



PLX PCI SDK
SOFTWARE DEVELOPMENT KIT
USER'S MANUAL

Version 3.0

Copyright© 1999 PLX Technology, Inc. All rights reserved.

Notice

Copyright © 1999, 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 Technology, INC..

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; 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: PCI-SDK-Man-PI-3.0



PLX SDK License Agreement

PLX SOFTWARE LICENSE AGREEMENT

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

LICENSE Copyright (c) 1999 PLX Technology, Inc.

This PLX Software License agreement is a legal agreement between you and PLX Technology, Inc. for the PLX Software Development Kit (SDK), also referred to as "PLX SDK" which is provided on the enclosed PLX CD-ROM, or may be recorded on other media included in this PLX SDK. PLX Technology owns this PLX SDK. The PLX SDK 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 PLX SDK, PLX grants you a license to use the PLX SDK 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 PLX SDK. 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 PLX SDK 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 PLX SDK 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 PLX SDK and promptly return the entire unused PLX SDK 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 PLX SDK. Any attempt to sub-license, rent, lease, assign or to transfer the PLX SDK except as expressly provided by this license, is hereby rendered null and void.

WARRANTY

PLX Technology, Inc. provides this PLX SDK 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 here under.

PLX Copyright Message Guidelines

The following copyright message along with the following text must appear in all software products generated and distributed which use the PLX API libraries:

"Copyright (c) 1999 PLX Technology, Inc."

Requirements:

- Arial font
- Font size 12 (minimum)
- 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

This page is intentionally left blank.

Table of Contents

1. INTRODUCTION.....	1-1
1.1 GENERAL INFORMATION	1-1
1.2 ABOUT THIS MANUAL.....	1-1
1.3 PCI SDK FEATURES.....	1-1
1.4 WHERE TO GO FROM HERE	1-1
1.5 OTHER PCI SDK MANUALS.....	1-2
1.6 CONVENTIONS.....	1-2
1.6.1 Windows Programming Conventions	1-2
1.7 TERMINOLOGY	1-2
1.8 DEVELOPMENT TOOLS	1-3
1.8.1 IOP 480 Third Party Development Tools	1-3
1.9 CUSTOMER SUPPORT	1-3
2. GETTING STARTED	2-1
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	2-2
2.1.4.1 Windows NT Installation Procedures	2-2
2.1.4.2 Windows 98 Installation Procedures	2-2
2.1.4.3 Uninstalling All Previous Versions Of The PCI SDK Software.....	2-4
2.1.4.4 PCI SDK V3.0 Compatibility	2-4
2.1.4.5 Troubleshooting.....	2-5
2.2 UNDERSTANDING THE PCI SDK.....	2-6
2.2.1 IOP Software.....	2-6
2.2.1.1 Introduction	2-6
2.2.1.2 IOP Applications	2-6
2.2.1.3 How to Compile the Samples	2-7
2.2.2 Windows Based Host Software.....	2-10
2.2.2.1 Introduction	2-10
2.2.2.2 Windows NT Device Drivers	2-11
2.2.2.3 Windows 98 Device Drivers.....	2-15
2.3 USING THE PCI SDK WITH A NEW BOARD	2-16
2.4 USING THE IOP API LIBRARIES WITH OTHER COMPILERS	2-17
3. PCI SDK SOFTWARE ARCHITECTURE OVERVIEW.....	3-1
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).....	3-4
3.4.1.1 Microprocessor Initialization Module.....	3-5
3.4.1.2 Board Initialization Module.....	3-5
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 BEM Command Format and Commands.....	3-13



3.4.3.2	BEM Reply Format	3-14
3.4.3.3	BEM Command Protocols.....	3-14
3.4.4	Methods For Debugging IOP Applications.....	3-20
3.4.4.1	Operation Of The Back-End Monitor In A System	3-20
3.4.5	IOP Applications	3-21
3.4.5.1	IOP Memory And IOP Applications	3-21
3.4.5.2	MiniBSP Application	3-22
3.4.6	Porting The PCI SDK To New Platforms	3-22
3.4.7	Support For Multiple PLX chips On One Board.....	3-23
3.5	HOST WIN32 SOFTWARE ARCHITECTURE.....	3-23
3.5.1	GUI Application PLXMon 99.....	3-24
3.5.1.1	Serial Communication.....	3-24
3.5.1.2	PCI API/Device Driver Communication	3-25
3.5.2	Win32 Applications And The PCI SDK	3-25
3.5.3	Win32 Device Driver Overview.....	3-25
3.5.3.1	PLX Chip Device Driver Module.....	3-26
3.5.3.2	PLX Chip Services Module.....	3-26
3.5.4	Creating A New Driver	3-26
3.5.5	Device Driver Features.....	3-27
3.5.6	Distribution of PLX Device Drivers and PLXApi.Dll File	3-27
3.5.6.1	Installation of PlxApi.dll File	3-27
3.5.6.2	Installation of PLX Device Driver.....	3-27
4.	REAL TIME OPERATING SYSTEM SUPPORT	4-1
4.1	GENERAL INFORMATION	4-1
4.2	GETTING STARTED	4-1
4.3	MINIMUM REQUIREMENTS	4-1
4.4	INSTALLATION.....	4-1
4.5	WHAT'S INCLUDED?.....	4-2
4.6	WHICH VxWORKS ROM IMAGE SHOULD I USE?.....	4-3
4.7	PLX VxWORKS BSP/PLX API DEMONSTRATION	4-3
4.7.1	Updating the PCI 9054RDK-860 or CompactPCI 9054RDK-860 onboard FLASH	4-3
4.7.2	PLX API functions Demonstration	4-3
4.7.2.1	Stand alone VxWorks Target Shell Demo (No Tornado and no Target Server present)	4-3
4.7.2.2	Tornado VxWorks Host Shell Demo.....	4-4
4.8	HOW TO REBUILD THE BSP AND APPLICATION IMAGES?	4-10
4.8.1	Setup the makefile to build PLX VxWorks Boot ROM.....	4-10
4.8.2	Setup the custom project to build PLX demo application.....	4-10
4.9	TORNADO 1.0.1 AND TORNADO 2.0 COMPATIBILITY	4-11
5.	RDK SOFTWARE QUICK REFERENCE	5-1
5.1	IOP 480RDK	5-1
5.2	PCI 9054RDK-860.....	5-2
5.3	COMPACTPCI 9054RDK-860.....	5-3
5.4	PCI 9080RDK-401B	5-5
5.5	PCI 9080RDK-860.....	5-6
5.6	PCI 9080RDK-SH3.....	5-8
5.7	PCI 9080RDK-RC32364.....	5-10
APPENDIX A.	INDEX.....	I

List of Figures

FIGURE 2-1 COMPONENTS OF THE PCI SDK.....	2-1
FIGURE 2-2 WINDOWS HOST SOFTWARE LAYOUT FOR PCI SDK V3.0.....	2-10
FIGURE 2-3 THE DEVICES UTILITY WINDOW.....	2-11
FIGURE 2-4 THE EVENT VIEW WINDOW.....	2-12
FIGURE 2-5 THE DETAILED EVENT WINDOW.....	2-12
FIGURE 2-6 REGISTRY INFORMATION FOR PCI 9080 DEVICE DRIVER ON WINDOWS NT	2-13
FIGURE 2-7 REGISTRY INFORMATION FOR PCI 9054 DEVICE DRIVER ON WINDOWS NT.....	2-13
FIGURE 2-8 REGISTRY INFORMATION FOR IOP 480 DEVICE DRIVER ON WINDOWS NT	2-13
FIGURE 2-9 THE PCI SDK DEVICE DRIVER WIZARD	2-15
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 DIAGRAM OF THE IOP MEMORY USAGE	3-22
FIGURE 3-8 THE HOST SOFTWARE ARCHITECTURE	3-24
FIGURE 3-9 THE PLX DEVICE DRIVER LAYOUT.....	3-26
FIGURE 4-1 STAND ALONE VxWORKS TARGET SHELL DEMO SCREEN	4-4
FIGURE 4-2 REBOOT THE STAND ALONE VxWORKS TARGET SHELL DEMO	4-5
FIGURE 4-3 BACK END PROPERTY PAGE.....	4-6
FIGURE 4-4 CORE FILE AND SYMBOLS PROPERTY PAGE	4-6
FIGURE 4-5 VIRTUAL CONSOLE PROPERTY PAGE.....	4-7
FIGURE 4-6 TGTSVR - CONSOLE.....	4-7
FIGURE 4-7 TORNADO SHELL PROMPT	4-8
FIGURE 4-8 TGTSVR - CONSOLE NO. 1.....	4-9
FIGURE 4-9 TGTSVR - CONSOLE NO. 2.....	4-9
FIGURE 4-10 CUSTOMIZE BUILDS SCREEN NO. 1	4-10
FIGURE 4-11 CUSTOMIZE BUILDS SCREEN NO. 2	4-11
FIGURE 4-12 OPTIONS	4-11
FIGURE 4-13 BUILD VxWORKS.....	4-12
FIGURE 4-14 CONFIGURE TARGET SERVERS.....	4-13
FIGURE 5-1 CONFIGURATION EEPROM SETTINGS FOR THE IOP 480RDK	5-2
FIGURE 5-2. CONFIGURATION EEPROM SETTINGS FOR THE PCI 9054RDK-860	5-3
FIGURE 5-3. CONFIGURATION EEPROM SETTINGS FOR THE COMPATCTPCI 9054RDK-860	5-5
FIGURE 5-4. CONFIGURATION EEPROM SETTINGS FOR THE PCI 9080RDK-401B.....	5-6
FIGURE 5-5. CONFIGURATION EEPROM SETTINGS FOR THE PCI 9080RDK-860	5-8
FIGURE 5-6. CONFIGURATION EEPROM SETTINGS FOR THE PCI 9080RDK-SH3	5-10



List of Tables

TABLE 3-1. BEM COMMANDS	3-13
TABLE 5-1. BASIC INFORMATION ABOUT IOP 480RDK.....	5-1
TABLE 5-2. BASIC INFORMATION ABOUT PCI 9054RDK-860.....	5-2
TABLE 5-3. BASIC INFORMATION ABOUT COMPACTPCI 9054RDK-860	5-4
TABLE 5-4. BASIC INFORMATION ABOUT PCI 9080RDK-401B	5-5
TABLE 5-5. BASIC INFORMATION ABOUT PCI 9080RDK-860.....	5-7
TABLE 5-6. BASIC INFORMATION ABOUT PCI 9080RDK-SH3	5-8
TABLE 5-7. BASIC INFORMATION ABOUT PCI 9080RDK-RC32364	5-10

1. Introduction

1.1 General Information

PLX Technology offers PCI bus interface chips that address a range of adapter and embedded system applications. Our PCI chips work well with a variety of CPUs and embedded controllers, including the IBM PowerPC 40x family, Motorola MPC860/850 and 68360, as well as the PowerPC 60x family, Analog Devices Sharc, various Texas Instruments DSPs, the Intel i960 family, Hitachi SuperH family, IDT MIPS, and others. If the design does not require a microprocessor, our chips are easily configured to run without the aid of a CPU. PLX provides Software Development Kits (SDK), Reference Design Kits (RDK) and Hardware Design Kits (HDK) that facilitate your PCI design development.

1.2 About This Manual

The PLX family of 32-bit I/O processor and I/O accelerator chips connects the PCI bus to Intel, Power PC and other processors. They provide full Intelligent I/O (I₂O) compatibility. The PCI Software Development Kit (PCI SDK) provides a powerful I/O Platform Application Programming Interface (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 chip. Users should consult this manual when installing the PCI SDK and for general information.

1.3 PCI SDK Features

The PCI SDK includes the following features:

- A feature based IOP API, with support for a variety of PLX PCI chips;
- Board Support Package (BSP) that allows customization of the PCI SDK;
- A Back-End Monitor application used for basic debugging;
- IOP DMA Resource Manager that supports three modes of operation;
- A PCI API and device drivers compatible with Windows NT/98; and,
- PLXMon 99, a Windows Graphical User Interface (GUI) application used to configure, modify PLX PCI devices, and download IOP applications to the local RAM.

1.4 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.

1.5 Other PCI SDK Manuals

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

- PCI SDK Programmer's Reference Manual: This manual covers all software design issues regarding the device drivers, API and user applications.
- PLXMon 99 User's Manual: This manual describes the usage of the PLXMon 99 application.
- IOP 480 CPU API Programmer's Reference Manual: This manual covers the IOP 480 CPU API and will be created only as a PDF file in Acrobat format during the PCI SDK installation.
- PLX Device Driver Manual: This manual covers the PLX device drivers and service modules, which are the core parts of the PLX device drivers. The manual will be created only as a PDF file in Acrobat format during the PCI SDK installation.
- PLX RDK Manufacturing Test Specification: This manual covers the basic information about the PLX RDK Manufacturing Test program and will be created only as a PDF file in Acrobat format during the PCI SDK installation.

1.6 Conventions

Please note that when software samples are provided the following notations are used:

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

1.6.1 Windows Programming Conventions

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

- *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.

1.7 Terminology

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

All references to Windows 98 may be denoted as Win98.

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 RDK board and all references to IOP software denote the software running on the RDK board.

All references to FLASH should be replaced with EPROM for users of the PCI 9080RDK-SH3 and PCI 9080RDK-RC32364 Reference Design Kits.

1.8 Development Tools

Development tools used to develop the PCI SDK include:

- Win32 Applications: Microsoft Visual C++ 5.0, with Microsoft Developer Studio;
- Win32 Applications: Microsoft Platform Software Development Kit (SDK);
- Windows NT 4.0 Drivers: Microsoft Windows NT 4.0 Device Driver Kit (DDK);
- Windows 98 Drivers: Microsoft Windows 98 Device Driver Kit (DDK);
- IOP 480RDK IOP Software:
 1. IBM High C/C++ PowerPC Cross-Compiler, version 1.0 (7/31/96); and IBM 401 EVB Software Support Package, version 1.6.4 (4/1/97)
 2. DIAB Data, Inc. Compiler and Linker for PowerPC, version 4.0b and version 4.3p6;
- PCI 9080RDK-860, CompactPCI and PCI 9054RDK-860 IOP Software: DIAB Data, Inc. Compiler and Linker for the PowerPC, version 4.0b and version 4.3p6;
- PCI 9080RDK-401B Software:
 1. IBM High C/C++ PowerPC Cross-Compiler, version 1.0 (7/31/96); and IBM 401 EVB Software Support Package, version 1.6.4 (4/1/97)
 2. DIAB Data, Inc. Compiler and Linker for PowerPC, version 4.0b and version 4.3p6;
- PCI 9080RDK-RC32364 IOP Software: IDT/c Cross Compiler System Version 5.5/7.0 GNU Developer's Kit.
- PCI 9080RDK-SH3 IOP Software: Cygnus GNU compiler: gcc version 2.7-96q3a

1.8.1 IOP 480 Third Party Development Tools

In addition to the IBM High C/C++ PowerPC Cross-Compiler and the Diab Data PowerPC Compiler and Linker, other PowerPC compilers may be used to develop IOP 480 based applications. The PLX PCI SDK version 3.0 complies to the ANSI C code standard. Minimal changes to batch and make files may be required if using compilers other than the IBM and Diab Data PowerPC compilers.

PLX Partners and recommended PowerPC compilers:

- MetaWare High C/C++ for PowerPC
- Green Hills Software PowerPC C/C++ Optimizing Compilers
- Mentor Graphics Microtec C & C++ Compilers for PowerPC
- Metrowerks CodeWarrior for PowerPC

1.9 Customer Support

Prior to contacting PLX customer support, please ensure that you are situated close to the computer that has the PCI SDK installed and have the following information:



1. Model number of the PLX PCI RDK (if any);
2. PLX PCI SDK version (if any);
3. Host Operating System and version;
4. Description of your intended design:
 - PLX chip used
 - Microprocessor
 - Local Operating System and version (if any)
 - I/O
5. Description of your problem; and
6. Steps to recreate the problem.

You may contact PLX customer support at:

Address: PLX Technology, Inc.
Attn. Technical Support
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
mid-apps@plxtech.com
east-apps@plxtech.com
euro-apps@plxtech.com
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)

- PCI SDK User's Manual (this document);
- PCI SDK Programmer's Reference Manual;
- PLXMon 99 User's Manual; and
- PCI SDK CD-ROM.

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

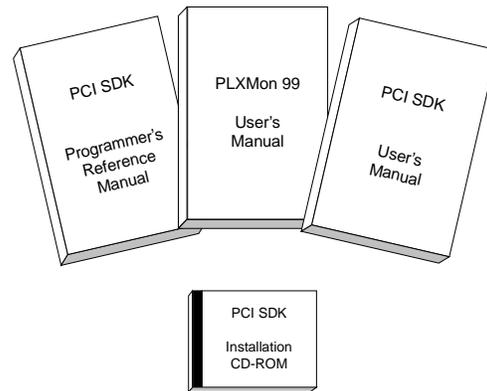


Figure 2-1 Components of the PCI SDK

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 Windows 98;
- 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);
- 80MB hard drive space; and,
- 1 RS 232 serial port.

2.1.3 Development Requirements

The PCI SDK development environment is either the Window NT 4.0 or Windows 98 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 Win32 Driver Model (WDM) device drivers were developed using the Microsoft Windows 98 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 after the installation. This completes the PCI SDK software installation.

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\PciSdk300) to any drive and path that is desired. This document uses “<INSTALLPATH>” to denote the installation directory.

2.1.4.1.1 Windows NT Device Driver Installation

The Windows NT installation wizard takes care of the device driver installation.

2.1.4.2 Windows 98 Installation Procedures

The installation of the PCI SDK software onto a Win98 system requires two steps: Install the PCI SDK files and start the Win98 device driver(s). The following two sections describe how to completely install the PCI SDK Support Software for Win98.

2.1.4.2.1 Windows 98 Software Installation

To install the PCI SDK 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 RDK boards are installed in your computer.
2. Insert the CD-ROM into the appropriate CD-ROM drive.
3. 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.
4. 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\PciSdk300) 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 Windows 98 Device Driver Installation

A device driver is necessary for the PCI SDK software to communicate to the PLX RDK board. PCI SDK applications cannot communicate with any RDK board through the PCI interface without a PCI SDK device driver installed. The installation software used to install the PCI SDK can not automatically start the device drivers for PLX devices. To start 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 RDK board into a free PCI slot.
3. Reboot the computer. Windows 98 should first detect the new hardware device with a “New Hardware Found” message box. Acknowledge this message box.
4. Windows 98 displays the “Add New Hardware” Wizard. Windows 98 displays the following message: “This wizard searches for new drivers for:” with the corresponding board name following it. If you are using a PLX RDK board proceed to step 5. If you are using a custom Reference Design Board with a PLX device then proceed to step 10.

Driver Installation for PLX Reference Design Boards:

5. Click on the “Next” button. Once Windows 98 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 Windows 98 displays the following message: “Windows has finished installing the software that your new hardware device requires”.
9. Click on “Finish” to complete the Win98 device driver installation.

Driver Installation for Custom Reference Design Boards with PLX devices:

In order for the custom reference design boards to be treated as one of “Custom (OEM)” boards during the following steps, the device and vendor IDs must be:

- Vendor ID = 0x10B5, Device ID = 0x9080 for a PCI 9080 device;
 - Vendor ID = 0x10B5, Device ID = 0x9054 for a PCI 9054 device; and
 - Vendor ID = 0x10B5, Device ID = 0xF480 for an IOP 480 device.
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 “Custom (OEM) PCI 9054 board” if using a PCI 9054 device. Otherwise choose “Custom (OEM) PCI 9080 board” if using a PCI 9080 device or “Custom (OEM) IOP 480 board” if using an IOP 480 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 Win98 device driver installation.

Note: If you change the PLX RDK board from one slot to another, Windows 98 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 Uninstalling All Previous Versions Of The PCI SDK Software

Prior to installing a new version of the PCI SDK, you should first uninstall all older versions. There is interaction between PCI and IOP software modules and it is important not to mix releases. 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 V3.0 Compatibility

Due to interaction between host and IOP software components it is very important that PCI SDK releases are the same on both sides (i.e. if the host software is from PCI SDK V3.0, then the IOP software should also be from PCI SDK V3.0 as well). If you have recently purchased a PLX RDK then it will already contain the correct IOP software preprogrammed, so you need not concern yourself with this section. However, if you have purchased the PCI SDK as an upgrade and intend to use it with an earlier PLX RDK board you will need to upgrade the RDK board’s FLASH with a current version. This is necessary to ensure that nothing unpredictable occurs due to incompatibilities with modules.

Although it is important to have your host and IOP software from the same release, the IOP API and PCI API are almost compatible with PCI SDK v2.0 and higher. PLX has developed an API model that is portable and will continue to support this model in future releases. The previous paragraph is necessary because some PLXMon 99 features rely on a communication protocol with the IOP BSP and the protocol has improved in each release. The IOP API and PCI API are fully compatible between releases.

Users who are upgrading their PCI SDK and intend to use it with an earlier PLX RDK board should follow the steps below.

To upgrade your FLASH image, follow the steps below:

- Users of the IOP480RDK may use PLXMon 99, as described in the PLXMon 99 User Manual, to download the FLASH image file “<INSTALLPATH>\hw\Flash\Iop480rdk.bin” to the IOP480RDK. The image should be usually programmed at FLASH offset 0x60000.
- Users of the PCI 9054RDK-860 may use PLXMon 99 to download the FLASH image “<INSTALLPATH>\hw\Flash\9054RDK-860.bin” to the PCI 9054RDK-860. The image should be programmed at FLASH offset 0x00000.
- Users of the CompactPCI 9054RDK-860 may use PLXMon 99 to download the FLASH image “<INSTALLPATH>\hw\Flash\CPCI9054RDK-860.bin” to the CompactPCI 9054RDK-860. The image should be programmed at FLASH offset 0x00000.
- Users of the PCI 9080RDK-401B may use PLXMon 99 to download the FLASH image “<INSTALLPATH>\hw\Flash\9080RDK-401B.bin” to the PCI 9080RDK-401B. The image should be usually programmed at FLASH offset 0x60000.
- Users of the PCI 9080RDK-860 must use a device programmer to reprogram the FLASH with FLASH image “<INSTALLPATH>\hw\Flash\9080RDK-860.bin”. The image should be programmed at FLASH offset 0x00000.
- Users of the PCI 9080RDK-SH3 must use a device programmer to reprogram the EPROM with the FLASH image “<INSTALLPATH>\hw\Flash\9080RDK-Sh3.bin”. The image should be programmed at EPROM offset 0x00000.
- Users of the PCI 9080RDK-RC32364 must use a device programmer to reprogram the EPROM with the image file “<INSTALLPATH>\hw\Flash\9080RDK-RC32364.bin”. The image should be programmed at EPROM offset 0x00000.

2.1.4.5 Troubleshooting

You may experience difficulties using the PCI SDK with Windows NT with low memory and multiple PLX RDK boards. If you notice that one of your RDK boards is not being assigned the proper memory resources by the PCI BIOS (e.g. no address provided to the Local Space 0 address) it is most likely due to a common memory problem with WinNT. It is recommended that users increase the amount of available system pages in their System Registry by following the steps below;

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.



2.1.4.5.1 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.

In PCI systems, the BIOS often assigns the same interrupt to multiple devices; however, the respective device drivers must support these "shared interrupts". A driver that does not support this feature may prevent the PLX driver from functioning correctly. A possible workaround 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 PLX chip from IOP software. The IOP applications are user-interactive and require PLXMon 99 with a serial cable link.

The IOP applications are designed specifically to run on a PLX RDK board. However, the IOP applications can be used as a good starting point when designing custom target platforms.

2.2.1.2 IOP Applications

The PCI SDK contains several sample applications. By default, all PLX RDK boards contain the "monitor" application preprogrammed in FLASH memory. This application is a command line interpreter, which accepts commands from the serial port and acts accordingly. The monitor program enables you to read from or write to memory in 32-bit, 16-bit and 8-bit units. Complete source code for this application is provided in the PCI SDK. Please refer to the PLXMon 99 User's Manual for more information on how to communicate to the PLX RDK boards IOP applications.

In the <INSTALLPATH>\hw\FLASH directory, there are default ROM images for all the RDKs supported by the PCI SDK. Every RDK FLASH is programmed and shipped with a relevant image from this directory. The user can choose to restore the FLASH to the default ROM image using the corresponding FLASH image from this directory later if the FLASH image on the user's RDK board is corrupted for whatever reason.

2.2.1.2.1 MiniBSP Application

MiniBSP 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 PLX chip, and proceeds to blink the LED, if any, that is connected to one of the PLX chip's USER pins. To use the MiniBSP application, you should program the binary image into the FLASH using a FLASH chip programmer. Once the FLASH is programmed, put the FLASH into the board, and reboot the board. If the LED blinks, then the MiniBSP application configured the

board properly. Otherwise, you have to modify the MiniBSP source file for your hardware configuration.

Note:

1. *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. *"MiniBSP" and "MiniROM" refer to essentially the same application. (Users of previous version of SDK may be familiar with the term "MiniROM").*

2.2.1.3 How to Compile the Samples

There are four compiler packages used in the PCI SDK to compile code for all the seven RDKs supported by the PCI SDK. The four compiler packages are as follows:

1. IBM High C/C++ PowerPC Cross-Compiler, version 1.0 (7/31/96); and IBM 401 EVB Software Support Package, version 1.6.4 (4/1/97). This compiler set supports the following two RDKs:
 - PCI 9080RDK-401B; and
 - IOP 480RDK.
2. DIAB Data, Inc. Compiler and Linker for the PowerPC, version 4.0b or version 4.3p6. This compiler package supports the following five RDKs:
 - PCI 9080RDK-860;
 - PCI 9054RDK-860;
 - CompactPCI 9054RDK-860;
 - PCI 9080RDK-401B; and
 - IOP 480RDK.
3. IDT/c Cross Compiler System Version 5.5/7.0 GNU Developer's Kit. This compiler package supports the PCI 9080RDK-RC32364 RDK.
4. Cygnus GNU compiler: gcc version 2.7-96q3a. This compiler Package supports the PCI 9080RDK-SH3 RDK.

2.2.1.3.1 Batch Files For Setting Up the Environment Variables

After the installation of the PCI SDK package, there are four batch files in the <INSTALLPATH>\bin directory. These batch files set up the compiler environment variables for one of the above-mentioned compilers and these batch files are:

1. SetIbm.bat file, which sets up compiler environment for IBM High C/C++ PowerPC Cross-Compiler and IBM 401 EVB Software Support Package;
2. SetDiab.bat file, which sets up compiler environment for DIAB Data, Inc. Compiler and Linker for the PowerPC;
3. SetIdt.bat file, which sets up compiler environment for IDT/c Cross Compiler System Version 5.5/7.0 GNU Developer's Kit;
4. SetSh3.bat file, which sets up compiler environment for Cygnus GNU compiler.



2.2.1.3.2 Batch Files Need Modification

The four batch files mentioned in the above section need modifying to reflect the directory structure on the customer's system. The batch files are well commented and should be easily modified with a text editor.

- The directory path to the `nmake.exe`, a make utility program from Microsoft Developer Studio and all the IOP make file are written in the `nmake` rules, must be included in the `PATH` environment variable.
- The directory path to the `doskey.com` or `doskey.exe` program must exist in one of the directories indicated by the `PATH` environment variable.

2.2.1.3.3 Set Up The Compiler Environment Variables

There are two ways to set up environment variables for the compiler the customer chooses. They are:

- 1. Select the Windows "Start" menu;
2. Select the "Programs" submenu;
3. Select the "PLX PCISDK v300" folder; and
4. Select one of the four "Compiler Environment" shortcuts;
- 1. Run the MS-DOS prompt;
2. Type one of the batch files listed in section 2.2.1.3.1 such as "setdiab", and press the ENTER key.

2.2.1.3.4 Recompile the Hello World Sample

To rebuild the Hello World application, execute the `nmake` file contained within the `<INSTALLPATH>\Iop\Samples\Hello` directory. To do this, first set up the desired compiler environment variables using one method listed in the above section, then change to the `<INSTALLPATH>\Iop\Samples\Hello` directory if necessary, and type the following to build the Hello World IOP RAM application for the PCI 9080RDK-401B:

```
nmake /f 80-401B.mak
```

To rebuild the Hello World IOP ROM application, type the following line instead:

```
nmake /f 80-401B.mak ROM=TRUE
```

The `nmake` file builds the application properly by using the environment variables set by the batch file and by any parameters passed in from the command prompt.

In the `<INSTALLPATH>\Iop\Samples\Hello` directory, there is only one source file, being the Hello World application's main file `hello.c` and a `nmake` file for each supported RDK. When rebuilding the IOP application, the `nmake` file links in the appropriate libraries from the PCI SDK library directory for all the support functions needed by the IOP application.

2.2.1.3.5 Troubleshooting During the Compilation

During the recompiling process, if you have difficulty compiling, look into the following factors, which might lead to the compiling problems:

- Have you updated the batch file for setting up the compiler environment variables?
- Is `Nmake.exe` utility program in the directory indicated by the PATH variable?
- Is `Nmake.exe` utility program in the directory indicated by the PATH variable the right utility program you intend to use?
- Have you set up your compiler environment variables?
- Have you run the batch file to set up the compiler variables?

2.2.1.3.6 “Out of Environment Space” Message on Windows 98

When you run the MS-DOS prompt under Windows 98, the environment space for existing variables are allocated by default. This may lead to the “Out of environment space” message when you are trying to set up the compiler environment variables by running one of the batch files mentioned above. There are two ways to fix this problem.

- 1. Run a MS-DOS prompt;
- 2. Type “set” and press the ENTER key;
- 3. Record the string after the “COMSPEC=”, let’s assume it is `C:\windows\command.com`;
- 4. Use a text editor such as `notepad.exe` to edit the `c:\config.sys` file. `C:` is assumed to be Windows 98 boot-up hard drive;
- 5. Check there is a “SHELL=” line in the `config.sys` file.
 - If there isn’t the line, add the following line “SHELL=**C:\Windows\Command.com** c:\ /E:2048 /P”. The bolded string in the line is the string recorded in step 3.
 - If there is the line, check the number after the “/E:” and increase it to multiples of 256;
- 6. Save the `C:\Config.sys` file and reboot.
- 1. Run a MS-DOS prompt either by selecting “Menu” -> “Programs” -> “PLX PCISDK v3.00” and then selecting one of the compiler environment shortcuts or by running another MS-DOS prompt shortcut;
- 2. Select “Properties” from the popup menu and select the tab “Memory” from the dialogue box that appears;
- 3. Change the item labeled in the “Initial Environment” from “Auto” to 2048 or even bigger;
- 4. Click “Apply” pushbutton and then click “OK” when a warning dialogue box appears;
- 5. Exit the MS-DOS prompt by typing “Exit”;
- 6. Re-run the MS-DOS prompt and you should not get the “Out of environment space” message. If you still do, repeat the step 1 to 5 and increase the size to a bigger environment size.

2.2.2 Windows Based Host Software

2.2.2.1 Introduction

The PCI SDK contains six distinct device drivers, an API, and a Windows monitor application

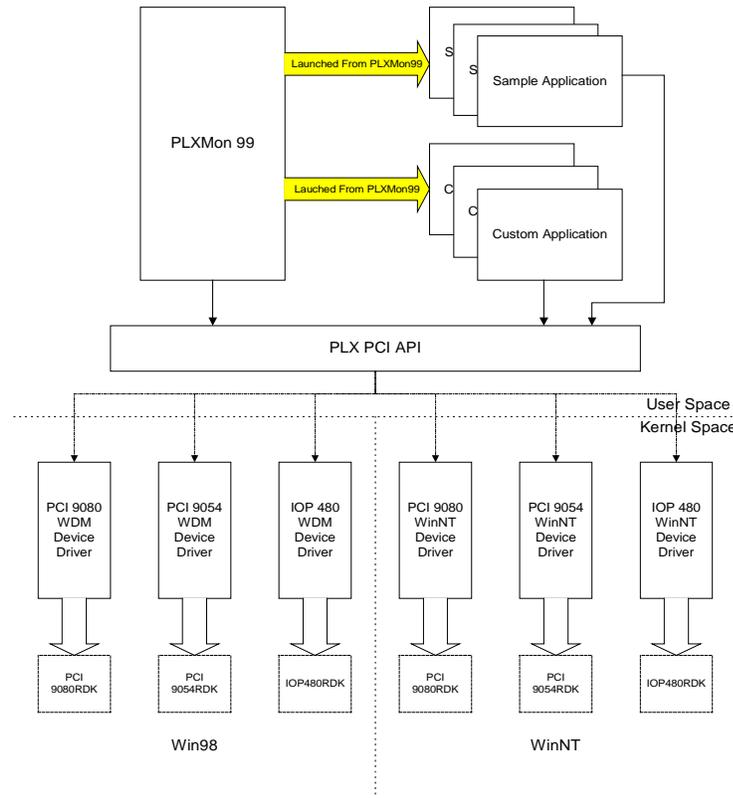


Figure 2-2 Windows Host software Layout for PCI SDK V3.0

(see Figure 2-2). They are as follows:

- Three PLX WinNT Device Drivers supporting the PCI 9080, the PCI 9054, and IOP 480;
- Three PLX WDM Win98 Device Drivers supporting the PCI 9080, the PCI 9054, and IOP 480 ;
- PCI API, a powerful API compatible with all PLX devices and PLX device drivers; and
- PLXMon 99, a Graphical User Interface (GUI) application that can be used to monitor and modify PLX chip registers. It can also download software to a PLX RDK board, and communicate to the software running on the RDK 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 PLXMon 99, please refer to the PLXMon 99 User's Manual.

2.2.2.2 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, or iop480.sys. For example, the device driver for the PCI 9080 device is named Pci9080.sys.

2.2.2.2.1 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.

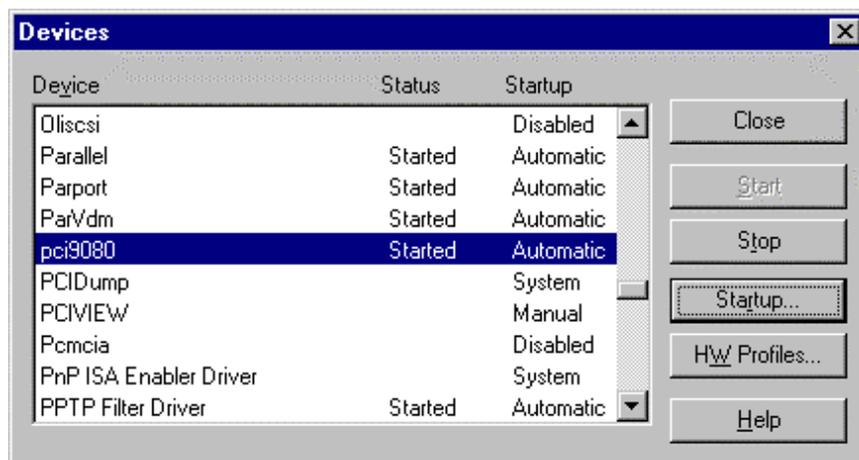


Figure 2-3 The Devices Utility Window.

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 2.2.2.2.4 for more details.

2.2.2.2.2 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.

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.

Date	Time	Source	Category	Event	User	Computer
8/12/99	10:13:05 AM	Pci9080	None	257	N/A	PETER
8/12/99	10:13:05 AM	Pci9080	None	256	N/A	PETER
8/12/99	10:13:03 AM	Pci9054	None	257	N/A	PETER
8/12/99	10:13:03 AM	Pci9054	None	256	N/A	PETER
8/12/99	10:12:58 AM	lop480	None	264	N/A	PETER
8/12/99	10:12:58 AM	lop480	None	256	N/A	PETER

Figure 2-4 The Event View Window.

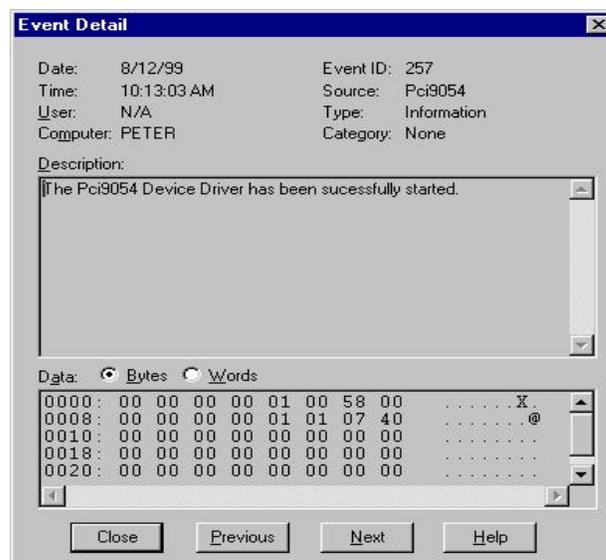


Figure 2-5 The Detailed Event Window

2.2.2.2.3 Registry Configuration

Every Windows NT device driver requires an entry in the registry. The registry contains lots of information used by the operating system as well as information used by the PLX device driver if necessary. The name in the registry for the PLX PCI SDK device driver 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. The values displayed in the figures may be different from those in your system after you use the PCI SDK package. These figures are shown here to demonstrate the registry entries used by PCI SDK.

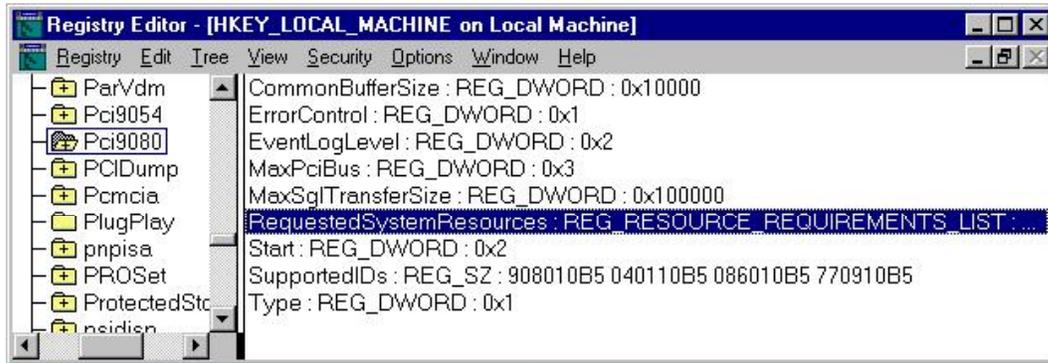


Figure 2-6 Registry Information For PCI 9080 Device Driver on Windows NT

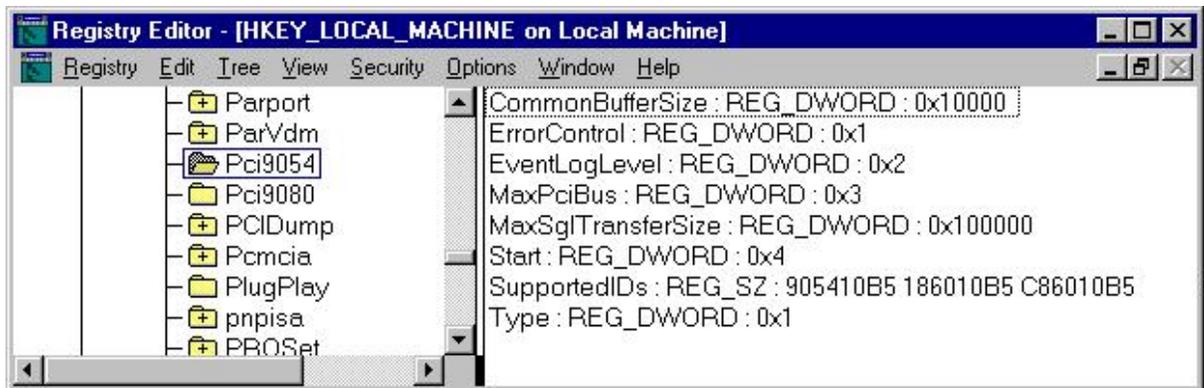


Figure 2-7 Registry Information for PCI 9054 Device Driver on Windows NT

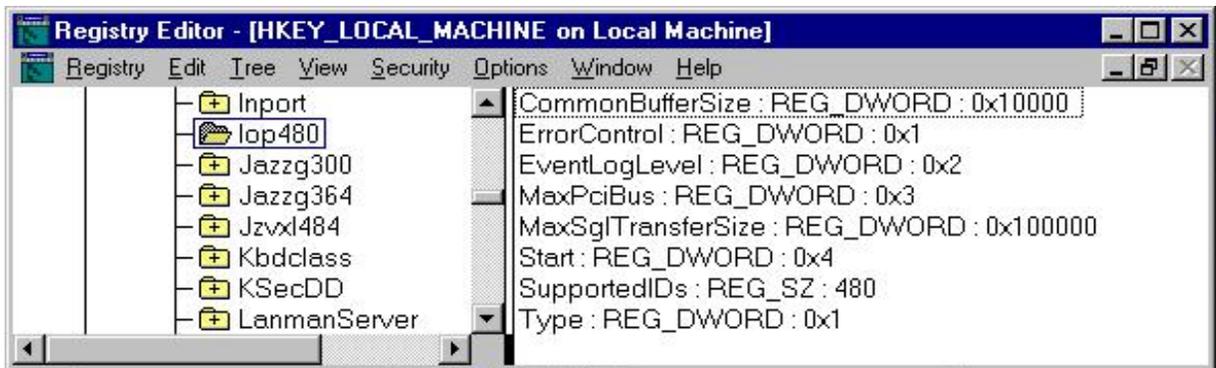


Figure 2-8 Registry Information for IOP 480 Device Driver on Windows NT



Note: *The registry editor should be used to modify registry entries only by advanced users with administrative rights. It is recommended that you NOT change any values contained in the registry.*

The registry entries for each of the PLX device drivers are listed as follows:

- *CommonBufferSize:* This value sets the size of the user buffer (the "hbuf" in PLXMon 99). Its default value is set to 64KB. *Warning: First, the device driver makes a request to the operating system for a buffer with the size indicated by this registry entry. However, if the device driver fails to get the buffer requested due to a lack of system resource, then it will decrement the size until it is given an allocated buffer by the operating system. 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.
- *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 application DriverWizard to modify this field. Modification of this field directly might make the DriverWizard application run erratically.
- *Type:* This value is required by the operating system and should not be modified.

2.2.2.2.4 Driver Configuration

Before using the device driver with a customer board, the driver must first be configured with the appropriate Vendor ID and Device ID. PLXMon 99 has a hot-link to a PCI SDK utility called the Device Driver Wizard, which is also in the program menu for the PCI SDK package. This utility is used to add or remove vendor and device IDs of the boards from the SupportedIDs entry for the appropriate device driver. It also lets you enable or disable the desired PLX device driver to start automatically at startup.

Note: If you are not using the PCI 9054 or PCI 9080 or IOP 480 device driver you should disable it by using this utility. You must either restart your computer or restart the device driver before the settings take effect.

2.2.2.2.5 Known Problem of Windows NT Device Drivers

There is one known problem for Windows NT device drivers. The problem is that the use of the PlxIntrAttach() function with SGL DMA transfers in Windows NT leads to exception faults.

An exception fault can occur when attaching a User Signal to the interrupt on the termination of a SGL transfer in Windows NT. This is due to the Windows NT subsystem getting I/O completion requests at too near an interval in time. The best remedy for this is to place a __try- except

handler around the code where the exception occurs. This will successfully handle the exception and no loss of data will occur.

2.2.2.3 Windows 98 Device Drivers

The PCI SDK includes Windows 98 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 or iop480.sys. For example, the device driver for the PCI 9080 device is named Pci9080.sys.



Figure 2-9 The PCI SDK Device Driver Wizard

2.2.2.3.1 Starting And Stopping

Unlike Windows NT drivers, Windows 98 device drivers are started and stopped as needed by the operating system. The PLX device drivers are started when Windows 98 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 still needs it.

There is no applet that controls the starting or stopping of a device driver under Windows 98.

2.2.2.3.2 Event Logging

Event logging is not accessible on Windows 98.



2.2.2.3.3 Registry Configuration

Every Windows 98 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 in the registry 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 or IOP480.sys, depending on the PLX chip in use).

Note: *The registry editor should be used to modify the registry entries only by advanced users. It is recommended that you NOT change any values contained in the registry.*

2.2.2.3.4 Known Problems of Windows 98 Device Drivers.

Windows 98 contains a few features that do not yet 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 NOT exceed 1 MB in size.

Changing Device Slot Numbers

If the RDK board is removed and placed into another PCI slot, Windows 98 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 and IOP 480

Windows 98 Power Management support was not complete when the PCI SDK was released. Therefore, the method recommended by Microsoft in its documentation to change the power level of a device does not work as expected. To overcome this problem two possible methods can be used:

1. Change the PCI 9054 or IOP 480 power level using a different method than the one recommended by Microsoft. The intended behavior can be obtained. However, this could cause problems in future releases of Windows 98.
2. Leave the device driver sections as is, in hopes that Microsoft will correct the problem in future releases of Windows 98.

The PCI SDK uses the first option in order to maintain Power Management capabilities. Both the IOP 480 driver and the PCI 9054 driver use the same algorithm for changing power levels.

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 add 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 2.2.2.2.4).
3. If using Windows 98, you will need to consult section 2.1.4.2.2 to register the new device with the Windows 98 device drivers.
4. Edit the MiniBSP application as necessary to support the new board.
5. Program the board's FLASH with the modified MiniBSP application binary image file.
6. PLXMon 99 can now access the board's configuration EEPROM. Using PLXMon 99's EEPROM Configuration window, customize the EEPROM settings for the new 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 board (This means the memory controller for the IOP memory has been set up and direct master access to the IOP memory by the PLX device has been initialized as well).

When the above 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 board. Consult the PCI SDK Programmer's Manual for more information on porting the PCI SDK to new boards.

2.4 Using The IOP API libraries With Other Compilers

The PCI SDK contains IOP API libraries and a PCI API library. The PCI API library is compiled to work with Windows operating systems (Windows NT and Windows 98). However, the IOP API library is compiled for a special microprocessor such as Motorola MPC860 by a compiler chosen, such as DIAB PowerPC compiler, in a format chosen, such as ELF format. 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 type-casted 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.



This page is intentionally left blank.

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 5, 6 and 7 are reserved for communication between PLXMon 99 and the IOP software when PLXMon 99 downloads RAM applications to the IOP.
- When a PLX PCI device driver is started, mailbox register 3 of the device supported by the driver 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 (BEM) 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 PLX chip is done by the IOP software only;
- The data expected by the application will not contain any data that could be interpreted by the BEM as a command if the BEM is linked into the application;
- All IOP applications must relinquish the processor periodically to avoid starvation of the BEM (cooperative or non-preemptive multitasking);
- When an application is downloaded to the IOP RAM or the application wants to reprogram the on-board FLASH using a serial connection, the IOP BSP must execute the *CheckPciDownloadToRam()* and the *CheckSerialDownloadToRam()* functions at microprocessor reset or re-execution of boot-up code initialized by the software;
- The *BlinkLed()* function assumes that the LED is connected to the PLX chip's USERo pin, or users can comment this function out if there is no LED connected, or choose to provide another way to blink a LED such as 9080RDK-RC32364 does; 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. Please send an email to software@plxtech.com if you require support for Little Endian processors.

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 RDK 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 PLX chip. Some function calls look and react similarly in both API's but may have different parameter lists.

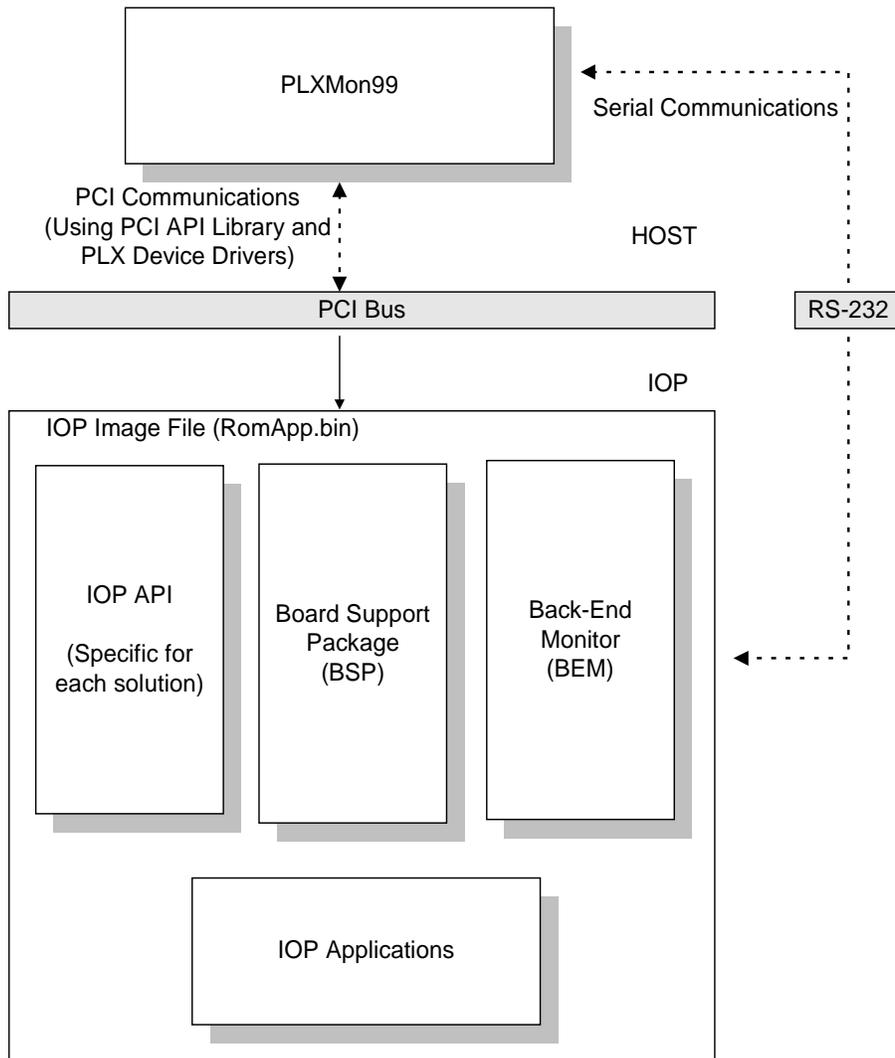


Figure 3-1 The PCI SDK Software Architecture

The IOP software contains three modules (excluding the IOP application), the IOP API library, the Board Support Package (BSP) and the Back-End Monitor (BEM). The IOP API is designed specifically for each PLX chip or for a combination of PLX chips on one RDK board. The IOP

API can be customized to run on any RDK board by modifying the Board Support Package. IOP debugging can be performed with PLXMon 99 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 RDK board. This package is implemented within the PLXMon 99 application.

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 returned 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:

- PLXMon 99: this module includes PCI Bus communication and serial communication to the Back-End Monitor;
- PCI API library file `PLXApi.dll` which translates API function calls into device driver function calls to the PLX device drivers ;
- PLX device drivers which actually control the access to the PLX devices;
- IOP API Library: this library contains the code that performs the API functions and accesses the PLX chip. There are at least two IOP APIs for each PCI device: Release and Debug. Both libraries are the same except the release version eliminates many of the parameter validation steps that are performed in the debug version, and hence performance is increased if the release version of the API is used. All debug libraries contain a 'd' suffix in their name (E.G. `api860d.a`). Release libraries do not contain the 'd' suffix in their name (E.G. `api860.a`).
- BSP Module: this module contains all board specific code, including the IOP bus memory map, the board and microprocessor initialization routines and the interrupt service routine for the PLX chip;
- Back-End Monitor: this module provides a monitor for debugging IOP applications which supports PLXMon 99 through the serial port; and,
- IOP Applications: this module contains the main application for the RDK board and the IOP.

3.4 IOP Software Architecture

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

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

- The IOP application software (user application modules).

The IOP software architecture is shown in Figure 3-2.

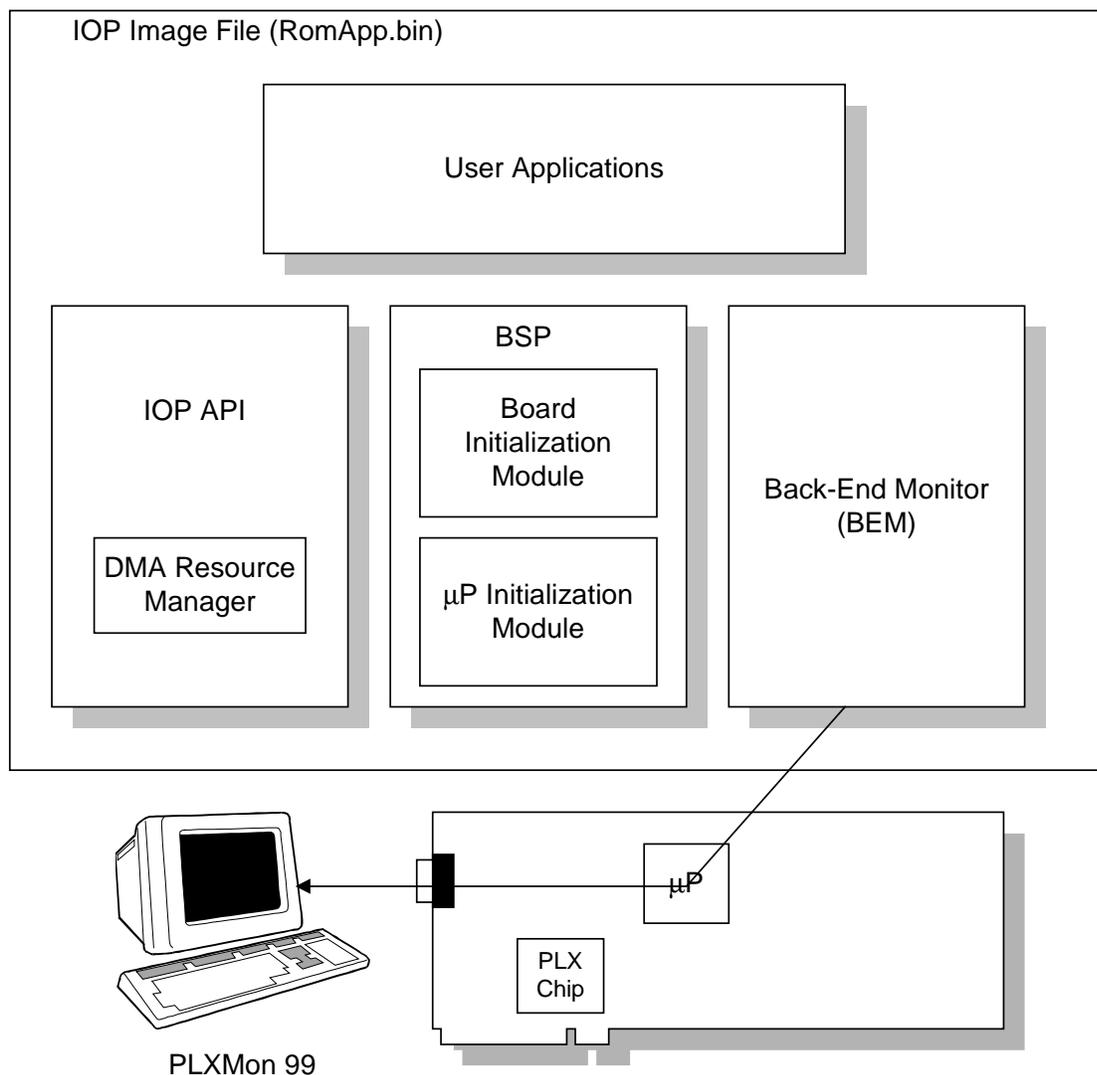


Figure 3-2 The IOP Software Architecture

3.4.1 Board Support Package (BSP)

The Board Support Package (BSP) contains all the information needed by the IOP API that is specific to the board. This module provides the necessary entry points needed to port the PCI SDK to new platforms. The BSP is composed of two main sub-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 PLX chip 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. The boot 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 MiniBSP application included in the PCI SDK provides a good starting point for users who have untested boards. The application is limited in features and functionality and should be the basis for porting the PCI SDK to new 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 a PLX chip. The ISR is divided into one main routine with one function to service each interrupt trigger on the chip. When an interrupt is generated, the 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 includes the memory map of the IOP bus, specifically where the following devices are located in memory:

- SRAM address and range, if any;



- DRAM address and range, if any;
- SDRAM address and range, if any;
- PLX chip 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-up 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 PLX chip (specifically which bytes and byte lanes are used by the IOP bus).

The Back-End Monitor needs to use UART-related functions. 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 PLX chip, the IOP API library, the Back-End Monitor and any other device on the RDK 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 PLX chip. A list of IOP API initialization functions is provided with each of its parameters set to the PLX chip'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 Initialization Status bit when the PLX chip is initialized (this asserts the NB# pin low). This bit allows the PCI BIOS to access the PLX chip. Once the PCI BIOS finished assigning the appropriate values to the PLX chip's PCI configuration registers, the PLX chip 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 Interrupt Service Routines to the appropriate interrupt lines of the microprocessor.
7. Initialize the application, if necessary, once all devices on the RDK board have been initialized and are operational.
8. Jump to the `AppMain()` 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 PLX chip (when there is no EEPROM connected to it), the

UART chip, and the Back-End Monitor. 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 RDK board and buffer it. This stream can be received by any task requiring data from the serial port.

The Back-End Monitor, *BemMain()*, does simple debugging. This monitoring task is used with PLXMon 99. The *BemMain()* monitor task accepts a variety of standardized commands for reading from, writing to memory or EEPROM, resetting the IOP.

With the stream of data received from the serial port (see Figure 3-3), the *BemMain()* task receives and parses through it, searching for commands. When *BemMain()* 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 *BemMain()* 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()*.

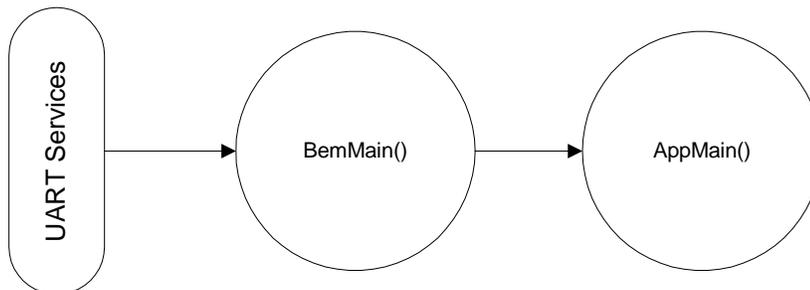


Figure 3-3 The Data Stream Flow Diagram.

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 *BemMain()* task. If the *BemMain()* 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 *PlxGetChars()* directly to retrieve the unfiltered data from the UART chip 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 PLX chip (no intermediary functions).

There are at least two IOP APIs for each PCI device: Release and Debug. Both libraries are the same except the release version eliminates many of the parameter validation steps that are performed in the debug version, and hence performance is increased when using the release version of the API. All debug libraries contain a 'd' suffix in their name (E.G. `api860d.a`). Release libraries do not contain the 'd' suffix in their name (E.G. `api860.a`).

For the PowerPC CPU type supported by DIAB compiler, two sets of release and debug version IOP API libraries are created: Release and Debug versions in ELF format; Release and Debug versions in COFF format. Libraries in ELF or in COFF format exist in DIAB-ELF or DIAB-COFF subdirectory respectively under the directory `<INSTALLPATH>\Iop\Lib\RDK_NAME`. `RDK_NAME` is one of the following five RDK directory names.

- IOP 480RDK;
- 9054RDK-860;
- CPC19054RDK-860;
- 9080RDK-860; and
- 9080RDK-401B.

For the 9080RDK-401B and IOP 480 RDK, one more set of release version and debug version IOP API libraries are created with the IBM High C/C++ PowerPC Cross-Compiler. Libraries created by IBM High C/C++ PowerPC Cross-Compiler are located under the `<INSTALLPATH>\Iop\Lib\RDK_NAME\IBM-ELF` directory where **RDK_NAME** can be either 9080RDK-401B or IOP480RDK.

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

3.4.2.1 DMA Resource Manager

The IOP API supports three different DMA (Direct Memory Access) transfer types and manages the DMA resources. The supported DMA transfer types are:

- Scatter-Gather DMA: Transfers data using Scatter-Gather Lists (SGL) and can transfer several blocks of data at a time (formally called chaining DMA);
- Block DMA: Transfers data one block at a time (IOP 480 Flyby DMA is a special block DMA); 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, therefore the performance is improved.

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).

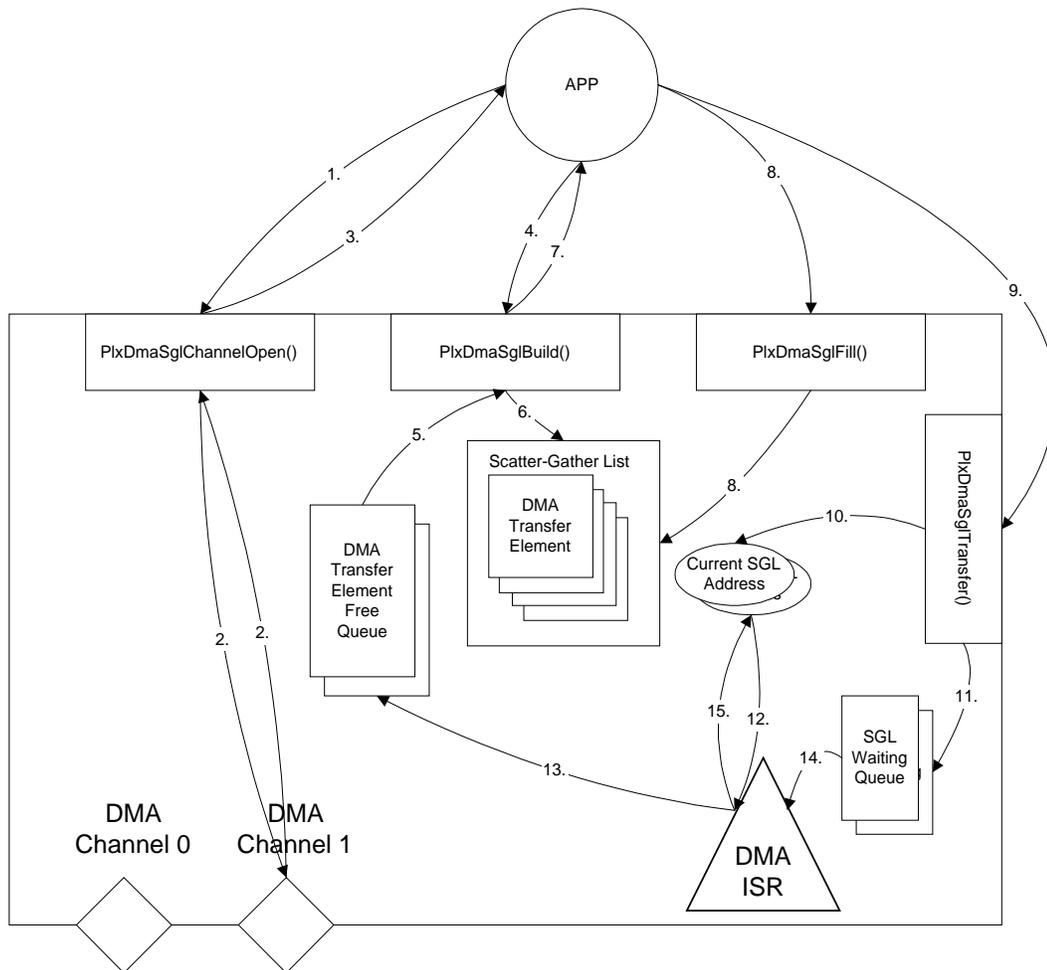


Figure 3-4 Scatter-Gather DMA Flow Diagram

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).

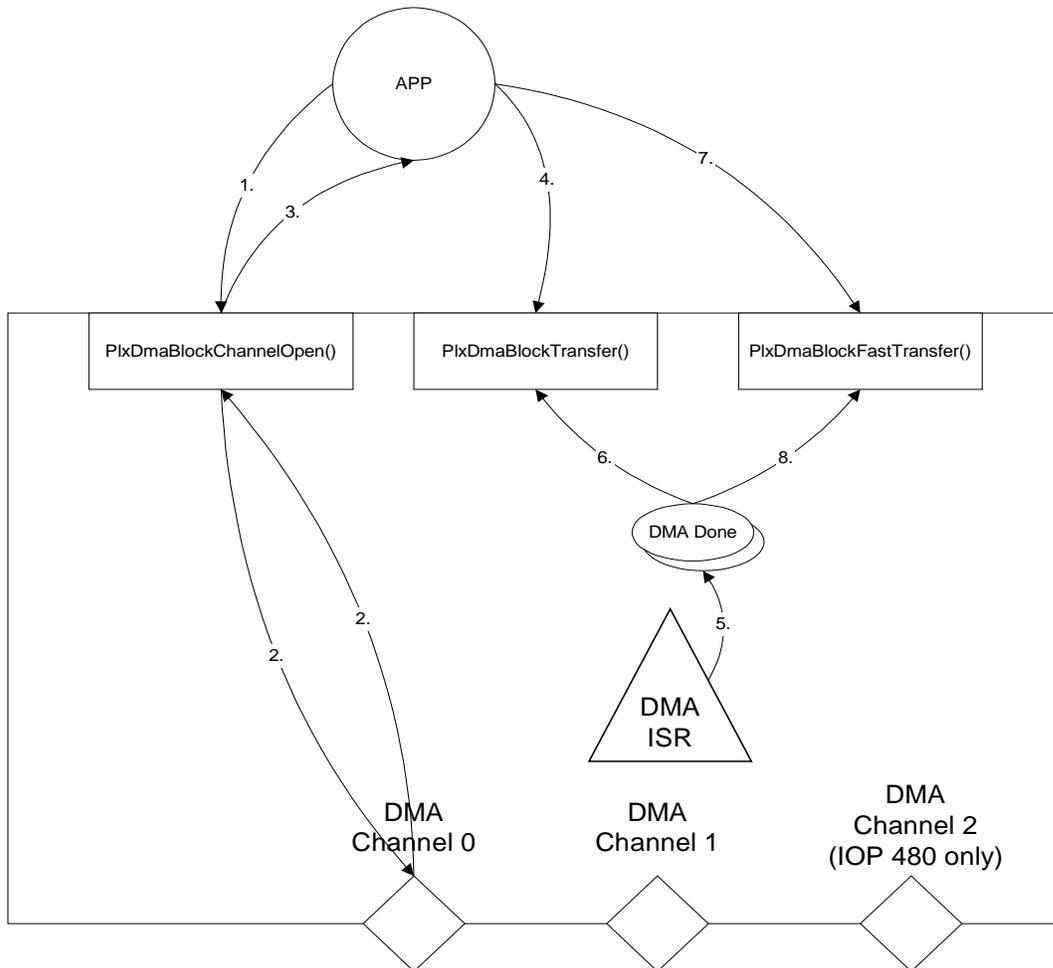


Figure 3-5 Block DMA Transfer Flow Diagram

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.

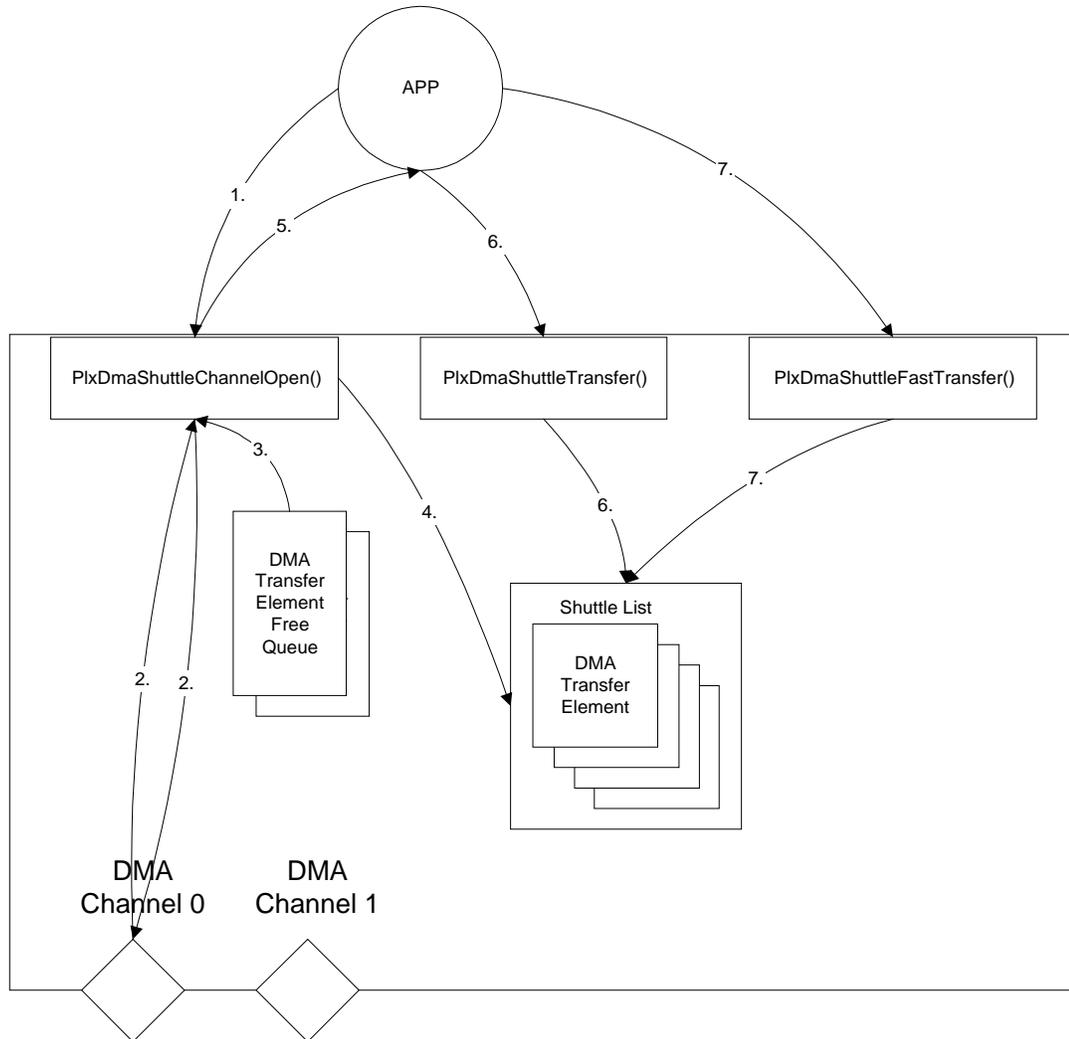


Figure 3-6 The Shuttle DMA Flow Diagram

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 *PlxDmaShuttleTransfer()* 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 PLX chip (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 debugging IOP applications. The BEM allows the host application PLXMon 99 to send commands to the IOP application using a serial connection to a RDK board. The Back-End Monitor supports several commands, including reading from and writing to IOP memory locations (these commands support different data sizes), resetting the IOP software, reading from or writing to the EEPROM connected to the PLX chip, and reprogramming the on-board FLASH (if supported), etc. These commands provide a generic interface for an IOP application. PLXMon 99 uses this monitor to retrieve data from the IOP. In normal operation, the BEM accesses the UART Services functions to get a stream of data that has been received by the UART chip. The monitor extracts commands (that the monitor recognizes) from the data stream, performs the necessary action and provides an appropriate response. The monitor provides the filtered data stream to the next task requiring serial data in the daisy chain if the data is not designed for the BEM.

There are times when a task may not want other tasks to extract data (or commands) from the data stream. Other tasks are prevented from extracting data from the data stream passed by the UART if one task access the UART support functions directly. A task, which wants to receive raw data and bypasses the previous task in the daisy chain, can call *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 return control back to the main routine (contained within the BSP) once the action is completed.

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.

The Back-End Monitor (BEM) can recognize different commands coming from PLXMon 99: reset the IOP microprocessor; read a memory location and write to a memory location etc. The protocol for the serial communication between the IOP application and the host PLXMon 99 is documented 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 another” parameter must be provided.

Carriage returns are denoted as <CR>.

3.4.3.1 BEM Command Format and Commands

Please note:

- *All BEM commands are case sensitive except hexadecimal value parameters.*
- *There will be no leading zero in the hex number string to speed up the serial communication. Of course, if the data is zero, there will be only one zero ASCII letter.*
- *If two consequent hexadecimal values are required, then they are separated by a space ASCII letter (0x20h).*
- *There is no space between a hexadecimal letter (0 . . 9, a . . f, or A . . F) and another non-hexadecimal letter such as P.*
- *Due to the fact the Microsoft Word automatically toggles the case of the first letter in the sentence when you type, there is a possibility that the command letter in this document might not be accurate even though every effort is made to make it accurate. For the latest and most accurate information on BEM, please read <INSTALLPATH>\inc\Bem.h file.*

Command Format

_____ 2 Bytes _____	_____ 1 Byte _____	_____ Extra Bytes _____	<CR>
Header (~p)	BEM Command	Command-Specific Info	

Table 3-1. BEM Commands

BEM Command	Definition
!	Reset the IOP board
@	Query the information for the board
g	Read a 8-bit data from IOP local memory
h	Read a 16-bit data from IOP local memory
i	Read a 32-bit data from IOP local memory
j	Read a 64-bit data from IOP local memory
k	Read from EEPROM into the EEPROM data buffer
m	Read multiple 8-bit data from IOP local memory
n	Read multiple 16-bit data from IOP local memory
o	Read multiple 32-bit data from IOP local memory
p	Read multiple 64-bit data from IOP local memory
z	Read from IOP 480 or 401B CPU register
G	Write a 8-bit data to IOP local memory
H	Write a 16-bit data to IOP local memory
I	Write a 32-bit data to IOP local memory
J	Write a 64-bit data to IOP local memory
K	Write from the EEPROM data buffer into the EEPROM
M	Write multiple 8-bit data to IOP local memory



BEM Command	Definition
N	Write multiple 16-bit data to IOP local memory
O	Write multiple 32-bit data to IOP local memory
P	Write multiple 64-bit data to IOP local memory
Z	Write to IOP 480 or 401B CPU register

3.4.3.2 BEM Reply Format

Reply Format

```
| ___ 1 Byte _____ | _____ Extra Bytes _____ | <CR>
| REPLY_HEADER (0x1) | Data or message returned |
```

Message Symbols	Explanation
!	Reply Success
@	Reply Error

3.4.3.3 BEM Command Protocols

- **Single Read**

Host to BEM:

```
~p<g | h | i | j><Address><CR>
```

BEM to Host:

```
<REPLY_HEADER><DATA><CR>
```

- **Single Write**

Host to BEM:

```
~p<G | H | I | J><Address><Space><Data><CR>
```

BEM to Host:

None

- **Reading from EEPROM**

Host to BEM:

```
~pk<ByteSize><CR>
```

BEM to Host:

1. Data is stored at the local memory indicated by the *BufferAddressLow* and *BufferAddressHigh* to be mentioned later in this section.
Note: The <ByteSize> cannot be bigger than the physical byte size of the on-board EEPROM.
2. <REPLY_HEADER>!<CR> if OK
3. <REPLY_HEADER>@<CR> if error

- **Writing to EEPROM**

Host to BEM:

1. Data is stored by the host at the local memory indicated by the *BufferAddressLow* and *BufferAddressHigh* to be described in section.
2. Then, the host issues the following string:
~pK<ByteSize><CR>

Note: The <ByteSize> cannot be bigger than the physical byte size of the on-board EEPROM.

BEM to Host:

<REPLY_HEADER>!<CR> if OK
<REPLY_HEADER>@<CR> if error

• **Block Read**

Host to BEM:

~p< m | n | o | p ><StartAddress><Space><ItemCount><CR>

BEM to Host:

The following data themselves are used to demonstrate.

For 8-bit read, the output will be:

<REPLY_HEADER>0 1 10 20 30 0 4 5 60. . .<CR>

For 16-bit read, the output will be:

<REPLY_HEADER>0 1 1234 104 0 1345 890 798. . .<CR>

For 32-bit read, the output will be:

<REPLY_HEADER>0 12345678 0 1234 78900 0. . .<CR>

For 64-bit read, the output will be:

<REPLY_HEADER>0 123456789abcdef0 . . . <CR>

• **Block Write**

Host to BEM:

The following data themselves are used to demonstrate.

8 bits:

~pM<StartAddress> 0 10 2 3 45 67 89 90 a . . . <CR>

16 bits:

~pN<StartAddress> 0 10 2 3 4 5 7890 1234 6789 . . .<CR>

32 bits:

~pO<StartAddress> 0 12345678 12 48 90000 12345 . . . <CR>

64 bits:

~pP<StartAddress> 0 123456789abcdef0 4 567890. . .<CR>



The input data can be as long as it is necessary, the BEM will continuously write data until it receives the ending <CR> or the format does not fit. For example, 16-bit data can not be longer than 5 hexadecimal numbers.

BEM to Host:

None.

• **Reading from IOP 480 CPU or IBM401GF Registers**

Host to BEM:

~pz<IOP480_CPU_INDEX><CR>

BEM to Host:

<REPLY_HEADER><DATA><CR>

• **Writing to IOP 480 CPU or IBM401GF Registers**

Host to BEM:

~pZ<IOP480_CPU_INDEX><Space><Data><CR>

BEM to Host:

None

The IOP 480 CPU Registers are indexed as follows:

/* 32 General Purpose Register */	#define CPU480_R26	26	
#define CPU480_R0	0	#define CPU480_R27	27
#define CPU480_R1	1	#define CPU480_R28	28
#define CPU480_R2	2	#define CPU480_R29	29
#define CPU480_R3	3	#define CPU480_R30	30
#define CPU480_R4	4	#define CPU480_R31	31
#define CPU480_R5	5		
#define CPU480_R6	6	/* Machine State Register */	
#define CPU480_R7	7	#define CPU480_MSR	32
#define CPU480_R8	8		
#define CPU480_R9	9	/* Condition Register */	
#define CPU480_R10	10	#define CPU480_CR	33
#define CPU480_R11	11		
#define CPU480_R12	12	/* Special Purpose Registers */	
#define CPU480_R13	13	#define CPU480_CDBCR	34
#define CPU480_R14	14	#define CPU480_CTR	35
#define CPU480_R15	15	#define CPU480_DAC	36
#define CPU480_R16	16	#define CPU480_DBCR	37
#define CPU480_R17	17	#define CPU480_DBSR	38
#define CPU480_R18	18	#define CPU480_DCCR	39
#define CPU480_R19	19	#define CPU480_DCWR	40
#define CPU480_R20	20	#define CPU480_DEAR	41
#define CPU480_R21	21	#define CPU480_ESR	42
#define CPU480_R22	22	#define CPU480_EVPR	43
#define CPU480_R23	23	#define CPU480_IAC	44
#define CPU480_R24	24	#define CPU480_ICCR	45
#define CPU480_R25	25	#define CPU480_ICDBDR	46

#define CPU480_LR	47	#define CPU480_SRR1	59
#define CPU480_PID	48	#define CPU480_SRR2	60
#define CPU480_PIT	49	#define CPU480_SRR3	61
#define CPU480_PVR	50	#define CPU480_TBHI	62
#define CPU480_SGR	51	#define CPU480_TBHU	63
#define CPU480_SKR	52	#define CPU480_TBLO	64
#define CPU480_SLER	53	#define CPU480_TBLU	65
#define CPU480_SPRG0	54	#define CPU480_TCR	66
#define CPU480_SPRG1	55	#define CPU480_TSR	67
#define CPU480_SPRG2	56	#define CPU480_XER	68
#define CPU480_SPRG3	57	#define CPU480_ZPR	69
#define CPU480_SRR0	58		

Note: Because the General Purpose Registers (GPRs) of IOP 480 are used constantly by the local CPU, it is probably meaningless to read any one of them. It is **EXTREMELY DANGEROUS** to modify IOP 480 CPU registers, no matter what registers are, GPRs or Special Purpose Registers (SPRs) unless you have a hardware or software debugger running.

- **Query the Information About The Board Connected.**

Host to BEM:

```
~p@<CR>
```

BEM to Host:

1. Interface structure between PLXMon 99 and BEM

The following structure is an interface used by the PLXMon 99 and the BEM to pass information from the BEM to the PLXMon 99. The exact information passed from BEM to PLXMon 99 will be described in detail later.

```
typedef struct _PLATFORM_PARAMS
{
    U32 Version; /* BEM Version & Platform type */
    U32 PlxAddressLow; /* Lower 32-bits of PLX Chip base address */
    U32 PlxAddressHigh; /* Upper 32-bits of PLX Chip base address */
    U16 PlxChipType; /* PLX Chip Type */
    U16 Capability; /* Capabilities of the BEM */
    U8 EEPROMType; /* EEPROM type, not used by BEM */
    U8 FlashType; /* FLASH type, not used by BEM */
    U16 Reserved; /* Space for future use */
    U32 FlashAddressLow; /* Lower 32-bits of FLASH base address */
    U32 FlashAddressHigh; /* Upper 32-bits of FLASH base address */
    U32 BufferAddressLow; /* Perm. buffer for EEPROM programming */
    U32 BufferAddressHigh;
    U32 BufferSize; /* Permanent Buffer size */
    U32 FlashBufferAddressLow; /* Temp Buffer for FLASH programming */
    U32 FlashBufferAddressHigh;
    U32 FlashBufferSize; /* Temp FLASH buffer size */
} PLATFORM_PARAMS, * PPLATFORM_PARAMS;
```

Version is defined as follows:



3	3	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	9	8	7	6	5	4	3	2	1	0
Version Major						Version Minor						Version Revision						Plat- form *	Reserved										

Platform:

Bit 6: 0 = 32-bit addressing

1 = 64-bit addressing

Bit 7: 0 = 32-bit data size

1 = 64-bit data size

Capability is defined as follows:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Read from or Write to Memory			FLASH Programming			EEPROM Programming			Reserved						

3-Bit Field	Description
0	Specifies whether operation is supported. 0 : Not supported 1 : Supported
2:1	Reserved

Example:

0x2080 (Memory R/W , No FLASH programming, EEPROM programming)

0	0	1	0	0	0	0	0	1	0	0	0	0	0	0	0
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

2. BEM responds to the inquiry posed by PLXMon 99.

BEM will respond to the PLXMon 99 inquiry with the following information separated by a space char between the consequent two items in the bolded block.

<REPLY_HEADER>

Version Information

PlxAddressLow

PlxAddressHigh

PlxChipType

Capabilities

EEPROMType (Note: An ASCII string from EEPROM_TYPE defined in PlxTypes.h)

FLASHType (Note: An ASCII string from FLASH_TYPE defined in PlxTypes.h)

FlashAddressLow

FlashAddressHigh

BufferAddressLow

BufferAddressHigh

BufferSize
FlashBufferAddressLow
FlashBufferAddressHigh
FlashBufferSize

<CR>

For example:

Version Information = 3000000 (Major 3, Minor 0, Rev0, 32-bit data and address)
 PlxAddressLow = 20000000
 PlxAddressHigh = 0
 PlxChipType = IOP 480
 Capabilities = 2480 (All programming supported: EEPROM, FLASH, Memory R/W)
 EEPROMType (String: e.g. Eeprom93CS66)
 FLASHType (String: e.g. AT49LV040)
 FlashAddressLow = FFF80000
 FlashAddressHigh = 0
 BufferAddressLow = 10000
 BufferAddressHigh = 0
 BufferSize = 100
 FlashBufferAddressLow = 10000000
 FlashBufferAddressHigh = 0
 FlashBufferSize = 80000

IOP 480 BEM would respond with the following data:

```

<REPLY_HEADER>3010000 20000000 0 480 2480 Eeprom93CS66
AT49LV040 FFF80000 0 10000 0 100 10000000 0 80000<CR>
  
```

- **FLASH Programming**

Because of serial mode communication properties, the host has to write the data, either for EEPROM or FLASH programming, into the local memory first, and it then informs the local CPU to program either EEPROM or FLASH. That's the reason why buffers are provided.

Query the local CPU for the data structure mentioned above. If the *Capabilities* bits indicate the BEM supports FLASH programming, then continue.

Write IOP_SERIAL_FLASH_PROTOCOL to the MailBox 6 and write the starting FLASH offset to program at to the MailBox 7, and write the FLASH image size to the MailBox 5. Finally, the host resets the local CPU.

When the local CPU is reset, it checks MailBox 6 for the IOP_SERIAL_FLASH_PROTOCOL. If the MailBox 6 contains IOP_SERIAL_FLASH_PROTOCOL, then the local CPU initializes and begins to receive FLASH image data through serial mode using X-Modem file transfer protocol.



When all the FLASH image data has been received, the local CPU begins to reprogram the FLASH. When the reprogramming is done, it resets itself.

Note:

The protocol for downloading a FLASH image from the host program to the embedded follows the same guidance as the downloading of a RAM application through a serial channel. Both targets of the serial downloading download a file in the Binary Block Format defined in the `protocol.h`. For FLASH, the block reads like this:

FlashBufferAddressLow	BlockImageSize	. . data . .	0	0
U32	U32	U32 units	U32	U32

Please notice that the buffer for the FLASH programming is to the DRAM or SDRAM buffer for storing FLASH image instead of directly to the FLASH itself.

• **EEPROM Programming**

1. Query the local CPU for the *BufferAddressLow*, *BufferAddressHigh*, *BufferSize* parameters as well as whether EEPROM programming is supported or not.
2. If the IOP side supports the EEPROM programming, write EEPROM data to the local *Buffer* using BEM commands if the host wants to write data to the EEPROM;
3. Issue `WRITE_TO_EEPROM` command for writing to the EEPROM or `READ_FROM_EEPROM` command for reading from EEPROM to the BEM module or issue;
4. Read EEPROM data from the local *Buffer* using BEM commands if the host wants to read data from the EEPROM.

3.4.4 Methods For Debugging IOP Applications

The PCI SDK supports two methods for debugging IOP applications. They are:

- **Win32 Debugging:** Using PLXMon 99. This method assumes that there is no IOP application running on the RDK board. With new RDK boards, this method provides the preliminary debugging and validation of new RDK boards.
- **PLXMon 99 with the BEM:** With the BEM linked into the IOP application, PLXMon 99 can communicate to the RDK board through the PC's COM port to the serial port on the RDK board. PLXMon 99 can be set up to communicate to the RDK 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 RDK 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. *BemMain()* 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. PLXMon 99 can be used to debug the application if the RDK board is inserted into a free slot in the host system's PCI Bus and PLXMon 99's PCI Communication is turned on.

Method 2: PLXMon 99 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 (by calling *PlxGetChars()*).

3.4.5 IOP Applications

The IOP API, the BSP and the Back-End Monitor libraries are linked with the IOP application objects 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 the way 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 cycle is created using the *main()* function. This cycle 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 executed as a ROM program, it 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.

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 as far as the source code is concerned, 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 is 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 RDK 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 into the resident ROM application.

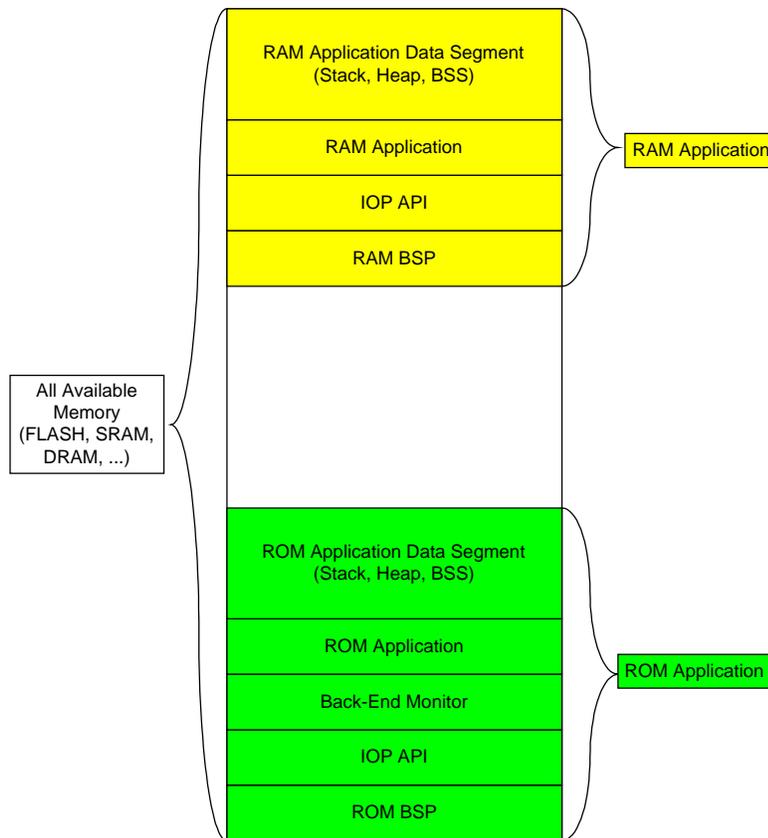


Figure 3-7 Diagram of the IOP Memory Usage

3.4.5.2 MiniBSP Application

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

Note: The MiniBSP application is provided as a bare bones IOP ROM application useful for confirming the functionality of new Reference Design Boards. It does not contain any PCI SDK features that are described in the PCI SDK manuals.

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 includes:

- The memory map of the IOP bus;
- The microprocessor boot code;

- The PLX chip interrupt service routine;
- The UART interrupt service routine or the polling mechanism for retrieving data from UART;
- 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 PLX chip, 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 to send and receive data from the serial port. Modify the UART code to support the UART on the board.

To initialize the PLX chip, modify the parameters for the IOP API initialization functions contained within the board initialization routine.

Within the BSP, there is 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 PLX chips On One Board

Each PLX chip has its own IOP API library. When two or more chips are present on one Reference Design 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 chips. Some multiple chip 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-chip combinations not currently supported.

3.5 Host Win32 Software Architecture

This section describes the Win32 software provided in the PCI SDK. The Win32 software provided in the PCI SDK includes a Graphic User Interface (GUI) application PLXMon 99, a PCI API library (`PLXApi.dll`), PLX device drivers and sample programs.

3.5.1 GUI Application PLXMon 99

PLXMon 99 can communicate with PCI devices via two different paths (see Figure 3-8):

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

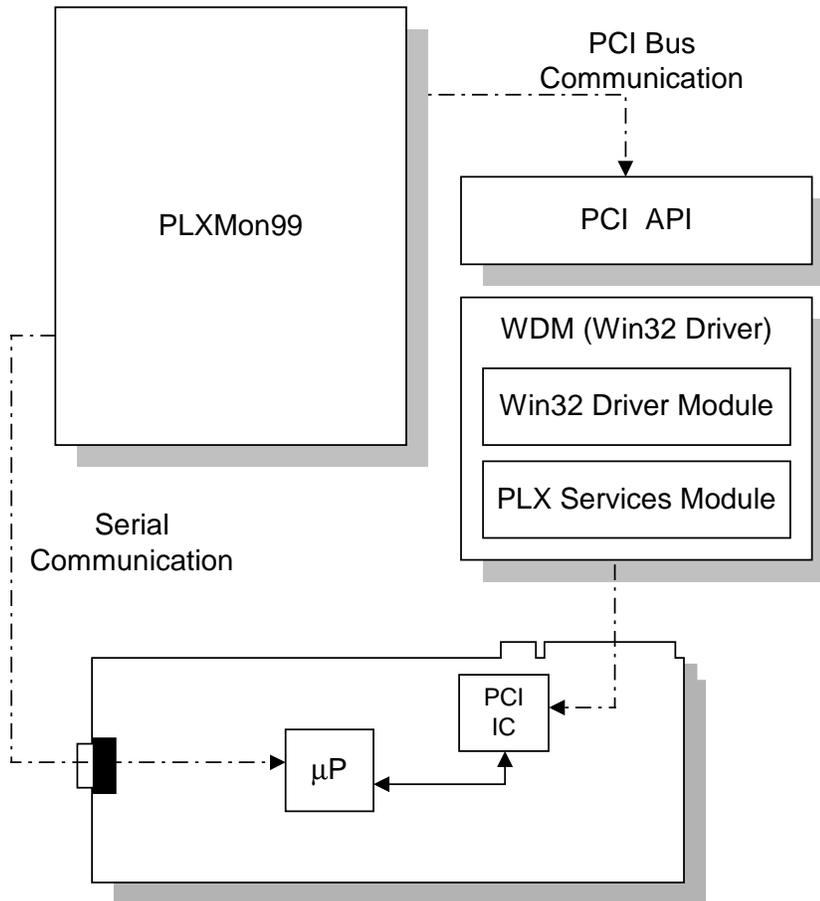


Figure 3-8 The Host Software Architecture

3.5.1.1 Serial Communication

This method of communicating with a PLX RDK 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 RDK board.

If PLXMon 99 is set to serial mode, it calls functions that reside in the PLXMon 99 Communications Module. It is the responsibility of the PLXMon 99 Communications Module code to convert the valid PLXMon 99 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. 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 RDK board's microprocessor must have a means of handling the incoming data. The Back-End Monitor calls UART service routines to retrieve data from the UART module, which can be implemented as interrupt-driven or polling. 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 dynamical-link library (DLL) file (PLXApi.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 RDK boards can be accessed and used. The PCI API provides API function groups, which manage the features of each PLX chip. Groups such as DMA access, direct data transfers, and interrupt handling contain functions that can be universal to any PCI RDK board.

The PLXMon 99 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 connect to the various PLX device drivers. This includes opening, closing, and searching for PLX 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 and Windows NT 2000 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 extensive 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 chip 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 Reference Design Boards, the following device driver model was created.

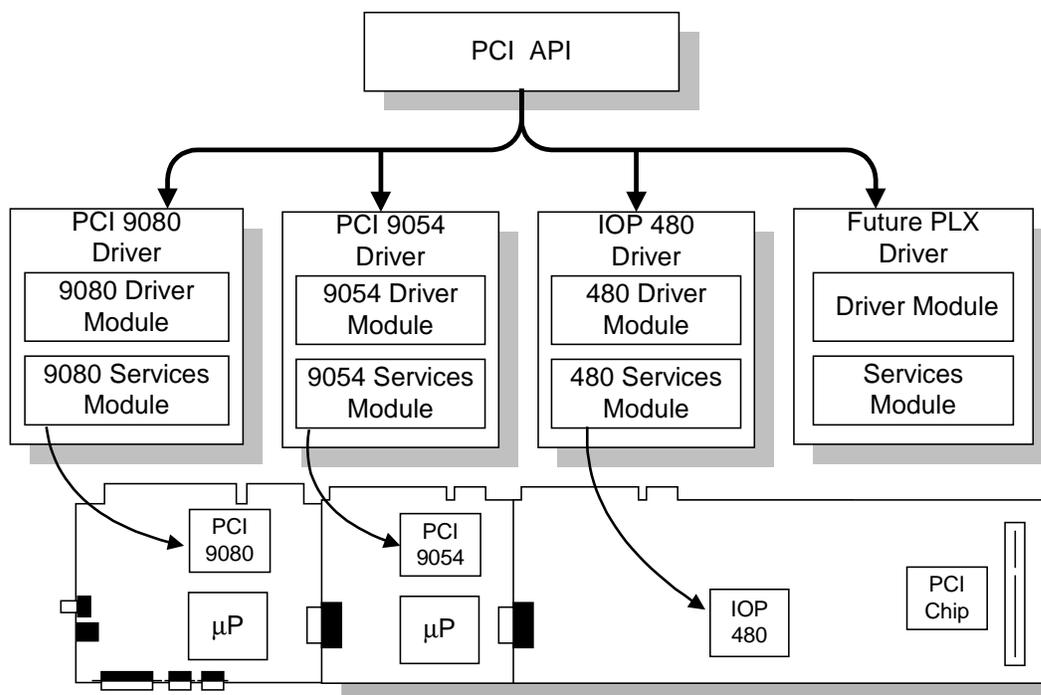


Figure 3-9 The PLX Device Driver Layout

One device driver handles one type of PLX chip, 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 chip is present on a Reference Design Board, the device driver can only see the one that is directly connected to the PCI bus on which the Windows system is running. All PCI API functions will access this PLX chip only.

3.5.3.1 PLX Chip Device Driver Module

This module provides the management of the PCI RDK boards in Windows NT/98. This management includes storing device specific information, processing PCI API and system messages, handling interrupts, and allocating resources for each RDK 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 PLX Chip Services Module

This module has access to the entire register set of the PLX chip, 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 PLX chip Services Module, which provides the real functionality for the device driver, is updated to support a new PLX chip, the old PLX chip Services Module is replaced. The new PLX chip Services Module would reflect the new register

set of the PLX chip and would support the existing PCI API by accessing the appropriate registers on the new PLX chip based on the PCI API function requested.

The PLX chip Device Driver Module would need some modifications to create a new PCI device driver. When a new API function is introduced, the Device Driver Module has to be modified to support the API the function.

3.5.5 Device Driver Features

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

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.

3.5.6 Distribution of PLX Device Drivers and PLXApi.Dll File

A Windows application, which uses PCI API functions from the PCI SDK package, requires 2 things:

1. the `PlxApi.dll` file installed in the system-wide path; and
2. PLX Windows device driver(s) installed properly and started.

It is legal to distribute the `PlxApi.dll` and device driver files, i.e. `Pci9080.sys`, `Pci9054.sys` and `Iop480.sys`, however it is ILLEGAL to distribute the entire PCI SDK without proper authorization by PLX. Customers must choose their own installation methods, such as using InstallShield® software to create installation disks, so that the device driver(s) and the DLL file can be correctly installed to their designated locations.

3.5.6.1 Installation of PlxApi.dll File

The `PlxApi.dll` file should be installed to:

- `Windows\System` directory on Windows 98; or
- `WinNT\System32` directory on Windows NT.

Note: “`Windows\`” and “`WinNT\`” directories mentioned in the above two lines are the default installation paths for Windows 98 and Windows NT respectively. Therefore, it has to be determined at run time by the installation program where the Windows 98 or Windows NT directory is really located and it is strongly recommended that the above paths NOT be hard-coded in the installation program. In InstallShield® terminology, the above “`Windows\`” and “`WinNT\`” are denoted as `<WINDIR>`.

3.5.6.2 Installation of PLX Device Driver

The installation of a device driver involves the following steps:

- Copy relevant files to their proper destination locations; and



- Set up registry entries on the Windows NT system (Refer to section 2.2.2.2.3 on page 2-12 for registry entry description).

3.5.6.2.1 Installation of PLX Device Drivers On Windows NT

1. Copy the device driver file (`Pci9080.sys`, `Pci9054.sys`, or `Iop480.sys`) to `WinNT\System32\Drivers` directory. The device driver files are located at `<INSTALLPATH>\Win32\Driver\WinNT\<DriverName>\i386\free` directory, where **DriverName** is `Pci9080`, `Pci9054`, or `Iop480`;
2. Add the following registry entries (`DriverName = Pci9080`, `Pci9054`, or `Iop480`)
Under `HKEY_LOCAL_MACHINE\System\CurrentControlSet\Services`, add:
 - `Services\<DriverName>\CommonBufferSize = 65536` in decimal, data type `REG_DWORD`;
 - `Services\<DriverName>\ErrorControl = 1`, data type `REG_DWORD`;
 - `Services\<DriverName>\Start = 2`, data type `REG_DWORD`;
 - `Services\<DriverName>\Type = 1`, data type `REG_DWORD`;
 - `Services\<DriverName>\MaxSglTransferSize = 1048576` in decimal, data type `REG_DWORD`;
 - `Services\<DriverName>\MaxPciBus = 3`, data type `REG_DWORD`
 - `Services\<DriverName>\EventLogLevel = 2`, data type `REG_DWORD`
 - `Services\<DriverName>\SupportedIDs = Dev0Vend Dev1Vend ...`, data type `REG_SZ`. `Dev0` and `Dev1` are the four hexadecimal letters for the device IDs, `Vend` is the four hexadecimal letters for the vendor ID. One example is "908010B5 040110B5". One pair of device ID and vendor ID must be separated from another pair by a space. There is no trailing space at the end of the string.
 - `Control\Session Manager\Memory Management\SystemPages = 80000` in decimal, data type `REG_DWORD`.

The registry key `<DriverName>` is the name of the executable without the `.sys` suffix.

In order for the Event Viewer under Windows NT to retrieve the messages logged by the PLX device drivers, the following registry values must be added as well.

Under `HKEY_LOCAL_MACHINE\System\CurrentControlSet\Services\`, add:

- `EventLog\System\<DriverName>\EventMessageFile = %SystemRoot%\System32\IoLogMsg.dll;%SystemRoot%\System32\drivers\<DriverName>.sys`, data type `REG_EXPAND_SZ`;
- `Services\EventLog\System\<DriverName>\TypesSupported = 7`, data type `REG_DWORD`;

3.5.6.2.2 Installation of PLX Device Drivers On Windows 98

1. Copy the device driver file (`Pci9080.sys`, `Pci9054.sys`, or `Iop480.sys`) to `Windows\System` directory. The device driver files are located at

<INSTALLPATH>\Win32\Driver\WDM**<DriveName>**\i386\free directory, where **DriveName** is Pci9080, Pci9054, or Iop480;

2. Copy the setup information file (INF) PciSdk.inf to Windows\Inf\Other directory. The PciSdk.inf file is in the <INSTALLPATH>\WIN32\Driver\WDM directory.

The PciSdk.inf must be modified first to include customized device and vendor IDs. When Windows is rebooted, the operating system will detect a new PCI card and should use the INF file to assign the correct driver to handle the PCI card. Refer to Windows documentation for INF files or consult book *Inside The Microsoft Windows 98[®] Registry* written by Günter Born, published in 1998 by Microsoft Press, ISBN 1-57231-824-4.



This page is intentionally left blank.

4. Real Time Operating System Support

4.1 General Information

This PCI SDK contains VxWorks BSP for PCI 9054RDK-860 and CompactPCI 9054RDK-860 boards. We will provide a VxWorks BSP for IOP 480RDK in the near future. Please send an email if you are interested in VxWorks BSP for IOP 480RDK to software@plxtech.com

If you are interested in developing a driver for other RTOS, then you can use almost all of the IOP API library functions from this version of the PCI SDK. The IOP API library is a set of functions that can be called from within a RTOS. The RTOS code needs to call the `PlxInitApi()` function to initialize the PLX API. This will integrate the IOP API with your RTOS. You may use this chapter as a reference.

4.2 Getting Started

This chapter assumes that you are familiar with PLX PCI 9054RDK or CompactPCI 9054RDK-860 reference design boards, PLX PCI SDK 3.0, and WindRiver's Tornado. Please refer to the PLX PCI 9054RDK-860 Hardware Reference Manual or CompactPCI 9054RDK-860 Hardware reference Manual for more details about the PLX reference design boards.

4.3 Minimum Requirements

Minimum requirements for the VxWorks IOP Porting Kit Extension for PCI/CPCI 9054RDK-860:

- PLX PCI 9054RDK-860 or CompactPCI 9054RDK-860 reference design board;
- PLX PCI SDK v3.0;
- Windows NT or Windows 98 based PC;
- WindRiver Tornado v1.0.1 or v2.0 Integrated Development Environment;
- Windows HyperTerminal using Xmodem protocol. (Baud rate 38400, 8 data bits, no parity, 1 stop bits, no flow control)

4.4 Installation

- Turn the power off on your Windows PC.
- Install either the PLX PCI 9054RDK-860 or CompactPCI 9054RDK-860 reference design board or your PCI 9054 based prototype board in an empty PCI slot in your PC.
- Turn your PC on.
- Install the PLX PCI SDK and the Tornado software, and
- reboot your PC.

Make a `plx9054-860` directory in `<TORNADO_INSTALLPATH>\tornado\target\config\` directory (`TORNADO_INSTALLPATH` is where you have installed the Tornado software). The "`<INSTALLPATH>\Rtos\VxWorks BSP`" contains two folders: "PCI 9054RDK-860" (for the



PCI 9054RDK-860 board) and “CPCI 9054RDK-860” (for the CompactPCI 9054RDK-860 board). Copy the appropriate files from the “<INSTALLPATH>\Rtos\VxWorks BSP” subdirectory into <TORNADO_INSTALLPATH>\tornado\target\config\ directory.

4.5 What’s Included?

<TORNADO_INSTALLPATH>\tornado\target\config\plx9054-860 directory will include the following source code files:

1. API860_vw.a: this is the PLX API library object file.
2. rominit.s, sysalib.s, sysserial.c, ppc860sio.c, syslib.c files initialize and configure the Motorola MPC860, connect SCC1 interrupt handler to handle interrupt from RXD1 pin in MPC860 port PA15, load VxWorks kernel and setup communication protocol with HyperTerminal or Tornado Host Shell.

NOTE:

In rominit.s file insert a line “#define BSP_CPCI9054” for PLX CompactPCI9054RDK-860, insert a line “#undef BSP_CPCI9054” for PCI 9054RDK-860.

3. plxvwdrv1.c, plxvwdrv2.c files initialize and configure the PCI 9054 chip, initialize the PLX API library, DMA resource manager, and connect interrupt handler for external interrupt IRQ1 pin in MPC860 from LINTO of PCI 9054 and IDMA2 interrupt handler for IDMA request from DREQ1 in MPC860 Port PC14.
4. plxvwapp1.c, plxvwapp2.c files create the sample test application. These samples demonstrate how to use the PLX API library to test Direct Master and DMA capabilities of the PCI 9054 chip and IDMA capability of the MPC860.
5. usrconfig.c file: In usrRoot() routine : Insert the line “plxdemo();” before the line “shellInit();” in order to run the plxdemo as shown in figure 1. Insert the line “plxinit();” before the line “taskSpawn();”, so that PCI 9054 chip will be initialized correctly after rebooting vxWorks_rom and it will allow you to download a new flash image using PLXMon 99.
6. VxWorks_rom (ELF format) and VxWorks_rom.hex (S record format) are the ROM code that can communicate with the Tornado Host Shell.
7. VxWorks.res_rom (ELF format) and vxWorks.res_rom.hex (S record format) are the standalone ROM code, acts as a VxWorks target shell. VxWorks.res_rom code calls the plxdemo program. plxdemo allows you to test the Direct Master, DMA and IDMA features from the Windows HyperTerminal.
8. makefile is the makefile to make last two set of ROM files.
9. Plxdemo.make creates the vwPlxDemo object file. This file can be loaded from the Tornado Host Shell into the local RAM on the PCI 9054RDK-860 or CompactPCI 9054RDK-860 board.
10. Config.h:
 - #define TARGET_SHELL to generate vxWorks.res_rom;
 - #undef TARGET_SHELL to generate vxWorks_rom.

4.6 Which VxWorks ROM Image Should I Use?

Two special versions of the PCI 9054 RDK boot code are provided in the directory `<TORNADO_INSTALLPATH>\tornado\target\config\plx9054-860`.

1. `VxWorks_rom.hex` is the ROM code that can communicate with the Tornado Host Shell.
2. `VxWorks.res_rom.hex` is the standalone ROM code, which acts as a VxWorks target shell.

If you don't have WindRiver Tornado tools yet, you can follow section 4.7.2.1 to download the `vxWorks.res_rom.hex` to the reference design board and exercise the `plxdemo` program from the HyperTerminal connected to the serial Port.

If you have WindRiver Tornado tools available, you can follow section 4.7.2.2 to download `vxWorks_rom.hex`, use `plxdemo.make` to make `vwPlxDemo` file, download this file to the PCI 9054RDK-860's or CompactPCI 9054RDK-860's RAM, and execute the `plxdemo` program from the Tornado Shell and Target server console connected to the serial Port.

If you don't have a PCI 9054RDK-860 or CompactPCI 9054RDK-860 board, and you like to integrate the PCI 9054 chip into your design, please refer to section 4.5 and make the necessary modifications to the files listed there.

4.7 PLX VxWorks BSP/PLX API Demonstration

This section will show the user how to use the PLX VxWorks BSP (Board Support Package), download the VxWorks boot ROM and application program images included in the PLX PCI SDK v3.0, and use the PCI 9054 chip under the VxWorks Real Time Operating System (RTOS).

4.7.1 Updating the PCI 9054RDK-860 or CompactPCI 9054RDK-860 onboard FLASH

Start the `PLXMon 99` utility (included in the PLX PCI SDK). Select Command from the pull down menu. Click on "Download to IOP". Choose a file you want to download (please refer to section 3.3, you can pick either `vxWorks_rom.hex` or `vxWorks.res_rom.hex` S record files). Choose "FLASH Device" as your Device Type. Select "S record" as your Image Type. Select memory Offset in Hex as "00000" and hit "Download" to update your FLASH to VxWorks boot ROM.

Note: It is very important that you refer to PLXMon 99 User's Manual before reprogramming the FLASH.

4.7.2 PLX API functions Demonstration

The VxWorks BSP demonstration consists of two parts. The first part assumes that you have no Tornado and no Target Server. The second part assumes that you have the Tornado Host Shell.

4.7.2.1 Stand alone VxWorks Target Shell Demo (No Tornado and no Target Server present)

Update the PCI 9054DK-860's or CompactPCI 9054RDK-860's FLASH with `vxWorks.res_rom.hex`, you can execute the `plxdemo` without Tornado tools running on your host system.



Figure 4-2 Reboot the Stand Alone VxWorks Target Shell Demo

1. Attach a serial cable from the PCI 9054DK-860 or CompactPCI 9054RDK-860 board to the serial port of the host development system. The RS232 port communicates using Baud rate 38400, 8 data bits, no parity, 1 stop bits, no flow control.
2. In Host Tornado Integrated Development Environment, select "Tools" from the pull down menu, choose "Target Server", then "Configure..." as shown in Figure 4-3.
3. At the "Configure Target Servers" dialog box, enter the following information for the given fields: - "Description": Configuration1, "Target Server Name": plx1.
4. Select the "Change Property" item to be: Back End, and enter the following information for the given fields: - "Available back Ends": wdbserial, "Serial Port": this depends on your system, "Speed": 38400, "Log File": User specified.
5. Select the "Change Property" item to be: Core file and symbols, and perform the following, select the "File" radio button, and then
 <TORNADO_INSTALLPATH>\tornado\target\config\plx9054-860\vxWorks_rom.
 (Please see Figure 4-4).

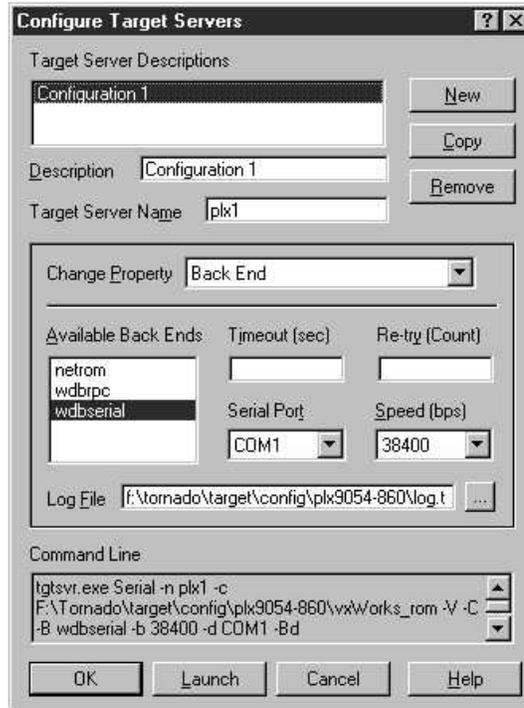


Figure 4-3 Back End Property Page

6. Select the “Change Property” item to be: Virtual Console, and select the “use Target Server Windows as Virtual Console” radio button. See Figure 4-5.

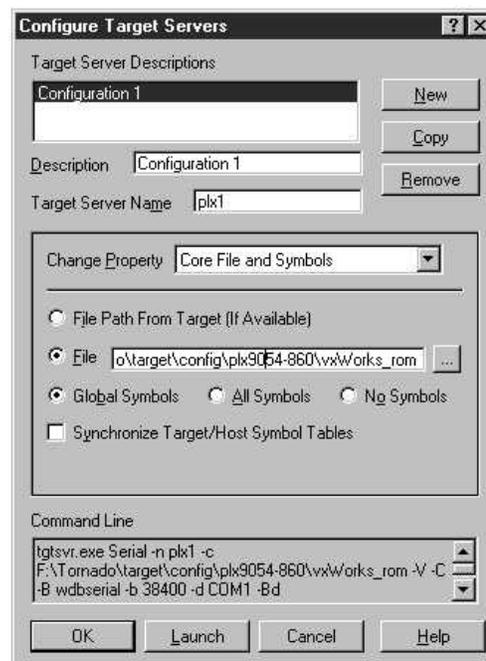


Figure 4-4 Core File and Symbols Property Page

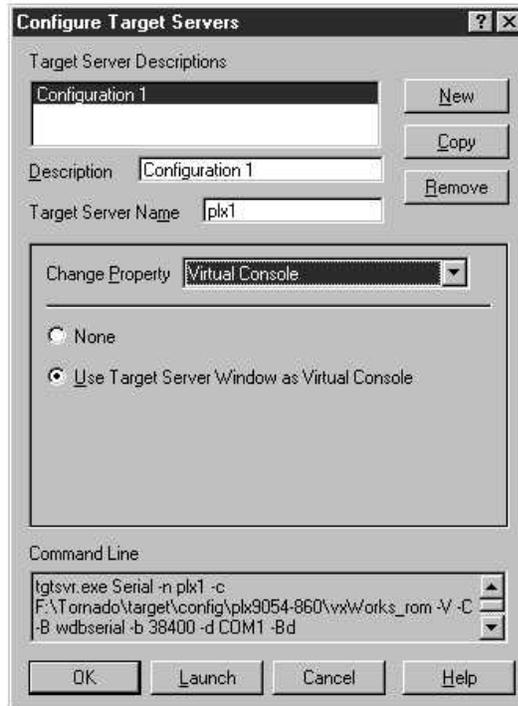


Figure 4-5 Virtual Console Property Page

7. Select the Launch button, the following text window should appear (see Figure 4-6):

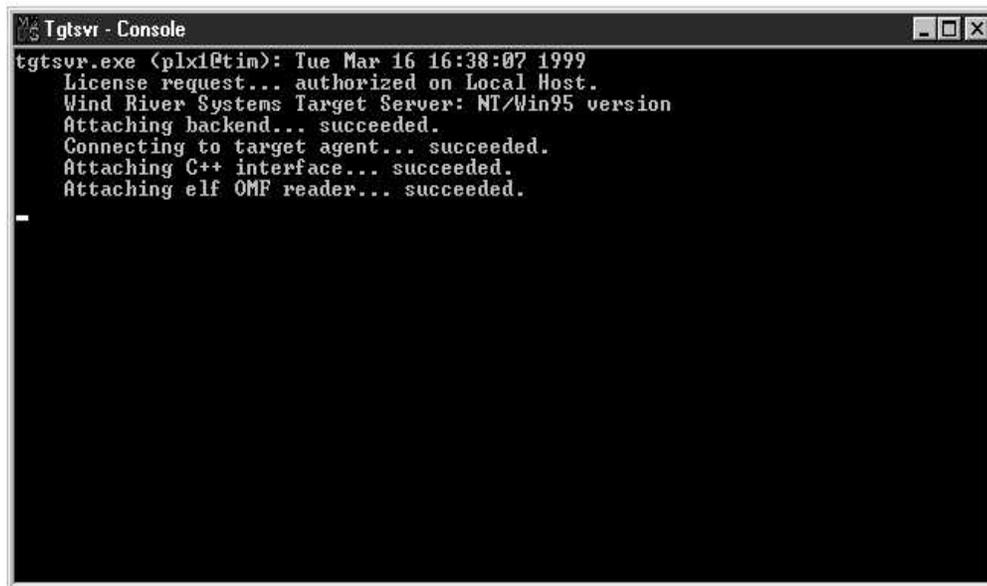


Figure 4-6 Tgtsvr - Console

8. If the previous output text is displayed, the host tornado target server is successfully talking to the target agent on the boot ROM and you can continue to step 9. If not, re-check your configuration information and launch the target server again.

Note: If the local bus of the PCI 9054RDK-860 or CompactPCI 9054RDK-860 board is hung, then re-launching the target server will fail. You need to reset the target system if this happens.

9. In Tornado, select the Tools menu title, and choose the Shell... menu item.
10. At the Shell Launch dialog box, select the OK button.
11. A Shell window should be displayed with a -> prompt, type "lkup", if you see the symbols for PLX IOP API functions listed in PLX PCI SDK (examples: PlxBusPciRead(), PlxDmaIsr(), etc.), you know the PLX vxWorks BSP boot ROM has been installed correctly.

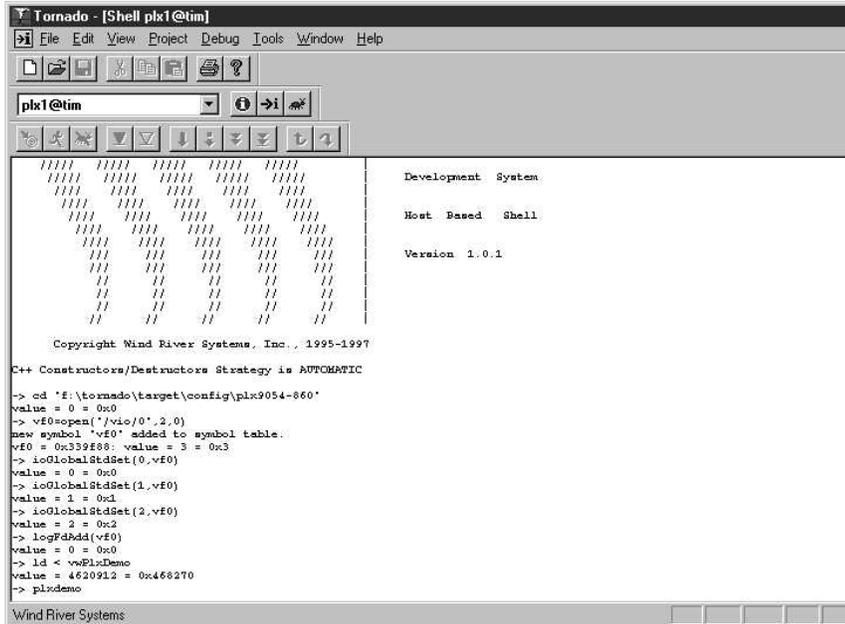


Figure 4-7 Tornado Shell Prompt

12. At the Shell prompt, type "cd <TORNADO_INSTALLPATH>\target\config\plx9054-860", then perform the following commands to reset standard input, standard output, and standard error to virtual I/O channel 0 (in our case, it will go to target server console), also send logging output to the same virtual device. (refer to WindRiver Tornado User's guide)

- ➔ vf0=open("/vio/0",2,0)
- ➔ ioGlobalStdSet(0,vf0)
- ➔ ioGlobalStdSet(1,vf0)
- ➔ ioGlobalStdSet(2,vf0)
- ➔ logFdAdd(vf0)

or you can type " < start " to save some time. ("start" is a script file included)

13. From shell, You can run PLX IOP API functions, examples:
 - type "PlxUserWrite(2,0,1)" to turn on PLX RDK LED
 - type "PlxUserWrite(2,0,0)" to turn off PLX RDK LED

4.8 How to rebuild the BSP and Application images?

This section will show the user how to rebuild the VxWorks boot ROM, driver and application program from the source code included in this SDK under the WindRiver Tornado Integrated Development Environment.

4.8.1 Setup the makefile to build PLX VxWorks Boot ROM

Copy the makefile included into the “<TORNADO_INSTALLPATH>\target\config\plx9054-860” directory. Now start the Tornado. You will see the entry “Make PLX9054-860” under the Tornado “Project” menu item. Choose “Make Plx9054-860”, then “VxWorks Target”, then either “vxWorks_rom.hex” to build Host shell version bootrom, or choose “vxWorks.res_rom.hex” to build resident stand alone boot ROM. Other entries are not fully tested in this release.

Note:

1. *Modify Config.h to add “#define TARGET_SHELL” and generate vxWorks.res_rom, and add “#undef TARGET_SHELL” to generate vxWorks_rom.*
2. *In rominit.s insert a line “#define BSP_CPCI9054” for PLX CompactPCI9054RDK-860, insert a line “#undef BSP_CPCI9054” for PCI 9054RDK-860.*

4.8.2 Setup the custom project to build PLX demo application

1. In Tornado, select the Project menu title.
2. In the Customize Builds dialog box, choose Add button, and Browse button to open the plxdemo.make file, edit the Menu Text window, Build target window as shown in Figure 4-10 and Figure 4-11. This completes the setup for building the VxWorks application.

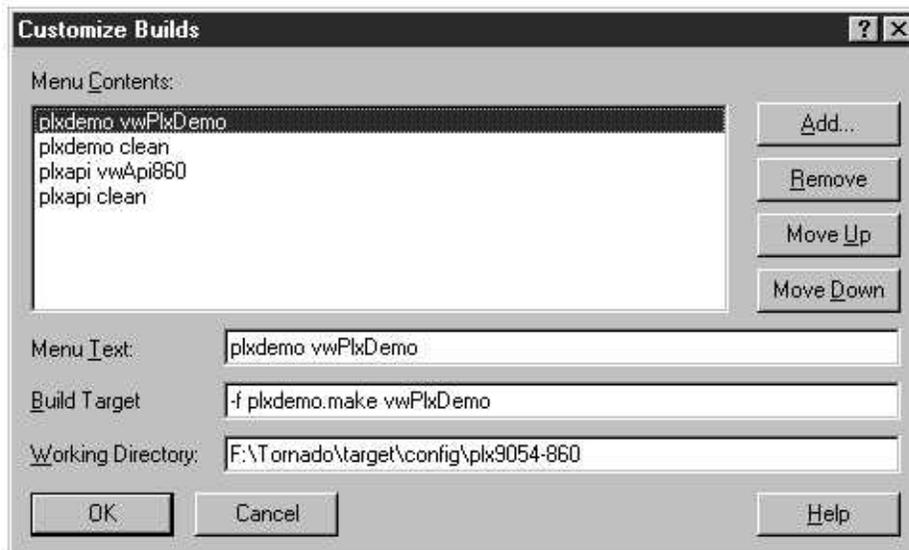


Figure 4-10 Customize Builds Screen No. 1

3. “Plxdemo vwPlxDemo” to create application image vwPlxDemo.

4. “Plxdemo clean” to remove image vwPlxDemo and related object files.

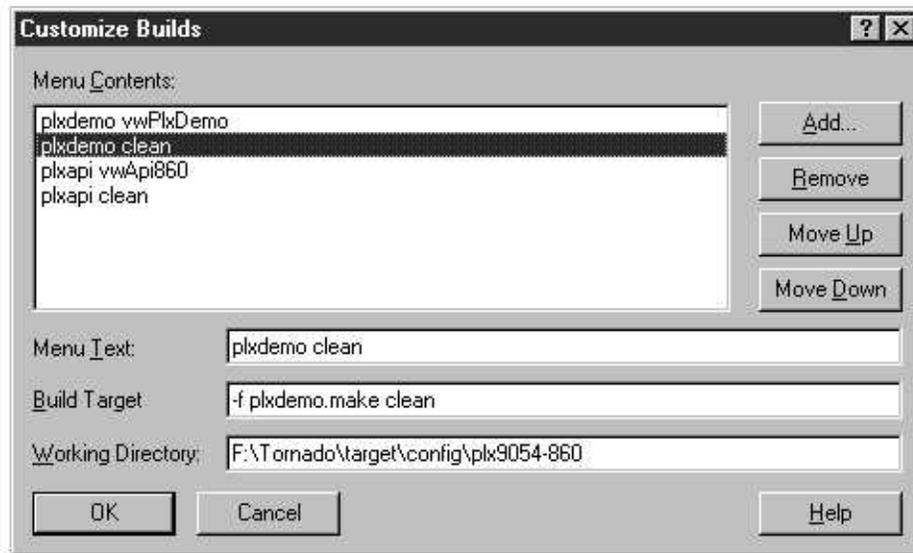


Figure 4-11 Customize Builds Screen No. 2

4.9 Tornado 1.0.1 and Tornado 2.0 compatibility

WindRiver provides compatibility between Tornado 2.0 and 1.0.1. However, you may have to do some minor configuration changes in Tornado 2.0 to make it compatible with Tornado 1.0.1

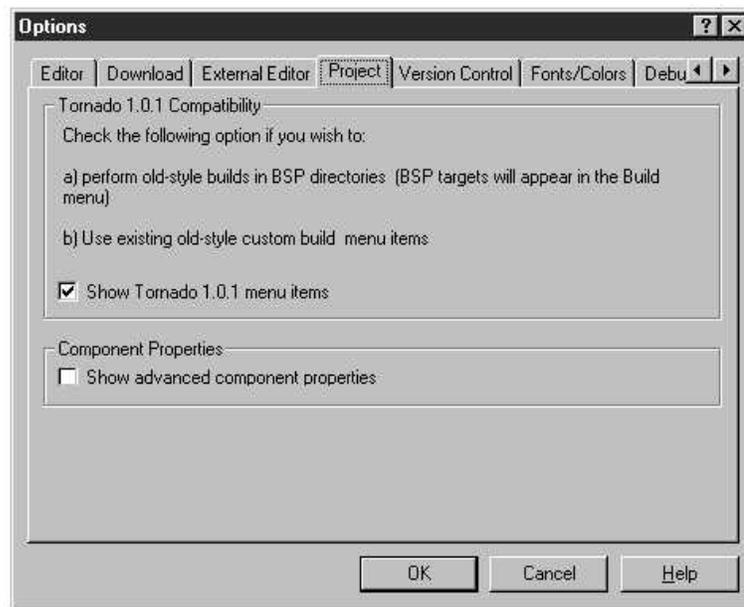


Figure 4-12 Options

1. After start Tornado 2.0, go to Tools-Options-Project, Check the ”Show Tornado 1.0.1 menu item” box, this will causes a customize item “standard BSP Build” to be added to the Build

menu. This allows you to perform Tornado 1.0.1-style builds in BSP directories, to use existing Tornado 1.0.1-style Build menu items, and to create additional Build menu customizations. (Figure 4-12)

2. Copy and modify `usrconfig.c` file from
`<TORNADO_INSTALLPATH>\tornado20\target\config\all`.
3. Tornado supports virtual I/O to the host from target applications. “Redirect target I/O” “redirect the target global `stdin`, `stdout`, and `stderr` to the target server. “Create Console Window” create a virtual console window on the target server host for target I/O. “Redirect target Shell” starts a console window into which the target shell’s standard input, output, and error will be redirected.

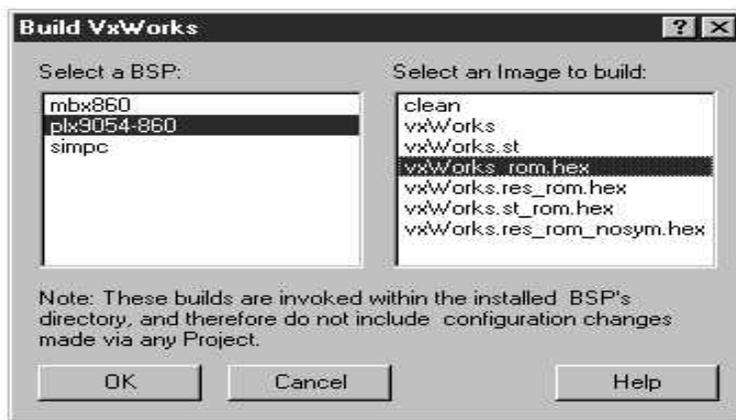


Figure 4-13 Build VxWorks

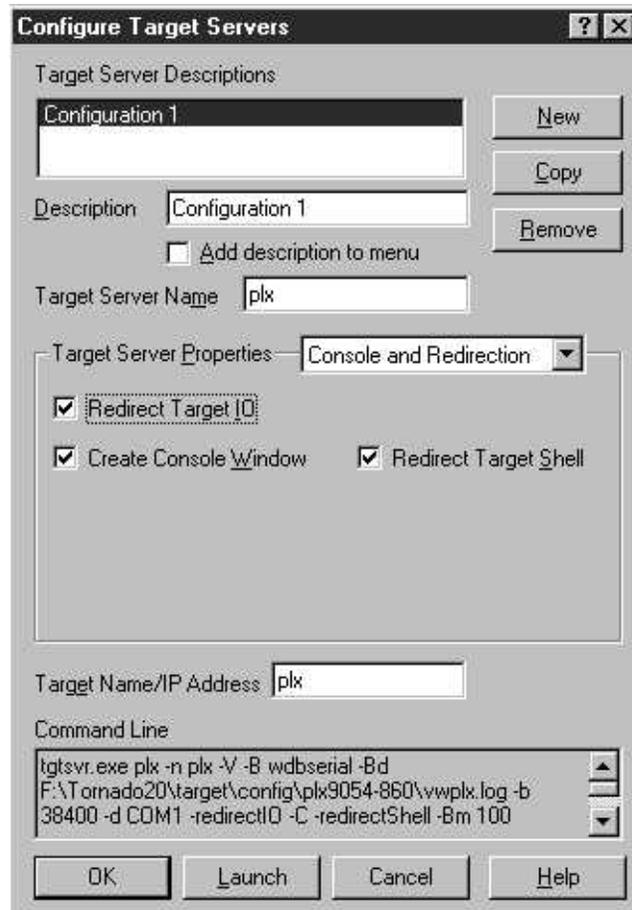


Figure 4-14 Configure Target Servers



This page is intentionally left blank.

5. RDK Software Quick Reference

The information below should be used as a guide to the PCI SDK software requirements for each PLX RDK board.

5.1 IOP 480RDK

The table below describes the memory requirements of the PCI SDK for the IOP 480RDK.

Table 5-1. Basic Information About IOP 480RDK

Item	IOP 480RDK
SDRAM Address	0x0000 0000 -> 0x01FF FFFF (32MB)
FLASH Type	ATMEL AT49LV040
FLASH Address	0xFFFF8 0000 -> 0xFFFF FFFF (512KB)
FLASH Image Offset at which to reprogram the FLASH	0x0006 0000 (If the user has a bigger FLASH image, this has to be decreased to proper offset.)
EEPROM Type	93CS66LEN
Memory space used by ROM code	0x0000 0000 -> 0x0003 FFFF
Memory space used by RAM code (PCI SDK Samples)	0x0004 0000 -> 0x0007 FFFF
Direct Master to PCI memory space Address	0x4000 0000 -> 0x4FFF FFFF (256MB)
Direct Master to PCI IO space Address	0x5000 0000 -> 0x5FFF FFFF (256MB)
IOP 480 Configuration Register Base Address	0x5000 0000 (default and EEPROM settings)
IOP 480 Configuration Register Base Address in PCI SDK	0x3000 0000 (re-initialized from default and used in SDK)
IOP 480 Serial Port Unit Base Address	0x1000 0000
Memory space recommend for user's use.	0x0008 0000 -> 0x01FF FFFF (31.5MB)

Users, who have upgraded the PCI SDK and intend to use it with an existing PLX RDK, must first reprogram the configuration EEPROM connected to the PLX chip. Skipping this step can cause unpredictable behavior of the PCI SDK. If you have purchased the PCI SDK with a PLX RDK then there is no need to upgrade the EEPROM because it was programmed at the factory before shipping. The following diagram shows the default EEPROM settings for the RDK.

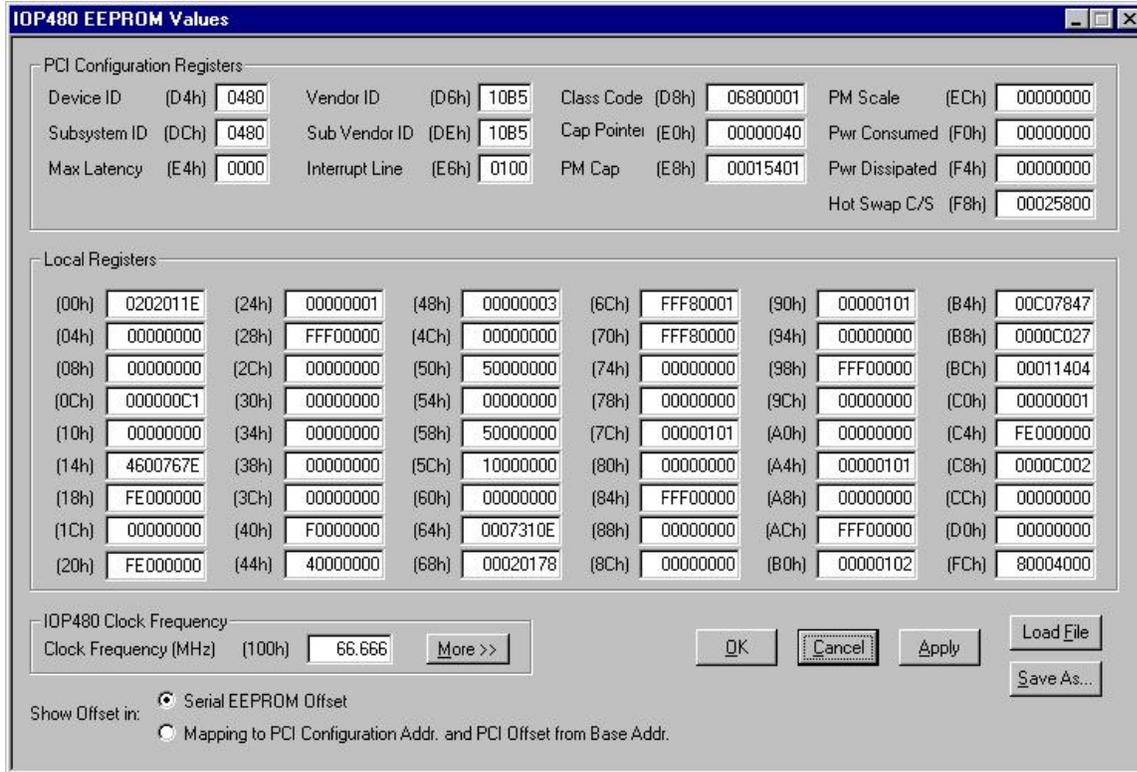


Figure 5-1. Configuration EEPROM Settings for the IOP 480RDK

5.2 PCI 9054RDK-860

The table below describes the memory requirements of the PCI SDK for the PCI 9054RDK-860.

Table 5-2. Basic Information About PCI 9054RDK-860

Item	PCI 9054RDK-860
SDRAM Address	0x0000 0000 -> 0x01FF FFFF (32MB)
FLASH Type	AT49LV040(default as shipped) AM29F040 also supported in software
FLASH Address	0xFFFF0 0000 -> 0xFFFF7 FFFF (512KB)
FLASH Image Offset at which to reprogram the FLASH	0x0000 0000
Memory space used by ROM code	0x0000 0000 -> 0x0003 FFFF
Memory space used by RAM code (PCI SDK Samples)	0x0004 0000 -> 0x0007 FFFF
Direct Master to PCI memory	0x4000 0000 -> 0x40FF FFFF

Item	PCI 9054RDK-860
space Address	(16MB)
Direct Master to PCI IO space Address	0x5000 0000 -> 0x50FF FFFF (16MB)
PCI 9054 Register Address	0x3000 0000 -> 0x3000 01FF
Memory space recommend for user's use.	0x0008 0000 -> 0x01FF FFFF (31.5MB)

Users, who have upgraded the PCI SDK and intend to use it with an existing PLX RDK, must first reprogram the configuration EEPROM connected to the PLX chip. Skipping this step can cause unpredictable behavior of the PCI SDK. If you have purchased the PCI SDK with a PLX RDK then there is no need to upgrade the EEPROM because it was programmed at the factory before shipping. The following diagram shows the default settings for the RDK.

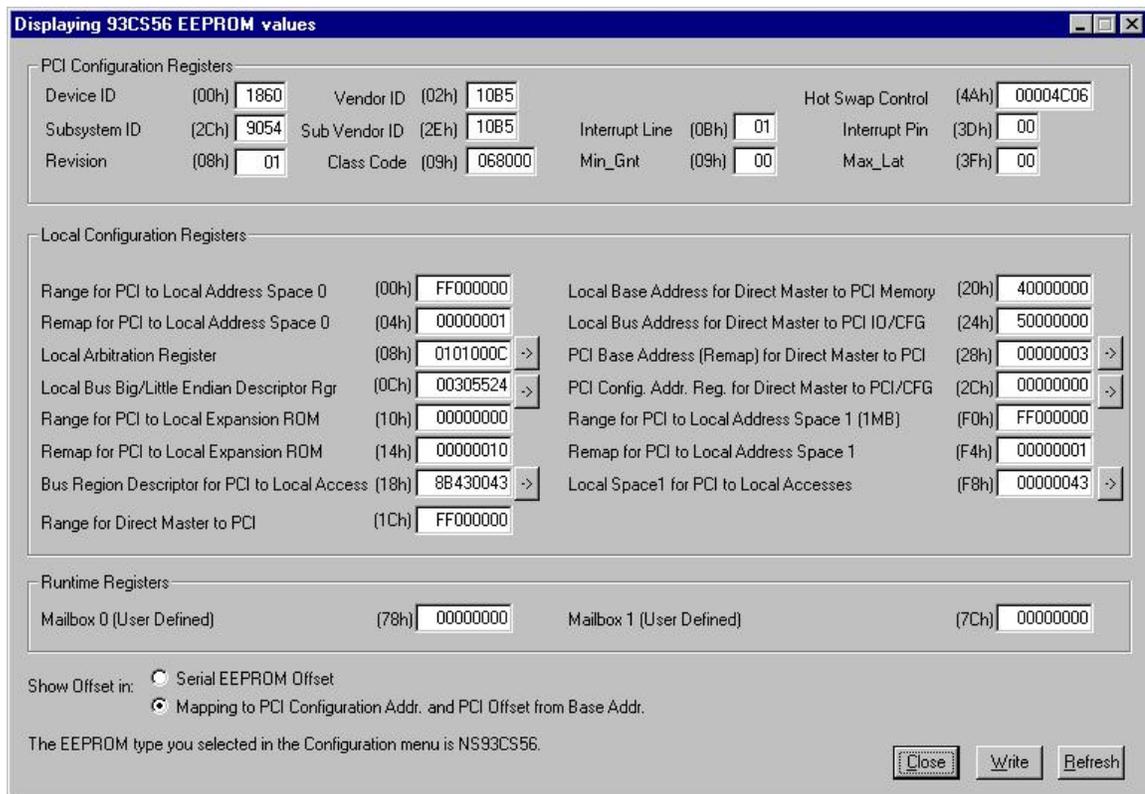


Figure 5-2. Configuration EEPROM Settings for the PCI 9054RDK-860

5.3 CompactPCI 9054RDK-860

The table below describes the memory requirements of the PCI SDK for the CompactPCI 9054RDK-860.



Table 5-3. Basic Information About CompactPCI 9054RDK-860

Item	CompactPCI 9054RDK-860
SDRAM Address	0x0000 0000 -> 0x01FF FFFF (32MB)
SBSRAM Address	0x2000 0000 -> 0x2007 FFFF (512KB)
FLASH Type	AT49LV040(default as shipped) AM29F040 also supported in software
FLASH Address	0xFFF0 0000 -> 0xFFF7 FFFF (512KB)
FLASH Image Offset at which to reprogram the FLASH	0x0000 0000
Memory space used by ROM code	0x0000 0000 -> 0x0003 FFFF
Memory space used by RAM code (PCI SDK Samples)	0x0004 0000 -> 0x0007 FFFF
Direct Master to PCI memory space Address	0x4000 0000 -> 0x40FF FFFF (16MB)
Direct Master to PCI IO space Address	0x5000 0000 -> 0x50FF FFFF (16MB)
PCI 9054 Register Address	0x3000 0000 -> 0x3000 01FF
Memory space recommend for user's use.	0x0008 0000 -> 0x01FF FFFF (31.5MB)

Users, who have upgraded the PCI SDK and intend to use it with an existing PLX RDK, must first reprogram the configuration EEPROM connected to the PLX chip. Skipping this step can cause unpredictable behavior of the PCI SDK. If you have purchased the PCI SDK with a PLX RDK then there is no need to upgrade the EEPROM because it was programmed at the factory before shipping. The following diagram shows the default settings for the RDK.

Displaying 93CS56 EEPROM values

PCI Configuration Registers

Device ID (00h) Vendor ID (02h) Hot Swap Control (48h)
 Subsystem ID (2Ch) Sub Vendor ID (2Eh) Interrupt Line (3Ch) Interrupt Pin (3Dh)
 Revision (08h) Class Code (09h) Max_Lat (3Fh) Min_Gnt (3Eh)

Local Configuration Registers

Range for PCI to Local Address Space 0 (00h) Local Base Address for Direct Master to PCI Memory (20h)
 Remap for PCI to Local Address Space 0 (04h) Local Bus Address for Direct Master to PCI IO/CFG (24h)
 Local Arbitration Register (08h) > PCI Base Address (Remap) for Direct Master to PCI (28h) >
 Endian-Local Misc-VPD Boundary Register (0Ch) > PCI Config. Addr. Reg. for Direct Master to PCI/CFG (2Ch) >
 Range for PCI to Local Expansion ROM (10h) Range for PCI to Local Address Space 1 (1MB) (F0h)
 Remap for PCI to Local Expansion ROM (14h) Remap for PCI to Local Address Space 1 (F4h)
 Bus Region Descriptor for PCI to Local Access (18h) > Local Space1 for PCI to Local Accesses (F8h) >
 Range for Direct Master to PCI (1Ch)

Runtime Registers

Mailbox 0 (User Defined) (78h) Mailbox 1 (User Defined) (7Ch)

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.

Figure 5-3. Configuration EEPROM Settings for the CompactPCI 9054RDK-860

5.4 PCI 9080RDK-401B

The table below describes the memory requirements of the PCI SDK for the PCI 9080RDK-401B.

Table 5-4. Basic Information About PCI 9080RDK-401B

Item	PCI 9080RDK-401B
SDRAM Address	0x0000 0000 -> 0x00FF FFFF (16MB)
SRAM Address	0x1000 0000 -> 0x1007 FFFF (512KB)
FLASH Type	AM29F040
FLASH Address	0xFFFF8 0000 -> 0xFFFF FFFF (512KB)
FLASH Image Offset at which to reprogram the FLASH	0x0006 0000 (If the user has a bigger FLASH image, this has to be decreased to proper offset.)
Memory space used by ROM code	0x1000 0000 -> 0x1003 FFFF
Memory space used by RAM code	0x1004 0000 -> 0x1007 FFFF

Item	PCI 9080RDK-401B
(PCI SDK Samples)	
Direct Master to PCI memory space Address	0xB800 0000 -> 0xBFFF FFFF (8MB)
Direct Master to PCI IO space Address	0xB000 0000 -> 0xB7FF FFFF (8MB)
PCI 9080 Register Address	0x2000 0000 -> 0x2000 01FF
Memory space recommend for user's use.	0x0 -> 0x00FF FFFF (16MB)

Users, who have upgraded the PCI SDK and intend to use it with an existing PLX RDK, must first reprogram the configuration EEPROM connected to the PLX chip. Skipping this step can cause unpredictable behavior of the PCI SDK. If you have purchased the PCI SDK with a PLX RDK then there is no need to upgrade the EEPROM because it was programmed at the factory before shipping. The following diagram shows the default settings for the RDK.

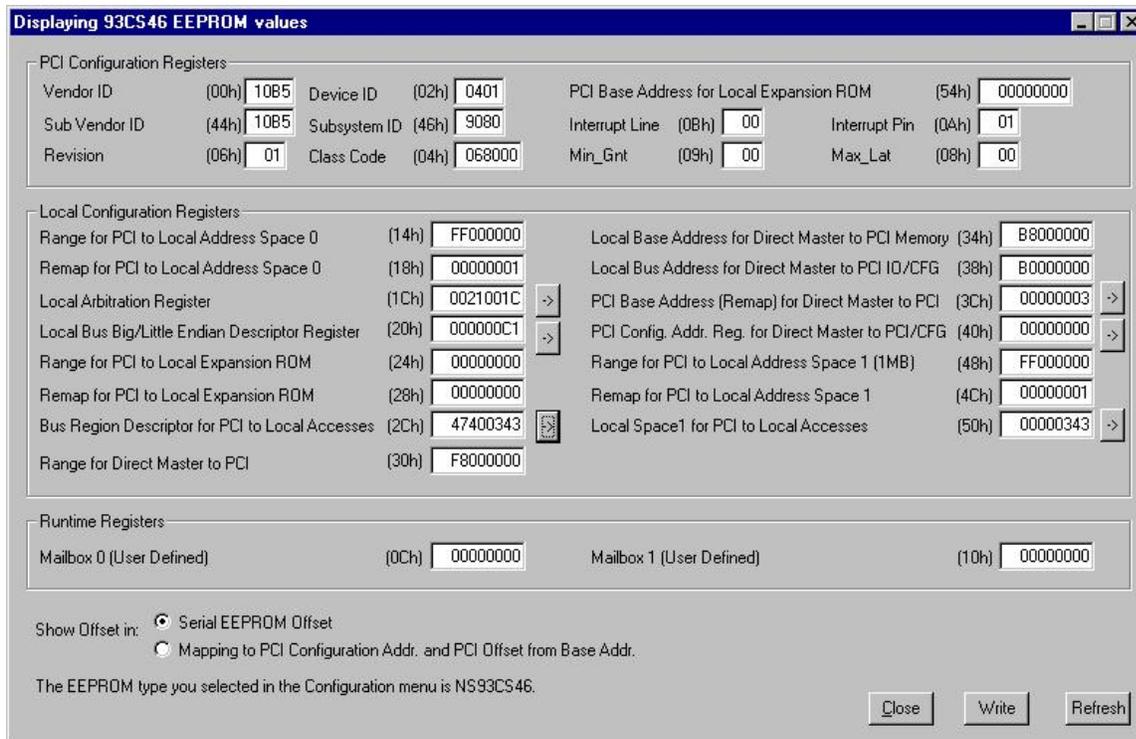


Figure 5-4. Configuration EEPROM Settings for the PCI 9080RDK-401B

5.5 PCI 9080RDK-860

The table below describes the memory requirements of the PCI SDK for the PCI 9080RDK-860.

Table 5-5. Basic Information About PCI 9080RDK-860

Item	PCI 9080RDK-860
DRAM Address	0x1000 0000 -> 0x10FF FFFF (16MB)
SRAM Address	0x0000 0000 -> 0x0007 FFFF (512KB)
FLASH Type	AM29F040 (IOP programming only)
FLASH Address	0xFFFF8 0000 -> 0xFFFF FFFF (512KB)
FLASH Image Offset at which to reprogram the FLASH	0x0000 0000
Memory space used by ROM code	0x0000 0000 -> 0x0003 FFFF
Memory space used by RAM code (PCI SDK Samples)	0x0004 0000 -> 0x0007 FFFF
Direct Master to PCI memory space Address	0x4000 0000 -> 0x40FF FFFF (16MB)
Direct Master to PCI IO space Address	0x5000 0000 -> 0x50FF FFFF (16MB)
PCI 9080 Register Address	0xC000 0000 -> 0xC000 01FF
Memory space recommend for user's use.	0x1000 0000 -> 0x10FF FFFF (16MB)

Users, who have upgraded the PCI SDK and intend to use it with an existing PLX RDK, must first reprogram the configuration EEPROM connected to the PLX chip. Skipping this step can cause unpredictable behavior of the PCI SDK. If you have purchased the PCI SDK with a PLX RDK then there is no need to upgrade the EEPROM because it was programmed at the factory before shipping. The following diagram shows the default settings for the RDK.

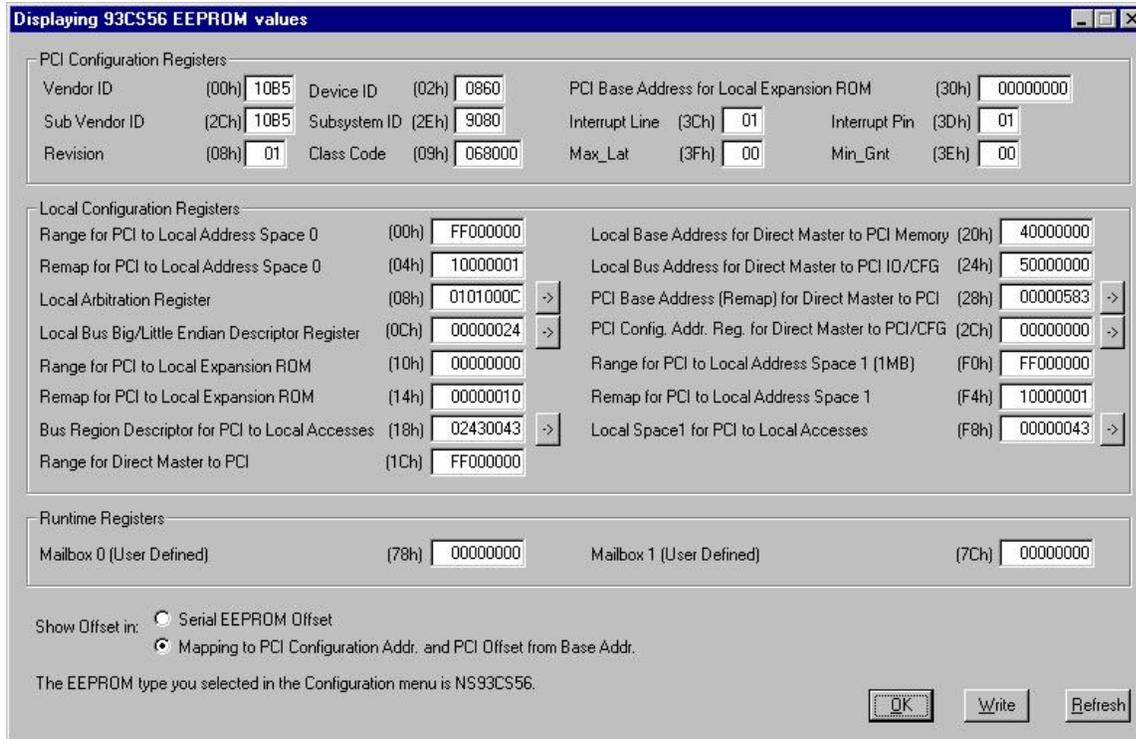


Figure 5-5. Configuration EEPROM Settings for the PCI 9080RDK-860

5.6 PCI 9080RDK-SH3

The table below describes the memory requirements of the PCI SDK for the PCI 9080RDK-SH3.

Table 5-6. Basic Information About PCI 9080RDK-SH3

Item	PCI 9080RDK-SH3
DRAM Address for SH3 access	0x0C00 0000 -> 0x0CFF FFFF (16MB)
DRAM Address for PCI 9080 access	Not Supported
SRAM Address for SH3 access	0x0900 0000 -> 0x0903 FFFF (256KB)
SRAM Address for PCI 9080 access	0x4000 0000 -> 0x4003 FFFF (256KB)
EPROM Type	TMS27C010A-10
EPROM Address for SH3 access (if boot device)	0x0000 0000 -> 0x0001 FFFF (256KB)
EPROM Address for PCI 9080 access	Not Supported
FLASH Type	MBM29LV160B-90
FLASH Address for SH3 access	0x0000 0000 -> 0x001F FFFF (2MB)

Item	PCI 9080RDK-SH3
(if boot device)	
FLASH Address for SH3 access (if non-boot device)	0x0100 0000 -> 0x011F FFFF (2MB)
FLASH Address for PCI 9080 access	Not Supported
FLASH Image Offset at which to reprogram the FLASH	0x0000 0000
Memory space used by ROM code	0x0C00 0000 -> 0x0C03 FFFF and 0x0900 0000 -> 0x0900 0FFF
Memory space used by RAM code (PCI SDK Samples)	0x0900 1000 -> 0x0900 FFFF
Direct Master to PCI memory space Address for SH3 access	0x0A00 0000 -> 0x0A0F FFFF (1MB)
Direct Master to PCI IO space Address for SH3 access	0x0B00 0000 -> 0x0B0F FFFF (1MB)
PCI 9080 Register Address for SH3 access	0x0800 0000 -> 0x0800 01FF
Memory space recommended for user's use for SH3 access.	0x0901 0000 -> 0x0903 FFFF (192KB)
Memory space recommended for user's use for PCI 9080 access.	0x4001 0000 -> 0x4003 FFFF (192KB)

Users, who have upgraded the PCI SDK and intend to use it with an existing PLX RDK, must first reprogram the configuration EEPROM connected to the PLX chip. Skipping this step can cause unpredictable behavior of the PCI SDK. If you have purchased the PCI SDK with a PLX RDK then there is no need to upgrade the EEPROM because it was programmed at the factory before shipping. The following diagram shows the default settings for the RDK.

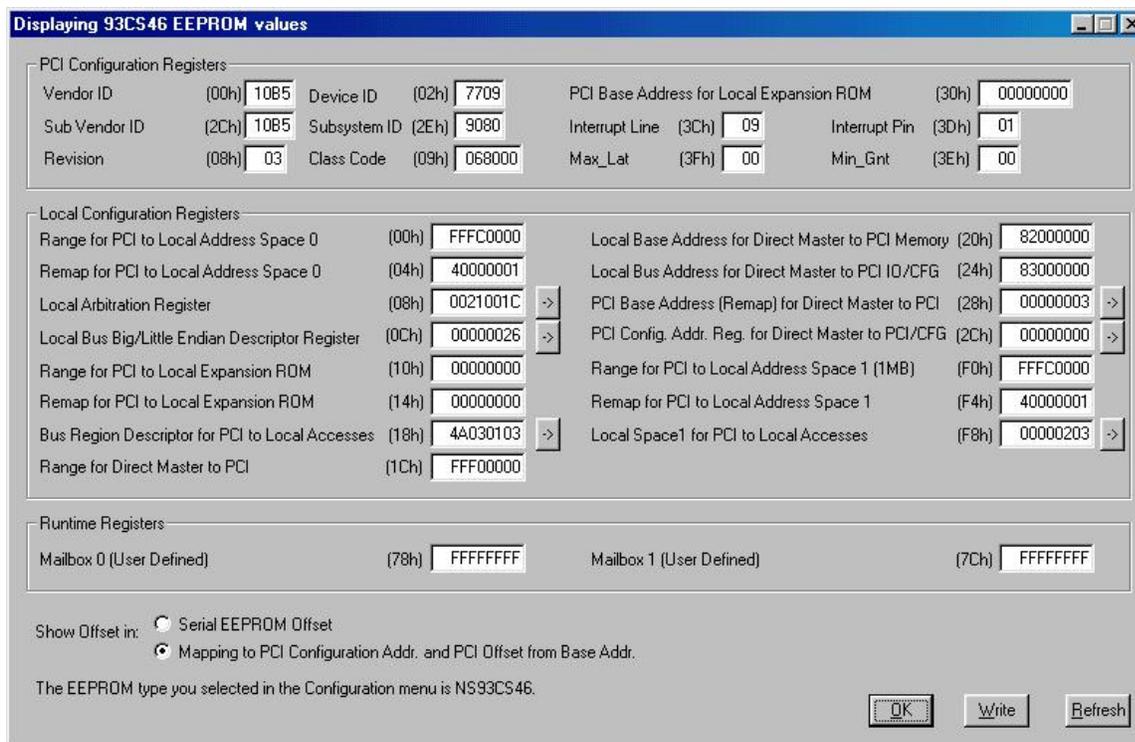


Figure 5-6. Configuration EEPROM Settings for the PCI 9080RDK-SH3

5.7 PCI 9080RDK-RC32364

The table below describes the memory requirements of the PCI SDK for the PCI 9080RDK-RC32364.

Table 5-7. Basic Information About PCI 9080RDK-RC32364

Item	PCI 9080RDK-RC32364
DRAM Virtual Address	0x0010 0000 -> 0x004F FFFF (4MB)
SRAM Virtual Address	0x0000 0000 -> 0x000F FFFF (1MB)
EPROM Type	M27C801-100
EPROM Virtual Address	0xBFC0 0000 -> 0xBFC7 FFFF (512KB)
EPROM Image Base Address	0x0000 0000
Virtual memory space used by ROM code	0x0000 0000 -> 0x0003 FFFF
Virtual memory space used by RAM code (PCI SDK Samples)	0x0004 0000 -> 0x0007 FFFF
Direct Master to PCI memory	0x1500 0000 -> 0x15FF FFFF

Item	PCI 9080RDK-RC32364
virtual space Address	(16MB)
Direct Master to PCI IO virtual space Address	0x1600 0000 -> 0x16FF FFFF (16MB)
PCI 9080 Register Virtual Address	0x1400 0000 -> 0x1400 01FF
Virtual Memory space recommend for user's use.	0x0008 0000 -> 0x004F FFFF (4.5MB)

Due to hardware design, the configuration EEPROM on this RDK does not work. So, the configuration EEPROM is not provided. However, the original hardware design for the serial EEPROM is 93CS56.



This page is intentionally left blank.

Appendix A. Index

A

AppMain..... 3-6, 3-7, 3-20, 3-21

B

BEM 3-2, 3-3, 3-12, 3-13, 3-14, 3-15, 3-16, 3-17, 3-19, 3-20, 3-21

BemMain 3-7, 3-20

Block DMA 3-8, 3-9, 3-10, 3-12

Boot up code..... 2-7, 3-5, 3-22

BSP 1-1, 2-4, 2-17, 3-1, 3-2, 3-3, 3-4, 3-5, 3-6, 3-7, 3-12, 3-21, 3-22, 3-23

C

CompactPCI..... 1-3, 2-5, 5-3, 5-4

D

Diab Data, Inc. Compiler..... 1-3, 2-7, 2-8

Direct Master 3-6, 5-1, 5-2, 5-3, 5-4, 5-6, 5-7, 5-9, 5-10, 5-11

Direct Slave 2-17

E

Endian..... 3-1

Exception vector table 3-5

G

GNU compiler 1-3, 2-7

I

I₂O..... 1-1

IBM 401 EVB..... 1-3

IBM High C/C++ PowerPC Cross-Compiler . 1-3

IDT/c Cross Compiler..... 1-3, 2-7, 2-8

IOP 480..... 1-2, 1-3, 2-4, 2-10, 2-13, 2-14, 2-16, 3-8, 3-13, 3-14, 3-16, 3-17, 3-19, 5-1

IOP 480RDK 1-3, 2-7, 5-1

ISR 3-5, 3-10, 3-11, 3-23

L

LED..... 2-7, 3-1, 3-22

M

Microprocessor 1-4, 3-4, 3-5

MiniBSP 2-6, 2-17, 3-5, 3-22

MPC860..... 1-3, 2-7, 2-17

P

PCI 9054RDK-860 1-3, 2-5, 5-2

PCI 9080RDK-401B..... 1-3, 2-5, 5-5

PCI 9080RDK-860 1-3, 2-5, 5-6, 5-7

PCI 9080RDK-RC32364 1-3, 2-5, 2-7, 5-10

PCI 9080RDK-SH3 1-3, 2-5, 2-7, 5-8

PCI API DLL..... 3-3, 3-25

PCI BIOS 2-5, 3-6

PLXMon 99 1-1, 1-2, 2-1, 2-4, 2-5, 2-6, 2-10, 2-11, 2-14, 2-17, 3-1, 3-3, 3-7, 3-12, 3-20, 3-21, 3-24, 3-25

Power Management 2-16

Protocols 3-14

R

Registry..... 2-5, 2-12, 2-16

S

SGL DMA 2-16, 3-8, 3-9

Shuttle DMA..... 2-14, 2-16, 3-8, 3-9, 3-11

U

UART 3-6, 3-7, 3-12, 3-21, 3-23

W

WDM 2-1, 2-10, 3-25



This page is intentionally left blank.