PLX TECHNOLOGY ®
THE I/O INTERCONNECT SOLUTION

PLX SDK

SOFTWARE DEVELOPMENT KIT

User's Manual

Version 3.2

March 2001

Website:	http://www.plxtech.com
Email:	apps@plxtech.com
Phone:	408 774-9060
	800 759-3735
Fax:	408 774-2169

PLX Technology, Inc. retains the right to make changes to this product at any time, without notice. Products may have minor variations to this publication. PLX assumes no liability whatsoever, including infringement of any patent or copyright, for sale and use of PLX products.

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.

Document number: PCI-SDK-MAN-P1-3.2

PLX SOFTWARE LICENSE AGREEMENT

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

LICENSE Copyright © 2001 PLX Technology, Inc.

This PLX Software License agreement is a legal agreement between you and PLX Technology, Inc. for the PLX Software, which is provided on the enclosed PLX CD-ROM. PLX Technology owns this PLX Software. The PLX Software 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 Software, PLX grants you a license to use the PLX Software 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 Software. 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 Software 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 Software except to the extent permitted by law.

PLX Software License Agreement

GENERAL

If you do not agree to the terms and conditions of this PLX Software License Agreement, do not install or use the PLX Software and promptly return the entire unused PLX Software to PLX Technology, Inc. You may terminate your PLX Software license at any time. PLX Technology may terminate your PLX Software 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 Software. Any attempt to sub-license, rent, lease, assign or to transfer the PLX Software except as expressly provided by this license, is hereby rendered null and void**.**

WARRANTY

PLX Technology, Inc. provides this PLX Software AS IS, WITHOUT ANY WARRANTY, EXPRESS OR IMPLIED, INCLUDING WITHOUT LIMITATION, AND ANY WARRANTY OF MERCHANTIBILITY 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 © 2001 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

Table of Contents

List of Figures

1 General Information

PLX Technology offers PCI bus interface chips that address a range of adapter and embedded system applications. PLX 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, PLX chips are easily configured to run without a local CPU. PLX provides Software Development Kits (SDK), Rapid Development Kits (RDK) and Hardware Design Kits (HDK) to facilitate your PCI design development.

1.1 About This Manual

This manual provides information about the functionality of the PLX SDK. Customers have the choice of using the PLX SDK with any PLX Rapid Development Kit (RDK), or any custom design containing a PLX chip. Users should consult this manual for PLX SDK installation and general information about the design architecture.

1.2 PLX SDK Features

The PLX SDK comes in two flavors, a Professional Edition (SDK-PRO) and a Standard Edition (SDK-LITE).

The SDK-LITE contains software for a Windows host environment where the PLX Chip is accessed across the PCI bus, typically through a device driver. This package includes the following features:

- A PCI Host API and device drivers compatible with Windows NT/98/2000;
- PLXMon, a Windows Graphical User Interface (GUI) application used to configure and modify PLX PCI devices and download applications to the PCI device.

The SDK-PRO is a superset of the SDK-LITE and includes a multitude of additional features. The SDK-PRO additionally provides software for the local-side or I/O Platform (IOP) where a PLX chip is connected to a local CPU. These features include:

- A feature based Local API, with support for a variety of CPUs and PLX PCI chips
- Local DMA Resource Manager that supports three modes of operation
- The PLX Back-End Monitor (BEM), which is the target server supporting serial mode communications with PlxMon
- Board Support Packages (BSP) for PLX RDK boards, which can be used as re-usable and customizable building blocks in customer designs, or for reference purposes
- VxWorks RTOS BSPs for PLX RDK boards, which can be used as a model for development of RTOS BSPs for custom boards
- Linux Host driver for the 9054, supporting a subset of the PLX Host API
- pSOS RTOS BSPs for PLX RDK boards, which can be used as a model for development of RTOS BSPs for custom boards

1.3 Where To Go From Here

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

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

Chapter 3, PLX SDK Software Architecture, describes the design and implementation of the PLX SDK software.

1.4 Terminology

- References to Windows NT assume Windows NT 4.0 and may be denoted as WinNT.
- References to Windows 98 may be denoted as Win98.
- References to Windows 2000 may be denoted as Win2k.
- References to Windows Me may be denoted as WinMe.
- References to Visual C/C++ or Visual C++ refer to Microsoft Visual C/C++ 6.0.
- 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) or Local-side throughout this manual denote a Custom board with a PLX chip or a PLX RDK board. All references to IOP or Local software denote the software running on the board. References to the IOP 480 denote the PLX IOP 480 chip, which includes “IOP” as part of its name. This is a special case for use of IOP.

1.5 Customer Support

Prior to contacting PLX customer support, please be prepared to provide the following information:

PLX chip used

PLX SDK version (if applicable)

Host Operating System and version

Model number of the PLX RDK (if any)

Description of your intended design, including:

- Local Microprocessor (if any)
- Local Operating System and version (if any)

Detailed description of your problem

Steps to recreate the problem.

If you have comments, corrections, or suggestions, you may contact PLX Customer Support at:

Address: PLX Technology, Inc.
Attn. Technical Support
870 Maude Avenue
Sunnyvale, CA 94085

Phone: 408-774-9060

Fax: 408-774-2169

Web: http://www.plxtech.com

Email: PLX-Helpdesk@plxtech.com

2 Getting Started

This section describes how to get up and running with the PLX SDK.

2.1 Unpacking the PLX SDK

The PLX SDK comes complete with the following items:

- PLX SDK User's Manual (this document)
- PLX SDK Programmer's Reference Manual
- PLXMon User's Manual
- PLX SDK CD-ROM.

Note*: The SDK-LITE does not provide hard copies of the SDK manuals. Adobe PDF versions of the manuals are included in the SDK installation. Acrobat Reader® is available free of charge from the Adobe website (www.adobe.com).*

2.2 Minimum System Requirements

To install and run the SDK in Windows, the minimum system requirements are as follows:

- Windows NT 4.0 with Service Pack 3 or above, Windows 98, or Windows 2000
- 16MB of RAM, 32MB or more recommended
- 80MB hard drive space
- free RS-232 serial port and cable, if serial communications will be used

2.3 Development Tools

Various tools were used to build the software included in the PLX SDK. There are many compatible alternative tools available for the various build environments. Customers are free to use their own preferred sets of compatible development tools; however, PLX has only verified the tools listed below and, as a result, cannot support tools not listed here. The development tools used to develop the PLX SDK components include:

Windows Applications and API DLL:
Microsoft Visual C/C++ 6.0, Service Pack 4

Windows NT 4.0 Drivers:
Microsoft Windows NT 4.0 Device Driver Kit (DDK) with Visual C/C++ 6.0

Windows Driver Model (WDM) Device Drivers for Win98 and Win2000:
Microsoft Windows 2000 Device Driver Kit (DDK) with Visual C/C++ 6.0

Local-side software:
DIAB-SDS PPC tools, version 4.3g
Metaware High C/C++ Power PC Toolset, version 4.3b

VxWorks BSPs:
Tornado 2.0 development tools with support for MPC860 and PPC403

Linux Device Driver:
Red Hat Linux 7.0 with Linux kernel 2.2.16

pSOS BSPs:
pRISM+ v2.0.0 for pSOSystem v2.5.0, which includes the DIAB compiler suite, v4.2b

2.4 Windows Software Installation

Note: *Before installing SDK 3.2, any previously installed PLX SDK versions should be removed. Installation of multiple SDK versions may result in erratic behavior due to file conflicts. Refer to section 2.7 for more details.*

To install the PLX SDK Software package, complete the following:

1. Insert the CD-ROM into the appropriate CD-ROM drive.
2. The installation will start automatically. If this does not load, run ***Setup.exe*** from the installation CD-ROM.
3. Follow the prompts to complete the SDK installation.
4. Reboot the computer after the installation.

Note: *For proper WinNT and Win2000 installation, a user with "Administrator" rights must install the SDK.*

2.4.1 PLX SDK and Windows Me

At this time, PLX has not verified the PLX SDK with Windows Me; therefore, it is not supported. Users may use the SDK with Windows Me at their own risk.

2.5 Installation in Unix and Other Non-Windows Environments

The PLX SDK installation package, which was created with InstallShield®, will only execute in a Microsoft Windows environment. In order to use the PLX SDK in Unix or other operating systems, PLX includes the complete set of SDK files on the installation CD-ROM, in the directory <*SdkFiles*>. This directory can be copied directly to the user's file system and used for development or reference purposes.

Although the Windows device drivers cannot be used in a non-Windows OS, the source code can be used as a reference for writing drivers. The local software included in the SDK-PRO can be used as-is with compatible development tools, which are available for various Operating Systems.

2.6 PLX SDK Version Compatibility

When using PLX SDK v3.2 it is important that all components are the same version, as follows:

- In Windows, the PLX device drivers (*.sys* files) and the PLX API DLL (*PlxApi.dll*) must both be the versions included in SDK 3.2.
- When building applications, it is important to use the C header files included in SDK 3.2. Application, built with older SDK versions should be re-built.
- If Serial Mode communications will be used between PLXMon and the device, the local software must be compatible with SDK 3.0 or above. The PLX serial protocol has not changed in SDK 3.2; however, due to improvements in the SDK 3.2 local software, it is recommended that this be used instead of SDK 3.0.
- PLX SDK versions prior to SDK 3.0 are incompatible. Many structures and components have changed. A small porting effort is required if SDK 3.2 will be used.

PLX RDKs contain local software versions to match the SDK version at the time of shipment; however, if the PLX SDK is purchased as an upgrade and intended for use with an existing PLX RDK board, the RDK's FLASH code should be upgraded to the current version. This is necessary to ensure that nothing unpredictable occurs due to incompatibilities with modules.

To upgrade the FLASH image, use PLXMon to reprogram the RDK FLASH. If PLXMon cannot be used for some reason, such as a corrupt FLASH on a 9054-860 board or Windows is not available, a device I/O programmer must be used. The updated FLASH images are provided in

<Sdk_Install_Dir>\SupportFiles\RdkFlashFiles. Please refer to the PLXMon manual for details on FLASH programming.

2.7 Uninstalling Previous Versions of the PLX SDK

Warning: *If any files have been modified in the original PLX SDK install directory, such as C source code files, the uninstaller may delete them. Please be careful before uninstalling an SDK package. The SDK directory can first be copied (not moved) to another safe location before removal.*

Prior to installation of a new version of the PLX SDK, any previously installed versions should be uninstalled. Many files change between SDK releases and since these files are used for development purposes, they may be incompatible with a previous release. To remove a PLX SDK package, including device drivers, complete the following:

1. Close any open applications
2. Open the Windows Control Panel
3. Select *Add/Remove Programs* icon in the Control Panel window
4. Choose the PLX SDK package from the item list
5. Click the *Add/Remove...* button

Note: *For proper removal in WinNT and Win2000, a user with “Administrator” rights must remove the PLX SDK.*

2.8 Windows Host Software

The following sections describe the components that comprise the Windows Host portion of the PLX SDK. These components are explained in greater detail later in this manual.

2.8.1 PLXMon Debug Tool

The PLXMon Debug Utility is included in the *<Sdk_Install_Dir>\Bin* directory. This is a powerful tool, which provides easy-to-use GUI screens for read/write of PLX chip registers, access to local bus devices, download of local software to RAM and ROM (FLASH programming), and EEPROM access. *Refer to the PLXMon User’s Manual for more information*.

2.8.2 PLX API Library

The PLX API library in the Windows environment file is implemented as a Dynamic Link Library (DLL). The file is called *PlxApi.dll* and can be found in *<Sdk_Install_Dir>\Win32\Api*. The DLL’s responsibility is to communicate with the selected device driver when a PLX API function is called. The API functions are actually implemented in the device drivers because they are different for each chip.

During the SDK installation, *PlxApi.dll* is copied to *<Win_Dir>\System32* in Windows NT/2000 and to *<Win_Dir>\System* in Windows 98.

2.8.3 Windows Device Drivers

A device driver is necessary for the PLX SDK software to access PLX PCI devices. Windows applications, including PLXMon, cannot access PCI devices without a device driver installed. The PLX SDK includes drivers for all supported PLX PCI chips.

2.8.3.1 Windows NT Device Drivers

The PLX device drivers included for Windows NT are considered NT kernel-mode drivers. Windows NT does not support Plug ‘n’ Play or Power Management. Drivers in NT run as services and are started when the Operating System starts, as opposed to PnP operating systems, which load drivers only if a device is physically present in the system. In fact, NT drivers are responsible for scanning the PCI buses to find the devices they are responsible for. The following sections describe installation of NT drivers and some configuration options.

2.8.3.1.1 Windows NT Device Driver Installation

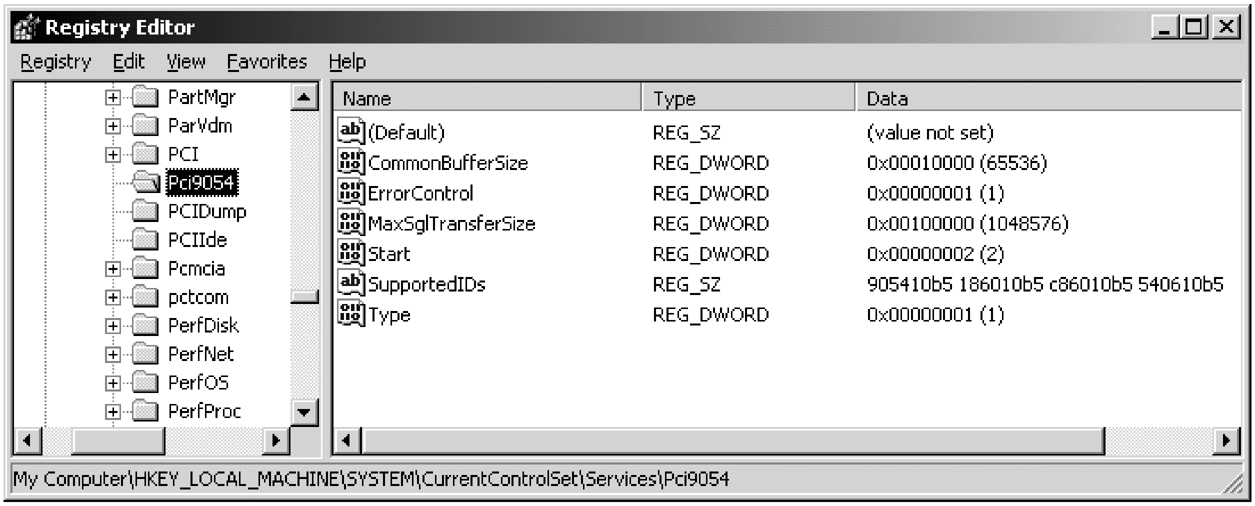
The PLX SDK installation package automatically installs the Windows NT drivers and adds the required registry entries.

Windows NT device drivers are provided for each supported PLX device. All device drivers are located in *<Sdk_Install_Dir>\Win32\Driver\WinNT\<DeviceName>*. The naming convention used for the device drivers is: *Pci<DeviceType>.sys*. or *Iop480.sys*. For example, the device driver for the PLX 9054 device is "*Pci9054.sys"*.

The *.sys* files are copied to *<Win_Dir>\System32\Drivers* during SDK installation.

2.8.3.1.2 Windows NT Drivers Registry Configuration

Every Windows NT device driver requires an entry in the registry. The registry entry contains some information required by the operating system, as well as allows for customized settings for the device driver. The registry entry is located under *HKLM\System\CurrentControlSet\Services* and must match the driver name without the *.sys* extension. The figure below demonstrates the entry for the 9054 device driver.



Name	Type	Data
(Default)	REG_SZ	(value not set)
CommonBufferSize	REG_DWORD	0x00010000 (65536)
ErrorControl	REG_DWORD	0x00000001 (1)
MaxSglTransferSize	REG_DWORD	0x00100000 (1048576)
Start	REG_DWORD	0x00000002 (2)
SupportedIDs	REG_SZ	905410b5 186010b5 c86010b5 540610b5
Type	REG_DWORD	0x00000001 (1)

My Computer\HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\Pci9054

Figure 2-1 Windows NT Registry Information for the 9054 Device Driver

Note: O*nly advanced users with administrative rights should modify entries in the registry.*

Windows NT required entries:

- *ErrorControl*
  Required by the operating system and should not be modified.
- *Start*
  Required by the operating system and should not be modified.
- *Type*
  Required by the operating system and should not be modified.

PLX-specific entries:

- CommonBufferSize
  This value sets the size of the Common buffer, which the driver attempts to allocate for use by all applications. This buffer is a non-paged contiguous buffer, so it can be used for DMA transfers. The default value is set to 64KB. Users may increase this value if a larger buffer size is needed.

Note: *Changing this entry does NOT guarantee a larger buffer will be allocated. The device driver makes a request to the operating system for a buffer with the size indicated by this registry entry. However, if the request fails, usually due to unavailable system resources, the driver will decrement the size and resubmit the request until the buffer allocation succeeds. The API call PlxPciCommonBufferGet() can be used to determine the common buffer information.*

- *MaxSglTransferSize*
  This value sets the size of an internal buffer that is required for SGL and Shuttle DMA transfers. This device driver uses this buffer as a temporary location to place SGL descriptors during a DMA transfer. A descriptor is required for each page of memory. Pages in Windows are typically 4k in size.

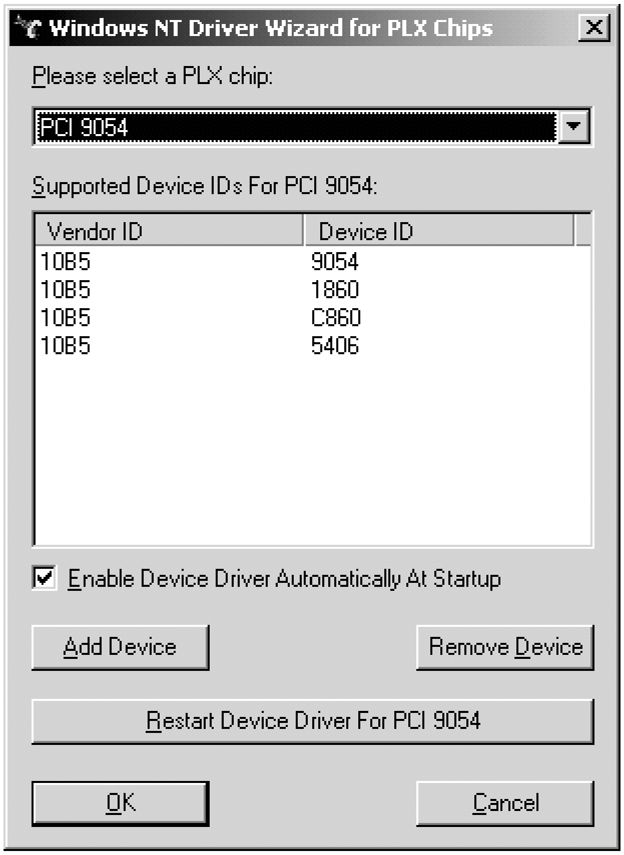
- *SupportedIDs*
  This value contains the Device/Vendor IDs for the PLX devices that the driver supports. When the PLX device driver scans the PCI bus at startup, it compares the ID of every device it finds with the IDs in this entry. The driver will only attach and claim a device when there is a match. It is recommended that included *DriverWizard* SDK application is used to modify this field. Invalid modification of this field directly may cause the *DriverWizard* to behave erratically or the device driver may fail to locate a device.

2.8.3.1.3 Windows NT Driver Installation for Custom Designs with PLX devices

For those with custom Device/Vendor IDs in Windows NT, the new ID must be made known to the PLX device driver or the driver will not recognize the device. The list of supported IDs, which the driver will use to compare with physically installed devices, is located in the device driver's registry entry *SupportedIDs*. Custom IDs must be added to this list in order for the driver to load properly.

PLX supplies the *DriverWizard* application for ease of managing the list of IDs. 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 allows users to control whether specific PLX device drivers start automatically.

Note: *Since all PLX device drivers are enabled by default, it is recommended that unused device drivers be disabled. For example, if a 9054 is the only PLX device that will ever be installed, use the DriverWizard, to disable all other drivers, such as PCI9080, PCI9030, and IOP480. The settings will take effect after reboot.*



Windows NT Driver Wizard for PLX Chips

Please select a PLX chip: PCI 9054

Supported Device IDs For PCI 9054:

Vendor ID	Device ID
10B5	9054
10B5	1860
10B5	C860
10B5	5406

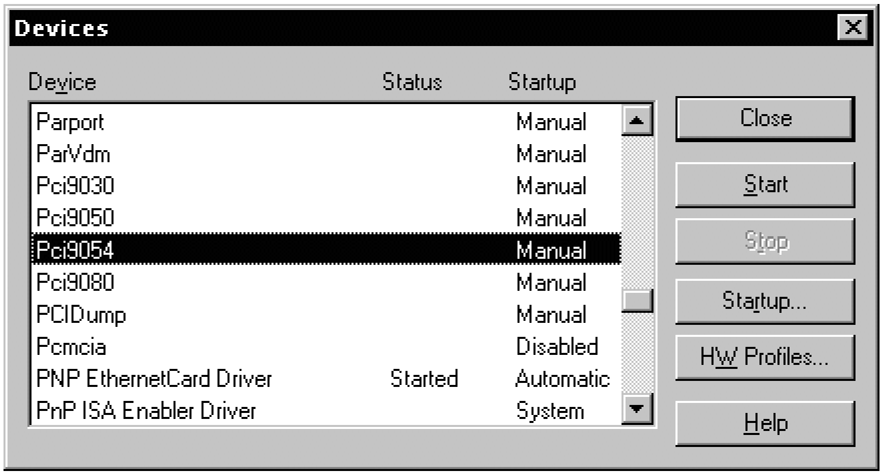
☑ Enable Device Driver Automatically At Startup

Figure 2-1 The PLX SDK Device Driver Wizard

2.8.3.1.4 Starting And Stopping Windows NT Drivers

In Windows NT, device drivers can be started or stopped dynamically, provided they are enabled. When driver registry settings have been changed, such as the *SupportedIDs* entry, the driver can be restarted for the settings to take effect.

To restart a device driver, open *Control Panel* and select the *Devices* option. Select the driver of choice, click *Stop*, and then click *Start*. Modified registry settings will take effect when the driver restarts.



Devices

Device	Status	Startup
Parport		Manual
ParVdm		Manual
Pci9030		Manual
Pci9050		Manual
Pci9054		Manual
Pci9080		Manual
PCIDump		Manual
Pcmcia		Disabled
PNP EthernetCard Driver	Started	Automatic
PnP ISA Enabler Driver		System

Figure 2-1 The Devices Utility Window

***Note**: Before stopping the device driver, all applications that have selected a PLX device, should be closed.*

By default, PLX device drivers are configured to startup automatically when Windows NT boots. Drivers can be configured to start manually or disabled by clicking the *Startup…* button and selecting the corresponding option.

The *DriverWizard* application can also be used to restart PLX device drivers. *Refer to section 2.8.3.1.3 for more details*.

2.8.3.2 WDM Device Drivers for Windows 98/2000

The Windows 98/2000 PLX device drivers conform to the Microsoft Windows Driver Model (WDM). These drivers support Plug 'n' Play (PnP) and Power Management. Windows 98 supports WDM version 1.0 and Windows 2000, at the time of this writing, supports WDM version 1.10. By definition, the WDM driver can be used as-is in either OS. Win98 does not support some WDM features and incorrectly implements some others. These are documented in the PLX WDM driver source code for those interested. The following sections describe WDM driver installation.

2.8.3.2.1 WDM Device Driver Installation

Since Windows 98/2000 are Plug 'n' Play (PnP) Operating Systems, the SDK installation package cannot automatically assign device drivers for PLX devices. The Windows PnP Manager is responsible for detecting devices and prompting the user for the correct driver. To assign a driver to a device, Windows refers to an INF file. The INF file provides instructions for Windows as to which driver files to install and which registry entries to insert.

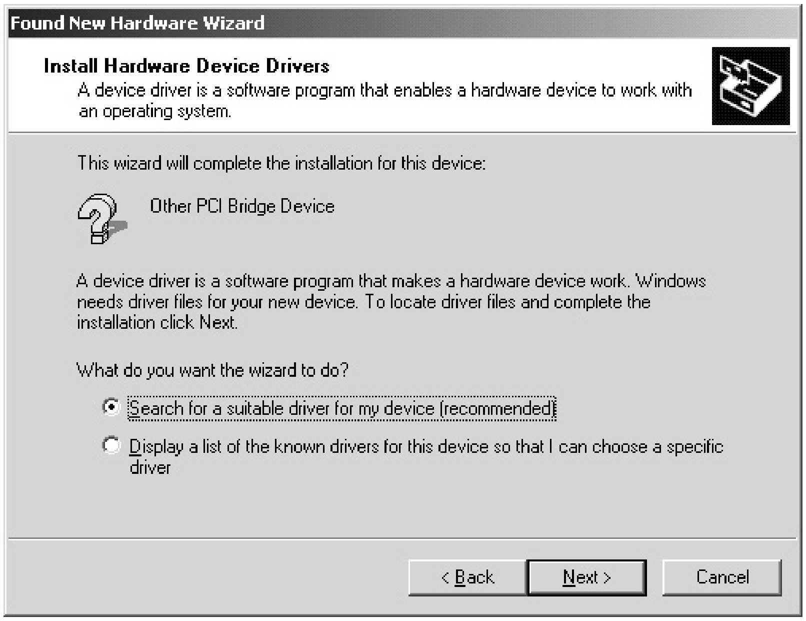
The PLX WDM device drivers, as well as the *PciSdk.inf* file, are located in *<Sdk_Install_Dir>\Win32\Driver\Wdm*. The naming convention used for the device drivers is: *Pci<DeviceType>.sys* or *Iop480.sys*. For example, the device driver for the 9054 device is named *Pci9054.sys*.

To install a driver for a board containing a PLX device in Windows 98/2000, complete the following steps:

1. After installing the PLX SDK successfully, shut down the computer.
2. Insert the PLX RDK board or your custom board with a PLX device into a free PCI slot.
3. Reboot the computer. Windows should first detect the new hardware device with a "*New Hardware Found*" message box. Acknowledge this message box.
4. Windows then displays the "Add New Hardware" Wizard, which will search for new drivers. If a PLX RDK board is installed proceed to section 2.8.3.2.1.1. If a custom board with a PLX device is installed, proceed to the section 2.8.3.2.1.2.

2.8.3.2.1.1 WDM Driver Installation for PLX RDKs

- Once Windows has completed its search, the following dialog is displayed: Select “Search for the best driver for your device”. Click ***Next***



Found New Hardware Wizard

Install Hardware Device Drivers

A device driver is a software program that enables a hardware device to work with an operating system.

This wizard will complete the installation for this device:

Other PCI Bridge Device

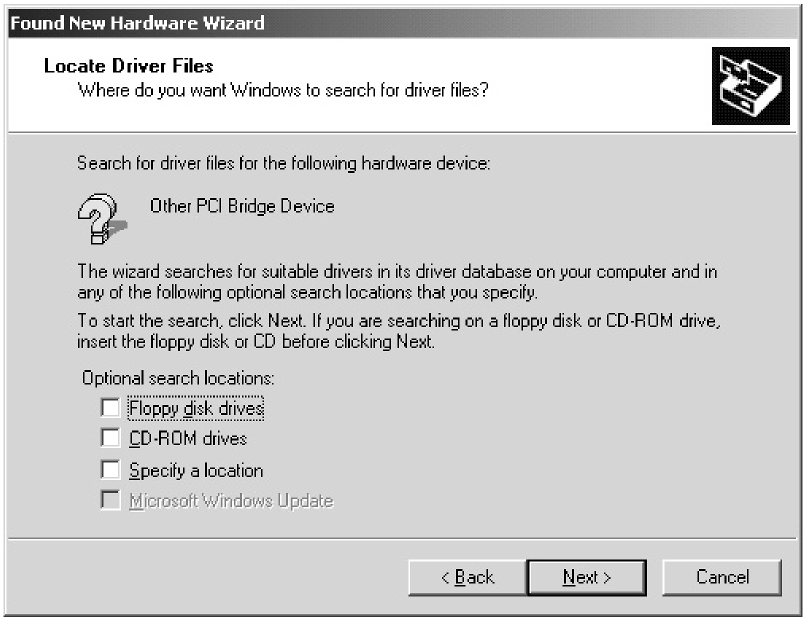
A device driver is a software program that makes a hardware device work. Windows needs driver files for your new device. To locate driver files and complete the installation click Next.

What do you want the wizard to do?

● Search for a suitable driver for my device (recommended)

○ Display a list of the known drivers for this device so that I can choose a specific driver

- Select the locations to search for INF the file. By default, PLX includes the INF file in *<Sdk_Install_Dir>\Win32\Driver\Wdm*. Click ***Next***.



Found New Hardware Wizard

Locate Driver Files

Where do you want Windows to search for driver files?

Search for driver files for the following hardware device:

Other PCI Bridge Device

The wizard searches for suitable drivers in its driver database on your computer and in any of the following optional search locations that you specify.

To start the search, click Next. If you are searching on a floppy disk or CD-ROM drive, insert the floppy disk or CD before clicking Next.

Optional search locations:

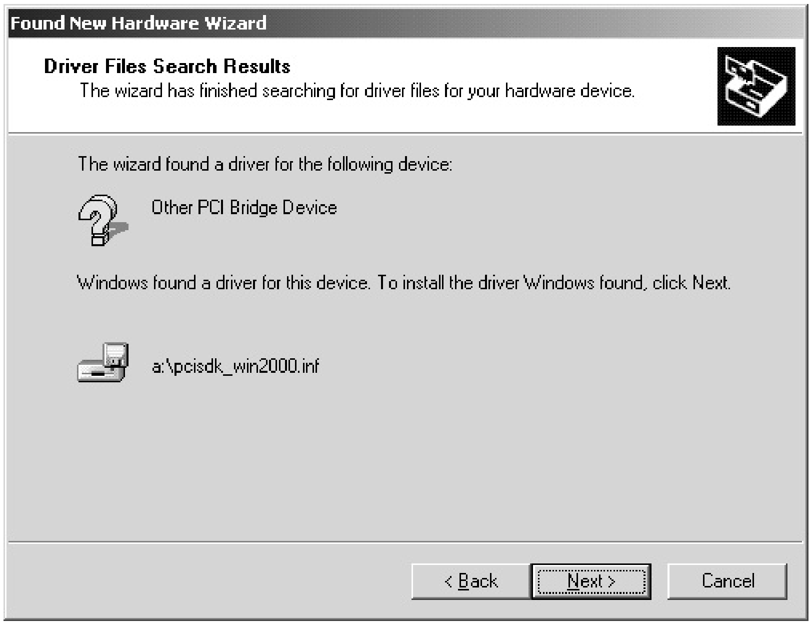
□ Floppy disk drives

□ CD-ROM drives

□ Specify a location

□ Microsoft Windows Update

- Windows will then scan through INF files to find a matching device driver. If one is found, the following dialog is displayed. Click ***Next***.



Found New Hardware Wizard

Driver Files Search Results
The wizard has finished searching for driver files for your hardware device.

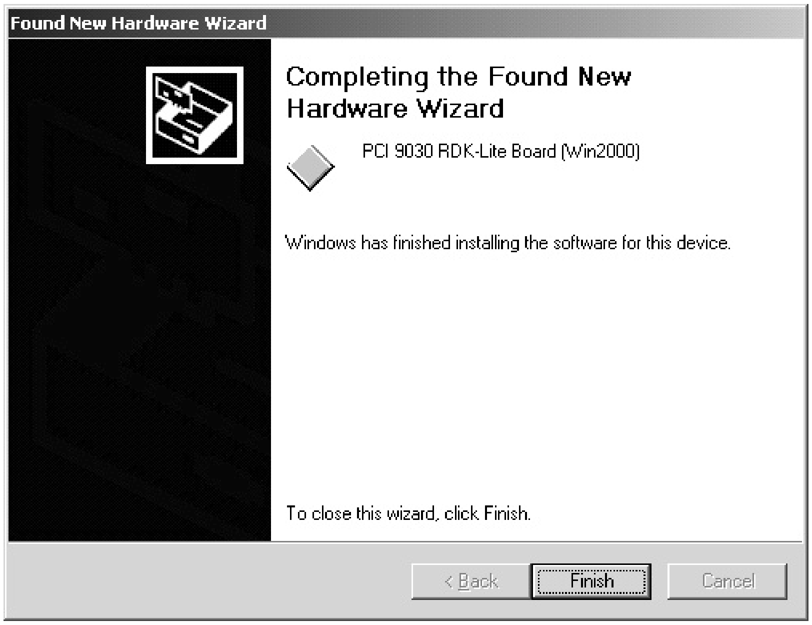
The wizard found a driver for the following device:

Other PCI Bridge Device

Windows found a driver for this device. To install the driver Windows found, click Next.

a:\pcisdk_win2000.inf

- When the following dialog is displayed, the device driver installation is complete. Click ***Finish***.



Found New Hardware Wizard

Completing the Found New Hardware Wizard

PCI 9030 RDK-Lite Board (Win2000)

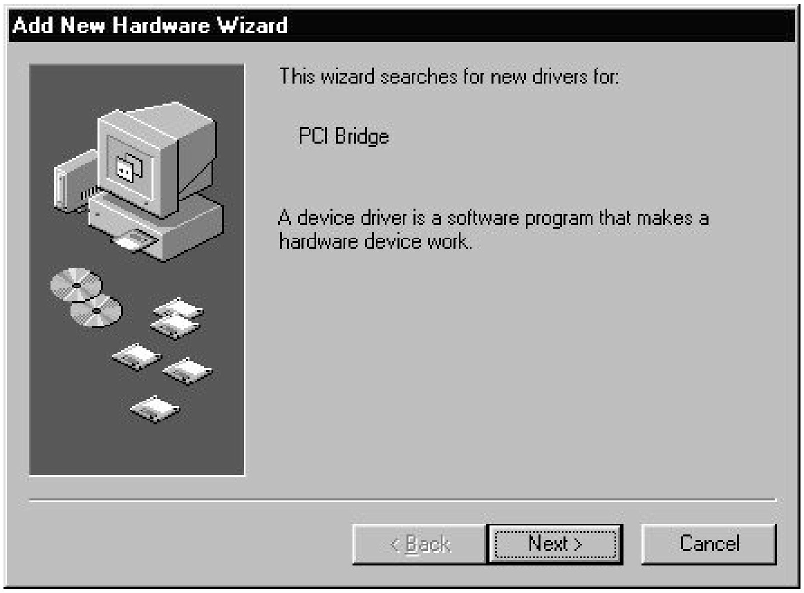
Windows has finished installing the software for this device.

To close this wizard, click Finish.

2.8.3.2.1.2 WDM Driver Installation for Custom Designs with PLX devices

To install a driver for a custom board containing a PLX PCI device, perform the following steps:

- Unless the PCI class code is changed, the installation wizard will detect the custom device as a ***PCI Bridge***. Click ***Next***



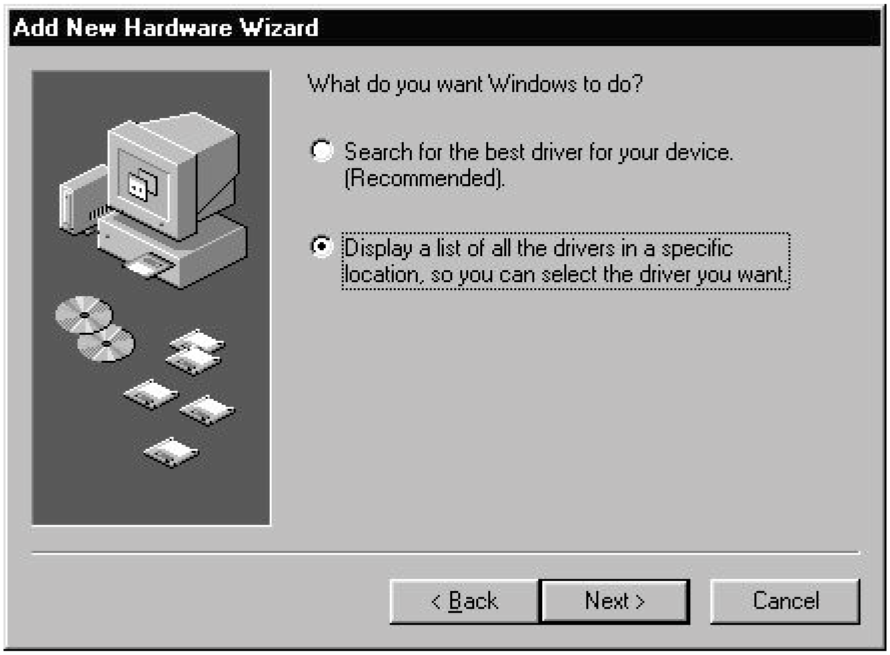
Add New Hardware Wizard

This wizard searches for new drivers for:

PCI Bridge

A device driver is a software program that makes a hardware device work.

- Select ***Display a list of all…*** since this is a custom device and click ***Next***.



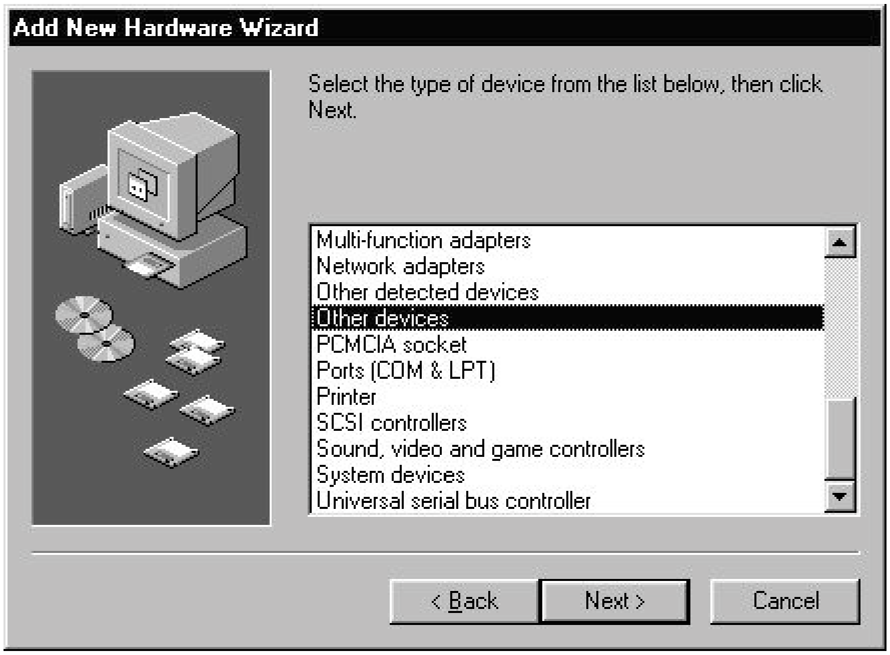
Add New Hardware Wizard

What do you want Windows to do?

○ Search for the best driver for your device. (Recommended).

● Display a list of all the drivers in a specific location, so you can select the driver you want.

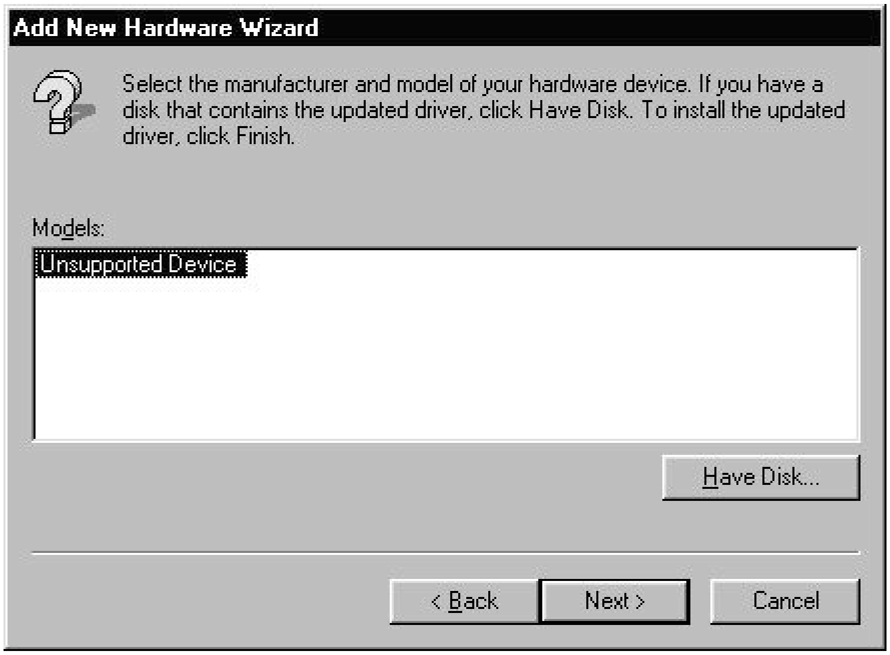
- Select ***Other devices*** and click ***Next***.



Add New Hardware Wizard

Select the type of device from the list below, then click Next.

- Select ***Have Disk…*** to search for a driver in another location and click ***Next***.



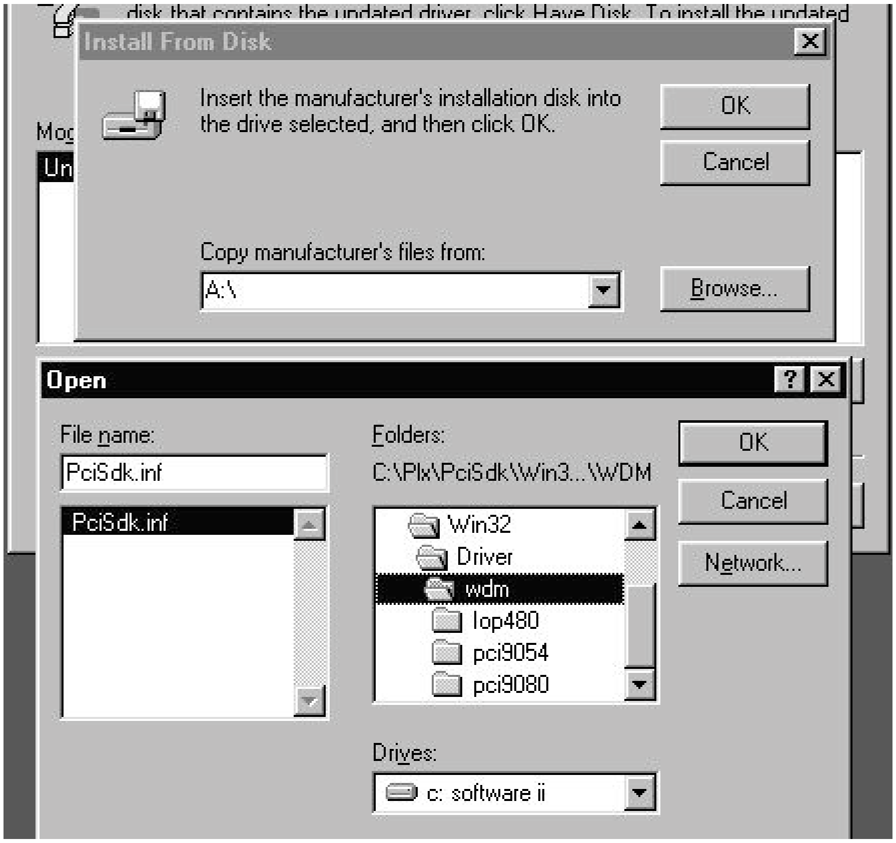
Add New Hardware Wizard

Select the manufacturer and model of your hardware device. If you have a disk that contains the updated driver, click Have Disk. To install the updated driver, click Finish.

Models:

Unsupported Device

- Browse to select the **PciSdk.inf** file, which is located in *Win32\Driver\Wdm* of the SDK.



Insert the manufacturer's installation disk into the drive selected, and then click OK.

Copy manufacturer's files from: A:\

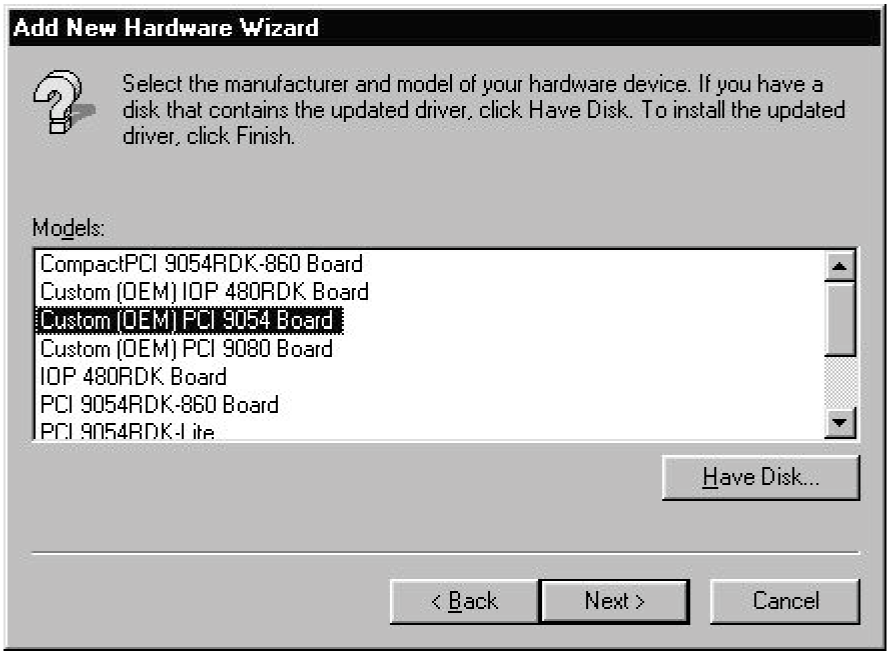
File name: PciSdk.inf

PciSdk.inf

Folders: C:\Plx\PciSdk\Win3...\WDM

Drives: c: software ii

- Once the INF file is located, Windows will parse it to provide a list of possible drivers. Select the driver that most closely matches the installed board type. Click ***Next***.

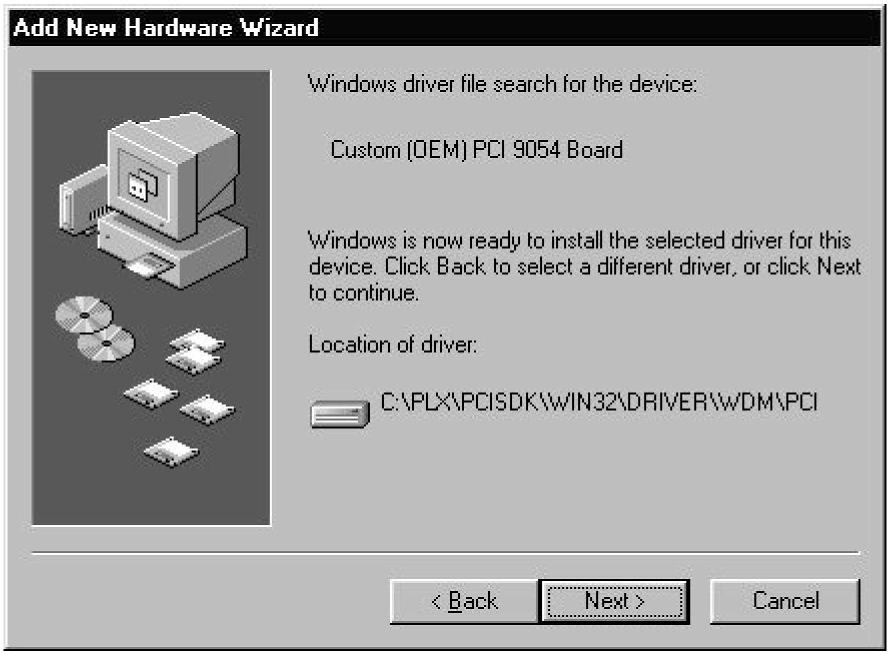


Select the manufacturer and model of your hardware device. If you have a disk that contains the updated driver, click Have Disk. To install the updated driver, click Finish.

Models:

CompactPCI 9054RDK-860 Board
Custom (OEM) IOP 480RDK Board
Custom (OEM) PCI 9054 Board
Custom (OEM) PCI 9080 Board
IOP 480RDK Board
PCI 9054RDK-860 Board
PCI 9054RDK-Lite

- Windows will now inform the user of the driver selected. If a warning appears which states, “The driver…was not written specifically for this device…”, install the driver anyway. This warning arises because the ID in the INF file does not match the installed device’s ID. Click ***Next***.



Add New Hardware Wizard

Windows driver file search for the device:

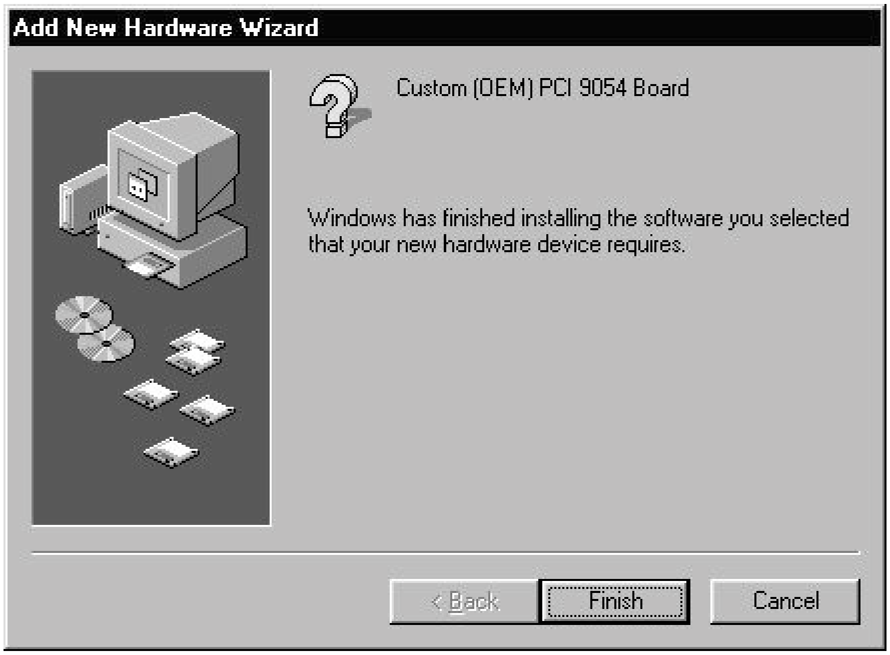
Custom (OEM) PCI 9054 Board

Windows is now ready to install the selected driver for this device. Click Back to select a different driver, or click Next to continue.

Location of driver:

C:\PLX\PCISDK\WIN32\DRIVER\WDM\PCI

- Click ***Finish*** to complete the installation.

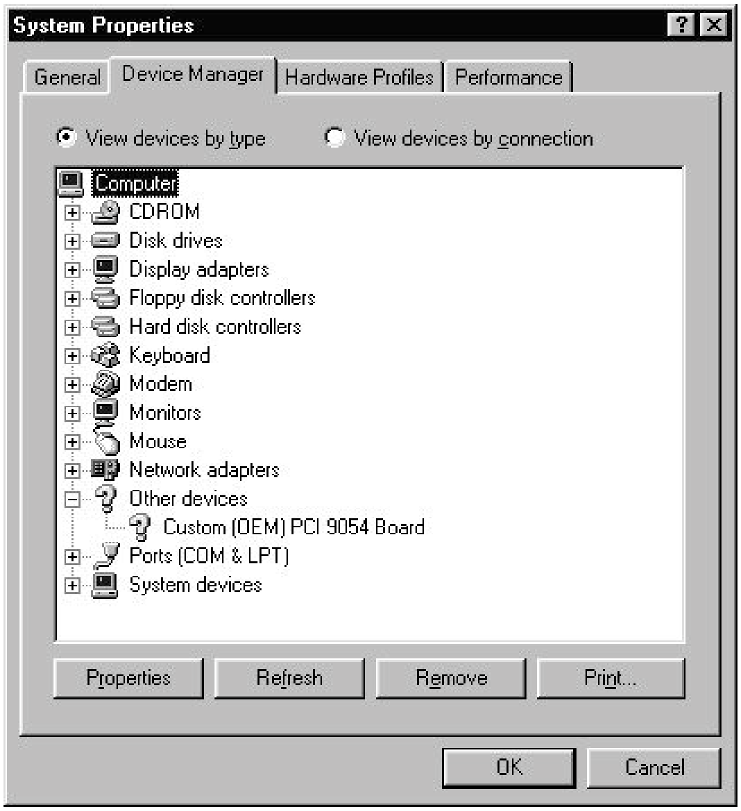


Add New Hardware Wizard

Custom (OEM) PCI 9054 Board

Windows has finished installing the software you selected that your new hardware device requires.

- If the device appears under ***Other devices***, the installation was successful. The PCI API and PLXMon may now be used to access the device.



Note*: If the Device/Vendor ID of the board is changed or the board is physically moved to a different PCI slot, Windows will recognize it as a completely new device and the process must be repeated.*

2.8.3.2.2 Registry Configuration

Every Windows 98/2000 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. Windows will add the registry entries automatically when it parses the INF file.

In Windows 98, device driver entries are located in the registry under *HKLM\System\CurrentControlSet\Services\Class\Unknown\000x*. *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 (e.g. *Pci9080.sys* or *Pci9054.sys*, depending upon the PLX chip in use).

In Windows 2000, the registry information is identical to Windows NT. *Refer to section 2.8.3.1.2 for information on changing driver options.*

2.8.3.2.3 Starting And Stopping WDM Drivers

Unlike Windows NT drivers, WDM device drivers are started and stopped as needed by the operating system. Device drivers are started when Windows detects a device that needs it. If, at a later time, the device is removed or uninstalled (from the Device Manager), the device driver will stop the device and release its resources. If no other devices are using the driver, it will be completely unloaded from memory.

2.8.4 Troubleshooting in Windows NT/2000

You may experience difficulties using the PLX SDK with WinNT or Win2000 if the amount of system memory is low, multiple PLX PCI devices are installed, or a device requests large PCI space sizes (8MB or more). When PLX device drivers load, they may be unable to map certain PCI spaces of the device if system resources are low. This is due to the limited amount of Windows Virtual-to-Physical Page Table Entries (PTE). The number of entries can be increased in the Registry by following the steps below:

1 Select ***Run*** from the *Start* button menu and execute *Regedit.exe* to start the Registry Editor.

2 Select the following path:
HKLM\System\CurrentControlSet\Control\Session Manager\Memory Management

3 The entry *SystemPages* controls the amount of system pages Windows can accommodate. By default, this number is 0, which means Windows automatically determines the number of pages. This number can be manually increased to force a greater amount of system pages at the expense of system memory. The PLX SDK installation increases this number to 80,000 or 0x13880. Users can try to increase the number further if problems persist with the virtual mappings.

2.9 Local Software

The Local software included in the PLX SDK contains a complete solution starting from the first instruction executed by the CPU all the way to the user application. *Refer to section 5 for detailed information on the local software components.*

2.10 Quick Start Guide to Using the PLX SDK with a Custom Board

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

1. Program the desired Vendor and Device IDs into the configuration EEPROM.
2. If using Windows NT, add the new Vendor and Device IDs to the Supported Device List with the *DriverWizard* utility.
3. If using Windows 98 or Windows 2000, refer to section 2.8.3.2.1 for driver installation.
4. Once the driver has loaded, PLXMon will be able to recognize and access the PCI device. One the board's properties are setup in PLXMon, the board's configuration EEPROM can be accessed and customized. New settings will take effect when the system is rebooted.
5. Try accessing local memory by using the Direct Slave memory accesses to the board (PLXMon '*dl*' command), which means the memory controller for local memory has been set up and the PLX device has been initialized correctly.
6. For boards with a local CPU, the PLX BSP can be ported to run on the local side. Alternatively, RTOS BSPs can be ported as well. If Serial Mode communications will be used with PLXMon, the board must support the BEM communication protocol found in PLX BSPs.

2.11 Distribution of PLX Software

This section is meant for users who have written applications with PLX software and intend to ship it with their product. PLX allows the necessary components to be shipped with the following restrictions:

- The complete SDK package may not be distributed without consent from PLX
- All source code written by PLX may not be distributed
- Applications, utilities, or tools, such as PLXMon, provided by PLX may not be distributed
- Documentation written by PLX to explain PLX software architecture or design or PLX reference documentation may not be distributed.

The following sections explain the components necessary for distribution in Host and Local environments.

2.11.1 Windows PCI Host Software Distribution

In a Windows environment, the components necessary for applications to use the PLX API and access PLX devices are the PLX API DLL and a PLX device driver. It is legal to distribute these components in binary format only. The installation of these components depends upon the version of Windows. *Refer to the Windows DDK for additional information on INF files and other installation requirements*.

The following sections describe the necessary files to copy and the registry entries to add. Typically, an installation application, such as InstallShield®, is used to create a package for distribution. In many cases, simple batch command files can be used.

2.11.1.1 Installation of the PCI Host Required Files

- The *PlxApi.dll* file should be copied as follows:

  Windows 98:
  Copy to the *<Windows_Dir>\System* directory

  Windows NT/2000:
  Copy to the *<Windows_Dir>\System32* directory

- The *PciSdk.inf* file is provided in *<Sdk_Install_Dir>\Win32\Driver\Wdm.* The INF file must be modified if the Device/Vendor ID does not match an entry in the provided INF. It should be copied as follows:

  Windows 98:
  Copy to the *<Windows_Dir>\Inf\Other* directory

  Windows 2000:
  Copy to the *<Windows_Dir>\Inf* directory

  Windows NT:
  This INF file is provided only for WDM device driver installation. It is not used for NT.

  ***Note**: Refer to the Windows DDK for details of INF files.*

- The PLX device driver file installation is dependent upon the Operating System version as follows:

  Windows 98/2000:
  These operating systems refer to the *PciSdk.inf* file for installation. The driver file(s) can be copied to the *<Windows_Dir>\System32\Drivers* directory to avoid prompts asking the user for the *.sys* file location.

  Windows NT:
  Installation of drivers in NT requires multiple steps. These are:

  1. Copy the corresponding driver file (e.g. *Pci9054.sys*) to the *<Windows_Dir>\System32\Drivers* directory.
  2. Multiple registry entries, which are documented in section 2.8.3.1.2, must then be added. The installation package can be configured to add these or, alternatively, PLX provides a set of *.reg* files, which allow automatic import of registry entries. All *.reg* files are located in *<Sdk_Install_Dir>\SupportFiles\WinNTRegistryEntries*. Each PLX driver has a corresponding file named to match the device driver, i.e. *Pci9054.reg*. *Refer to the Registry Editor Help for more information on importing .reg files.*

2.11.2 Local Software Distribution

Local software is typically distributed in either a downloadable image format or already programmed in ROM memory, such as FLASH. Applications built with PLX Local-side software components can be shipped in either of these formats and is royalty-free.

3 PLX SDK Software Architecture

This section of will describe the components that make up the PLX SDK and explain how they fit together.

The PLX SDK is separated into two distinct sets of software, the local software that runs on a board and the PCI Host software that executes in the Windows environment. Figure 3-1 demonstrates the various components and how they fit together.

In both environments, there is an application layer and a PLX API. The SDK is provided to handle most of the low-level functionality so users can concentrate on building their applications.

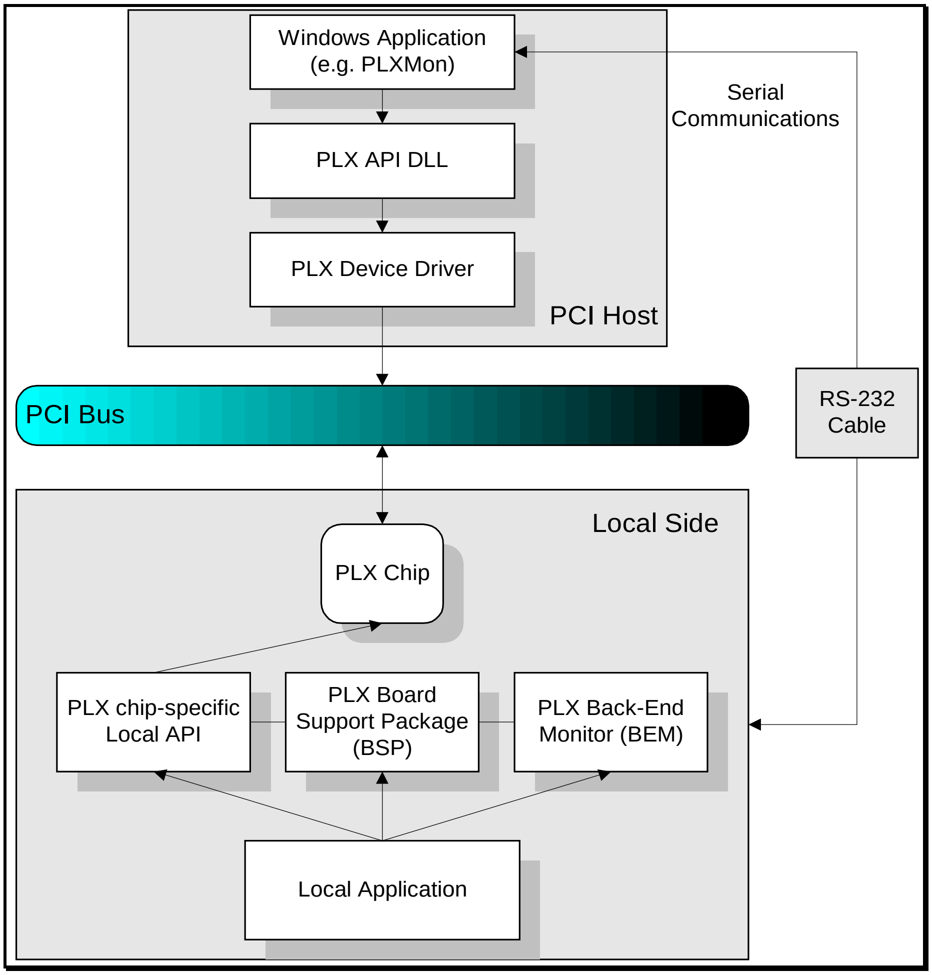


Figure 3-1 The PLX SDK Software Architecture

3.1 PLX SDK Software Assumptions

There are some assumptions made by the PLX SDK which users should be aware of. Note that these assumptions apply to the PLX software as it is shipped. Users are free to rebuild the software and customize it to their needs or change the implementation of certain features. The assumptions are as follows:

- The PCI Host software and the Local Software operate independently of each other. In other words, the PLX Windows device driver does not require a local CPU executing local code. If a local CPU exists, its code is self-contained on the board, usually in a FLASH chip, and it does not expect code to first be downloaded from the PCI Host.
- The PLX chip is initialized only by the EEPROM and/or the local software. The PCI Host does not perform PLX chip initialization.
- When the PLX Windows device driver is loaded, it may allocate a reserved, contiguous, non-paged buffer for shared use by applications and local software. The driver “informs” the local software of this buffer by writing the PCI base address of the buffer to Mailbox register 3 and the size of the buffer in Mailbox register 4. The local CPU is free to ignore this information and use Mailboxes 3 and 4, if it needs them. If the local CPU needs the Mailboxes and the buffer information, it can first store the information internally, and then use the Mailboxes as needed.

  ***Note**: The PLX driver does NOT control access to this buffer. It is up to users to implement shared memory protection schemes or synchronization if needed.*

- Some doorbell values are reserved for use by PLXMon and the PLX BSP. These trigger local interrupts and the value controls the action taken by the local CPU. The interrupts are defined below and the name is descriptive of the function. *Refer to the proper SDK header file for values*.
  - QUERY_EEPROM_TYPE
  - DOORRBELL_KERNEL_RESET
  - FLASH_READ
  - FLASH_WRITE
- For the Back-End Monitor to function properly, the board must have one available serial port, which is configured by the PLX BSP. The BEM does not need a dedicated serial port. The protocol is designed to share the port with raw serial data generated by the application.
- When PLXMon downloads applications to RAM, it uses Mailbox registers 5, 6 and 7 to communicate with the local software. These registers are only used during the download process, so they are available to application any other time.
- Some features in PLXMon rely on the local software to implement the corresponding protocol to perform the action. These features will not work in PLXMon, otherwise. One example is the download of applications to local RAM. *Refer to the proper SDK header file for documentation regarding the various protocols*.

4 PCI Host Software

This section describes the Windows Host software provided in the PLX SDK. The Windows PCI Host consists of the following components:

- PLX Host API DLL
- PLX Windows Device Driver
- User Applications

Applications use the PLX API by first selecting and opening a PLX device. Subsequent API calls can then be used to perform functions on the device. Many sample applications are provided in the SDK for reference purposes.

4.1 PLX API Library

The PLX API DLL is provided to communicate with the PLX device drivers. When an API function is called by an application, the API DLL handles the call and translates it to an I/O control message and sends it to the driver. Once the driver completes the request, control returns to the API DLL and then to the calling application.

The PCI API consists of a library of functions, from which multiple PLX chip-based PCI 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 PLX PCI board.

4.2 PLX Windows Device Drivers

The PLX device driver contains the API implementation for the PLX chip it supports and the basic functionality required by all device drivers in a Windows environment. The device driver accesses the PLX chip across the PCI bus by using Windows system calls. The driver is also responsible for handling PCI interrupts from the PLX chip.

The Windows Driver Model (WDM) is a specification for developing device drivers for Windows 98, Windows 2000, and future Windows operating systems. It is based on the device driver architecture found in Windows NT 4.0 and allows for driver binary compatibility between the aforementioned operating systems.

Each PLX chip type has an associated driver as depicted in Figure 4-1. Device drivers are not associated with a specific board, but are generic in design to be used for any board containing the specified PLX chip. A single driver is responsible for all devices in the system containing the PLX chip the driver was written for. Each device driver communicates with the PCI API on a one-to-one basis; there is no driver-to-driver communication.

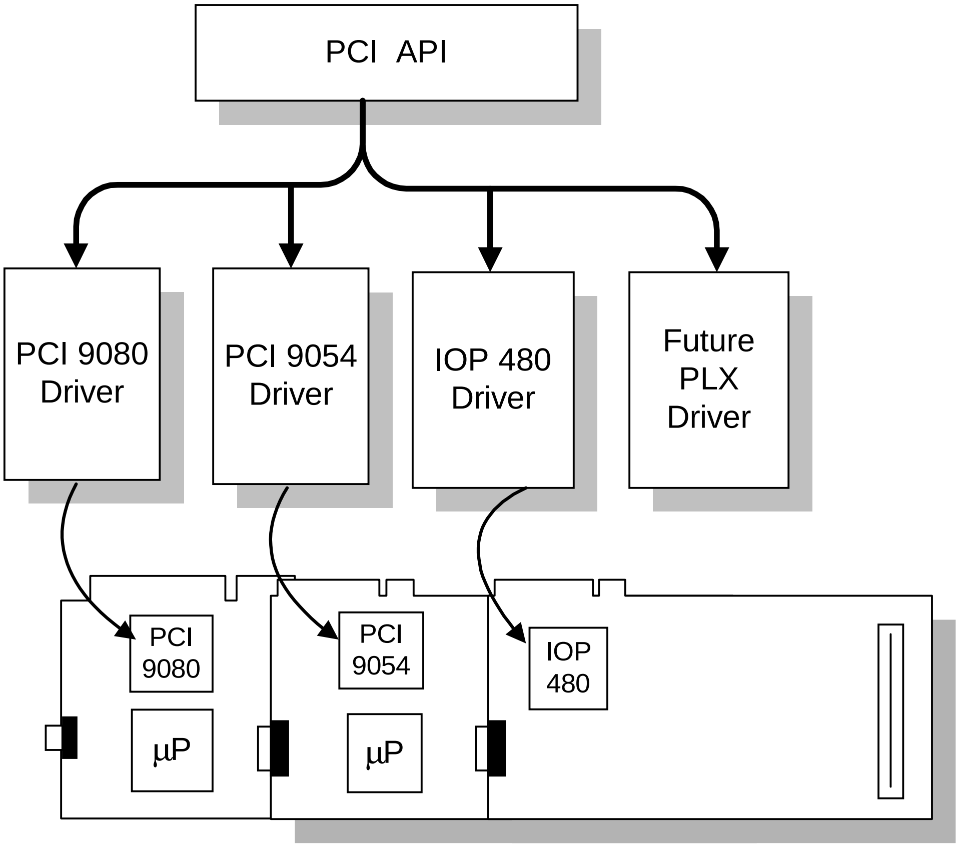


Figure 4-1 The PLX Driver Layout

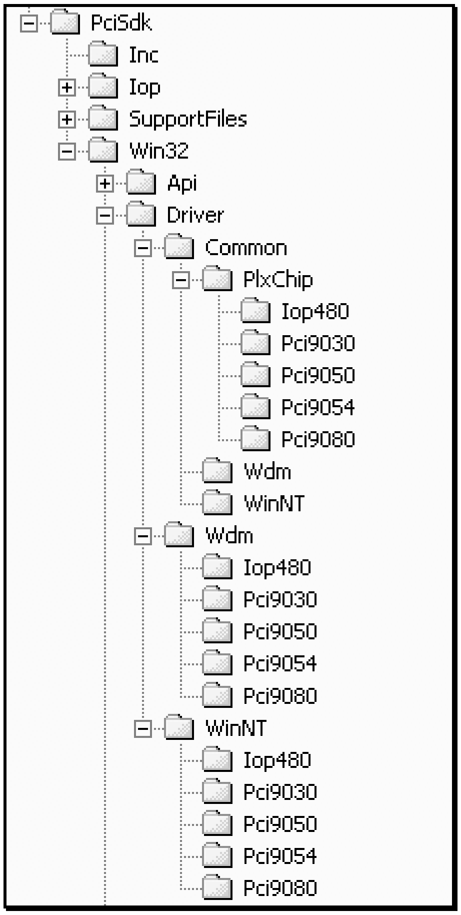
PLX assumes this generic approach to device drivers in order to accommodate all custom OEM devices. Customers are free to customize the driver for their particular needs.

The major components of the device driver are as follows:

- *Windows-specific module*
  This module is common to all PLX drivers and implements the items required by all Windows drivers as specified by the Windows DDK.
- *Dispatch module*
  This module receives I/O request messages from the API DLL and dispatches control to the corresponding function used to perform the I/O operation.
- *PCI API implementation*
  This module contains the PCI API implementation for a designated PLX chip.
- *Support module*
  This module contains general support functions for the other components in the driver. Some examples are EEPROM access functions, locking of user-mode buffers, building of SGL descriptors, and IRP handling functions.
- *PLX chip-specific support module*
  This module contains the PLX chip-specific implementation of various support functions. Examples of these functions include disabling of the PCI interrupt or the chip for the power states it supports.

4.2.1 Device Driver Directory Structure

The PLX drivers are designed to take advantage of common code; therefore many files are shared between all PLX drivers. Figure 4-1 depicts the Windows device driver directory structure as found in the PLX SDK installation.



```
⊟ PciSdk
    Inc
  ⊞ Iop
  ⊞ SupportFiles
  ⊟ Win32
    ⊞ Api
    ⊟ Driver
      ⊟ Common
        ⊟ PlxChip
            Iop480
            Pci9030
            Pci9050
            Pci9054
            Pci9080
          Wdm
          WinNT
      ⊟ Wdm
          Iop480
          Pci9030
          Pci9050
          Pci9054
          Pci9080
      ⊟ WinNT
          Iop480
          Pci9030
          Pci9050
          Pci9054
          Pci9080
```

Figure 4-1 PLX Windows Driver Directory Strcuture

The driver directories are described below:

- *Common*
  The files in this directory are common to all PLX drivers.

- *Common\WinNT*
  This directory contains files common to all NT PLX drivers.

- *Common\Wdm*
  This directory contains files common to all WDM PLX drivers.

- *Common\PlxChip\<DriverName>*
  This directory contains PLX chip-specific files used for both NT and WDM drivers.

- *Common\Wdm\<DriverName>*
  This directory is used to build a chip-specific WDM version of the driver. It contains the makefile and other files used in the build process.

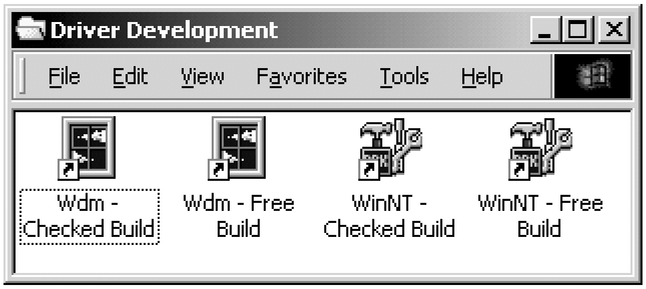
- *Common\WubNT\<DriverName>*
  This directory is used to build a chip-specific NT version of the driver. It contains the makefile and other files used in the build process.

4.2.2 Building PLX Device Drivers

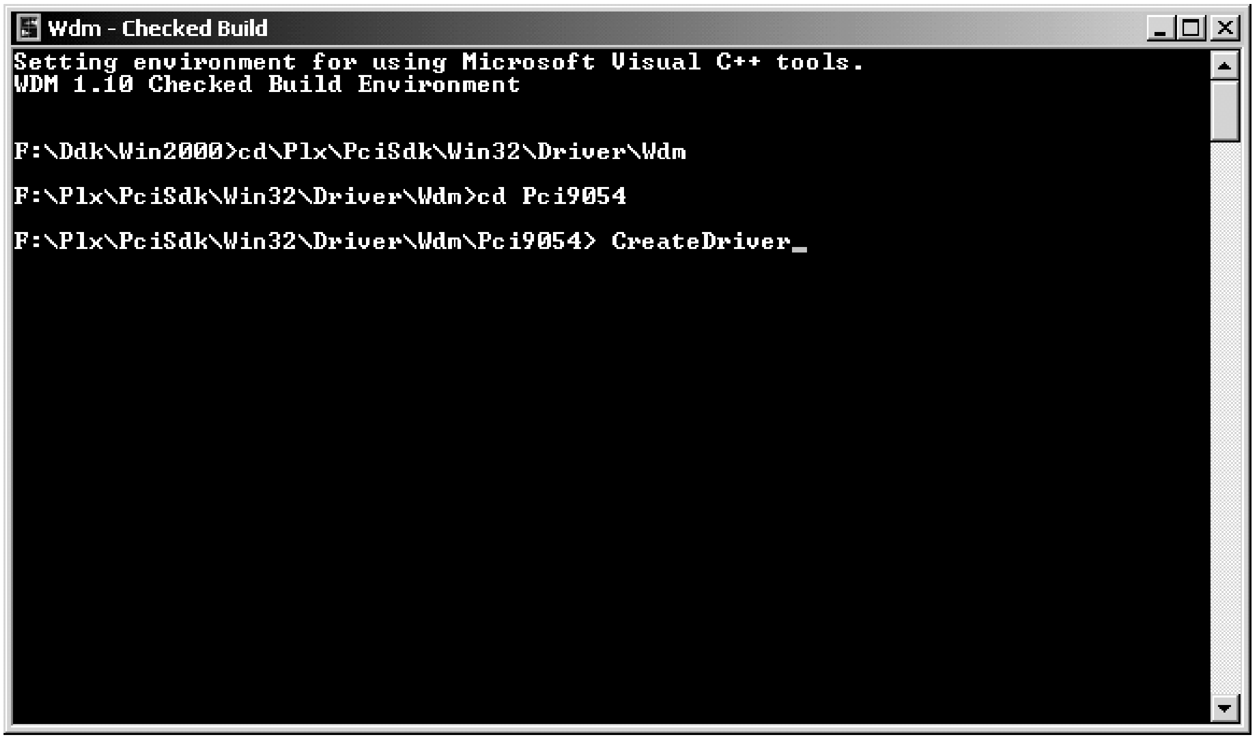
***Note**: Due to limitations in the **build** utility provided in the Windows DDK, the PLX-supplied batch file, **CreateDriver.bat**, must be used to build a driver. The **build** utility does not easily support compiling of files in a common directory; therefore, it cannot be used to build PLX drivers.*

To build a driver, the Windows DDK and Microsoft Visual C/C++ must first be installed. Follow the steps below to build the driver. The DDK environment determines the version of the driver built; otherwise, the build process is identical for all environments.

- Select and open the desired DDK environment.

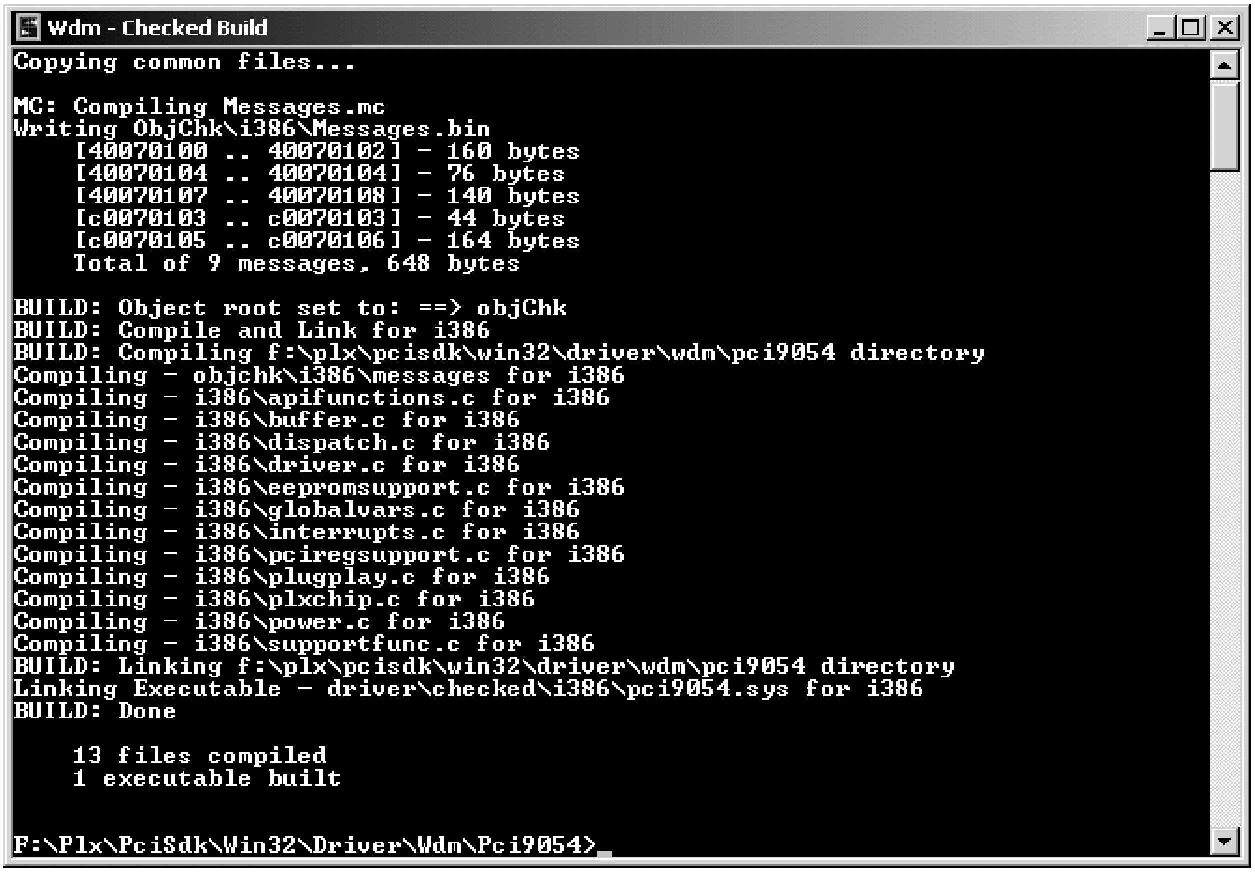


- Move to the SDK directory of the driver to build. *Note: Make sure the directory matches the build environment, e.g. Driver\Wdm\<DriverName> for WDM environments or Driver\WinNT\<DriverName> for Windows NT environments.*



```
Setting environment for using Microsoft Visual C++ tools.
WDM 1.10 Checked Build Environment


F:\Ddk\Win2000>cd\Plx\PciSdk\Win32\Driver\Wdm

F:\Plx\PciSdk\Win32\Driver\Wdm>cd Pci9054

F:\Plx\PciSdk\Win32\Driver\Wdm\Pci9054> CreateDriver_
```

- Type *CreateDriver* to build the driver



```
Copying common files...

MC: Compiling Messages.mc
Writing ObjChk\i386\Messages.bin
    [40070100 .. 40070102] - 160 bytes
    [40070104 .. 40070104] - 76 bytes
    [40070107 .. 40070108] - 140 bytes
    [c0070103 .. c0070103] - 44 bytes
    [c0070105 .. c0070106] - 164 bytes
    Total of 9 messages, 648 bytes

BUILD: Object root set to: ==> objChk
BUILD: Compile and Link for i386
BUILD: Compiling f:\plx\pcisdk\win32\driver\wdm\pci9054 directory
Compiling - objchk\i386\messages for i386
Compiling - i386\apifunctions.c for i386
Compiling - i386\buffer.c for i386
Compiling - i386\dispatch.c for i386
Compiling - i386\driver.c for i386
Compiling - i386\eepromsupport.c for i386
Compiling - i386\globalvars.c for i386
Compiling - i386\interrupts.c for i386
Compiling - i386\pciregsupport.c for i386
Compiling - i386\plugplay.c for i386
Compiling - i386\plxchip.c for i386
Compiling - i386\power.c for i386
Compiling - i386\supportfunc.c for i386
BUILD: Linking f:\plx\pcisdk\win32\driver\wdm\pci9054 directory
Linking Executable - driver\checked\i386\pci9054.sys for i386
BUILD: Done

    13 files compiled
    1 executable built


F:\Plx\PciSdk\Win32\Driver\Wdm\Pci9054>_
```

CreateDriver.bat will automatically perform the necessary steps to build the desired device driver. Since source code files are shared between drivers, the batch file will copy the common files to a single temporary directory, compile them, and then delete the directory. For those interested, *CreateDriver.bat* and the other batch files used, *WdmBuildDriver.bat* and Win*NTBuildDriver.bat*, contain many comments explaining the steps involved to build the driver.

Once the driver is built, the new *.sys* file can be used in Windows. *Refer to the Windows DDK for additional information on building and debugging drivers.*

4.3 User-mode Applications

User-mode applications connect to and use the PCI API DLL to control any PCI device with a PLX chip. For most situations, a user-mode application using the PLX API is sufficient to perform the desired functionality. PLX drivers are generic in design to minimize the need for driver customization. Typically, drivers are modified to take advantage of specific OEM hardware on a device, or possibly to add functionality, such as additional processing in the Interrupt Service Routine.

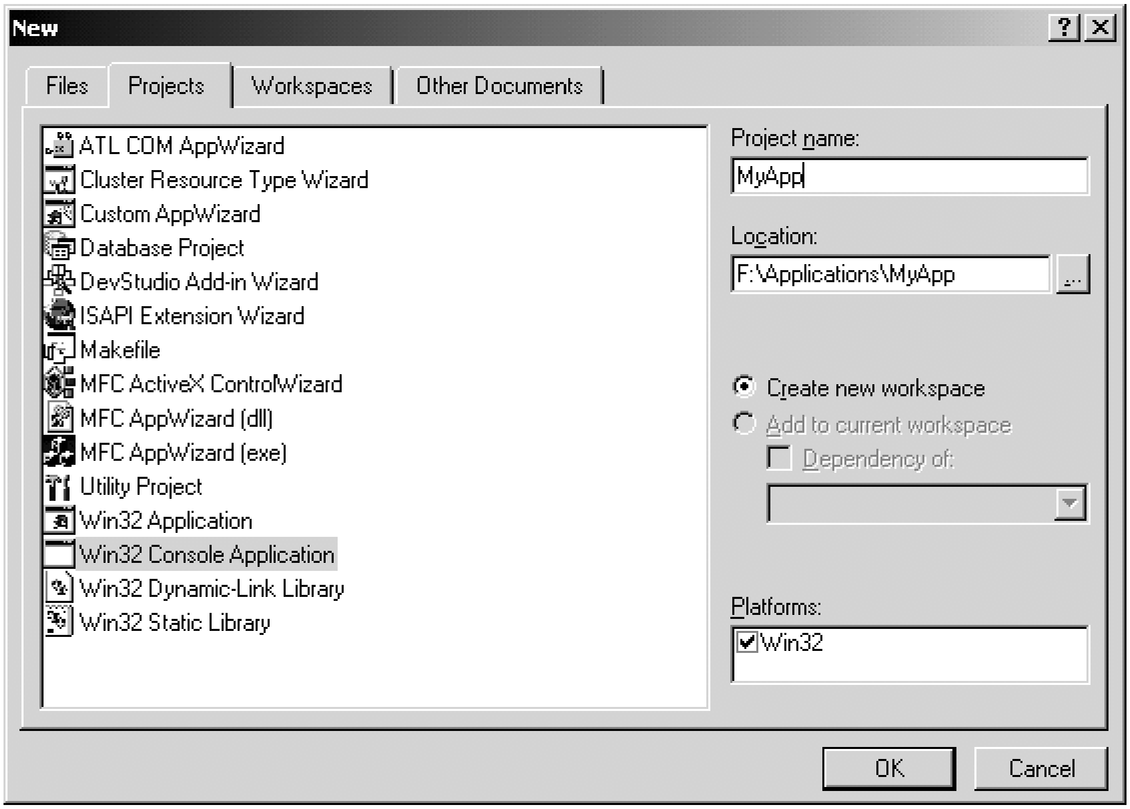
This section will explain some techniques for building user-mode applications and use of the API. The following text refers to Microsoft Visual C/C++ 6.0, but customers are free to use any compatible developer tool of preference.

4.3.1 PLX Sample Applications

Several sample applications, located in *<Sdk_Install_Dir>\Win32\Samples*, are included in the PLX SDK. These demonstrate how an application can use the PCI Host API to perform various functions with PLX PCI devices across the PCI bus. The project files included are for Microsoft Visual C/C++ 6.0.

4.3.2 Creating Windows PCI Host Applications

The first step in creating a Windows PCI Host application is to create a Microsoft Project File. A new project file can be created or one of the sample projects can be opened and modified. Typically, a *Win32 Console application* is used to create a project, but any C or C++ project, such as *MFC AppWizard*, is compatible with the PCI Host API. Figure 4-1 demonstrates the new project dialog.



Project name: MyApp

Location: F:\Applications\MyApp

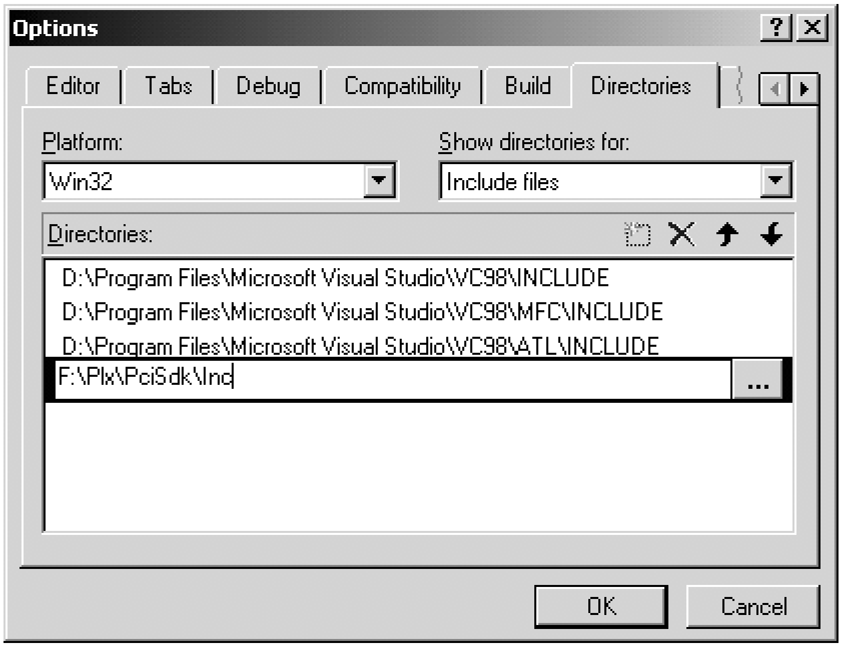
● Create new workspace

Platforms: ☑ Win32

Figure 4-1 Visual C/C++ New Project Dialog

Once the project has been opened, source code can be written and inserted into the project. Before an application can be built successfully, however, the steps below must be completed. Figure 4-3 demonstrates a typical Visual C project that is configured for the PLX API.

- ***Add the PLX SDK Include directory***
  This ensures that the development tools refer to and can find the correct version of the PLX C header files. In Visual C/C++, for example, the directory is specified in the *Options* dialog, as shown in Figure 4-2.



Platform: Win32

Show directories for: Include files

Directories:

D:\Program Files\Microsoft Visual Studio\VC98\INCLUDE
D:\Program Files\Microsoft Visual Studio\VC98\MFC\INCLUDE
D:\Program Files\Microsoft Visual Studio\VC98\ATL\INCLUDE
F:\Plx\PciSdk\Inc

Figure 4-2 Visual C/C++ Include Files Directory

- ***Define PCI_CODE***
  This definition must be defined before any PLX header file is included. Since many header files are shared between the Windows PCI Host and the local software, this definition is used to ensure that only Windows PCI Host related definitions are included. The definition can be defined using the compiler '#define' directive or defined globally in the project settings (recommended).

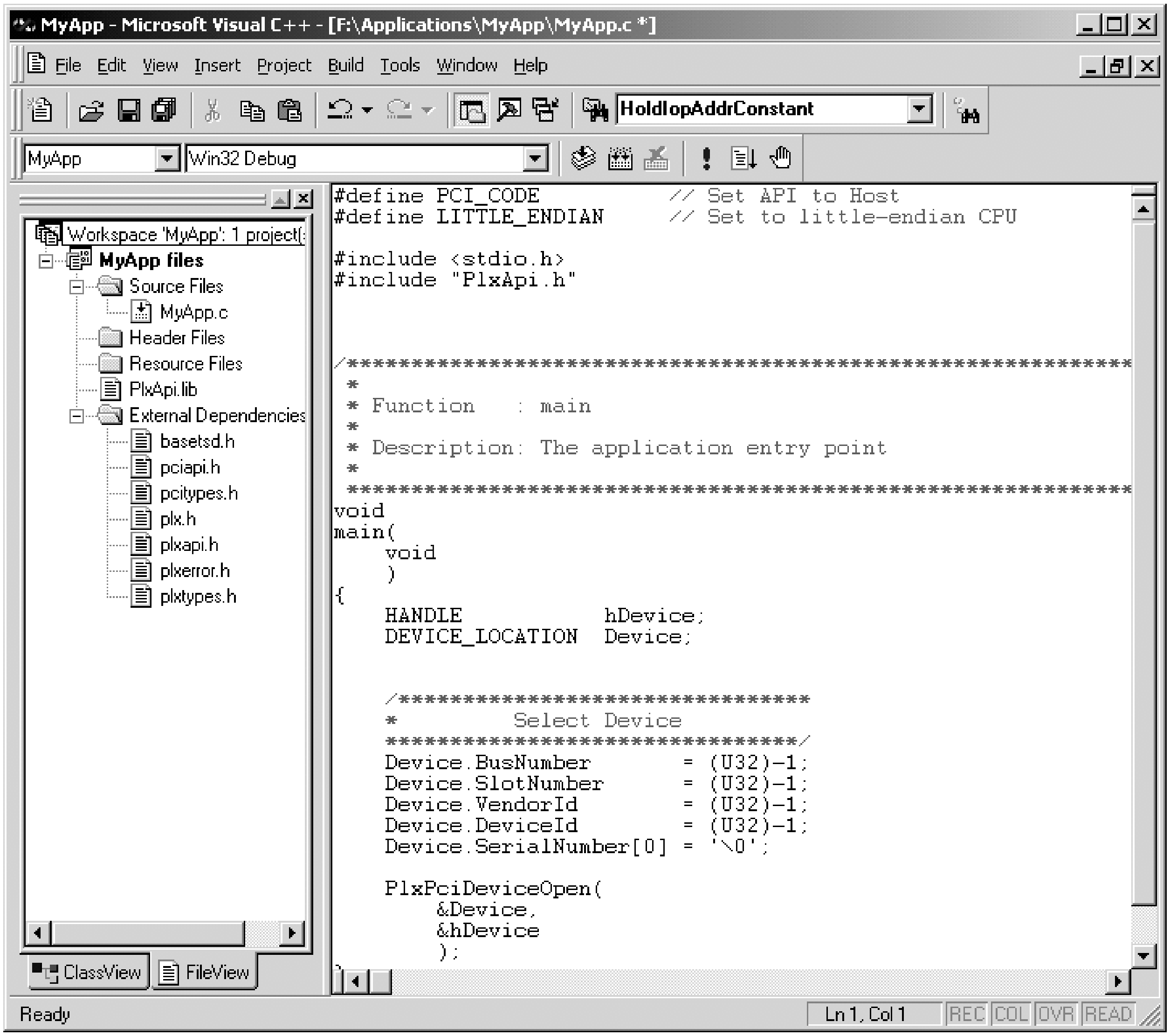
  Not: *If '#define' is used, it must come before any '#include' directives and must be included in every file where PLX-specific code is used.*

- ***Define LITTLE_ENDIAN***
  This definition must be defined before any PLX header file is included. Since many header files are shared between the Windows PCI Host and the local software, this definition is used to ensure that little-endian structures are used for the x86 CPU. The definition can be defined using the compiler '#define' directive or defined globally in the project settings (recommended).

  Note: *If '#define' is used, it must come before any '#include' directives and must be included in every file where PLX-specific code is used.*

- ***Include "PlxApi.h"***
  This file must be included to provide prototypes for PLX functions and any PLX-specific data types.

- ***Insert "PlxApi.lib" into the Project***
  This library file contains link information for the *PlxApi.dll* file. When the application is launched, the API DLL will automatically be used when a PLX API function is called. The library file is provided in the *<Sdk_Install_Dir>\Win32\Api\Release* directory.



```
#define PCI_CODE          // Set API to Host
#define LITTLE_ENDIAN     // Set to little-endian CPU

#include <stdio.h>
#include "PlxApi.h"



/*************************************************************
 *
 * Function   : main
 *
 * Description: The application entry point
 *
 *************************************************************
void
main(
    void
    )
{
    HANDLE           hDevice;
    DEVICE_LOCATION  Device;


    /*********************************
    *         Select Device
    *********************************/
    Device.BusNumber       = (U32)-1;
    Device.SlotNumber      = (U32)-1;
    Device.VendorId        = (U32)-1;
    Device.DeviceId        = (U32)-1;
    Device.SerialNumber[0] = '\0';

    PlxPciDeviceOpen(
        &Device,
        &hDevice
        );
}
```

Figure 4-3 Typical Visual C/C++ Project

4.4 PLX Chip Debug Tool - PlxMon

The PLXMon debug tool is an easy-to-use graphical debug utility for PLX devices. It's features include read/write access to PLX chip registers, download of applications to RAM, programming of FLASH devices, access to local bus devices, and EEPROM access. PLXMon access the PLX chip in one of two ways: through the PCI bus or, if BEM compatible code is running on the local-side, through a serial cable connection. Figure 4-1 shows the various modes of PLXMon. *Refer to the PLXMon User's Guide for more information on the PLXMon tool.*

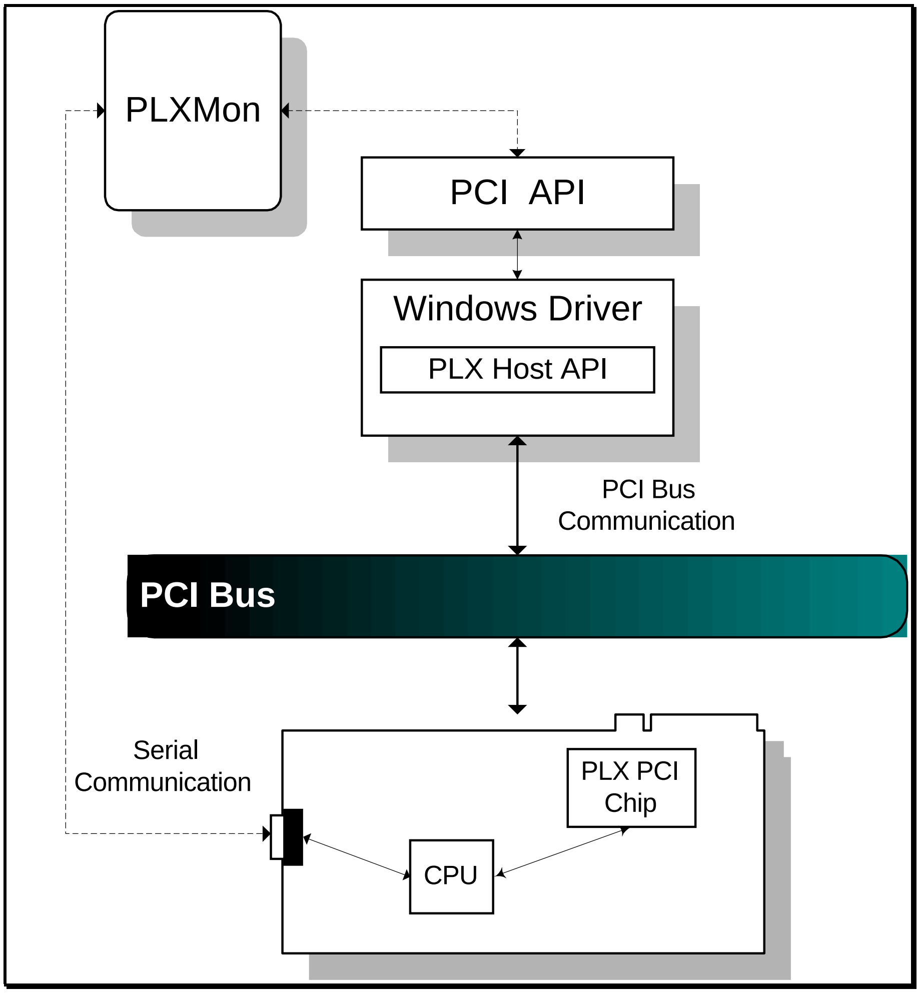


Figure 4-1 PLXMon Communications Modes

5 Local Software

This section describes the Local software provided in the PLX SDK. Figure 5-1 shows the Local software architecture. The following components are included:

- PLX Board Support Package (BSP)
- PLX Local API
- PLX Back-End Monitor (BEM)
- User Applications

***Note**: Local software is included only in the SDK-PRO and not in the SDK-LITE.*

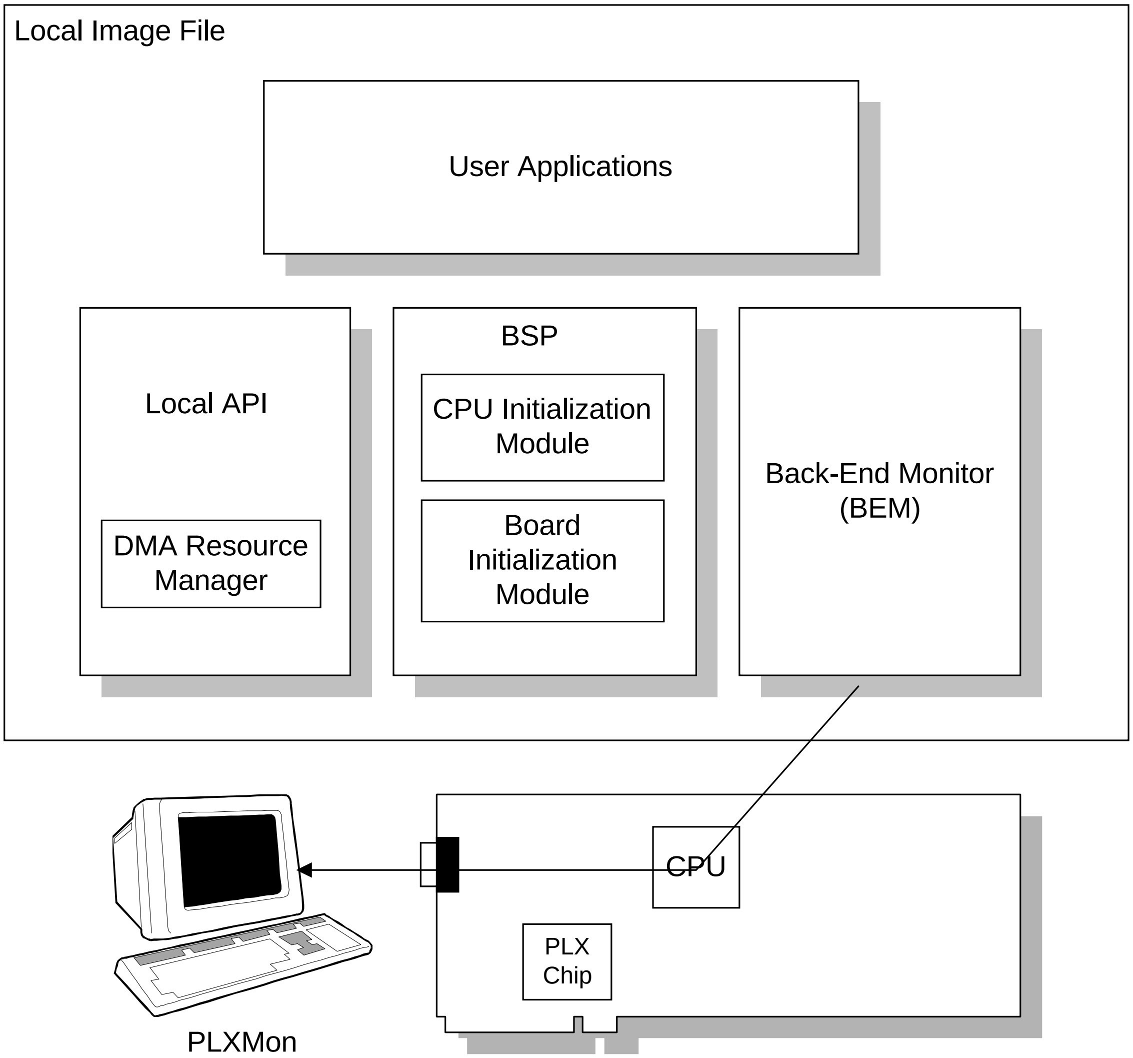


Figure 5-1 Local Software Architecture

5.1 Board Support Package (BSP)

The PLX Board Support Package contains the necessary functionality to boot the CPU and initialize all hardware components on the board. It also provides implementation-dependant services to the other local software components. By definition, the BSP contains all board-specific code, which allows for modularity in design and re-usable portions for non-PLX BSPs. The BSPs are located in *<Sdk_Install_Dir>\Iop\Bsp\<BspName>*. The BSP naming convention generally used by PLX, although not required, is *<PLX_Chip>-<Cpu>.rdk*.

The PLX BSPs follow a common design architecture to minimize porting efforts to new hardware designs. Figure 5-1 depicts the flow of the PLX BSP.

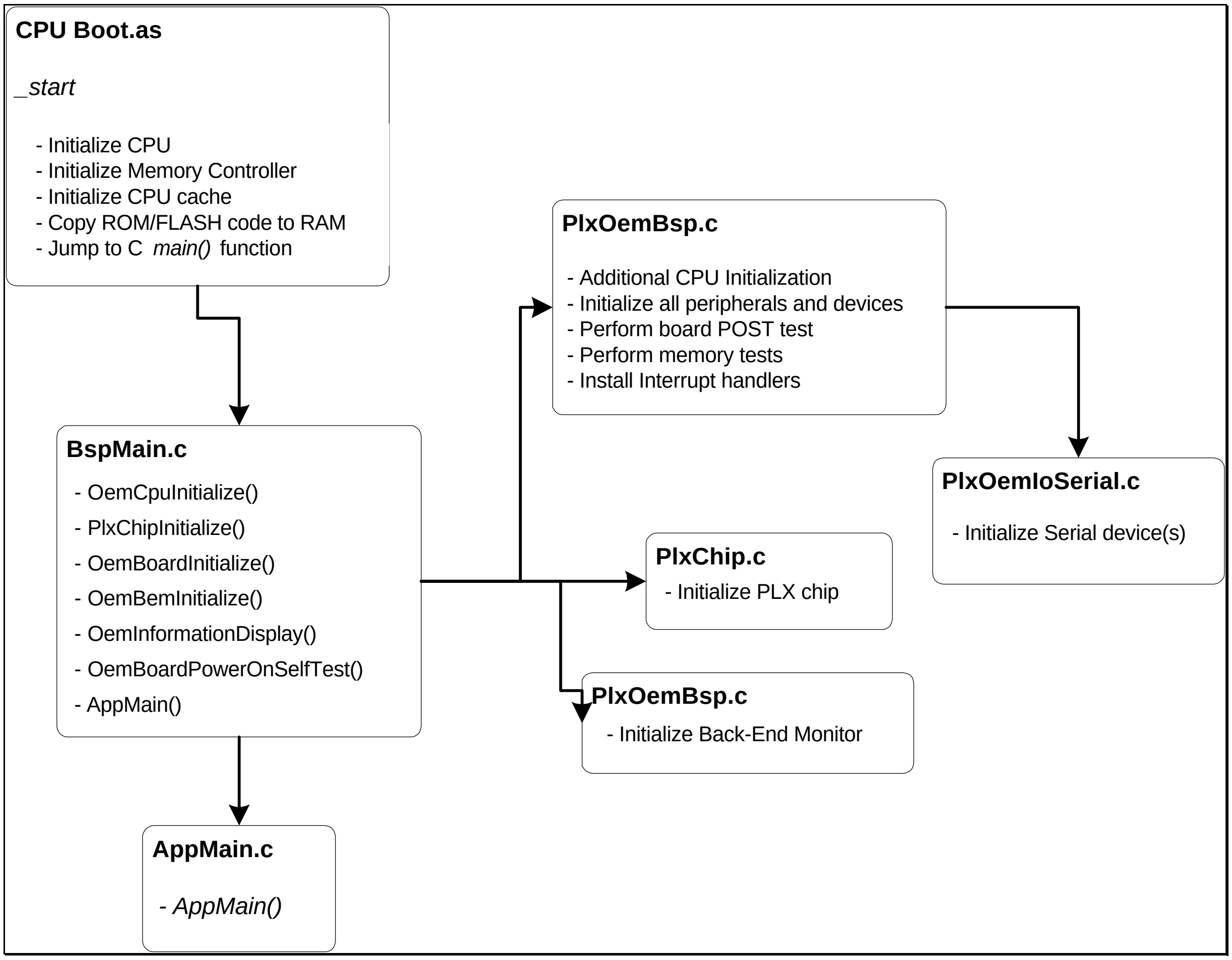

CPU Boot.as
_start
- Initialize CPU
- Initialize Memory Controller
- Initialize CPU cache
- Copy ROM/FLASH code to RAM
- Jump to C main() function
PlxOemBsp.c
- Additional CPU Initialization
- Initialize all peripherals and devices
- Perform board POST test
- Perform memory tests
- Install Interrupt handlers
BspMain.c
- OemCpuInitialize()
- PlxChipInitialize()
- OemBoardInitialize()
- OemBemInitialize()
- OemInformationDisplay()
- OemBoardPowerOnSelfTest()
- AppMain()
PlxOemIoSerial.c
- Initialize Serial device(s)
PlxChip.c
- Initialize PLX chip
PlxOemBsp.c
- Initialize Back-End Monitor
AppMain.c
- AppMain()

Figure 5-1 PLX BSP Flowchart

5.1.1 BSP Common Source Code

The PLX BSP directory does not contain all source code found in the BSP library. The directory *<Sdk_Install_Dir>\Iop\Common* contains multiple files that are shared among all BSPs. These files include generic support functions, such as file download and EEPROM access. When hardware-dependent functionality is required, such as serial I/O, the files will call standard functions implemented in the BSP.

5.1.2 Porting the PLX BSP to Custom Designs

The PLX BSP was designed to minimize porting efforts to custom designs. In the source code, a standard naming convention is used to provide developers with a straightforward path during the porting effort. To port the PLX BSP, follow the steps below:

1. Go to the BSP directory to determine which BSP most closely matches the custom design. Copy the directory to another and name it accordingly.
2. Using the BSP flowchart in Figure 5-1 as a reference, the modification of source files can commence with the following guidelines:
   - By default, the CPU entry point is labeled *_start* in the file *CpuBoot.as*. The first instruction executed by the CPU is here. Insert the required code to initialize the CPU and other necessary peripherals, such as the memory controller. The boot code may also perform other tasks such as copy code from ROM/FLASH to RAM and jump there.
   - After CPU initialization, the C entry point *main()* should be called. The *main()* function will then continue the initialization process by calling multiple support functions found in the BSP.
   - In the BSP directory, many files are named in the format *PlxOemXxx.c*. The *PlxOem* prefix is a flag to developers noting that the files contain functions, which must be implemented for the BSP to function correctly. Furthermore, most of these functions are named with an *Oem* prefix, again to aid developers in the port. These functions provide entry points to allow for custom initialization and to contain board-dependent implementations of support functions used later by all other Local software components.
   - Include any additional required files to perform other essential tasks, such as installation of exception vectors and CPU cache support functions.
   - Modify the makefile as necessary to successfully build the BSP library.
3. Build the Local API and the BEM libraries if the local CPU does not match one of the shipped libraries. This step may require makefile changes to support the local CPU.
4. Go to the desired application directory and modify the makefile to reference the new custom BSP.

At this point, the porting effort is generally complete. Of course, the amount of work involved is dependent upon the complexity of the design in question. The steps above are provided as a general guideline approach. The PLX BSPs are written in a manner that is easy to follow and much of the code can be re-used in custom BSPs.

5.2 Local API

The Local API, much like the Host API, is a powerful tool provided to utilize the features of PLX chips. The API applies to all PLX chips capable of mastering the bus, although the implementation may differ between chips. The chip-specific implementation of the local API is provided in *<Sdk_Install_Dir>\Iop\Api\<ChipName>*.

The Local API contains the code for all documented API functions supported by the PLX chip. This code is completely modular in design and, therefore, independent of the board configuration. As a result, there is little or no effort involved for the API if it is ported to a custom design.

***Note**: It is important to note that many of the Local API functions are similar to the Host API functions with regards to function names and functionality; however, there are a number of differences. Parameters are different in most cases and, although the result may be the same, the API implementations differ because the local CPU accesses the PLX chip directly, versus across the PCI bus in a Host environment.*

5.3 Back-End Monitor (BEM)

The PLX Back-End Monitor is a transparent target server provided to support communications with PLXMon in Serial Mode. As of SDK 3.2, the BEM is interrupt-driven, versus previous SDK releases where it required cooperative scheduling with the application. The BEM supports many features such as memory read/write, EEPROM access, downloading of applications, and programming of FLASH. The BEM serial protocol details are provided in 7.3Appendix B for those interested.

The BEM can be a useful tool for debugging hardware and software. Additionally, it can be deployed in the customer product for future troubleshooting because it provides an entry point into the local bus through a serial connection.

The BEM works by parsing serial input data as it arrives from the serial port. If a BEM message header is found, serial data is routed to the BEM until the end of message is received; otherwise, all other data is appended to the serial input queue. Figure 5-1 demonstrates the BEM as it parses serial input data.

Note: *The BEM does not support CPU or source-level debugging. The BEM is implemented in software, meaning the local CPU must be running valid code.*

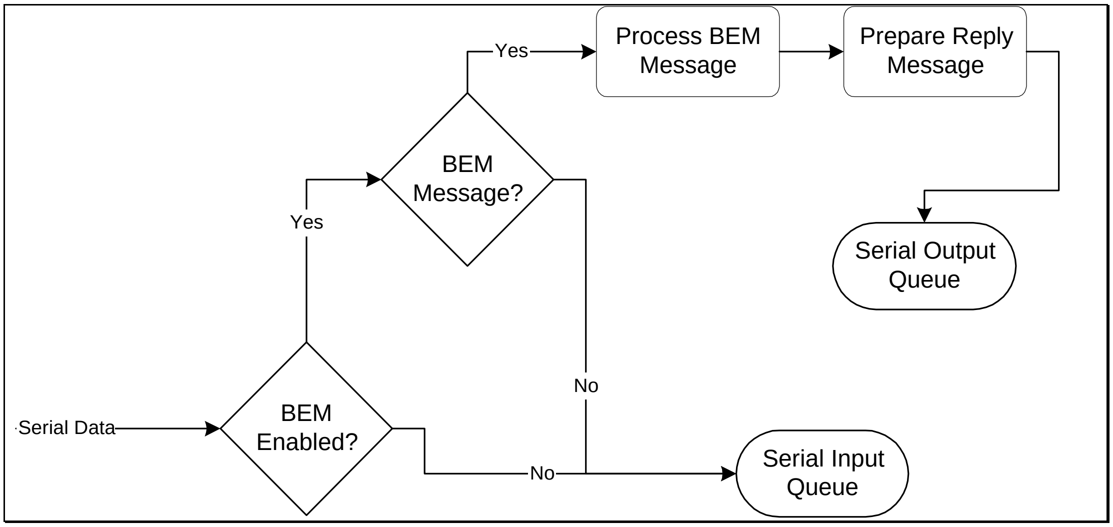


Figure 5-1 PLX BEM Serial Data Parsing

5.4 Local Applications

The Local API, BSP, and Back-End Monitor libraries are linked with the Local application to create the final image for download. If this is a ROM image, it is programmed into FLASH memory. RAM images are downloaded to RAM memory and executed.

All Local applications contain an *AppMain()* function, which will be called by the BSP when all initialization has completed. This function is the user entry point where application code begins. At this point, the application can assume that the board is functioning properly and all peripherals are initialized. Additionally, the application can use multiple support functions provided by the BSP, API, and BEM.

5.5 Local Memory Usage

Memory used by PLX applications is shown in Figure 5-1. Although code varies from CPU to CPU, the basic steps are identical on all platforms.

1. CPU starts execution at its boot vector, which is located in the FLASH/ROM region.
2. The boot code performs only necessary initialization, such as CPU registers and memory controller.
3. Once the RAM device is initialized, ROM code is copied to RAM and the CPU begins to execute code from RAM. ROM code is typically copied to the first available area in RAM to avoid fragmentation.
4. Additional boot code copies the CPU exception vectors to their proper location

RAM applications, in a PLX environment, require the local CPU to already by running valid code. This is because PLXMon communicates with the local CPU to provide the application entry point. RAM applications perform initialization similar to ROM application, except where it is not applicable, such as copying of code to RAM or RAM device initialization.

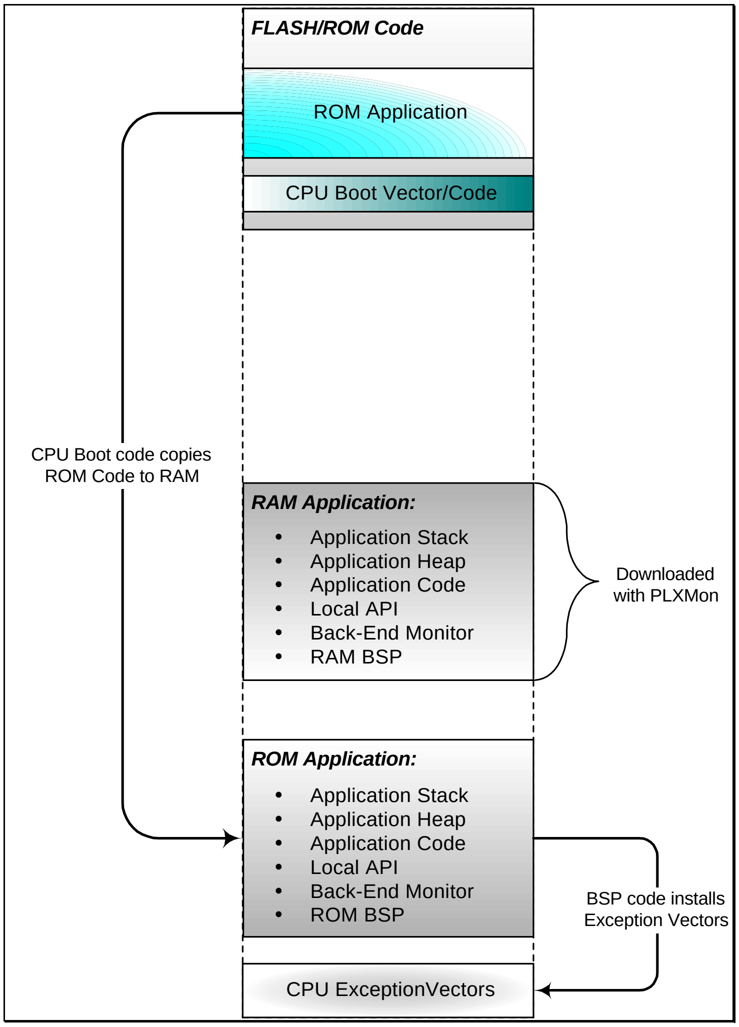


Figure 5-1 Typical Local Memory Usage

5.6 Local-Side Directory Structure

The Local-side code is organized into multiple directories in order to take advantage of common code; and maximize modularity. Figure 5-1 depicts the Local-side directory structure as found in the PLX SDK installation.

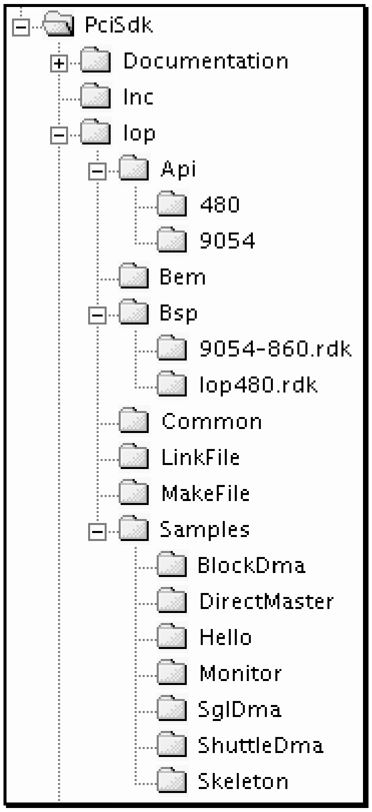


Figure 5-1 Local-side Directory Strcuture

The directories, relative to *PciSdk\Iop* are described below:

- *Api\<PlxChipName>*
  The chip-specific Local API implementation.
- *Bem*
  This directory contains the BEM module.
- *Bsp\<BspName>*
  The complete PLX BSP for supported PLX RDKs.
- *Common*
  This directory contains files shared by all BSPs.
- *LinkFile*
  This directory contains the linker directive files shared by all applications. These files specify the memory usage and code/data placement to the linker.
- *MakeFile*
  This directory contains common makefiles used for building BSPs, libraries, and applications.
- *Samples\<SampleName>*
  These directories contain sample applications demonstrating use of the PLX API and how to write code in the PLX environment.

5.7 Creating Local-side Images

The creation of local-side images is typically performed in a MS-DOS environment on Windows systems; however, most items in this section apply to other platforms as well, such as Unix. This section explains the details of creating and building PLX local applications, BSPs, and libraries.

5.7.1 Make Files

Make files are an essential component of the build process. These provide precise control of how to build applications and affect multiple items, such as location of built images, which compiler toolset to use, the type of image to build, etc. PLX make files include numerous comments detailing the items included. There are many references for make file format and syntax, including the MSDN documentation.

PLX make files are modular in design to allow for versatility among toolsets and sharing of common files. Figure 5-1 demonstrates the flow during make file processing.

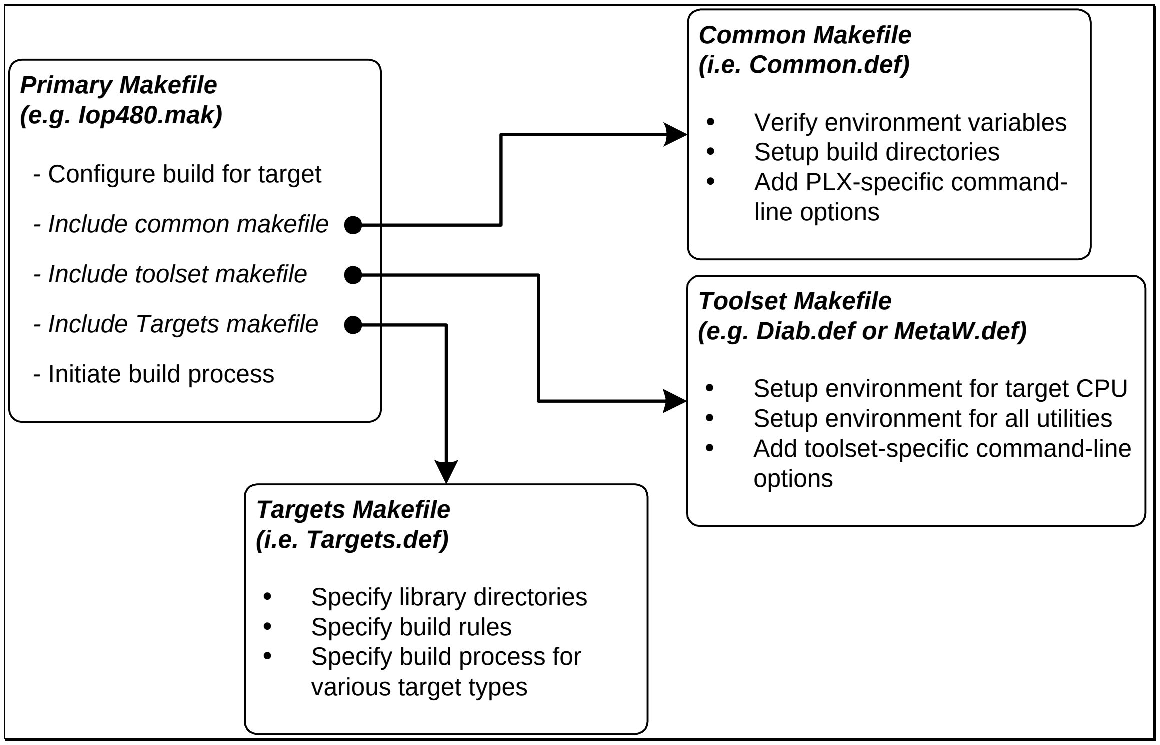

Primary Makefile
(e.g. Iop480.mak)
- Configure build for target
- Include common makefile
- Include toolset makefile
- Include Targets makefile
- Initiate build process
Common Makefile
(i.e. Common.def)
• Verify environment variables
• Setup build directories
• Add PLX-specific command-line options
Toolset Makefile
(e.g. Diab.def or MetaW.def)
• Setup environment for target CPU
• Setup environment for all utilities
• Add toolset-specific command-line options
Targets Makefile
(i.e. Targets.def)
• Specify library directories
• Specify build rules
• Specify build process for various target types

Figure 5-1 Make File Processing Flow

Make files are named according to the platform supported. The file name determines which platform a particular make file will build for. The format is as follows:

<***PLX_Chip***> – <***CPU***>*.mak*

where:

PLX_Chip = Numeric associated with the PLX Chip model (*for example*, 54=9054).

CPU = Numeric associated with CPU type (*for example*, 860=Motorola 860).

The make file for the PCI 9054RDK-860 is then *54-860.mak*.

All make files are similar in structure and require minimal changes for recompilation on a developer's system. Any text editor can be used to modify a make file as necessary. There are many comments and instructions throughout the make files that allow for customization.

For ease of maintenance, PLX uses the *Nmake* make file processing utility provided with Visual C/C++. Typically, toolsets provide their own make file processing utility, such as *Dmake* in the DIAB tools. All PLX make files are compatible with *Nmake*, but should work as-is with other make file utilities. If not, the required modifications should be minimal. Developers are free to use their make file utility of choice.

The make files use a common command-line interface file as follows:

Nmake –f <*Makefile*> [*Clean* | *CleanAll*]

where:

Makefile = Make file name

Clean = Delete all intermediate build files and leave final image

CleanAll = Delete all build files, including final image

5.7.2 Linker Command Files

When building local-side images, the linker references a special file to specify how sections are organized in memory. These files are called Linker Command or Linker Directive files. These files specify where to place modules in memory and heap and stack allocation. Two directive files are provided—a ROM version for booting from a FLASH and a RAM version for building an application executing in RAM. The files are located in *<Sdk_Install_Dir>\Iop\LinkFile* and are named using the following syntax:

<ROM or RAM>-<BSP Name>.ld

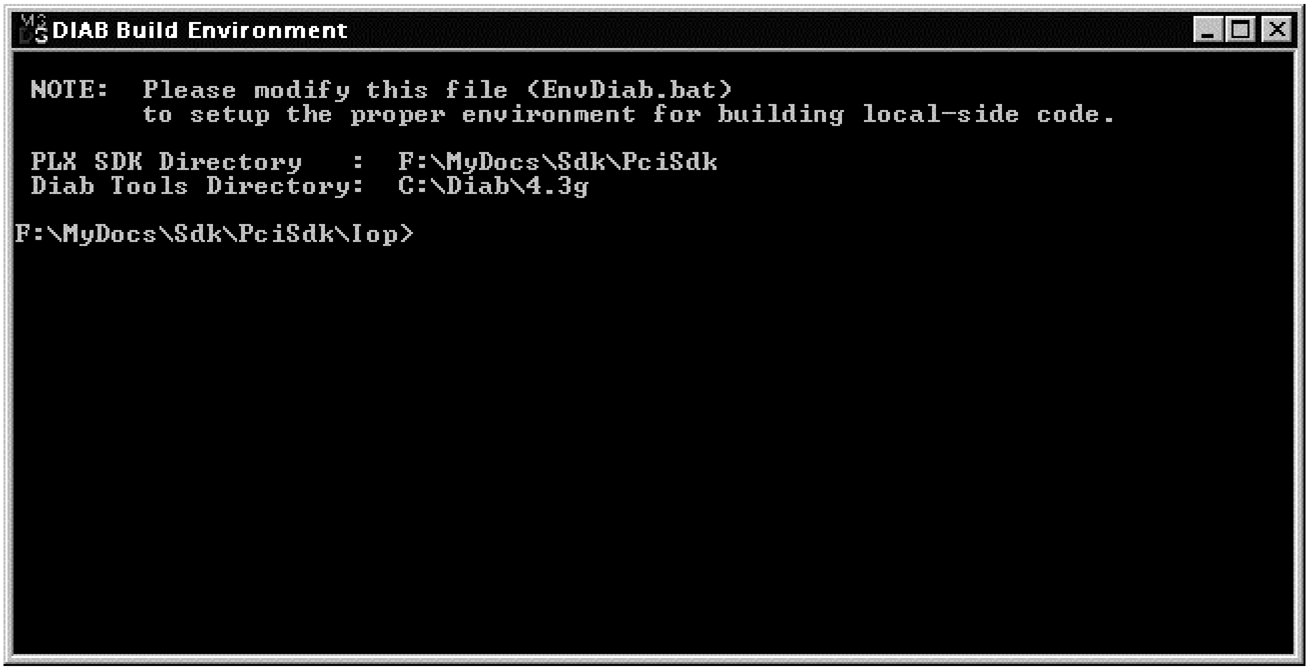
During the link phase, the make file will reference a specific Linker Command file, depending upon the build type and BSP selected.

The Linker Command files provided in the SDK are compatible with the DIAB toolset. The Metaware toolset supports this format, as well, with a specific command-line option. If other toolsets are used, new Linker Command files may be required.

5.7.3 Building Local-side Images

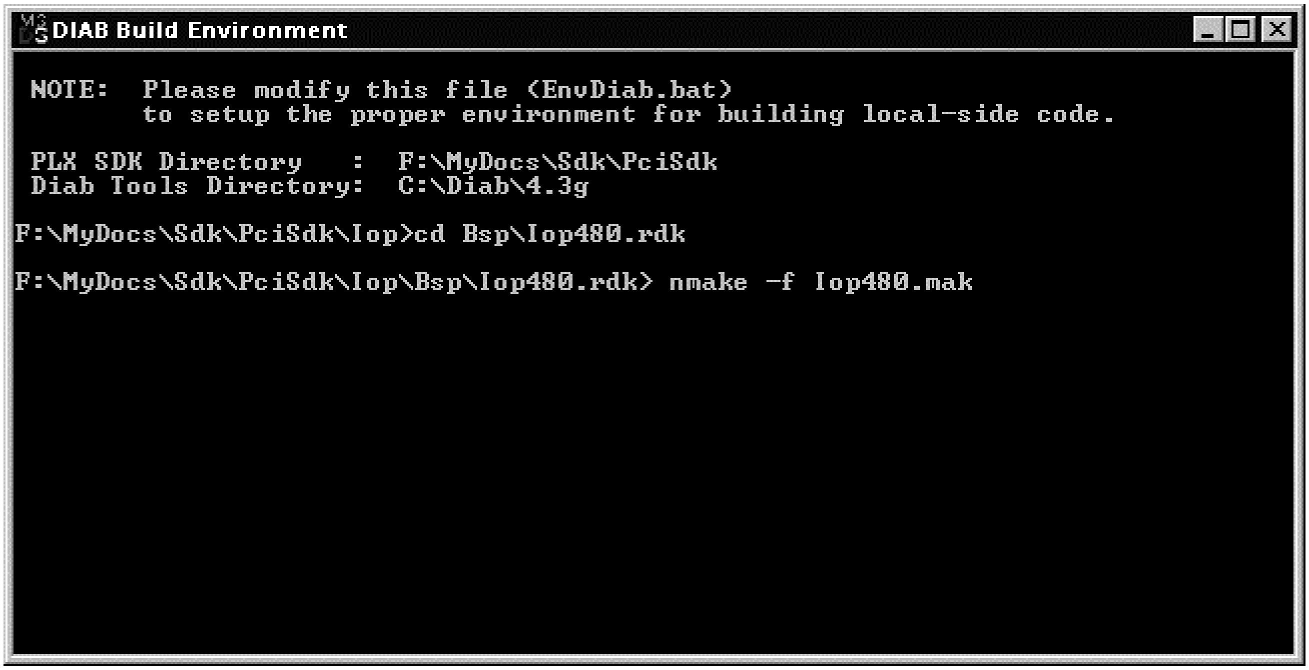
Local-side images are built in an MS-DOS environment using environment variables for configuration options. When building local applications, it is important to note that applications link with the API, BEM, and BSP libraries. As a result, these libraries must be built before applications are linked. To build a local image, perform the following steps:

- Open the desired build environment, using the provided PLX icon. This opens an MS-DOS window and calls the *EnvXxx.bat* file, where Xxx is the compiler toolset. ***Note:*** *Make sure to modify the selected EnvXxx.bat file to match the installation directories. The file contains comments to explain the necessary changes.*



```
 NOTE:  Please modify this file (EnvDiab.bat)
        to setup the proper environment for building local-side code.

 PLX SDK Directory   :  F:\MyDocs\Sdk\PciSdk
 Diab Tools Directory:  C:\Diab\4.3g

F:\MyDocs\Sdk\PciSdk\Iop>
```

- Go to the desired build directory and use *Nmake* to build the desired target. ***Note:*** *Ensure that the make utility is in the <PATH> environment variable.*

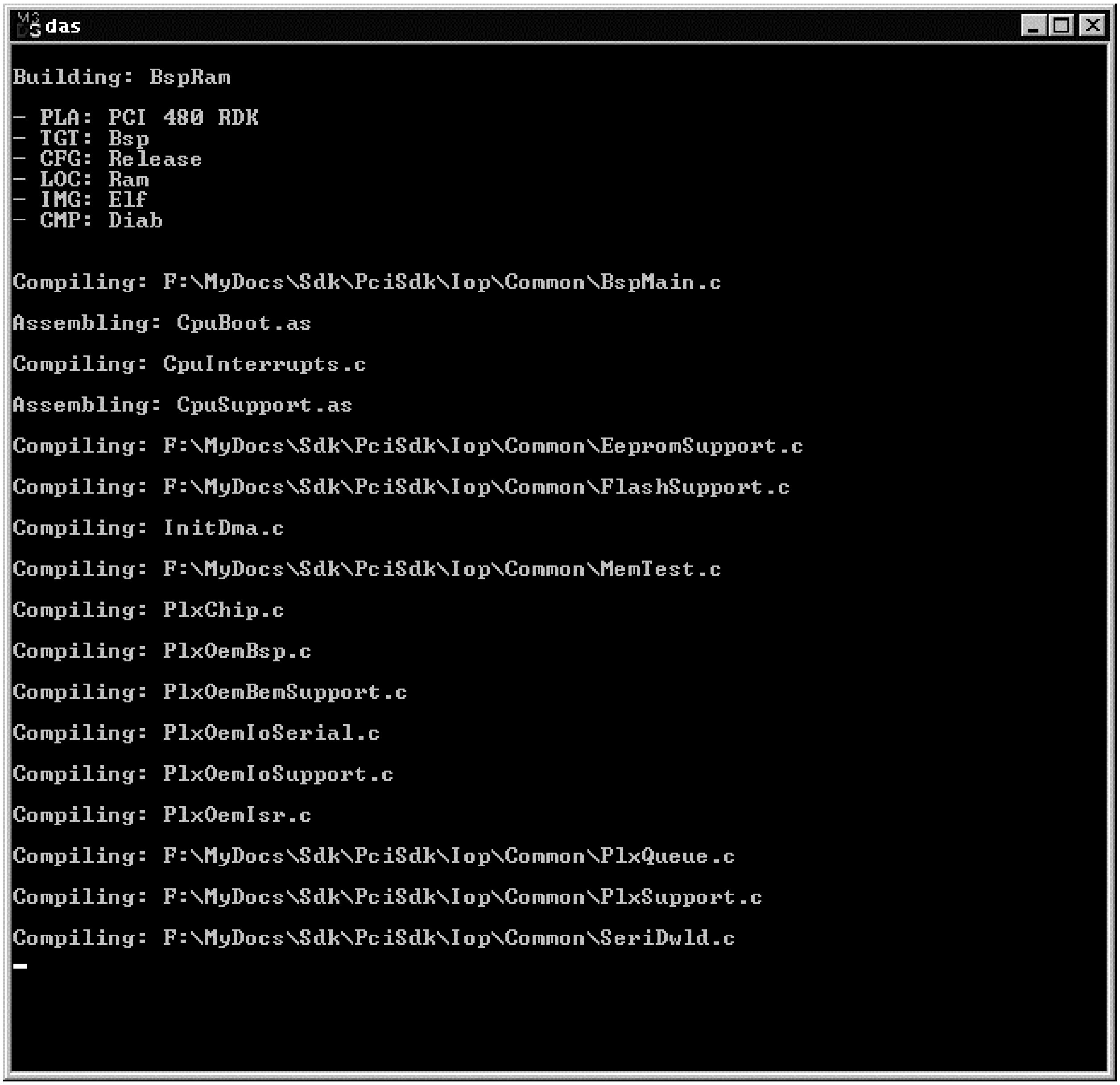


```
 NOTE:  Please modify this file (EnvDiab.bat)
        to setup the proper environment for building local-side code.

 PLX SDK Directory   :  F:\MyDocs\Sdk\PciSdk
 Diab Tools Directory:  C:\Diab\4.3g

F:\MyDocs\Sdk\PciSdk\Iop>cd Bsp\Iop480.rdk

F:\MyDocs\Sdk\PciSdk\Iop\Bsp\Iop480.rdk> nmake -f Iop480.mak
```

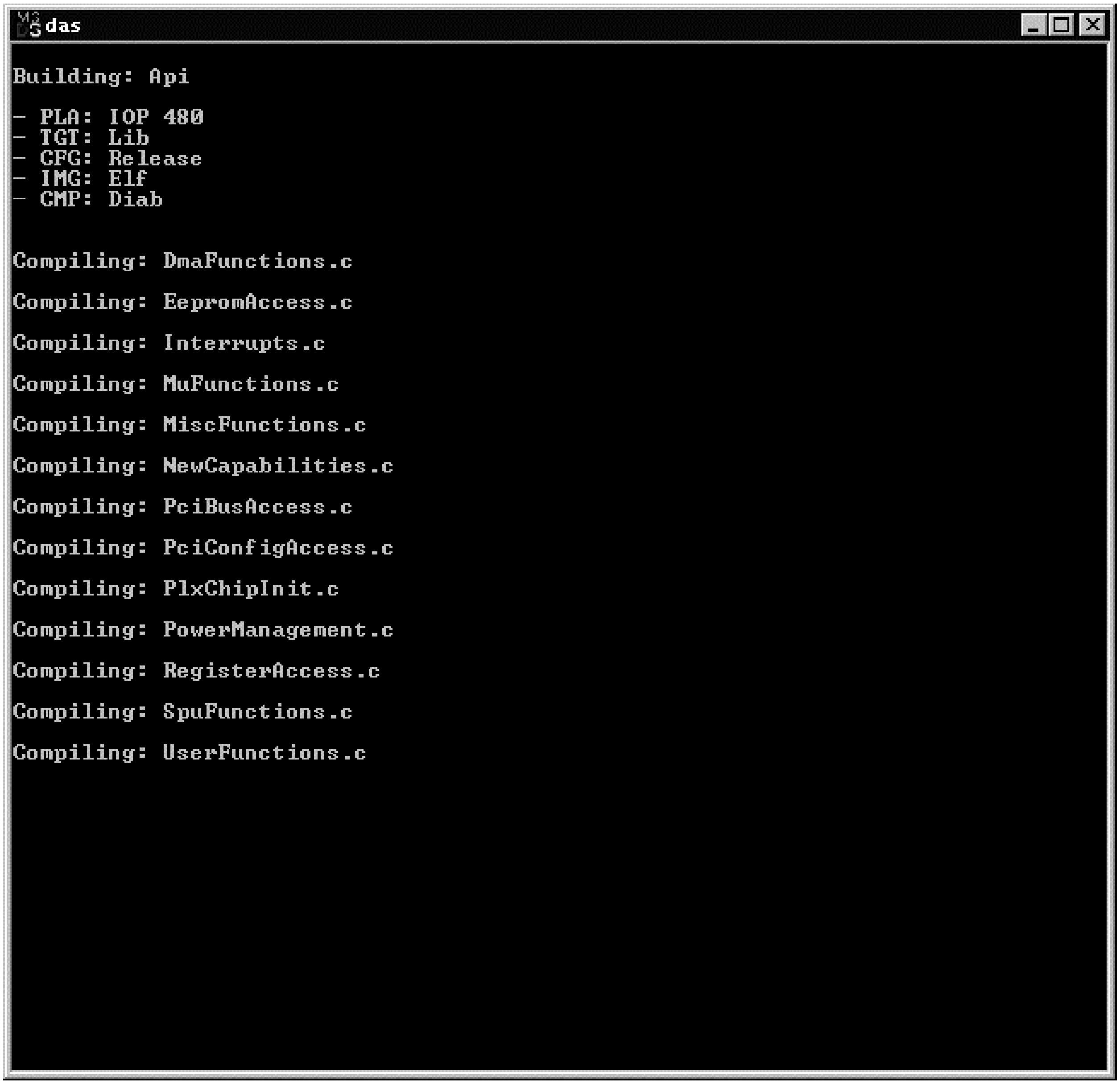
- The target will build similar to the figures below. During the build process, information about the target will be displayed as follows:

  PLA: The target platform
  TGT: The target type to build, i.e. BSP, Library, or Application
  CFG: Build configuration type, Release or Debug build
  LOC: The selected location of the target, ROM or RAM
  IMG: The selected image type, ELF, COFF, or Binary
  CMP: The selected compiler toolset

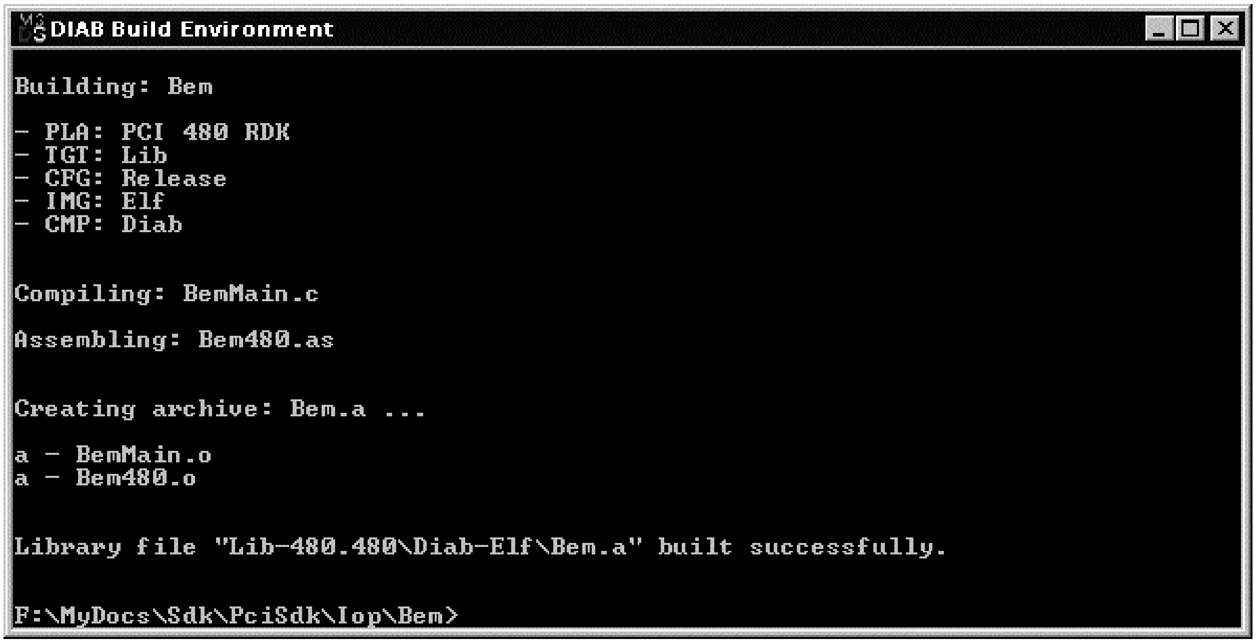
  The following figures demonstrate build of the BSP library, API library, BEM library, and final application image.



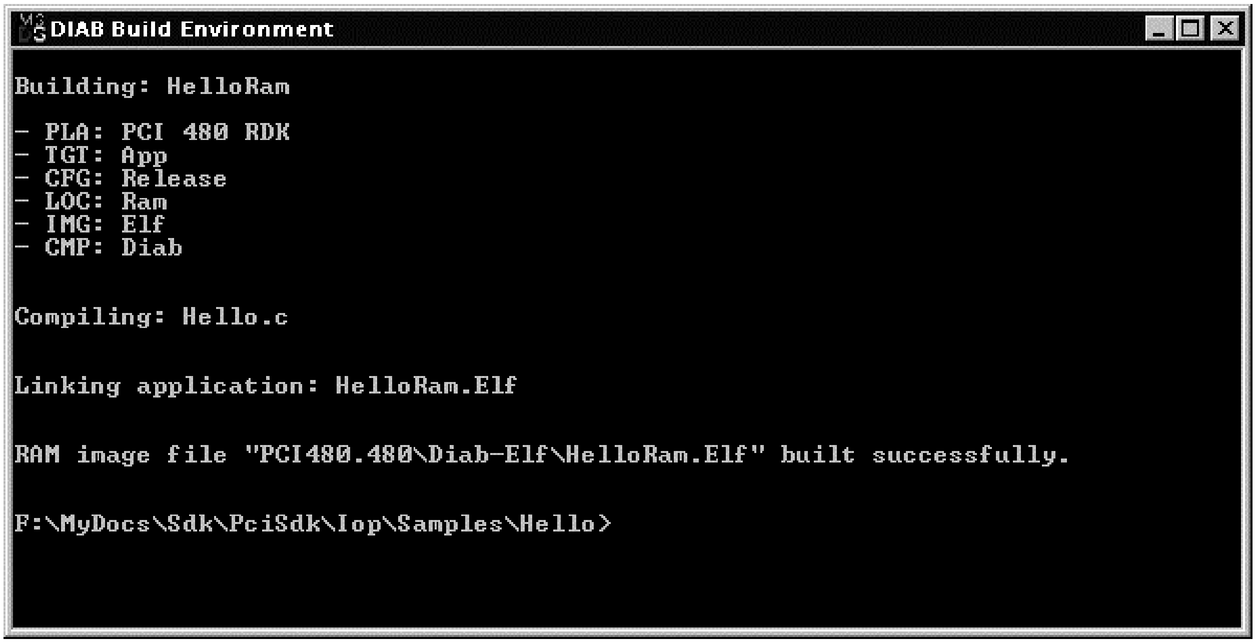
```
Building: BspRam

- PLA: PCI 480 RDK
- TGT: Bsp
- CFG: Release
- LOC: Ram
- IMG: Elf
- CMP: Diab


Compiling: F:\MyDocs\Sdk\PciSdk\Iop\Common\BspMain.c

Assembling: CpuBoot.as

Compiling: CpuInterrupts.c

Assembling: CpuSupport.as

Compiling: F:\MyDocs\Sdk\PciSdk\Iop\Common\EepromSupport.c

Compiling: F:\MyDocs\Sdk\PciSdk\Iop\Common\FlashSupport.c

Compiling: InitDma.c

Compiling: F:\MyDocs\Sdk\PciSdk\Iop\Common\MemTest.c

Compiling: PlxChip.c

Compiling: PlxOemBsp.c

Compiling: PlxOemBemSupport.c

Compiling: PlxOemIoSerial.c

Compiling: PlxOemIoSupport.c

Compiling: PlxOemIsr.c

Compiling: F:\MyDocs\Sdk\PciSdk\Iop\Common\PlxQueue.c

Compiling: F:\MyDocs\Sdk\PciSdk\Iop\Common\PlxSupport.c

Compiling: F:\MyDocs\Sdk\PciSdk\Iop\Common\SeriDwld.c
```



```
Building: Api

- PLA: IOP 480
- TGT: Lib
- CFG: Release
- IMG: Elf
- CMP: Diab


Compiling: DmaFunctions.c

Compiling: EepromAccess.c

Compiling: Interrupts.c

Compiling: MuFunctions.c

Compiling: MiscFunctions.c

Compiling: NewCapabilities.c

Compiling: PciBusAccess.c

Compiling: PciConfigAccess.c

Compiling: PlxChipInit.c

Compiling: PowerManagement.c

Compiling: RegisterAccess.c

Compiling: SpuFunctions.c

Compiling: UserFunctions.c
```

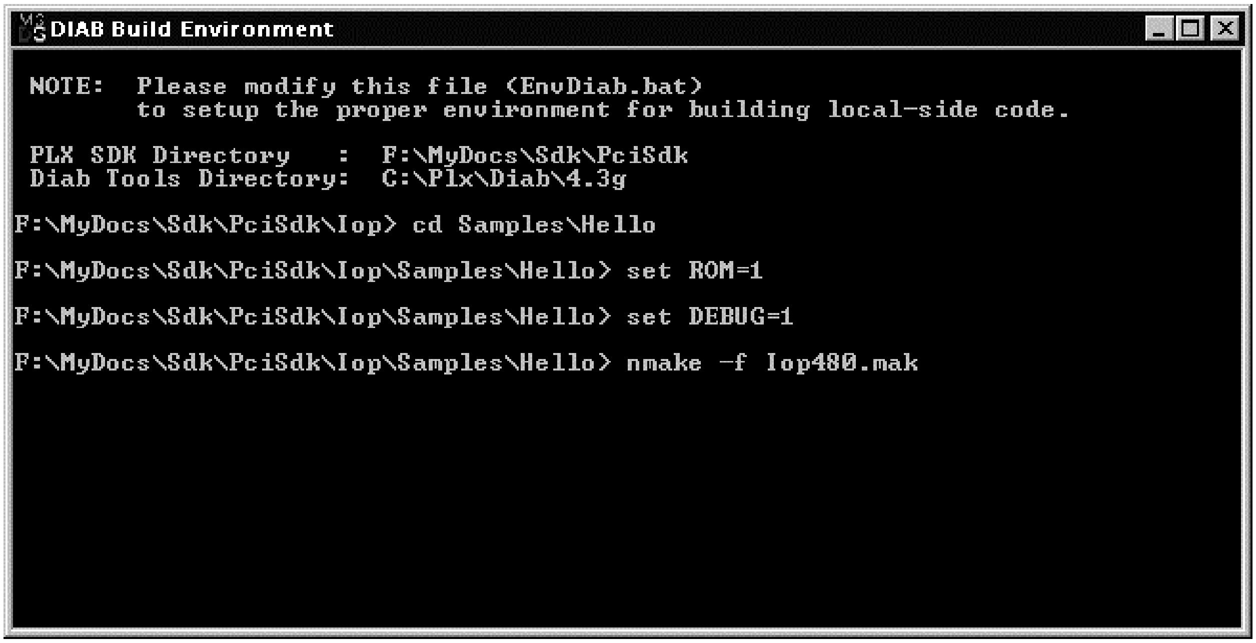


```
Building: Bem

- PLA: PCI 480 RDK
- TGT: Lib
- CFG: Release
- IMG: Elf
- CMP: Diab


Compiling: BemMain.c

Assembling: Bem480.as


Creating archive: Bem.a ...

a - BemMain.o
a - Bem480.o


Library file "Lib-480.480\Diab-Elf\Bem.a" built successfully.


F:\MyDocs\Sdk\PciSdk\Iop\Bem>
```

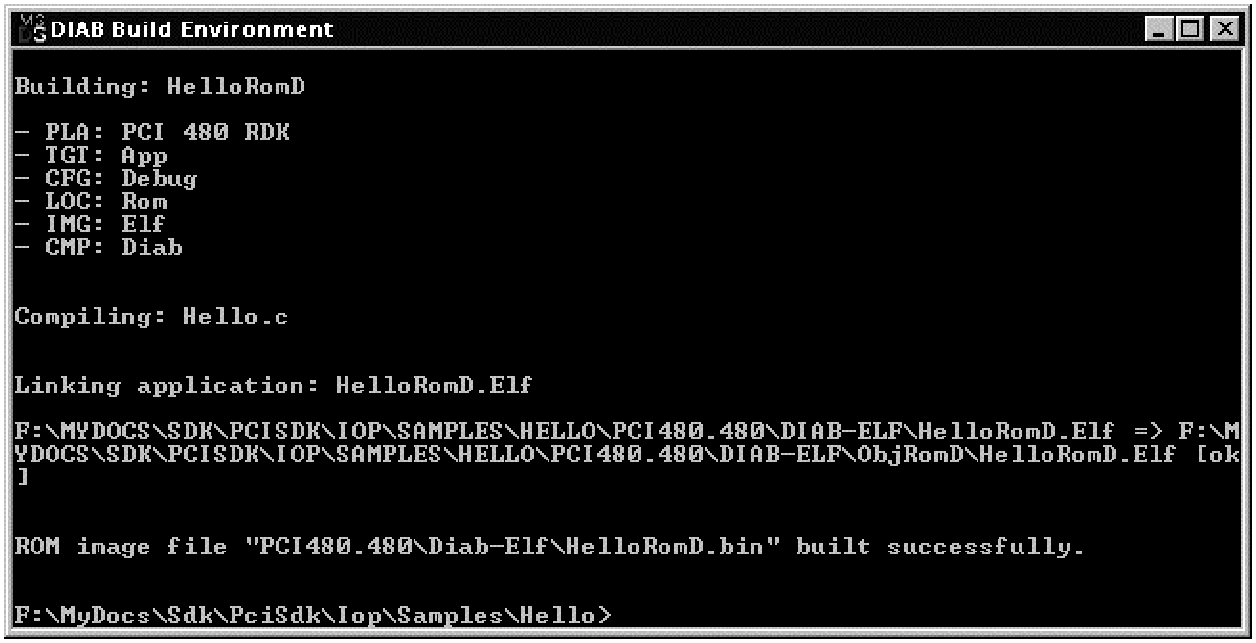


```
Building: HelloRam

- PLA: PCI 480 RDK
- TGT: App
- CFG: Release
- LOC: Ram
- IMG: Elf
- CMP: Diab


Compiling: Hello.c


Linking application: HelloRam.Elf


RAM image file "PCI480.480\Diab-Elf\HelloRam.Elf" built successfully.


F:\MyDocs\Sdk\PciSdk\Iop\Samples\Hello>
```

- To change configuration parameters, modify environment variables as needed. Settings in the make file override environment variables. The figures below demonstrate a ROM build with Debug symbol information in the image.



```
 NOTE:  Please modify this file (EnvDiab.bat)
        to setup the proper environment for building local-side code.

 PLX SDK Directory   :  F:\MyDocs\Sdk\PciSdk
 Diab Tools Directory:  C:\Plx\Diab\4.3g

F:\MyDocs\Sdk\PciSdk\Iop> cd Samples\Hello

F:\MyDocs\Sdk\PciSdk\Iop\Samples\Hello> set ROM=1

F:\MyDocs\Sdk\PciSdk\Iop\Samples\Hello> set DEBUG=1

F:\MyDocs\Sdk\PciSdk\Iop\Samples\Hello> nmake -f Iop480.mak
```



```
Building: HelloRomD

- PLA: PCI 480 RDK
- TGT: App
- CFG: Debug
- LOC: Rom
- IMG: Elf
- CMP: Diab


Compiling: Hello.c


Linking application: HelloRomD.Elf

F:\MYDOCS\SDK\PCISDK\IOP\SAMPLES\HELLO\PCI480.480\DIAB-ELF\HelloRomD.Elf => F:\M
YDOCS\SDK\PCISDK\IOP\SAMPLES\HELLO\PCI480.480\DIAB-ELF\ObjRomD\HelloRomD.Elf [ok
]


ROM image file "PCI480.480\Diab-Elf\HelloRomD.bin" built successfully.


F:\MyDocs\Sdk\PciSdk\Iop\Samples\Hello>
```

5.8 Local-side Debugging Tools and Techniques

When working with local-side environments, it is helpful to use many techniques for debugging code. The greater the developer's knowledge of debug techniques, the easier and faster projects will be completed. The list below provides some tips for debug of local code.

- If the budget allows for it, use a debugger tool. These provide the most powerful control of local CPUs by supporting CPU register sets and source-level debug. Additional features, such as FLASH programming/code download and memory tests, are typically included as well. Note that debugger tools are not required, but their advantages typically justify the cost.

- If a debugger tool is used, an assembly code function to “halt” the CPU can be useful. Then, the debugger tool can be used to increment the instruction pointer and view parameters. For example:

  In assembly code:

  ```
      .global MyPause
      MyPause:
          b    MyPause
  ```

  In C code:

  ```
          MyPause(0x14);
  ```

- When building local code, build with the DEBUG version to avoid optimizations and to include symbol information for debuggers.
- When writing applications, it is helpful to first test with RAM applications, rather than ROM applications, which may corrupt the FLASH.
- When writing boot code, some common techniques for indicating achievement of successful code points are to blink an LED, set an LED counter to a unique value, or write a unique value to a specific location in memory. With PLX chips, values can be written to mailbox registers, then verified with PLXMon.
- It is sometimes helpful to force code to “wait” until the developer is ready. This can be accomplished by writing code to constantly read a specific location in memory or a PLX chip’s mailbox register until it is equal to a unique value. For example:

  In C code:

  ```
          while ( *(volatile U32*)0x100000) != 0x12345678 );
  ```

6 Real Time Operating System Support

***Note**: RTOS support is included only in the SDK-PRO and not in the SDK-LITE. The RTOS support provided is completely independent of the remainder of the SDK. Rebuilding an RTOS component will not reference PLX SDK files in any other directory.*

The SDK-PRO contains additional support for RTOS environments. This section describes the current support provided by PLX. Note that the software mentioned here can be used as a reference or a starting point in designs, which are not directly supported. For example, Linux host-side support for the IOP 480 can be achieved with a minimal porting effort of the Linux 9054 driver.

6.1 VxWorks BSP Support

The PLX SDK includes VxWorks local-side BSPs for the PCI 9054-860RDK and the IOP 480RDK. All VxWorks related files and directories are located in <*Sdk_Install_Dir*>\Rtos\VxWorks. Documentation for the BSPs is not included this manual. It is provided as a link to an HTML format file in the PLX SDK group in the *Start* menu. Please refer to this documentation for details.

For those using non-Windows environments, the VxWorks HTML files can be found in <*Sdk_Install_Dir*>*\Documentation*.

6.2 pSOS BSP Support

The PLX SDK includes pSOS local-side BSPs for the PCI 9054-860RDK and the IOP 480RDK. All pSOS related files and directories are located in <*Sdk_Install_Dir*>\Rtos\pSOS. Documentation for the BSPs is not included this manual. It is provided as a link to an HTML format file in the PLX SDK group in the *Start* menu. Please refer to this documentation for details.

For those using non-Windows environments, the pSOS HTML files can be found in <*Sdk_Install_Dir*>*\Documentation*.

6.3 Linux Host-Side Support

The PLX SDK includes Linux Host-side driver support for the PCI 9054-860RDK. All Linux related files and directories are located in <*Sdk_Install_Dir*>\Rtos\Linux. Documentation for the Linux support is not included this manual. It is provided as a link to an HTML format file in the PLX SDK group in the *Start* menu. Please refer to this documentation for details.

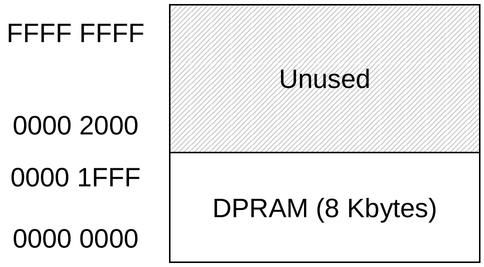
For those using non-Windows environments, the Linux HTML file can be found in <*Sdk_Install_Dir*>*\Documentation*.

7 RDK Software Quick Reference

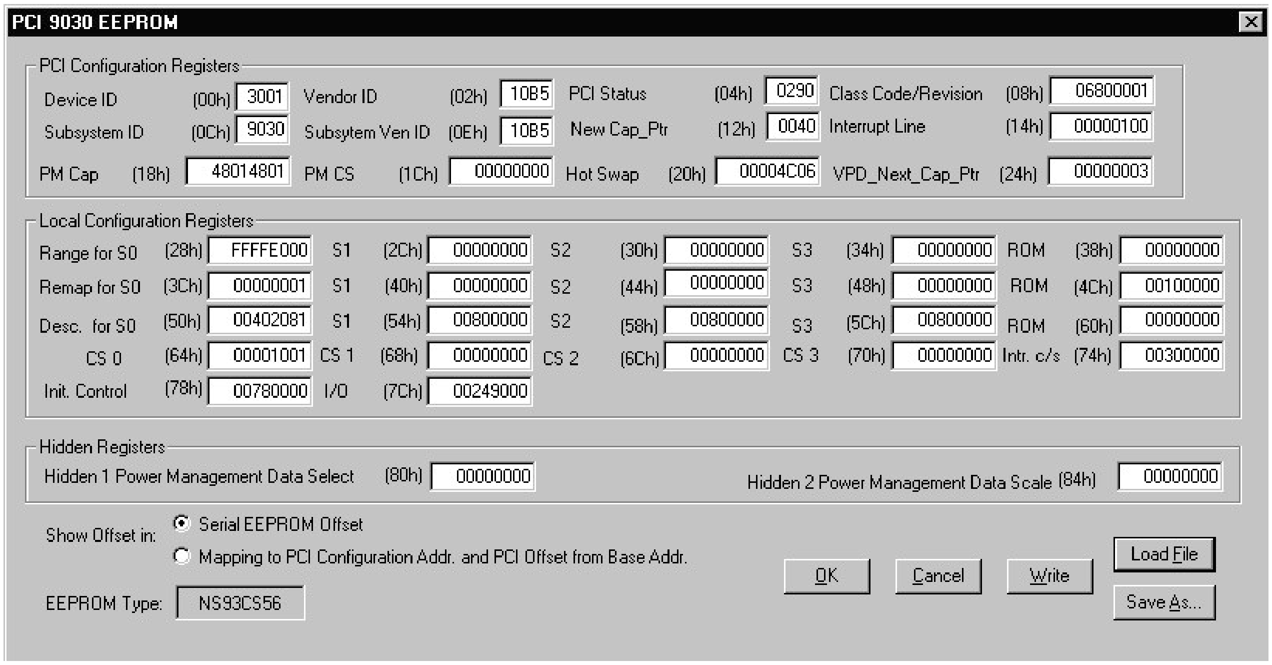
The information below should be used as a guide to the PLX SDK software requirements for PLX RDK boards.

7.1 PCI and Compact PCI 9030 RDK-LITE

The following below details the 9030 RDK-LITE memory map.



The following EEPROM values are used in the 9030 RDK-LITE. The Compact PCI EEPROM values are identical, except for the Device ID, which is 30C1h versus 3001h of the PCI board.



PCI 9030 EEPROM

PCI Configuration Registers

Device ID	(00h)	3001	Vendor ID	(02h)	10B5	PCI Status	(04h)	0290	Class Code/Revision	(08h)	06800001
Subsystem ID	(0Ch)	9030	Subsytem Ven ID	(0Eh)	10B5	New Cap_Ptr	(12h)	0040	Interrupt Line	(14h)	00000100
PM Cap	(18h)	48014801	PM CS	(1Ch)	00000000	Hot Swap	(20h)	00004C06	VPD_Next_Cap_Ptr	(24h)	00000003

Local Configuration Registers

Range for S0	(28h)	FFFFE000	S1	(2Ch)	00000000	S2	(30h)	00000000	S3	(34h)	00000000	ROM	(38h)	00000000
Remap for S0	(3Ch)	00000001	S1	(40h)	00000000	S2	(44h)	00000000	S3	(48h)	00000000	ROM	(4Ch)	00100000
Desc. for S0	(50h)	00402081	S1	(54h)	00800000	S2	(58h)	00800000	S3	(5Ch)	00800000	ROM	(60h)	00000000
CS 0	(64h)	00001001	CS 1	(68h)	00000000	CS 2	(6Ch)	00000000	CS 3	(70h)	00000000	Intr. c/s	(74h)	00300000
Init. Control	(78h)	00780000	I/O	(7Ch)	00249000									

Hidden Registers

Hidden 1 Power Management Data Select	(80h)	00000000	Hidden 2 Power Management Data Scale (84h)	00000000

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

EEPROM Type: NS93CS56

7.2 IOP 480RDK

The versatility of IOP 480 memory controller allows for dynamic modification of the memory map by the CPU. PLX boot code does, in fact, re-map the IOP 480 RDK memory map in the BSP boot code. The primary reason for this is to provide a large segment for the Direct Master memory region. Since this region cannot conflict with any other local bus devices, a large unused portion is left unused to allow for a large Direct Master memory space.

The following memory maps provide the IOP 480 RDK default map and the memory map after the boot code has re-configured the memory controller.

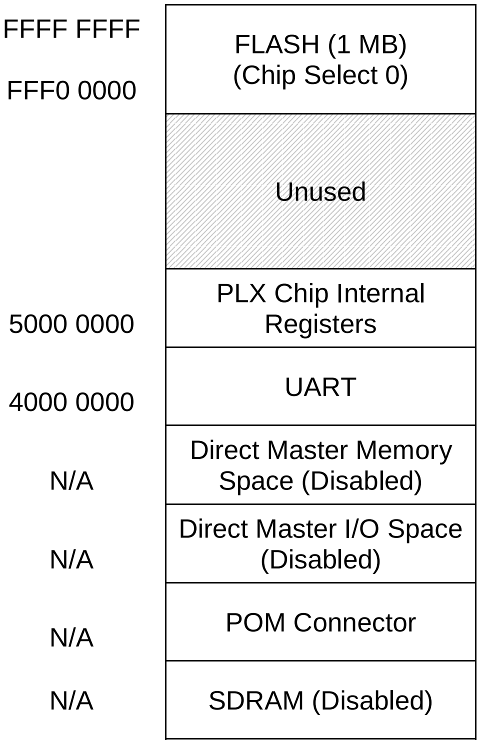


Figure 7-1 IOP 480 RDK Default Memory Map (at Boot-up)

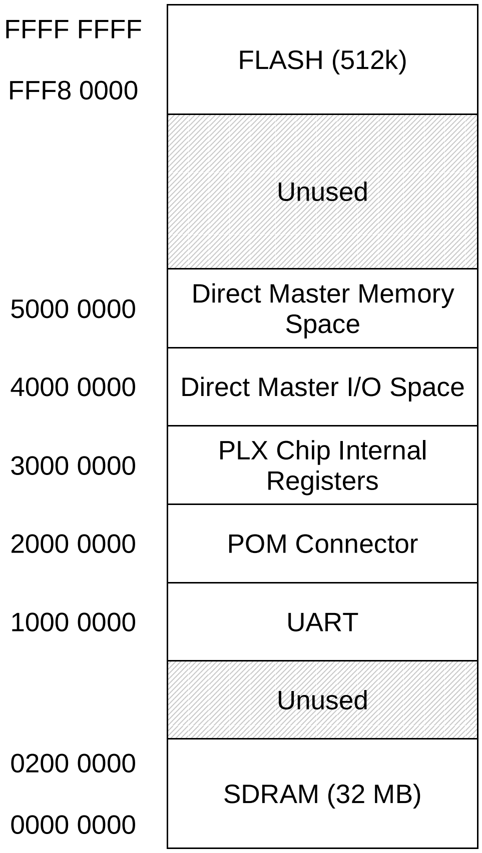
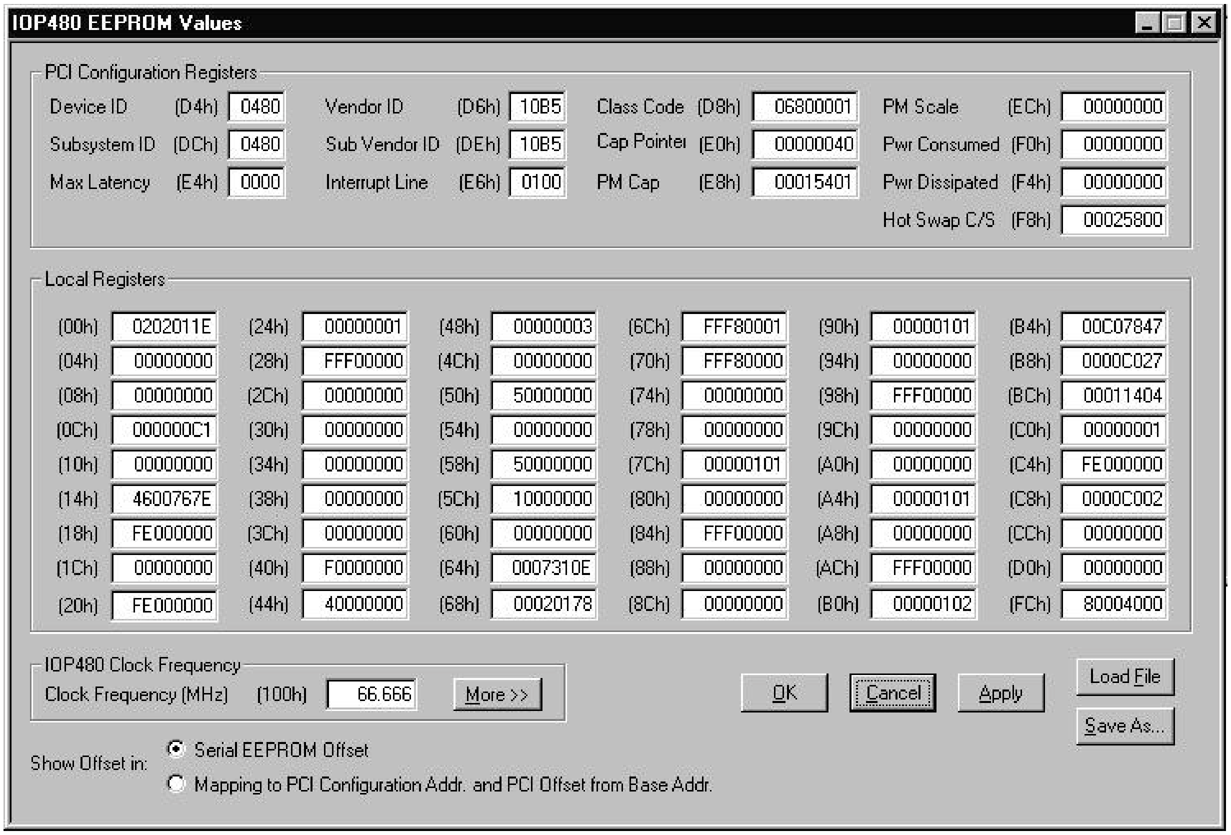


Figure 7-2 IOP 480 RDK Memory Map After Boot code

IOP 480 RDK additional information:

FLASH Type:	ATMEL AT49LV040
FLASH Program Offset:	0x0006 0000 (assuming 128k image size)
EEPROM Type:	NM93CS66



PCI Configuration Registers

Device ID	(D4h)	0480	Vendor ID	(D6h)	10B5	Class Code	(D8h)	06800001	PM Scale	(ECh)	00000000
Subsystem ID	(DCh)	0480	Sub Vendor ID	(DEh)	10B5	Cap Pointer	(E0h)	00000040	Pwr Consumed	(F0h)	00000000
Max Latency	(E4h)	0000	Interrupt Line	(E6h)	0100	PM Cap	(E8h)	00015401	Pwr Dissipated	(F4h)	00000000
									Hot Swap C/S	(F8h)	00025800

Local Registers

(00h)	0202011E	(24h)	00000001	(48h)	00000003	(6Ch)	FFF80001	(90h)	00000101	(B4h)	00C07847
(04h)	00000000	(28h)	FFF00000	(4Ch)	00000000	(70h)	FFF80000	(94h)	00000000	(B8h)	0000C027
(08h)	00000000	(2Ch)	00000000	(50h)	50000000	(74h)	00000000	(98h)	FFF00000	(BCh)	00011404
(0Ch)	000000C1	(30h)	00000000	(54h)	00000000	(78h)	00000000	(9Ch)	00000000	(C0h)	00000001
(10h)	00000000	(34h)	00000000	(58h)	50000000	(7Ch)	00000101	(A0h)	00000000	(C4h)	FE000000
(14h)	4600767E	(38h)	00000000	(5Ch)	10000000	(80h)	00000000	(A4h)	00000101	(C8h)	0000C002
(18h)	FE000000	(3Ch)	00000000	(60h)	00000000	(84h)	FFF00000	(A8h)	00000000	(CCh)	00000000
(1Ch)	00000000	(40h)	F0000000	(64h)	0007310E	(88h)	00000000	(ACh)	FFF00000	(D0h)	00000000
(20h)	FE000000	(44h)	40000000	(68h)	00020178	(8Ch)	00000000	(B0h)	00000102	(FCh)	80004000

IOP480 Clock Frequency

Clock Frequency (MHz) (100h) 66.666

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

Figure 7-3 IOP 480RDK Configuration EEPROM Values

***Note**: The IOP 480 EEPROM screen includes a value for "Clock Frequency". Since the IOP 480 CPU can be run at various frequencies, this affect initialization of the Baud-Rate Divisor for the UART. To provide a configurable solution, the PLX IOP 480 BSP software will read offset 0x100 in the EEPROM and use the value as the local bus clock frequency, if the number is within a valid range. This number will be used in the Baud-Rate Divisor calculation for the UART.*

The Clock Frequency is not a value automatically loaded by the IOP 480, it is only used by PLX boot code. Refer to the IOP 480 BSP or the "More >>" button in the EEPROM screen for additional information.

7.3 PCI 9054RDK-860

Figure 7-1 provides the memory map for the 9054-860 RDK boards. Note that the memory map is identical for the PCI and Compact PCI boards except that the Compact PCI version contains Synchronous Burst SRAM (SBSRAM), which is not found on the PCI board.

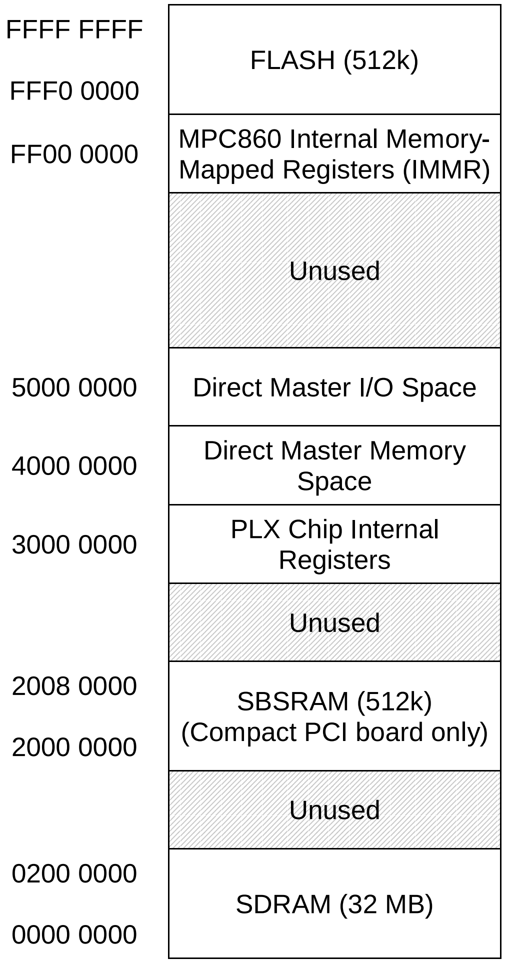
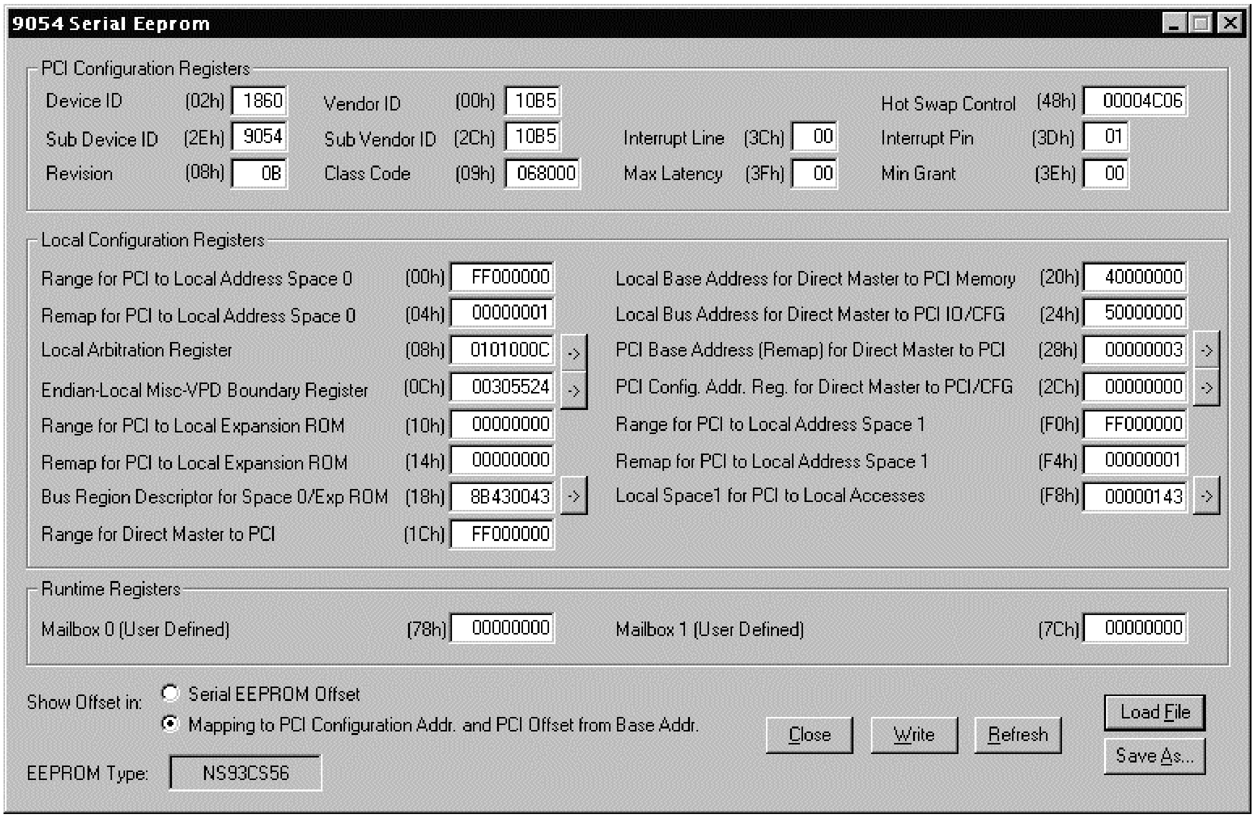


Figure 7-1 9054-860 RDK Memory Map

9054-860 RDK additional information:

FLASH Type:	ATMEL AT49LV040 or AMD 29LV040B
FLASH Program Offset:	0x0000 0000
EEPROM Type:	NM93CS56



9054 Serial Eeprom

PCI Configuration Registers

Device ID (02h) 1860

Vendor ID (00h) 10B5

Hot Swap Control (48h) 00004C06

Sub Device ID (2Eh) 9054

Sub Vendor ID (2Ch) 10B5

Interrupt Line (3Ch) 00

Interrupt Pin (3Dh) 01

Revision (08h) 0B

Class Code (09h) 068000

Max Latency (3Fh) 00

Min Grant (3Eh) 00

Local Configuration Registers

Range for PCI to Local Address Space 0 (00h) FF000000

Remap for PCI to Local Address Space 0 (04h) 00000001

Local Arbitration Register (08h) 0101000C

Endian-Local Misc-VPD Boundary Register (0Ch) 00305524

Range for PCI to Local Expansion ROM (10h) 00000000

Remap for PCI to Local Expansion ROM (14h) 00000000

Bus Region Descriptor for Space 0/Exp ROM (18h) 8B430043

Range for Direct Master to PCI (1Ch) FF000000

Local Base Address for Direct Master to PCI Memory (20h) 40000000

Local Bus Address for Direct Master to PCI IO/CFG (24h) 50000000

PCI Base Address (Remap) for Direct Master to PCI (28h) 00000003

PCI Config. Addr. Reg. for Direct Master to PCI/CFG (2Ch) 00000000

Range for PCI to Local Address Space 1 (F0h) FF000000

Remap for PCI to Local Address Space 1 (F4h) 00000001

Local Space1 for PCI to Local Accesses (F8h) 00000143

Runtime Registers

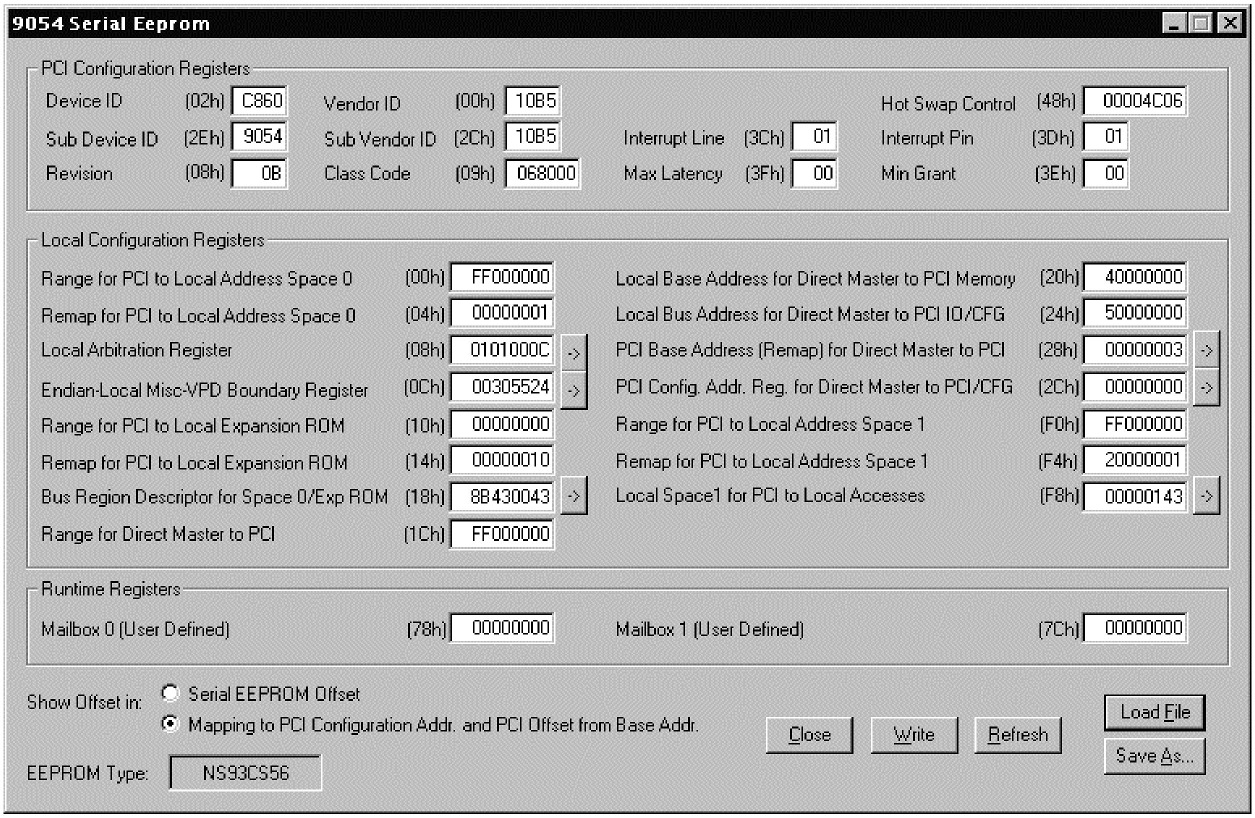
Mailbox 0 (User Defined) (78h) 00000000

Mailbox 1 (User Defined) (7Ch) 00000000

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

EEPROM Type: NS93CS56

Figure 7-2 PCI 9054RDK-860 Configuration EEPROM Values



PCI Configuration Registers

Device ID (02h) C860
Vendor ID (00h) 10B5
Hot Swap Control (48h) 00004C06
Sub Device ID (2Eh) 9054
Sub Vendor ID (2Ch) 10B5
Interrupt Line (3Ch) 01
Interrupt Pin (3Dh) 01
Revision (08h) 0B
Class Code (09h) 068000
Max Latency (3Fh) 00
Min Grant (3Eh) 00

Local Configuration Registers

Range for PCI to Local Address Space 0 (00h) FF000000
Remap for PCI to Local Address Space 0 (04h) 00000001
Local Arbitration Register (08h) 0101000C ->
Endian-Local Misc-VPD Boundary Register (0Ch) 00305524 ->
Range for PCI to Local Expansion ROM (10h) 00000000
Remap for PCI to Local Expansion ROM (14h) 00000010
Bus Region Descriptor for Space 0/Exp ROM (18h) 8B430043 ->
Range for Direct Master to PCI (1Ch) FF000000
Local Base Address for Direct Master to PCI Memory (20h) 40000000
Local Bus Address for Direct Master to PCI IO/CFG (24h) 50000000
PCI Base Address (Remap) for Direct Master to PCI (28h) 00000003 ->
PCI Config. Addr. Reg. for Direct Master to PCI/CFG (2Ch) 00000000 ->
Range for PCI to Local Address Space 1 (F0h) FF000000
Remap for PCI to Local Address Space 1 (F4h) 20000001
Local Space1 for PCI to Local Accesses (F8h) 00000143 ->

Runtime Registers

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

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

EEPROM Type: NS93CS56

Figure 7-3 Compact PCI 9054RDK-860 Configuration EEPROM Values

Appendix A. References

1. *Windows 2000 DDK Documentation*, Microsoft Corporation, August 2000.
2. *Windows NT 4.0 DDK Documentation*, Microsoft Corporation, July 1997.
3. *Programming the Microsoft Windows Driver Model,* Walter Oney, 1999.
4. *PLX SDK Programmer's Reference Manual, Version 3.2*, PLX Technology, Inc., January 2001
5. *PLXMon User's Manual, Version 3.2*, PLX Technology, Inc., January 2001

Appendix B. BEM Serial Protocol Details

This section is provided as a reference for the PLX proprietary protocol used between the BEM and PLXMon.

Notes:

- *All BEM commands are case sensitive except hexadecimal values.*
- *Insignificant leading zeros in the hex number strings are not sent for efficiency.*
- *When multiple consecutive hexadecimal values are sent, an ASCII space character separates them.*

B.1 BEM Command Message Packet

BEM Command

The BEM waits for messages sent from PLXMon. These are command-based messages sent in the format below.

2 Bytes	1 Byte	*n* Bytes	1 Byte
Message Header (~p)	BEM Command	Command-specific Data	End-Message (<CR>)

Table 7-1 BEM Commands

BEM Command	Value	Definition
RESET_CPU	!	Reset the IOP board
HOST_INQUIRY	@	Query the information for the board
READ_BIT_8	g	Read a 8-bit data from local memory
READ_BIT_16	h	Read a 16-bit data from local memory
READ_BIT_32	i	Read a 32-bit data from local memory
READ_BIT_64	j	Read a 64-bit data from local memory
BLOCK_READ_BIT_8	m	Read multiple 8-bit data from local memory
BLOCK_READ_BIT_16	n	Read multiple 16-bit data from local memory
BLOCK_READ_BIT_32	o	Read multiple 32-bit data from local memory
BLOCK_READ_BIT_64	p	Read multiple 64-bit data from local memory
WRITE_BIT_8	G	Write a 8-bit data to local memory
WRITE_BIT_16	H	Write a 16-bit data to local memory
WRITE_BIT_32	I	Write a 32-bit data to local memory
WRITE_BIT_64	J	Write a 64-bit data to local memory
BLOCK_WRITE_BIT_8	M	Write multiple 8-bit data to local memory
BLOCK_WRITE_BIT_16	N	Write multiple 16-bit data to local memory
BLOCK_WRITE_BIT_32	O	Write multiple 32-bit data to local memory
BLOCK_WRITE_BIT_64	P	Write multiple 64-bit data to local memory
EEPROM_READ	k	Read data from EEPROM and place into the EEPROM data buffer

BEM Command	Value	Definition
EEPROM_WRITE	K	Write data to the EEPROM from the EEPROM data buffer
READ_480CPU_REG	z	Read from IOP 480 CPU internal register
WRITE_480CPU_REG	Z	Write to IOP 480 CPU internal register

B.2 BEM Reply Message Packet

For some BEM commands, a reply is required either to provide data or status information. The BEM reply packet is provided below.

1 Byte	*n* Bytes	1 Byte
Message Header (01h)	Command-specific Data or status	End-Message (<CR>)

In cases where status information is required, the following is used:

BEM Status	Value	Description
REPLY_SUCCESS	!	The operation was successful
REPLY_ERROR	@	An error occurred

B.3 BEM Commands Description

This section will provide details of the data sent and received for the various BEM commands.

Reset the Board

***Note**: The reset is implemented as a software reset. In other words, it is not much more than some cleanup and a jump to the local CPU boot vector. A crashed local CPU may not respond to this command.*

Command Message:

2 Bytes	1 Byte	1 Byte
Message Header	RESET_CPU	End-Message

Reply Message:

None

Query for Board Information

This command is the first one sent by PLXMon when attempting to establish a connection. It is used to retrieve information about the device. PLXMon uses this information throughout the Serial Mode session for register access, EEPROM access, etc.

Command Message:

2 Bytes	1 Byte	1 Byte
Message Header	HOST_INQUIRY	End-Message

Reply Message:

1 Byte	*n* Bytes	1 Byte
Reply Header	<Platform_Information>	End-Message

The information sent by the BEM will provide information about the platform connected. The following is the structure shared by the local-side and PLXMon to contain information about the local device.

```
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;               // Reserved for future use
    U32 FlashAddressLow;        // Lower 32-bits of FLASH base address
    U32 FlashAddressHigh;       // Upper 32-bits of FLASH base address
    U32 BufferAddressLow;       // Local EEPROM buffer
    U32 BufferAddressHigh;
    U32 BufferSize;             // Permanent Buffer size
    U32 FlashBufferAddressLow;  // Temp Buffer for FLASH programming
    U32 FlashBufferAddressHigh;
    U32 FlashBufferSize;        // Temp FLASH buffer size
} PLATFORM_PARAMS;
```

The Platform Information is sent by the BEM as follows:

- **Version Information**

  This provides the SDK version and platform addressing and data capability. It is provided first in case the protocol changes in the future. The version can be used to determine how to parse the remainder of the data. The version data is formatted as follows:

31 24	**23 16**	**15 8**	**7 6**	**5 0**
Version Major	Version Minor	Version Revision	Platform*	Reserved

  Where:
  Version Major = The local SDK major version number
  Version Minor = The local SDK minor version number
  Version Revision = The local SDK revision version number
  Platform = The addressing and data access capability, as follows:
  Bit 6: 0 = 32-bit addressing
  1 = 64-bit addressing
  Bit 7: 0 = 32-bit data size
  1 = 64-bit data size

- **PlxAddressLow**

  The lower 32-bits of the PLX Chip's base address

- **PlxAddressHigh**

  The upper 32-bits of the PLX Chip's base address

- **PlxChipType**

  A hex representation of the PLX chip type (e.g. 480 = IOP 480, 9054 = 9054, etc.)

- **Capabilities**
  This provides the capabilities of the platform. It reports the supported BEM features of the platform. The capability is formatted into 3-bit fields as follows:

15 13	**12 10**	**9 7**	**6 0**
Read/Write to Memory	Local FLASH Programming	Local EEPROM Access	Reserved

  Where:
  Bit 0 : 0 – *Not Supported*
  1 – *Supported*
  Bit 2:1 : *Reserved*

- **EEPROMType**
  An ASCII string specifying the EEPROM type *PlxType.s* contains the EEPROM names. Some examples are *Eeprom93CS56* and *Eeprom93CS66*.

- **FLASHType**
  An ASCII string specifying the FLASH type *PlxType.s* contains the FLASH names. Some examples are *AT49LV040* and *AM29F040*.

- **FlashAddressLow**
  The lower 32-bits of the FLASH address

- **FlashAddressHigh**
  The upper 32-bits of the FLASH address

- **BufferAddressLow**
  The lower 32-bits of the local EEPROM buffer

- **BufferAddressHigh**
  The upper 32-bits of the local EEPROM buffer

- **BufferSize**
  The size of the local EEPROM buffer

- **FlashBufferAddressLow**
  The lower 32-bits of the FLASH buffer address

- **FlashBufferAddressHigh**
  The upper 32-bits of the FLASH buffer address

- **FlashBufferSize**
  The size of the FLASH buffer

For example, the following serial data is returned as the Platform Information and interpreted below:

```
3020000 20000000 0 480 2480 Eeprom93CS66 AT49LV040 FFF80000 0 10000 0 100 1F80000 0 80000
```

Version	= 3.20, 32-bit addressing, 32-bit data support
PLX chip base address	= 0x00000000_20000000 (64-bit)
PLX chip type	= IOP 480
Capabilities (0x2480)	= Mem R/W, FLASH programming, and EEPROM access supported
EEPROM Type	= National 93CS66
FLASH Type	= Atmel 49LV040
Flash Base Address	= 0x00000000_FFF80000 (64-bit)
EEPROM Buffer Address	= 0x00000000_00010000 (64-bit)

EEPROM Buffer Size	= 0x100 (256 bytes)
FLASH Buffer Address	= 0x00000000_01F80000 (64-bit)
FLASH Buffer Size	= 0x80000 (512k)

Read Single Unit from Memory

Command Message:

2 Bytes	1 Byte	*n* Bytes	1 Byte
Message Header	READ_BIT_XX	<Address>	End-Message

Where:
XX = The memory access type (8, 16, 32, 64)
Address = Local bus address to read from

Reply Message:

1 Byte	*n* Bytes	1 Byte
Reply Header	Data read	End-Message

Read Multiple Units from Memory

Command Message:

2 Bytes	1 Byte	*n* Bytes	1 Byte
Message Header	BLOCK_READ_BIT_XX	<Address><Space><Count>	End-Message

Where:
XX = The memory access type (8, 16, 32, 64)
Address = Local bus address to start reading from
Count = The number of units to read, (not number of bytes)

Reply Message:

1 Byte	*n* Bytes	1 Byte
Reply Header	<Data><Sp><Data>….<Sp><Data>	End-Message

Where:
Data = The data read from memory. Multiple data is separated by spaces (<Sp>).

Write Single Unit to Memory

Command Message:

2 Bytes	1 Byte	*n* Bytes	1 Byte
Message Header	WRITE_BIT_XX	<Address><Space><Data>	End-Message

Where:
XX = The memory access type (8, 16, 32, 64)
Address = Local bus address to write to
Data = The data to write

Reply Message:

None

Write Multiple Units to Memory

Command Message:

2 Bytes	1 Byte	*n* Bytes	1 Byte
Message Header	BLOCK_WRITE_BIT_XX	<Address><Sp><Data>…<Sp><Data>	End-Message

Where:

XX	= The memory access type (8, 16, 32, 64)
Address	= Local bus address to write to
Data	= The data to write, separated by spaces (<Sp>)

Reply Message:

None

The BEM will continue to write data to memory until an end-of-message is received or an error occurs, such as invalid data, e.g., 5 characters were sent for a 16-bit data write.

Read from the EEPROM

***Note**: EEPROM access must be supported by the local-side. This is reported in the capabilities of the Platform Information.*

Command Message:

2 Bytes	1 Byte	*n* Bytes	1 Byte
Message Header	EEPROM_READ	<Size>	End-Message

Where:

Size	= The number of bytes to read from EEPROM (must be a multiple of 4)

Reply Message:

1 Byte	1 Byte	1 Byte
Reply Header	<Status>	End-Message

Where:

Status	= REPLY_SUCCESS or REPLY_ERROR

If this command returns successfully, the EEPROM data will be stored in the local EEPROM buffer (<BufferAddressHigh_BufferAddressLow>). The memory read command can then be used to retrieve the data from the EEPROM buffer.

Write to the EEPROM

***Note**: EEPROM access must be supported by the local-side. This is reported in the capabilities of the Platform Information.*

Command Message:

2 Bytes	1 Byte	*n* Bytes	1 Byte
Message Header	EEPROM_WRITE	<Size>	End-Message

Where:

Size	= The number of bytes to write to EEPROM (must be a multiple of 4)

Before this command is issued, the EEPROM data must already be in the local EEPROM buffer (<BufferAddressHigh_BufferAddressLow>). The memory write command can be used to place the data into the EEPROM buffer.

Reply Message:

1 Byte	1 Byte	1 Byte
Reply Header	<Status>	End-Message

Where:
Status = REPLY_SUCCESS or REPLY_ERROR

FLASH Programming

***Note**: FLASH programming must be supported by the local-side. This is reported in the capabilities of the Platform Information.*

At this time, the BEM itself does not support FLASH programming directly. Instead a protocol is implemented which uses the Mailbox registers and a file transfer protocol, such as X-Modem, to transfer the FLASH image and program it. The only BEM commands involved are memory read and write and board reset. The protocol is documented in *PlxProtocol.h* for those interested.

Read IOP 480 CPU Core Register

***Note**: This command is not recommended for use. It is provided for reference purposes only. The CPU core registers are constantly used by local software and change frequently. The registers may even change before this command has completed.*

Command Message:

2 Bytes	1 Byte	1 Byte	1 Byte
Message Header	READ_480CPU_REG	<RegNum>	End-Message

Where:
RegNum = The index number of the register to read.

Reply Message:

1 Byte	*n* Bytes	1 Byte
Reply Header	Data read	End-Message

Write to IOP 480 CPU Registers

***Note**: This command is not recommended for use and may result in local CPU crashes. It is provided for reference purposes only. The CPU core registers are constantly used by local software and change frequently. Modification of these registers should only be performed with a debugger tool.*

Command Message:

2 Bytes	1 Byte	1 Byte	1 Byte
Message Header	WRITE_480CPU_REG	<RegNum><Sp><Data>	End-Message

Where:
RegNum = The index number of the register to write.
Data = The data to write to the register

Reply Message:

None