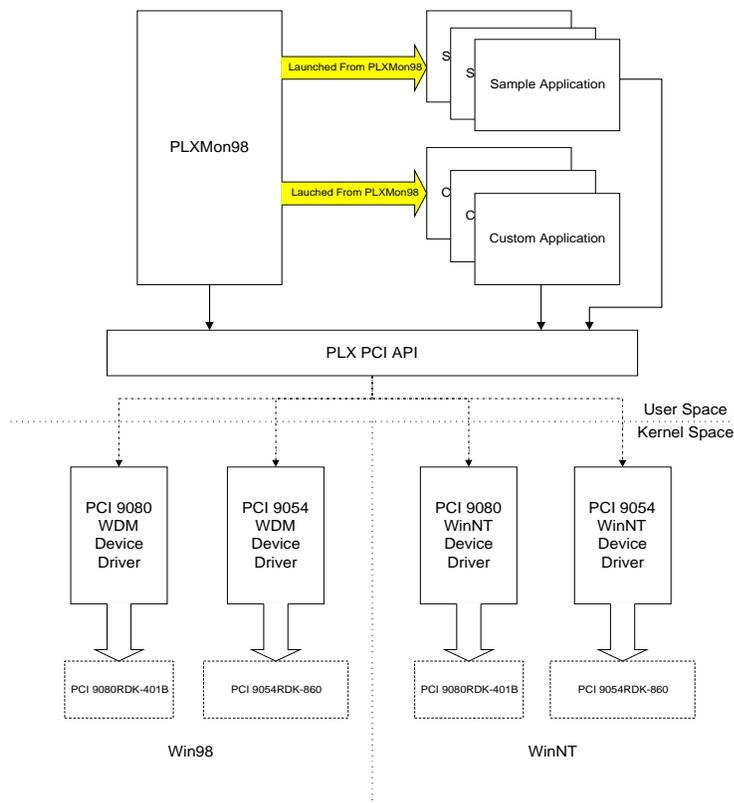
### 2.2.1.2.1 MiniRom Application

MiniRom is included in the PCI SDK to provide a good starting point for users who have an untested hardware device and for this reason, it is limited in features and functionality. It provides bare minimum boot up code for most boards. This application configures the microprocessor, the PCI IC, and proceeds to blink the LED that is connected to one of the PCI IC's USER pins. To use the MiniRom application, you should program the binary image into the FLASH using a FLASH chip programmer. Once the FLASH is programmed, reboot the board and if the LED blinks then the MiniRom application configured the board properly. If this test is successful, the FLASH can be reprogrammed with the PCI SDK PLXRom image (supplied with the PCI SDK).

*Note: This ROM application is provided as a bare bones ROM application useful for confirming the functionality of new boards. It does not contain any PCI SDK features that are described in any PCI SDK manual.*

## 2.2.2 Windows Based Software

### 2.2.3 Introduction



**Figure 2-2 Windows Host software Layout for PCI SDK v2.1**

The PCI SDK contains four distinct device drivers, an API, and a Windows monitor application (see Figure 2-2). They are as follows:

- Two PLX WinNT Device Drivers supporting the PCI 9080 and the PCI 9054;
- Two PLX WDM Win98 Device Drivers supporting the PCI 9080 and the PCI 9054 ;
- PCI API, a powerful API compatible with all PLX devices and PLX device drivers; and
- PLXMon98, a Graphical User Interface (GUI) application that can be used to monitor and modify PCI IC registers. It can also download software to a PLX engineering board, and communicate to the software running on the engineering board.

All Win32 executables included in the PCI SDK are located in the “<INSTALLPATH>\bin” directory. Furthermore, this path is added to the environment variables when the PCI SDK is installed.

For more information on PLXMon98, please refer to the PLXMon98 User's Manual.

### 2.2.3.1 Windows NT Device Drivers

The PCI SDK includes Windows NT device drivers for each PLX device. All device drivers are located in the <WINDOWS SYSTEM DIR>\system32\drivers directory. The naming convention used for the device drivers is: Pci<DeviceType>.sys. For example, the device driver for the PCI 9080 device is named Pci9080.sys.

#### 2.2.3.1.1 Starting And Stopping

There may be times when you will need to restart the Windows NT device driver. For instance, you must restart the device driver after changing the supported device list.

To restart the Windows NT device driver you should use the Windows NT Control Panel. The Control Panel contains a utility called ‘Devices’ that allows you to start and stop the device driver (see Figure 2-3).

*Note: Before stopping the device driver, all PCI SDK applications should be closed.*

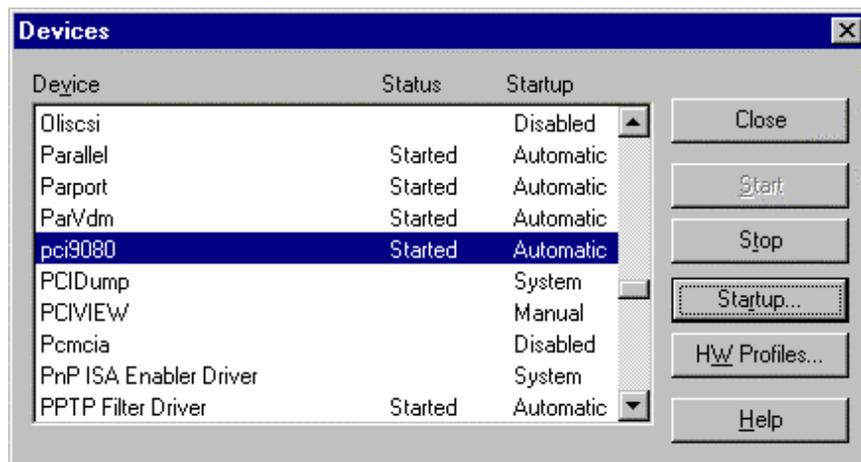


Figure 2-3 The Devices Utility Window.



By default, the device driver is configured to startup automatically at Windows NT boot time. You may configure the device driver to start manually by selecting the 'Startup...' button. However, no PCI SDK applications will function without the device driver being started.

You may also use the PCI SDK applet DriverWizard to restart the device drivers. Consult Section 2.2.3.1.4 for more details.

### 2.2.3.1.2 Event Logging

The Windows NT Device Driver has the capability to record errors into the Windows NT Event Viewer. When trouble shooting problems with the device driver it is recommended that the event viewer be used.

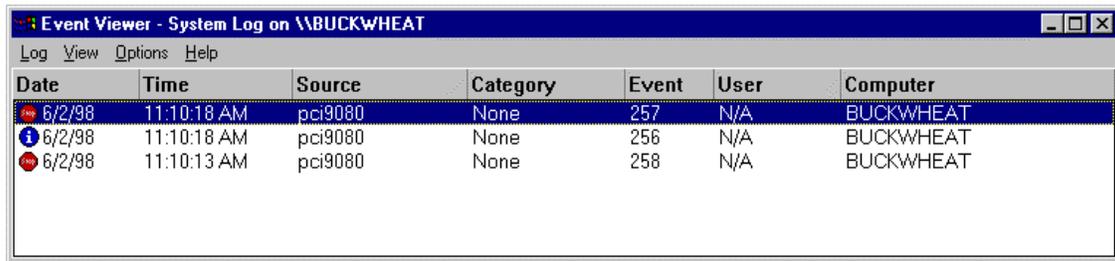


Figure 2-4 The Event Viewer Window.

Events can be viewed by selecting an event item. Figure 2-4 shows an example of the event viewer and Figure 2-5 shows details of an event.

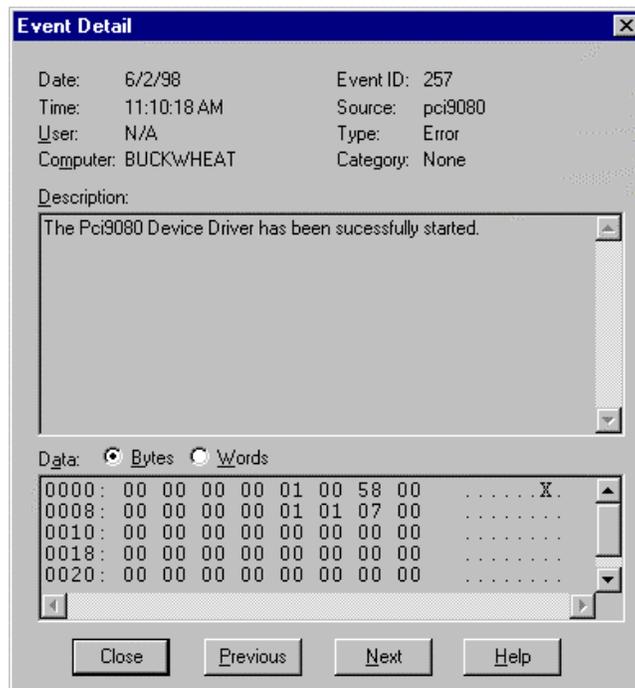


Figure 2-5 The Event Detail Window

### 2.2.3.1.3 Registry Configuration

Every Windows NT device driver requires an entry in the registry editor. The registry editor contains information required by the operating system as well as information required by the device driver. The name in the registry will be the same as the driver name. For instance, the `pci9080.sys` device driver has a `pci9080` registry item as shown in Figure 2.6. All device drivers are located under the `LocalMachine\System\CurrentControlSet\Services` tree.

The figures below show the required registry settings for the PCI SDK device drivers.

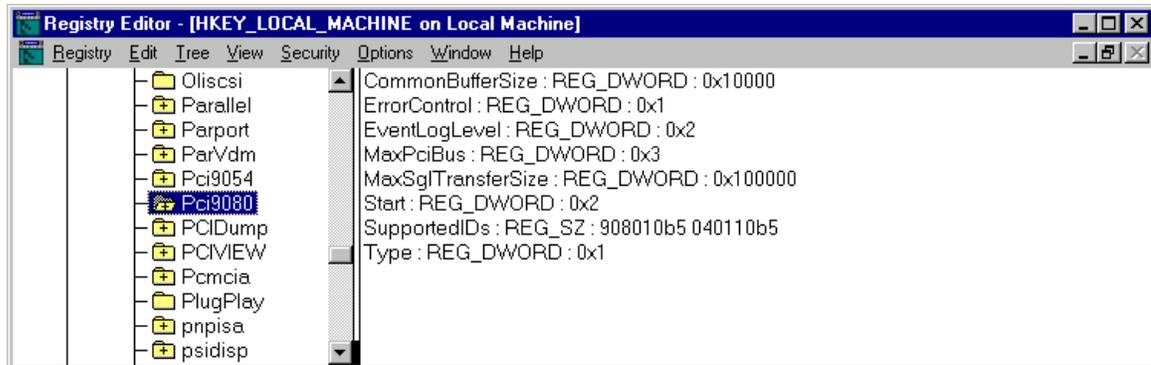


Figure 2-6 Registry Information for PCI 9080 Driver

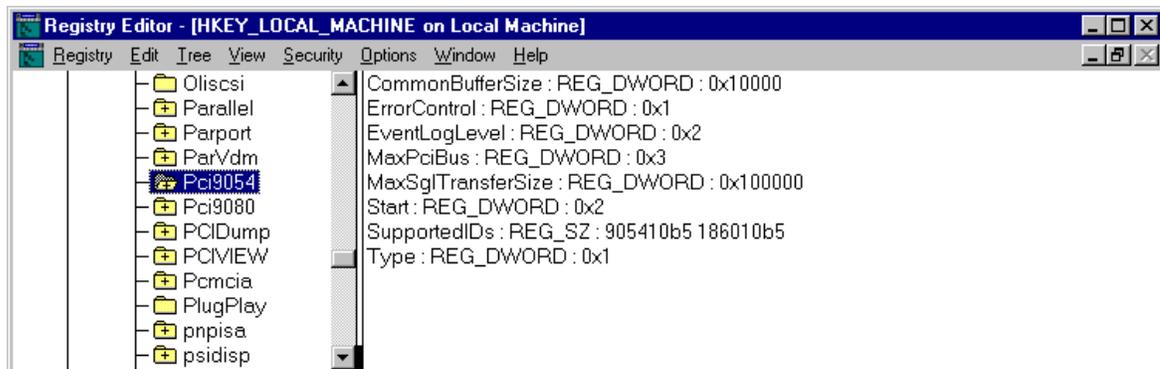


Figure 2-7 Registry Information for PCI 9054 Driver

**Note:** *The registry editor should only be modified by advanced users with administrative rights. Therefore, it is recommended that you do not change any values contained in the registry.*

The registry values for the device driver are:

- *CommonBufferSize*: This value sets the size of the user buffer (PlxMon98 hbuf). By default it is set to 64KB. *Warning: The device driver will try to allocate the size requested but if it can not due to a lack of system resources, it will decrement the size until it can allocate a buffer. You should use the `PlxPciCommonBufferGet()` API function to determine the actual buffer size.*



- *ErrorControl*: This value is required by the operating system and should not be modified.
- *EventLogLevel*: This value sets the event logging mode in the device drivers. If this value is 0 then events will not be logged. If this value is 1 then high severity events will be logged. If this value is 2 then all events will be logged.
- *MaxPciBus*: This value sets the highest PCI bus that the device driver will scan for PLX devices. By default it is set to 0x3.
- *MaxSglTransferSize*: This value sets the size of an internal buffer that is required for SGL and Shuttle DMA transfers. It should not be modified.
- *Start*: This value is required by the operating system and should not be modified.
- *SupportedIDs*: This value contains the Vendor Ids and Device Ids for the PLX devices that the driver supports. Users should use the PCI SDK applet DriverWizard to indirectly modify this field.
- *Type*: This value is required by the operating system and should not be modified.

#### 2.2.3.1.4 Driver Configuration

Before using the device driver with a custom engineering board, the driver must first be configured with the appropriate Vendor ID and Device ID. PLXMon98 has a hot-link to a PCI SDK utility called the Device Driver Wizard. This utility should be used to add or remove engineering board IDs to the appropriate device driver. It also lets you enable or disable the desired PLX device driver.

*Note: If you are not using the PCI 9054 or PCI 9080 device driver you should disable it using this utility. Before the settings take effect you must either restart your computer or restart the device driver.*

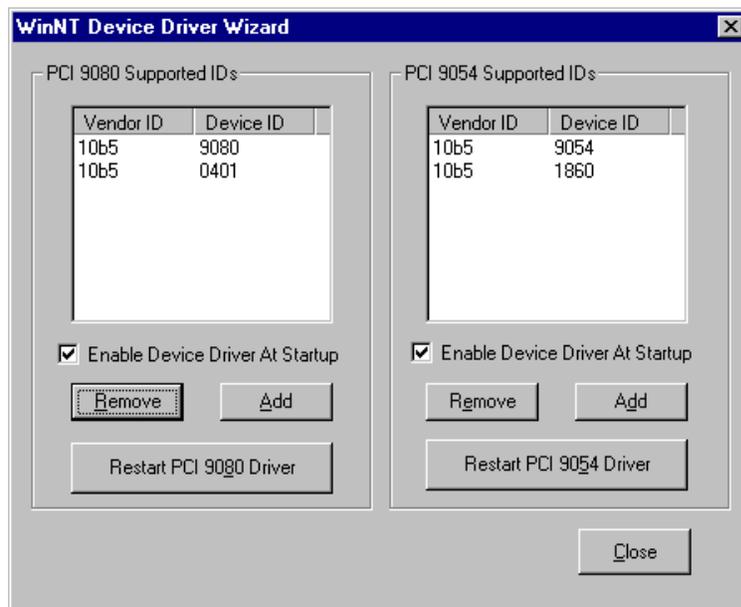


Figure 2-8 The PCI SDK Device Driver Wizard

### 2.2.3.2 Windows98 Device Drivers

The PCI SDK includes Windows98 device drivers for each PLX device. All device drivers are located in the <WINDOWS SYSTEM DIR>\system directory. The naming convention used for the device drivers is: Pci<DeviceType>.sys. For example, the device driver for the PCI 9080 device is named Pci9080.sys.

#### 2.2.3.2.1 Starting And Stopping

Unlike Windows NT drivers, Windows98 device drivers are started and stopped as needed by the operating system. The PLX device drivers are started when Windows98 detects a device that needs it. If, at a later time, the device is removed (by hitting the “Remove” button for a device from the Device Manager window), the device driver will be stopped unless there was another device that needs it.

There is no applet that controls the starting or stopping of a device driver.

#### 2.2.3.2.2 Event Logging

Event logging is not accessible on Windows98.

#### 2.2.3.2.3 Registry Configuration

Every Windows98 device driver requires an entry into the registry. The registry contains information required by the operating system as well as information required by the device driver. All device drivers are located using the registry editor under the: `HKEY_LOCAL_MACHINE\System\CurrentControlSet\Services\Class\Unknown\000X` tree, where 000X is the driver number within the “Unknown” class of drivers. The PLX device driver can be found within the Unknown class by looking at the `NTMPDriver` value of each key, which should describe the driver name (`pci9080.sys` or `pci9054.sys`, depending on the PCI IC in use).

**Note:** *The registry editor should only be modified by advanced users. Therefore, it is recommended that you do not change any values contained in the registry.*

#### 2.2.3.2.4 Known problems with the Windows98 device drivers.

Windows98 is not a mature operating system like Windows NT and it contains a few features that do not perform as expected. The following list contains some known features that affect the operation of the PCI SDK.

#### Scatter-Gather And Shuttle DMA

The Win98 device driver can periodically fail to transfer huge Scatter-Gather and Shuttle DMA data buffers. This affects the following PCI API functions: `PlxDmaSglTransfer()` and `PlxDmaShuttleTransfer()`. It is recommended that all data buffers used in the DMA transfers do not exceed 1 MB in size.

#### Changing Device Slot Numbers

If the engineering board is removed and placed into another PCI slot, Windows98 will consider the board as a “New Hardware device” and will show the “Add New Hardware Wizard”. You



must then repeat the Win98 device driver installation procedure (see section 2.1.4.2.2 for information on installing Win98 device drivers).

### **Using Power Management Features Of The PCI 9054**

Windows98 Power Management support was not complete when the PCI SDK was released. Therefore, the method recommended by Microsoft to change the power level of a device does not work as described in the Microsoft documentation. To overcome this problem two possible methods can be used:

1. Change the PCI 9054 power level using a different method than the one recommended by Microsoft. The intended behavior is obtained however this could cause problems in future releases of Windows98.
2. Leave the device driver sections as is, in hopes that Microsoft will correct the problem in future releases of Windows98.

The PCI SDK uses the first option in order to maintain Power Management capabilities.

## **2.3 Using The PCI SDK With A New Board**

The following steps can be used as a guide on how to use the PCI SDK with a new board.

1. Program the desired Vendor and Device IDs into the configuration EEPROM.
2. If using Windows NT, you will need to the new Vendor and Device IDs to the Supported Device List. To add support for new IDs, use the Device Driver Wizard utility (see section 2.2.3.1.4).
3. If using Windows98, you will need to consult section 2.1.4.2.2 to register the new device with the Windows98 device drivers.
4. Edit the MiniRom application as necessary to support the new engineering board.
5. Program the board's FLASH with the modified MiniRom application binary image file.
6. PLXMon98 can now access the engineering board's configuration EEPROM. Using PLXMon98's EEPROM Configuration window, customize the EEPROM settings for the new engineering board and reboot the system for the changes to take effect.
7. Try accessing IOP memory by using the Direct Slave memory accesses to the engineering board.

When the following steps have been performed and are working properly, modify the IOP Board Support Package (BSP) module to begin porting the PCI SDK to the new engineering board. Consult the PCI SDK Programmer's Manual for more information on porting the PCI SDK to new engineering boards.

## **2.4 Using PCI SDK API Libraries With Other Operating Systems And Compilers**

This version of the PCI SDK contains IOP API and PCI API libraries which are compiled to work with Windows operating systems (Windows NT and Windows 98) and with IBM PPC401GF and Motorola MPC860 embedded microprocessors. You can create new libraries to work with other operating systems and other compilers not supported by this SDK. Most libraries should be

recompilable without any errors. However with some compilers, compiler warnings or errors may arise. The following list can be used to determine some causes of warning or errors:

- Ensure that the IOP base data types for S8, U8, S16... are typecasted to the appropriate data types for the compiler used;
- If the embedded operating system/compiler supports 64 bit code ensure that the appropriate 64 bit data type is used for S64 and U64 data types; and,
- Some embedded operating systems/compiler may not provide functions that are needed by the PCI SDK. It may be necessary to recreate the operation of these functions or redirect these functions to similar functions provided by the operating system/compiler.

## 2.5 RTOS Support

If you are interested in developing a driver for a RTOS (Real Time Operating System), then you can use almost all of the “C” functions from this version of the PCI SDK. The IOP API library supplies a library of functions that can be called from within a RTOS. The RTOS code needs to call the `PlxInitApi()` function to initialize the PLX API. Next, you can proceed to complete your RTOS driver.

## 2.6 Source Code Availability

PLX provides the source code for the PCI API and IOP API under a separate software license agreement. Please contact PLX Technical support or send an email to [software@plxtech.com](mailto:software@plxtech.com).



## 3. PCI SDK Software Architecture Overview

### 3.1 Assumptions

This section discusses some assumptions made in the design of the PCI SDK.

#### 3.1.1 PCI SDK Assumptions

The assumptions for the PCI SDK are as follows:

- Mailbox register 0, 1, 6 and 7 are reserved for communication between PLXMon98 and the IOP software when downloading applications.
- After starting the host device drivers, mailbox register 0 will contain the address of the PCI common buffer, and mailbox register 4 will contain the size of the buffer.

#### 3.1.2 IOP API And IOP Software Assumptions

The assumptions for the IOP API and the IOP software are as follows:

- For the Back-End Monitor to function properly the IOP board must have one available serial port, configurable by the Board Support Package software;
- The data received by the serial port must be retrieved in a timely manner in order to eliminate any lost data;
- The initialization of the PCI IC is done by the IOP software only;
- The data expected by the application will not contain any data that could be interpreted by the Back-End Monitor as a command if they are linked in with the application;
- All IOP applications must be reentrant, cyclic and relinquish the processor periodically to avoid starvation of the Back-End Monitor (cooperative or non-preemptive multitasking);
- When an application is downloaded to the IOP RAM memory the IOP BSP must execute the *CheckPciDownloadToRam()* and the *CheckSerialDownloadToRam()* functions at microprocessor reset;
- The *BlinkLed()* function assumes that the LED is connected to the PCI IC's USERo pin; and,
- Supplied IOP Libraries are compiled for Big Endian processors only and contain no support for 64 bit processors or compilers. However, the source code does support Little Endian processors but it must be recompiled for that purpose.

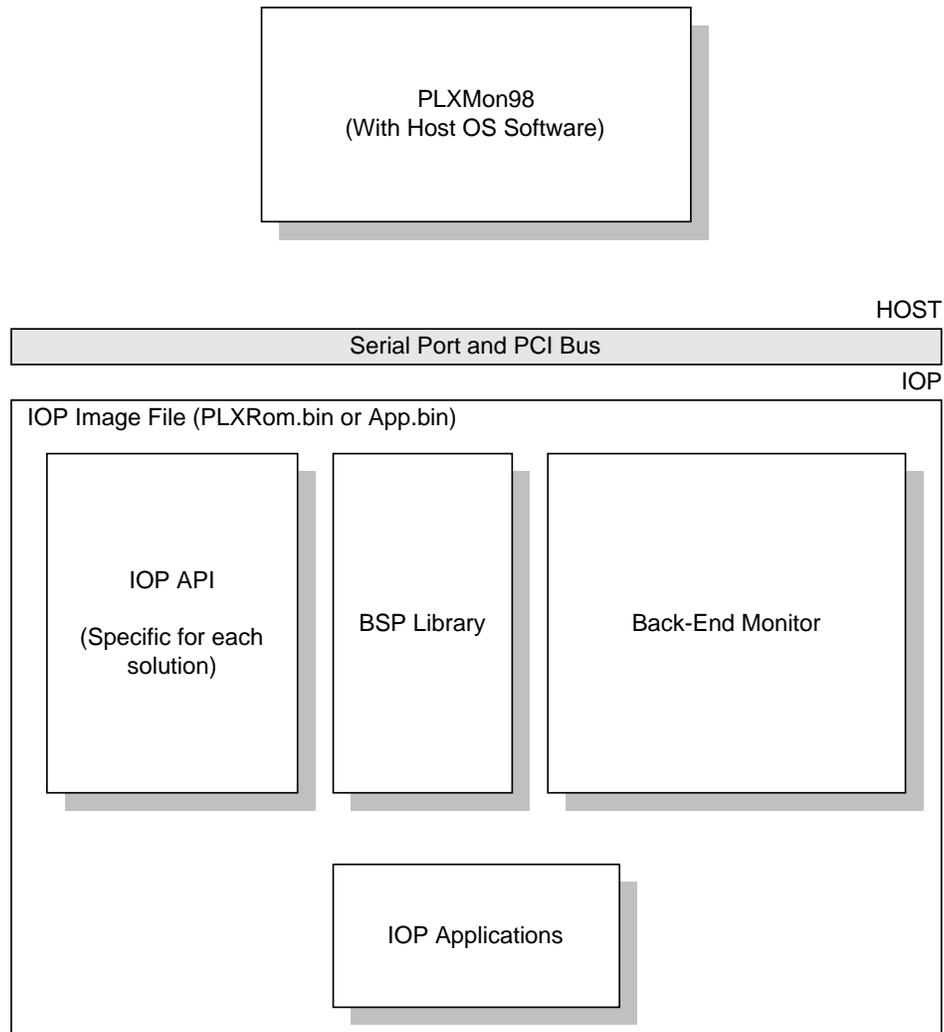
#### 3.1.3 PCI API and Win32 Software Assumptions

The assumptions for the PCI API and the Win32 software are as follows:

- All Win32 applications supplied with the PCI SDK will provide full functionality to all PLX registered devices; and,
- The doorbell interrupts, `QUERY_EEPROM_TYPE`, `DOORBELL_KERNEL_RESET`, `FLASH_READ`, and `FLASH_WRITE` are reserved for PCI SDK purposes.

### 3.2 Overview

The PCI SDK is separated into two distinct sets of software, the IOP software that runs on the engineering board and the PCI software that runs on the Windows host system (as shown in Figure 3-1). Each API contains distinct function calls that emphasize the features of the PCI IC. Some function calls look and react similarly in both API's but may have different parameter lists.



**Figure 3-1 The PCI SDK Software Architecture**

The IOP software contains three modules (excluding the IOP application), the IOP API library, the Board Support Package (BSP) and the Back-End Monitor. The IOP API is designed specifically for each PCI IC or for a combination of PCI ICs on one engineering board. The IOP API can be customized to run on any engineering board by modifying the Board Support Package. IOP debugging can be performed by PLXMon98 by including the Back-End Monitor into the IOP application.

The PCI software can be separated into two different packages, the Serial Communication package and the PCI Bus Communication package (see Chapter 3.5). The Serial Communication package accesses the information from the board using messages sent through the serial port of

the board. This communication method requires having the Back-End Monitor included into the IOP application running on the desired engineering board.

The PCI Communication package consists of two modules, being the PCI API Dynamic Link Library (DLL) and the Windows Device Driver. PCI applications make calls to the PCI API DLL where they are translated into the appropriate device driver calls. The device driver performs the requested action and provides a response, where appropriate, to the PCI API DLL. The status of the API call is passed back to the calling application.

### 3.3 Software Architecture

The PCI SDK software architecture is shown in Figure 3-1. The SDK software is divided into five major components:

- PLXMon98: this module includes the Host PCI API and device driver for PCI Bus communications, and the PLXMon98 Communications module for serial communications to the Back-End Monitor;
- IOP API Library: this library contains the code that performs the API functions and accesses the PCI IC;
- BSP Library: this library contains all board specific code, including the IOP bus memory map, the board and microprocessor initialization routines and the interrupt service routine for the PCI IC;
- Back-End Monitor: this module provides a monitor for debugging IOP applications which supports PLXMon98 through the serial port; and,
- IOP Applications: this module contains the main application for the engineering board and the IOP.

### 3.4 IOP Software Architecture

The IOP software architecture is separated into four modules, being:

- The IOP API library;
- The Board Support Package (BSP) library;
- The Back-End Monitor; and,
- The IOP application software.

The PCI SDK software architecture is shown in Figure 3-2.

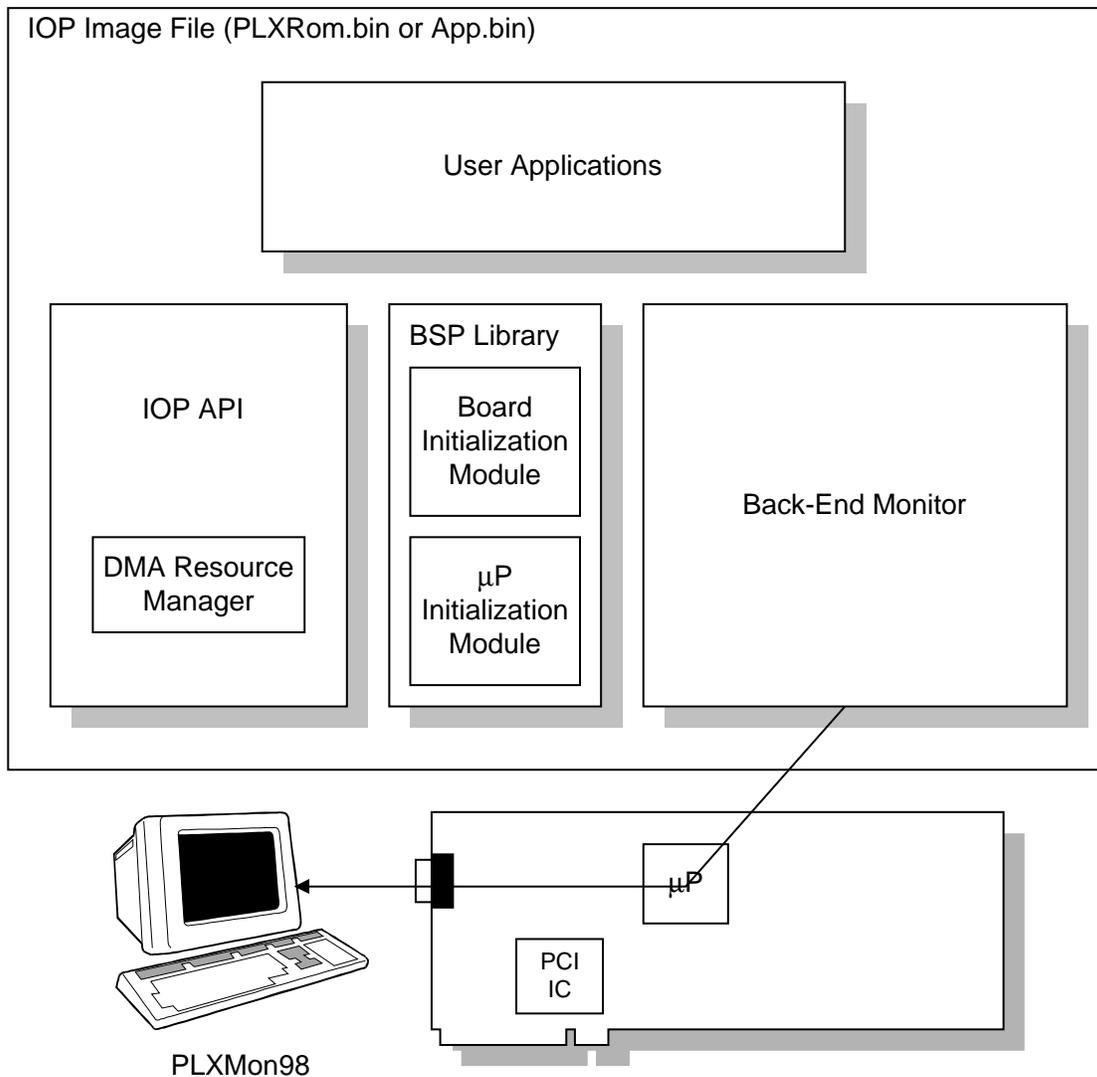


Figure 3-2 The IOP Software Architecture

### 3.4.1 Board Support Package (BSP) Library

The Board Support Package library contains all the information needed by the IOP API that is specific to the board. This library provides the necessary entry points needed to port the PCI SDK to new platforms. The BSP is composed of two main modules, being:

- The Microprocessor Initialization module; and,
- The Board Initialization module.

*Note: Prior to porting the PCI SDK to new boards an understanding of the BSP and its functionality should be acquired.*

### 3.4.1.1 Microprocessor Initialization Module

The microprocessor initialization module contains all the necessary information about the microprocessor required by the IOP API. Some of the information contained within this module are the microprocessor boot code, the main default interrupt service routine (ISR) for the PCI SDK and the default PCI IC interrupt trigger support functions.

#### 3.4.1.1.1 Microprocessor Boot Code

When the board is powered up, the microprocessor starts executing the boot code. This code initializes the microprocessor, configures the memory controller, copies data and code (if necessary for performance reasons) from the boot FLASH to RAM memory and brings the microprocessor to a ready state. The sequence of events is as follows:

1. The board is powered on.
2. The microprocessor begins at the reset address where it immediately jumps to the boot code.
3. The boot code configures the memory controller.
4. The data section and the code section (if necessary) of the boot application is copied to RAM memory.
5. The exception vector table is initialized.
6. Any other microprocessor specific initialization is done, such as configuring the endian registers, configuring the clock (if internal clocks are available), setting up any peripheral units internal to the microprocessor.
7. Once the microprocessor is initialized and is ready to run, the boot code jumps to the board initialization routine (see section 3.4.1.2).

*Note: The MiniRom application included in the PCI SDK provides a good starting point for users who have untested engineering boards. The application is limited in features and functionality and should be the basis for porting the PCI SDK to new engineering boards (see section 3.4.5.2 for more information).*

#### 3.4.1.1.2 Interrupt Service Routine

The interrupt service routine (ISR) provided in the BSP controls all interrupts generated by any PCI IC. The ISR is divided into one main routine with one function to service each interrupt trigger on the IC. When an interrupt is generated, the main ISR determines the interrupt trigger and calls the appropriate interrupt trigger service routine to service the interrupt.

This method allows modification of individual interrupt trigger service routines or modification of the main interrupt service routine to customize the handling of interrupts for each application.

#### 3.4.1.2 Board Initialization Module

The Board Initialization module contains information on the features of the board and the board initialization routine. Some of the information it provides include the memory map of the IOP bus, specifically where the following devices are located in memory:

- SRAM address and range;
- DRAM address and range;



- SDRAM address and range;
- PCI IC Register Base address;
- UART ports (Control/Status, Data);
- Flash Memory address and range;
- Direct Master Memory Remap address and range;
- Direct Master I/O Remap address and range; and,
- Boot address.

The PCI SDK needs to know the endianness for each memory region. If the IOP bus is less than 32 bits wide, the PCI SDK needs to know how the IOP bus is connected to the PCI IC (specifically which bytes and byte lanes are used by the IOP bus).

The Back-End Monitor needs to know about the UART. The necessary UART ISRs and serial communication functions are included in this module.

#### **3.4.1.2.1 Board Initialization Routine**

The board initialization routine contains the necessary API functions to configure and initialize the PCI IC, the IOP API library, the Back-End Monitor and any other device on the engineering board. This function is called from the microprocessor initialization routine (the microprocessor boot code, see section 3.4.1.1.1) at start up. The board initialization sequence is as follows:

1. Initialize the PCI IC. A list of IOP API initialization functions is provided with each of its parameters set to the PCI IC's default values (set by calling the *PlxInitApi()* function).
2. Change the default values for the parameters as necessary before calling the respective IOP API initialization function.
3. Set the Local Init Status bit when the PCI IC is initialized (this asserts the NB# pin low). This bit allows the PCI BIOS to access the PCI IC. Once the PCI BIOS has assigned the appropriate values to the PCI IC's configuration registers, the PCI IC is completely initialized and is ready to run.
4. Initialize the different debugging levels of the Back-End Monitor with the necessary board specific information.
5. Initialize any other peripheral on the board.
6. Connect the ISRs to the appropriate interrupt lines of the microprocessor.
7. Initialize the application, if necessary, once all devices on the engineering board have been initialized and are operational.
8. Jump to the main application routine.

#### **3.4.1.3 The Main() And AppMain() Functions**

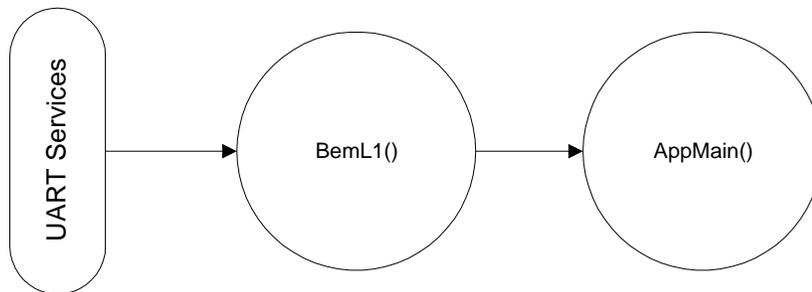
The BSP Library contains the *main()* function for any application using the PCI SDK. This function controls the operation of the IOP API. The function starts by initializing the microprocessor and its peripherals, the PCI IC (when there is no EEPROM connected to it), the UART IC, and the Back-End Monitor (when in use). The function proceeds to test the available memory on board and begins the main application section.

The main application section consists of a loop that allows execution of several tasks on a round-robin priority scheme. Each task is allowed as much time as it needs to run (non-preemptive and no priority levels). This loop runs without interruption in a cyclic fashion and therefore all the tasks must eventually return (tasks must be reentrant).

The Back-End Monitor can be used to filter the stream of data, supplied by the UART Services functions, to help in debugging new applications. The UART Services functions receive a stream of data from the RS-232 port on the engineering board and buffer it. This stream can be received by any task requiring data from the serial port.

The Back-End Monitor, *BemL1()*, does simple debugging. This monitor task is used with PLXMon98's serial debugger support turned on. The *BemL1()* monitor task only accepts three commands, read and write to an IOP memory location, and a board reset command.

With the stream of data received from the serial port (see Figure 3-3), the *BemL1()* task receives and parses through it, searching for commands. When *BemL1()* finds a command that it recognizes, the monitor removes the command from the stream, reacts accordingly to the command and returns a response when appropriate. Once the stream of data has been completely parsed and all *BemL1()* commands have been removed from it the filtered stream is made available to the next task wishing data from the serial port. The filtered data stream is received by the application, *AppMain()*.



**Figure 3-3 The Data Stream Flow Diagram.**

The filtering of the data stream can be bypassed by a task at any point in time by calling the UART Services functions. An example of this feature is when a task starts an application download to memory. The application binary file being downloaded may contain data that looks similar to a command for the *BemL1()* task. If the *BemL1()* task is retrieving the data from the UART Services functions then some information about the application will be lost. Therefore, while the task downloads an application, it calls directly *PlxGetChars()* to retrieve the unfiltered data from the UART IC until the application is completely downloaded. Once the download is complete, the task returns control to the BSP Module's main loop to allow other tasks to run.

This feature should be used with caution however because it directly affects the operation of the other tasks dependant on the data stream coming from the serial port. When an application requests unfiltered data the task calls *PlxGetChars()* function and this function returns an unfiltered data stream. This task should not return to the main loop (within the BSP Module) to continue processing of debug commands until all the necessary unfiltered data has been received by the application. By doing this the Back-End Monitor task will not scan through the data and remove command data from the stream that was not intended to be a command for the debug monitors.



## 3.4.2 IOP API Library

The IOP API library contains the code for all the documented API functions. This code is standard for all IOP applications and is independent of the board configuration. The code directly calls the PCI IC (no intermediary functions).

*Note: Each PCI IC has its own IOP API library specifically designed to complement its features. To implement more than one PCI IC on one board, a new library must be created. This library would combine the features of each IC and have new functions to accent the features achieved by grouping the PCI ICs.*

### 3.4.2.1 DMA Resource Manager

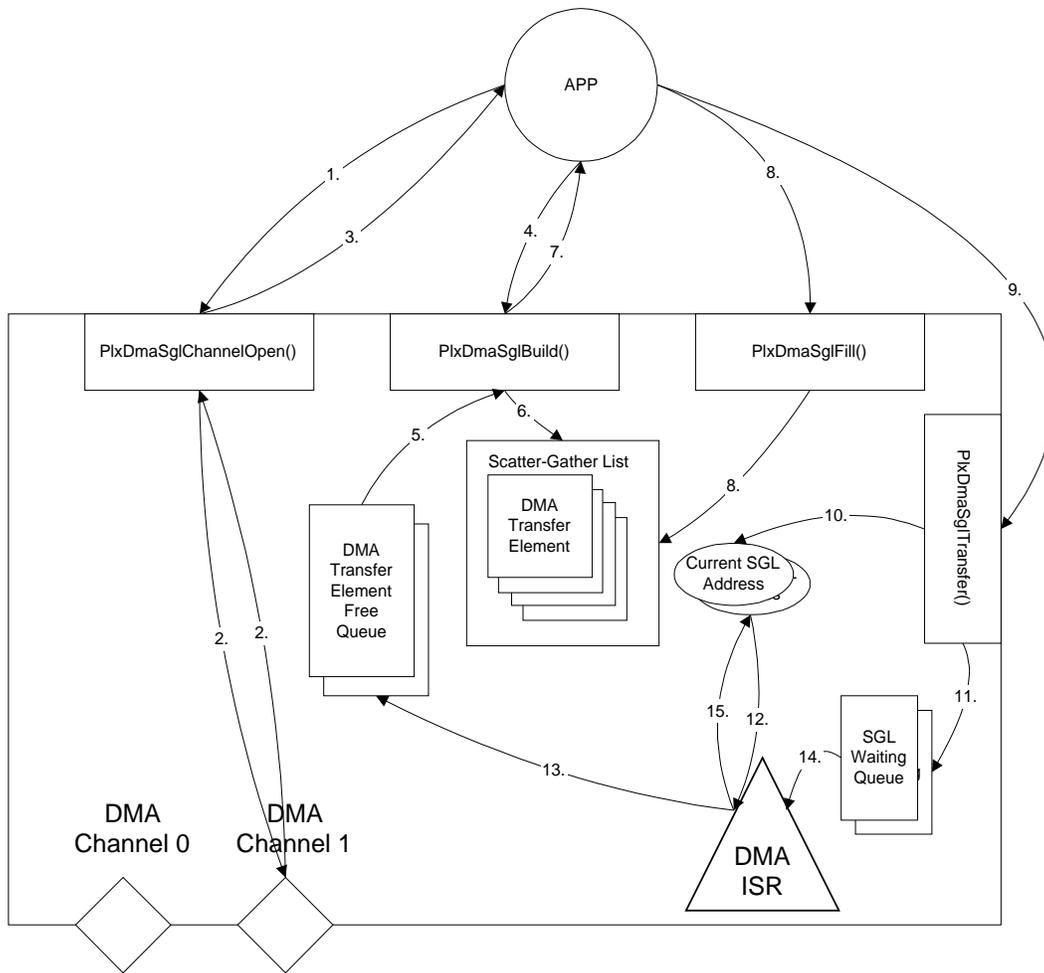
The IOP API supports three different DMA transfer types and manages the DMA resources. The supported DMA transfer types are:

- Scatter-Gather DMA: Transfers data using Scatter-Gather Lists and can transfer several blocks of data at a time (formally called chaining DMA);
- Block DMA: Transfers data one block at a time; and,
- Shuttle DMA: a circular Scatter-Gather DMA transfer.

The Scatter-Gather DMA transfer is most commonly used of all DMA transfers. This method supports DMA transfers where either the source or destination memory locations are not contiguous (this is common with most operating system memory allocation) the best. By grouping multiple DMA transfer requests, the IOP application is interrupted less often providing improved performance.

The Block DMA transfer is used primarily for single DMA transfers and where the number of transfer requests is small.

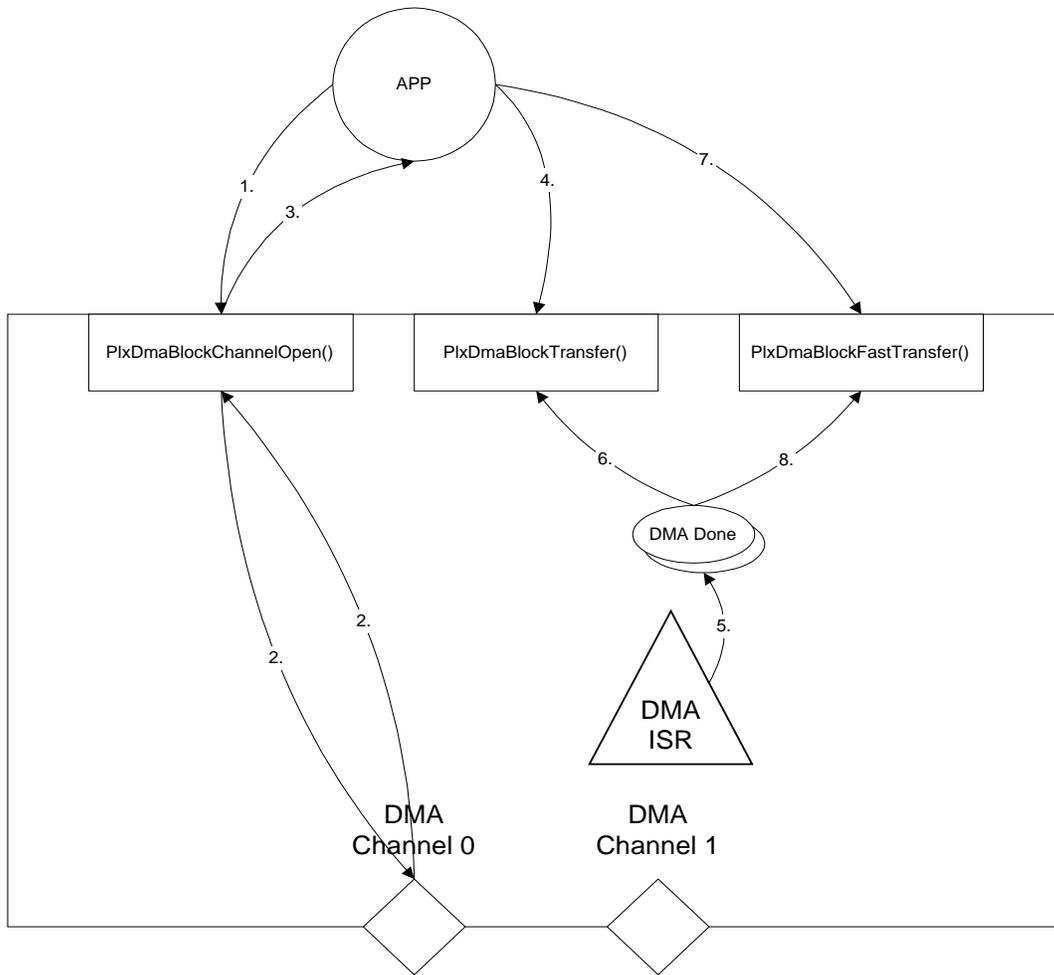
The Shuttle DMA Transfer is best used when the data transfers are repetitive (where the source and destination locations remain relatively constant but the transfer direction may switch or the transfer size is different).



**Figure 3-4 Scatter-Gather DMA Flow Diagram**

### Scatter-Gather DMA Transfers

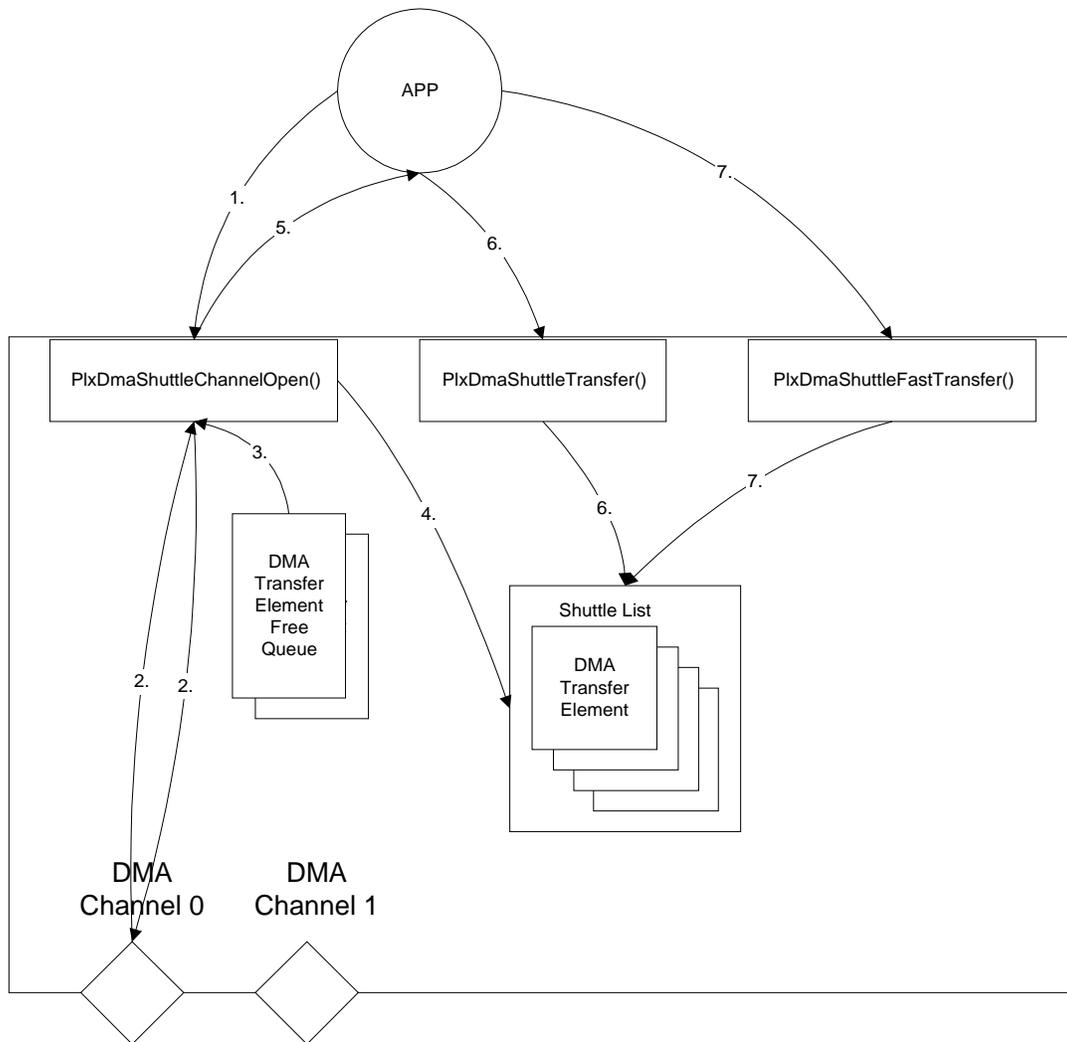
In Scatter-Gather DMA transfers (see Figure 3-4), a SGL DMA channel is opened (steps 1-2). With a successful return (step 3), a Scatter-Gather List (SGL) is acquired from the DMA resource manager (steps 4-6) by calling `PlxDmaSglBuild()` and a handle to a list of DMA transfer element addresses is returned (step 7). The DMA transfer elements are programmed with the appropriate source and destination data addresses, the transfer size and the DMA transfer descriptors by calling `PlxDmaSglFill()` (step 8). The SGL is passed to the `PlxDmaSglTransfer()` function (step 9). If there is not a SGL currently executing on the DMA channel, this function programs the list address into the DMA descriptor register for the opened DMA channel and also into the Current SGL Address buffer (one buffer for each DMA channel), and the DMA transfer is started (step 10). If there is a SGL executing then this function places the SGL address into the SGL Waiting Queue (one queue for each DMA channel) (step 11). When the SGL currently executing is completed the ISR reads the Current SGL Address buffer (step 12) and frees the DMA transfer elements for this SGL to the DMA Transfer Element Free Queue (one queue for each DMA channel) (step 13). The ISR then removes all the current SGL entries in the SGL Waiting Queue and joins them together (step 14). The new SGL address is placed into the Current SGL Address buffer and it is placed and started on the DMA Channel (step 15).



**Figure 3-5 Block DMA Transfer Flow Diagram**

### Block DMA Transfers

In Block DMA transfers (see Figure 3-5), a Block DMA channel is opened (steps 1-2). With a successful return (step 3), the `PlxDmaBlockTransfer()` function is called with the appropriate source and destination data addresses, the transfer size and the DMA transfer descriptor (step 4). This function checks the status of the DMA channel to determine if there is a transfer in progress by checking the DMA Done flag. If there is a transfer in progress then the function returns the “In Progress” error code. Otherwise the DMA data is programmed into the DMA registers for the DMA channel and the transfer is started. When the transfer is completed, the ISR will set the DMA Done flag (step 5). If the `PlxDmaBlockTransfer()` function is set to not return immediately then this function polls the DMA Done flag (step 6) and when the flag is set the function will return. The `PlxDmaBlockTransferRestart()` function is used to quickly restart a Block DMA transfer that was pre-programmed with the `PlxDmaBlockTransfer()` function (step 7). The only parameter needed is the transfer size. All other DMA information is reused from the previous transfer. This function also supplies an immediate return feature where, when the parameter is set to FALSE, the function polls the DMA Done flag (step 8) until it is set then returns.



**Figure 3-6 The Shuttle DMA Flow Diagram**

### Shuttle DMA Transfers

In Shuttle DMA transfers (see Figure 3-6), a Shuttle DMA Engine is started by opening a Shuttle DMA channel (steps 1-2). A number of DMA transfer elements are acquired from the DMA resource manager (step 3). The DMA transfer elements are linked to create a Shuttle List (step 4). This Shuttle List is placed on the opened DMA channel and is started thereby starting the Shuttle DMA Engine. A list of the DMA transfer element addresses is returned to the application (step 5). From this point, each DMA transfer element of the Shuttle List can be treated as a unique DMA channel. To start a transfer, the `PldmaShuttleTransfer()` function is called with the appropriate source and destination data addresses, the transfer size and the DMA transfer descriptors (step 6). This function checks the status of the Shuttle DMA channel to determine if there is a transfer in progress by checking the transfer size for the given DMA transfer element. If there is a transfer in progress then the function returns the “In Progress” error code. Otherwise the DMA data is programmed into the DMA transfer element provided by the application and the transfer is started. When the transfer is completed the PCI IC (through the PLX DMA Descriptor Write Back Feature) sets the transfer size for the completed DMA transfer element to zero. If the



*PlxDmaShuttleTransfer()* function is set for blocking then this function will poll the DMA transfer element's transfer size and when the size is set to zero the function will return. The *PlxDmaShuttleTransferRestart()* function is used to quickly restart a Block DMA transfer that was pre-programmed with the *PlxDmaShuttleTransfer()* function (step 7). The only parameter needed is the transfer size. All other DMA information is reused from the previous transfer. This function also supplies a blocking feature where it polls the DMA transfer element's transfer size until it is set to zero.

### 3.4.3 Back-End Monitor

The Back-End Monitor (BEM) provides features that help in the debugging of IOP applications. The monitor supports PLXMon98 serial command passing to the IOP application from the serial port of the engineering board. This monitor only supports three commands, reading and writing to IOP memory locations (these commands support different data sizes) and reset the IOP software. These commands provide a generic interface for any application. PLXMon98 uses this monitor to retrieve data from the IOP. In normal operation, this monitor accesses the UART Services functions to get a stream of data that has been received by the UART IC. The monitor extracts commands (that the monitor recognizes) from the data stream, performs the necessary action and provides a response when appropriate. This monitor provides the filtered data stream to the next task requiring serial data in the daisy chain.

There are times when a task may not want other tasks to extract data (or commands) from the data stream. This can be done by accessing the UART Support functions directly. A task wishing to receive raw data, bypasses the previous task in the daisy chain and calls *PlxGetChars()* to retrieve an unfiltered data stream. If a task chooses to access the unfiltered data stream it should take all the data necessary to perform the action and, only once the action is complete, return control back to the main routine (contained within the BSP).

The next application in the daisy chain, if required, retrieves the filtered data stream from the BEM monitor. The application can do whatever it needs to do with the data. The application can choose to provide a filtered stream of data from what is left over from its parsing of the data stream so that the data stream can be passed down to the next task in the chain.

#### 3.4.3.1 Back-End Monitor Serial Protocol

The Back-End Monitor (BEM) can recognize three different commands coming from PlxMon98: reset the IOP microprocessor, read a memory location and write to a memory location. They all use a serial protocol that is discussed in the following sections.

Some commands use parameters. Parameters listed are normally necessary for the command except when a parameter is within square braces ('[' and ']'). These parameters are optional to the command.

Parameters listed with the vertical bar ( '| ' ) indicate that "one or the other" parameter must be provided, but not both.

Carriage returns are noted as <ENTER>.

Note that all commands should be lowercase. The Back-End Monitor does not recognize uppercase commands except in hexadecimal value parameters.

### 3.4.3.1.1 Reset IOP Microprocessor Command

**Syntax:**

~plx!

**Response from the BEM:**

No response

### 3.4.3.1.2 IOP Memory Read Command

**Syntax:**

~plxr[b|w|l|d] address<ENTER>

Parameter	Description
address	Address of the memory location to read. The address must be a hexadecimal number such as f0000.

**Response from the BEM:**

~plx value<ENTER>

Parameter	Description
value	Value read at the selected memory location. The format is a hexadecimal number preceded by "0x", such as "0x7fff".

**Description:**

This command reads an 8 bit byte (b), a 16 bit word (w), a 32 bit double word (d) or a 64 bits long double word (l) at the specified address and returns the value read.

### 3.4.3.1.3 IOP Memory Write Command

**Syntax:**

~plxw[b|w|d|l] address value<ENTER>

Parameter	Description
address	Address of the memory location to write to. The format must be a hexadecimal number such as "f0000".
value	Value to write at the memory location. The format of the value must be a hexadecimal number such as "905410b5".

**Response from the BEM:**

No response

**Description:**

This command writes an 8 bit byte (b), a 16 bit word (w), a 32 bit double word (d) or a 64 bit long double word (l) at the specified address.



### 3.4.4 Methods For Debugging IOP Applications

The PCI SDK supports two methods for debugging IOP applications, being:

- Win32 Debugging: Using PLXMon98. This method assumes that there is no IOP application running on the engineering board. With newer engineering boards, this method provides the preliminary debugging and validation of new engineering boards.
- PLXMon98 with the BEM: With the BEM linked into the IOP application, PLXMon98 can communicate to the engineering board through the PC's COM port to the serial port on the engineering board. PLXMon98 can be setup to communicate to the engineering board or IOP application using either the serial port or the PCI bus.

#### 3.4.4.1 Operation Of The Back-End Monitor In A System

This section describes how the BEM can be used on an engineering board and how it affects system performance.

The Back-End Monitor combinations are as follows:

1. *AppMain()* only: the IOP application is running without any BEM tasks; and,
2. *BemL1()* and *AppMain()*: the IOP application is running with BEM debugger.

**Method 1:** This method is used once the application has been fully tested and is working properly. There is no monitor tasks running so this method provides the best performance for the application. PLXMon98 can be used to debug the application if the engineering board is inserted into a free slot in the host system's PCI Bus and PLXMon98's PCI Communication is turned on.

**Method 2:** PLXMon98 is used to debug the application through the serial port. IOP application performance will be affected using this method because the BEM monitor is processing commands and copy data to and from different memory buffers. There is a possibility of lost data destined for the IOP application. If IOP application data matches BEM commands, the monitor will remove them from the serial data stream. When the IOP application requires data that could be captured by the monitor, the IOP application should access the UART Services module directly, bypassing the monitor (done by calling *PlxGetChars()*).

### 3.4.5 IOP Applications

All IOP applications are connected to the IOP API, the BSP and the Back-End Monitor to create the binary image. This image is then programmed into FLASH memory, or downloaded to RAM memory and executed.

All IOP applications have an *AppMain()* function which is the main application function. The *main()* function is kept within the BSP module. This limitation is imposed on all applications because of how the Back-End Monitor (BEM) is implemented. The BEM needs to run periodically to operate properly. Since there can only be one execution thread running at one time, a cyclic thread is created using the *main()* function. This thread loops forever calling the BEM and then the main application function sequentially (cooperative multitasking or non-preemptive multitasking). The *AppMain()* function should be cyclic in nature and should return control periodically back to the *main()* function.

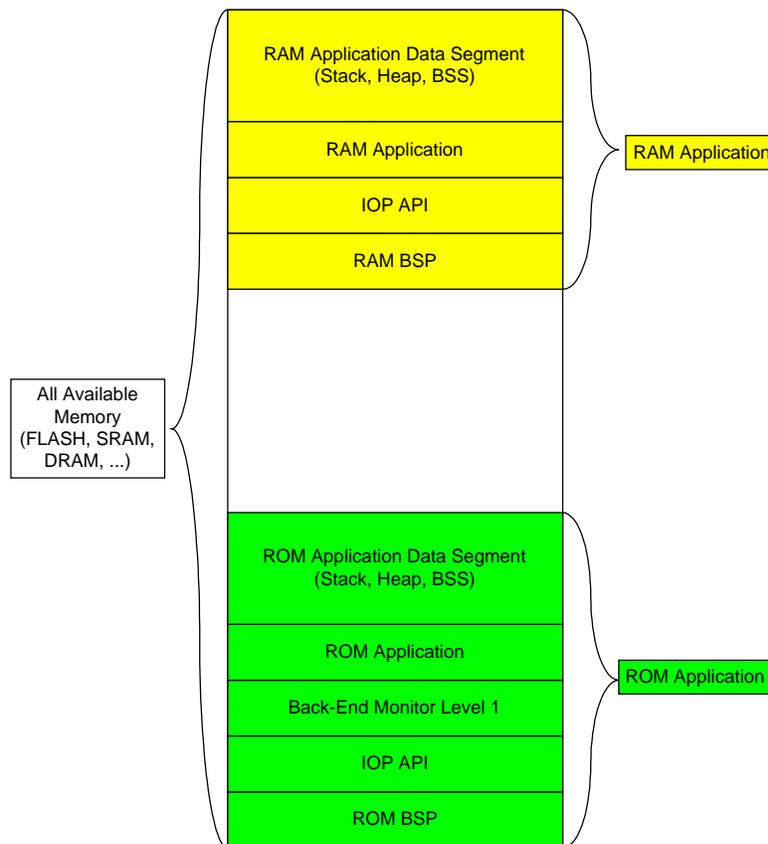
### 3.4.5.1 IOP Memory And IOP Applications

IOP applications running in ROM or in RAM use memory in different ways. When an IOP application is running in ROM the application contains all the modules it needs, such as the Back-End Monitor. A ROM application contains:

- The main IOP application module;
- The IOP API;
- The BSP module; and,
- The Back-End Monitor (debug version of ROM application).

Figure 3-7 shows how the ROM application uses memory.

IOP RAM applications are built differently from IOP ROM applications. The IOP RAM applications look very similar to IOP ROM applications from a source code point of view but they differ when the IOP RAM application is linked to the libraries. IOP RAM type applications borrow the Back-End Monitor from the resident IOP ROM application. The size of IOP RAM applications are normally smaller because a lot of the code used by the IOP RAM application resides in the IOP ROM application. Therefore the IOP ROM application on the engineering board must have the modules needed for the IOP RAM application and the IOP ROM application must provide the links to those modules. The BSP provided with the PCI SDK contains the links for IOP RAM based applications.



**Figure 3-7 IOP Memory Diagram**



### 3.4.5.2 MiniRom Application

MiniRom is included in the PCI SDK to provide a good starting point for users who have an untested engineering board and for this reason, it is limited in features and functionality. It provides bare minimum boot up code for most engineering boards. This application configures the microprocessor, the PCI IC, and proceeds to blink the LED that is connected to one of the PCI IC's USER pins. To use the MiniRom application, program the binary image into the FLASH using a FLASH chip programmer. Once the FLASH is programmed reboot the engineering board, and if the LED blinks then the MiniRom application configured the engineering board properly. If this test is successful, the FLASH can be reprogrammed with the PCI SDK PLXRom image (supplied with the PCI SDK).

*Note: This IOP ROM application is provided as a bare bones IOP ROM application useful for confirming the functionality of new engineering boards. It does not contain any PCI SDK features that are described in any PCI SDK manual.*

### 3.4.6 Porting The PCI SDK To New Platforms

All information needed to port the PCI SDK to new platforms is contained within the BSP module. Some of the information contained within the BSP include:

- The memory map of the IOP bus;
- The microprocessor boot code;
- The PCI IC interrupt service routine;
- The UART interrupt service routine;
- The board initialization routine; and,
- The board and/or application specific controls for the IOP API and the Back-End Monitor.

The IOP API and the Back-End Monitor need to know where certain devices are located on the IOP bus, such as the PCI IC, DRAM, SRAM and the UART. These values need to be updated when creating an application for new boards.

When the microprocessor is changed on a board, the microprocessor boot code must be modified to support the new microprocessor. This boot code is provided within the BSP module.

Most interrupt service routines are customized to the application. To customize the PCI ISR for an application, either modify the interrupt trigger service routines or modify the main ISR.

The Back-End Monitor relies on the UART ISR to send and receive data from the serial port. Modify the UART ISR to support the UART on the board.

To initialize the PCI IC modify the parameters for the IOP API initialization functions contained within the board initialization routine.

Within the BSP, there are some control parameters for the IOP API and the Back-End Monitor that can be modified to improve performance of the PCI SDK. These parameters are platform and application dependent and can affect the operation of the application differently on different systems.

### 3.4.7 Support For Multiple PCI ICs On One Board

Each PCI IC has its own IOP API library. When two or more ICs are present on one engineering board, a new IOP API library must be created. This library will contain the normal API functions (defined in this document) and some new or modified functions that represent new features made available by combining the features of the multiple ICs. Some multiple IC libraries will be available (for the more popular implementations), however it will be up to the designer to create his/her own library for multi-IC combinations not currently supported.

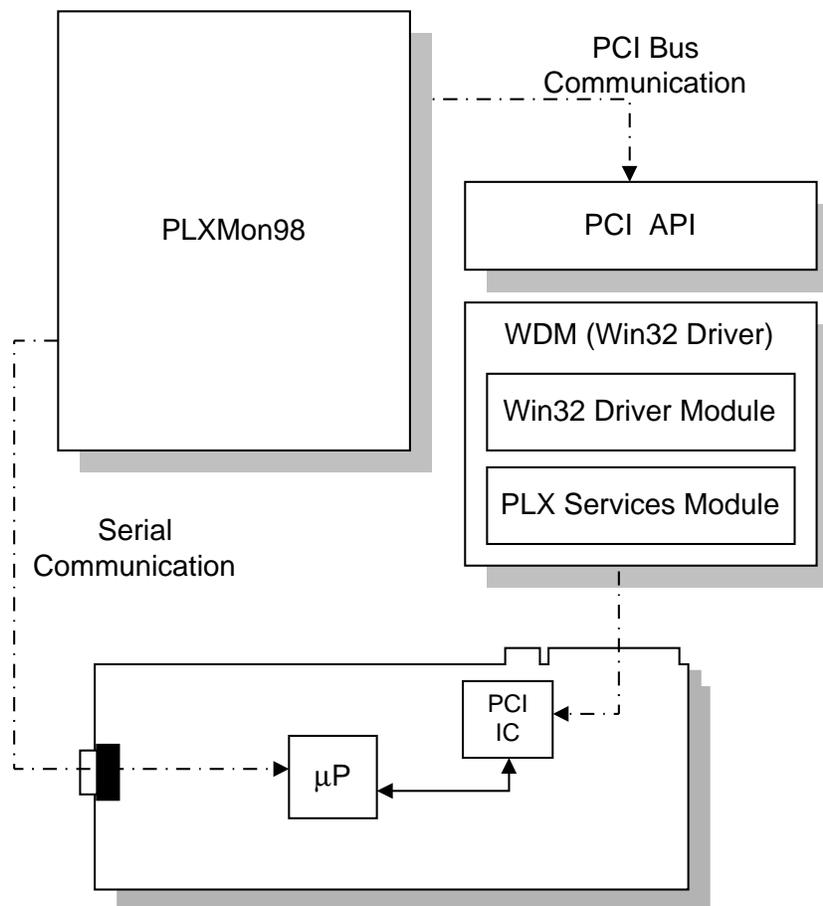
## 3.5 Host Win32 Software Architecture

This section describes the Win32 software provided in the PCI SDK.

### 3.5.1 PLXMon98

PLXMon98 can communicate with PCI devices via two different paths (see Figure 3-8):

- Direct Serial Communications;
- Host API/Device Driver Interface.



**Figure 3-8 The Host Software Architecture**



### **3.5.1.1 Serial Communication**

This method of communicating with a PLX engineering board is mainly used for debugging purposes. While a custom host Win32 device driver is being created, it is helpful to be able to read and write values directly to and from the engineering board.

If PLXMon98 is set to serial mode, it calls functions that reside in the PLXMon98 Communications Module. It is the responsibility of the PLXMon98 Communications Module code to convert the valid PLXMon98 commands into a serial data stream. The protocol used in passing the data is based on an ASCII translation scheme (for more information on the serial protocol, see section 3.4.3.1). This stream of data is sent to the IOP application. The Win32 operating system provides a device driver to control the serial port. The Win32 SDK provides services to access this device driver.

When the data arrives, the engineering board's microprocessor must have a means of handling the incoming data. The Back-End Monitor contains functions that are hooked via an interrupt, so when data does arrive, they are called (when the UART generates an interrupt) to retrieve the data from the UART. The Back-End Monitor decodes the command and data, and acts on the command and returns a reply. If the data received by the Back-End Monitor is not a command the data is queued for the IOP application.

### **3.5.1.2 PCI API/Device Driver Communication**

PCI Bus Communication is performed using the PCI API DLL and the Win32 device driver supplied with the PCI SDK.

#### **3.5.1.2.1 PCI API Library**

The PCI API consists of a library of functions, from which multiple PCI engineering boards can be accessed and used. The PCI API provides API function groups, which manage the features of each PCI IC. Groups such as DMA access, direct data transfers, and interrupt handling contain functions that can be universal to any PCI engineering board.

The PLXMon98 application makes extensive use of the PCI API functions. For the most part, the PCI API's purpose is to translate application functions calls and send them to the appropriate device driver. The only functionality present in the PCI API is to manage the various device drivers. This includes opening, closing and searching for devices that are present on the PCI bus.

#### **3.5.1.2.2 Win32 Device Driver**

The device driver's role in the system is to store device data within the kernel and to execute the commands given to it from the PCI API. The device driver can be used as a framework to create custom software for managing PCI devices as well.

The Win32 Driver Model (WDM) is a new platform for developing device drivers on the Windows 98/NT 5.0 operating systems. It is very similar in device driver architecture to that of Windows NT 4.0 and allows the creation of one device driver that can be used for both operating systems without any porting or modifications.

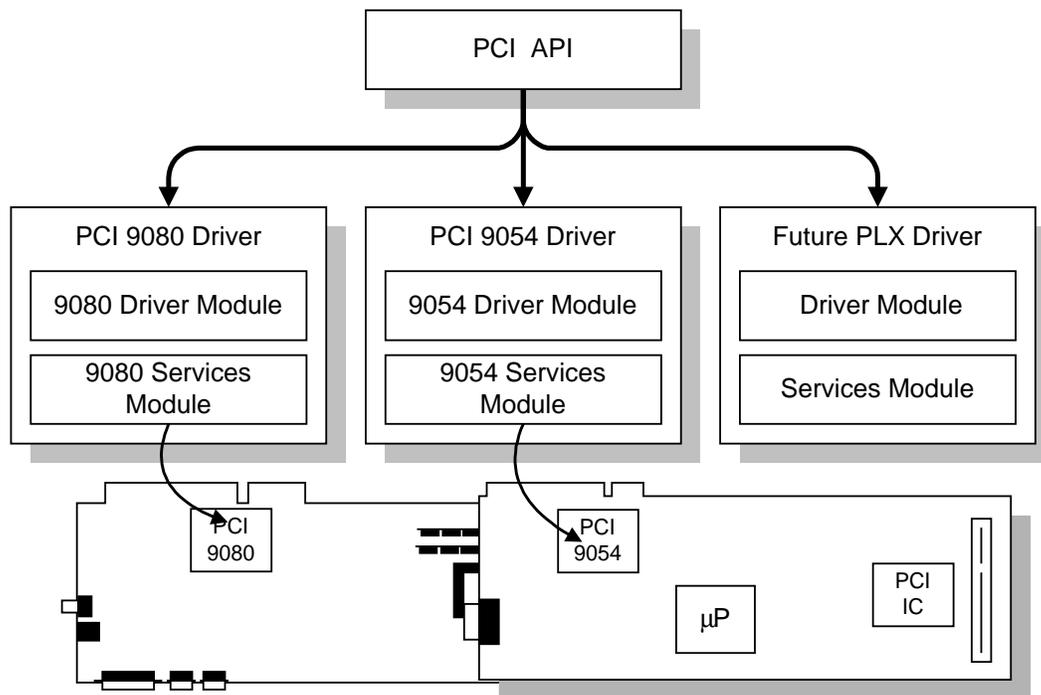
The architecture of the device driver is designed to reduce the time needed to create a new device driver for customer boards that contain a PLX device. By modifying the source code provided for the device driver, a new custom device driver can be created in minimal time.

### 3.5.2 Win32 Applications And The PCI SDK

All Win32 applications connect to and use the PCI API DLL. The Win32 application can communicate to any PCI device with a PLX IC by using the PCI API DLL. Each Win32 application can be created like any other Windows application. For more information on creating a Win32 application using the PCI API DLL, see the PCI SDK Programmer's Manual.

### 3.5.3 Win32 Device Driver Overview

This section describes the overall layout and concept of a PLX device driver. To accommodate the need for one common PCI API as well as to reduce development time for device driver design for new engineering boards, the following device driver model was created.



**Figure 3-9 The PLX Device Driver Layout**

One device driver handles each type of PLX IC, as seen in Figure 3-9. Each device driver communicates with the PCI API on a one-to-one basis; there is no device driver inter-communication. If a new device driver is developed and added to the system, it can be integrated simply by installing it into the Win32 operating system. If more than one PLX IC is present on an engineering board, the device driver can only see the one that is directly connected to the primary PCI bus. All PCI API functions will access this PLX IC only.

#### 3.5.3.1 PCI IC Device Driver Module

This module provides the management of the PCI engineering boards in Windows NT/98/95. This management includes storing device specific information, processing PCI API and system messages, handling interrupts, and allocating resources for each engineering board. Some non-PLX specific functionality is handled in this module, such as reading from and writing to PCI configuration registers.



### **3.5.3.2 PCI IC Services Module**

This module has access to the entire register set of the PCI IC, and thus is in charge of providing the functionality for the device driver.

### **3.5.4 Creating A New Driver**

This section briefly covers how a new device driver can be created using the existing device driver as a template. When a new PCI IC Services Module, which provides the real functionality for the device driver, is updated to support a new PCI IC, the old PCI IC Services Module is replaced. The new PCI IC Services Module would reflect the new register set of the PCI IC and would support the existing PCI API by accessing the appropriate registers on the new PCI IC based on the PCI API function requested.

The PCI IC Device Driver Module would need little modification to create a new PCI device driver. If new API functions are created, the handling of those functions would force the modification this module to support the new functions.

### **3.5.5 Device Driver Features**

The Win32 device driver supports the sharing of interrupts between many PLX engineering boards. The device driver uses one interrupt line on the PCI bus that all PLX engineering boards share to interrupt the host PC. The interrupt service routine determines which board caused the interrupt and services that board's interrupt.

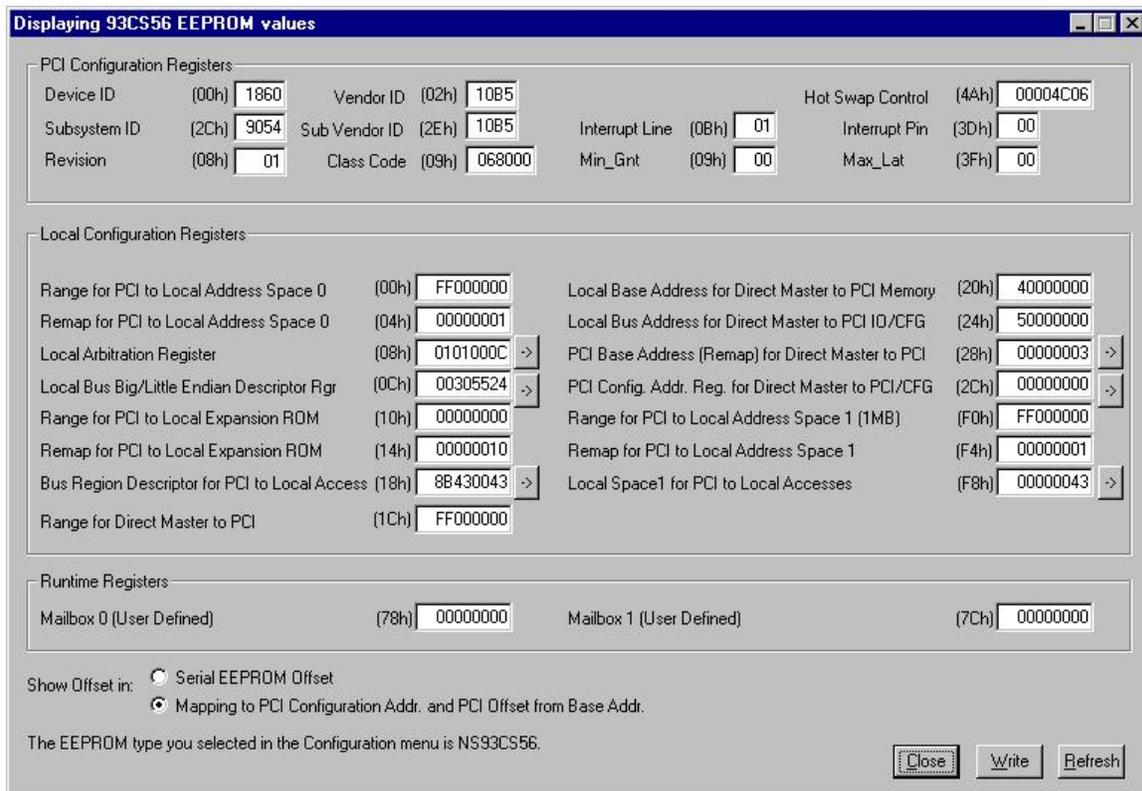
The device driver supports event logging into the system log file. When the device driver determines an error in operation, it updates the system event log file with the appropriate information concerning the cause of the error. This log file can be used to debug the device driver when the device driver is started at boot time. This file contains the reasons why the device driver was not loaded and started.

# Appendix A. Serial EEPROM Settings

Before using the PCI SDK software, the configuration EEPROM connected to the PLX IC must be programmed on the PLX engineering board with specific values. Skipping this step can cause unpredictable behavior of the PCI SDK.

To upgrade the configuration EEPROM on a PLX engineering board follow the steps below:

- User's of the PCI 9054RDK-860 need to reprogram the configuration EEPROM with the settings shown in Figure A-1.



The screenshot shows a software window titled "Displaying 93CS56 EEPROM values" with three main sections: PCI Configuration Registers, Local Configuration Registers, and Runtime Registers. Each register is represented by a label, a hexadecimal address, and a text input field containing a value. Some fields have right-pointing arrows, indicating they are expandable. At the bottom, there are radio buttons for "Show Offset in:", a status message about the EEPROM type, and "Close", "Write", and "Refresh" buttons.

PCI Configuration Registers					
Device ID (00h)	1860	Vendor ID (02h)	10B5	Hot Swap Control (4Ah)	00004C06
Subsystem ID (2Ch)	9054	Sub Vendor ID (2Eh)	10B5	Interrupt Line (0Bh)	01
Revision (08h)	01	Class Code (09h)	068000	Interrupt Pin (3Dh)	00
		Min_Gnt (09h)	00	Max_Lat (3Fh)	00

Local Configuration Registers			
Range for PCI to Local Address Space 0 (00h)	FF000000	Local Base Address for Direct Master to PCI Memory (20h)	40000000
Remap for PCI to Local Address Space 0 (04h)	00000001	Local Bus Address for Direct Master to PCI IO/CFG (24h)	50000000
Local Arbitration Register (08h)	0101000C	PCI Base Address (Remap) for Direct Master to PCI (28h)	00000003
Local Bus Big/Little Endian Descriptor Rgr (0Ch)	00305524	PCI Config. Addr. Reg. for Direct Master to PCI/CFG (2Ch)	00000000
Range for PCI to Local Expansion ROM (10h)	00000000	Range for PCI to Local Address Space 1 (1MB) (F0h)	FF000000
Remap for PCI to Local Expansion ROM (14h)	00000010	Remap for PCI to Local Address Space 1 (F4h)	00000001
Bus Region Descriptor for PCI to Local Access (18h)	8B430043	Local Space1 for PCI to Local Accesses (F8h)	00000043
Range for Direct Master to PCI (1Ch)	FF000000		

Runtime Registers	
Mailbox 0 (User Defined) (78h)	00000000
Mailbox 1 (User Defined) (7Ch)	00000000

Show Offset in:  Serial EEPROM Offset  
 Mapping to PCI Configuration Addr. and PCI Offset from Base Addr.

The EEPROM type you selected in the Configuration menu is NS93CS56.

Buttons: Close, Write, Refresh

Figure A-1 Configuration EEPROM Settings for the PCI 9054RDK-860

- User's of the PCI 9080RDK-401B need to reprogram the configuration EEPROM with the settings shown in Figure A-2.

**Displaying 93CS46 EEPROM values**

PCI Configuration Registers			
Vendor ID (00h)	1085	Device ID (02h)	0401
Sub Vendor ID (44h)	1085	Subsystem ID (46h)	9080
Revision (06h)	01	Class Code (04h)	068000
PCI Base Address for Local Expansion ROM (54h)		00000000	
Interrupt Line (08h)	00	Interrupt Pin (0Ah)	01
Min_Gnt (09h)	00	Max_Lat (08h)	00

Local Configuration Registers			
Range for PCI to Local Address Space 0 (14h)	FF000000	Local Base Address for Direct Master to PCI Memory (34h)	B8000000
Remap for PCI to Local Address Space 0 (18h)	00000001	Local Bus Address for Direct Master to PCI IO/CFG (38h)	80000000
Local Arbitration Register (1Ch)	0021001C	PCI Base Address (Remap) for Direct Master to PCI (3Ch)	00000003
Local Bus Big/Little Endian Descriptor Register (20h)	000000C1	PCI Config. Addr. Reg. for Direct Master to PCI/CFG (40h)	00000000
Range for PCI to Local Expansion ROM (24h)	00000000	Range for PCI to Local Address Space 1 (1MB) (48h)	FF000000
Remap for PCI to Local Expansion ROM (28h)	00000000	Remap for PCI to Local Address Space 1 (4Ch)	00000001
Bus Region Descriptor for PCI to Local Accesses (2Ch)	47400343	Local Space1 for PCI to Local Accesses (50h)	00000343
Range for Direct Master to PCI (30h)	F8000000		

Runtime Registers			
Mailbox 0 (User Defined) (0Ch)	00000000	Mailbox 1 (User Defined) (10h)	00000000

Show Offset in:  Serial EEPROM Offset  
 Mapping to PCI Configuration Addr. and PCI Offset from Base Addr.

The EEPROM type you selected in the Configuration menu is NS93CS46.

**Figure A-2 Configuration EEPROM Settings for the PCI 9080RDK-401B**

