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 © 2006 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 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 MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. PLX makes no guarantee or representations regarding the use of, or the results based on the use of the software and documentation in terms of correctness, or otherwise; and that you rely on the software, documentation, and results solely at your own risk. In no event shall PLX be liable for any loss of use, loss of business, loss of profits, incidental, special or, consequential damages of any kind. In no event shall PLX’s total liability exceed the sum paid to PLX for the product licensed here under.

PLX Copyright Message Guidelines

The following copyright message must appear, as explained in the following requirements, in all software products generated and distributed, which use the PLX API libraries:

“Copyright © 2006 PLX Technology, Inc.”

Requirements:

- Arial font
- Font size 10 (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

1 General Information ........................................................................................................... 1-1
   1.1 About This Manual .................................................................................................................... 1-1
   1.2 PCI SDK Features ..................................................................................................................... 1-1
   1.3 Where To Go From Here .......................................................................................................... 1-1
   1.4 Terminology ............................................................................................................................... 1-1
   1.5 Customer Support ..................................................................................................................... 1-2

2 Getting Started ................................................................................................................... 2-1
   2.1 Development Tools ................................................................................................................... 2-1
   2.2 PCI SDK Version Compatibility ................................................................................................. 2-1
   2.3 Installing the SDK ...................................................................................................................... 2-2
      2.3.1 Installation in Linux/Unix and Other Non-Windows Environments............................... 2-2
      2.3.2 Installation in a Microsoft Windows Environment ......................................................... 2-2
      2.3.3 Removing Previous Versions of the PCI SDK ............................................................. 2-3
   2.4 Installation of PLX Device Drivers ............................................................................................. 2-3
      2.4.1 WDM Device Driver Installation for Windows 98/Me/2000/XP .................................... 2-3
         2.4.1.1 WDM Driver Installation for PLX RDKs ............................................................... 2-4
         2.4.1.2 WDM Driver Installation for Custom Designs with PLX devices ......................... 2-5
            2.4.1.2.1 Modifying the PLX INF File ............................................................................... 2-5
            2.4.1.2.2 Forcing a Manual Installation ............................................................................ 2-6
         2.4.1.3 PLX WDM Drivers and Windows Logo Testing ................................................... 2-13
      2.4.2 Windows NT Device Driver Installation ........................................................................ 2-14
         2.4.2.1 Windows NT Driver Installation for Custom Designs with PLX devices............. 2-14
         2.4.2.2 Starting and Stopping Windows NT Drivers ............................................................. 2-15
      2.4.3 Modifying PLX Driver Options in the Registry ............................................................ 2-16
         2.4.3.1 PLX Driver Options Wizard ............................................................................... 2-18
         2.4.3.2 System Page-Table Entries (PTEs) in Windows NT/2000/XP .............................. 2-19
   2.5 Support for the PLX PCI 6000 Series ..................................................................................... 2-20
      2.5.1 PLX PCI 6000 in Non-Transparent Mode ........................................................................ 2-20
      2.5.2 PLX PCI 6000 in Transparent Mode ................................................................................. 2-20
         2.5.2.1 Installation of PlxPci Service ............................................................................. 2-20
         2.5.2.2 Using the PLX PCI Service with a Custom (OEM) ID Device ............................ 2-20
         2.5.2.3 Starting and Stopping the PlxPci Service .............................................................. 2-21
            2.5.2.3.1 Windows 2000/XP .............................................................................................. 2-21
            2.5.2.3.2 Windows NT .................................................................................................... 2-23
            2.5.2.3.3 Windows 98/Me VxD ...................................................................................... 2-24
List of Figures

Figure 2-1 Windows Logo Testing Driver Warning ................................................................. 2-14
Figure 2-2 The PLX Driver Option Wizard .......................................................................... 2-15
Figure 2-3 The NT Devices Dialog ...................................................................................... 2-16
Figure 2-4 PLX Device Driver Registry Information for Windows NT/2000/XP ...................... 2-16
Figure 2-5 PLX Device Driver Registry Information for Windows 98/Me .............................. 2-17
Figure 2-6 PLX Driver Options Wizard ............................................................................... 2-19
Figure 2-7 PLX PCI Service VxD Configured for Dynamic Loading .................................... 2-24
Figure 2-8 PLX PCI Service VxD Configured for Static Loading ......................................... 2-24
Figure 3-1 The PCI SDK Software Architecture ................................................................. 3-1
Figure 4-1 The PLX Driver Layout ...................................................................................... 4-2
Figure 4-2 PLX Windows Driver Directory Structure .......................................................... 4-3
Figure 4-3 Visual C/C++ New Project Dialog .................................................................... 4-10
Figure 4-4 Visual C/C++ Include Files Directory ............................................................... 4-11
Figure 4-5 Typical Visual C/C++ Project ............................................................................ 4-12
Figure 5-1 Local Software Architecture ............................................................................ 5-1
Figure 5-2 PLX BSP Program Flow .................................................................................... 5-2
Figure 5-3 PLX BEM Serial Data Parsing .......................................................................... 5-4
Figure 5-4 Typical Local Memory Usage ............................................................................ 5-5
Figure 5-5 Local-side Directory Structure ........................................................................ 5-6
Figure 5-6 Make File Processing Flow ............................................................................... 5-7
Figure 7-1 PLXMon Communications Modes ................................................................... 7-1
Figure 7-2: PLXMon in PCI Mode .................................................................................. 7-2
Figure 7-3: EEPROM Edit Mode .................................................................................... 7-3
Figure 7-4: PLXMon in Serial Mode ............................................................................... 7-4
Figure 7-5: PLXMon Toolbar .......................................................................................... 7-4
Figure 7-6: Typical PLXMon Register Dialog ................................................................. 7-6
Figure 7-7: Typical EEPROM Dialog .............................................................................. 7-7
Figure 7-8: Memory Access Dialog ................................................................................ 7-8
Figure 7-9: Memory Fill Dialog ....................................................................................... 7-8
Figure 7-10: Device Properties Needed Notification ......................................................... 7-9
Figure 7-11: Device Properties ......................................................................................... 7-9
Figure 7-12: Download Image Dialog ............................................................................... 7-10
Figure 7-13 Manually Setting the PLX Chip Type .............................................................. 7-11
Figure 7-14 Completed PLX Chip Type Override ............................................................. 7-12
Figure 7-15  Performance Measure Dialog ............................................................................................. 7-13
Figure 7-16 Sample DMA Performance Test .......................................................................................... 7-14
Figure 7-17 Sample Direct Slave Performance Test .............................................................................. 7-16
Figure 7-18: Command-line Interface ................................................................................................. 7-17
Figure 7-19: PLXMon Variables ........................................................................................................... 7-18
1 General Information

1.1 About This Manual

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

1.2 PCI SDK Features

The SDK contains software for various host environments 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, Me, and XP;
- PLXMon, a Windows Graphical User Interface (GUI) application used to configure and modify PLX PCI devices and download applications to the PCI device.
- PCI Host API and drivers compatible with Linux.

The SDK 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 rich Local API, with support for a variety of CPUs and PLX PCI chips
- The PLX Back-End Monitor (BEM), which is the target server supporting serial mode communications with PLXMon
- Generic Board Support Packages (BSP) for PLX RDK boards, which are re-usable and customizable building blocks in customer designs, or for reference purposes

1.3 Where To Go From Here

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

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

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

Chapter 4, PCI Host Software, describes the PLX Host-side software components and how to utilize them.

Chapter 5, Local Software, describes the PLX Local-side software components and how to utilize them.

1.4 Terminology

- References to Windows NT assume Windows NT 4.0 and may be denoted as WinNT or NT.
- References to Windows 98 may be denoted as Win98.
- References to Windows 2000 may be denoted as Win2000 or Win2k.
- References to Windows Me may be denoted as WinMe.
- References to Windows XP may be denoted as WinXP or XP.
- References to Visual C/C++ or Visual C++ refer to Microsoft Visual C/C++ 6.0.
Section 1
General Information

- 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.

1.5 Customer Support

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

- PLX chip used
- PCI 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/support](http://www.plxtech.com/support)
2 Getting Started

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

Note: The SDK 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).

2.1 Development Tools

Various tools were used to build the software included in the PCI 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 PCI SDK components include:

**Windows Applications and API DLL:**
Microsoft Visual C/C++ 6.0, Service Pack 6

**Windows NT 4.0 Drivers:**
Microsoft Windows NT 4.0 Device Driver Kit (DDK) with Visual C/C++ 6.0 (Microsoft Win32 Platform SDK is typically also required by the NT DDK)

**Windows Driver Model (WDM) Device Drivers for Windows 98/Me/2000/XP:**
Microsoft Windows Device Driver Kit (DDK)

**Windows VxD Drivers Windows 98/Me:**
Microsoft Windows 98 Device Driver Kit (DDK), July 1999

**Linux Applications and API Library:**
Standard Linux distribution, such as RedHat or Fedora, using GCC.

**Linux Device Driver:**
Standard Linux distribution, such as RedHat or Fedora with kernel source installed, using GCC

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

2.2 PCI SDK Version Compatibility

When using the PCI SDK, it is important that all components are of the same version, as follows:

- In Windows & Linux, the PLX device drivers (e.g. .sys files) and the PLX API library (e.g. PlxApi.dll) versions must match. In other words, loading a driver built with SDK 4.1 and running an application, which calls the API library from version 3.5, will result in erratic behavior.

- When building applications, it is important to use the C header files included in the installed PCI SDK version. Applications built with older SDK versions should be re-built.

- Applications built with a PCI SDK version prior to SDK 3.5 are not be compatible with newer SDK versions. In the interests of performance enhancements, many structures and components were modified. If applications were originally designed with SDK version 3.4 or earlier, a small porting effort may be required to adapt code to the latest SDK. Please refer to the Programmer's Reference Manual for details on API differences between SDK versions.

- 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 is sometimes modified in newer SDK versions; therefore, it is recommended that the ROM be programmed with the latest SDK version used.
Section 2
Getting Started

Note: PLX RDKs contain local software versions to match the SDK version at the time of shipment; however, if the PCI 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 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 section of this manual for details on FLASH programming.

2.3 Installing the SDK

2.3.1 Installation in Linux/Unix and Other Non-Windows Environments

The PCI SDK installation package, which was created with InstallShield®, will only execute in a Microsoft Windows environment. In order to use the PCI SDK in non-Windows operating systems, PLX includes the complete set of SDK files on the installation CD-ROM, in the directory `<CD-ROM>\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 can be used as-is with compatible development tools, which are available for various Operating Systems.

2.3.2 Installation in a Microsoft Windows Environment

Minimum System Requirements:

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

- Windows 98/Me/2000/XP or Windows NT 4.0 (Service Pack 3 or above)
- 32MB RAM
- 80MB hard drive space
- Free RS-232 serial port and cable, if serial communications will be used

Note: Before installing the SDK, any previously installed PCI SDK versions should be removed. Installation of multiple SDK versions may result in erratic behavior due to file conflicts. Refer to section 2.3.3 for more details.

To install the PCI 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, Win2k, and WinXP installation, a user with “Administrator” rights must install the SDK.
2.3.3 Removing Previous Versions of the PCI SDK

**Warning:** If any files have been modified in the original PCI 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 PCI 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 PCI 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 PCI SDK package from the item list
5. Click the Add/Remove... button

**Note:** For proper removal in WinNT, Win2k, and WinXP, a user with “Administrator” rights must remove the PCI SDK.

2.4 Installation of PLX Device Drivers

During SDK installation, the installation package will automatically make the necessary registry entries and copy the files needed to load PLX device drivers. The files and registry settings vary between Windows NT 4.0 and newer versions of Windows.

2.4.1 WDM Device Driver Installation for Windows 98/Me/2000/XP

The Windows 98/Me/2000/XP PLX device drivers conform to the Microsoft Windows Driver Model (WDM). These drivers support Plug ‘n’ Play (PnP) and Power Management. Although there are some limitations, WDM drivers can be binary-compatible between different versions of Windows.

Since Windows 98/Me/2000/XP 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 for 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. For example, the device driver for the 9054 is named Pci9054.sys.

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

1. After installing the PCI 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 a suitable driver. If a PLX RDK board is installed proceed to section 2.4.1.1. If a custom board with a PLX device is installed, proceed to the section 2.4.1.2.
2.4.1.1 WDM Driver Installation for PLX RDKs

- Once the Found New Hardware Wizard starts, the following dialog is displayed: Select “Search for a suitable driver for my device”. Click **Next**.

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

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

![Found New Hardware Wizard]

### 2.4.1.2 WDM Driver Installation for Custom Designs with PLX devices

Installing a device with a custom ID can be performed in one of two methods. In general, modifying the PLX INF file is recommended as forcing a manual installation does not always work. The sections below document each method.

#### 2.4.1.2.1 Modifying the PLX INF File

When a new device is plugged into a Plug ‘n’ Play version of Windows, such as 2000 or XP, Windows will prompt the user for driver files. The way Windows determines what files to install is through an INF file.
Section 2
Getting Started

PLX already provides an INF file called *PciSdk.inf*, which contains setup information for all PLX RDKs and all PLX parts with a default ID. The INF may be found in *PciSdk\Win32\Driver\Wdm*, but the install package also installs a copy under `<Windows_Dir>\Inf`.

The recommended method for installing a device where the ID has been changed is to open the PLX INF file and add an entry for the device with a custom ID. The procedure for this is documented inside the INF file itself, which is a simple text file. Follow the instructions to add an entry for the custom ID and then re-install the device. Windows will then automatically detect the device and install the necessary driver files.

2.4.1.2.2 Forcing a Manual Installation

Another method for installing a driver is to attempt to force Windows to load a driver for the custom device. The steps below document the procedure, but beware that this method does not always work properly. Modifying the INF is the recommended method. To force a driver install for a custom board containing a PLX PCI device, perform the steps below. Dialogs from both Windows 98/Me and 2000/XP are provided as a reference.

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

  **Win 98/Me:**

  ![Add New Hardware Wizard](image)

  This wizard searches for new drivers for:

  PCI Bridge

  A device driver is a software program that makes a hardware device work.
Win 2000/XP:

- Choose a manual selection from a driver list, since this is a custom device, and click Next.

Win 98/Me:

- Search for the best driver for your device.
- Display a list of all the drivers in a specific location, so you can select the driver you want.
Win 2000/XP:

- Select Other devices or Show All Devices and click Next.

Win 98/Me:

- Select Other devices or Show All Devices and click Next.
Win 2000/XP:

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

Win 98/Me:

- Select Have Disk... to search for a driver in another location and click Next.
Win 2000/XP:

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

![Image of Found New Hardware Wizard]

![Image of Install From Disk dialog box]

...
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.

Windows will now inform the user of the driver selected. A dialog will appear to warn that the driver is not specifically written for the device and may result in an unstable system. This is expected behavior since the custom ID is not specified in the PLX INF file. Continue to install the driver.
For **Win98/Me**, click **Next** to start the driver installation.

![Add New Hardware Wizard](image1)

Click **Finish** to complete the installation.
• If the device appears under **Other devices**, the installation was successful. Applications that use the PLX Host API, such as 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.

### 2.4.1.3 PLX WDM Drivers and Windows Logo Testing

Windows 2000/XP introduced a new feature called **Driver Signing**, in which drivers are certified by Microsoft Logo testing program and a digital signature added to inform the OS of this.

Since PLX drivers are provided for general use and as a reference for customer development, they are not submitted to Microsoft for certification. As a result, during driver installation, a dialog box will be displayed to inform the user that the driver is not signed. This is default behavior of the OS, but can be overridden by modifying the registry.

This message does *not* indicate that a problem exists with the PLX driver. It is purely an informational message to notify users that the driver does not contain a digital signature, which means Microsoft has not certified it. To complete the driver installation, users should select the "Continue Anyway" button. **Users interested in additional information should refer to the Microsoft documentation on Driver Signing.**
2.4.2 Windows NT Device Driver Installation

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 typically 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. NT drivers are responsible for scanning the PCI buses to find and claim the devices they are responsible for.

The PCI 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>`.

All device drivers are located in `<Sdk_Install_Dir>\Win32\Driver\WinNT\<DeviceName>`. The SDK installation copies the drivers `<Win_Dir>\System32\Drivers`. The naming convention used for the device drivers is: `Pci<DeviceType>.sys`. For example, the device driver for the PLX 9054 device is “Pci9054.sys”.

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 PLX Driver Options Wizard to disable all other drivers, such as PCI9080, PCI9030, and PCI9056. The settings will take effect after reboot.

2.4.2.1 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 claim 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 Driver Options Wizard application for ease of managing the list of IDs. This utility contains the ability to manage the vendor and device IDs of the boards from the SupportedIDs entry for the appropriate device driver. Refer to section 2.4.3.1 for additional details about the PLX Driver Option Wizard.
2.4.2.2 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 the Control Panel and select the Devices icon. Select the driver of choice, click Stop, and then click Start. Modified registry settings will take effect when the driver restarts.

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

The PLX Driver Options Wizard may also be used to start and stop PLX device drivers. Refer to section 2.4.3.1 for more details.

Note: Before stopping the device driver, all applications that have selected a PLX device, should be closed.
2.4.3 Modifying PLX Driver Options in the Registry

All Windows drivers have entries in the Registry, which are required by the OS. Additionally, there may be driver-specific entries, which can be used to customize driver behavior. Some features of PLX drivers are customizable through registry settings and are documented below. The location of the registry settings is dependent upon the OS and the desired driver.

For Windows NT/2000/XP, the registry entry is located in the path specified below. Figure 2-4 demonstrates a typical entry.

```
HKLM\System\CurrentControlSet\Services\<DriverName>
```

In Windows 98/Me, there is no centralized location for global driver options. As a result, PLX drivers are designed to search the location below for driver options. Figure 2-5 demonstrates a typical entry.

```
HKLM\Software\Plx Technology, Inc.\DriverOptions\<DriverName>
```
Section 2
Getting Started

Figure 2-5 PLX Device Driver Registry Information for Windows 98/Me

The registry entries are described in detail below. **Note:** Only advanced users with administrative rights should modify entries in the registry. Please refer to Microsoft’s documentation on modifying the registry.

**Windows NT/2000/XP 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.

**Windows 98/Me required entries (in HKLM\...\Services\Class\Unknown\XXX):**

- **DevLoader**
  Required by the operating system and should not be modified.

- **NTMPDriver**
  Required by the operating system and should not be modified.

- **ErrorLogLevel**
  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 allocation of a larger buffer. The device driver makes a request to the operating system for a buffer with the size indicated by this registry entry. If the request fails, however, usually due to unavailable system resources, the driver will decrement the size and resubmit the request until the buffer allocation succeeds. The API call `PlxPciCommonBufferProperties()` can be used to determine the common buffer information.

- **PhysicalMemoryCacheable**
  This value determines whether physical memory allocated by the driver should be [data] cacheable. Physical memory is the driver-allocated DMA common buffer and any buffers allocated with `PlxPciPhysicalMemoryAllocate()`. Enabling caching typically results in significant
performance improvements when accessing these buffers. By default, this value is 0, or caching disabled. To enable caching, set the value to 1.

**Note:** Care must be taken when enabling caching. Since multiple masters, such as the Host CPU and the DMA engine of the PLX chip, can modify PCI physical buffers independently, corrupt data may result due to cache incoherency issues. For example, if data is in the Host CPU cache and the DMA engine transfers data to physical memory, the cache data and data in memory may not match.

In general, it is assumed that the Host CPU will perform “bus snooping” and automatically update its cache to avoid any cache coherency issues. PLX, however, cannot guarantee this on all architectures. As a result, caching of physical memory allocations is disabled, by default. Customers who decide to enable caching should perform data integrity tests to verify cache coherency issues do not exist.

- **SupportedIDs**
  
  *(NT 4.0 only)* This value contains the Device/Vendor IDs for the PLX devices that the driver supports. When the PLX NT 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 the included **DriverWizard** SDK application be used to modify this field. Invalid modification of this field may cause the **DriverWizard** to behave erratically or the device driver may fail to locate a device.

### 2.4.3.1 PLX Driver Options Wizard

The PLX SDK includes the **PLX Driver Options Wizard** application to manage all PLX driver settings. Using the wizard avoids the need to manually modify the registry. The wizard may be used in all supported versions of Windows. Details about each configuration option are displayed at the bottom whenever the item is selected.

After launching the wizard, select the desired driver and modify the options as needed. The updated settings will take effect when the driver is reloaded, either manually or after a system reboot.
2.4.3.2 System Page-Table Entries (PTEs) in Windows NT/2000/XP

When PLX device drivers load, they map PCI BAR spaces into kernel virtual space for access to the PLX chip’s registers and possible future access to the local bus. Additionally, mappings are required for some DMA transfers and various user application mapping requests. These mappings require system Page-Table Entries (PTEs), which are a limited resource. The larger the requested mapping area is, the greater the number of PTEs required.

Depending upon available system resources (PTEs), mappings attempted by PLX device drivers may fail. The chance of failure increases when multiple PLX PCI devices are installed, or devices request large PCI space sizes (8MB or more). To decrease the chance of failure, it is possible to increase the number of system PTEs 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

---

**Figure 2-6 PLX Driver Options Wizard**
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 PCI SDK installation increases this number to 80,000 or 0x13880. Users can try to increase the number further if virtual mapping requests continue to fail.

2.5 Support for the PLX PCI 6000 Series

The PLX PCI 6000 series is supported by different drivers, depending upon the mode it is running in. In general, 6000 series devices are configured as standard Transparent PCI-to-PCI bridges. Some PLX 6000 series devices, however, may be configured as Non-transparent bridges. The sections below describe how the PCI SDK supports both of these modes.

2.5.1 PLX PCI 6000 in Non-Transparent Mode

When a PCI 6000 device is running in Non-Transparent Mode, the system recognizes it as a standard bridge device. PLX provides a PCI 6000 specific driver for this device. Installation of this driver is identical to the PLX 9000-series drivers. Please refer to section 2.4 for more information.

2.5.2 PLX PCI 6000 in Transparent Mode

The PCI 6000 devices, running in Transparent Mode, are supported by means of a PLX PCI service driver, called *PlxPci*, running in Windows as a service or virtual device driver (VxD), rather than the traditional functional device driver. This driver provides basic access to the device for PCI register and configuration EEPROM accesses. In Plug ’n’ Play versions of Windows, Windows already provides a generic driver for standard PCI-to-PCI bridges, which the PLX PCI service does not replace.

The Windows NT/2000/XP service driver is located in `<Sdk_Install_Dir>\Win32\Driver\PlxPci` and the Windows 98/Me VxD driver is located in `<Sdk_Install_Dir>\Win32\Driver\PlxPci.vxd`.

2.5.2.1 Installation of *PlxPci* Service

The *PlxPci* driver is installed automatically by the SDK installation package but may also be installed manually. To perform a manual installation, follow the steps below:

- **Add the required driver registry entries**
  The registry files are located in `<Sdk_Install_Dir>\SupportFiles\WindowsDriverRegistryEntries`. For Win NT/2000/XP, the file is *PlxPci.reg* and *PlxPci_VxD.reg* for Win 98/Me. Double-clicking the file will automatically launch *RegEdit* and add the necessary entries.

- **Copy the PLX PCI Service driver to Windows**
  For Win NT/2000/XP, copy the driver file *PlxPci.sys* to `<Win_Dir>\System32\Drivers`. *PlxPci.sys* may be found in `<Sdk_Dir>\PciSdk\Win32\Driver\PlxPci\Driver\checked or free>`. For Win 98/Me, copy the driver file *PlxPci.vxd* to the `<Win_Dir>\System`. *PlxPci.vxd* may be found in `<Sdk_Dir>\PciSdk\Win32\Driver\PlxPci.vxd\Driver\checked or free>.

- **Copy the PLX PCI service driver API library to Windows**
  Copy *PciDrvApi.dll* to the Windows system directory. *PciDrvApi.dll* may be found in `<Sdk_Dir>\PciSdk\Win32\PciDrvApi\<Debug or Release>`.

- **Restart the system**

2.5.2.2 Using the PLX PCI Service with a Custom (OEM) ID Device

In cases where the Device/Vendor ID of a PLX PCI 6000-series device is modified from its default, the PLX PCI Service will not be able to properly identify the device. In this case, PLXMon will only provide...
limited access since it will not know what type of device is installed. The EEPROM dialog, for example, will be unavailable.

When modifying the Device/Vendor ID, it is recommended to modify the PLX PCI Service driver to properly identify the device. For debug purposes, the device type may be temporarily overridden in PLXMon. Refer to Section 7.6 for additional details.

2.5.2.3 Starting and Stopping the PlxPci Service

Since the PLX PCI Service runs like a background task, it may be stopped and started dynamically. The method for doing this depends upon the OS used. The following sections describe how to control the service in the various Windows OSes. Additionally, the PLX Driver Options Wizard may be used to start and stop drivers.

2.5.2.3.1 Windows 2000/XP

- Open the Device Manager (%My Computer Properties, Hardware tab) and display the hidden devices as shown below.
• Under **Non-Plug and Play Drivers**, find the **PLX PCI Service Driver** entry and double-click it.
The following dialog will appear. The **Start** and **Stop** buttons control loading and unloading of the driver, respectively.

### 2.5.2.3.2 Windows NT

- In control panel, select the **Devices** icon and select the **PlxPci** entry. The **Start** and **Stop** buttons control loading and unloading of the driver, respectively.
2.5.2.3.3 Windows 98/Me VxD

In Windows 98/Me, the PLX PCI Service VxD may be configured to load dynamically or statically. If the VxD is configured to load dynamically, it will load when an application calls one of the PLX PCI Service API calls and will unload when it is no longer needed. This is ideal for debugging since new builds of the driver may be installed without rebooting the system.

If the VxD is configured to load statically, it will load when Windows boots, but may never be unloaded or stopped. A static VxD should only be used once the driver is reliable and development is complete.

The default PLX VxD driver itself supports both methods of loading the driver. The only difference between the two configurations is the single registry entry "StaticVxD". To load the driver dynamically, the PlxPci registry entry should look similar to Figure 2-7. Figure 2-8 demonstrates the VxD configured to load statically.

![Figure 2-7 PLX PCI Service VxD Configured for Dynamic Loading](image)

![Figure 2-8 PLX PCI Service VxD Configured for Static Loading](image)

**Note:** The PLX Driver Option Wizard may be used to configure the PLX PCI VxD to load statically or dynamically. In other words, it will add or remove the "StaticVxD" entry automatically. Refer to section 2.4.3.1 for additional information.

2.6 Distribution of PLX Software

This section is meant for OEM customers who have written applications with PLX software and intend to ship it with their product. The following restrictions apply to the distribution of PLX software:

- The complete SDK package may not be distributed.
- Any and all source code written by PLX may not be distributed. PLX source code is covered by a proprietary license and is, therefore, not governed by the GNU Public License.
• Applications, utilities, or tools, such as PLXMon or PlxCm, 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 copyright message must appear in all software products generated and distributed, which use the PLX API libraries:

"Copyright © 2006 PLX Technology, Inc."

PLX allows distribution of the components required for OEM software to function properly and utilize the PLX chip through the PLX API. These are:

• PLX Device Drivers/Linux Modules in binary format only
• PLX API library in dynamically loadable binary format only (i.e PciDrvApi.dll)
• PLX PCI Service Driver API library in dynamically loadable binary format only (i.e PciDrvApi.dll)
• Any scripts needed to install the software and/or load modules
• PLX API libraries in binary archive format, only if end-users need to re-build OEM software that calls the PLX Host API. **See Note below.**
• Relevant C header files (e.g. `<PlxLinux/include/PlxApi.h>`), only if end-users need to re-build OEM software that calls the PLX Host API. **See Note below.**

**Note:** Distribution of the API library object or PLX header files is only permitted if OEM software, which calls the PLX Host API, needs to be rebuilt by an end-user. If end-users plan to call PLX API functions directly from their own software (not OEM distributed software), this is considered end-user development and they must purchase the PLX SDK separately. Furthermore, end-users must be made aware that PLX proprietary files are included in the package and should, therefore, not be distributed.

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

### 2.6.1 Windows PCI Host Software Distribution

In a Windows environment, the components necessary for applications to use the PLX API and access PLX devices are an INF file for the device, the PLX API library DLLs 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.

#### 2.6.1.1 Installation of the PCI Host Required Files

This section applies to the PCI 9000 series and 6000 series (Non-transparent Mode) installation files. For installation of the PCI 6000 series (Transparent Mode) related files, refer to section 2.5.2.1.

• 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 already in the provided INF. It should be copied as follows:

  - **Windows 98/Me/2000/XP:**
    - 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.
• The PlxApi.dll and PciDrvApi.dll files should be copied as follows:
  
  \textit{Windows 98/Me:}
  
  Copy to the <Windows_Dir>\System directory

  \textit{Windows NT/2000/XP:}
  
  Copy to the <Windows_Dir>\System32 directory

• The PLX device driver file installation is dependent upon the Operating System version as follows:
  
  \textit{Windows 98/Me/2000/XP:}
  
  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.

  \textit{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 2.4.2.2, 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, e.g. Pci9054.reg. \textit{Refer to the Registry Editor Help for more information on importing .reg files.}

\subsection{2.6.2 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 PCI SDK Software Architecture

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

The PCI 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 PCI SDK Software Architecture

3.1 PCI SDK Software Assumptions

There are some assumptions made by the PCI 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. The “PCI Configuration” of the PLX chip is performed by the system Host software, such as the BIOS or a Plug ‘n’ Play Host Operating System.

When the PLX Host (Windows/Linux) 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
- DOORBELL_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 transparent and is designed to share the port with raw serial data generated by the application.

When PLXMon downloads applications to RAM, it uses some Mailbox registers to communicate with the local software. These registers are only used during the download process, so they are available to applications at 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 PCI Host software components provided in the PCI SDK, which applies to Windows and Linux. The Windows PCI Host consists of the following components:

- PLX Host API Library
- PLX 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 library is provided to communicate with the PLX device drivers. When an API function is called by an application, the API library 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 and then to the calling application.

The PCI Host 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 used with any PLX PCI board.

There are two PLX API libraries, one for the PCI 9000 Host API and one for the PCI 6000 Host API. The PLX API libraries in the Windows environment file are implemented as Dynamically Linked Libraries (DLL). The files are called PlxApi.dll, which can be found in `<Sdk_Install_Dir>\Win32\Api` and PciDrvApi.dll, which can be found in `<Sdk_Install_Dir>\Win32\PciDrvApi`. Applications linked with these libraries will attempt to load the DLL when started; therefore, the DLLs must be found somewhere in the system path. DLLs are typically placed in the Windows system directory.

The PLX API library in the Linux environment file is implemented as a statically linked library, rather than dynamically loaded. Applications will link with the API library during the build process and will, therefore, contain API library code in the executable.

4.2 PLX Device Drivers

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

The PLX device drivers contain the API implementation for the PLX chip they support and the basic functionality required by all device drivers for the OS environment. The device driver accesses the PLX chip across the PCI bus by using OS system calls. The driver is also responsible for handling PCI interrupts from the PLX chip.

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.
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:

- **OS-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 library and dispatches control to the corresponding function used to perform the I/O operation.

- **Host 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, etc.

- **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 query of the chip for the power states it supports.
4.2.1 Windows Device Driver Directory Structure

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

![Figure 4-2 PLX Windows Driver Directory Structure](image)

---

PLX SDK User's Manual 4-3
The driver directories are described below:

- **Driver\PlxPCI**
  Contains source code for Windows NT/2000/XP PlxPCI service driver.

- **Driver\PlxPCI.vxd**
  Contains source code for the Windows 98/Me PlxPCI VxD service driver.

- **Driver\Common**
  Contains source code common to all PLX PCI 9000 drivers.

- **Driver\Common\WinNT**
  Contains source code common to all NT PLX PCI 9000 drivers.

- **Driver\Common\Wdm**
  Contains source code common to all WDM PLX PCI 9000 drivers.

- **Driver\Common\PlxChip\<DriverName>**
  Contains PLX chip-specific source code used for both NT and WDM 9000 drivers.

- **Driver\Common.6000**
  Contains source code for the PCI 6000 Non-transparent mode drivers.

- **Driver\Common.6000\WinNT**
  Contains Windows NT-specific source code for the PCI 6000 Non-transparent mode drivers.

- **Driver\Common.6000\Wdm**
  Contains WDM-specific source code for the PCI 6000 Non-transparent mode drivers.

- **Driver\Wdm\<DriverName>**
  This directory is used to build a chip-specific WDM version of the PCI 9000 and PCI 6000 (Non-transparent mode) drivers.

- **Driver\WinNT\<DriverName>**
  This directory is used to build a chip-specific NT version of the PCI 9000 and PCI 6000 (Non-transparent mode) drivers.

### 4.2.2 Building Windows Device 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.

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 is not used directly to build PLX drivers.

- Select and open the desired DDK environment (icons are installed by the DDK).

**Notes:**

- **Building the PLX PCI VxD service support both the Windows 98 DDK and the Windows Me environments (included with the latest Windows DDK).** If the Windows Me environment is used, the resulting VxD driver will not load properly in Windows 98 due to a DDK-imposed version conflict. If the driver is built with the Windows 98 DDK, it will load properly in both Windows 98 and Windows Me. The included PLX PCI service VxD driver binary is built with the Windows 98 DDK.

- **Similarly, the NT/2000/XP PLX PCI Service driver may be built with the Windows NT 4.0, 2000, or XP DDK.** If driver will be used in NT 4.0, the NT DDK environment must be used or the driver will not load properly; otherwise, the either the 2000 or XP DDK environments may be used. The included PLX PCI service driver binary is built with the Windows NT 4.0 DDK.
• 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.
For the **PlxPci** service driver, change to the **PlxPci** driver folder.

```
E:\Ddk\WinXP\2608~1.10> Cd \Plx\PciSdk\Win32\Driver\PlxPci
E:\Plx\PciSdk\Win32\Driver\PlxPci> CreateDriver.bat
```

For the Windows 908/Me **PlxPci VXD** service driver, change to the **PlxPci.vxd** driver folder.

```
F:\Ddk\Win98> Cd \Plx\PciSdk\Win32\Driver\PlxPci.vxd
F:\Plx\PciSdk\Win32\Driver\PlxPci.vxd> CreateDriver.bat
```

- Type **CreateDriver** to build the driver. **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 **WinNTBuildDriver.bat**, contain many comments explaining the steps involved to build the driver. Some sample build screenshots are provided below.

Once the driver is built, the new driver file may be used in Windows. **Refer to the Windows DDK for additional information on building and debugging drivers.**
Section 4
PCI Host Software

Win XP SDK - Checked Build Environment

******************************************************************************
* NOTE: Building of PLX drivers has been tested with the
* Windows 2000 SDK and Windows XP SDK (v2600.1106).
* 32-bit environments. 64-bit is not currently supported.
******************************************************************************

Building Windows Device Driver...
- TYP: WDM Driver
- CPU: x86
- CFG: chk
- PLX: Pc19054

Copying common files...

BUILD: Object root set to: --> objchk_wxp_x86
BUILD: Compile and Link for i386
BUILD: Examining f:\mydocs\sdk\pcisdk\win32\driver\wdm\pc19054 directory for files to compile.
BUILD: Compiling f:\mydocs\sdk\pcisdk\win32\driver\wdm\pc19054 directory
Compiling - driversversion.c for i386
Compiling - apifunctions.c for i386
Compiling - buffer.c for i386
Compiling - commanapl.c for i386
Compiling - dispatch.c for i386
Compiling - driver.c for i386
Compiling - eepromsupport.c for i386
Compiling - globalsave.c for i386
Compiling - pcisupport.c for i386
Compiling - plchip.c for i386
Compiling - plxinterrupt.c for i386
Compiling - plugplay.c for i386
Compiling - power.c for i386
Compiling - supportfunc.c for i386
Compiling - generating code... for i386
BUILD: Linking f:\mydocs\sdk\pcisdk\win32\driver\wdm\pc19054 directory
Linking executable - driver\chk\i386\pc19054.sys for i386
BUILD: Done

15 files compiled
1 executable built

F:\MyDocs\Sdk\Pcisdk\Win32\Driver\Wdm\Pc19054
Building Windows Device Driver...

- TYP: NT 4.0 Kernel-Mode Driver
- CPU: i386
- CFG: free
- PLX: PCi6000

Copying common files...

BUILD: Compile and link for i386
BUILD: Compiling f:\mydocs\sdk\pcisd\win32\driver\winnt\pci6000 directory
Compiling - driverversion.c for i386
Compiling - i386\apifunctions.c for i386
Compiling - i386\dispatch.c for i386
Compiling - i386\driver.c for i386
Compiling - i386\eepromsupport.c for i386
Compiling - i386\globalsave.c for i386
Compiling - i386\pcisupport.c for i386
Compiling - i386\pxlinterrupt.c for i386
Compiling - i386\supportfunc.c for i386
BUILD: Linking f:\mydocs\sdk\pcisd\win32\driver\winnt\pci6000 directory
Linking Executable - driver\i386\free\pci6000.sys for i386
BUILD: Done

9 files compiled
1 executables built

f:\MyDocs\Sdk\PciSdk\Win32\Driver\WinNT\Pci6000}
4.3 User-mode Applications

User-mode applications use the PCI API library 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 PCI 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 4-3 demonstrates the new project dialog.

![Figure 4-3 Visual C/C++ New Project Dialog](image)

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 4-5 demonstrates a typical Visual C project that is configured for the PLX API.

- **Add the PCI 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 4-4.
Section 4

PCI Host Software

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

- **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.

- **Insert “PciDrvApi.lib” into the Project**
  This library file contains link information for the PciDrvApi.dll file. If any PCI service driver API functions will be used in the application, this file is required. When the application is launched, the API DLL will automatically be loaded. The library file is provided in the <Sdk_Install_Dir>\Win32\PciDrvApi\Release directory.
#include 'PlxApi.h'

#if defined(_WIN32)
  #include '..\\Consac\ConsFunc.h'
  #include '..\\Consac\PlxInit.h'
#endif

#if defined(PLX_LINUX)
  #include 'ConsFunc.h'
  #include 'PlxInit.h'
#endif

/***************************************************************************/
/*
 * Function : main
 * Description: The main entry point
 */
/***************************************************************************/

void
main()
{
  HANDLE hDevice;
  DEVICE_LOCATION Device;

  /***************************************************************************/
  /* Select Device            */
  /***************************************************************************/
  Device.BusNumber = (U8)-1;
  Device_SLOTNumber = (U16)-1;
  Device_VENDORID = (U16)-1;
  Device_SERIALNumber[0] = '0';

  PlxPciDeviceOpen(
    &Device,
    &hDevice
  ).

Figure 4-5 Typical Visual C/C++ Project
5 Local Software

5.1 Local Software Components

The Local software included in the PCI SDK contains a complete solution starting from the first instruction executed by the CPU all the way to the user application. This section describes the Local software provided in the PCI SDK. Figure 5-1 shows the Local software architecture. The following components are included:

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

![Figure 5-1 Local Software Architecture](image)

5.1.1 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-dependent services to the other local software components. By definition, the BSP contains all board-specific code, which allows for modularity in design. 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 5-2 depicts the flow of the PLX BSP.
5.1.1.1 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.

5.1.1.2 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 5-2 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.

### 5.1.2 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 regard 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.

### 5.1.3 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 “Programmer’s Reference Manual” 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 5-3 demonstrates the BEM as it parses serial input data.
Section 5
Local Software

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 5-3 PLX BEM Serial Data Parsing](image)

5.1.4 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.

5.2 Local Memory Usage

Memory used by PLX applications is shown in Figure 5-4. 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 be running valid code. This is because PLXMon communicates with the local CPU to provide the application entry point. RAM applications perform initialization similar to the ROM application, except where it is not applicable, such as copying of code to RAM or RAM device initialization.
Figure 5-4 Typical Local Memory Usage

5.3 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 5-5 depicts the Local-side directory structure as found in the PCI SDK installation.
Section 5
Local Software

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.

### 5.4 Creating Local-side Images

The creation of local-side images is typically performed in an 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.
5.4.1 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 5-6 demonstrates the flow during make file processing.

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:

\(<\text{PLX\_Chip}> – <\text{CPU}>.\text{mak}\)

where:

- \(\text{PLX\_Chip}\) = Numeric associated with the PLX Chip model (for example, 54=9054).
- \(\text{CPU}\) = 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 \textit{Nmake} make file processing utility provided with Visual C/C++. Typically, toolsets provide their own make file processing utility, such as \textit{Dmake} in the DIAB tools. All PLX make files are compatible with \textit{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 as follows:

\(\text{Nmake} -f <\text{Makefile}> [\text{Clean} | \text{CleanAll}]\)
where:

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Makefile</td>
<td>Make file name</td>
</tr>
<tr>
<td>Clean</td>
<td>Delete all intermediate build files and leave final image</td>
</tr>
<tr>
<td>CleanAll</td>
<td>Delete all build files, including final image</td>
</tr>
</tbody>
</table>

### 5.4.2 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.

### 5.4.3 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.
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.

```plaintext
NOTE: Please modify this file \( \text{<F:\PLX\PciSdk\Bin\EnvDiab.bat>} \) to setup the proper environment for building local-side code.

**PLX SDK Directory:** F:\PLX\PciSdk
**Diab Tools Directory:** F:\PLX\Diab\4.3g

F:\PLX\PciSdk\Top > cd Bsp\9656-860.rdk
F:\PLX\PciSdk\Top\Bsp\9656-860.rdk > Nmake -f Makefile.nak
```
During the build process, information about the target will be displayed as follows:

- **PLA**: The target platform
- **CPU**: Target CPU
- **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** or **Binary**
- **CMP**: The selected compiler toolset
NOTE: Please modify this file (F:\PLx\PciSdk\Bin\Env\Diab.bat) to setup the proper environment for building local-side code.

PLX SDK Directory: F:\PLx\PciSdk
Diab Tools Directory: F:\PLx\Diab\4.3g

F:\PLx\PciSdk\Iop> cd Api\9656
F:\PLx\PciSdk\Iop\Api\9656> Nmake /nologo -f 96-860.mak

Building: Api
- PLx: 9656-860
- CPU: 860
- TGI: Lib
- CFG: Release
- IMG: Elf
- CMF: Diab

Compiling: DmaFunctions.c
Compiling: EepromAccess.c
Compiling: InterruptFunctions.c
Compiling: MuFunctions.c
Compiling: MiscFunctions.c
Compiling: NewCapabilities.c
Compiling: PciBusAccess.c
Compiling: PciConfigAccess.c
Compiling: P1xChipInit.c
Compiling: PowerManagement.c
NOTE: Please modify this file (F:\Plx\PciSdk\Bin\EnvDiab.bat) to setup the proper environment for building local-side code.

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

F:\Plx\PciSdk\Iop> cd Ben
F:\Plx\PciSdk\Iop\Ben> Nmake /nologo -f 96-860.mak

Building: Ben
- PLA: 9656-860
- CPU: 860
- TGT: Lib
- CFG: Release
- IMG: Elf
- CMP: Diab

Compiling: BenMain.c

Creating archive: Ben.a ...

a - BenMain.o

Library file "Lib-9656.860\Diab-Elf\Ben.a" built successfully.

NOTE: Please modify this file (F:\Plx\PciSdk\Bin\EnvDiab.bat) to setup the proper environment for building local-side code.

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

F:\Plx\PciSdk\Iop> cd Samples\Hello
F:\Plx\PciSdk\Iop\Samples\Hello> Nmake /nologo -f 96-860.mak

Building: HelloRam
- PLA: CPC1 9656 RDK
- CPU: 860
- TGT: App
- CFG: Release
- LOC: Ram
- IMG: Elf
- CMP: Diab

Compiling: Hello.c

Linking application: HelloRam.Elf

RAM image file "CPC19656.860\Diab-Elf\HelloRam.Elf" built successfully.
• To change configuration parameters, modify environment variables as needed. Settings in the make file override environment variables.

5.5 Local-side Debugging Tools and Tips

When working with local-side environments, it is helpful to use many techniques for debugging code. The greater the developer’s knowledge of debugging 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 you can write an assembly function that loops forever to “freeze” the CPU and then call it from a C program when needed:

In assembly code:
```
.global MyPause
MyPause:
    b MyPause
```

In C code:
```c
MyPause(0x14);
```

When building local code, build with the DEBUG version to avoid optimizations and to include symbol information for debuggers. The RELEASE version should be built once everything is functioning.

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, and 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:
```c
while (*((volatile U32*)0x100000) != 0x12345678);
```
6 Non-Windows Operating System Support

Note: The Non-Windows OS support provided is completely independent of the remainder of the SDK. Rebuilding a component will not reference PCI SDK files in any other directory.

The PCI SDK contains support for Linux environments, which is described in this section. Note that the software mentioned here can be used as a reference or a starting point in designs, which are not directly supported.

6.1 Linux Host-Side Support

The PCI SDK includes Linux Host-side support for the PLX chips. All Linux related files and directories are located in <Sdk_Install_Dir>\Linux_Host. Documentation for the Linux support is not included this manual; however, much of the Windows host-side architecture applies to the Linux Host-side support as-well. The Linux Host Release Notes are provided as an HTML file, which can be found in the folder <Sdk_Install_Dir>\Documentation.
7 PLX Chip Debug Tool - PLXMon

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, programming of FLASH devices, and EEPROM access.

7.1 PLXMon Access Modes

PLXMon accesses 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 7-1 shows the PLX communication modes.

![Figure 7-1 PLXMon Communications Modes](image)

7.1.1 PCI Mode

In PCI mode, all accesses to the PLX chip are performed directly through the PCI bus, via the SDK API and PLX device driver. If a PLX driver is not installed/loaded, PCI mode will be unavailable. In PCI mode, the upper pane in PLXMon is disabled. The lower pane is an interpreter that accepts commands to access registers and memory.
7.1.2 EEPROM File Edit Mode

If a PLX device is not detected in the system, PLXMon displays a dialog (Figure 7-3), which provides two options: Enter EEPROM File Edit mode or attempt a connection to enter Serial mode.

The EEPROM edit mode is provided for those who need to create or modify EEPROM files, which will be used with an I/O programmer. In this mode, since no PLX devices are physically present in the system, PLXMon cannot program the EEPROM device directly.
7.1.3 Serial Mode

In Serial Mode, PLXMon establishes a serial connection with a device. In this mode, the software executing on the local CPU (PLX BEM) accepts and carries out commands from PLXMon to perform necessary tasks. While connected, the upper pane of PLXMon provides a terminal interfaces, similar to other serial terminal applications, such as HyperTerminal. The lower pane is an interpreter that accepts commands to access registers and memory. It is important to note that in Serial mode, the local CPU handles commands entered in the lower pane, so memory and registers are accessed from the local CPU’s point of view. In Serial mode, the command `dl 100000` will read from the local address location 1MB. Conversely, in PCI mode, only virtual addresses are allowed, so the same command will most likely result in an invalid address.
Section 7
PLX Chip Debug Tool - PLXMon

Figure 7-4: PLXMon in Serial Mode

7.2 PLXMon Toolbar

Figure 7-5: PLXMon Toolbar

PLX Mon - Serial Connection to 9054 Device [Dev/Ven ID = 1860 10B5]

PLX PCI 9054RDK-860 Evaluation Board
Copyright (c) 2003 PLX Technology, Inc.

PLX SDK : 4.20
PLX Chip: 9054-A8

Testing DRAM (32MB), press any key to cancel...
Memory Test: Ok

PLX Command-line Monitor application

>
The PLXMon toolbar (Figure 7-5) provides multiple options, which are described below:

- **Select a Device**
  View all PLX devices found and select one to work with. Only devices, for which a PLX driver is loaded, will be available.

- **Download to device**
  Opens the download dialog, which allows downloading of RAM images and programming of the FLASH ROM.

- **View all PCI devices**
  Open a dialog, which displays all PCI devices in the system. Selecting one displays all PCI registers of the device

- **Reset device**
  In PCI mode, resets a device by using the Software Reset feature of PLX chips. In Serial Mode, issues a reset command to the local CPU.

- **Memory Access**
  Opens the memory access dialog.

- **Performance Measure Dialog**
  In PCI mode for PLX 9000-series devices, provides a software measure for DMA and Direct Slave transfers. Refer to the Performance Measure Dialog, section 7.7.

- **Connect to device**
  Attempt a serial connection to the device. If the local software implements the BEM protocol, PLXMon will establish a connection.

- **View Register Groups**
  Open dialogs for the various register groups and EEPROM. The PLX chip type determines available groups.

### 7.3 Working with PLXMon Dialogs

#### 7.3.1 Register Dialogs

The register dialogs in PLXMon are very simple to use. Users simply enter values, in Hexadecimal format, and PLXMon will update the value in the chip. For some registers with numerous bit-fields, PLXMon provides additional detail screens, which can be selected with the details button - . Figure 7-6 demonstrates a typical register dialog.

**Tips on working with register dialogs:**

- All values are in Hexadecimal format
- The register dialogs are available in both Serial and PCI modes. In Serial mode, PLXMon sends commands to the local CPU to perform register accesses. In PCI mode, PLXMon calls the PLX Host API to access registers.
- The register offsets displayed are dependent upon the mode of operation. In Serial mode, the offsets are from the local CPU’s point of view. Refer to the PLX chip data book for more information regarding offsets.
• In the register dialogs, PLXMon will update a register value as soon as focus shifts from the field (i.e. the TAB key or clicking on a different field with the mouse).

**Figure 7-6: Typical PLXMon Register Dialog**

### 7.3.2 EEPROM Dialogs

The EEPROM dialogs in PLXMon behave very similar to the register dialog, with a few exceptions. Additionally, the EEPROM dialogs provide options to save/load values to/from files. Figure 7-7 demonstrates a typical EEPROM dialog.

**EEPROM Dialog Differences from Register Dialogs:**

- Displayed offsets are from the EEPROM base (default), but offsets of the target register in the chip can be selected, as well.
- Values are not written to the EEPROM device until the **Write** button is selected.
- Values can be loaded from or saved to a file. When working with EEPROM files, PLXMon will only load or save enough values to fill the PLX chip’s portion of the EEPROM. Additional values are discarded.
7.3.3 Memory Access Dialog

Selecting the memory access button will open the dialog shown in Figure 7-8. The memory dialog allows reading of blocks of memory from the local bus or from the DMA buffer, as well as the ability to fill memory, as shown in Figure 7-9. For more control over memory accesses, use the \texttt{db}, \texttt{dw}, \texttt{dl}, \texttt{eb}, \texttt{ew} and \texttt{el} commands. Note that in PCI mode, virtual addresses are used. \textit{Refer to Section 0 for more information.}
Section 7
PLX Chip Debug Tool - PLXMon

7.4 Enter Information for a New Device

In order for PLXMon to perform download tasks, properties of the device must be provided. These include the local CPU endian setting, as well as the Flash device types and local base addresses. PLXMon associates device properties based on the Device/Vendor ID combination. Properties of PLX Reference Design Kits are already included in PLXMon. If a device is installed which PLXMon does not know properties of, PLXMon will notify the user to add the information, as shown in Figure 7-10. The
properties can be added by selecting **File->Options->Properties**, which will display the dialog shown in Figure 7-11.

![Figure 7-10: Device Properties Needed Notification](image)

![Figure 7-11: Device Properties](image)

### 7.5 Download of Images to the Local-Side

PLXMon provides the ability to download images to a device. The images can be RAM applications or a ROM image, which will be programmed in FLASH. In PCI mode, images are transferred directly across the PCI bus. In Serial mode, images are transferred through the serial port and the local CPU handles programming of the FLASH and executing a RAM application. Figure 7-12 demonstrates the PLXMon download dialog.

![Figure 7-12: PLXMon Download Dialog](image)
Section 7
PLX Chip Debug Tool - PLXMon

Download to Board

File to download:
F:\Plx\Plx\Samples\Monitor\CPCI9556.860\Diab-Elf\MonitorRom.bin

Download to/from:
- RAM device
- FLASH device 1
- FLASH device 2

Image Type:
- Binary
- ELF
- COFF
- IBM ELF

Reset device after download

Addresses:
Flash Offset (Hex):
00000

CPU Entry Point (Hex):

Ready to download...

Save Log As...

Figure 7-12: Download Image Dialog

Download Options:

- **RAM download** – An image of supported type (Binary, ELF, COFF, etc) is transferred to local memory. The local CPU is then informed of the CPU entry point, which is used to start executing the RAM application. For Binary images, the CPU entry point and download address must be provided. For other image types, PLXMon automatically determines the values from the image header information. *Note: The local CPU must be executing valid code in order to accept a RAM application.*

- **FLASH download** – An image of supported type (Binary, ELF, COFF, etc) is programmed directly into the FLASH ROM device of the board. The FLASH offset, which determines where PLXMon starts programming, must be provided. *Note: In PCI mode, the properties of the device must be provided in order for PLXMon to successfully program the FLASH. Refer to Section 7.4 for more information. In Serial mode, the image is transferred and the local CPU handles programming of the FLASH ROM.*

- **Reset device after download** – When programming a FLASH ROM, typically, the device is reset so that the local CPU restarts and executes from FLASH ROM. In PCI mode, it is possible to simply program the FLASH, but disable the reset feature. This is useful during development when testing programming of the FLASH, since a corrupt FLASH may lead to device boot problems.

- **Save Log As...** - The message log provides detailed information during each step of the download. The log can be saved to a text file for debug purposes.

### 7.6 Specifying PLX Chip Type for Unknown Devices

If the Device/Vendor ID of a PCI 6000 series bridge is modified from its default, PLX software may fail to properly identify the device as a PLX chip. In this case, PLXMon will not be able to properly display all of the PCI registers and the EEPROM contents. PLX drivers rely on known Device/Vendor ID combinations
Section 7

PLX Chip Debug Tool - PLXMon

to detect PLX PCI 6000 and 8111 devices. As a result, the IDs are hard-coded into the driver source code. A customer that changes an ID will, therefore, need to modify the driver source and rebuild it. PLX software, however, provides an option to manually override the chip type in the event that it is not detected properly. This can be performed in PLXMon in the “Select a PCI Device” dialog.

Simply select a device and then select the option to manually set the chip type. Figure 7-13 shows how to manually select a chip type.

![Select a PCI Device dialog](image)

<table>
<thead>
<tr>
<th>Bus</th>
<th>Slot</th>
<th>Fn</th>
<th>Dev ID</th>
<th>Ver ID</th>
<th>PLX</th>
<th>Rev</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>00</td>
<td>00</td>
<td>0605</td>
<td>1106</td>
<td>--</td>
<td>--</td>
<td>Host bridge device</td>
</tr>
<tr>
<td>00</td>
<td>01</td>
<td>00</td>
<td>8605</td>
<td>1106</td>
<td>--</td>
<td>--</td>
<td>PCI-to-PCI bridge device</td>
</tr>
<tr>
<td>00</td>
<td>04</td>
<td>00</td>
<td>0686</td>
<td>1106</td>
<td>--</td>
<td>--</td>
<td>ISA bridge device</td>
</tr>
<tr>
<td>00</td>
<td>04</td>
<td>01</td>
<td>0571</td>
<td>1106</td>
<td>--</td>
<td>--</td>
<td>IDE controller</td>
</tr>
<tr>
<td>00</td>
<td>04</td>
<td>02</td>
<td>3038</td>
<td>1106</td>
<td>--</td>
<td>--</td>
<td>USB controller (Universal Host)</td>
</tr>
<tr>
<td>00</td>
<td>04</td>
<td>03</td>
<td>3038</td>
<td>1106</td>
<td>--</td>
<td>--</td>
<td>USB controller (Universal Host)</td>
</tr>
<tr>
<td>00</td>
<td>04</td>
<td>04</td>
<td>3057</td>
<td>1106</td>
<td>--</td>
<td>--</td>
<td>Host bridge device</td>
</tr>
<tr>
<td>00</td>
<td>0D</td>
<td>00</td>
<td>9050</td>
<td>1087</td>
<td>--</td>
<td>--</td>
<td>Ethernet controller</td>
</tr>
<tr>
<td>00</td>
<td>0E</td>
<td>00</td>
<td>00A2</td>
<td>3388</td>
<td>--</td>
<td>--</td>
<td>PCI-to-PCI bridge device</td>
</tr>
<tr>
<td>00</td>
<td>0F</td>
<td>00</td>
<td>0022</td>
<td>3388</td>
<td>6150</td>
<td>BB</td>
<td>PCI-to-PCI bridge device</td>
</tr>
<tr>
<td>00</td>
<td>10</td>
<td>00</td>
<td>1860</td>
<td>1085</td>
<td>9054</td>
<td>AB</td>
<td>Other bridge device</td>
</tr>
<tr>
<td>01</td>
<td>00</td>
<td>00</td>
<td>4742</td>
<td>1002</td>
<td>--</td>
<td>--</td>
<td>VGA-compatible device</td>
</tr>
</tbody>
</table>

![Update PLX Chip Type dialog](image)

Figure 7-13 Manually Setting the PLX Chip Type

After the selection has been made, PLXMon will treat the device as the user-selected type, as can be seen in Figure 7-14. Before setting the PLX chip type, it is important to note the following:

- No error checking is performed when setting the PLX chip type. If a PLX chip is selected that does not match the installed hardware, the PLXMon and/or the system may behave erratically.

- Once the chip type is selected, the PLX driver will attempt to automatically detect the PLX revision. If this is not detected, the revision will default to the value in the PCI revision ID register.

- Modification of the PLX chip type is not permanent. It will remain in effect as long as the PLX driver is loaded and not re-started. For a permanent setting, it is recommended that the appropriate PLX PCI Service driver is modified and rebuilt to properly detect the custom ID. If the PLX PCI VxD is configured to load dynamically in Windows 98/Me, the modified PLX chip type will be lost if PLXMon is closed or possibly when selecting another device.

- This option may only be used with PLX PCI 6000 series and 8111 devices.
7.7 Performance Measure Dialog

PLXMon includes a performance measure dialog, which provides a software measure of data transfer performance. The dialog supports DMA and Direct Slave transfers, with multiple options for each. This section describes the details of how to use the dialog. Figure 7-15 shows a snapshot of the dialog.
7.7.1 Notes before Using the Performance Measure

Before using the performance dialog, it is important to be aware of the following imitations and notes:

- The Performance Measure is a simple software measurement of performance. The transfer rate is calculated by dividing the total number of bytes transferred by the total elapsed time. As a result, software overhead is a factor in the measure, although the Performance Measure is very efficient and includes very little overhead.

- The transfer rates provided by the Performance Measure should be treated as relative numbers rather than absolute values. The intention is to start with some base configuration, tweak some options and/or chip settings, then re-run the test to determine if performance has improved and repeat to achieve the optimal configuration.

- The Performance Measure does not validate the addresses used to transfer data to/from. This includes the PCI and local addresses for DMA and the local address for Direct Slave. It is left to the user to ensure that sufficient memory is provided for the transfer.

- The Performance Measure does not perform any data error checking. It is assumed that hardware is working properly.

- When selecting to use the PLX API to transfer data, it is important to note that there is a significant overhead with doing so. The API sends and receives messages from the PLX driver, which performs the actual transfer. If data transfer sizes are relatively small, the API overhead will
be a significant impact to performance. As data transfer sizes get larger, the API overhead becomes less significant.

- The Performance Measure cannot guarantee burst transactions. Software has no means to force burst transactions. All software can do is enable burst in the hardware and, if conditions are right, the hardware will initiate burst transactions.

- Other than the options specified, the Performance Measure will leave chip settings intact. It is assumed that the chip is properly configured to access the intended devices. For example, if PCI BAR 2 on a 9054 will be used to access an 8-bit device, it is assumed that the Space 0 Bus Region Descriptor is configured properly and that the Space 0 Remap register is set to properly access the desired device.

7.7.2 Performance Measure Options

The performance measure provides numerous options to perform different type of transfers in different configurations. The individual options are explained below.

7.7.2.1 DMA Performance Test

When the DMA test is selected, the Performance Measure will perform DMA transfers to or from the specified addresses. The test continuously repeats the same DMA transfer until it is halted. Figure 7-16 shows a DMA performance test and the reported results.
The items below provide details about the individual DMA options. When the Performance Measure is initially opened and DMA is available, it will provide the DMA Common Buffer properties, which are provided by the PLX driver. This is the same information obtained with `PlxPciCommonBufferProperties`.

**Note:** DMA is available only to PLX devices that include a DMA engine, including the 9080, 9054, 9056, and 9656.

- **Local Address** This determines the starting 32-bit local address where data is transferred to/from. This value is placed directly into the Local address register of the DMA engine.

- **PCI Address** This determines the starting 32-bit PCI physical address where data is transferred to/from. This address, for example, may be the PLX DMA Common Buffer PCI address or an address taken from the PCI BAR of another PCI device, such as an Ethernet controller. This value is placed directly into the PCI address register of the DMA engine.

- **Channel** This determines which DMA channel the Performance Measure will use.

- **Use** This determines whether DMA completion is detected by waiting for the interrupt or polling the DMA done bit. In general, polling results in better transfer rates due to less overhead, but the CPU is highly utilized, so the end user system performance suffers.

- **Transfer** This determines which direction the DMA engine will transfer data.

- **Byte Count** This is the number of bytes transferred during each test iteration.

- **Bursting** This option determines whether DMA bursting is enabled in the hardware. Note that the devices that the DMA engine transfers to/from must support the selected type of burst transaction.

### 7.7.2.2 Direct Slave Performance Test

When the Direct Slave test is selected, the Performance Measure will use the Host CPU to transfer data to/from a PLX device through one of the PCI BAR spaces. The test will repeat continuously until it is halted. Figure 7-17 depicts a completed Direct Slave test and the reported results.
The items below provide details about the individual Direct Slave options.

- **PCI BAR to Use** This determines which PCI BAR space to use for the transfer. The PCI BAR must be a valid PCI memory space that is enabled on the PLX device. I/O type spaces are not supported. It is assumed that the PCI space is properly configured to access the desired local device. This includes the remap and bus region descriptors.

- **Offset into PCI BAR** This value determines the starting offset into the PCI BAR where the Performance Measure will transfer data.

- **Method** This option determines whether the PLX API is used to transfer data or a direct access is performed. The PLX API method will use the functions `PlxBusIopRead` and `PlxBusIopWrite`, whereas, the direct method will obtain a virtual address for the PCI BAR with `PlxPciBarMap`, then use that address to directly access the PCI space. The direct method effectively bypasses the PLX API.

- **Access Size** This option determines how data is accessed, whether it is 8-bit, 16-bit, or 32-bit. This option should not be confused with the "Bus Width" of the Bus Region Descriptor for a space. The Bus Width is used to specify the port-size of the connected local device, for example, a 16-bit flash device. The Access Size determines the type of cycle issued by the Host CPU.

- **Transfer** This determines whether the Host CPU reads from or writes to the PCI BAR.

- **Byte Count** This is the number of bytes transferred during each test iteration.
- **Bursting** This option determines whether Direct Slave busting is enabled in the Bus Region descriptor for the PCI space. This option does not guarantee that burst transactions will occur, since software is not able to force bursting. In a standard PC, for example, the Host Bridge does not allow burst reads from PCI devices to the Host CPU, resulting in typically poor burst read performance. Note that the devices that data will be transferred to/from must support the selected type of burst transaction.

### 7.8 The Command-Line Interface

In the lower pane of PLXMon, a command-line interface is provided, as shown in Figure 7-18. The list of available command is shown in Table 7-1.

![Command-line Interface](image)

#### Figure 7-18: Command-line Interface

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>db, dw, dl</td>
<td>Read memory using Byte (8-bit), Word (16-bit), Longword (32-bit)</td>
</tr>
<tr>
<td>eb, ew, el</td>
<td>Write to memory using Byte (8-bit), Word (16-bit), Longword (32-bit)</td>
</tr>
<tr>
<td>ib, iw, il</td>
<td>Read from I/O port using Byte (8-bit), Word (16-bit), Longword (32-bit)</td>
</tr>
<tr>
<td>ob, ow, ol</td>
<td>Write to I/O port using Byte (8-bit), Word (16-bit), Longword (32-bit)</td>
</tr>
<tr>
<td>pci</td>
<td>Read/Write to a PCI register of the PLX chip</td>
</tr>
<tr>
<td>reg</td>
<td>Read/Write to a local register of the PLX chip</td>
</tr>
<tr>
<td>vars</td>
<td>Display PLXMon variables. See Section 7.8.2</td>
</tr>
<tr>
<td>ver</td>
<td>Display version information</td>
</tr>
<tr>
<td>clear</td>
<td>Clear the command-line pane</td>
</tr>
<tr>
<td>quit</td>
<td>Exits PLXMon</td>
</tr>
</tbody>
</table>

#### Table 7-1: PLXMon Command-line Commands

### 7.8.1 Working with Virtual Addresses

In PCI mode, PLXMon executes as an application and, therefore, must use virtual addresses to access memory. A PCI BAR address, for example, cannot be referenced directly. As a result, PLXMon relies on PLX drivers to provide a virtual mapping for all memory spaces that may be accessed. This includes any valid PCI BAR memory spaces and the DMA buffer allocated by the driver.
Section 7
PLX Chip Debug Tool - PLXMon

Note: Virtual addresses are not used for I/O ports, only for memory regions. Although the driver performs the actual I/O access, the referenced port address is the actual address found in the PCI BAR register. I/O regions are not mapped into virtual space.

7.8.2 Command-Line Variables

PLXMon creates some variables to aid users with dealing with virtual addresses. Figure 7-19 demonstrates the `vars` command in PLXMon, which lists the default variables and the memory region they represent. Variables can be used with the `d(b,w,l)` or `e(b,w,l)` commands.

Note: Accessing memory with these variables results in a direct memory access from PLXMon. The PLX driver just provides the initial virtual mapping, but is completely bypassed during memory accesses.

![Figure 7-19: PLXMon Variables](image)

<table>
<thead>
<tr>
<th>Variable</th>
<th>Virtual Address</th>
<th>Physical Address</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Plx</td>
<td>00D80000</td>
<td>---</td>
<td>Plx register address</td>
</tr>
<tr>
<td>s0</td>
<td>00D90000</td>
<td>---</td>
<td>Space 0 address</td>
</tr>
<tr>
<td>s1</td>
<td>01D90000</td>
<td>---</td>
<td>Space 1 address</td>
</tr>
<tr>
<td>s2</td>
<td>00000000</td>
<td>---</td>
<td>Space 2 address</td>
</tr>
<tr>
<td>s3</td>
<td>00000000</td>
<td>---</td>
<td>Space 3 address</td>
</tr>
<tr>
<td>HBuf</td>
<td>00D70000</td>
<td>0183D000</td>
<td>DMA scratch buffer (64 Kb)</td>
</tr>
</tbody>
</table>

```bash
>vars

Plx 00D80000
s0 00D90000
s1 01D90000
s2 00000000
s3 00000000
HBuf 00D70000 0183D000
```

Figure 7-19: PLXMon Variables
Appendix A  References