PLX PCI SDK

SOFTWARE DEVELOPMENT KIT

Service Module Manual

Version 3.0

© 1999 PLX Technology, Inc. All rights reserved.

Notice

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

PLX provides the information and data included in this document for your benefit, but it is not possible to entirely verify and test all information, in all circumstances, particularly information relating to non-PLX manufactured products. PLX makes neither warranty nor represents relating to the quality, content, or adequacy of this information. The information in this document is subject to change without notice. Although every effort has been made to ensure the accuracy of this manual, PLX shall not be liable for any errors, incidental, or consequential damages in connection with the furnishing, performance, or use of this manual or examples herein. PLX assumes no responsibility for damage or loss resulting from the use of this manual, for loss or claims by third parties, which may arise through the use of the RDK or for any damage or loss caused by deletion of data as a result of malfunction or repair.

Document number: PCI-SDK-MAN-SM-P1-3.0

PLX SDK License Agreement

PLX SOFTWARE LICENSE AGREEMENT

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

LICENSE Copyright © 1999 PLX Technology, Inc.

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

GENERAL

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

WARRANTY

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

PLX Copyright Message Guidelines

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

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

Requirements:

- Arial font
- Font size 12 (minimum)
- Bold type
- Must appear as shown above in the first section or the so called "Introduction Section" of all manuals
- Must also appear as shown above in the beginning of source code as a comment

This page is intentionally left blank.

TABLE OF CONTENTS

1. INTRODUCTION 1-1

1.1 About this Manual 1-1
1.2 Conventions 1-1
1.3 Development Requirements 1-1
1.4 Customer Support 1-1

2. PCI SDK V3.0 DRIVER DESIGN 2-1

2.1 Overview 2-1

3. DRIVER FUNCTIONS 3-1

3.1 Windows NT Driver Maintenance Functions 3-2
DriverEntry 3-2
PlxDrvUnload 3-3
PlxCreateDeviceObject 3-4
PlxMapInUserSpace 3-5
PlxUnmapUserSpaceAddr 3-6
PlxMapAllUserAddresses 3-7
PlxUnmapAllUserAddresses 3-8
IsValidXXXXDevice 3-9
PlxValidIdRead 3-10
PlxSearchDevices 3-11
PlxInitializeEventLog 3-12
PlxReportEvent 3-13
PlxDumpRegistryInformation 3-14
PlxSleep 3-15
GetBarIndex 3-16
GetNTErrorCode 3-17
3.2 Windows NT Interrupt functions 3-18
plxIsr 3-18
plxDpc 3-19
PlxUnmapSglTransfer 3-20
PlxUnmapBlockTransfer 3-21
3.3 Windows NT Buffer Functions 3-22
DriverBufferInit 3-22
DriverBufferTerminate 3-23
DriverBufferMalloc 3-24
DriverBufferFree 3-25
DriverBufferVaToPhys 3-26
DriverBufferFlush 3-27

 © PLX Technology, Inc., 1999

3.4 Windows NT Dispatch functions 3-28
PlxDrvDispatch 3-28
PlxProcessIO 3-29
PlxIrpQueueCancel 3-30
PlxIrpQueueCancelAll 3-31
PlxCompleteIrp 3-32
PlxCancelAttachIrp 3-33
3.5 Windows 98 Driver Functions 3-34
DriverEntry 3-34
PlxDrvUnload 3-35
PlxCreateDeviceObject 3-36
PlxAllocateCommonBuffer 3-37
PlxDeallocateCommonBuffer 3-38
AddDevice 3-39
CompleteRequest 3-40
ForwardAndWait 3-41
LockDevice 3-42
UnlockDevice 3-43
OnRequestComplete 3-44
RemoveDevice 3-45
PlxDumpRegistryInformation 3-46
GetNTErrorCode 3-47
PlxReportEvent 3-48
PlxSleep 3-49
3.6 Windows 98 Interrupt functions 3-50
PlxIsr 3-50
PlxDpc 3-51
PlxUnmapSglTransfer 3-52
PlxUnmapBlockTransfer 3-53
3.7 Windows 98 Buffer Functions 3-54
DriverBufferInit 3-54
DriverBufferTerminate 3-55
DriverBufferMalloc 3-56
DriverBufferFree 3-57
DriverBufferVaToPhys 3-58
DriverBufferFlush 3-59
3.8 Windows 98 Plug and Play Functions 3-60
RequestPnp 3-60
DefaultPnpHandler 3-61
HandleRemoveDevice 3-62
PlxReadConfigReg 3-63

 © PLX Technology, Inc., 1999

PlxWriteConfigReg 3-64
HandleStartDevice 3-65
HandleStopDevice 3-66
ShowResources (DEBUG only) 3-67
3.9 Windows 98 Power Functions 3-68
RequestPower 3-68
DefaultPowerHandler 3-69
HandleSetPower 3-70
HandleQueryPower 3-71
OnFinishPowerUp 3-72
SendDeviceSetPower 3-73
SendSelfSetPowerRequest 3-74
OnPowerRequestComplete 3-75
3.10 Windows 98 Read/Write Functions 3-76
DisableDevice 3-76
EmpowerDevice 3-77
EnableDevice 3-78
PlxGetBusSlotNumber 3-79
RequestCreate 3-80
RequestClose 3-81
SetBarIndex 3-85
PlxUnmapAll 3-86
StartDevice 3-87
PlxCancelAllPendingIrp 3-88

4. SERVICES FUNCTIONS 4-1

BusIopRead 4-5
BusIopWrite 4-7
DmaBlockChannelClose 4-9
DmaBlockChannelOpen 4-10
DmaBlockTransfer 4-12
DmaBlockTransferRestart 4-14
DmaSglChannelClose 4-15
DmaSglChannelOpen 4-16
DmaSglTransfer 4-18
DmaShuttleChannelClose 4-20
DmaShuttleChannelOpen 4-21
DmaShuttleTransfer 4-23
IntrDisable 4-25
IntrEnable 4-26
IntrStatusGet 4-27

 © PLX Technology, Inc., 1999

IoPortRead 4-28
IoPortWrite 4-30
MuInboundPortRead 4-32
MuInboundPortWrite 4-33
MuOutboundPortRead 4-34
MuOutboundPortWrite 4-35
PowerLevelGet 4-36
PowerLevelQuery 4-37
PowerLevelSet 4-38
RegisterDoorbellRead 4-39
RegisterDoorbellSet 4-40
RegisterRead 4-41
RegisterMailboxRead 4-42
RegisterMailboxWrite 4-43
RegisterReadAll 4-44
RegisterWrite 4-45
SerialEepromRead 4-46
SerialEepromWrite 4-48
UserRead 4-50
UserWrite 4-51
DmaControl 4-52
DmaStatus 4-54
HotSwapIdRead 4-56
HotSwapNcpRead 4-57
HotSwapStatus 4-58
MuOutboundIndexRead 4-59
MuOutboundIndexWrite 4-60
PciAbortAddrRead 4-61
PmIdRead 4-62
PmNcpRead 4-63
SerialEepromPresent 4-64
VpdIdRead 4-65
VpdRead 4-66
VpdNcpRead 4-67
VpdWrite 4-68

 © PLX Technology, Inc., 1999

1. Introduction

1.1 About this Manual

The PCI SDK v3.0 provides Windows 98 and Windows NT device drivers for PLX reference design boards. This manual provides information that you may need if you are writing a Windows driver for your board. This PCI SDK supports IOP 480, PCI 9054 and PCI 9080 devices.

1.2 Conventions

Please note:

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

1.3 Development Requirements

The PCI SDK was developed for Window NT 4.0 and Windows 98 operating systems.

The IOP 480 drivers were developed using the Microsoft Visual C++ Developer Studio version 5.0 nmake utility as well as the Windows NT and Windows 98 DDKs.

1.4 Customer Support

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

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

 © PLX Technology, Inc., 1999

You may contact PLX customer support at:

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

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

You may send email to one of the following addresses:

`west-apps@plxtech.com`
`mid-apps@plxtech.com`
`east-apps@plxtech.com`
`euro-apps@plxtech.com`
`asia-apps@plxtech.com`

 © PLX Technology, Inc., 1999

2. PCI SDK v3.0 Driver Design

2.1 Overview

This document is provided to give developers more information on the current drivers and how they can be modified to suit their needs.

The following appendices are provided for reference use. They describe the function prototypes for all the driver's functions, complete with possible return values, a small code sample, and a brief description on how to use the function.

 © PLX Technology, Inc., 1999

This page is intentionally left blank.

 © PLX Technology, Inc., 1999

3. Driver Functions

This section describes the functions used by each driver that are not related directly to the PCI API functions. Each function call will follow the following format.

Sample Function Entry

Syntax:

`function(modifier` *`parameter`*`[,...]);`
This gives the declaration syntax for each function. Each parameter is *italicized*.

Description:

Summary of the function's purpose followed by the parameters it takes. It also includes any relevant information pertaining to the function.

Return Value:

The value returned by the function.

Comments:

A sample is provided to demonstrate the function's use.

 © PLX Technology, Inc., 1999

3.1 Windows NT Driver Maintenance Functions

DriverEntry

Syntax:

```
NTSTATUS DriverEntry(IN PDRIVER_OBJECT pDriverObject,
                     IN PUNICODE_STRING pRegistryPath);
```

Description:

Called at system startup, the first entry point for the IOP 480 driver. This function allocates startup tables, enumerates all devices, and creates all device objects for them.

- *pDriverObject* is given by the operating system and points to a DRIVER_OBJECT structure;
- pRegistryPath is given by the operating system and points to a registry location where the driver information for the PLX chip;

Return Value:

Return Value	Description
STATUS_SUCCESS	The function returned successfully.
STATUS_UNSUCCESSFUL	Various Errors See below.
STATUS_INSUFFICIENT_RESOURCES	Internal buffers could not be allocated.

Comments:

Various Errors include:

- Failure to enumerate the PCI bus;
- Failure to properly read the registry; or,
- Failure by Windows NT to allocate the internal object(s) needed to manage the driver.

 © PLX Technology, Inc., 1999

PlxDrvUnload

Syntax:

```
VOID PlxDrvUnload(IN PDRIVER_OBJECT pDriverObject);
```

Description:

The operating system calls this function in the event system is shutting down. It traverses the list of installed devices for the driver and removes all resources given to the device by the operating sytem. Once this is done, the driver itself is removed.

- *pDriverObject* is given by the operating system and points to a DRIVER_OBJECT structure;

Return Value:

Return Value	Description
None	

Comments:

None.

 © PLX Technology, Inc., 1999

PlxCreateDeviceObject

Syntax:

```
NTSTATUS PlxCreateDeviceObject(IN PDRIVER_OBJECT pDriverObject,
                               IN PUNICODE_STRING pRegistryPath,
                               IN PDEVICE_LOCATION pDevice,
                               IN U32 deviceNumber,
                               IN PPCI_COMMON_CONFIG pConfigData)
```

Description:

This function is called each time the Driver recognizes the **Vendor ID and Device ID** of a particular adapter card in the system. This function will attempt to map all address ranges, attach an interrupt, and allocate all Windows NT internal structures needed to manage the device.

- *pDriverObject* is given by the operating system and points to a DRIVER_OBJECT structure;
- *pRegistryPath* is given by the operating system and points to a registry location where the driver information for the PLX chip;
- *pDevice* is given by the operating system and points to a DEVICE_OBJECT structure;
- *deviceNumber* the zero-based enumeration of the IOP 480 device to be created.; and,
- *pConfigData* a pointer to all the PCI configuration data for the bus and slot for this device.

Return Value:

Return Value	Description
STATUS_SUCCESS	The function returned successfully.
STATUS_UNSUCCESSFUL	Various Errors, See below

Comments:

Various Errors include:

- Error codes returned by Windows NT DDK calls that are critical to the operation of the device and have failed.

 © PLX Technology, Inc., 1999

PlxMapInUserSpace

Syntax:

```
U32 PlxMapInUserSpace(IN PHYSICAL_ADDRESS inAddress,
                      IN U32 sizeInBytes,
                      IN BOOLEAN addressSpace);
```

Description:

This function is called to create a user region of memory in which access to the physical addresses is allowed. ***Note:*** *No address validation is made when accessing these ranges from user programs, so caution is advised.*

- *inAddress* the physical address to be mapped;
- *sizeInBytes* the region's length;
- *addressSpace* the function will not work with I/O mapped memory;

Return Value:

Return Value	Description
STATUS_SUCCESS	The function mapped all ranges successfully.
STATUS_UNSUCCESSFUL	A fatal error occurred.

Comments:

None.

 © PLX Technology, Inc., 1999

PlxUnmapUserSpaceAddr

Syntax:

```
VOID PlxUnmapUserSpaceAddr(IN PU32 pUserAddress);
```

Description:

This function is called to remove a user view of kernel memory.

- *pUserAddress* points to the address to be unmapped;

Return Value:

Return Value	Description
None.	

Comments:

None.

 © PLX Technology, Inc., 1999

PlxMapAllUserAddresses

Syntax:

```
NTSTATUS PlxMapAllUserAddresses(IN PDEVICE_EXTENSION pExt,
                                IN PVIRTUAL_ADDRESSES pVirtAddr);
```

Description:

This function examines the Base Address Registers and if they can support user regions in memory, calls **PlxMapInUserSpace()**. It also records the addresses in the device extension.

- *pExt* a pointer to the device extension;
- *pVirtAddr* a pointer to a structure that holds all the user virtual addresses of the Base Address Registers;

Return Value:

Return Value	Description
STATUS_SUCCESS	The function mapped all ranges successfully.

Comments:

None.

 © PLX Technology, Inc., 1999

PlxUnmapAllUserAddresses

Syntax:

```
NTSTATUS PlxUnmapAllUserAddresses( IN PDEVICE_EXTENSION pExt,
                                   IN PVIRTUAL_ADDRESSES VirtAddr);
```

Description:

This function will free the memory used to store the user virtual addresses of all Base Address Registers.

- *pExt* a pointer to the device extension;
- *pVirtAddr* a pointer to a structure that holds all the user virtual addresses of the Base Address Registers;

Return Value:

Return Value	Description
STATUS_SUCCESS	The function removed all user-mapped addresses from the device.

Comments:

None.

 © PLX Technology, Inc., 1999

IsValidXXXXDevice

Syntax:

```
BOOLEAN IsValidXXXXDevice(IN PDEVICE_LOCATION pDevice);
```

Description:

This function reads the **Vendor ID and Device ID** of the device object passed in and compares them to the list of Supported Devices found in the registry. If a match is made, then function returns true, else false.

- *pDevice* is given by the operating system and points to a DEVICE_OBJECT structure;

Return Value:

Return Value	Description
NT_SUCCESS	The function was able to return the list of Vendor and Device IDs.
FALSE	No supported device exists.

Comments:

XXXX refers to the driver type and can be one of the following (9080/9054/480).

 © PLX Technology, Inc., 1999

PlxValidIdRead

Syntax:

```
NTSTATUS PlxValidIdRead(IN PU32 pIdList,
                        OUT PU32 pNumberFound);
```

Description:

This function reads the registry and returns all Supported Vendor ID and Device ID found in the registry. This function works with IsValidXXXXDevice to determine if the PCI card is supported.

- *pIdList* is an array of U32 Vendor and Device Ids; and,
- *pNumberFound* is a pointer indicating the number of valid Vendor Device ID values stored in the array;

Return Value:

Return Value	Description
TRUE	A supported device was found.
FALSE	No supported device exists.

Comments:

None.

 © PLX Technology, Inc., 1999

PlxSearchDevices

Syntax:

```
NTSTATUS PlxSearchDevices(IN PDEVICE_EXTENSION pExt,
                          IN OUT PDEVICE_LOCATION pDeviceInfo,
                          IN OUT PU32 pSelection);
```

Description:

This function will scan the driver's (functional, not physical) data to match the partial data that exists within the DEVICE_LOCATION structure passed in as a pointer. The function will complete the structure with the information of the first matching device.

- *pExt* is a pointer to the extension which contains device specific information;
- *pDeviceInfo* is a pointer to a DEVICE_LOCATION structure; and,
- *pSelection* a pointer to the number of matches found;

Return Value:

Return Value	Description
STATUS_SUCCESS	The function returned successfully.
STATUS_UNSUCCESSFUL	A serious error occurred.

Comments:

None.

 © PLX Technology, Inc., 1999

PlxInitializeEventLog

Syntax:

```
VOID PlxInitializeEventLog(IN PDRIVER_OBJECT pDriverObject);
```

Description:

This function will setup the event viewer access by writing to various registry locations.

- *pDriverObject* is a pointer to the driver object, which is needed to report events;

Return Value:

Return Value	Description
None	

Comments:

None.

 © PLX Technology, Inc., 1999

PlxReportEvent

Syntax:

```
BOOLEAN PlxReportEvent(IN ULONG MessageLevel,
                       IN NTSTATUS ErrorCode,
                       IN PVOID pIoObject,
                       IN PIRP pIrp);
```

Description:

This function will write a message to the Event Viewer. The most basic error message will at least contain a text message and an error code. Both the driver object and the device objects can create events.

- *messageLevel* indicates the importance of the message. Depending on the priority level set in the registry a message may not be displayed;
- *ErrorCode* hold the NTSTATUS error code (if one exists);
- *pIoObject* a pointer used to accept a generic I/O object that could own an Event to be reported;
- *pIrp* a pointer to the IRP for device object that called this function. Some IRP data is passed into the error packet;

Return Value:

Return Value	Description
TRUE	Writing to the event queue was successful.
FALSE	The function was unable to write to the event viewer.

Comments:

None.

 © PLX Technology, Inc., 1999

PlxDumpRegistryInformation

Syntax:

```
NTSTATUS PlxDumpRegistryInformation(OUT PU32 pIdArray,
                                    OUT PU32 pArraySize,
                                    OUT PU32 pCommonBuffer,
                                    OUT PU32 pSglBuffer);
```

Description:

This function reads all the device parameters found in the registry. All the data read is copied to global variables which store universal driver data (error logging level, buffer sizes, etc).

- *pIdArray* is the pointer to the array of Valid **Vendor and Device Ids** that this driver can support.;
- *pArraySize* is a pointer to the maximum element index for the previous array argument;
- *pCommonBuffer* is a pointer to the common buffer size, used in Block DMA transfers, which is stored in the registry; and,
- *pSglBuffer* is a pointer to the Sgl buffer size, which is also stored in the registry;

Return Value:

Return Value	Description
`NT_SUCCESS`	The function was able to gather all the registry information.
`Else`	An error occurred while reading the registry.

Comments:

None.

 © PLX Technology, Inc., 1999

PlxSleep

Syntax:

```
VOID PlxSleep(IN U32 time);
```

Description:

This function stalls the kernel thread for a specified number of milliseconds.

- *time* the number of milliseconds to delay execution;

Return Value:

Return Value	Description
None	

Comments:

None.

 © PLX Technology, Inc., 1999

GetBarIndex

Syntax:

```
U32 GetBarIndex( IN PHYSICAL_ADDRESS address,
                 IN PPCI_COMMON_CONFIG pciRegs);
```

Description:

This function returns the BAR number associated with the physical address passed in.

- *address* possible Base Address Register address and search criteria; and,
- *pciRegs* a pointer to the PCI configuration registers, which store all the BAR addresses.

Return Value:

Return Value	Description
`<Positive Integer>`	Sucessful BAR index.
`-1`	No match found.

Comments:

None.

 © PLX Technology, Inc., 1999

GetNTErrorCode

Syntax:

```
U32 GetNTErrorCode(IN U32 serviceError);
```

Description:

This function converts PlxError Codes into a NTErrorCode for transport to the API.

- *serviceError* PLX error code which are found in the PlxError.h header file;

Return Value:

Return Value	Description
STATUS_SUCCESS	Converted from ApiSuccess
STATUS_UNSUCCESSFUL	Converted from ApiFailed
STATUS_NULL_PARAM	Converted from ApiNullParam
STATUS_DMA_SGL_BUILD_FAILED	Converted from ApiDmaSglBuildFailed
STATUS_DMA_CHANNEL_INVALID	Converted from ApiDmaChannelInvalid
STATUS_DMA_CHANNEL_ UNAVAILABLE	Converted from ApiDmaChannelUnavailable
STATUS_DMA_IN_PROGRESS	Converted from ApiDmaInProgress

Comments:

These error codes could not be passed back by the primary method of data transfer between the Kernel and the User level, the IOCTL data structure, so the are now passed via custom return codes and are converted back to PLX error codes in the PCI API.

 © PLX Technology, Inc., 1999

3.2 Windows NT Interrupt functions

plxIsr

Syntax:

```
BOOLEAN plxIsr(IN PKINTERRUPT pInterrupt,
               IN PVOID pDeviceExtension);
```

Description:

The Interrupt Service Routine called by the operating system when the attached interrupt line is active. This function will scan the device for various sources of PCI interrupts, (DMA channels, Doorbell Register, Abort conditions). If an interrupt is serviced, then **plxIsr()** will return TRUE, else the return value is FALSE.

- *pInterrupt* a pointer to the interrupt object that caused the interrupt; and,
- *pDeviceExtension* a pointer to the device's specific data. This is used to service the interrupt;

Return Value:

Return Value	Description
TRUE	This device caused the interrupt and it was serviced.
FALSE	This device did not cause the interrupt and it still must be serviced. (This is possible as PCI interrupt can be shared)

Comments:

None.

 © PLX Technology, Inc., 1999

plxDpc

Syntax:

```
VOID plxDpc(IN PKDPC pDpc,
            IN PVOID pContext,
            IN PVOID pArg1,
            IN PVOID pArg2);
```

Description:

The DPC function is called by the operating system when processing is required that must be sequential but not immediate. The ISR will use this function to finish servicing the interrupts after it finds them.

- *pDpc* a pointer to the DPC object;
- *pContext* a pointer to the device's specific context information which can be setup at DPC initialization; and,
- *pArg1*, *pArg2* argument for this function that are passed in when DPC is queued.

Return Value:

Return Value	Description
None	

Comments:

The Dpc must handle **all** interrupts that are outstanding, including ones that have been signaled while the Dpc was in the queue waiting to be executed. This is because two Dpc requests cannot be in the queue at the same time.

 © PLX Technology, Inc., 1999

PlxUnmapSglTransfer

Syntax:

```
VOID PlxUnmapSglTransfer( IN PDEVICE_EXTENSION pExtension,
                          IN DMA_CHANNEL dmaChannel);
```

Description:

This function is called by the DPC function when its has determined that a SGL DMA transfer has just finished. The purpose of this function is to complete the pending IRP that was associated with the DMA transfer, as well as free any system resources and internal buffers that were being used by the DMA transfer (the location of the DMA descriptors, for example).

- *pExtension* is a pointer to the device extension; and,
- *dmaChannel* informs the function of which IRP to release and which resources to free;

Return Value:

Return Value	Description
None	

Comments:

This function must be run within a spinlock on the DMA channel, as only one transfer can occur per channel.

 © PLX Technology, Inc., 1999

PlxUnmapBlockTransfer

Syntax:

```
VOID PlxUnmapBlockTransfer(IN PDEVICE_EXTENSION pExtension,
                           IN DMA_CHANNEL dmaChannel);
```

Description:

This function is called by the DPC function when its has determined that a Block DMA transfer has just finished. The purpose of this function is to complete the pending IRP that was associated with the DMA transfer.

- *pExtension* is a pointer to the device extension; and,
- *dmaChannel* informs the function of which IRP to release and which resources to free;

Return Value:

Return Value	Description
None	

Comments:

This function must be run within a spinlock on the DMA channel, as only one transfer can occur per channel.

 © PLX Technology, Inc., 1999

3.3 Windows NT Buffer Functions

DriverBufferInit

Syntax:

```
BOOLEAN DriverBufferInit(IN U32 length);
```

Description:

Allocates an internal buffer and determines the physical address ranges that it is comprised of.

- *length* the buffers length in bytes;

Return Value:

Return Value	Description
TRUE	Allocation was successful.
FALSE	The function could not create a buffer.

Comments:

None.

 © PLX Technology, Inc., 1999

DriverBufferTerminate

Syntax:

```
VOID DriverBufferTerminate(VOID);
```

Description:

Terminates the internal buffer;

Return Value:

Return Value	Description
None	

Comments:

None.

 © PLX Technology, Inc., 1999

DriverBufferMalloc

Syntax:

```
PVOID DriverBufferMalloc(U32 size);
```

Description:

Allocates an internal buffer and determines the physical address ranges that it is comprised of. The data is reserved on 16-byte boundary.

- *size* the amount of bytes to reserve out of the internal buffer;

Return Value:

Return Value	Description
`Address`	Allocation was successful.
`NULL`	The function could not create a buffer.

Comments:

None.

 © PLX Technology, Inc., 1999

DriverBufferFree

Syntax:

```
VOID DriverBufferFree(PVOID pointer);
```

Description:

Frees a segment of memory allocated previously using **DriverBufferMalloc()**.

- *pPointer* the address of the buffer that was allocated;

Return Value:

Return Value	Description
None	

Comments:

None.

 © PLX Technology, Inc., 1999

DriverBufferVaToPhys

Syntax:

```
PVOID DriverBufferVaToPhys(PVOID vaAddress);
```

Description:

Determines the physical location in memory of a virtual memory segment.

- *pVaAddress* the virtual address of a memory segment;

Return Value:

Return Value	Description
None	

Comments:

None.

 © PLX Technology, Inc., 1999

DriverBufferFlush

Syntax:

```
VOID DriverBufferFlush(VOID);
```

Description:

Puts all cached data of the internal buffer into memory.

Return Value:

Return Value	Description
None	

Comments:

None.

 © PLX Technology, Inc., 1999

3.4 Windows NT Dispatch functions

PlxDrvDispatch

Syntax:

```
NTSTATUS PlxDrvDispatch(IN PDEVICE_OBJECT pDeviceObject,
                        IN PIRP pIrp);
```

Description:

The operating system calls this function in the event any major Win32 function is called using a handle to a symbolic device that this driver has created. The DriverEntry function allocated a table that causes this function to be called for IRP_MJ_READ, IRP_MJ_WRITE, and IRP_MJ_DEVICE_CONTROL functions. This function serves as a giant case statement validating the user level instruction, then calling the appropriate kernel level functions.

- *pDeviceObject* is given by the operating system and points to this DEVICE_OBJECT structure; and,
- pIrp is given by the operating system and points to an IRP which holds all the I/O information needed to process the message;

Return Value:

Return Value	Description
STATUS_SUCCESS	The function returned successfully.
STATUS_UNSUCCESSFUL	Various Errors, See below.
STATUS_PENDING	The I/O Request will be finished later.

Comments:

This function must handle synchronous and non-synchronouse I/O requests. This is done by servicing all the former within the switch and falling through for the remainer of the I/O requests. All asynchronous I/O operation have the IRP returned as STATUS_PENDING.

Almost all Failure Values are passed through the IOCTL data structure, with the acutal I/O request completing successfully. The exception to this is for IRPs that can be done in a non-blocking fashion. These function convert their error codes into NTErrorCodes and allow the PCI API to convert them back.

 © PLX Technology, Inc., 1999

PlxProcessIO

Syntax:

```
BOOLEAN PlxProcessIO( IN PDEVICE_EXTENSION pDeviceExtension,
                      IN PIRP pIrp);
```

Description:

Execute all Service Functions that have the capacity to do asynchronous I/O operations.

- *pDeviceExtension* pointer to the device extension, which is needed to be passed to the various service functions; and,
- *pIrp* a pointer to the IRP which this I/O operation belongs

Return Value:

Return Value	Description
TRUE	The I/O operation did not fail.
FALSE	The I/O operation did fail.

Comments:

None.

 © PLX Technology, Inc., 1999

PlxIrpQueueCancel

Syntax:

```
VOID PlxIrpQueueCancel( IN PDEVICE_OBJECT pDeviceObject,
                        IN PIRP pIrp);
```

Description:

Registered callback function used to hande the cancellation of a specific IRP.

- *pDeviceObject* pointer to the device object structure, which is a mandatory argument for this callback function; and,
- *pIrp* a pointer to the IRP which this I/O operation belongs and is also required for this callback.

Return Value:

Return Value	Description
None.	

Comments:

This function will check to see if this IRP was handling a DMA request. If so then it will de-allocate any resources that are necessary.

 © PLX Technology, Inc., 1999

PlxIrpQueueCancelAll

Syntax:

```
VOID PlxIrpQueueCancelAll( IN PDEVICE_OBJECT pDeviceObject);
```

Description:

This function will complete any pending IRPs and free all resources held by them. This function is only called when the driver is being removed.

- *pDeviceObject* pointer to the device object structure,

Return Value:

Return Value	Description
None.	

Comments:

None.

 © PLX Technology, Inc., 1999

PlxCompleteIrp

Syntax:

```
VOID PlxCompleteIrp(IN NTSTATUS ntStatus,
                    IN PVOID pPendingIrp);
```

Description:

This function will complete the IRPs passed in by calling the DDK function **IoCompleteRequest()**. This function should only be used for IRP that do not have a cancel function *(see below)*.

- *ntStatus* is the NT return value to send back to the API; and,
- *pPendingIrp* is the pointer to the IRP to be completed.

Return Value:

Return Value	Description
None.	

Comments:

The function IoSetCancelRoutine() sets the callback for IRP cancellation. If this function is called, the previous function **must** be called first with the callback parameter set to NULL.

 © PLX Technology, Inc., 1999

PlxCancelAttachIrp

Syntax:

```
VOID PlxCancelAttachIrp(IN PDEVICE_OBJECT pDeviceObject,
                        IN PIRP pIrp);
```

Description:

This function will complete any pending IRPs that the PlxAttach function left in a pending state.

- *pDeviceObject* pointer to the device object structure; and,
- *pIrp* is a pointer to the IRP to be cancelled.

Return Value:

Return Value	Description
None.	

Comments:

The function IoSetCancelRoutine() sets the callback for IRP cancellation. If this function is called, the previous function **must** be called first with the callback parameter set to NULL.

 © PLX Technology, Inc., 1999

3.5 Windows 98 Driver Functions

DriverEntry

Syntax:

```
NTSTATUS DriverEntry(IN PDRIVER_OBJECT pDriverObject,
                     IN PUNICODE_STRING pRegistryPath);
```

Description:

Called at system startup, this is the first entry point for the PLX driver. This function allocates startup tables, and initializes the internal buffer.

- *pDriverObject* is given by the operating system and points to a DRIVER_OBJECT structure; and,
- pRegistryPath is given by the operating system and points to a registry location where the driver information for the PLX chip;

Return Value:

Return Value	Description
STATUS_SUCCESS	The function returned successfully.

Comments:

None.

 © PLX Technology, Inc., 1999

PlxDrvUnload

Syntax:

```
VOID PlxDrvUnload(IN PDRIVER_OBJECT pDriverObject);
```

Description:

The operating system calls this in the event that all devices for this driver are removed. Therefore the only work this function must handle is the termination of the internal memory manager.

- *pDriverObject* is given by the operating system and points to a DRIVER_OBJECT structure;

Return Value:

Return Value	Description
None	

Comments:

None.

 © PLX Technology, Inc., 1999

PlxCreateDeviceObject

Syntax:

```
NTSTATUS PlxCreateDeviceObject( IN PDRIVER_OBJECT pDriverObject,
                                IN PDEVICE_LOCATION *DeviceObject);
```

Description:

This function will create the Device Object and the Win32 link to the device and associate the functional device object with the physical device object. It also initializes the device object by setting various internal values.

- *pDriverObject* is given by the operating system and points to a DRIVER_OBJECT structure; and,
- *pDevice* is given by the operating system and points to a DEVICE_OBJECT structure;

Return Value:

Return Value	Description
STATUS_SUCCESS	The function returned successfully.
STATUS_INSUFFICIENT_ RESOURCES	Could not get enough memory for internal objects.
STATUS_UNSUCCESSFUL	Various internal errors.

Comments:

None.

 © PLX Technology, Inc., 1999

PlxAllocateCommonBuffer

Syntax:

```
NTSTATUS PlxAllocateCommonBuffer( IN PDEVICE_OBJECT pdo,
                                  IN PDEVICE_EXTENSION pDeviceExt);
```

Description:

This function will allocate the common buffer and the Adapter Object that is using this buffer.

- *pdo* the physical device object; and,
- *pDeviceExt* the device extension, needed to store the allocated buffer.

Return Value:

Return Value	Description
STATUS_SUCCESS	Common buffer allocated successfully.
STATUS_INSUFFICIENT_ RESOURCES	Low system memory.
STATUS_UNSUCCESSFUL	Invalid length given for the common buffer.

Comment:

The common buffer size is stored within the registry.

 © PLX Technology, Inc., 1999

PlxDeallocateCommonBuffer

Syntax:

```
VOID PlxDeallocateCommonBuffer(VOID);
```

Description:

- This function will free the common buffer and the Adapter Object that is using this buffer.

Return Value:

Return Value	Description
None	

Comment:

None.

 © PLX Technology, Inc., 1999

AddDevice

Syntax:

```
NTSTATUS AddDevice( IN PDRIVER_OBJECT DriverObject,
                    IN PDEVICE_OBJECT pdo);
```

Description:

AddDevice() is called automatically when a device is found that must be added to the system. This function will first call **PlxCreateDeviceObject()**. If this call is successful, then the function will initialize the Common Buffer if this is the first device found for the driver.

- *DriverObject* is a pointer to the Windows Driver Object structure needed to add the device to the system; and,
- *pdo* is a pointer to a physical device object, which represents the actual device to be added to the system;

Return Value:

Return Value	Description
STATUS_SUCCESS	The device was successfully added.

Comments:

None.

 © PLX Technology, Inc., 1999

CompleteRequest

Syntax:

```
NTSTATUS CompleteRequest(IN PIRP Irp,
                         IN NTSTATUS status,
                         IN U32 info);
```

Description:

CompleteRequest will complete the IRP that was passed in to it. This function assumes that the IRP was in a pending state when passed in.

- *pPendingIrp* is a pointer to the IRP to be completed;
- *status* sets the status of the IRP to be completed; and,
- *info* sets the information field of the IRP to be completed;

Return Value:

Return Value	Description
`status`	The value passed in.

Comments:

None.

 © PLX Technology, Inc., 1999

ForwardAndWait

Syntax:

```
NTSTATUS ForwardAndWait(IN PDEVICE_OBJECT fdo,
                        IN PIRP Irp);
```

Description:

Forward and wait passes an IRP down the driver layer stack. It waits until a response returns.

- *fdo* the functional device object, used to obtain the device Extension; and,
- *pIrp* is a pointer to the IRP to be completed;

Return Value:

Return Value	Description
`Status`	The status of the IRP that was returned.

Comments:

None.

 © PLX Technology, Inc., 1999

LockDevice

Syntax:

```
BOOLEAN LockDevice(PDEVICE_EXTENSION pdx);
```

Description:

Allows a kernel thread to assign use to the device atomically (Mirrors a Semaphore). If the usage number falls to 0, the device removal event is signalled.

- *pdx* the device Extension;

Return Value:

Return Value	Description
TRUE	The device could be locked.
FALSE	The device is being removed, it cannot be locked for use.

Comments:

None.

 © PLX Technology, Inc., 1999

UnlockDevice

Syntax:

```
VOID UnlockDevice(PDEVICE_EXTENSION pdx);
```

Description:

Allows a kernel thread to give up the use of a physical device (mirrors a Semaphore). If the usage number falls to 0, the device removal event is signalled.

- *pdx* the device Extension;

Return Value:

Return Value	Description
None.	

Comments:

None.

 © PLX Technology, Inc., 1999

OnRequestComplete

Syntax:

```
NTSTATUS OnRequestComplete(IN PDEVICE_OBJECT fdo,
                           IN PIRP Irp,
                           IN PKEVENT pev);
```

Description:

This is a callback function for when a lower level driver completes an IRP. OnRequestComplete will set an event which is being polled for higher up the call stack.

- *fdo* the function device object, is not used.
- *pIrp* is a pointer to the IRP to be completed, this is not used;
- *pev* gets the status of the IRP to be completed;

Return Value:

Return Value	Description
`status`	The value passed in.

Comments:

None.

 © PLX Technology, Inc., 1999

RemoveDevice

Syntax:

```
VOID RemoveDevice(IN PDEVICE_OBJECT fdo);
```

Description:

This function removes the function device object associated with a physcial device object (IOP 480) on the system. If this is the last functional device on the system, the common buffer is released as well.

- *pDriverObject* points to a DRIVER_OBJECT structure;

Return Value:

Return Value	Description
None	

Comments:

None.

 © PLX Technology, Inc., 1999

PlxDumpRegistryInformation

Syntax:

```
NTSTATUS PlxDumpRegistryInformation(OUT PU32 pIdArray,
                                    OUT PU32 pArraySize,
                                    OUT PU32 pCommonBuffer,
                                    OUT PU32 pSglBuffer);
```

Description:

This function reads all the device parameters found in the registry. All the data read is copied to global variables which store universal driver data (error logging level, buffer sizes, etc).

- *pIdArray* is the pointer to the array of Valid **Vendor and Device Ids** that this driver can support.;
- *pArraySize* is a pointer to the maximum element index for the previous array argument;
- *pCommonBuffer* is a pointer to the common buffer size, used in Block DMA transfers, which is stored in the registry; and,
- *pSglBuffer* is a pointer to the Sgl buffer size, which is also stored in the registry;

Return Value:

Return Value	Description
`NT_SUCCESS`	The function was able to gather all the registry information.
`Else`	An error occurred while reading the registry.

Comments:

None.

 © PLX Technology, Inc., 1999

GetNTErrorCode

Syntax:

```
U32 GetNTErrorCode(U32 serviceError);
```

Description:

This function converts PlxError Codes into a NTErrorCode for transport to the API.

- *serviceError* PLX error code which are found in the PlxError.h header file;

Return Value:

Return Value	Description
STATUS_SUCCESS	Converted from ApiSuccess
STATUS_UNSUCCESSFUL	Converted from ApiFailed
STATUS_NULL_PARAM	Converted from ApiNullParam
STATUS_DMA_SGL_BUILD_FAILED	Converted from ApiDmaSglBuildFailed
STATUS_DMA_CHANNEL_INVALID	Converted from ApiDmaChannelInvalid
STATUS_DMA_CHANNEL_ UNAVAILABLE	Converted from ApiDmaChannelUnavailable
STATUS_DMA_IN_PROGRESS	Converted from ApiDmaInProgress

Comments:

These error codes could not be passed back by the primary method of data transfer between the Kernel and the User level, the IOCTL data structure, so the are now passed via custom return codes and are converted back to PLX error codes in the PCI API.

 © PLX Technology, Inc., 1999

PlxReportEvent

Syntax:

```
BOOLEAN PlxReportEvent(IN ULONG MessageLevel,
                       IN NTSTATUS ErrorCode,
                       IN PVOID pIoObject,
                       IN PIRP pIrp);
```

Description:

This function will write a message to the Event Viewer. The most basic error message will at least contain a text message and an error code. Both the driver object and the device objects can create events.

- *messageLevel* indicates the importance of the message. Depending on the priority level set in the registry a message may not be displayed;
- *ErrorCode* hold the NTSTATUS error code (if one exists);
- *pIoObject* a pointer to any NT Driver Object that could own a event to be reported; and,
- *pIrp* a pointer to the IRP for device object that called this function. Some IRP data is passed into the error packet;

Return Value:

Return Value	Description
TRUE	Writing to the event queue was successful.
FALSE	The function was unable to write to the event viewer.

Comments:

None.

 © PLX Technology, Inc., 1999

PlxSleep

Syntax:

```
VOID PlxSleep(U32 time);
```

Description:

This function stalls the kernel thread for a specified number of milliseconds.

- *time* the number of milliseconds to delay execution;

Return Value:

Return Value	Description
None	

Comments:

None.

 © PLX Technology, Inc., 1999

3.6 Windows 98 Interrupt functions

PlxIsr

Syntax:

```
BOOLEAN PlxIsr(IN PKINTERRUPT pInterrupt, IN PVOID pDeviceExtension);
```

Description:

The Interrupt Service Routine called by the operating system when the attached interrupt line is active. This function will scan the device for various sources of PCI interrupts, (DMA channels, Doorbell Register, Abort conditions). If an interrupt is serviced, then PlxIsr() will return TRUE, else the return value is FALSE.

- *pInterrupt* a pointer to the interrupt object that caused the interrupt; and,
- *pDeviceExtension* a pointer to the device's specific data. This is used to service the interrupt;

Return Value:

Return Value	Description
TRUE	This device caused the interrupt and it was serviced.
FALSE	This device did not cause the interrupt and it still must be serviced. (This is possible as PCI interrupt can be shared)

Comments:

None.

 © PLX Technology, Inc., 1999

PlxDpc

Syntax:

```
VOID PlxDpc(IN PKDPC pDpc,
            IN PVOID pContext,
            IN PVOID pArg1,
            IN PVOID pArg2);
```

Description:

The DPC function is called by the operating system when processing is required that must be sequential but not immediate. The ISR will use this function to finish servicing the interrupts after it finds them.

- *pDpc* a pointer to the DPC object;
- *pContext* a pointer to the device's specific context information which can be setup at DPC initialization; and,
- *pArg1*, *pArg2* argument for this function that are passed in when DPC is queued.

Return Value:

Return Value	Description
none	

Comments:

None.

 © PLX Technology, Inc., 1999

PlxUnmapSglTransfer

Syntax:

```
VOID PlxUnmapSglTransfer( IN PDEVICE_EXTENSION pExtension,
                          IN DMA_CHANNEL dmaChannel);
```

Description:

This function is called by the DPC function when its has determined that a SGL DMA transfer has just finished. The purpose of this function is to complete the pending IRP that was associated with the DMA transfer, as well as free any system resources and internal buffers that were being used by the DMA transfer (the location of the DMA descriptors, for example).

- *pExtension* is a pointer to the device extension; and,
- *dmaChannel* informs the function of which IRP to release and which resources to free;

Return Value:

Return Value	Description
None	

Comments:

This function must be run within a spinlock on the DMA channel, as only one transfer can occur per channel.

 © PLX Technology, Inc., 1999

PlxUnmapBlockTransfer

Syntax:

```
VOID PlxUnmapBlockTransfer(IN PDEVICE_EXTENSION pExtension,
                           IN DMA_CHANNEL dmaChannel);
```

Description:

This function is called by the DPC function when its has determined that a Block DMA transfer has just finished. The purpose of this function is to complete the pending IRP that was associated with the DMA transfer.

- *pExtension* is a pointer to the device extension; and,
- *dmaChannel* informs the function of which IRP to release and which resources to free;

Return Value:

Return Value	Description
None	

Comments:

This function must be run within a spinlock on the DMA channel, as only one transfer can occur per channel.

 © PLX Technology, Inc., 1999

3.7 Windows 98 Buffer Functions

DriverBufferInit

Syntax:

```
BOOLEAN DriverBufferInit(U32 length);
```

Description:

Allocates an internal buffer and determines the physical address ranges that it is comprised of.

- *length* the buffers length in bytes;

Return Value:

Return Value	Description
TRUE	Allocation was successful.
FALSE	The function could not create a buffer.

Comments:

None.

 © PLX Technology, Inc., 1999

DriverBufferTerminate

Syntax:

```
VOID DriverBufferTerminate(VOID);
```

Description:

Terminates the internal buffer;

Return Value:

Return Value	Description
none	

Comments:

None.

 © PLX Technology, Inc., 1999

DriverBufferMalloc

Syntax:

```
PVOID DriverBufferMalloc(U32 size);
```

Description:

Allocates an internal buffer and determines the physical address ranges that it is comprised of. The data is reserved on 16 byte boundary.

- *size* the amount of bytes to reserve out of the internal buffer;

Return Value:

Return Value	Description
TRUE	Allocation was successful.
FALSE	The function could not create a buffer.

Comments:

None

 © PLX Technology, Inc., 1999

DriverBufferFree

Syntax:

```
VOID DriverBufferFree(PVOID pointer);
```

Description:

Frees a segment of memory allocated previously using DriverBufferMalloc().

- *pPointer* the address of the buffer that was allocated;

Return Value:

Return Value	Description
None	

Comments:

None.

 © PLX Technology, Inc., 1999

DriverBufferVaToPhys

Syntax:

```
PVOID DriverBufferVaToPhys(PVOID vaAddress);
```

Description:

Determines the physical location in memory of a virtual memory segment.

- *pVaAddress* the virtual address of a memory segment;

Return Value:

Return Value	Description
None	

Comments:

None.

 © PLX Technology, Inc., 1999

DriverBufferFlush

Syntax:

```
VOID DriverBufferFlush(VOID);
```

Description:

Puts all cached data of the internal buffer into memory.

Return Value:

Return Value	Description
None	

Comments:

None.

 © PLX Technology, Inc., 1999

3.8 Windows 98 Plug and Play Functions

RequestPnp

Syntax:

```
NTSTATUS RequestPnp(IN PDEVICE_OBJECT fdo,
                    IN PIRP pIrp);
```

Description:

This function is called whenever the IRP_MJ_PNP message is sent. Its purpose is to route the message to the correct routine, depending on which minor message is embedded within the request.

- *fdo* the functional device object that this message is bound for; and,
- *pIrp* a pointer to the IRP structure, which holds all the I/O information.

Return Value:

Return Value	Description
`Status`	The return code of the handler function that is called for the specific message.

Comments:

None.

 © PLX Technology, Inc., 1999

DefaultPnpHandler

Syntax:

```
NTSTATUS DefaultPnpHandler(IN PDEVICE_OBJECT fdo,
                           IN PIRP pIrp);
```

Description:

Will handle the PNP requests that do not require special processing.

- *fdo* the functional device object passed from the RequestPnp() function; and,
- *pIrp* a pointer to the IRP structure that was passed from the RequestPnp() function;

Return Value:

Return Value	Description
`Status`	Returns the ntStatus value from lower drivers.

Comments:

None.

 © PLX Technology, Inc., 1999

HandleRemoveDevice

Syntax:

```
NTSTATUS HandleRemoveDevice(IN PDEVICE_OBJECT fdo,
                            IN PIRP Irp);
```

Description:

Handles the Plug and Play message that a functional device is to be deleted.

- *fdo* the functional device object passed from the RequestPnp() function; and,
- *pIrp* a pointer to the IRP structure that was passed from the RequestPnp() function;

Return Value:

Return Value	Description
`Status`	Returns ntStatus values from lower layer devices.

Comments:

None.

 © PLX Technology, Inc., 1999

PlxReadConfigReg

Syntax:

```
NTSTATUS PlxReadConfigReg(IN PDEVICE_OBJECT fdo,
                          U32 offset,
                          U32 length,
                          PVOID buffer);
```

Description:

Processes the request to read the PCI configuration registers on the IOP 480.

- *fdo* the functional device object passed from the RequestPnp() function;
- *offset* the PCI offset to begin reading from;
- *length* the number of bytes to read from the PCI configuration registers; and,
- *buffer* the address to write the data to;

Return Value:

Return Value	Description
STATUS_SUCCESS	The function was able to read the PCI configuration registers.
STATUS_INSUFFICIENT_ RESOURCES	The function was unable to allocate space for internal structures.

Comments:

None.

 © PLX Technology, Inc., 1999

PlxWriteConfigReg

Syntax:

```
NTSTATUS PlxWriteConfigReg(IN PDEVICE_OBJECT fdo,
                           U32 offset,
                           U32 length,
                           PVOID buffer);
```

Description:

Processes the request to write to the PCI configuration registers on the IOP 480.

- *fdo* the functional device object passed from the RequestPnp() function;
- *offset* the PCI offset to begin reading from;
- *length* the number of bytes to read from the PCI configuration registers; and,
- *buffer* the address of the data to be written;

Return Value:

Return Value	Description
STATUS_SUCCESS	The function was able to write to the PCI configuration registers.
STATUS_INSUFFICIENT_ RESOURCES	The function was unable to allocate space for internal structures.

Comments:

None.

 © PLX Technology, Inc., 1999

HandleStartDevice

Syntax:

```
NTSTATUS HandleStartDevice(IN PDEVICE_OBJECT fdo,
                           IN PIRP Irp);
```

Description:

This processes the message to start a device. It passes the message down the stack then calls the StartDevice() function.

- *fdo* the functional device object passed from the RequestPnp() function; and,
- *pIrp* a pointer to the IRP structure that was passed from the RequestPnp() function;

Return Value:

Return Value	Description
STATUS_SUCCESS	The function was able to write to the PCI configuration registers.
STATUS_INSUFFICIENT_ RESOURCES	The function was unable to allocate space for internal structures.

Comments:

None.

 © PLX Technology, Inc., 1999

HandleStopDevice

Syntax:

```
NTSTATUS HandleStopDevice(IN PDEVICE_OBJECT fdo,
                          IN PIRP Irp);
```

Description:

This processes the message to stop a device. It passes the message down the stack then calls the StopDevice() function.

- *fdo* the functional device object passed from the RequestPnp() function; and,
- *pIrp* a pointer to the IRP structure that was passed from the RequestPnp() function;

Return Value:

Return Value	Description
STATUS_SUCCESS	The function was able to write to the PCI configuration registers.
STATUS_INSUFFICIENT_ RESOURCES	The function was unable to allocate space for internal structures.

Comments:

None.

 © PLX Technology, Inc., 1999

ShowResources (DEBUG only)

Syntax:

```
VOID ShowResources(IN PCM_PARTIAL_RESOURCE_LIST list);
```

Description:

Displays the hardware resources assigned to this device.

- *list* a pointer to the resource list structure that contains all the information on resource allocation;

Return Value:

Return Value	Description
none	

Comments:

None.

 © PLX Technology, Inc., 1999

3.9 Windows 98 Power Functions

RequestPower

Syntax:

```
NTSTATUS RequestPower(IN PDEVICE_OBJECT fdo,
                      IN PIRP Irp);
```

Description:

This function is called whenever the IRP_MJ_POWER message is sent. Its purpose is to route the message to the correct routine, depending on which minor message is embedded within the request.

- *fdo* the functional device object that this message is bound for; and,
- *pIrp* a pointer to the IRP structure, which holds all the I/O information.

Return Value:

Return Value	Description
`status`	The return code of the handler function that is called for the specific message.

Comments:

None.

 © PLX Technology, Inc., 1999

DefaultPowerHandler

Syntax:

```
NTSTATUS DefaultPowerHandler(IN PDEVICE_OBJECT fdo,
                             IN PIRP pIrp);
```

Description:

Will handle the PNP requests that do not require special processing.

- *fdo* the functional device object passed from the RequestPnp() function; and,
- *pIrp* a pointer to the IRP structure that was passed from the RequestPnp() function;

Return Value:

Return Value	Description
`Status`	Returns the ntStatus value from lower drivers.

Comments:

None.

 © PLX Technology, Inc., 1999

HandleSetPower

Syntax:

```
NTSTATUS HandleSetPower(IN PDEVICE_OBJECT fdo,
                        IN PIRP Irp);
```

Description:

Handles the Plug and Play message that the power level is to be changed.

- *fdo* the functional device object passed from the RequestPnp() function; and,
- *pIrp* a pointer to the IRP structure that was passed from the RequestPnp() function;

Return Value:

Return Value	Description
`Status`	Returns ntStatus values from lower layer devices.

Comments:

None.

 © PLX Technology, Inc., 1999

HandleQueryPower

Syntax:

```
NTSTATUS HandleQueryPower(IN PDEVICE_OBJECT fdo,
                          IN PIRP Irp);
```

Description:

Processes the request to read the current power state of the board.

- *fdo* the functional device object passed from the RequestPnp() function;
- *pIrp* a pointer to the IRP structure that was passed from the RequestPnp() function;

Return Value:

Return Value	Description
STATUS_SUCCESS	The function was able to return the current power level.

Comments:

None.

 © PLX Technology, Inc., 1999

OnFinishPowerUp

Syntax:

```
NTSTATUS OnFinishPowerUp(IN PDEVICE_OBJECT fdo,
                         IN PIRP pIrp,
                         IN PVOID junk);
```

Description:

Processes the request to power a device up due to an increase in system power level.

- *fdo* the functional device object passed from the RequestPnp() function;
- *pIrp* a pointer to the IRP structure that was passed from the RequestPnp() function; and,
- *junk* not used;

Return Value:

Return Value	Description
STATUS_SUCCESS	The function was able to write to the PCI configuration registers.
STATUS_INSUFFICIENT_ RESOURCES	The function was unable to allocate space for internal structures.

Comments:

None.

 © PLX Technology, Inc., 1999

SendDeviceSetPower

Syntax:

```
NTSTATUS SendDeviceSetPower(IN PDEVICE_OBJECT fdo,
                            IN DEVICE_POWER_STATE state,
                            U32 context);
```

Description:

This function is used to send itself a message to change the device's power state.

- *fdo* the functional device object passed from the RequestPnp() function;
- *state* the desired power state; and,
- *context* device specific information passed on to a lower level driver.

Return Value:

Return Value	Description
`Status`	Passed up from lower level drivers.

Comments:

None.

 © PLX Technology, Inc., 1999

SendSelfSetPowerRequest

Syntax:

```
NTSTATUS SendSelfSetPowerRequest(IN PDEVICE_OBJECT fdo,
                                 IN DEVICE_POWER_STATE state);
```

Description:

Uses the RDK to actually set the power level of the device.

- *fdo* the functional device object passed from the RequestPnp() function;
- *state* the desired power state;

Return Value:

Return Value	Description
STATUS_SUCCESS	The function was able to change the power state.

Comments:

None.

 © PLX Technology, Inc., 1999

OnPowerRequestComplete

Syntax:

```
VOID OnPowerRequestComplete(IN PDEVICE_OBJECT pDeviceObject,
                            IN U8 MinorFunction,
                            IN POWER_STATE PowerState,
                            IN PVOID context,
                            IN PIO_STATUS_BLOCK ioStatus);
```

Description:

Displays the hardware resources assigned to this device.

- *pDeviceObject* a pointer to the DEVICE_OBJECT structure for which this request belongs;
- *MinorFunction* is the type of message under IRP_MJ_POWER;
- *PowerState* is the desired power state;
- *context* is the specific device information for this message; and,
- *ioStatus* gives the status of the IRP message being processed;

Return Value:

Return Value	Description
none	

Comments:

None.

 © PLX Technology, Inc., 1999

3.10 Windows 98 Read/Write Functions

DisableDevice

Syntax:

```
BOOLEAN DisableDevice(IN PDEVICE_EXTENSION pdx);
```

Description:

This function is called to disable all PCI interrupts to the device.

- *pdx* a pointer to a device extension that holds specific device information;

Return Value:

Return Value	Description
TRUE	

Comments:

None.

 © PLX Technology, Inc., 1999

EmpowerDevice

Syntax:

```
BOOLEAN EmpowerDevice(IN PPOWER_STUFF pps);
```

Description:

Will set the power level of the device found in the POWER_STUFF structure.

- *pps* a pointer to the POWER_STUFF structure;

Return Value:

Return Value	Description
TRUE	The device power state was changed.

Comments:

None.

 © PLX Technology, Inc., 1999

EnableDevice

Syntax:

```
BOOLEAN EnableDevice(IN PDEVICE_EXTENSION pdx);
```

Description:

Enables all interrupts for the selected device.

- *pdx* a pointer to the device extension that hold specific device information;

Return Value:

Return Value	Description
TRUE	The interrupts were enabled.

Comments:

None.

 © PLX Technology, Inc., 1999

PlxGetBusSlotNumber

Syntax:

```
NTSTATUS PlxGetBusSlotNumber(PDEVICE_OBJECT pdo,
                             PDEVICE_EXTENSION pdx);
```

Description:

Returns the bus number and slot number of the supported device (pdo).

- *pdo* the physical device object which is being queried;
- *pdx* a pointer to the device extension struture which keeps the bus number and slot number information;

Return Value:

Return Value	Description
STATUS_SUCCESS	The function was able to read the PCI configuration registers.

Comments:

None.

 © PLX Technology, Inc., 1999

RequestCreate

Syntax:

```
NTSTATUS RequestCreate(IN PDEVICE_OBJECT fdo,
                       IN PIRP Irp);
```

Description:

Handles the IRP_MJ_CREATE message. This function does no special processing on this message, it just returns STATUS_SUCCESS.

- *fdo* the functional device object passed from the RequestPnp() function;
- *pIrp* a pointer to the IRP structure that was passed from the RequestPnp() function;

Return Value:

Return Value	Description
STATUS_SUCCESS	The function was able to write to the PCI configuration registers.

Comments:

None.

 © PLX Technology, Inc., 1999

RequestClose

Syntax:

```
NTSTATUS RequestClose(IN PDEVICE_OBJECT fdo,
                      IN PIRP Irp);
```

Description:

Handles the IRP_MJ_CLOSE message. This function does no special processing on this message, it just returns STATUS_SUCCESS.

- *fdo* the functional device object passed from the RequestPnp() function;
- *pIrp* a pointer to the IRP structure that was passed from the RequestPnp() function;

Return Value:

Return Value	Description
STATUS_SUCCESS	The function was able to write to the PCI configuration registers.

Comments:

None.

 © PLX Technology, Inc., 1999

RequestWrite

Syntax:

```
NTSTATUS RequestWrite(IN PDEVICE_OBJECT fdo,
                      IN PIRP Irp);
```

Description:

Handles the IRP_MJ_WRITE message. This function does no special processing on this message, it just returns STATUS_SUCCESS.

- *fdo* the functional device object passed from the RequestPnp() function;
- *pIrp* a pointer to the IRP structure that was passed from the RequestPnp() function;

Return Value:

Return Value	Description
STATUS_SUCCESS	The function was able to write to the PCI configuration registers.

Comments:

None.

 © PLX Technology, Inc., 1999

RequestRead

Syntax:

```
NTSTATUS RequestRead(IN PDEVICE_OBJECT fdo,
                     IN PIRP Irp);
```

Description:

Handles the IRP_MJ_READ message. This function does no special processing on this message, it just returns STATUS_SUCCESS.

- *fdo* the functional device object passed from the RequestPnp() function; and,
- *pIrp* a pointer to the IRP structure that was passed from the RequestPnp() function;

Return Value:

Return Value	Description
STATUS_SUCCESS	The function was able to write to the PCI configuration registers.

Comments:

None.

 © PLX Technology, Inc., 1999

SetPowerState

Syntax:

```
VOID SetPowerState(IN PDEVICE_OBJECT fdo,
                   IN DEVICE_POWER_STATE state);
```

Description:

This function is used to send itself a message to change the device's power state.

- *fdo* the functional device object passed from the RequestPnp() function;
- *state* the desired power state;

Return Value:

Return Value	Description
none	

Comments:

None.

 © PLX Technology, Inc., 1999

SetBarIndex

Syntax:

```
U32 GetBarIndex(PHYSICAL_ADDRESS address,
                PCI_COMMON_CONFIG *pciRegs);
```

Description:

This function will scan the available Base Address Registers until a match is made with the physicial address passed in. If successful, the index of the BAR is returned. If the function fails, the return value is –1.

- *address* the physical address to be searched for.;
- *pciRegs* a pointer a structure which hold an image of the PCI Configuration registers.;

Return Value:

Return Value	Description
`Index [0…5]`	A successful search.
`-1`	The BAR value was not found.

Comments:

None.

 © PLX Technology, Inc., 1999

PlxUnmapAll

Syntax:

```
VOID PlxUnmapAll(PDEVICE_EXTENSION pdx);
```

Description:

This function will scan all the available Base Address Registers and unmap user access to these memory ranges, if and only if, they were mapped previously.

- *pdx* a pointer a DEVICE_EXTENSION structure which holds information on the selected device;

Return Value:

Return Value	Description
None	

Comments:

None.

 © PLX Technology, Inc., 1999

StartDevice

Syntax:

```
NTSTATUS StartDevice(IN PDEVICE_OBJECT fdo,
                     IN PCM_PARTIAL_RESOURCE_LIST rawlist,
                     IN PCM_PARTIAL_RESOURCE_LIST list);
```

Description:

This function parses the large resource structure and calls the necessary DDK calls to map memory, attach interrupts, and do any resource allocation that is necessary.

- *fdo* a pointer a DEVICE_OBJECT structure which holds information on the selected device;
- *rawlist* the non-translated resources that are allowed to be used for this device; and,
- *list* the translated resources that are allowed to be used for this device;

Return Value:

Return Value	Description
STATUS_SUCCESS	The resources were allocated successfully.
STATUS_INVALID_DEVICE_STATE	This occurs when the driver cannot enable the device.
STATUS_NO_MEMORY	There are no Base Address Registers

Comments:

None.

 © PLX Technology, Inc., 1999

PlxCancelAllPendingIrp

Syntax:

```
VOID PlxCancelAllPendingIrp(IN PDEVICE_EXTENSION pExtension);
```

Description:

This function cancels any outstanding I/O Requests for the selected device.

- *pExtension* a pointer an DEVICE_EXTENSION structure which holds information on the selected device;

Return Value:

Return Value	Description
none	

Comments:

None.

 © PLX Technology, Inc., 1999

4. Services functions

These functions are the direct interface to the hardware from the PCI API. The interfaces are identical for both the Window NT and Windows 98 drivers. A driver can be created on any platform by linking this object file into the respective driver.

While all function prototypes here have the initial name … they are descriptive of service modules for the IOP480, PCI9054 and PCI9080 chips. Therefore if in the under the Compatibility: heading, the function is compatible with another chip as well, the function name would have the chip name itself prepended to the function name. For example, since the direct slave functionality is supported across all PLX chips, the PCI9054 service module contains Pci9054BusIopRead().

All code samples will be given for the Iop480Service Modules, but usage guidelines apply to the other PLX chips as well.

Service Function Name	Purpose	Comp.
...BusIopRead()	Read the IOP bus.	PCI9080, PCI9054, IOP480
...BusIopWrite()	Write to the IOP bus.	PCI9080, PCI9054, IOP480
...DmaBlockChannelClose()	Close a DMA channel opened for Block DMA.	PCI9080, PCI9054, IOP480
...DmaBlockChannelOpen()	Open a DMA channel for Block DMA.	PCI9080, PCI9054, IOP480
...DmaBlockTransfer()	Control a Block DMA transfer.	PCI9080, PCI9054, IOP480
...DmaBlockTransferRestart()	Redo a Block DMA transfer.	PCI9080, PCI9054, IOP480
...DmaSglChannelClose()	Close a DMA channel opened for SGL DMA.	PCI9080, PCI9054, IOP480
...DmaSglChannelOpen()	Open a DMA channel for SGL DMA.	PCI9080, PCI9054, IOP480
...DmaSglTransfer()	Control a DMA SGL transfer.	PCI9080, PCI9054, IOP480
...DmaShuttleChannelClose()	Close a DMA channel opened for Shuttle DMA.	PCI9080, PCI9054,

 © PLX Technology, Inc., 1999

Service Function Name	Purpose	Comp.
		IOP480
...DmaShuttleChannelOpen()	Open a DMA channel for Shuttle DMA.	PCI9080, PCI9054, IOP480
...DmaShuttleTransfer()	Control a DMA Shuttle transfer.	PCI9080, PCI9054, IOP480
...DmaControl()	Pause, Resume or Abort DMA transfers.	PCI9080, PCI9054, IOP480
...DmaStatus()	Get the status of a DMA transfer.	PCI9080, PCI9054, IOP480
...IntrDisable()	Disable interrupts on the PLX Chip.	PCI9080, PCI9054, IOP480
...IntrEnable()	Enable interrupts on the PLX Chip.	PCI9080, PCI9054, IOP480
...IntrStatusGet()	Get the last interrupt active on the PLX Chip.	PCI9080, PCI9054, IOP480
...IoPortRead()	Read an IO port on the PCI bus.	PCI9080, PCI9054, IOP480
...IoPortWrite()	Write to an IO port on the PCI bus.	PCI9080, PCI9054, IOP480
...MuInboundPortRead()	Read the Inbound Port.	PCI9080, PCI9054, IOP480
...MuInboundPortWrite()	Write to the Inbound Port.	PCI9080, PCI9054, IOP480
...MuOutboundPortRead()	Read the Outbound Port.	PCI9080, PCI9054, IOP480
...MuOutboundPortWrite()	Write to the Outbound Port.	PCI9080, PCI9054, IOP480
...PowerLevelGet()	Get the power level on the PLX Chip.	PCI9080,

 © PLX Technology, Inc., 1999

Service Function Name	Purpose	Comp.
		PCI9054, IOP480
...PowerLevelQuery()	Verify that a power level is supported.	PCI9080, PCI9054
...PowerLevelSet()	Change the power level on the PLX Chip.	PCI9080, PCI9054, IOP480
...RegisterDoorbellRead()	Get the last value of the local-to PCI doorbell register.	PCI9080, PCI9054, IOP480
...RegisterDoorbellSet()	Write to the PCI-to-local doorbell register.	PCI9080, PCI9054, IOP480
...RegisterMailboxRead()	Read a mailbox register.	PCI9080, PCI9054, IOP480
...RegisterMailboxWrite()	Write to a mailbox register.	PCI9080, PCI9054, IOP480
...RegisterRead()	Read a register.	PCI9080, PCI9054, IOP480
...RegisterReadAll()	Read a group of registers.	PCI9080, PCI9054, IOP480
...RegisterWrite()	Write to a register.	PCI9080, PCI9054, IOP480
...SerialEepromRead()	Read the Serial EEPROM.	PCI9080, PCI9054, IOP480
...SerialEepromWrite()	Write to the Serial EEPROM.	PCI9080, PCI9054, IOP480
...UserRead()	Read a USERi# pin.	PCI9080, PCI9054, IOP480
...UserWrite()	Write to a USERo# pin.	PCI9080, PCI9054, IOP480
...HotSwapIdRead()	Read the Hot Swap Id value.	PCI9080,

 © PLX Technology, Inc., 1999

Service Function Name	Purpose	Comp.
		PCI9054
...HotSwapNcpRead()	Read the Hot Swap Next Configuration Pointer value.	PCI9080, PCI9054
...HotSwapStatus()	Read the Hot Swap Status Register.	PCI9080, PCI9054, IOP480
...MuOutboundIndexRead()	Read the Outbound Index Register.	PCI9080
...MuOutboundIndexWrite()	Write to the Outbound Index Register	PCI9080
...AbortAddrRead()	Read the abort address register	PCI9080
...PmIdRead()	Read the Power Management Id value.	PCI9080, PCI9054
...PmNcpRead()	Read the Power Management Next Configuration Pointer value.	PCI9080, PCI9054
...EepromPresent()	Determine if a blank or non-blank eeprom is present	PCI9080, PCI9054, IOP480
...VpdIdRead()	Read the Vital Product Data Id value.	PCI9080, PCI9054
...VpdNcpRead()	Read the Vital Product Data Next Configuration Pointer value.	PCI9080, PCI9054
...VpdRead()	Read a U32 value from the Vital Product Data memory.	PCI9080, PCI9054
...VpdWrite()	Write a U32 value to the Vital Product Data memory.	PCI9080, PCI9054

 © PLX Technology, Inc., 1999

…BusIopRead

Syntax:

```
RETURN_CODE Iop480BusIopRead(  IN PDEVICE_EXTENSION pExtension,
                               IN IOP_SPACE iopSpace,
                               IN U32 address,
                               IN BOOLEAN remapAddress,
                               OUT PU32 destination,
                               IN U32 transferSize,
                               IN ACCESS_TYPE accessType);
```

Compatibility:

Iop480, Pci9054, Pci9080

Description:

Reads a range of values from the local bus of a PCI device (Direct Slave Read).

- *pExtension* is the device extension of the device object associated with the PLX PCI device for the PLX driver;
- *iopSpace* defines which Local Address Space register is used;
- *address* is the starting offset from the IOP space PCI remap address (*remapAddress*=TRUE) or the actual IOP bus address to start reading from (*remapAddress*=FALSE);
- *remapAddress* states how to treat the IOP address given. FALSE means that the IOP address is an offset from the IOP space PCI remap address; TRUE means the the IOP address is the actual IOP bus address;
- *destination* is a pointer to the storage buffer for the data read;
- *transferSize* defines the number of bytes to read from the IOP bus; and,
- *accessType* defines the access type size.

Note: *Before this function can be used, the device extension must be fully initialized. If* remapAddress *is set to* `TRUE`*, this API function will remap the IOP space window according to the* address *given.*

Return Value:

Return Value	Description
`ApiSuccess`	The function returned successfully.
`ApiNullParam`	The *pExtension* or the *destination* parameter is NULL.
`ApiInvalidAccessType`	The *accessType* size is not supported for this device.
`ApiInvalidAddress`	The *address* parameter is not aligned

 © PLX Technology, Inc., 1999

Return Value	Description
	based on the *accessType* provided.
`ApiInvalidSize`	The *transferSize* parameter is 0 or is not aligned based on the *accessType* provided.

Comments:

This function should be used with small buffers. For transfers exceeding 4 KBytes, it is recommended that DMA transfers be used instead to optimize the performance of the host system.

Note: *This function monopolizes CPU time and memory and thereby hinders the performance of the host system.*

Usage:

```
RETURN_CODE rc;
PDEVICE_OBJECT fdo;
U32 length = 0x100;
U32 buf[0x40], localStartOffset = 0x20000;

rc = Iop480BusIopRead(fdo->DeviceExtension,
                      IopSpace0,
                      localStartOffset,
                      TRUE,       /* remap */
                      (PU32)buf,
                      length,
                      BitSize32);
if (rc != ApiSuccess)
{
  PlxKdPrint(("Error: Unable to read data.\n"));
  return -1;
}
```

 © PLX Technology, Inc., 1999

…BusIopWrite

Syntax:

```
RETURN_CODE Iop480BusIopWrite(  IN PDEVICE_EXTENSION pExtension,
                                IN IOP_SPACE iopSpace,
                                IN U32 address,
                                IN BOOLEAN remapAddress,
                                IN PU32 source,
                                IN U32 transferSize,
                                IN ACCESS_TYPE accessType);
```

Compatibility:

Iop480, Pci9054, Pci9080

Description:

Writes a range of values to the local bus of a PCI device (Direct Slave Write).

- *pExtension* is the device extension of the device object associated with the PLX PCI device for the PLX driver;
- *iopSpace* defines which Local Address Space register to used;
- *address* is the starting offset from the IOP space PCI remap address or the actual IOP bus address to start writing to;
- *remapAddress* states how to treat the IOP address given. FALSE means that the IOP address is an offset from the IOP space PCI remap address; TRUE means the the IOP address is the actual IOP bus address;
- *source* is a pointer to the data buffer;
- *transferSize* defines the number of bytes to write to the IOP bus; and,
- *accessType* defines the access type size.

Note: *Before this function can be used, the device extension must be fully initialized. If* remapAddress *is set to* `TRUE` *this service function will remap the IOP space window according to the* address *given.*

Return Value:

Return Value	Description
`ApiSuccess`	The function returned successfully.
`ApiNullParam`	The *pExtension* or the *source* parameter is NULL.
`ApiInvalidAccessType`	The *accessType* size is not supported for this device.
`ApiInvalidAddress`	The *address* parameter is not aligned based on the *accessType* provided.
`ApiInvalidSize`	The *transferSize* parameter is 0 or is not

 © PLX Technology, Inc., 1999

Return Value	Description
	aligned based on the *accessType* provided.

Comments:

This function should be used with small buffers. For transfers exceeding 4 KBytes, it is recommended that DMA transfers be used instead to optimize the performance of the host system.

Note: *This function monopolizes CPU time and memory and thereby hinders the performance of the host system.*

Usage:

```
  RETURN_CODE rc;
  PDEVICE_OBJECT fdo;
  U32 length = 0x100;
  U32 buf[0x40], localStartOffset = 0x20000;

  /* Clear buffer */
  memset(buf, 0, length);

  /*
    Now clear an IOP buffer starting at localStartOffset
    over a length of 0x100.
  */
  rc = Iop480BusIopWrite(fdo->DeviceExtension,
                         IopSpace0,
                         localStartOffset,
                         TRUE,
                         (PU32)buf,
                         length,
                         BitSize32);

  if (rc != ApiSuccess)
  {
    PlxKdPrint(("Error: Unable to write data.\n"));
    return -1;
}
```

 © PLX Technology, Inc., 1999

…DmaBlockChannelClose

Syntax:

```
RETURN_CODE
Iop480DmaBlockChannelClose(  IN PDEVICE_EXTENSION pExtension,
                             IN DMA_CHANNEL dmaChannel);
```

Compatibility:

Iop480, Pci9054, Pci9080

Description:

Closes the Block DMA channel.

- *pExtension* is the device extension of the device object associated with the PLX PCI device for the PLX driver; and,
- *dmaChannel* is the DMA channel number previously opened.

Note: *Before calling this function the appropriate DMA channel must be successfully opened using Iop480DmaBlockChannelOpen().*

Return Value:

Return Value	Description
`ApiSuccess`	The function returned successfully.
`ApiNullParam`	The *pExtension* parameter is NULL.
`ApiDmaChannelInvalid`	The *dmaChannel* parameter is not supported by this PLX Chip.
`ApiDmaChannelUnavailable`	The DMA channel was not opened for Block DMA.
`ApiDmaInProgress`	A DMA transfer is in progress.
`ApiDmaPaused`	The DMA channel is paused.

Usage:

```
RETURN_CODE rc;
PDEVICE_OBJECT fdo;
if ((rc= Iop480DmaBlockChannelClose(fdo->DeviceExtension,
                                    PrimaryPciChannel0))
    != ApiSuccess)
{
  PlxKdPrint(("\n\nErrors in closing channel.\n"));
  PlxKdPrint("Returned code %d\n",rc));
  return;
}
```

 © PLX Technology, Inc., 1999

…DmaBlockChannelOpen

Syntax:

```
RETURN_CODE
Iop480DmaBlockChannelOpen(  IN PDEVICE_EXTENSION pExtension,
                            IN DMA_CHANNEL dmaChannel,
                            IN PDMA_CHANNEL_DESC dmaChannelDesc);
```

Compatibility:

Iop480, Pci9054, Pci9080

Description:

Opens and initializes a DMA channel for Block DMA transfers.

- *pExtension* is the device extension of the device object associated with the PLX PCI device for the PLX driver;
- *dmaChannel* is the DMA channel number; and,
- *dmaChannelDesc* is a structure containing the DMA channel descriptors.

Note: *Before this function can be used, the device extension must be fully initialized. If the* dmaChannelDesc *parameter is NULL the function uses the current setting for the channel.*

Return Value:

Return Value	Description
`ApiSuccess`	The function returned successfully.
`ApiDmaChannelInvalid`	This *dmaChannel* parameter is not supported by this PLX chip.
`ApiNullParam`	The *pExtension* parameter is NULL.
`ApiDmaChannelUnavailable`	The DMA channel is not closed.
`ApiDmaInvalidChannelPriority`	The `DmaChannelPriority` member of *dmaChannelDesc* is not valid.

Usage:

```
RETURN_CODE rc;
PDEVICE_OBJECT fdo;
DMA_CHANNEL_DESC desc;

/* Setup DMA configuration structure */

desc.EnableWriteInvalidMode = 0;
desc.EnableDmaEOTPin = 0;
desc.DmaStopTransferMode = AssertBLAST;
desc.HoldIopAddrConst = 0;
desc.DemandMode = 0;
desc.EnableTransferCountClear = 0;
```

 © PLX Technology, Inc., 1999

```
desc.DmaChannelPriority = Rotational; /* rotational priority */
desc.EnableDualAddressCycles = 0;
desc.TholdForIopWrites = 0;
desc.TholdForIopReads = 0;
desc.TholdForPciWrites = 0;
desc.TholdForPciReads = 0;

if ((rc=Iop480DmaBlockChannelOpen(fdo->DeviceExtension,
                                  PrimaryPciChannel0,
                                  &desc))
       != ApiSuccess)
{
  PlxKdPrint(("\n\nErrors in opening channel.\n"));
  PlxKdPrint(("Returned code %d\n",rc));
  return;
}
```

 © PLX Technology, Inc., 1999

…DmaBlockTransfer

Syntax:

```
RETURN_CODE Iop480DmaBlockTransfer( IN PDEVICE_EXTENSION pExtension,
                                    IN DMA_CHANNEL dmaChannel,
                                    IN PDMA_TRANSFER_ELEMENT dmaData);
```

Compatibility:

Iop480, Pci9054, Pci9080

Description:

Controls the Block DMA transfer for a given DMA channel.

- *pExtension* is the device extension of the device object associated with the PLX PCI device for the PLX driver;
- *dmaChannel* is the DMA channel number previously opened;and,
- *dmaData* is the data for the DMA transfer (see the table in the comments section); and,

Note: *Before calling this function the appropriate DMA channel must be successfully opened using Iop480DmaBlockChannelOpen().*

Return Value:

Return Value	Description
`ApiSuccess`	The function returned successfully.
`ApiDmaChannelInvalid`	The *dmaChannel* parameter is not supported by this PLX chip.
`ApiNullParam`	The *pExtension* or *dmaData* (if *dmaCommand* is not `DmaStart`) parameters are NULL.
`ApiDmaChannelUnavailable`	The DMA channel was not opened for Block DMA.
`ApiDmaInProgress`	The DMA channel is in progress.

Comments:

Block DMA transfers are useful with contiguous host buffers described by a *PCI address*. Therefore, it is incorrect to allocate a buffer (using malloc(), at compile time or in the stack) and use its address with the Block DMA functions. Instead, users should use a known PCI address (not *user address*) of, for example, the common buffer or another device's known PCI base address.

When a Block DMA transfer is done, it will generate a PCI interrupt which will be handled by the driver.

 © PLX Technology, Inc., 1999

The DMA_TRANSFER_ELEMENT structure contains members whose signification may differ depending on the DMA transfer type. Also, some other members are ignored by the API call. Here is the structure:

Structure Element	Signification
UserAddr	Ignored.
LowPciAddr	Low PCI address of the host buffer.
HighPciAddr	(Pci9054 and Iop480 only) High PCI address of the host buffer.
IopAddr	The IOP address for the transfer.
TransferCount	The number of bytes for the transfer.
PciSglLoc	Ignored.
LastSglElement	Ignored.
TerminalCountIntr	Indicates if an interrupt is to be generated when the DMA transfer is done. A 1 means yes.
IopToPciDma	Direction of the transfer. A 1 means an IOP-to-PCI transfer.
NextSglPtr	Ignored.

Usage:

```
PDEVICE_OBJECT fdo;
DMA_TRANSFER_ELEMENT dmaData;
U32 hostPhysicalAddress;


dmaData.Iop480Dma.LowPciAddr = hostPhysicalAddress;
dmaData.Iop480Dma.IopAddr = 0x80000;
dmaData.Iop480Dma.TransferCount = 0x1000;
dmaData.Iop480Dma.IopToPciDma = 0;
dmaData.Iop480Dma.TerminalCountIntr = 0;

if((rc=Iop480DmaBlockTransfer(fdo->DeviceExtension,
                              PrimaryPciChannel0,
                              &dmaData))
   != ApiSuccess)
{
  PlxKdPrint(("\n\nErrors in Block Dma.\n"));
  PlxKdPrint(("Returned code %d\n",rc));
  return -1;
}
```

 © PLX Technology, Inc., 1999

…DmaBlockTransferRestart

Syntax:

```
RETURN_CODE
Iop480DmaBlockTransferRestart( IN PDEVICE_EXTENSION pExtension,
                               IN DMA_CHANNEL dmaChannel,
                               IN U32 transferSize);
```

Compatibility:

Iop480, Pci9054, Pci9080

Description:

Restarts the Block DMA transfer for a pre-programmed DMA channel.

- *pExtension* is the device extension of the device object associated with the PLX PCI device for the PLX driver;
- *dmaChannel* is the DMA channel number previously opened and programmed; and,
- *transferSize* is the DMA transfer size;

Note: *Before calling this function the appropriate DMA channel must be successfully programmed using Iop480DmaBlockTransfer().*

Return Value:

Return Value	Description
`ApiSuccess`	The function returned successfully.
`ApiNullParam`	The *pExtension* parameter is NULL.
`ApiDmaChannelInvalid`	The *dmaChannel* parameter is not supported by this PLX chip.
`ApiDmaChannelUnavailable`	The DMA channel was not opened for Block DMA.
`ApiDmaInProgress`	The DMA channel is in progress.

Usage:

```
RETURN_CODE rc;
PDEVICE_OBJECT fdo;
U32 totalSize = 0x100;

rc = Iop480DmaBlockTransferRestart(fdo->DeviceExtension,
                                   PrimaryPciChannel0,
                                   totalSize);
if (rc != ApiSuccess)
{
    PlxKdPrint(("Restart failed\n"));
    return -1;
}
```

 © PLX Technology, Inc., 1999

…DmaSglChannelClose

Syntax:

```
RETURN_CODE Iop480DmaSglChannelClose( IN PDEVICE_EXTENSION pExtension,
                                      IN DMA_CHANNEL dmaChannel);
```

Compatibility:

Iop480, Pci9054, Pci9080

Description:

Closes the Scatter-Gather DMA channel.

- *pExtension* is the device extension of the device object associated with the PLX PCI device for the PLX driver; and,
- *dmaChannel* is the DMA channel number previously opened.

Note: *Before calling this function the appropriate DMA channel must be successfully opened using Iop480DmaSglChannelOpen().*

Return Value:

Return Value	Description
`ApiSuccess`	The function returned successfully.
`ApiNullParam`	The *pExtension* parameter is NULL.
`ApiDmaChannelInvalid`	The *dmaChannel* parameter is not supported by this PLX Chip.
`ApiDmaChannelUnavailable`	The DMA channel was not opened for Scatter-Gather DMA.
`ApiDmaInProgress`	A DMA transfer is in progress.
`ApiDmaPaused`	The DMA channel is paused.

Usage:

```
RETURN_CODE rc;
PDEVICE_OBJECT fdo;

if ((rc= Iop480DmaSglChannelClose(fdo->DeviceExtension,
                                  PrimaryPciChannel0))
    != ApiSuccess)
{
  PlxKdPrint(("\n\nErrors in closing channel.\n"));
  PlxKdPrint(("Returned code %d\n",rc));
  return;
}
```

 © PLX Technology, Inc., 1999

…DmaSglChannelOpen

Syntax:

```
RETURN_CODE
Iop480DmaSglChannelOpen( IN PDEVICE_EXTENSION pExtension,
                         IN DMA_CHANNEL dmaChannel,
                         IN PDMA_CHANNEL_DESC dmaChannelDesc);
```

Compatibility:

Iop480, Pci9054, Pci9080

Description:

Opens and initializes a DMA channel for Scatter-Gather DMA transfers.

- *pExtension* is the device extension of the device object associated with the PLX PCI device for the PLX driver;
- *dmaChannel* is the DMA channel number; and,
- *dmaChannelDesc* is a structure containing the DMA channel descriptors.

Note: *Before this function can be used, the device extension must be fully initialized. If the* dmaChannelDesc *parameter is NULL the function uses the current setting for the channel. Scatter-Gather DMA needs the DMA interrupts enabled for proper operation and thus this function enables the corresponding PCI DMA interrupt.*

Return Value:

Return Value	Description
`ApiSuccess`	The function returned successfully.
`ApiDmaChannelInvalid`	This *dmaChannel* parameter is not supported by this PLX chip.
`ApiNullParam`	The *pExtension* parameter is NULL.
`ApiDmaChannelUnavailable`	The DMA channel is not closed.
`ApiDmaInvalidChannelPriority`	The `DmaChannelPriority` member of *dmaChannelDesc* is not valid.

Usage:

```
RETURN_CODE rc;
PDEVICE_OBJECT fdo;
DMA_CHANNEL_DESC desc;

/* Setup DMA configuration structure */
desc.DmaChannelPriority = Rotational; /* rotational priority */
desc.HoldIopAddrConst = 0;
desc.DemandMode = 0;
desc.EnableWriteInvalidMode = 0;
desc.EnableDmaEOTPin = 0;
```

 © PLX Technology, Inc., 1999

```
desc.DmaStopTransferMode = AssertBLAST;
desc.EnableTransferCountClear = 0;
desc.EnableDualAddressCycles = 0;
desc.HoldIopSourceAddrConst = 0;
desc.ValidModeEnable = 0;
desc.ValidStopControl = 0;
desc.TholdForIopWrites = 0;
desc.TholdForIopReads = 0;
desc.TholdForPciWrites = 0;
desc.TholdForPciReads = 0;

if ((rc=Iop480DmaSglChannelOpen(fdo->DeviceExtension,
                                PrimaryPciChannel0,
                                &desc))
       != ApiSuccess)
{
  PlxKdPrint(("\n\nErrors in opening channel.\n"));
  PlxKdPrint(("Returned code %d\n",rc));
  return;
}
```

 © PLX Technology, Inc., 1999

…DmaSglTransfer

Syntax:

```
RETURN_CODE Iop480DmaSglTransfer(  IN PDEVICE_EXTENSION pExtension,
                                   IN DMA_CHANNEL dmaChannel,
                                   IN PDMA_TRANSFER_ELEMENT dmaData);
```

Compatibility:

Iop480, Pci9054, Pci9080

Description:

Controls a Scatter-Gather DMA transfer for a given DMA channel.

- *pExtension* is the device extension of the device object associated with the PLX PCI device for the PLX driver;
- *dmaChannel* is the DMA channel number previously opened;and,
- *dmaData* is the data for the DMA transfer (see the table in the comments section); and,

Note: *Before calling this function the appropriate DMA channel must be successfully opened using Iop480DmaSglChannelOpen().*

Return Value:

Return Value	Description
`ApiSuccess`	The function returned successfully.
`ApiDmaChannelInvalid`	The *dmaChannel* parameter is not supported by this PLX Chip.
`ApiNullParam`	The *pExtension* or *dmaData* (if *dmaCommand* is `DmaStart`) parameters are NULL.
`ApiDmaChannelUnavailable`	The DMA channel was not opened for SGL DMA.
`ApiDmaInProgress`	The DMA channel is in progress.
`ApiFailed`	Failed to start the DMA transfer. ***Note:*** *Ensure that there is enough memory to build the Scatter-Gather List. Memory for the Scatter-Gather List can be added by increasing the* `MaxSglTransferSize` *registry entry.*

Comments:

Scatter-Gather transfers are useful with host buffers described by a *user address*.

 © PLX Technology, Inc., 1999

When a SGL transfer is done, it will generate an interrupt to the PCI which will be handled by the driver.

The DMA_TRANSFER_ELEMENT structure contains members whose signification may differ depending on the DMA transfer type. Also, some other members are ignored by the API call. Here is the structure:

Structure Element	Signification
UserAddr	User address of the host buffer.
LowPciAddr	Ignored
HighPciAddr	Ignored
IopAddr	The IOP address for the transfer.
TransferCount	The number of bytes for the transfer.
PciSglLoc	Ignored.
LastSglElement	Ignored.
TerminalCountIntr	Ignored.
IopToPciDma	Direction of the transfer. A 1 means an IOP-to-PCI transfer.
NextSglPtr	Ignored.

Usage:

```
PDEVICE_OBJECT fdo;
DMA_TRANSFER_ELEMENT dmaData;
U8 buffer[0x10000];

dmaData.Iop480Dma.UserAddr = buffer;
dmaData.Iop480Dma.IopAddr = 0x80000;
dmaData.Iop480Dma.TransferCount = 0x10000;
dmaData.Iop480Dma.IopToPciDma = 1;

if((rc=Iop480DmaSglTransfer(fdo->DeviceExtension,
                            PrimaryPciChannel0,
                            &dmaData))
   != ApiSuccess)
{
  PlxKdPrint(("\n\nErrors in Sgl Dma.\n"));
  PlxKdPrint(("Returned code %d\n",rc));
  return -1;
}
```

 © PLX Technology, Inc., 1999

…DmaShuttleChannelClose

Syntax:

```
RETURN_CODE
Iop480DmaShuttleChannelClose( IN PDEVICE_EXTENSION pExtension,
                              IN DMA_CHANNEL dmaChannel);
```

Compatibility:

Iop480, Pci9054, Pci9080

Description:

Closes the Shuttle DMA channel. If the transfer is in progress, then the transfer will be aborted.

- *pExtension* is the device extension of the device object associated with the PLX PCI device for the PLX driver; and,
- *dmaChannel* is the DMA channel number previously opened.

Note: *Before calling this function the appropriate DMA channel must be successfully opened using Iop480DmaShuttleChannelOpen().*

Return Value:

Return Value	Description
`ApiSuccess`	The function returned successfully.
`ApiNullParam`	The *pExtension* parameter is NULL.
`ApiDmaChannelInvalid`	The *dmaChannel* parameter is not supported by this PLX Chip.
`ApiDmaChannelUnavailable`	The DMA channel was not opened for Shuttle DMA.

Usage:

```
RETURN_CODE rc;
PDEVICE_OBJECT fdo;

if ((rc= Iop480DmaShuttleChannelClose(fdo->DeviceExtension,
                                      PrimaryPciChannel0))
    != ApiSuccess)
{
  PlxKdPrint(("\nErrors in closing channel.\nReturned code %d",rc));
  return;
}
```

 © PLX Technology, Inc., 1999

…DmaShuttleChannelOpen

Syntax:

```
RETURN_CODE
Iop480DmaShuttleChannelOpen(  IN PDEVICE_EXTENSION pExtension,
                              IN DMA_CHANNEL dmaChannel,
                              IN PDMA_CHANNEL_DESC dmaChannelDesc);
```

Compatibility:

Iop480, Pci9054, Pci9080

Description:

Opens and initializes a DMA channel for Shuttle DMA transfers.

- *pExtension* is the device extension of the device object associated with the PLX PCI device for the PLX driver;
- *dmaChannel* is the DMA channel number; and,
- *dmaChannelDesc* is a structure containing the DMA channel descriptors.

Note: *Before this function can be used, the device extension must be fully initialized. If the* dmaChannelDesc *parameter is NULL the function uses the current setting for the channel.*

Return Value:

Return Value	Description
`ApiSuccess`	The function returned successfully.
`ApiDmaChannelInvalid`	This *dmaChannel* parameter is not supported by this PLX chip.
`ApiNullParam`	The *pExtension* parameter is NULL.
`ApiDmaChannelUnavailable`	The DMA channel is not closed.
`ApiDmaInvalidChannelPriority`	The `DmaChannelPriority` member of *dmaChannelDesc* is not valid.

Usage:

```
RETURN_CODE rc;
PDEVICE_OBJECT fdo;
DMA_CHANNEL_DESC desc;

/* Setup DMA configuration structure */
desc.EnableReadyInput = 1;
desc.EnableBTERMInput = 0;
desc.EnableIopBurst = 0;
desc.EnableWriteInvalidMode = 0;
desc.EnableDmaEOTPin = 0;
desc.DmaStopTransferMode = AssertBLAST;
desc.HoldIopAddrConst = 0;
```

 © PLX Technology, Inc., 1999

```
desc.HoldIopSourceAddrConst = 0;
desc.HoldIopDestAddrConst = 0;
desc.DemandMode = 0;
desc.EnableTransferCountClear = 0;
desc.DmaChannelPriority = Rotational; /* rotational priority */
desc.WaitStates = 0;
desc.IopBusWidth = 3; /* 32 bit bus */
desc.Reserved1 = 0;
desc.TholdForIopWrites = 0;
desc.TholdForIopReads = 0;
desc.TholdForPciWrites = 0;
desc.TholdForPciReads = 0;
desc.Reserved2 = 0;

if ((rc=Iop480DmaShuttleChannelOpen(fdo->DeviceExtension,
                                    PrimaryPciChannel0,
                                    &desc))
       != ApiSuccess)
{
  PlxKdPrint(("\n\nErrors in opening channel.\n"));
  PlxKdPrint(("Returned code %d\n",rc));
  return;
}
```

 © PLX Technology, Inc., 1999

…DmaShuttleTransfer

Syntax:

```
RETURN_CODE
Iop480DmaShuttleTransfer(  IN PDEVICE_EXTENSION pExtension,
                           IN DMA_CHANNEL dmaChannel,
                           IN PDMA_TRANSFER_ELEMENT dmaData);
```

Compatibility:

Iop480, Pci9054, Pci9080

Description:

Controls a Shuttle DMA transfer for a given DMA channel.

- *pExtension* is the device extension of the device object associated with the PLX PCI device for the PLX driver;
- *dmaChannel* is the DMA channel number previously opened;
- *dmaCommand* is the action to perform on this DMA channel; and,
- *dmaData* is the data for the DMA transfer (see the table in the comments section).

Note: *Before calling this function the appropriate DMA channel must be successfully opened using Iop480DmaShuttleChannelOpen().*

Return Value:

Return Value	Description
`ApiSuccess`	The function returned successfully.
`ApiDmaChannelInvalid`	The *dmaChannel* parameter is not supported by this PLX Chip.
`ApiNullParam`	The *pExtension* or *dmaData* (if *dmaCommand* is not `DmaStart`) parameters are NULL.
`ApiDmaChannelUnavailable`	The DMA channel was not opened for Shuttle DMA.
`ApiDmaInProgress`	The DMA channel is in progress.
`ApiFailed`	Failed to start the DMA transfer. ***Note:*** *Ensure that there is enough memory to build the Scatter-Gather List. Memory for the Scatter-Gather List can be added by increasing the* `MaxSglTransferSize` *registry entry.*

Comments:

 © PLX Technology, Inc., 1999

Shuttle transfers are useful with host buffers described by a *user address*.

Because shuttle transfers are perpetual, no interrupts are generated by the Shuttle DMA functions. If an interrupt would occur, it would be perpetual as well and may hang the host OS.

The DMA_TRANSFER_ELEMENT structure contains members whose signification may differ depending on the DMA transfer type. Also, some other members are ignored by the API call. Here is the structure:

Structure Element	Signification
UserAddr	User address of the host buffer.
LowPciAddr	Ignored
HighPciAddr	Ignored
IopAddr	The IOP address for the transfer.
TransferCount	The number of bytes for the transfer.
PciSglLoc	Ignored.
LastSglElement	Ignored.
TerminalCountIntr	Ignored.
IopToPciDma	Direction of the transfer. A 1 means an IOP-to-PCI transfer.
NextSglPtr	Ignored.

Usage:

```
  PDEVICE_OBJECT fdo;
  DMA_TRANSFER_ELEMENT dmaData;
  U8 buffer[0x10000];

  dmaData.Iop480Dma.UserAddr = buffer;
  dmaData.Iop480Dma.IopAddr = 0x80000;
  dmaData.Iop480Dma.TransferCount = 0x10000;
  dmaData.Iop480Dma.IopToPciDma = 1;

  if((rc=Iop480DmaShuttleTransfer(fdo->DeviceExtension,
                                  PrimaryPciChannel0,
                                  &dmaData));
!= ApiSuccess)
  {
    PlxKdPrint(("\n\nErrors in Shuttle Dma.\n"));
    PlxKdPrint(("Returned code %d\n",rc));
    return -1;
  }
```

 © PLX Technology, Inc., 1999

…IntrDisable

Syntax:

```
RETURN_CODE Iop480IntrDisable(  IN PDEVICE_EXTENSION pExtension,
                                IN PPLX_INTR plxIntr);
```

Compatibility:

Iop480, Pci9054, Pci9080

Description:

Disables specific interrupts of a PCI device.

- *pExtension* is the device extension of the device object associated with the PLX PCI device for the PLX driver; and,
- *plxIntr* is the interrupt structure that describes which interrupts will be disabled.

Return Value:

Return Value	Description
`ApiSuccess`	The function returned successfully.
`ApiNullParam`	The *pExtension* or the *plxIntr* parameter is NULL.

Usage:

```
RETURN_CODE rc;
PDEVICE_OBJECT fdo;
PLX_INTR plxIntrStatus;

plxIntrStatus.InboundPost = 1;
plxIntrStatus.OutboundPost = 1;
plxIntrStatus.IopDmaChannel0 = 1;
plxIntrStatus.PciDmaChannel0 = 1;
plxIntrStatus.IopDmaChannel1 = 1;
plxIntrStatus.PciDmaChannel1 = 1;

rc = Iop480IntrDisable(fdo->DeviceExtension, plxIntrStatus);
if ( rc != ApiSuccess )
  PlxKdPrint(("\a\nPlxIntrDisable failed with error code %08lx.",
              rc));
else
  PlxKdPrint(("\nPlxIntrDisable tested: OK."));
```

 © PLX Technology, Inc., 1999

…IntrEnable

Syntax:

```
RETURN_CODE Iop480IntrEnable(  IN PDEVICE_EXTENSION pExtension,
                               IN PPLX_INTR plxIntr);
```

Compatibility:

Iop480, Pci9054, Pci9080

Description:

Enables specific interrupts of a PCI device.

- *pExtension* is the device extension of the device object associated with the PLX PCI device for the PLX driver; and,
- *plxIntr* is the interrupt structure that describes which interrupts will be enabled.

Return Value:

Return Value	Description
`ApiSuccess`	The function returned successfully.
`ApiNullParam`	The *pExtension* or the *plxIntr* parameter is NULL.

Usage:

```
RETURN_CODE rc;
PDEVICE_OBJECT fdo;
PLX_INTR plxIntrStatus;

plxIntrStatus.InboundPost = 1;
plxIntrStatus.OutboundPost = 1;
plxIntrStatus.PciDmaChannel0 = 1;
plxIntrStatus.PciDmaChannel1 = 1;

rc = Iop480IntrEnable(fdo->DeviceExtension, &plxIntrStatus);
if ( rc != ApiSuccess )
  PlxKdPrint(("\a\nPlxIntrEnable failed with error code %08lx.",
              rc));
else
  PlxKdPrint(("\nPlxIntrEnable tested: OK."));
```

 © PLX Technology, Inc., 1999

…IntrStatusGet

Syntax:

```
RETURN_CODE Iop480IntrStatusGet(  IN PDEVICE_EXTENSION pExtension,
                                  OUT PPLX_INTR plxIntr);
```

Compatibility:

Iop480, Pci9054, Pci9080

Description:

Returns the interrupts of the PCI device are were last active. The interrupts are cleared once they are read.

- *pExtension* is the device extension of the device object associated with the PLX PCI device for the PLX driver; and,
- *plxIntr* is the interrupt structure that contains information detailing which interrupts are active.

Return Value:

Return Value	Description
`ApiSuccess`	The function returned successfully.
`ApiNullParam`	The *pExtension* or the *plxIntr* parameter is NULL.

Usage:

```
RETURN_CODE rc;
PDEVICE_OBJECT fdo;
PLX_INTR plxIntrStatus;
rc = Iop480IntrStatusGet(fdo->DeviceExtension, &plxIntrStatus);
if ( rc != ApiSuccess )
  PlxKdPrint(("PlxIntrStatusGet failed with error code %08lx.", rc));
else {
  PlxKdPrint(("Interrupted with following interrupt status."));
  PlxKdPrint(("OutboundOverflow %s",
         (plxIntrStatus.OutboundOverflow)?"Yes":"NO"));
  PlxKdPrint(("IopDmaChannel0   %s",
         (plxIntrStatus.IopDmaChannel0)?"Yes":"NO"));
  PlxKdPrint(("PciDmaChannel0   %s",
         (plxIntrStatus.PciDmaChannel0)?"Yes":"NO"));
  PlxKdPrint(("IopDmaChannel1   %s",
         (plxIntrStatus.IopDmaChannel1)?"Yes":"NO"));
  PlxKdPrint(("PciDmaChannel1   %s",
         (plxIntrStatus.PciDmaChannel1)?"Yes":"NO"));
}
```

 © PLX Technology, Inc., 1999

…IoPortRead

Syntax:

```
RETURN_CODE Iop480IoPortRead(  IN PDEVICE_EXTENSION pExtension,
                               IN U32 address,
                               IN ACCESS_TYPE bits,
                               OUT PVOID pOutData);
```

Compatibility:

Iop480, Pci9054, Pci9080

Description:

Reads a value from an I/O port.

- *pExtension* is the device extension of the device object associated with the PLX PCI device for the PLX driver;
- *address* is the I/O port address to read from;
- *accessType* defines the access type size; and,
- *pOutData* is the data read from the I/O port.

Return Value:

Return Value	Description
`ApiSuccess`	The function returned successfully.
`ApiNullParam`	The *pExtension* or the *pOutData* parameter is NULL.
`ApiInvalidAddress`	The *address* parameter is 0.
`ApiInvalidSize`	The *accessType* size is not supported for this device.

Usage:

```
RETURN_CODE rc;
PDEVICE_OBJECT fdo;
U32 value;

rc = Iop480IoPortRead(fdo->DeviceExtension, 0x6F00, BitSize32,
                      &value);
if (rc != ApiSuccess)
{
  PlxKdPrint(("Error: Unable to read data.\n"));
  return -1;
}
rc = Iop480IoPortWrite(fdo->DeviceExtension, 0x6F04, BitSize32,
                       &value);
if (rc != ApiSuccess)
{
```

 © PLX Technology, Inc., 1999

```
    PlxKdPrint(("Error: Unable to write data.\n"));
    return -1;
  }
```

 © PLX Technology, Inc., 1999

…IoPortWrite

Syntax:

```
RETURN_CODE Iop480IoPortWrite(  IN PDEVICE_EXTENSION pExtension,
                                IN U32 address,
                                IN ACCESS_TYPE bits,
                                IN PVOID pValue);
```

Compatibility:

Iop480, Pci9054, Pci9080

Description:

Writes a value to an I/O port.

- *pExtension* is the device extension of the device object associated with the PLX PCI device for the PLX driver;
- *address* is the I/O port address to write to;
- *accessType* defines the access type size; and,
- *pOutData* is the data to write to the I/O port.

Return Value:

Return Value	Description
`ApiSuccess`	The function returned successfully.
`ApiNullParam`	The *pExtension* or the *pValue* parameter is NULL.
`ApiInvalidAddress`	The *address* parameter is 0.
`ApiInvalidAccessType`	The *accessType* size is not supported for this device.

Usage:

```
RETURN_CODE rc;
PDEVICE_OBJECT fdo;
U32 value;

rc = Iop480IoPortRead(fdo->DeviceExtension, 0x6F00, BitSize32,
                      &value);
if (rc != ApiSuccess)
{
  PlxKdPrint(("Error: Unable to read data.\n"));
  return -1;
}
rc = Iop480IoPortWrite(fdo->DeviceExtension, 0x6F04, BitSize32,
                       &value);
if (rc != ApiSuccess)
{
```

 © PLX Technology, Inc., 1999

```
    PlxKdPrint(("Error: Unable to write data.\n"));
    return -1;
  }
```

 © PLX Technology, Inc., 1999

…MuInboundPortRead

Syntax:

```
RETURN_CODE Iop480MuInboundPortRead(  IN PDEVICE_EXTENSION pExtension,
                                      OUT PU32 framePointer);
```

Compatibility:

Iop480, Pci9054, Pci9080

Description:

Reads the Inbound Port and gets a Free Message Frame.

- *pExtension* is the device extension of the device object associated with the PLX PCI device for the PLX driver; and,
- *framePointer* is the address of the Message Frame (MFA).

Note: *Before this function can be used, the Messaging Unit must be initialized (initialized by the IOP application).*

Return Value:

Return Value	Description
`ApiSuccess`	The function returned successfully.
`ApiNullParam`	The *pExtension* or *framePointer* parameters are NULL.

Usage:

```
RETURN_CODE rc;
PDEVICE_OBJECT fdo;

/* Read inbound port */
rc = Iop480MuInboundPortRead(fdo->DeviceExtension, &framePointer);
if (rc != ApiSuccess)
{
  PlxKdPrint(("Error: Unable to read inbound port.\n"));
  return -1;
}
```

 © PLX Technology, Inc., 1999

…MuInboundPortWrite

Syntax:

```
RETURN_CODE Iop480MuInboundPortWrite( IN PDEVICE_EXTENSION pExtension,
                                      IN PU32 framePointer);
```

Compatibility:

Iop480, Pci9054, Pci9080

Description:

Writes to the Inbound Port with a posted message frame.

- *pExtension* is the device extension of the device object associated with the PLX PCI device for the PLX driver; and,
- *framePointer* is the address of the Message Frame (MFA) to write.

Note: *Before this function can be used, the device extension must be fully initialized and the Messaging Unit must be initialized (initialized by the IOP application).*

Return Value:

Return Value	Description	IC Support
`ApiSuccess`	The function returned successfully.	PCI 9080, PCI 9054
`ApiNullParam`	The *pExtension* or *framePointer* parameters are NULL.	PCI 9080, PCI 9054

Usage:

```
RETURN_CODE rc;
PDEVICE_OBJECT fdo;
U32 framePointer;

/* Write to the inbound port */
rc = Iop480MuInboundPortWrite(fdo->DeviceExtension, &framePointer);
if (rc != ApiSuccess)
{
  PlxKdPrint(("Error: Unable to write to the inbound port.\n"));
  return -1;
}
```

 © PLX Technology, Inc., 1999

…MuOutboundPortRead

Syntax:

```
RETURN_CODE Iop480MuOutboundPortRead( IN PDEVICE_EXTENSION pExtension,
                                      OUT PU32 framePointer);
```

Compatibility:

Iop480, Pci9054, Pci9080

Description:

Reads the Outbound Port and gets a posted message frame.

- *pExtension* is the device extension of the device object associated with the PLX PCI device for the PLX driver; and,
- *framePointer* is the address of the Message Frame (MFA).

Note: *Before this function can be used, the device extension must be fully initialized and the Messaging Unit must be initialized (initialized by the IOP application).*

Return Value:

Return Value	Description
`ApiSuccess`	The function returned successfully.
`ApiNullParam`	The *pExtension* or *framePointer* parameters are NULL.

Usage:

```
RETURN_CODE rc;
PDEVICE_OBJECT fdo;

/* Read outbound port */
rc = Iop480MuOutboundPortRead(fdo->DeviceExtension, &framePointer);
if (rc != ApiSuccess)
{
  PlxKdPrint(("Error: Unable to read the outbound port.\n"));
  return -1;
}
```

 © PLX Technology, Inc., 1999

…MuOutboundPortWrite

Syntax:

```
RETURN_CODE Iop480MuOutboundPortWrite(  IN PDEVICE_EXTENSION pExtension,
                                        IN PU32 framePointer);
```

Compatibility:

Iop480, Pci9054, Pci9080

Description:

Writes to the Outbound Port with a free message frame.

- *pExtension* is the device extension of the device object associated with the PLX PCI device for the PLX driver; and,
- *framePointer* is the address of the Message Frame (MFA) to write.

Note: *Before this function can be used, the device extension must be fully initialized and the Messaging Unit must be initialized (initialized by the IOP application).*

Return Value:

Return Value	Description
`ApiSuccess`	The function returned successfully.
`ApiNullParam`	The *pExtension* or *framePointer* parameters are NULL.

Usage:

```
RETURN_CODE rc;
PDEVICE_OBJECT fdo;
U32 framePointer;

/* Write to the outbound port */
rc = Iop480MuOutboundPortWrite(fdo->DeviceExtension, &framePointer);
if (rc != ApiSuccess)
{
  PlxKdPrint(("Error: Unable to write to the outbound port.\n"));
  return -1;
}
```

 © PLX Technology, Inc., 1999

…PowerLevelGet

Syntax:

```
PLX_POWER_LEVEL Iop480PowerLevelGet(  IN PDEVICE_EXTENSION pExtension,
                                      OUT PRETURN_CODE returnCode);
```

Compatibility:

Iop480, Pci9054, Pci9080

Description:

Gets the current power level of a PCI device.

- *pExtension* is the device extension of the device object associated with the PLX PCI device for the PLX driver; and,
- *returnCode* is the return code of the function.

Note: *Before this function can be used, the device extension must be fully initialized.*

Return Value:

This function returns the current power level of the PLX Chip. The status of the function call is returned via the *returnCode* parameter. The return codes are as follows:

Return Value	Description
`ApiSuccess`	The function returned successfully.
`ApiConfigAccessFailed`	The power level is unreadable.
`ApiUnsupportedFunction`	Power management is not supported by the board.
`ApiNullParam`	The *pExtension* parameter is NULL.

Usage:

```
RETURN_CODE rc;
PDEVICE_OBJECT fdo;
PLX_POWER_LEVEL powerLevel;
powerLevel = Iop480PowerLevelGet(fdo->DeviceExtension, &rc);
if (rc != ApiSuccess)
  PlxKdPrint(("\n Error Getting Power Level. RC = %x", rc));
else {
  if (powerLevel == D0)
    PlxKdPrint(("\n OK: Power Level received as D0"));
  else
    PlxKdPrint(("\n ERROR: Power Level received as %d", powerLevel));
}
```

 © PLX Technology, Inc., 1999

…PowerLevelQuery

Syntax:

```
BOOLEAN Iop480PowerLevelQuery(  IN PDEVICE_EXTENSION pExtension,
                                IN PLX_POWER_LEVEL plxPowerLevel);
```

Compatibility:

Iop480, Pci9054

Description:

Verify that a power level is supported by the PCI device.

- *pExtension* is the device extension of the device object associated with the PLX PCI device for the PLX driver; and,
- *plxPowerLevel* is the power level being queried.

Note: *Before this function can be used, the device extension must be fully initialized.*

Return Value:

Return Value	Description
TRUE	The *plxPowerLevel* power level is supported.
FALSE	The *plxPowerLevel* power level is not supported.

Usage:

```
PDEVICE_OBJECT fdo;

if (Iop480PowerLevelQuery(fdo->DeviceExtension, D3Hot))
  PlxKdPrint(("\n D3Hot Power Level is supported\n"));
else
  PlxKdPrint(("\n D3Hot Power Level is not supported\n"));
```

 © PLX Technology, Inc., 1999

…PowerLevelSet

Syntax:

```
RETURN_CODE Iop480PowerLevelSet(  IN PDEVICE_EXTENSION pExtension,
                                  IN PLX_POWER_LEVEL plxPowerLevel);
```

Compatibility:

Iop480, Pci9054, Pci9080

Description:

Sets the power level of a PCI device.

- *pExtension* is the device extension of the device object associated with the PLX PCI device for the PLX driver; and,
- *plxPowerLevel* is the new power level.

Return Value:

Return Value	Description
`ApiSuccess`	The function returned successfully.
`ApiConfigAccessFailed`	The power level is unreadable.
`ApiUnsupportedFunction`	Power management is not supported by the board.
`ApiNullParam`	The *pExtension* parameter is NULL.
`ApiInvalidPowerState`	The *plxPowerLevel* parameter is invalid for this PLX Chip.

Usage:

```
RETURN_CODE rc;
PDEVICE_OBJECT fdo;

rc = Iop480PowerLevelSet(fdo->DeviceExtension, D0);
if (rc != ApiSuccess)
{
  PlxKdPrint(("\n.Error Setting a valid Power Level (D0). RC = %x",
              rc));
else
  PlxKdPrint(("\n.OK: Valid Power Level Set"));
```

 © PLX Technology, Inc., 1999

…RegisterDoorbellRead

Syntax:

```
U32 Iop480RegisterDoorbellRead( IN PDEVICE_EXTENSION pExtension,
                                OUT PRETURN_CODE returnCode);
```

Compatibility:

Iop480, Pci9054, Pci9080

Description:

Clears and reads value to any doorbell register on a PCI device.

- *pExtension* is the device extension of the device object associated with the PLX PCI device for the PLX driver; and,
- *returnCode* is a pointer to a buffer to store the return code.

Return Value:

This function returns the value read from the PCI-to-IOP doorbell register. The status of the function call is returned via the *returnCode* parameter. The return codes are as follows:

Return Value	Description
`ApiSuccess`	The function returned successfully.
`ApiNullParam`	The *pExtension* parameter is NULL.

Usage:

```
RETURN_CODE rc;
PDEVICE_OBJECT fdo;
U32 value;

value = Iop480RegisterDoorbellRead(fdo->DeviceExtension, &rc);
if (rc != ApiSuccess)
  PlxKdPrint(("\n ERROR: Reading Doorbell did not work"));
else
  PlxKdPrint(("\n OK: Doorbell Read succeeded. Value is %x", value();
```

 © PLX Technology, Inc., 1999

…RegisterDoorbellSet

Syntax:

```
RETURN_CODE Iop480RegisterDoorbellSet(  IN PDEVICE_EXTENSION pExtension,
                                        IN U32 data);
```

Compatibility:

Iop480, Pci9054, Pci9080

Description:

Writes a value to any doorbell register on a PCI device.

- *pExtension* is the device extension of the device object associated with the PLX PCI device for the PLX driver; and,
- *data* is a U32 value to store in the register.

Return Value:

Return Value	Description
`ApiSuccess`	The function returned successfully.
`ApiNullParam`	The *pExtension* parameter is NULL.

Usage:

```
RETURN_CODE rc;
PDEVICE_OBJECT fdo;
U32 value;

rc = Iop480RegisterDoorbellSet(fdo->DeviceExtension, 0x1234567);
if (rc != ApiSuccess)
  PlxKdPrint(("\n ERROR: Setting Doorbell did not work"));
else
{
  if(Iop480RegisterRead(fdo->DeviceExtension, 0x60, &rc)==0x1234567)
    PlxKdPrint(("\n OK: DoorbellSet succeeded"));
  else
  {
    PlxKdPrint(("\n ERROR: After setting doorbell and reading the "
                "register the return data was wrong!"));
    PlxKdPrint(("\n ERROR: This is normal since the BSP will clear "
                "the doorbell immediately!"));
  }
}
```

 © PLX Technology, Inc., 1999

…RegisterRead

Syntax:

```
U32 Iop480RegisterRead(  IN PDEVICE_EXTENSION pExtension,
                         IN U32 registerOffset,
                         OUT PRETURN_CODE pReturnCode);
```

Compatibility:

Iop480, Pci9054, Pci9080

Description:

Reads a register on the PLX chip.

- *pExtension* is the device extension of the device object associated with the PLX PCI device for the PLX driver;
- *registerOffset* is the register offset; and,
- *pReturnCode* is a pointer to the return code;

Return Value:

This function returns the value read from the register. The status of the function call is returned via the *returnCode* parameter. The return codes are as follows:

Return Value	Description
`ApiSuccess`	The function returned successfully.
`ApiNullParam`	The *pExtension* parameter is NULL.
`ApiInvalidRegister`	The *registerOffset* parameter is out of range or is not on a 4-byte boundary.

Usage:

```
RETURN_CODE rc;
DEVICE_OBJECT fdo;
U32 value;

value = Iop480RegisterRead(fdo->DeviceExtension, 0x18, &rc);
if (rc != ApiSuccess)
  PlxKdPrint(("\n Error Reading a valid Register"));
else
  PlxKdPrint(("\n Read succeeded with a value of %x",
              value));
```

 © PLX Technology, Inc., 1999

…RegisterMailboxRead

Syntax:

```
U32 Iop480RegisterMailboxRead(  IN PDEVICE_EXTENSION pExtension,
                                IN MAILBOX_ID mailboxId,
                                OUT PRETURN_CODE returnCode);
```

Compatibility:

Iop480, Pci9054, Pci9080

Description:

Reads any mailbox register on the currently selected PCI device.

- *pExtension* is the device extension of the device object associated with the PLX PCI device for the PLX driver;
- *mailboxId* is the mailbox register ID; and,
- *returnCode* is a pointer to a buffer to store the return code.

Return Value:

This function returns the value read from the mailbox register. The status of the function call is returned via the *returnCode* parameter. The return codes are as follows:

Return Value	Description
`ApiSuccess`	The function returned successfully.
`ApiNullParam`	The *pExtension* parameter is NULL.
`ApiInvalidRegister`	The *mailboxId* parameter is not a valid mailbox ID.

Usage:

```
RETURN_CODE rc;
PDEVICE_OBJECT fdo;
U32 value;

value = Iop480RegisterMailboxRead(fdo->DeviceExtension, MailBox0,
                                  &rc);
if (rc != ApiSuccess)
  PlxKdPrint(("\n Error Reading a valid Mailbox"));
else
  PlxKdPrint(("\n Mailbox Read succeeded with a value of %x",
              value));
```

 © PLX Technology, Inc., 1999

…RegisterMailboxWrite

Syntax:

```
RETURN_CODE
Iop480RegisterMailboxWrite(  IN PDEVICE_EXTENSION pExtension,
                             IN MAILBOX_ID mailboxId,
                             IN U32 data);
```

Compatibility:

Iop480, Pci9054, Pci9080

Description:

Writes a value to any mailbox register on a PCI device.

- *pExtension* is the device extension of the device object associated with the PLX PCI device for the PLX driver;
- *mailboxId* is the mailbox register ID; and,
- *data* is a U32 value to store in the register.

Return Value:

Return Value	Description
`ApiSuccess`	The function returned successfully.
`ApiNullParam`	The *pExtension* parameter is NULL.
`ApiInvalidRegister`	The *mailboxId* parameter is not a valid mailbox ID.

Usage:

```
RETURN_CODE rc;
PDEVICE_OBJECT fdo;

rc = Iop480RegisterMailboxWrite(fdo->DeviceExtension, MailBox0,
                                0x3DC87A00);
if (rc != ApiSuccess)
  PlxKdPrint(("\n Error Writing a valid Mailbox"));
else
  PlxKdPrint(("\n OK: Valid Mailbox Write succeeded"));
```

 © PLX Technology, Inc., 1999

…RegisterReadAll

Syntax:

```
RETURN_CODE Iop480RegisterReadAll(  IN PDEVICE_EXTENSION pExtension,
                                    IN U32 startOffset,
                                    IN U32 registerCount,
                                    OUT PU32 buffer);
```

Compatibility:

Iop480, Pci9054, Pci9080

Description:

Reads multiple registers on a PCI device.

- *pExtension* is the device extension of the device object associated with the PLX PCI device for the PLX driver;
- *startOffset* is the register offset to start reading at;
- *registerCount* is the number of bytes to read starting at *startOffset*; and,
- *buffer* is the storage location for the register values.

Note: registerCount is the number of **bytes** to read. *Before this function can be used, the device extension must be fully initialized.*.

Return Value:

Return Value	Description
`ApiSuccess`	The function returned successfully.
`ApiNullParam`	The *pExtension* or the *buffer* parameter is NULL.
`ApiInsufficientResources`	There is no memory available for the PCI API.
`ApiInvalidRegister`	The *startOffset* and the *registerCount* parameters combined exceed the valid range of registers or *startOffset* is not on a 4-byte boundary.

Usage:

```
RETURN_CODE rc;
PDEVICE_OBJECT fdo;
U32 buffer[0x40];
rc = Iop480RegisterReadAll(fdo->DeviceExtension, 0x0, 0x100,
                           buffer);
if (rc != ApiSuccess)
  PlxKdPrint(("\n ERROR: Reading All registers failed API call"));
```

 © PLX Technology, Inc., 1999

…RegisterWrite

Syntax:

```
RETURN_CODE Iop480RegisterWrite(   IN PDEVICE_EXTENSION pExtension,
                                   IN U32 registerOffset,
                                   IN U32 data);
```

Compatibility:

Iop480, Pci9054, Pci9080

Description:

Writes a value to any register on a PCI device.

- *pExtension* is the device extension of the device object associated with the PLX PCI device for the PLX driver;
- *registerNumber* is the register number; and,
- *data* is a U32 value to store in the register.

Return Value:

Return Value	Description
`ApiSuccess`	The function returned successfully.
`ApiNullParam`	The *pExtension* parameter is NULL.
`ApiInvalidRegister`	The *registerOffset* parameter is out of range or is not on a 4-byte boundary.

Usage:

```
RETURN_CODE rc;
DEVICE_OBJECT fdo;

rc = Iop480RegisterWrite(fdo->DeviceExtension, 0x18, 0x234980C4);
if (rc != ApiSuccess)
  PlxKdPrint(("\n Error Writing a valid Register"));
else
  PlxKdPrint(("\n OK: Valid register Write succeeded"));
```

 © PLX Technology, Inc., 1999

…SerialEepromRead

Syntax:

```
RETURN_CODE Iop480SerialEepromRead(  IN PDEVICE_EXTENSION pExtension,
                                     IN EEPROM_TYPE eepromType,
                                     OUT PU32 buffer,
                                     IN U32 size);
```

Compatibility:

Iop480, Pci9054, Pci9080

Description:

Reads values from the configuration EEPROM.

- *pExtension* is the device extension of the device object associated with the PLX PCI device for the PLX driver;
- *eepromType* is the type of EEPROM on the PCI device;
- *buffer* is a pointer a buffer to store the data read; and,
- *size* defines the number of bytes you want to read from the EEPROM.

Return Value:

Return Value	Description
`ApiSuccess`	The function returned successfully.
`ApiNullParam`	The *pExtension* or *buffer* parameters are NULL.
`ApiInvalidSize`	The *size* parameter is 0, is too large for this *eepromType*, or is not 2 byte aligned.
`ApiFailed`	The function failed to access the serial EEPROM.
`ApiEepromNotPresent`	There is no EEPROM connected to the PLX Chip.
`ApiEepromTypeNotSupported`	The *eepromType* parameter is not supported for this PLX Chip.

Usage:

```
RETURN_CODE rc;
PDEVICE_OBJECT fdo;
U16 eepromData[0x16];          /* Holding data read from EEPROM */

/* Reading EEPROM into eepromData buffer */
rc = Iop480SerialEepromRead(fdo->DeviceExtension,
                            Eeprom93CS56,
```

 © PLX Technology, Inc., 1999

```
                                (U32 *)eepromData,
                                0x16 * sizeof(U16)); // size in bytes
if (rc != ApiSuccess)
{
  PlxKdPrint(("Error: Unable to read the serial EEPROM.\n"));
  return -1;
}
```

 © PLX Technology, Inc., 1999

…SerialEepromWrite

Syntax:

```
RETURN_CODE Iop480SerialEepromWrite(  IN PDEVICE_EXTENSION pExtension,
                                      IN EEPROM_TYPE eepromType,
                                      OUT PU32 buffer,
                                      IN U32 size);
```

Compatibility:

Iop480, Pci9054, Pci9080

Description:

Writes values to the configuration EEPROM.

- *pExtension* is the device extension of the device object associated with the PLX PCI device for the PLX driver;
- *eepromType* is the type of EEPROM on the PCI device;
- *buffer* is a pointer a buffer that contains the data; and,
- *size* defines the number of bytes you want to write to the EEPROM.

Return Value:

Return Value	Description
`ApiSuccess`	The function returned successfully.
`ApiNullParam`	The *pExtension* or *buffer* parameters are NULL.
`ApiInvalidSize`	The *size* parameter is 0, is too large for this *eepromType*, or is not 2 byte aligned.
`ApiFailed`	The function failed to acess the serial EEPROM.
`ApiInsufficientResources`	There is no memory available for the PCI API.
`ApiEepromNotPresent`	There is no EEPROM connected to the PLX Chip.
`ApiEepromTypeNotSupported`	The *eepromType* parameter is not supported for this PLX Chip.

Usage:

```
RETURN_CODE rc;
PDEVICE_OBJECT fdo;
U16 eepromData[0x16];          /* Contains valid data for EEPROM */
```

 © PLX Technology, Inc., 1999

```
/* Write eepromData buffer to the serial EEPROM */
rc = Iop480SerialEepromWrite(fdo->DeviceExtension,
                             Eeprom93CS56,
                             (U32 *)eepromData,
                             0x16 * sizeof(U16)); // size in bytes
if (rc != ApiSuccess)
{
  PlxKdPrint(("Error: Unable to write to the serial EEPROM.\n"));
  return -1;
}
```

 © PLX Technology, Inc., 1999

…UserRead

Syntax:

```
PIN_STATE Iop480UserRead(  IN PDEVICE_EXTENSION pExtension,
                           IN USER_PIN userPin,
                           OUT PRETURN_CODE returnCode);
```

Compatibility:

Iop480, Pci9054, Pci9080

Description:

Reads the PCI device's USERI pins.

- *pExtension* is the device extension of the device object associated with the PLX PCI device for the PLX driver;
- *userPin* is the USERI pin number to be read; and,
- *returnCode* is a pointer to a buffer to store the return code.

Return Value:

This function returns the state of the USER pin, being either `Active` or `Inactive`. The status of the function call is returned via the *returnCode* parameter. The return codes are as follows:

Return Value	Description
`ApiSuccess`	The function returned successfully.
`ApiNullParam`	The *pExtension* parameter is NULL.
`ApiInvalidUserPin`	This *userPin* is not present on this PLXChip.

Usage:

```
RETURN_CODE rc;
PDEVICE_OBJECT fdo;
PIN_STATE pinState;

pinState = Iop480UserRead(fdo->DeviceExtension, USER0, &rc);
if (rc != ApiSuccess)
  PlxKdPrint(("\n ERROR: Cannot Read USER In . RC = %x", rc));
else
  PlxKdPrint(("\n OK: USER0 pin was read as %x", pinState));
```

 © PLX Technology, Inc., 1999

…UserWrite

Syntax:

```
RETURN_CODE Iop480UserWrite(  IN PDEVICE_EXTENSION pExtension,
                              IN USER_PIN userPin,
                              IN PIN_STATE pinState);
```

Compatibility:

Iop480, Pci9054, Pci9080

Description:

Writes to the PCI device's USERO pins, setting them either `ACTIVE` or `INACTIVE`.

- *pExtension* is the device extension of the device object associated with the PLX PCI device for the PLX driver;
- *userPin* is the USERO pin number to be written; and,
- *pinState* is the new state to set the USERO pin.

Return Value:

Return Value	Description
`ApiSuccess`	The function returned successfully.
`ApiNullParam`	The *pExtension* parameter is NULL.
`ApiInvalidUserPin`	This *userPin* is not present on this PLX Chip.

Usage:

```
RETURN_CODE rc;
PDEVICE_OBJECT fdo;

rc = Iop480UserWrite(fdo->DeviceExtension, USER0, Active);
if (rc != ApiSuccess)
  PlxKdPrint(("\n ERROR: Cannot Write to USER Out. RC = %x", rc));
else
  PlxKdPrint(("\n OK: USER0 pin was written"));
```

 © PLX Technology, Inc., 1999

…DmaControl

Syntax:

```
RETURN_CODE Iop480DmaControl(PDEVICE_EXTENSION pDeviceExtension,
                             IN DMA_CHANNEL dmaChannel,
                             IN DMA_COMMAND dmaCommand);
```

Description:

Controls the DMA transfer for a given DMA channel.

- *pExtension* is the device extension of the device object associated with the PLX PCI device for the PLX driver;
- *dmaChannel* is the DMA channel number previously opened, and;
- *dmaCommand* is the action to perform on this DMA channel;

Note: *Before calling this function the appropriate DMA channel must be successfully opened using Iop480DmaBlockChannelOpen().*

Return Value:

Return Value	Description
`ApiSuccess`	The function returned successfully.
`ApiInvalidHandle`	The function was passed a handle which was not previously opened.
`ApiDmaChannelInvalid`	The *dmaChannel* parameter is not supported by this chip.
`ApiDmaChannelUnavailable`	The DMA channel was not opened by the selected device.
`ApiDmaNotPaused`	The DMA channel is in progress or done (return code returned when `DmaResume` command is requested and the DMA channel is not paused). Also will occur when attempting to pause a Flyby DMA transfer.
`ApiDmaDone`	Returned when attempting to pause or abort a non-active channel.
`ApiDmaCommandInvalid`	The *dmaCommand* parameter is invalid.

Comments:

This function can be used with any type of DMA transfer.

Note: *Under the current DMA architecture, this command cannot start a DMA transfer, as no DMA information is given with this function. So this command is useful only for pausing, restarting, or aborting DMA transfers.*

 © PLX Technology, Inc., 1999

Usage:

```
  PCI_MEMORY PciMemory;
  DMA_TRANSFER_ELEMENT dmaData;

/* PciMemory contains a valid PCI address */

  /* zero out the structure */
  memset(&dmaData, 0, sizeof(dmaData));

  dmaData.Iop480Dma.LowPciAddr = PciMemory.PhysicalAddr;
  dmaData.Iop480Dma.IopAddr = 0x80000;
  dmaData.Iop480Dma.TransferCount = 0x1000;
  dmaData.Iop480Dma.IopToPciDma = 0;
  dmaData.Iop480Dma.TerminalCountIntr = 0;

  if((rc=Iop480DmaBlockTransfer(fdo->deviceExtension,
                              PrimaryPciChannel0,
                              &dmaData))
     != ApiSuccess)
  {
    printf ("\n\nErrors in Block Dma.\n");
    printf ("Returned code %d\n",rc);
    getch();
    return -1;
  }

  if(rc=Iop480DmaControl(fdo->deviceExtension,
                        PrimaryPciChannel0,
                        DmaPause) != ApiSuccess)
  {
    printf ("\n\nCould not pause DMA transfer.\n");
    printf ("Returned code %d\n",rc);
    getch();
    return -1;
  }
```

 © PLX Technology, Inc., 1999

…DmaStatus

Syntax:

```
RETURN_CODE Iop480DmaStatus( IN PDEVICE_EXTENSION pExtension,
                             IN DMA_CHANNEL dmaChannel);
```

Compatibility:

Iop480, Pci9054, Pci9080

Description:

Returns the status for the DMA transfer of a given DMA channel.

- *pExtension* is the device extension of the device object associated with the PLX PCI device for the PLX driver; and,
- *dmaChannel* is the DMA channel number previously opened.

Note: *Before calling this function the appropriate DMA channel must be successfully opened using Iop480DmaBlockChannelOpen().*

Return Value:

Return Value	Description
`ApiSuccess`	The function returned successfully.
`ApiInvalidHandle`	The function was passed a handle which was not previously opened.
`ApiDmaChannelInvalid`	The *dmaChannel* parameter is not supported by this chip.
`ApiDmaChannelUnavailable`	The DMA channel was not opened by the selected device.
`ApiDmaDone`	The DMA channel is done.
`ApiDmaPaused`	The DMA channel is paused.
`ApiDmaInProgress`	The DMA channel is in progress.

Comments:

This function can be used with any type of DMA transfer.

Usage:

```
  if(rc=Iop480DmaBlockTransfer(fdo->deviceExtension,
                               PrimaryPciChannel0,
                               &dmaData)
                  != ApiSuccess)
  {
    printf ("\n\nErrors in Block Dma.\n");
    printf ("Returned code %d\n",rc);
```

 © PLX Technology, Inc., 1999

```
  getch();
  return -1;
}

rc = Iop480DmaStatus(plxHandle,
                  PrimaryPciChannel0);

printf ("Iop480DmaStatus returned code %d\n",rc);
getch();
return -1;
```

 © PLX Technology, Inc., 1999

…HotSwapIdRead

Syntax:

```
U8 Iop480HotSwapIdRead(IN PDEVICE_EXTENSION pExtension,
                       OUT PRETURN_CODE pReturnCode);
```

Compatibility:

Iop480, Pci9054

Description:

Reads the ID for the Hot Swap capability registers.

- *pExtension* is the device extension of the device object associated with the PLX PCI device for the PLX driver; and,
- *pReturnCode* is a pointer to the return value.

Return Value:

Return Value	Description
`ApiSuccess`	The function returned successfully.
`ApiInvalidHandle`	The function was passed a handle which was not previously opened.
`ApiConfigAccessFailed`	The driver was unable to read the configuration registers.

Usage:

```
  HANDLE drvHandle;
  RETURN_CODE rc;
  U8 hotSwapId;

  /* this function assumes we have a valid drvHandle */

  hotSwapId = Iop480HotSwapIdRead(fdo->deviceExtension,
                                  &rc);

  if (rc != ApiSuccess)
  {
      printf("Iop480HotSwapIdRead() failed, returned rc= %d\n",rc);
      return (-1);
  }
 return (hotSwapId);
```

 © PLX Technology, Inc., 1999

…HotSwapNcpRead

Syntax:

```
U8 Iop480HotSwapNcpRead(IN PDEVICE_EXTENSION pExtension,
                        IN PRETURN_CODE pReturnCode);
```

Compatibility:

Iop480, Pci9054

Description:

Reads the location of the next element in the linked list of capabilities. If this is the last item in the list, the value will return 0.

- *pExtension* is the device extension of the device object associated with the PLX PCI device for the PLX driver; and,
- *pReturnCode* is a pointer to the return value.

Return Value:

Return Value	Description
`ApiSuccess`	The function returned successfully.
`ApiInvalidHandle`	The function was passed a handle which was not previously opened.
`ApiConfigAccessFailed`	The driver was unable to read the configuration registers.

Usage:

```
HANDLE drvHandle;
RETURN_CODE rc;
U8 nextElement;

/* this function assumes we have a valid drvHandle */

nextElement = Iop480HotSwapNcpRead(fdo->deviceExtension,
                                   &rc);

if (rc != ApiSuccess)
{
    printf("Iop480HotSwapNcpRead() failed, returned rc= %d\n",rc);
    return (-1);
}
return(nextElement);
```

 © PLX Technology, Inc., 1999

…HotSwapStatus

Syntax:

```
U8 Iop480HotSwapStatus( IN PDEVICE_EXTENSION pExtension);
```

Compatibility:

Iop480, Pci9054

Description:

Returns the Hot-Swap status of the PLX chip.

- *pExtension* is the device extension of the device object associated with the PLX PCI device for the PLX driver;

Return Value:

If there is no error, the return value is an ORed combination of these bits:

Return Value	Description
HOT_SWAP_LED_IS_ON	The Hot Swap LED is on.
HOT_SWAP_BOARD_IS_BEING_REMOVED	The board is in process of being removed.
HOT_SWAP_BOARD_IS_BEING_INSERTED	The board was inserted and is being initialized.

However, if Hot Swap is not supported or not present on the board, the return value will be 0xFF.

Usage:

```
U8 hsStat = 0;

/* this function assumes we have a valid drvHandle */

hsStat = Iop480HotSwapStatus(fdo->deviceExtension);

if (hsStat == 0xFF)
{
    printf("Iop480HotSwapStatus() failed\n");
    return (-1);
}

if (hsStat & HOT_SWAP_LED_IS_ON)
    printf("The Hot Swap LED is on\n");
if (hsStat & HOT_SWAP_BOARD_IS_BEING_REMOVED)
    printf("The Hot Swap board is being removed.\n");
if (hsStat & HOT_SWAP_BOARD_IS_BEING_INSERTED)
    printf("The Hot Swap board is being inserted.\n");
return (0);
```

 © PLX Technology, Inc., 1999

…MuOutboundIndexRead

Syntax:

```
U32 Iop480MuOutboundIndexRead(IN PDEVICE_EXTENSION pExtension,
                              OUT PRETURN_CODE pReturnCode);
```

Compatibility:

Iop480

Description:

Reads the Host Outbound Index register to determine the number of IOP frames that were processed (Outbound Option).

- *pExtension* is the device extension of the device object associated with the PLX PCI device for the PLX driver; and,
- *pReturnCode* is a pointer to the return value.

Note: *Before this function can be used, a PCI device must be selected using PlxPciDeviceOpen() and the Messaging Unit must be initialized (initialized by the IOP application).*

Return Value:

Return Value	Description
`ApiSuccess`	The function returned successfully.
`ApiInvalidHandle`	The function was passed a handle which was not previously opened.

Usage:

```
RETURN_CODE rc;
U32 indexValue;

/* Read the Outbound Index value */
indexValue = Iop480MuOutboundIndexRead(fdo->deviceExtension, &rc);
if (rc != ApiSuccess)
{
    printf("Iop480MuOutboundIndexRead() failed,"
           " returned rc= %d\n",rc);
    return (-1);
}
return (indexValue);
```

 © PLX Technology, Inc., 1999

…MuOutboundIndexWrite

Syntax:

```
RETURN_CODE Iop480MuOutboundIndexWrite(IN PDEVICE_EXTENSION pExtension,
                                       IN PU32 pRegisterData);
```

Compatibility:

Iop480

Description:

Writes to the Host Outbound Index register to signal the IOP the number of frames that were processed (Outbound Option).

- *pExtension* is the device extension of the device object associated with the PLX PCI device for the PLX driver; and,
- *pRegisterData* is a pointer to the index value.

Return Value:

Return Value	Description
`ApiSuccess`	The function returned successfully.
`ApiInvalidHandle`	The function was passed a handle which was not previously opened.
`ApiNullParam`	The *drvHandle* or *framePointer* parameters are NULL.

Usage:

```
RETURN_CODE rc;
U32 indexValue;

/* Update the Outbound Index Value, */
/* This is an absolute index each time I call it! */
rc = Iop480MuOutboundIndexWrite(fdo->deviceExtension, &indexValue);
if (rc != ApiSuccess)
{
    printf("Iop480MuOutboundIndexRead() failed,"
           " returned rc= %d\n",rc);
    return (-1);
}
```

 © PLX Technology, Inc., 1999

…PciAbortAddrRead

Syntax:

```
U32 Iop480PciAbortAddrRead(IN PDEVICE_EXTENSION pExtension,
                           OUT PRETURN_CODE pReturnCode);
```

Compatibility:

Iop480

Description:

Returns the starting address to the operation that caused the Abort.

- *pExtension* is the device extension of the device object associated with the PLX PCI device for the PLX driver; and,
- *pReturnCode* a pointer to the return value.

Return Value:

Return Value	Description
`ApiSuccess`	The function returned successfully.
`ApiInvalidHandle`	The function was passes a handle that was improperly opened.

Usage:

```
RETURN_CODE rc;
U32 abortAddr;

/* Read the Pci Abort Address */
abortAddr = Iop480PciAbortAddrRead(fdo->deviceExtension, &rc);

if (rc != ApiSuccess)
{
    printf("Iop480PciAbortAddrRead() failed,"
           " returned rc= %d\n",rc);
    return (-1);
}
return (abortAddr);
```

 © PLX Technology, Inc., 1999

…PmIdRead

Syntax:

```
U8 Iop480PmIdRead(IN PDEVICE_EXTENSION pExetension,
                  OUT PRETURN_CODE pReturnCode);
```

Compatibility:

Iop480, Pci9054

Description:

Returns the Power Management new capabilities ID.

- *pExtension* is the device extension of the device object associated with the PLX PCI device for the PLX driver; and,
- *pReturnCode* a pointer to the return value.

Return Value:

Return Value	Description
`ApiSuccess`	The function returned successfully.
`ApiInvalidHandle`	The function was passes a handle that was improperly opened.

Usage:

```
RETURN_CODE rc;
U8 powerManId;


powerManId = Iop480PmIdRead(fdo->deviceExtension, &rc);

if (rc != ApiSuccess)
{
    printf("Iop480PmIdRead() failed, returned rc= %d\n",rc);
    return (-1);
}
return (powerManId);
```

 © PLX Technology, Inc., 1999

…PmNcpRead

Syntax:

```
U8 Iop480PmNcpRead( IN PDEVICE_EXTENSION pExtension,
                    OUT PRETURN_CODE pReturnCode);
```

Compatibility:

Iop480, Pci9054

Description:

Reads the location of the next element in the linked list of capabilities. If this is the last item in the list, the value will return 0.

- *pExtension* is the device extension of the device object associated with the PLX PCI device for the PLX driver; and,
- *pReturnCode* a pointer to the return code;

Return Value:

Return Value	Description
`ApiSuccess`	The function returned successfully.
`ApiInvalidHandle`	The function was passes a handle that was improperly opened.

Usage:

```
RETURN_CODE rc;
U8 nextElement;

nextElement = Iop480PmNcpRead(fdo->deviceExtension, &rc);

if (rc != ApiSuccess)
{
    printf("Iop480PmNcpRead() failed, returned rc= %d\n",rc);
    return (-1);
}
return(nextElement);
```

 © PLX Technology, Inc., 1999

…SerialEepromPresent

Syntax:

```
BOOLEAN Iop480SerialEepromPresent( IN PDEVICE_EXTENSION pExtension,
                                   IN PRETURN_CODE pReturnCode);
```

Compatibility:

Iop480, Pci9054, Pci9080

Description:

Determines via a read to the PLX chips registers if a Serial EEPROM device is found on the device selected.

- *pExtension* is the device extension of the device object associated with the PLX PCI device for the PLX driver; and,
- *pReturnCode* a pointer to the return value.

Return Value:

Return Value	Description
`ApiSuccess`	The function completed successfully.
`ApiInvalidHandle`	The function was passes a handle that was improperly opened.

Usage:

```
RETURN_CODE rc;
U8 nextElement;
BOOLEAN isPresent;

isPresent = Iop480SerialEepromPresent(fdo->deviceExtension, &rc)

if (rc != ApiSuccess)
{
    printf("Iop480SerialEepromPresent() failed\n");
    return (-1);
}
if (isPresent)
    printf ("Eeprom Dectected!\n");

return(0);
```

 © PLX Technology, Inc., 1999

…VpdIdRead

Syntax:

```
U8 Iop480IdRead(IN PDEVICE_EXTENSION pExtension,
                OUT PRETURN_CODE pReturnCode);
```

Compatibility:

Iop480, Pci9054

Description:

Returns the Power Management new capabilities ID.

- *pExtension* is the device extension of the device object associated with the PLX PCI device for the PLX driver; and,
- *pReturnCode* a pointer to the return value.

Return Value:

Return Value	Description
`ApiSuccess`	The function returned successfully.
`ApiInvalidHandle`	The function was passes a handle that was improperly opened.

Usage:

```
RETURN_CODE rc;
U8 vpdId;

vpdId = Iop480VpdIdRead(drvHandle, &rc);

if (rc != ApiSuccess)
{
    printf("Iop480VpdIdRead() failed, returned rc= %d\n",rc);
    return (-1);
}
return (vpdId);
```

 © PLX Technology, Inc., 1999

…VpdRead

Syntax:

```
U32 Iop480VpdRead(IN PDEVICE_EXTENSION pExtension,
                  IN U32 offset,
                  OUT PRETURN_CODE pReturnCode);
```

Compatibility:

Iop480, Pci9054

Description:

Reads the Vital Product Data from the specified byte offset.

- *pExtension* is the device extension of the device object associated with the PLX PCI device for the PLX driver;
- *offset* is the byte offset to read or write, it must be U32 aligned; and,
- *pReturnCode* a pointer to the return value.

Return Value:

Return Value	Description
`ApiSuccess`	The function returned successfully.
`ApiInvalidHandle`	The function was passes a handle that was improperly opened.

Usage:

```
RETURN_CODE rc;
U32 vpdData;

/* Read the Vital Product Data region */
for (offset = 0; offset <= MAX_VPD_OFFSET; offset +=4)
{
    vpdData = Iop480VpdRead(fdo->deviceExtension, offset, &rc);
    if (rc != ApiSuccess)
        break;

    printf("Vpd offset %d = 0x%08x\n",offset, vpdData);
}

if (rc != ApiSuccess)
{
    printf("Iop480VpdRead() failed, returned rc= %d\n",rc);
    return (-1);
}

return (0);
```

 © PLX Technology, Inc., 1999

…VpdNcpRead

Syntax:

```
U8 Iop480VpdNcpRead(IN PDEVICE_EXTENSION pExtension,
                    OUT PRETURN_CODE pReturnCode);
```

Compatibility:

Iop480, Pci9054

Description:

Read the location of the next element in the linked list of capabilities. If this is the last item in the list, the value will return 0.

- *pExtension* is the device extension of the device object associated with the PLX PCI device for the PLX driver; and,
- *pReturnCode* a pointer to the return code;

Return Value:

Return Value	Description
`ApiSuccess`	The function returned successfully.
`ApiInvalidHandle`	The function was passes a handle that was improperly opened.

Usage:

```
RETURN_CODE rc;
U8 nextElement;

/* this function assumes we have a valid drvHandle */

nextElement = Iop480VpdNcpRead(drvHandle, &rc);

if (rc != ApiSuccess)
{
    printf("Iop480VpdNcpRead() failed, returned rc= %d\n",rc);
    return (-1);
}
return(nextElement);
```

 © PLX Technology, Inc., 1999

…VpdWrite

Syntax:

```
RETURN_CODE Iop480VpdWrite(IN PDEVICE_EXTENSION pExtension,
                           IN U32 offset,
                           OUT PU32 pData);
```

Compatibility:

Iop480, Pci9054

Description:

Determines via a read to the PLX chips registers if a Serial EEPROM device is found on the device selected.

- *pExtension* is the device extension of the device object associated with the PLX PCI device for the PLX driver;
- *offset* is the byte offset to read or write, it must be U32 aligned; and,
- *pData* a pointer to the data to be written.

Return Value:

Return Value	Description
`ApiSuccess`	The function returned successfully.
`ApiInvalidHandle`	The function was passes a handle that was improperly opened.

Usage:

```
RETURN_CODE rc;
  PDEVICE_OBJECT fdo;

  if ((rc= Pci9054DmaSglChannelClose(fdo->DeviceExtension,
                                     PrimaryPciChannel0))
      != ApiSuccess)
  {
    PlxKdPrint(("\n\nErrors in closing channel.\n"));
    PlxKdPrint(("Returned code %d\n",rc));
    return;
  }
```

 © PLX Technology, Inc., 1999