DLL User's Guide for the AlphiDll

SOFTWARE REFERENCE

745-06-002-4000
Revision G
01/22/01

ALPHI Technology Corp.
6202 S. Maple Ave., #120
Tempe, AZ 85283
Tel: (480) 838-2428
Fax: (480) 838-4477
info@alphitech.com
www.alphitech.com

Table of Contents

Description

This document describes a Windows NT device driver and a 32 bit DLL which runs under Windows NT allowing for high level control of all of ALPHI Technology's PCI cards.

This package is intended to provide the base functionality which is common to all of our PCI cards. Specific functionality for certain cards is provided by other DLLs which rely upon this package.

This package allows the following tasks to be performed.

Finding out what cards are present and their capabilities.
Downloading DSP code to RAM, executing, and stopping under host control.
Communicating with the DSP via mailbox registers, doorbells, and FIFO registers.
Supporting memory mapped access to card resources.
Downloading DSP code and burning into FLASH for standalone BOOT.
Interrupting the Host processor.
Uniquely identifying board resources when identical boards exist in a host.
Utilizing PCI Bus Mastering to transfer data between the DSP and the host.

The API to this device driver and the DLL is described in this document, as well as in a Windows Helpfile in the Document directory.

The documentation includes the following descriptions.

Modules	The source files required to build the DLL portion of this board support package. Also, any global variables are listed here. Full source is provided for these files. (No source is provided for the device driver.)
HRESULTs	The return HRESULTs from this DLL and any other DLLs, and the associated text messges.
Classes and Members	The C++ classes, member functions, and member data implementing the functionality of this package. These classes are exported from the DLL for use by applications written in C++.
Functions	C functions implementing the functionality of this package. Many of these functions are exported for use by applications written in C or other languages.
IOCTLs	The primary means of communication between the DLL or user's application and the device driver.
Constants and Typedefs	Implementation details of this package.
Structures and Enumerations	Non class type data structures. Also used to pass data during IOCTL calls to the driver.

Device Names

User applications which wish to utilize the resources of ALPHI Technology's PCI cards tend to fall into two categories. There are applications which know the exact types of cards required in the host system in order to operate correctly, and those which do not really care about what type of card is present or the quantity, or will enumerate all the cards in a host system.

In order to facilitate both types of applications, the ALPHIPCI (and ALPHIPLX) device driver publishes two different names for each card, a generic name and a specific name. Both names are followed by a decimal digit which will uniquely identify the card. All the functionality of the driver for a particular card type is available regardless of which name was used to open the device.

The generic name is of the form ALPHIPCIn (or ALPHIPLXn) where n is a digit starting at 0. Every card handled by this driver will have a name of this form. An application wishing to enumerate the resources on the host, or interested in performing a task which is common to many different cards, (such as downloading and running DSP code, or burning a new DSP code into FLASH) can use the generic name in the Win32 API call to CreateFile. By opening each device in sequence (0, 1, 2...) until one fails for not existing, the application can enumerate every card in the host system. By using the IOCTL **IOCTL_ALPHIPCI_GET_DEVICE_CAPABILITIES**, the application can get the equivalent specific name (or true name) of the card, as well as the resources available.

This is in effect, exactly what the **AlphiPciDevice::Open** by number call does. All the known device drivers are opened, one at a time, to create a list of the true names of the cards present. Then this list is accessed to actually perform the open by number.

The specific name or true name is of a similiar form and depends upon the device of interest. For example, the CPCI-IPC card will have a specific name starting with CPCI-IPC0. This allows an application expecting a specific type of board to find it quickly.

Direct Access by DSP (PLX)

The PLX chip can be configured to directly access HOST memory and other PCI/CPCI cards by correctly programming the chip's registers. Several C routines are present in the alphi_io library (PLX Version) to accomplish this easily. For some cases, however, the support of the HOST program is needed, and some functions exist in AlphiDll to make this possible.

There are two different reasons to have the DSP directly accesses PCI space. The first is to access a separate PCI/CPCI card, and the second is to make use of the HOST memory, shared with the HOST processor. Normally, no help by the HOST program is needed to support accessing separate PCI/CPCI cards.

For the case where the DSP is accessing shared HOST memory, several functions exist to help with allocating a region of memory suitable for sharing with the DSP, and then making it accessible to the DSP.

AllocSharableMemory will allocate a buffer which is aligned to a 4k page boundary. This is important so that the DSP doesn't have to constantly deal with an offset to the start of the shared buffer. All the alphi_io functions related to the shared buffer rely upon expecting page jumps at exact 4k pages.

FreeSharableMemory frees the buffer when the program is done.

AlphiPciDevice::ShareHostMemory actually makes the memory sharable with the DSP. The device driver will lock the pages into physical memory (so that they can't be swapped to disk), and set up a mapping table of physical addresses for the use of the DSP. Although to the HOST program the memory appears contiguous, in fact, the physical pages are scattered through the physical memory space. For C programs, **ShareHostMemory** accomplishes the same thing.

AlphiPciDevice::UnshareHostMemory cancels the mapping. Again, use **UnshareHostMemory** for C programs.

DMA (AMCC)

The AMCC chip has two DMA channels, one in each direction, for transferring data between the card and the HOST. The DSP sees these DMA channels indirectly through the 8 DWORD FIFO present in the AMCC chip.

The device driver is programmed to emulate a scatter/gather type bus master DMA model. After transferring each page, the device driver receives an interrupt and sets up the transfer to the next page. Although it sounds as though this might take a significant amount of HOST processing time, the driver is only copying 3 DWORDS from a table to the AMCC to start the next page.

Making use of the DMA engine is very simple.

Access in C++

The functions **AlphiPciDevice::ReadFifo** and **AlphiPciDevice::WriteFifo** set up a transfer to or from the FIFO. The user provides a completion function and parameter which is called when the DMA is complete. Multiple transfers can be queued up to ensure continuous transfer. **AlphiPciDevice::CancelPendingReadRequests** and **AlphiPciDevice::CancelPendingWriteRequests** unqueues any pending requests.

Access in C and other languages

The functions **ReadFifo** and **WriteFifo** set up a transfer to or from the FIFO. The user provides a completion function and parameter which is called when the DMA is complete. Multiple transfers can be queued up to ensure continuous transfer. **CancelPendingReadRequests** and **CancelPendingWriteRequests** unqueues any pending requests.

DMA (PLX)

The PLX chip has two DMA channels. The device driver and DSP library utilize one in each direction, for transferring data between the card and the HOST. The DSP usually programs these DMA channels directly through the PLX registers.

The device driver is programmed to set up tables of physical addresses and sizes in DSP memory that describe each physical page of the buffer. The DSP is interrupted via a known doorbell, and the DSP then starts and throttles any transfer, a page at a time. When the DMA is complete, the DSP issues a known doorbell to the device driver. The driver then can set up the next transfer, and complete the previous one.

Making use of the DMA engine is very simple.

Access in C++

The functions **AlphiPciDevice::ReadFifo** and **AlphiPciDevice::WriteFifo** set up a transfer to or from the FIFO. The user provides a completion function and parameter which is called when the DMA is complete. Multiple transfers can be queued up to ensure continuous transfer. **AlphiPciDevice::CancelPendingReadRequests** and **AlphiPciDevice::CancelPendingWriteRequests** unqueues any pending requests.

Access in C and other languages

The functions **ReadFifo** and **WriteFifo** set up a transfer to or from the FIFO. The user provides a completion function and parameter which is called when the DMA is complete. Multiple transfers can be queued up to ensure continuous transfer.

CancelPendingReadRequests and **CancelPendingWriteRequests** unqueues any pending requests.

Doorbells (PLX)

The PLX has two 32 bit doorbell registers for communication between the HOST and the DSP. Certain bits are already used by the DSP library, the device driver, and this DLL.

Bit 31 of **PLX_REGS**.l2pdbell is used to indicate that the hardware generated an interrupt.

Bits 30 **-** 9 are available for user applications.

Bit 8 is used for DMA communication between the driver and the DSP library.

Bits 7 **-** 4 indicate that the associated outgoing mailbox is empty. Bit 4 is for mailbox 0.

Bits 3 **-** 0 indicate that the associated incoming mailbox is full. Bit 0 is for mailbox 0.

Bits 31 **-** 9 of **PLX_REGS**.p2ldbell are available for user applications.

Bit 8 is used for DMA communication between the driver and the DSP library.

Bits 7 **-** 4 tell the DSP that the associated incoming mailbox is empty. Bit 4 is for mailbox 0.

Bits 3 **-** 0 tell the DSP that the associated outgoing mailbox is full. Bit 0 is for mailbox 0.

Driver Installation

The Windows NT device driver can be set up in the host system by two means. First, a regular installation from the board support disks will set the driver up correctly.

Second, for customers creating installation disks for their products, if the ALPHIPCI.SYS (or ALPHIPLX.SYS) driver is copied to the WINNT/SYSTEM32/DRIVERS directory, and if the registry is set up, the system will see the device driver.

Setting up the system registry is easily accomplished by using the REGINI.EXE program supplied and the ALPHIPCI.INI (or ALPHIPLX.INI) file. By typing

REGINI ALPHIPCI.INI

the registry will be configured correctly. Reboot in order to complete the installation. Alternatively, the customer's installation program can set the registry to match the contents of the INI file.

Driver Load

The driver by default is set to load automatically at system boot. This can be easily changed by making use of the Settings / Control Panel / Devices control. By setting the ALPHIPCI (or ALPHIPLX) device, it can be started automatically at boot time, started manually, or disabled.

The device can be started manually via the same Control Panel control or from a command prompt, by typing (for instance)

```
net start alphipci
```

and stopped with

```
net stop alphipci
```

Confirmation of a successful load, or alternatively, the cause of a failure to load will be made in the Programs / Administration Tools / Event Viewer. The devices found, and their true names, will be reported for a successful load.

FIFOs

On an AMCC based card, there is an 8 DWORD FIFO in each direction between the host and the processor on cards which have them. This FIFO can be used for communication between the processors in a manner similiar to that of the mailbox registers. The advantage is increased decoupling between the two processors.

On the AMCC, the FIFO will be utilized as part of the bus mastering packet transfer to and from the card.

The AMCC 5933 chip does not allow interrupts to be generated as a result of FIFO activity.

There is no corresponding FIFO on a PLX based design. The PLX can, however, directly write to the HOST memory, both by DMA and by direct access.

Function Returns (HRESULTs)

The DLLs which make up the Board Support Package return a success or error code depending on the results of the function. These HRESULT values are compatible with the error codes returned by WIN32 functions and COM/OLE.

One advantage of using these error codes is that there is a textual description of the meaning of the error code available in the DLL AlphiErrorCode, along with some useful functions to display the text to the console or to a MessageBox. See DisplayErrorCode.h for more details.

WIN32 provides two macros which are helpful in error checking. It is a good idea to use these macros, since the actual error codes which are returned may be added to in the future.

FAILED() will be true if the HRESULT is an error.

SUCCEEDED() will be true if the HRESULT is anything but an error.

Results can be classified into 4 catagories, and the HRESULT names reflect these catagories.

ALPHI_SEVERITY_SUCCESS:
ALPHI_S_...
Operation succeeded with the specified result.

ALPHI_SEVERITY_INFORMATIONAL:
ALPHI_I_...

Operation succeeded, but there is some information also. Message boxes will be displayed with the blue "i" icon.

ALPHI_SEVERITY_WARNING:
ALPHI_W_...
Operation succeeded, but there is something wrong. Message boxes will be displayed with the yellow "!" icon.

ALPHI_SEVERITY_ERROR:
ALPHI_E_...
Operation failed. Message boxes will be displayed with the red stop sign icon.

Developer Notes

Note for users of earlier versions of AlphiDll. All of the earlier S_OK and E_FAIL HRESULTs have been replaced with these more meaningful returns. If you were previously making use of the FAILED() and SUCCEEDED() macros, your code should not need any changes.

All the MessageBoxes which poped up as a result of failures in functions have been removed. The user now should present an appropriate message using the AlphiErrorCode DLL or other means.

Host Interrupts (AMCC)

It is possible to have the card interrupt the host processor as a result various causes.

An interrupt can be generated when a single mailbox is written to by the processor on the card or by the hardware as on a non intelligent board like the CPCI-SIP. Of less use, an additional interrupt can be generated when the processor on the card reads a mailbox.

Access in C++

The function **AlphiPciDevice::HookMailboxInterrupt** allows the user to specify the mailboxes of interest, and to provide the ALPHIDLL the user's function to call when an interrupt occurs. **AlphiPciDevice::UnhookMailboxInterrupt** turns off the interrupt.

Access in C and other languages

The function **HookMailboxInterrupt** allows the user to specify the mailboxes of interest, and to provide the ALPHIDLL the user's function to call when an interrupt occurs. **UnhookMailboxInterrupt** turns off the interrupt.

Direct access to the driver

There are several IOCTL calls which provide support for host interrupts.

IOCTL_ALPHIPCI_WAIT_FOR_INTERRUPT Blocks waiting for an interrupt to occur. Returns the value of the **AMCC**.INTCSR at the time of the interrupt. This tells the application whether the mailbox read or the mailbox write caused the interrupt. If an interrupt has already occured before the wait for it, this IOCTL will immediately return that status.

IOCTL_ALPHIPCI_SELECT_MAILBOX_FOR_INTERRUPT Enables and disables the mailbox interrupts. It is better to use this IOCTL instead of directly accessing the **AMCC**.INTCSR register since the driver is reading and writing this register too. This IOCTL serializes access to the register.

IOCTL_ALPHIPCI_CANCEL_SPECIFIC_IO Allows the cancellation of an outstanding **IOCTL_ALPHIPCI_WAIT_FOR_INTERRUPT** by any thread.

Performance issues

The interrupt latentcy on a Pentium 166 INTEL motherboard has been profiled at less than 400 uS with a typical latency much lower. This required that the task priority class be elevated to realtime. Otherwise, other events and tasks running on NT will cause the thread calling the user's completion routine to not run immediately after the NT driver unblocks it.

It is important to note that the completion routine must return before the thread can call it again. No interrupts will be lost as a result of taking too long, but the latency will obviously be high.

Host Interrupts (PLX)

It is possible to have the card interrupt the host processor as a result various causes.

An interrupt can be generated when a mailbox is written or read by the DSP, a hardware event has occured, or a bit in the doorbell register has been set. Certain doorbell bits are reserved for use by the device driver, alphi_dll, and alphi_io for maintaining mailbox states and passing DMA information.

Access in C++

The function **AlphiPciDevice::HookMailboxInterrupt** allows the user to causes of interest, and to provide the ALPHIDLL the user's function to call when an interrupt occurs. **AlphiPciDevice::UnhookMailboxInterrupt** turns off the interrupt.

Access in C and other languages

The function **HookMailboxInterrupt** allows the user to specify the causes of interest, and to provide the ALPHIDLL the user's function to call when an interrupt occurs. **UnhookMailboxInterrupt** turns off the interrupt.

Direct access to the driver

There are several IOCTL calls which provide support for host interrupts.

IOCTL_ALPHIPCI_WAIT_FOR_INTERRUPT Blocks waiting for an interrupt to occur. Returns the value of the **PLX**.intcsr at the time of the interrupt. This allows the application to determine the cause of the interrupt. If an interrupt has already occured before the wait for it, this IOCTL will immediately return that status.

IOCTL_ALPHIPCI_SELECT_MAILBOX_FOR_INTERRUPT Enables and disables the interrupts. Please note that the driver always has interrupts turned on, and this call tells the driver which causes are to be passed down.

IOCTL_ALPHIPCI_CANCEL_SPECIFIC_IO Allows the cancellation of an outstanding **IOCTL_ALPHIPCI_WAIT_FOR_INTERRUPT** by any thread.

It is important to note that the completion routine must return before the thread can call it again. No interrupts will be lost as a result of taking too long, but the latency will obviously be high.

Language Issues

The exports from this DLL are in a form which matches those of the WIN 32 API DLLs under Windows NT. Therefore, the DLLs should be callable from any 32 bit programming language available under NT. However, a header file may have to be created for languages other than Microsoft Visual C/C++.

This DLL was compiled under Microsoft Visual C/C++ Version 5.0.

Linking to this DLL

The import library is named AlphiDll.lib and is located in the Library directory. If any error reporting calls are made, then AlphiErrorCode.lib should be included too. The customer should be add this file to the list of files to link with. This should resolve any function references.

If AlphiDll.dll (and possibly AlphiErrorCode.dll) is placed in the directory with the end application, or in the WINNT\SYSTEM32 directory, the system will find them when the customer's application is started. Alternatively, the directory containing the DLLs can be placed on the PATH.

Mailboxes

Mailboxes provide a primary means of transfering information between the host and the processor on the card.

On an AMCC based design, there are three 32 bit mailbox registers available for use. The fourth mailbox is only useable as 24 bits because the high byte is used as a hardware register in the configuration used. The high byte reflects the current hardware state in certain cards, such as the CPCI-SIP where it indicates which IP is generating an interrupt.

It is possible to generate a host interrupt when a mailbox is accessed. An interrupt can be generated when a single mailbox is written to by the processor on the card or by the hardware as on the CPCI-SIP. Of less use, an additional interrupt can be generated when the processor on the card reads a mailbox. See Host Interrupts for more details.

On a PLX based design, there are eight mailboxes total, without any empty or full flags. This DLL and the alphi_io DSP library work together to allow 4 mailboxes in each direction, and they use certain doorbell bits to maintain empty/full flags. The doorbell registers will generate interrupts to the HOST device driver, which can be hooked by the user.

Maps

Certain PCI cards make use of additional resources which can be directly accessed from the host processor. Examples include the IPs installed in a CPCI-SIP card, and the Dual Port Ram of the PCI4PACK. Additionally, every card makes use of either the AMCC register set or the PLX register set, which is also directly accessible from the host processor. ALPHI Technology has specified that these resources be memory mapped, in order that they can be directly accessed from user tasks. If these had been I/O mapped, they would have required constant calls into the device driver to read and write ports, and the user application would pay a performance penalty.

The user application can get a pointer to these resources by using the **AlphiPciDevice::Map** function. This asks that the driver create a mapping in the page table for the application task to directly access the physical card. The resources available are listed in **eTypeOfAccess**.

When the task is finished with the mapping, it can be released via **AlphiPciDevice::Unmap**.

Software Modules

The source files required to build the DLL portion of this board support package. Also, any global variables are listed here. Full source is provided for these files. (No source is provided for the device driver.)

Module AlphiDll.cpp

Filename: D:/ALPHIPCI/ALPHIDLL/ALPHIDLL.CPP

Description Function definitions for the custom DLL to interface to all ALPHIPCI devices.

Global Variables **const char *gszFriendlyName = "ALPHI PCI Device Interface DLL";**
Name of the DLL for presentation in message boxes.

Module AlphiDll.h

Filename: D:/ALPHIPCI/INCLUDE/ALPHIDLL.H

Description Function declarations for the custom DLL to interface to all ALPHIPCI devices.

Module Amcc.cpp

Filename: D:/ALPHIPCI/ALPHIDLL/AMCC.CPP

Description Specific functionality for Amcc 5933 chip.

Parameters *Version*
Reference to the returned version identifier.

Module Amcc.h

Filename: D:/ALPHIPCI/ALPHIDLL/AMCC.H

Description Specific functionality for Amcc 5933 chip.

Module AmccPort.cpp

Filename: D:/ALPHIPCI/ALPHIDLL/AMCCPORT.CPP

Description Specific functionality for Amcc 5933 chip in I/O mode.

Parameters *Version*
Reference to the returned version identifier.

Module AmccPort.h

Filename: D:/ALPHIPCI/ALPHIDLL/AMCCPORT.H

Description Specific functionality for Amcc 5933 chip in I/O mode.

Module Base.cpp

Filename: D:/ALPHIPCI/ALPHIDLL/BASE.CPP

Description Base functionality for PCI Interface chips.

Module Base.h

Filename: D:/ALPHIPCI/ALPHIDLL/BASE.H

Description Base functionality for PCI Interface chips.

Module ddalphip.h

Filename: D:/ALPHIPCI/INCLUDE/DDALPHIP.H

Description User interface to the ALPHIPCI and ALPHIPLX Windows NT device driver.

Module DisplayErrorCode.cpp

Filename: D:/ALPHIPCI/HOST EXAMPLES/ALPHIERRORCODE/DISPLAYERRORCODE.CPP

Description Functions to simplify outputting textual meanings of HRESULTs.

Module DisplayErrorCode.h

Filename: D:/ALPHIPCI/INCLUDE/DISPLAYERRORCODE.H

Description Functions to simplify outputting textual meanings of HRESULTs.

Module ErrorCode.h

Filename: D:/ALPHIPCI/INCLUDE/ERRORCODE.H

Description HRESULTs returned by all DLL functions and their associated textual messages.

Module Plx.cpp

Filename: D:/ALPHIPCI/ALPHIDLL/PLX.CPP

Description Specific functionality for PLX 9080 chip.

Parameters *Page*
Select which IP space is mapped into unified space.

Module Plx.h

Filename: D:/ALPHIPCI/ALPHIDLL/PLX.H

Description Specific functionality for Plx 9080 chip.

HRESULTs and associated messages

The return HRESULTs from this DLL and any other DLLs, and the associated text messges.

ALPHI_E_BAD_CARD_CONFIGURATION

Message Text: This function requires the correct PCI configuration for the device in order to succeed. Correct the NVRAM configuration for the device.

Defined in: D:/ALPHIPCI/INCLUDE/ERRORCODE.H

ALPHI_E_BAD_FILENAME

Message Text: The function failed because the specified file could not be opened.

Defined in: D:/ALPHIPCI/INCLUDE/ERRORCODE.H

ALPHI_E_CANT_GET_DEVICE_CAPABILITIES

Message Text: Open is unable to get the device capabilities from the device driver.

Defined in: D:/ALPHIPCI/INCLUDE/ERRORCODE.H

ALPHI_E_CANT_SUPPORT_THIS_DEVICE

Message Text: This software component does not support this type of device.

Defined in: D:/ALPHIPCI/INCLUDE/ERRORCODE.H

ALPHI_E_DEVICE_FAILED_RESET

Message Text: The DSP has failed to respond as expected to a RESET. The DSP is supposed to restart the BootRom. If using an emulator, ensure that the DSP is free to run.

Defined in: D:/ALPHIPCI/INCLUDE/ERRORCODE.H

ALPHI_E_DEVICE_NOT_OPEN

Message Text: Operation failed because this object is not connected to an open device.

Defined in: D:/ALPHIPCI/INCLUDE/ERRORCODE.H

ALPHI_E_DSP_FAILED_ACK_COMMAND_PACKET

Message Text: The DSP failed to acknowledge a command packet.

Defined in: D:/ALPHIPCI/INCLUDE/ERRORCODE.H

ALPHI_E_DSP_FAILED_COMMAND_PACKET

Message Text: The DSP failed to perform the command packet.

Defined in: D:/ALPHIPCI/INCLUDE/ERRORCODE.H

ALPHI_E_DSP_FAILED_READ_COMMAND_PACKET

Message Text: The DSP failed to read a mailbox while sending a command packet.

Defined in: D:/ALPHIPCI/INCLUDE/ERRORCODE.H

ALPHI_E_DSP_NOT_RESPONDING

Message Text: The DSP is not currently running the BootRom, or has failed to respond as expected. Perhaps a downloaded program has overwritten the BootRom. Check the memory map of the downloaded DSP code.

Defined in: D:/ALPHIPCI/INCLUDE/ERRORCODE.H

ALPHI_E_DSP_NOT_RUNNING_KERNEL

Message Text: The DSP is not currently running the BootRom, or has failed to respond as expected. If using an emulator, ensure that the DSP is free to run.

Defined in: D:/ALPHIPCI/INCLUDE/ERRORCODE.H

ALPHI_E_ERROR_PARSING_FILE

Message Text: The function ran into an unexpected error reading or parsing the file.

Defined in: D:/ALPHIPCI/INCLUDE/ERRORCODE.H

ALPHI_E_FAILED_TO_MAP_REGION

Message Text: A necessary region to control the card could not be mapped into the address space.

Defined in: D:/ALPHIPCI/INCLUDE/ERRORCODE.H

ALPHI_E_FAILED_TO_OPEN_FILE_FOR_DOWNLOAD

Message Text: The appropriate DSP code file could not be opened for download to the card.

Defined in: D:/ALPHIPCI/INCLUDE/ERRORCODE.H

ALPHI_E_FAILED_TO_START_DOWNLOAD

Message Text: The DSP failed to start the downloaded DSP code.

Defined in: D:/ALPHIPCI/INCLUDE/ERRORCODE.H

ALPHI_E_FUNCTION_NOT_APPLICABLE

Message Text: This function is not applicable to this device's capabilities.

Defined in: D:/ALPHIPCI/INCLUDE/ERRORCODE.H

ALPHI_E_HOOK_ALREADY_SET

Message Text: Only one routine can handle interrupts. Unhook the previous one.

Defined in: D:/ALPHIPCI/INCLUDE/ERRORCODE.H

ALPHI_E_MAP_NOT_APPLICABLE

Message Text: The requested region is not applicable to this device's capabilities.

Defined in: D:/ALPHIPCI/INCLUDE/ERRORCODE.H

ALPHI_E_NO_DEVICE_BY_THIS_NAME

Message Text: Open failed because no device by this name or number was found.

Defined in: D:/ALPHIPCI/INCLUDE/ERRORCODE.H

ALPHI_E_NO_DOWNLOAD

Message Text: No DSP code has been downloaded to this card.

Defined in: D:/ALPHIPCI/INCLUDE/ERRORCODE.H

ALPHI_E_OPERATION_TIMED_OUT

Message Text: The operation did not complete within the time limit.

Defined in: D:/ALPHIPCI/INCLUDE/ERRORCODE.H

ALPHI_E_PARM_OUT_OF_RANGE

Message Text: The function failed because a parameter was out of range.

Defined in: D:/ALPHIPCI/INCLUDE/ERRORCODE.H

ALPHI_E_UNEXPECTED_CONDITION

Message Text: An unexpected condition has occurred inside the support library. Contact the factory.

Defined in: D:/ALPHIPCI/INCLUDE/ERRORCODE.H

ALPHI_E_WRONG_PROCESSOR

Message Text: This DSP code is not compiled for the type of DSP present on the device.

Defined in: D:/ALPHIPCI/INCLUDE/ERRORCODE.H

ALPHI_I_FUNCTION_NOT_NEEDED

Message Text: The function was not performed because it was in the state already.

Defined in: D:/ALPHIPCI/INCLUDE/ERRORCODE.H

ALPHI_I_OPEN_LIMITED_FUNCTIONALITY

Message Text: Device was opened for limited functionality.

Defined in: D:/ALPHIPCI/INCLUDE/ERRORCODE.H

ALPHI_S_DSP_ACK

Message Text: The DSP acknowledges that the operation was successful.

Defined in: D:/ALPHIPCI/INCLUDE/ERRORCODE.H

ALPHI_S_OK

Message Text: Operation was successful.

Defined in: D:/ALPHIPCI/INCLUDE/ERRORCODE.H

ALPHI_W_DEVICE_FAILED_RESET

Message Text: The DSP has failed to respond as expected to a RESET. If using an emulator, ensure that the DSP is free to run. Functionality will be limited for this device.

Defined in: D:/ALPHIPCI/INCLUDE/ERRORCODE.H

ALPHI_W_OPEN_FAILED_TO_MAP_PCI_REGISTERS

Message Text: This device has an incorrect PCI configuration. Operations on this card will be limited until the NVRAM configuration for the device is corrected.

Defined in: D:/ALPHIPCI/INCLUDE/ERRORCODE.H

Classes and Members

The C++ classes, member functions, and member data implementing the functionality of this package. These classes are exported from the DLL for use by applications written in C++.

AlphiPciDevice Class

class AlphiPciDevice

This class represents one card in the host system. It provides the functionality to communicate with all of the resources available on the card.

Defined in: D:/ALPHIPCI/INCLUDE/ALPHIDLL.H

Comments

This class is exported from the DLL for direct use by Microsoft compatible compilers. I do not believe that Microsoft name mangling will work for Borland or other compilers.

Class Members

Public Members

__stdcall AlphiPciDevice()
Constructor.

__stdcall ~AlphiPciDevice()
Destructor.

HRESULT __stdcall Open(const char *pszLinkName, bool bDoNotReset = false)
Open the specified device by name.

HRESULT __stdcall Open(unsigned BoardNumber, bool bDoNotReset = false)
Open the specified device by consecutive device number.

HRESULT __stdcall Close()
Close the specified device and cleanup any resources.

bool __stdcall IsOpen()
Ensure that this class is opened and representing a valid card.

Public Members for control of DSP

HRESULT __stdcall Reset()
Reset the DSP and/or any IPs. If a DSP, ensure successful communication with kernel.

HRESULT __stdcall IsKernelRunning()
Ensure that the DSP can respond to requests from this API.

HRESULT __stdcall RetrieveSoftwareVersion(DWORD &Version)
Retrieve version of the bootloader (or application if supported).

HRESULT __stdcall Download(const char *szFilename)
Download the COFF file to the DSP.

HRESULT __stdcall Start()
Begin execution of the downloaded DSP code.

HRESULT __stdcall DownloadX0(const char *szFilename)
Download the Tektronix X0 file to the buffer on the card.

HRESULT __stdcall StartX0()
Begin execution of the previously downloaded Tektronix file.

HRESULT __stdcall BurnBootFlashX0()
Burn the boot area of the FLASH with the previously downloaded Tektronix file.

HRESULT __stdcall BurnUserFlashX0()
Burn the user area of the FLASH with the previously downloaded Tektronix file.

Public Members to directly access card resources

HRESULT __stdcall Map(eTypeOfAccess TypeOfAccess, LinearAddress &Address)
Retrieve a pointer to the physical card resource.

HRESULT __stdcall Unmap(LinearAddress &Address)
Unmap pointer to physical resource.

HRESULT __stdcall SelectPage(DWORD Page)
Select page of IP memory space mapped to region.

Public Members to read and write to DSP memory

HRESULT __stdcall WriteDword(DWORD DspAddress, DWORD Value)
Ask the DSP to write to the specified DSP address.

HRESULT __stdcall ReadDword(DWORD DspAddress, DWORD *pValue)
Ask the DSP to read from the specified DSP address.

Public Members to directly read and write to the FIFOs

HRESULT __stdcall ReadFifoDirect(DWORD *pBuffer, DWORD NumDWords)
Read directly from the FIFO port.

HRESULT __stdcall ReadFifoDirect(WORD *pBuffer, DWORD NumDWords)
Read directly from the FIFO port.

HRESULT __stdcall ReadFifoDirect(BYTE *pBuffer, DWORD NumDWords)
Read directly from the FIFO port.

HRESULT __stdcall WriteFifoDirect(DWORD *pBuffer, DWORD NumDWords)
Write directly to the FIFO port.

HRESULT __stdcall WriteFifoDirect(WORD *pBuffer, DWORD NumDWords)
Write directly to the FIFO port.

HRESULT __stdcall WriteFifoDirect(BYTE *pBuffer, DWORD NumDWords)
Write directly to the FIFO port.

HRESULT __stdcall WriteFifo(DWORD *pBuffer, DWORD NumDWords, ucr CompletionFunction, PVOID UserData)
Write to the FIFO port using Bus Master DMA.

HRESULT __stdcall ReadFifo(DWORD *pBuffer, DWORD NumDWords, ucr CompletionFunction, PVOID UserData)
Read from the FIFO port using Bus Master DMA.

Public Members to directly read and write to the mailboxes

HRESULT __stdcall WriteMbox(WORD WhichMailbox, DWORD dwData)
Write a DWORD to the DSP.

HRESULT __stdcall WriteMbox(WORD WhichMailbox, DWORD dwData, bool fWait)
Write a DWORD to the DSP.

HRESULT __stdcall ReadMbox(WORD WhichMailbox, bool fWait, DWORD *pData = 0)
Read a DWORD from the DSP.

HRESULT __stdcall ReadMbox(WORD WhichMailbox, bool fWait, WORD *pData)
Read a DWORD from the DSP and cast to WORD.

HRESULT __stdcall ReadMbox(WORD WhichMailbox, bool fWait, BYTE *pData)
Read a DWORD from the DSP and cast to BYTE.

Public Members to support host interrupts

HRESULT __stdcall HookMailboxInterrupt(eIntType IntType, UsersIntCompletionRoutine uicr, void *UsersInterruptData)
Install a hook to call *uicr* as a result of mailbox activity.

HRESULT __stdcall UnhookMailboxInterrupt()
Remove a hook installed with **HookMailboxInterrupt**.

Public miscellaneous

const DeviceCaps & __stdcall GetDeviceCapabilities()
Get reference to device capabilities.

HRESULT __stdcall CancelPendingReadRequests()
Cancel any pending read requests.

HRESULT __stdcall CancelPendingWriteRequests()
Cancel any pending write requests.

HRESULT __stdcall ReadAmccNvram(AmccNvramImage &Image)
Read the NVRAM image stored in the device.

HRESULT __stdcall WriteAmccNvram(AmccNvramImage &Image)
Write the NVRAM image to the device.

HRESULT __stdcall ReadPlxNvram(PlxNvramImage &Image)
Read the NVRAM image stored in the device.

HRESULT __stdcall WritePlxNvram(PlxNvramImage &Image)
Write the NVRAM image to the device.

HRESULT ShareHostMemory(void *pSharedMemory, DWORD Size)
Share the designated buffer with the DSP.

HRESULT UnshareHostMemory()
Disable sharing the buffer with the DSP.

Private Members

BaseAlphiDevice * pDevice
Virtual base class of the class which implements the actual functions.

DeviceCaps m_DeviceCaps
Device capabilities reported by driver.

AlphiPciDevice::AlphiPciDevice

AlphiPciDevice::AlphiPciDevice()

Constructor.

Defined in: D:/ALPHIPCI/ALPHIDLL/ALPHIDLL.CPP

Comments

Initializes member variables to a known state.

Developer Notes

Should initialize m_DeviceCaps.

AlphiPciDevice::BurnBootFlashX0

HRESULT AlphiPciDevice::BurnBootFlashX0()

Burn the boot area of the FLASH with the previously downloaded Tektronix file.

Defined in: D:/ALPHIPCI/ALPHIDLL/ALPHIDLL.CPP

Return Codes

ALPHI_S_OK	Operation was successful.
Otherwise	Operation failed. See **HRESULTs**.

See Also

DownloadX0

AlphiPciDevice::BurnUserFlashX0

HRESULT AlphiPciDevice::BurnUserFlashX0()

Burn the user area of the FLASH with the previously downloaded Tektronix file.

Defined in: D:/ALPHIPCI/ALPHIDLL/ALPHIDLL.CPP

Return Codes

ALPHI_S_OK	Operation was successful.
Otherwise	Operation failed. See **HRESULTs**.

See Also

DownloadX0

AlphiPciDevice::CancelPendingReadRequests

HRESULT AlphiPciDevice::CancelPendingReadRequests()

Cancel any pending read requests.

Defined in: D:/ALPHIPCI/ALPHIDLL/ALPHIDLL.CPP

Comments Cancels any outstanding read packets and cleans up.

Return Codes

ALPHI_S_OK	Operation was successful.
Otherwise	Operation failed. See **HRESULTs**.

See Also **ReadFifo**

AlphiPciDevice::CancelPendingWriteRequests

HRESULT AlphiPciDevice::CancelPendingWriteRequests()

Cancel any pending write requests.

Defined in: D:/ALPHIPCI/ALPHIDLL/ALPHIDLL.CPP

Comments Cancels any outstanding read packets and cleans up. Resets the PCI to DSP FIFO on AMCC.

Return Codes

ALPHI_S_OK	Operation was successful.
Otherwise	Operation failed. See **HRESULTs**.

See Also **WriteFifo**

AlphiPciDevice::Close

HRESULT AlphiPciDevice::Close()

Close the specified device and cleanup any resources.

Defined in: D:/ALPHIPCI/ALPHIDLL/ALPHIDLL.CPP

Comments Unhook interrupt, delete the PCI register mappings, close the handle, and reinitialize the member variables.

Return Codes

ALPHI_S_OK	Operation was successful.
Otherwise	Operation failed. See **HRESULTs**.

Developer Notes Should initialize m_DeviceCaps.

AlphiPciDevice::Download

HRESULT AlphiPciDevice::Download(
    const char * *szFilename***)**

Download the COFF file to the DSP.

Defined in: D:/ALPHIPCI/ALPHIDLL/ALPHIDLL.CPP

Parameters *szFilename*
NULL terminated filename.

Return Codes

ALPHI_S_OK	Operation was successful.
Otherwise	Operation failed. See **HRESULTs**.

Developer Notes This function has been fixed to support all three COFF versions defined (-v0, -v1, and -v2).

AlphiPciDevice::DownloadX0

HRESULT AlphiPciDevice::DownloadX0(
const char * *szFilename***)**

Download the Tektronix X0 file to the buffer on the card.

Defined in: D:/ALPHIPCI/ALPHIDLL/ALPHIDLL.CPP

Parameters *szFilename*
NULL terminated filename.

Return Codes

ALPHI_S_OK	Operation was successful.
Otherwise	Operation failed. See **HRESULTs**.

See Also **StartX0 BurnBootFlashX0 BurnUserFlashX0**

AlphiPciDevice::GetDeviceCapabilities

const DeviceCaps & __stdcall AlphiPciDevice::GetDeviceCapabilities()

Get a reference to the Device Capabilities.

Defined in: D:/ALPHIPCI/INCLUDE/ALPHIDLL.H

Return Value Constant reference to the **DeviceCaps** stored during the **Open**.

See Also **DeviceCaps**

AlphiPciDevice::HookMailboxInterrupt

HRESULT AlphiPciDevice::HookMailboxInterrupt(
eIntType *IntType***,**
UsersIntCompletionRoutine *UsersInterruptRoutine***,**
void * *UsersInterruptData***)**

Install a hook to call *UsersInterruptRoutine* as a result of a valid interrupt cause.

Defined in: D:/ALPHIPCI/ALPHIDLL/ALPHIDLL.CPP

Parameters

IntType
Which cause(s) to look at. See **eIntType** for AMCC class devices. PLX device causes have not been defined in the header file as yet.

UsersInterruptRoutine
Pointer to the user's completion routine. See **(*UsersIntCompletionRoutine)**.

UsersInterruptData
User data passed to the completion function.

Comments

On the AMCC, an interrupt can be generated from up to two causes at a time. One interrupt cause is the DSP writing to a particular mailbox, or on certain cards, a hardware event could generate an interrupt. The other cause is generated by the DSP reading from a particular mailbox.

On the PLX, an interrupt can be generated on any of the following: DSP read of a mailbox, DSP write to a mailbox, a hardware event, or a doorbell write by the DSP.

This function operates by creating a separate thread running at the highest priority. This thread blocks in the device driver until the interrupt event has occurred. When the thread is freed, it calls *UsersInterruptRoutine* with the parameter *UsersInterruptData* and the state of the interrupts at the time of calling. When the user function completes, the thread then blocks again in the device driver. It is impossible not to be called from an interrupt cause, but it is possible to have several interrupt events combined into one call to the user function, if they occur fast enough.

Developer Notes

This function really deserves a better name.

Return Codes

ALPHI_S_OK	Operation was successful.
Otherwise	Operation failed. See **HRESULTs**.

See Also

UnhookMailboxInterrupt (*UsersIntCompletionRoutine)

AlphiPciDevice::IsKernelRunning

HRESULT AlphiPciDevice::IsKernelRunning()

Ensure that the DSP is in HOST control mode and can respond to requests from this API.

Defined in: D:/ALPHIPCI/ALPHIDLL/ALPHIDLL.CPP

Comments

Exchange tokens with the kernel to ensure that it is running.

Token exchange is accomplished via a request for GET_ID which incidentally is the old method for identifying cards in the host.

Return Codes

ALPHI_S_DSP_ACK	Operation was successful.
Otherwise	Operation failed. See **HRESULTs**.

AlphiPciDevice::IsOpen

bool __stdcall AlphiPciDevice::IsOpen()

Ensure that this class is opened and representing a valid card.

Defined in: D:/ALPHIPCI/INCLUDE/ALPHIDLL.H

Return Value Returns true if the card is opened, false otherwise.

AlphiPciDevice::Map

HRESULT AlphiPciDevice::Map(
eTypeOfAccess *TypeOfAccess***,**
LinearAddress & *Address***)**

Retrieve a pointer to the physical card resource.

Defined in: D:/ALPHIPCI/ALPHIDLL/ALPHIDLL.CPP

Parameters

TypeOfAccess
Describes the type of access requested. See **eTypeOfAccess**.

Address
Linear address directly accessible by pointer dereference. See **LinearAddress**.

Return Codes

ALPHI_S_OK	Operation was successful.
Otherwise	Operation failed. See **HRESULTs**.

See Also **IOCTL_ALPHIPCI_MAP**

AlphiPciDevice::Open

HRESULT AlphiPciDevice::Open(
const char * *pszLinkName***,**
bool *bDoNotReset***)**

Open the specified device by name.

Defined in: D:/ALPHIPCI/ALPHIDLL/ALPHIDLL.CPP

Parameters

pszLinkName
Name of the device to be opened

bDoNotReset
If true, do not issue reset, assume all is well. Defaults to false .

Comments If a previous board was open, close it. Open the device, get pointer to the PCI registers, and reset the board if specified.

Resetting the board will force the Bootloader into a mode which accepts HOST commands. Use **StartX0** to boot the user code in FLASH, if desired.

Return Codes

ALPHI_S_OK	Operation was successful.
Otherwise	Operation failed. See **HRESULTs**.

See Also Device Names **Close StartX0**

AlphiPciDevice::Open

HRESULT AlphiPciDevice::Open(
  unsigned *BoardNumber***,**
  bool *bDoNotReset***)**

Open the specified device by device number.

Defined in: D:/ALPHIPCI/ALPHIDLL/ALPHIDLL.CPP

Parameters

BoardNumber
  Index of the card to open starting at zero.

bDoNotReset
  If true, do not issue reset, assume all is well. Defaults to false .

Comments

Devices are numbered consecutively starting from zero. To enumerate all devices in the system, consecutively open each device from zero until the first one fails. **GetDeviceCapabilities** will allow for querying the device capabilities and device name.

If a previous board was open, close it. Open the device, get pointer to the PCI registers, and reset the board if specified.

Resetting the board will force the Bootloader into a mode which accepts HOST commands. Use **StartX0** to boot the user code in FLASH, if desired.

Return Codes

ALPHI_S_OK	Operation was successful.
Otherwise	Operation failed. See **HRESULTs**.

See Also Device Names **Close StartX0 GetDeviceCapabilities**

AlphiPciDevice::ReadAmccNvram

HRESULT AlphiPciDevice::ReadAmccNvram(
  AmccNvramImage & *Image***)**

Read the NVRAM image stored in the device.

Defined in: D:/ALPHIPCI/ALPHIDLL/ALPHIDLL.CPP

Parameters

Image
  Reference to the NVRAM image to copy to.

Return Codes

ALPHI_S_OK	Operation was successful.
Otherwise	Operation failed. See **HRESULTs**.

See Also **AmccNvramImage**

AlphiPciDevice::ReadDword

HRESULT AlphiPciDevice::ReadDword(
 DWORD *DspAddress***,**
 DWORD * *pValue***)**

Ask the Bootloader to read from the specified DSP address.

Defined in: D:/ALPHIPCI/ALPHIDLL/ALPHIDLL.CPP

Parameters

DspAddress
 Address in the DSP to read from.

pValue
 Pointer to where to put the result.

Comments Using the READ_DWORD request, ask the Bootloader to read the DWORD.

Return Codes

ALPHI_S_OK	Operation was successful.
Otherwise	Operation failed. See **HRESULTs**.

AlphiPciDevice::ReadFifo

HRESULT AlphiPciDevice::ReadFifo(
 DWORD * *pBuffer***,**
 DWORD *NumDWords***,**
 ucr *UserFunction***,**
 PVOID *UserData***)**

Read from the FIFO port using Bus Master DMA.

Defined in: D:/ALPHIPCI/ALPHIDLL/ALPHIDLL.CPP

Parameters

pBuffer
 Pointer to the buffer to put read data.

NumDWords
 Number of DWORDS in the buffer.

UserFunction
 Pointer to user supplied completion function.

UserData
 User data provided to completion function.

Comments This function queues a request to the driver to read the contents of *pBuffer* from the DSP. On the AMCC, this is through the actual FIFO present on the AMCC. On the

PLX, this sets up the table of physical pointers to the pages and interrupts the DSP. The function returns immediately.

Upon completion, the user supplied function *UserFunction* is called with the address and size of the buffer, as well as the value *UserData*. The user completion function is called on a separate thread and at a higher priority, and will interrupt the main thread of execution, similiar to a hardware interupt.

Return Codes

ALPHI_S_OK	Operation was successful.
Otherwise	Operation failed. See **HRESULTs**.

See Also **(*ucr) WriteFifo CancelPendingReadRequests**

AlphiPciDevice::ReadFifoDirect

HRESULT AlphiPciDevice::ReadFifoDirect(
  DWORD * *pBuffer*,
  DWORD *NumDWords*)

Read directly from the FIFO port.

Defined in: D:/ALPHIPCI/ALPHIDLL/ALPHIDLL.CPP

Parameters

pBuffer
  Pointer to where to put the read values.

NumDWords
  Number of values to be read.

Comments Read the specified number of values from the FIFO port of the AMCC chip.

Return Codes

ALPHI_S_OK	Operation was successful.
Otherwise	Operation failed. See **HRESULTs**.

Developer Notes At present, there is no provision for timeouts in this function.

AlphiPciDevice::ReadFifoDirect

HRESULT AlphiPciDevice::ReadFifoDirect(
  WORD * *pBuffer*,
  DWORD *NumWords*)

Read directly from the FIFO port.

Defined in: D:/ALPHIPCI/ALPHIDLL/ALPHIDLL.CPP

Parameters

pBuffer
  Pointer to where to put the read values.

NumWords
  Number of values to be read.

Comments Read the specified number of values from the FIFO port of the AMCC chip.

Return Codes

ALPHI_S_OK	Operation was successful.
Otherwise	Operation failed. See **HRESULTs**.

Developer Notes At present, there is no provision for timeouts in this function.

AlphiPciDevice::ReadFifoDirect

HRESULT AlphiPciDevice::ReadFifoDirect(
BYTE * *pBuffer*,
DWORD *NumBytes***)**

Read directly from the FIFO port.

Defined in: D:/ALPHIPCI/ALPHIDLL/ALPHIDLL.CPP

Parameters

pBuffer
Pointer to where to put the read values.

NumBytes
Number of values to be read.

Comments Read the specified number of values from the FIFO port of the AMCC chip.

Return Codes

ALPHI_S_OK	Operation was successful.
Otherwise	Operation failed. See **HRESULTs**.

Developer Notes At present, there is no provision for timeouts in this function.

AlphiPciDevice::ReadMbox

HRESULT AlphiPciDevice::ReadMbox(
WORD *WhichMailbox*,
bool *fWait*,
DWORD * *pData***)**

Read a DWORD from the specified mailbox written by the DSP.

Defined in: D:/ALPHIPCI/ALPHIDLL/ALPHIDLL.CPP

Parameters

WhichMailbox
Mailbox to read from. Ranges from 0-3.

fWait
Should we wait for something to be sent by the DSP? Otherwise return the current value in the mailbox.

pData
Where to put the read DWORD, NULL if we don't want it.

Comments On the AMCC 5933, there are four mailboxes in each direction (to the DSP and from the DSP).

On the PLX9080, there are a total of 8 bidirectional mailboxes. This DLL and the DSP library work to simulate the 4 mailboxes in each direction, as well as the full/empty flags and interrupt support on full/empty status.

Return Codes

ALPHI_S_OK	Operation was successful.
Otherwise	Operation failed. See **HRESULTs**.

AlphiPciDevice::ReadPlxNvram

HRESULT AlphiPciDevice::ReadPlxNvram(
PlxNvramImage & ***Image*****)**

Read the NVRAM image stored in the device.

Defined in: D:/ALPHIPCI/ALPHIDLL/ALPHIDLL.CPP

Parameters

Image
Reference to the NVRAM image to copy to.

Return Codes

ALPHI_S_OK	Operation was successful.
Otherwise	Operation failed. See **HRESULTs**.

See Also **PlxNvramImage**

AlphiPciDevice::Reset

HRESULT AlphiPciDevice::Reset()

Reset the DSP and/or any IPs. If a DSP is present, ensure successful communication with DSP.

Defined in: D:/ALPHIPCI/ALPHIDLL/ALPHIDLL.CPP

Comments

Reset the card for 200 mS (as per IP specification), determine whether the DSP has responded, and by which mailbox, and set **AlphiPciDevice**::m_InterfaceType accordingly. Perform an **IsKernelRunning** token exchange to confirm successful reset.

By having a value in mailbox 0 as the DSP comes out of reset, the Bootloader knows to load into a HOST control mode.

Return Codes

ALPHI_S_OK	Operation was successful.
Otherwise	Operation failed. See **HRESULTs**.

Developer Notes

Bootloaders for PCI-4Pack prior to Version 1.4 used mailbox 0 exclusively for communication. Version 1.4 and later is common to all products, and use standard mailboxes. PLX devices utilize a similiar handshake to force loading the bootloader.

AlphiPciDevice::RetrieveSoftwareVersion

HRESULT AlphiPciDevice::RetrieveSoftwareVersion(
DWORD & *Version***)**

Retrieve version of the bootloader (or DSP application if supported).

Defined in: D:/ALPHIPCI/ALPHIDLL/ALPHIDLL.CPP

Parameters

Version
Reference to the returned version identifier.

Comments

Bootloaders prior to version 1.7 do not support this method of retrieving the version. For Versions 1.4 and later, the serial port will report the version after a RESET (in certain boot modes).

Return Codes

ALPHI_S_DSP_ACK	Operation was successful.
Otherwise	Operation failed. See **HRESULTs**.

AlphiPciDevice::SelectPage

HRESULT AlphiPciDevice::SelectPage(
DWORD *Page***)**

Select which IP memory space is mapped to the unified memory space.

Defined in: D:/ALPHIPCI/ALPHIDLL/ALPHIDLL.CPP

Parameters

Page
Select which IP space is mapped into unified space.

Comments

1-4: IP A-D 16 bit, 5: IP AB 32 bit, 6: IP CD 32 bit, 7: ABCD unified 16 bit, 8: ABCD unified 32 bit.

Return Codes

ALPHI_S_OK	Operation was successful.
Otherwise	Operation failed. See **HRESULTs**.

AlphiPciDevice::ShareHostMemory

HRESULT AlphiPciDevice::ShareHostMemory(
void * *pSharedMemory***,**
DWORD *Size***)**

Share the designated buffer with the DSP.

Defined in: D:/ALPHIPCI/ALPHIDLL/ALPHIDLL.CPP

Parameters

pSharedMemory
Pointer to beginning of shared memory.

Size
Size of buffer to share in bytes.

Comments

Allocate and free the buffer using **AllocSharableMemory** and **FreeSharableMemory**. This way, the buffer is guaranteed to start at an address that is divisible by the page size of the processor, and therefore, greatly simplifying the DSP's addressing.

Cancel the sharing by calling **UnshareHostMemory**.

Return Codes

ALPHI_S_OK	Operation was successful.
Otherwise	Operation failed. See **HRESULTs**.

See Also

AllocSharableMemory FreeSharableMemory UnshareHostMemory

AlphiPciDevice::Start

HRESULT AlphiPciDevice::Start()

Begin execution of the downloaded DSP code.

Defined in: D:/ALPHIPCI/ALPHIDLL/ALPHIDLL.CPP

Comments

Ensure that there is a program starting address (from a successful **Download**), and ask the DSP to begin execution.

Return Codes

ALPHI_S_OK	Operation was successful.
Otherwise	Operation failed. See **HRESULTs**.

AlphiPciDevice::StartX0

HRESULT AlphiPciDevice::StartX0()

Begin execution of the previously downloaded Tektronix file.

Defined in: D:/ALPHIPCI/ALPHIDLL/ALPHIDLL.CPP

Return Codes

ALPHI_S_OK	Operation was successful.
Otherwise	Operation failed. See **HRESULTs**.

See Also

DownloadX0

AlphiPciDevice::UnhookMailboxInterrupt

HRESULT AlphiPciDevice::UnhookMailboxInterrupt()

Remove a hook installed with **HookMailboxInterrupt**.

Defined in: D:/ALPHIPCI/ALPHIDLL/ALPHIDLL.CPP

Comments Cancels any outstanding interrupt requests, and waits for the interrupt thread to exit.

Return Codes

ALPHI_S_OK	Operation was successful.
Otherwise	Operation failed. See **HRESULTs**.

See Also **HookMailboxInterrupt**

AlphiPciDevice::Unmap

HRESULT AlphiPciDevice::Unmap(
LinearAddress & *Address***)**

Unmap pointer to physical resource.

Defined in: D:/ALPHIPCI/ALPHIDLL/ALPHIDLL.CPP

Parameters

Address
Linear address directly accessible by pointer dereference. See **LinearAddress**.

Return Codes

ALPHI_S_OK	Operation was successful.
Otherwise	Operation failed. See **HRESULTs**.

See Also **IOCTL_ALPHIPCI_UNMAP**

AlphiPciDevice::UnshareHostMemory

HRESULT AlphiPciDevice::UnshareHostMemory()

Disable sharing the buffer with the DSP.

Defined in: D:/ALPHIPCI/ALPHIDLL/ALPHIDLL.CPP

Return Codes

ALPHI_S_OK	Operation was successful.
Otherwise	Operation failed. See **HRESULTs**.

See Also **ShareHostMemory**

AlphiPciDevice::WriteAmccNvram

HRESULT AlphiPciDevice::WriteAmccNvram(
AmccNvramImage & *Image***)**

Write the NVRAM image to the device.

Defined in: D:/ALPHIPCI/ALPHIDLL/ALPHIDLL.CPP

Parameters

Image
Reference to the NVRAM image to copy from.

Return Codes

ALPHI_S_OK	Operation was successful.
Otherwise	Operation failed. See **HRESULTs**.

See Also

AmccNvramImage

AlphiPciDevice::WriteDword

HRESULT AlphiPciDevice::WriteDword(
DWORD *DspAddress***,**
DWORD *Value***)**

Ask the Bootloader to write to the specified DSP address.

Defined in: D:/ALPHIPCI/ALPHIDLL/ALPHIDLL.CPP

Parameters

DspAddress
Address in the DSP to write to.

Value
Value to be written.

Comments

Using the WRITE_DWORD request, ask the Bootloader to write the DWORD.

Return Codes

ALPHI_S_OK	Operation was successful.
Otherwise	Operation failed. See **HRESULTs**.

AlphiPciDevice::WriteFifo

HRESULT AlphiPciDevice::WriteFifo(
DWORD * *pBuffer***,**
DWORD *NumDWords***,**
ucr *UserFunction***,**
PVOID *UserData***)**

Write to the FIFO port using Bus Master DMA.

Defined in: D:/ALPHIPCI/ALPHIDLL/ALPHIDLL.CPP

Parameters

pBuffer
Pointer to the buffer to be written.

NumDWords
Number of DWORDS in the buffer.

UserFunction
Pointer to user supplied completion function.

UserData
User data provided to completion function.

Comments

This function queues a request to the driver to copy the contents of *pBuffer* to the DSP. On the AMCC, this is through the actual FIFO present on the AMCC. On the PLX, this sets up the table of physical pointers to the pages and interrupts the DSP. The function returns immediately.

Upon completion, the user supplied function *UserFunction* is called with the address and size of the buffer, as well as the value *UserData*. The user completion function is called on a separate thread and at a higher priority, and will interrupt the main thread of execution, similiar to a hardware interupt.

Return Codes

ALPHI_S_OK	Operation was successful.
Otherwise	Operation failed. See **HRESULTs**.

See Also

(*ucr) ReadFifo CancelPendingWriteRequests

AlphiPciDevice::WriteFifoDirect

HRESULT AlphiPciDevice::WriteFifoDirect(
    DWORD * *pBuffer***,**
    DWORD *NumDWords***)**

Write directly to the FIFO port.

Defined in: D:/ALPHIPCI/ALPHIDLL/ALPHIDLL.CPP

Parameters

pBuffer
    Pointer to where to find the values to write.

NumDWords
    Number of values to be written.

Comments

Write the specified number of values to the FIFO port of the AMCC chip.

Return Codes

ALPHI_S_OK	Operation was successful.
Otherwise	Operation failed. See **HRESULTs**.

Developer Notes

At present, there is no provision for timeouts in this function.

AlphiPciDevice::WriteFifoDirect

HRESULT AlphiPciDevice::WriteFifoDirect(
    WORD * *pBuffer***,**
    DWORD *NumWords***)**

Write directly to the FIFO port.

Defined in: D:/ALPHIPCI/ALPHIDLL/ALPHIDLL.CPP

Parameters

pBuffer
    Pointer to where to find the values to write.

NumWords
Number of values to be written.

Comments Write the specified number of values to the FIFO port of the AMCC chip.

Return Codes

ALPHI_S_OK	Operation was successful.
Otherwise	Operation failed. See **HRESULTs**.

Developer Notes At present, there is no provision for timeouts in this function.

AlphiPciDevice::WriteFifoDirect

HRESULT AlphiPciDevice::WriteFifoDirect(
BYTE * *pBuffer***,**
DWORD *NumBytes***)**

Write directly to the FIFO port.

Defined in: D:/ALPHIPCI/ALPHIDLL/ALPHIDLL.CPP

Parameters

pBuffer
Pointer to where to find the values to write.

NumBytes
Number of values to be written.

Comments Write the specified number of values to the FIFO port of the AMCC chip.

Return Codes

ALPHI_S_OK	Operation was successful.
Otherwise	Operation failed. See **HRESULTs**.

Developer Notes At present, there is no provision for timeouts in this function.

AlphiPciDevice::WriteMbox

HRESULT AlphiPciDevice::WriteMbox(
WORD *WhichMailbox***,**
DWORD *dwData***)**

Write a DWORD to the specified mailbox to be read by the DSP.

Defined in: D:/ALPHIPCI/ALPHIDLL/ALPHIDLL.CPP

Parameters

WhichMailbox
Mailbox to write to. Ranges from 0-3.

dwData
Data to write.

Comments The function will wait until the mailbox has read the previous value before writing the new value. If the DSP does not read the old value, this function times out.

On the AMCC 5933, there are four mailboxes in each direction (to the DSP and from the DSP).

On the PLX9080, there are a total of 8 bidirectional mailboxes. This DLL and the DSP library work to simulate the 4 mailboxes in each direction, as well as the full/empty flags and interrupt support on full/empty status.

Return Codes

ALPHI_S_OK	Operation was successful.
Otherwise	Operation failed. See **HRESULTs**.

AlphiPciDevice::WriteMbox

HRESULT AlphiPciDevice::WriteMbox(
WORD *WhichMailbox***,**
DWORD *dwData***,**
bool *fWait***)**

Write a DWORD to the specified mailbox to be read by the DSP.

Defined in: D:/ALPHIPCI/ALPHIDLL/ALPHIDLL.CPP

Parameters

WhichMailbox
Mailbox to write to. Ranges from 0-3.

dwData
Data to write.

fWait
Wait for the mailbox to be empty by the DSP before writing a new value.

Comments

The function will wait until the mailbox has read the previous value before writing the new value. If the DSP does not read the old value, this function times out.

On the AMCC 5933, there are four mailboxes in each direction (to the DSP and from the DSP).

On the PLX9080, there are a total of 8 bidirectional mailboxes. This DLL and the DSP library work to simulate the 4 mailboxes in each direction, as well as the full/empty flags and interrupt support on full/empty status.

Return Codes

ALPHI_S_OK	Operation was successful.
Otherwise	Operation failed. See **HRESULTs**.

AlphiPciDevice::WritePlxNvram

HRESULT AlphiPciDevice::WritePlxNvram(
PlxNvramImage & *Image***)**

Write the NVRAM image to the device.

Defined in: D:/ALPHIPCI/ALPHIDLL/ALPHIDLL.CPP

Parameters

Image
Reference to the NVRAM image to copy to.

Return Codes

ALPHI_S_OK	Operation was successful.
Otherwise	Operation failed. See **HRESULTs**.

See Also **PlxNvramImage**

AlphiPciDevice::~AlphiPciDevice

AlphiPciDevice::~AlphiPciDevice()

Destructor.

Defined in: D:/ALPHIPCI/ALPHIDLL/ALPHIDLL.CPP

Comments Closes the device, if not done previously.

See Also **Close**

C Functions and Callbacks

C functions implementing the functionality of this package. Many of these functions are exported for use by applications written in C or other languages.

(*ucr)

typedef void (*ucr)(
    DWORD * *pBuffer***,**
    DWORD *NumDwords***,**
    void * *UserData***)**

User supplied Completion Routine.

Defined in: D:/ALPHIPCI/INCLUDE/ALPHIDLL.H

Parameters

pBuffer
    Pointer to the buffer which has completed.

NumDwords
    Size of the buffer which has completed.

UserData
    User's data supplied to the queueing function.

Developer Notes

Make sure that this function is declared stdcall, or the program will crash.

This function is called on a separate thread which is running at THREAD_PRIORITY_TIME_CRITICAL. This is done to simulate the reception of an interrupt, where this function is the interrupt handler. It is strongly urged to return from this function in a timely manner, as it will run to completion as the highest priority thread. If significant processing is to be performed, it is better to make use of NT scheduling objects such as semaphores, or to post a message to a window.

(*UsersIntCompletionRoutine)

typedef void (*UsersIntCompletionRoutine)(
    void * *UserData***,**
    DWORD *Cause***)**

User supplied Interrupt Completion Routine.

Defined in: D:/ALPHIPCI/INCLUDE/ALPHIDLL.H

Parameters

UserData
    User's data supplied to the queueing function.

Cause
    Device dependent cause for calling this function.

Comments

For AMCC based devices, *Cause* is the value that was in **AMCC**.**INTCSR** at the time of the interrupt. For PLX based devices, *Cause* is the value that was in **PLX**.**intcsr** masked by the bits of interest, at the time of the interrupt.

Developer Notes

Make sure that this function is declared stdcall, or the program will crash.

This function is called on a separate thread which is running at THREAD_PRIORITY_TIME_CRITICAL. This is done to simulate the reception of an interrupt, where this function is the interrupt handler. It is strongly urged to return from this function in a timely manner, as it will run to completion as the highest priority thread. If significant processing is to be performed, it is better to make use of NT scheduling objects such as semaphores, or to post a message to a window.

AllocSharableMemory

Dll HRESULT __stdcall AllocSharableMemory(
void ** *pSharedMemory***,**
DWORD *Size***)**

Allocate memory intended to be shared between board and HOST.

Defined in: D:/ALPHIPCI/ALPHIDLL/ALPHIDLL.CPP

Parameters

pSharedMemory
Address of returned pointer.

Size
Desired size in bytes.

Comments

Calls the system function **VirtualAlloc** to allocate memory aligned to a 4K page. By aligning the buffer to a 4K page, the DSP does not have to constantly add offsets when calculating the correct physical addresses of this memory.

Return Codes

ALPHI_S_OK	Operation was successful.
Otherwise	Operation failed. See **HRESULTs**.

See Also

FreeSharableMemory

BurnBootFlashX0

Dll HRESULT __stdcall BurnBootFlashX0(
AlphiPciDevice * *pThis***)**

Burn the boot area of the FLASH with the previously downloaded Tektronix file.

Defined in: D:/ALPHIPCI/ALPHIDLL/ALPHIDLL.CPP

Parameters

pThis
Pointer to the card object.

Comments

Calls **AlphiPciDevice::BurnBootFlashX0** and returns the result.

Return Codes

ALPHI_S_OK	Operation was successful.
Otherwise	Operation failed. See **HRESULTs**.

BurnUserFlashX0

Dll HRESULT __stdcall BurnUserFlashX0(
AlphiPciDevice * *pThis***)**

Burn the user area of the FLASH with the previously downloaded Tektronix file.

Defined in: D:/ALPHIPCI/ALPHIDLL/ALPHIDLL.CPP

Parameters

pThis
Pointer to the card object.

Comments

Calls **AlphiPciDevice::BurnUserFlashX0** and returns the result.

Return Codes

ALPHI_S_OK	Operation was successful.
Otherwise	Operation failed. See **HRESULTs**.

CancelPendingReadRequests

Dll HRESULT __stdcall CancelPendingReadRequests(
AlphiPciDevice * *pThis***)**

Cancel any pending read requests.

Defined in: D:/ALPHIPCI/ALPHIDLL/ALPHIDLL.CPP

Parameters

pThis
Pointer to the card object.

Comments

Calls **AlphiPciDevice::CancelPendingReadRequests** and returns the result.

Cancels any outstanding read packets and cleans up.

Return Codes

ALPHI_S_OK	Operation was successful.
Otherwise	Operation failed. See **HRESULTs**.

See Also

ReadFifo

CancelPendingWriteRequests

Dll HRESULT __stdcall CancelPendingWriteRequests(
AlphiPciDevice * *pThis***)**

Cancel any pending write requests.

Defined in: D:/ALPHIPCI/ALPHIDLL/ALPHIDLL.CPP

Parameters

pThis
Pointer to the card object.

Comments

Calls **AlphiPciDevice::CancelPendingWriteRequests** and returns the result.

Cancels any outstanding read packets and cleans up. Resets the PCI to DSP FIFO on AMCC.

Return Codes

ALPHI_S_OK	Operation was successful.
Otherwise	Operation failed. See **HRESULTs**.

See Also

WriteFifo

Close

Dll HRESULT __stdcall Close(
AlphiPciDevice * *pThis***)**

Free resources and close the device.

Defined in: D:/ALPHIPCI/ALPHIDLL/ALPHIDLL.CPP

Parameters

pThis
Pointer to the card object.

Comments

Calls **AlphiPciDevice::Close**. Then deletes *pThis*.

Return Codes

ALPHI_S_OK	Operation was successful.
Otherwise	Operation failed. See **HRESULTs**.

DisplayErrorMessageBox

HRESULT __stdcall DisplayErrorMessageBox(
char * *szApplicationName***,**
HRESULT *Error***)**

Display a modal dialog box with the text of any ERROR or higher message associated with the HRESULT.

Defined in: D:/ALPHIPCI/HOST EXAMPLES/ALPHIERRORCODE/DISPLAYERRORCODE.CPP

Return Value

The value of *Error*.

Parameters

szApplicationName
Text placed in the title bar of the dialog box.

Error
Returned HRESULT from a DLL call.

Developer Notes

This function is in AlphiErrorCode.dll, which means that AlphiErrorCode.lib needs to be included in your link.

DisplayErrorToConsole

HRESULT __stdcall DisplayErrorToConsole(
HRESULT *Error***)**

Display the text of any ERROR or higher message associated with the HRESULT to the console.

Defined in: D:/ALPHIPCI/HOST EXAMPLES/ALPHIERRORCODE/DISPLAYERRORCODE.CPP

Return Value

The value of *Error*.

Parameters

Error
Returned HRESULT from a DLL call.

Developer Notes

This function is in AlphiErrorCode.dll, which means that AlphiErrorCode.lib needs to be included in your link.

DisplayInfoMessageBox

HRESULT __stdcall DisplayInfoMessageBox(
char * *szApplicationName***,**
HRESULT *Error***)**

Display a modal dialog box with the text of any INFORMATIONAL or higher message associated with the HRESULT.

Defined in: D:/ALPHIPCI/HOST EXAMPLES/ALPHIERRORCODE/DISPLAYERRORCODE.CPP

Return Value

The value of *Error*.

Parameters

szApplicationName
Text placed in the title bar of the dialog box.

Error
Returned HRESULT from a DLL call.

Developer Notes

This function is in AlphiErrorCode.dll, which means that AlphiErrorCode.lib needs to be included in your link.

DisplayInfoToConsole

HRESULT __stdcall DisplayInfoToConsole(
HRESULT *Error***)**

Display the text of any INFORMATIONAL or higher message associated with the HRESULT to the console.

Defined in: D:/ALPHIPCI/HOST EXAMPLES/ALPHIERRORCODE/DISPLAYERRORCODE.CPP

Return Value The value of *Error*.

Parameters

Error
Returned HRESULT from a DLL call.

Developer Notes This function is in AlphiErrorCode.dll, which means that AlphiErrorCode.lib needs to be included in your link.

DisplayResultMessageBox

HRESULT __stdcall DisplayResultMessageBox(
char * *szApplicationName***,**
HRESULT *Error***)**

Display a modal dialog box with the text of any message associated with the HRESULT.

Defined in: D:/ALPHIPCI/HOST EXAMPLES/ALPHIERRORCODE/DISPLAYERRORCODE.CPP

Return Value The value of *Error*.

Parameters

szApplicationName
Text placed in the title bar of the dialog box.

Error
Returned HRESULT from a DLL call.

Developer Notes This function is in AlphiErrorCode.dll, which means that AlphiErrorCode.lib needs to be included in your link.

DisplayResultToConsole

HRESULT __stdcall DisplayResultToConsole(
HRESULT *Error***)**

Display the text of any message associated with the HRESULT to the console.

Defined in: D:/ALPHIPCI/HOST EXAMPLES/ALPHIERRORCODE/DISPLAYERRORCODE.CPP

Return Value The value of *Error*.

Parameters

Error
Returned HRESULT from a DLL call.

Developer Notes This function is in AlphiErrorCode.dll, which means that AlphiErrorCode.lib needs to be included in your link.

DisplayWarningMessageBox

HRESULT __stdcall DisplayWarningMessageBox(
char * *szApplicationName***,**
HRESULT *Error***)**

Display a modal dialog box with the text of any WARNING or higher message associated with the HRESULT.

Defined in: D:/ALPHIPCI/HOST EXAMPLES/ALPHIERRORCODE/DISPLAYERRORCODE.CPP

Return Value

The value of *Error*.

Parameters

szApplicationName
Text placed in the title bar of the dialog box.

Error
Returned HRESULT from a DLL call.

Developer Notes

This function is in AlphiErrorCode.dll, which means that AlphiErrorCode.lib needs to be included in your link.

DisplayWarningToConsole

HRESULT __stdcall DisplayWarningToConsole(
HRESULT *Error***)**

Display the text of any WARNING or higher message associated with the HRESULT to the console.

Defined in: D:/ALPHIPCI/HOST EXAMPLES/ALPHIERRORCODE/DISPLAYERRORCODE.CPP

Return Value

The value of *Error*.

Parameters

Error
Returned HRESULT from a DLL call.

Developer Notes

This function is in AlphiErrorCode.dll, which means that AlphiErrorCode.lib needs to be included in your link.

Download

Dll HRESULT __stdcall Download(
AlphiPciDevice * *pThis***,**
const char * *szFilename***)**

Download the COFF file to the DSP.

Defined in: D:/ALPHIPCI/ALPHIDLL/ALPHIDLL.CPP

Parameters

pThis
Pointer to the card object.

szFilename
NULL terminated filename.

Comments

Calls **AlphiPciDevice::Download** and returns the result.

Return Codes

ALPHI_S_OK	Operation was successful.
Otherwise	Operation failed. See **HRESULTs**.

DownloadX0

Dll HRESULT __stdcall DownloadX0(
AlphiPciDevice * *pThis*,
const char * *szFilename***)**

Download the Tektronix X0 file to the buffer on the card.

Defined in: D:/ALPHIPCI/ALPHIDLL/ALPHIDLL.CPP

Parameters

pThis
Pointer to the card object.

szFilename
NULL terminated filename.

Comments

Calls **AlphiPciDevice::DownloadX0** and returns the result.

Return Codes

ALPHI_S_OK	Operation was successful.
Otherwise	Operation failed. See **HRESULTs**.

FreeSharableMemory

Dll HRESULT __stdcall FreeSharableMemory(
void * *pSharedMemory***)**

Free memory previously allocated by **AllocSharableMemory**.

Defined in: D:/ALPHIPCI/ALPHIDLL/ALPHIDLL.CPP

Parameters

pSharedMemory
Pointer to the buffer.

Comments

Calls the system function **VirtualFree** to free memory aligned to a 4K page. By aligning the buffer to a 4K page, the DSP does not have to constantly add offsets when calculating the correct physical addresses of this memory.

Return Codes

ALPHI_S_OK	Operation was successful.
Otherwise	Operation failed. See **HRESULTs**.

See Also

AllocSharableMemory

GetDeviceCapabilities

Dll const DeviceCaps * __stdcall GetDeviceCapabilities(
AlphiPciDevice * *pThis***)**

Get a pointer to the Device Capabilities.

Defined in: D:/ALPHIPCI/ALPHIDLL/ALPHIDLL.CPP

Return Value Constant pointer to the **DeviceCaps**.

Parameters *pThis*
Pointer to the card object.

Comments Calls **AlphiPciDevice::GetDeviceCapabilities** and returns the result.

See Also **DeviceCaps**

GetResultText

HRESULT __stdcall GetResultText(
HRESULT *Error***,**
char * *buffer***,**
size_t *size***)**

Retrieve the text of any message associated with the HRESULT.

Defined in: D:/ALPHIPCI/HOST EXAMPLES/ALPHIERRORCODE/DISPLAYERRORCODE.CPP

Return Value S_OK if successful, E_FAIL if it was not found (but the text "Could not find error message." was be copied to the buffer).

Parameters *Error*
Returned HRESULT from a DLL call.

buffer
Pointer to user buffer to receive the message.

size
Size of the user buffer.

Developer Notes This function is in AlphiErrorCode.dll, which means that AlphiErrorCode.lib needs to be included in your link.

HookMailboxInterrupt

Dll HRESULT __stdcall HookMailboxInterrupt(
AlphiPciDevice * *pThis***,**
eIntType *IntType***,**

UsersIntCompletionRoutine *UsersInterruptRoutine*,
void * *UsersInterruptData*)

Install a hook to call *UsersInterruptRoutine* as a result of a valid interrupt cause.

Defined in: D:/ALPHIPCI/ALPHIDLL/ALPHIDLL.CPP

Parameters

pThis
Pointer to the card object.

IntType
Which cause(s) to look at. See **eIntType** for AMCC class devices. PLX device causes have not been defined in the header file as yet.

UsersInterruptRoutine
Pointer to the user's completion routine. See **(*UsersIntCompletionRoutine)**.

UsersInterruptData
User data passed to the completion function.

Comments

Calls **AlphiPciDevice::HookMailboxInterrupt** and returns the result.

On the AMCC, an interrupt can be generated from up to two causes at a time. One interrupt cause is the DSP writing to a particular mailbox, or on certain cards, a hardware event could generate an interrupt. The other cause is generated by the DSP reading from a particular mailbox.

On the PLX, an interrupt can be generated on any of the following: DSP read of a mailbox, DSP write to a mailbox, a hardware event, or a doorbell write by the DSP.

This function operates by creating a separate thread running at the highest priority. This thread blocks in the device driver until the interrupt event has occurred. When the thread is freed, it calls *UsersInterruptRoutine* with the parameter *UsersInterruptData* and the state of the interrupts at the time of calling. When the user function completes, the thread then blocks again in the device driver. It is impossible not to be called from an interrupt cause, but it is possible to have several interrupt events combined into one call to the user function, if they occur fast enough.

Developer Notes

This function really deserves a better name.

Return Codes

ALPHI_S_OK	Operation was successful.
Otherwise	Operation failed. See **HRESULTs**.

See Also

UnhookMailboxInterrupt (*UsersIntCompletionRoutine)

IsKernelRunning

Dll HRESULT __stdcall IsKernelRunning(
AlphiPciDevice * *pThis*)

Ensure that the DSP can respond to requests from this API.

Defined in: D:/ALPHIPCI/ALPHIDLL/ALPHIDLL.CPP

Parameters

pThis
Pointer to the card object.

Comments Calls **AlphiPciDevice::IsKernelRunning** and returns the result.

Return Codes

ALPHI_S_OK	Operation was successful.
Otherwise	Operation failed. See **HRESULTs**.

Map

Dll HRESULT __stdcall Map(
AlphiPciDevice * *pThis***,**
eTypeOfAccess *TypeOfAccess***,**
LinearAddress * *pAddress***)**

Retrieve a pointer to the physical card resource.

Defined in: D:/ALPHIPCI/ALPHIDLL/ALPHIDLL.CPP

Parameters

pThis
Pointer to the card object.

TypeOfAccess
Describes the type of access requested. See **eTypeOfAccess**.

pAddress
Linear address directly accessible by pointer dereference. See **LinearAddress**.

Comments Calls **AlphiPciDevice::Map** and returns the result.

Return Codes

ALPHI_S_OK	Operation was successful.
Otherwise	Operation failed. See **HRESULTs**.

Open

Dll HRESULT __stdcall Open(
AlphiPciDevice ** *pThis***,**
const char * *pszLinkName***)**

Allocate resources for a new board, and attempt to open it.

Defined in: D:/ALPHIPCI/ALPHIDLL/ALPHIDLL.CPP

Parameters

pThis
Returned handle to the open board if successful.

pszLinkName
Name of the device to open.

Comments Allocates a new **AlphiPciDevice** and calls **AlphiPciDevice::Open**. If the open fails, deletes the **AlphiPciDevice**. Returns the result of the Open.

Return Codes

ALPHI_S_OK	Operation was successful.
Otherwise	Operation failed. See **HRESULTs**.

See Also Device Names

OpenByNumber

Dll HRESULT __stdcall OpenByNumber(
AlphiPciDevice ** *pThis***,**
unsigned *DeviceNumber***)**

Allocate resources for a new board, and attempt to open it.

Defined in: D:/ALPHIPCI/ALPHIDLL/ALPHIDLL.CPP

Parameters

pThis
Returned handle to the open board if successful.

DeviceNumber
Name of the device to open.

Comments Allocates a new **AlphiPciDevice** and calls **AlphiPciDevice::Open**. If the open fails, deletes the **AlphiPciDevice**. Returns the result of the Open.

Return Codes

ALPHI_S_OK	Operation was successful.
Otherwise	Operation failed. See **HRESULTs**.

See Also Device Names

OpenByNumberWithoutReset

Dll HRESULT __stdcall OpenByNumberWithoutReset(
AlphiPciDevice ** *pThis***,**
unsigned *DeviceNumber***)**

Allocate resources for a new board, and attempt to open it.

Defined in: D:/ALPHIPCI/ALPHIDLL/ALPHIDLL.CPP

Parameters

pThis
Returned handle to the open board if successful.

DeviceNumber
Name of the device to open.

Comments Allocates a new **AlphiPciDevice** and calls **AlphiPciDevice::Open**. If the open fails, deletes the **AlphiPciDevice**. Returns the result of the Open with the bDoNotReset set to true.

Return Codes

ALPHI_S_OK	Operation was successful.
Otherwise	Operation failed. See **HRESULTs**.

See Also Device Names

OpenWithoutReset

Dll HRESULT __stdcall OpenWithoutReset(
AlphiPciDevice ** *pThis***,**
const char * *pszLinkName***)**

Allocate resources for a new board, and attempt to open it.

Defined in: D:/ALPHIPCI/ALPHIDLL/ALPHIDLL.CPP

Parameters

pThis
Returned handle to the open board if successful.

pszLinkName
Name of the device to open.

Comments

Allocates a new **AlphiPciDevice** and calls **AlphiPciDevice::Open**. If the open fails, deletes the **AlphiPciDevice**. Returns the result of the Open with the bDoNotReset set to true.

Return Codes

ALPHI_S_OK	Operation was successful.
Otherwise	Operation failed. See **HRESULTs**.

See Also

Device Names

ReadAmccNvram

Dll HRESULT __stdcall ReadAmccNvram(
AlphiPciDevice * *pThis***,**
AmccNvramImage * *pImage***)**

Read the NVRAM image stored in the device.

Defined in: D:/ALPHIPCI/ALPHIDLL/ALPHIDLL.CPP

Parameters

pThis
Pointer to the card object.

pImage
Pointer to the NVRAM image to copy to.

Comments

Calls **AlphiPciDevice::ReadAmccNvram** and returns the result.

Return Codes

ALPHI_S_OK	Operation was successful.
Otherwise	Operation failed. See **HRESULTs**.

See Also

AmccNvramImage

ReadDword

Dll HRESULT __stdcall ReadDword(
AlphiPciDevice * *pThis***,**
DWORD *DspAddress***,**
DWORD * *pValue***)**

Ask the Bootloader to read from the specified DSP address.

Defined in: D:/ALPHIPCI/ALPHIDLL/ALPHIDLL.CPP

Parameters

pThis
Pointer to the card object.

DspAddress
Address in the DSP to read from.

pValue
Pointer to where to put the result.

Comments

Calls **AlphiPciDevice::ReadDword** and returns the result.

Return Codes

ALPHI_S_OK	Operation was successful.
Otherwise	Operation failed. See **HRESULTs**.

ReadFifo

Dll HRESULT __stdcall ReadFifo(
DWORD * *pBuffer***,**
DWORD *NumDWords***,**
ucr *UserFunction***,**
PVOID *UserData***)**

Read from the FIFO port using Bus Master DMA.

Defined in: D:/ALPHIPCI/ALPHIDLL/ALPHIDLL.CPP

Parameters

pBuffer
Pointer to the buffer to be written.

NumDWords
Number of DWORDS in the buffer.

UserFunction
Pointer to user supplied completion function.

UserData
User data provided to completion function.

Comments

Calls **AlphiPciDevice::ReadFifo** and returns the result.

This function queues a request to the driver to read the contents of *pBuffer* from the DSP. On the AMCC, this is through the actual FIFO present on the AMCC. On the PLX, this sets up the table of physical pointers to the pages and interrupts the DSP. The function returns immediately.

Upon completion, the user supplied function *UserFunction* is called with the address and size of the buffer, as well as the value *UserData*. The user completion function is called on a separate thread and at a higher priority, and will interrupt the main thread of execution, similiar to a hardware interupt.

Return Codes

ALPHI_S_OK	Operation was successful.
Otherwise	Operation failed. See **HRESULTs**.

See Also **(*ucr) WriteFifo CancelPendingReadRequests**

ReadFifoDirect

Dll HRESULT __stdcall ReadFifoDirect(
 AlphiPciDevice * *pThis***,**
 DWORD * *pBuffer***,**
 DWORD *NumDWords***)**

Read directly from the FIFO port.

Defined in: D:/ALPHIPCI/ALPHIDLL/ALPHIDLL.CPP

Parameters

pThis
 Pointer to the card object.

pBuffer
 Pointer to where to put the read values.

NumDWords
 Number of values to be read.

Comments Calls **AlphiPciDevice::ReadFifoDirect** and returns the result.

Return Codes

ALPHI_S_OK	Operation was successful.
Otherwise	Operation failed. See **HRESULTs**.

ReadFifoDirectByte

Dll HRESULT __stdcall ReadFifoDirectByte(
 AlphiPciDevice * *pThis***,**
 BYTE * *pBuffer***,**
 DWORD *NumBytes***)**

Read directly from the FIFO port.

Defined in: D:/ALPHIPCI/ALPHIDLL/ALPHIDLL.CPP

Parameters

pThis
 Pointer to the card object.

pBuffer
 Pointer to where to put the read values.

NumBytes
Number of values to be read.

Comments Calls **AlphiPciDevice::ReadFifoDirect** and returns the result.

Return Codes

ALPHI_S_OK	Operation was successful.
Otherwise	Operation failed. See **HRESULTs**.

ReadFifoDirectWord

Dll HRESULT __stdcall ReadFifoDirectWord(
AlphiPciDevice * *pThis***,**
WORD * *pBuffer***,**
DWORD *NumWords***)**

Read directly from the FIFO port.

Defined in: D:/ALPHIPCI/ALPHIDLL/ALPHIDLL.CPP

Parameters

pThis
Pointer to the card object.

pBuffer
Pointer to where to put the read values.

NumWords
Number of values to be read.

Comments Calls **AlphiPciDevice::ReadFifoDirect** and returns the result.

Return Codes

ALPHI_S_OK	Operation was successful.
Otherwise	Operation failed. See **HRESULTs**.

ReadMbox

Dll HRESULT __stdcall ReadMbox(
AlphiPciDevice * *pThis***,**
WORD *WhichMailbox***,**
BOOL *fWait***,**
DWORD * *pData***)**

Read a DWORD from the specified PCI mailbox.

Defined in: D:/ALPHIPCI/ALPHIDLL/ALPHIDLL.CPP

Parameters

pThis
Pointer to the card object.

WhichMailbox
Mailbox to write to. Ranges from 0-3.

fWait
Should we wait for something to be sent by the DSP? Otherwise return the last thing sent.

pData
Where to put the read DWORD, NULL if we don't want it.

Comments Calls **AlphiPciDevice::ReadFifoDirect** and returns the result.

Return Codes

ALPHI_S_OK	Operation was successful.
Otherwise	Operation failed. See **HRESULTs**.

ReadMboxByte

Dll HRESULT __stdcall ReadMboxByte(
AlphiPciDevice * *pThis*,
WORD *WhichMailbox*,
BOOL *fWait*,
BYTE * *pData***)**

Read a BYTE from the specified PCI mailbox.

Defined in: D:/ALPHIPCI/ALPHIDLL/ALPHIDLL.CPP

Parameters

pThis
Pointer to the card object.

WhichMailbox
Mailbox to write to. Ranges from 0-3.

fWait
Should we wait for something to be sent by the DSP? Otherwise return the last thing sent.

pData
Where to put the read WORD, NULL if we don't want it.

Comments Calls **AlphiPciDevice::ReadFifoDirect** and returns the result.

Return Codes

ALPHI_S_OK	Operation was successful.
Otherwise	Operation failed. See **HRESULTs**.

ReadMboxWord

Dll HRESULT __stdcall ReadMboxWord(
AlphiPciDevice * *pThis*,
WORD *WhichMailbox*,
BOOL *fWait*,
WORD * *pData***)**

Read a WORD from the specified PCI mailbox.

Defined in: D:/ALPHIPCI/ALPHIDLL/ALPHIDLL.CPP

Parameters

pThis
Pointer to the card object.

WhichMailbox
Mailbox to write to. Ranges from 0-3.

fWait
Should we wait for something to be sent by the DSP? Otherwise return the last thing sent.

pData
Where to put the read WORD, NULL if we don't want it.

Comments

Calls **AlphiPciDevice::ReadFifoDirect** and returns the result.

Return Codes

ALPHI_S_OK	Operation was successful.
Otherwise	Operation failed. See **HRESULTs**.

ReadPlxNvram

Dll HRESULT __stdcall ReadPlxNvram(
AlphiPciDevice * ***pThis*****,**
PlxNvramImage * ***pImage*****)**

Read the NVRAM image stored in the device.

Defined in: D:/ALPHIPCI/ALPHIDLL/ALPHIDLL.CPP

Parameters

pThis
Pointer to the card object.

pImage
Pointer to the NVRAM image to copy to.

Comments

Calls **AlphiPciDevice::ReadPlxNvram** and returns the result.

Return Codes

ALPHI_S_OK	Operation was successful.
Otherwise	Operation failed. See **HRESULTs**.

See Also

PlxNvramImage

Reset

Dll HRESULT __stdcall Reset(
AlphiPciDevice * ***pThis*****)**

Reset the DSP and any IPs.

Defined in: D:/ALPHIPCI/ALPHIDLL/ALPHIDLL.CPP

Parameters

pThis
Pointer to the card object.

Comments

Calls **AlphiPciDevice::Reset** and returns the result.

Return Codes

ALPHI_S_OK	Operation was successful.
Otherwise	Operation failed. See **HRESULTs**.

RetrieveSoftwareVersion

Dll HRESULT __stdcall RetrieveSoftwareVersion(
AlphiPciDevice * *pThis***,**
DWORD * *pVersion***)**

Retrieve version of the bootloader (or application if supported).

Defined in: D:/ALPHIPCI/ALPHIDLL/ALPHIDLL.CPP

Parameters

pThis
Pointer to the card object.

pVersion
Pointer to the returned version identifier.

Comments

Bootloaders prior to version 1.7 do not support this method of retrieving the version. For these versions, the serial port will report the version after a RESET (in certain boot modes).

Calls **AlphiPciDevice::RetrieveSoftwareVersion** and returns the result.

Return Codes

ALPHI_S_OK	Operation was successful.
Otherwise	Operation failed. See **HRESULTs**.

ShareHostMemory

Dll HRESULT __stdcall ShareHostMemory(
AlphiPciDevice * *pThis***,**
void * *pSharedMemory***,**
DWORD *Size***)**

Share the designated buffer with the DSP.

Defined in: D:/ALPHIPCI/ALPHIDLL/ALPHIDLL.CPP

Parameters

pThis
Pointer to the card object.

pSharedMemory
Pointer to beginning of shared memory.

Size
Size of buffer to share in bytes.

Comments Allocate and free the buffer using **AllocSharableMemory** and **FreeSharableMemory**. This way, the buffer is guaranteed to start at an address that is divisible by the page size of the processor, and therefore, greatly simplifying the DSP's addressing.

Cancel the sharing by calling **UnshareHostMemory**.

Return Codes

ALPHI_S_OK	Operation was successful.
Otherwise	Operation failed. See **HRESULTs**.

See Also **AllocSharableMemory FreeSharableMemory UnshareHostMemory**

Start

Dll HRESULT __stdcall Start(
    AlphiPciDevice * ***pThis*****)**

Begin execution of the downloaded DSP code.

Defined in: D:/ALPHIPCI/ALPHIDLL/ALPHIDLL.CPP

Parameters

pThis
    Pointer to the card object.

Comments Calls **AlphiPciDevice::Start** and returns the result.

Return Codes

ALPHI_S_OK	Operation was successful.
Otherwise	Operation failed. See **HRESULTs**.

StartX0

Dll HRESULT __stdcall StartX0(
    AlphiPciDevice * ***pThis*****)**

Begin execution of the previously downloaded Tektronix file.

Defined in: D:/ALPHIPCI/ALPHIDLL/ALPHIDLL.CPP

Parameters

pThis
    Pointer to the card object.

Comments Calls **AlphiPciDevice::StartX0** and returns the result.

Return Codes

ALPHI_S_OK	Operation was successful.
Otherwise	Operation failed. See **HRESULTs**.

UnhookMailboxInterrupt

Dll HRESULT __stdcall UnhookMailboxInterrupt(
AlphiPciDevice * *pThis***)**

Remove a hook installed with **HookMailboxInterrupt**.

Defined in: D:/ALPHIPCI/ALPHIDLL/ALPHIDLL.CPP

Parameters

pThis
Pointer to the card object.

Comments

Calls **AlphiPciDevice::UnhookMailboxInterrupt** and returns the result.

Cancels any outstanding interrupt requests, and waits for the interrupt thread to exit.

Return Codes

ALPHI_S_OK	Operation was successful.
Otherwise	Operation failed. See **HRESULTs**.

See Also

HookMailboxInterrupt

Unmap

Dll HRESULT __stdcall Unmap(
AlphiPciDevice * *pThis***,**
LinearAddress * *pAddress***)**

Unmap pointer to physical resource.

Defined in: D:/ALPHIPCI/ALPHIDLL/ALPHIDLL.CPP

Parameters

pThis
Pointer to the card object.

pAddress
Linear address directly accessible by pointer dereference. See **LinearAddress**.

Comments

Calls **AlphiPciDevice::Unmap** and returns the result.

Return Codes

ALPHI_S_OK	Operation was successful.
Otherwise	Operation failed. See **HRESULTs**.

UnshareHostMemory

Dll HRESULT __stdcall UnshareHostMemory(
AlphiPciDevice * *pThis***)**

Disable sharing the buffer with the DSP.

Defined in: D:/ALPHIPCI/ALPHIDLL/ALPHIDLL.CPP

Parameters

pThis
Pointer to the card object.

Return Codes

ALPHI_S_OK	Operation was successful.
Otherwise	Operation failed. See **HRESULTs**.

See Also **ShareHostMemory**

WriteAmccNvram

Dll HRESULT __stdcall WriteAmccNvram(
AlphiPciDevice * *pThis***,**
AmccNvramImage * *pImage***)**

Write the NVRAM image to the device.

Defined in: D:/ALPHIPCI/ALPHIDLL/ALPHIDLL.CPP

Parameters

pThis
Pointer to the card object.

pImage
Pointer to the NVRAM image to copy from.

Comments Calls **AlphiPciDevice::WriteAmccNvram** and returns the result.

Return Codes

ALPHI_S_OK	Operation was successful.
Otherwise	Operation failed. See **HRESULTs**.

See Also **AmccNvramImage**

WriteDword

Dll HRESULT __stdcall WriteDword(
AlphiPciDevice * *pThis***,**
DWORD *DspAddress***,**
DWORD *Value***)**

Ask the Bootloader to write to the specified DSP address.

Defined in: D:/ALPHIPCI/ALPHIDLL/ALPHIDLL.CPP

Parameters

pThis
Pointer to the card object.

DspAddress
Address in the DSP to write to.

Value
Value to be written.

Comments Calls **AlphiPciDevice::WriteDword** and returns the result.

Return Codes

ALPHI_S_OK	Operation was successful.
Otherwise	Operation failed. See **HRESULTs**.

WriteFifo

Dll HRESULT __stdcall WriteFifo(
  AlphiPciDevice * *pThis***,**
  DWORD * *pBuffer***,**
  DWORD *NumDWords***,**
  ucr *UserFunction***,**
  PVOID *UserData***)**

Write to the FIFO port using Bus Master DMA.

Defined in: D:/ALPHIPCI/ALPHIDLL/ALPHIDLL.CPP

Parameters

pThis
  Pointer to the card object.

pBuffer
  Pointer to the buffer to be written.

NumDWords
  Number of DWORDS in the buffer.

UserFunction
  Pointer to user supplied completion function.

UserData
  User data provided to completion function.

Comments

Calls **AlphiPciDevice::WriteFifo** and returns the result.

This function queues a request to the driver to copy the contents of *pBuffer* to the DSP. On the AMCC, this is through the actual FIFO present on the AMCC. On the PLX, this sets up the table of physical pointers to the pages and interrupts the DSP. The function returns immediately.

Upon completion, the user supplied function *UserFunction* is called with the address and size of the buffer, as well as the value *UserData*. The user completion function is called on a separate thread and at a higher priority, and will interrupt the main thread of execution, similiar to a hardware interupt.

Return Codes

ALPHI_S_OK	Operation was successful.
Otherwise	Operation failed. See **HRESULTs**.

See Also

(*ucr) ReadFifo CancelPendingWriteRequests

WriteFifoDirect

Dll HRESULT __stdcall WriteFifoDirect(
  AlphiPciDevice * *pThis***,**

DWORD * *pBuffer***,**
DWORD *NumDWords***)**

Write directly to the FIFO port.

Defined in: D:/ALPHIPCI/ALPHIDLL/ALPHIDLL.CPP

Parameters

pThis
Pointer to the card object.

pBuffer
Pointer to where to find the values to write.

NumDWords
Number of values to be written.

Comments

Calls **AlphiPciDevice::WriteFifoDirect** and returns the result.

Return Codes

ALPHI_S_OK	Operation was successful.
Otherwise	Operation failed. See **HRESULTs**.

WriteFifoDirectByte

Dll HRESULT __stdcall WriteFifoDirectByte(
AlphiPciDevice * *pThis***,**
BYTE * *pBuffer***,**
DWORD *NumBytes***)**

Write directly to the FIFO port.

Defined in: D:/ALPHIPCI/ALPHIDLL/ALPHIDLL.CPP

Parameters

pThis
Pointer to the card object.

pBuffer
Pointer to where to find the values to write.

NumBytes
Number of values to be written.

Comments

Calls **AlphiPciDevice::WriteFifoDirect** and returns the result.

Return Codes

ALPHI_S_OK	Operation was successful.
Otherwise	Operation failed. See **HRESULTs**.

WriteFifoDirectWord

Dll HRESULT __stdcall WriteFifoDirectWord(
AlphiPciDevice * *pThis***,**
WORD * *pBuffer***,**
DWORD *NumWords***)**

Write directly to the FIFO port.

Defined in: D:/ALPHIPCI/ALPHIDLL/ALPHIDLL.CPP

Parameters

pThis
Pointer to the card object.

pBuffer
Pointer to where to find the values to write.

NumWords
Number of values to be written.

Comments

Calls **AlphiPciDevice::WriteFifoDirect** and returns the result.

Return Codes

ALPHI_S_OK	Operation was successful.
Otherwise	Operation failed. See **HRESULTs**.

WriteMbox

Dll HRESULT __stdcall WriteMbox(
AlphiPciDevice * *pThis***,**
WORD *WhichMailbox***,**
DWORD *dwData***)**

Write to the specified PCI mailbox.

Defined in: D:/ALPHIPCI/ALPHIDLL/ALPHIDLL.CPP

Parameters

pThis
Pointer to the card object.

WhichMailbox
Mailbox to write to. Ranges from 0-3.

dwData
Data to write.

Comments

Calls **AlphiPciDevice::WriteMbox** and returns the result.

Smaller values are promoted to DWORDs by the compiler.

Return Codes

ALPHI_S_OK	Operation was successful.
Otherwise	Operation failed. See **HRESULTs**.

WritePlxNvram

Dll HRESULT __stdcall WritePlxNvram(
AlphiPciDevice * *pThis***,**
PlxNvramImage * *pImage***)**

Write the NVRAM image to the device.

Defined in: D:/ALPHIPCI/ALPHIDLL/ALPHIDLL.CPP

Parameters

pThis
Pointer to the card object.

pImage
Pointer to the NVRAM image to copy from.

Comments

Calls **AlphiPciDevice::WritePlxNvram** and returns the result.

Return Codes

ALPHI_S_OK	Operation was successful.
Otherwise	Operation failed. See **HRESULTs**.

See Also

AmccNvramImage

IOCTL Messages

The primary means of communication between the DLL or user's application and the device driver.

IOCTL_ALPHIPCI_CANCEL_SPECIFIC_IO

Description

Ask the device driver to cancel a specific type of pending I/O.

Defined in: D:/ALPHIPCI/INCLUDE/DDALPHIP.H

Comments

Unfortunately, the WIN32 API function **CancelIo** will cancel ALL outstanding I/O requests, and must be called from the originating thread. Since the originating thread is typically blocked waiting for a completion of an **IOCTL_ALPHIPCI_WAIT_FOR_INTERRUPT** request, it cannot be used to cancel only the outstanding **IOCTL_ALPHIPCI_WAIT_FOR_INTERRUPT** requests, leaving the ReadFile and WriteFile requests alone.

Input

IoType
Type of I/O requests to cancel.

Output

none

Return Codes

STATUS_INVALID_PARAMETER	Bad parameter passed.
STATUS_UNSUCCESSFUL	Operation failed because this device not fully map the expected interface.
STATUS_SUCCESS	Operation was successful.

IOCTL_ALPHIPCI_GET_AMCC_MODE

Description

Asks the device driver whether the device has I/O or memory mapped AMCC registers.

Defined in: D:/ALPHIPCI/INCLUDE/DDALPHIP.H

Input

none

Output

bool
True if the AMCC is mapped into I/O space, false if memory space.

Return Codes

STATUS_INVALID_PARAMETER	Bad parameter passed.
STATUS_SUCCESS	Operation was successful.

Developer Notes

This IOCTL exists to support very limited functionality when the AMCC registers are I/O mapped. By default (and preference) no board is shippped in this configuration. It is documented for completeness only, and is not intended for use.

IOCTL_ALPHIPCI_GET_DEVICE_CAPABILITIES

Description Query the driver for the device-based name and the card capabilities. In particular, when the device is opened via the generic "ALPHIPCIn" name, this IOCTL responds with the device class name such as "PCI4PACK0" or "CPCI-SIP0".

Defined in: D:/ALPHIPCI/INCLUDE/DDALPHIP.H

Input *none*

Output ***DeviceCaps***
Returned device capabilities of the card.

Return Codes

STATUS_INVALID_PARAMETER	Bad parameter passed.
STATUS_UNSUCCESSFUL	Operation failed because this device not fully map the expected interface.
STATUS_SUCCESS	Operation was successful.

IOCTL_ALPHIPCI_GET_MAILBOX_STATUS

Description Asks the device driver to retrieve the cached mailbox status.

Defined in: D:/ALPHIPCI/INCLUDE/DDALPHIP.H

Comments Since the doorbell registers are used to monitor the mailbox status, and the interrupts are enabled at all times by the driver, the mailbox status must be read and stored in the driver. This IOCTL allows AlphiDll to query it's current state and to clear select bits.

Input ***MailboxStatus***
Structure with **MailboxStatus**.Value containing the bits to fetch (and clear).

Output ***MailboxStatus***
Returned structure with **MailboxStatus**.Value containing the returned bits.

Return Codes

STATUS_INVALID_PARAMETER	Bad parameter passed.
STATUS_SUCCESS	Operation was successful.

IOCTL_ALPHIPCI_GET_PHYS_ADDRS_OF_CARDS_RESOURCES

Description Query the driver for the physical addresses of the AMCC registers and the passthrough regions. This is useful for users who access the card with realtime add-ons to Windows NT which cannot directly enumerate the PCI bus for this information.

Defined in: D:/ALPHIPCI/INCLUDE/DDALPHIP.H

Input *none*

Output ***PhysAddrsOfCardResources***
Returned physical addresses of card resources.

Return Codes

STATUS_INVALID_PARAMETER	Bad parameter passed.
STATUS_UNSUCCESSFUL	Operation failed because this device not fully map the expected interface.
STATUS_SUCCESS	Operation was successful.

IOCTL_ALPHIPCI_MAP

Description Ask the driver to set up a mapping of physical addresses for the specified card resource in the page table for this task.

Defined in: D:/ALPHIPCI/INCLUDE/DDALPHIP.H

Input ***CardResource***
Describes the physical range to map and the type of access.

Output ***LinearAddress***
Returned linear pointer and size if the operation was successful.

Return Codes

STATUS_INVALID_PARAMETER	Bad parameter passed.
STATUS_ACCESS_VIOLATION	Would result in an invalid access.
STATUS_UNSUCCESSFUL	Operation failed because this device not fully map the expected interface.
STATUS_SUCCESS	Operation was successful.

See Also Maps

IOCTL_ALPHIPCI_READ_AMCC_NVRAM

Description Asks the device driver to read the contents of the NVRAM into the supplied buffer.

Defined in: D:/ALPHIPCI/INCLUDE/DDALPHIP.H

Input *none*

Output ***AmccNvramImage***
NVRAM Image from the AMCC.

Return Codes

STATUS_INVALID_PARAMETER	Bad parameter passed.
STATUS_SUCCESS	Operation was successful.

Developer Notes In spite of the name, it works on the PLX too. Call with a **PlxNvramImage** structure.

IOCTL_ALPHIPCI_READ_PORT

Description Asks the device driver to read the specified AMCC register as an I/O port.

Defined in: D:/ALPHIPCI/INCLUDE/DDALPHIP.H

Comments Reads the 32 bit register in the AMCC registers at the specified offset.

Input ***PortType***
Description of what to read.

Output ***PortType***
Returned structure with value read.

Return Codes

STATUS_INVALID_PARAMETER	Bad parameter passed.
STATUS_SUCCESS	Operation was successful.

Developer Notes This IOCTL exists to support very limited functionality when the AMCC registers are I/O mapped. By default (and preference) no board is shippped in this configuration. It is documented for completeness only, and is not intended for use.

IOCTL_ALPHIPCI_REPORT_VERSION_IDENTIFIER

Description Ask the driver to report its version identifier. That is, report the value of szDriverVersionIdentifier from when it was compiled.

Defined in: D:/ALPHIPCI/INCLUDE/DDALPHIP.H

Input *none*

Output ***DriverVersionIdentifier***
Returned version identifier.

Return Codes

STATUS_INVALID_PARAMETER	Bad parameter passed.
STATUS_SUCCESS	Operation was successful.

IOCTL_ALPHIPCI_SELECT_MAILBOX_FOR_INTERRUP T

Description Allows the selection of the mailbox which will generate an interrupt.

Defined in: D:/ALPHIPCI/INCLUDE/DDALPHIP.H

Comments Because the interrupt handling and FIFO Bus Master DMA logic use the interrupt control register rather intimately, this IOCTL will allow for its manipulation without causing interference.

Input ***IntType***
Type of mailbox interrupt to enable or disable.

Output *none*

Return Codes

STATUS_INVALID_PARAMETER	Bad parameter passed.
STATUS_UNSUCCESSFUL	Operation failed because this device not fully map the expected interface.
STATUS_SUCCESS	Operation was successful.

IOCTL_ALPHIPCI_SHARE_HOST_MEMORY

Description Asks the device driver to make the specified buffer accessible to the DSP.

Defined in: D:/ALPHIPCI/INCLUDE/DDALPHIP.H

Comments Windows NT utilizes paged memory during its normal processing. This means that a normal block of memory might be swapped out to disk at any point, and then be swapped in when a page fault occurs. Normally, this happens transparently to the application. Additionally, the physical page address has no bearing on the logical address that the HOST application deals with. This complication is handled by the DSP and the HOST by using this IOCTL to make the application buffer accessible to the DSP by 1) page locking the buffer into HOST memory so that the DSP can access it, 2) once the pages are locked, they will not be moved in physical address by the system, 3) creating a mapping table in DSP memory so that the logical address (read: offset into a shared buffer) can be mapped to physical page addresses. The DSP application still must call the alphi_io library to make sure that the physical page is accessible, but the tables are handled automatically.

When the sharing is finished, the HOST calls **IOCTL_ALPHIPCI_CANCEL_SPECIFIC_IO**, which cancels the mapping.

Input *pBuffer*
Pointer uffer to be shared.

Output *none*

Return Codes

STATUS_INVALID_PARAMETER	Bad parameter passed.
STATUS_SUCCESS	Operation was successful.

IOCTL_ALPHIPCI_UNMAP

Description Ask the driver to delete a previous mapping.

Defined in: D:/ALPHIPCI/INCLUDE/DDALPHIP.H

Input ***LinearAddress***
Linear pointer previously returned by **IOCTL_ALPHIPCI_MAP**.

Output *none*

Return Codes

STATUS_INVALID_PARAMETER	Bad parameter passed.
STATUS_UNSUCCESSFUL	Operation failed because this device not fully map the expected interface.
STATUS_SUCCESS	Operation was successful.

See Also Maps

IOCTL_ALPHIPCI_WAIT_FOR_INTERRUPT

Description Informs the driver that the user task is interested in being informed when a interrupt is received from the device as a result of mailbox activity.

Defined in: D:/ALPHIPCI/INCLUDE/DDALPHIP.H

Comments This IOCTL is placed in a queue separate from the read fifo and write fifo queues, and will complete upon receipt of an interrupt. Multiple requests can be queued, however, the user task will have to run and clear the cause before the same type of interrupt can be received.

The real interrupt routine in the driver stores the value of the INTCSR register in the AMCC 5933, resets the appropriate bits in the INTCSR to stop the interruption, and returns. Later another routine checks if there were any outstanding IOCTLs of this type in the queue. If there is, then the IOCTL is returned with the INTCSR from the interrupt routine. If there were no IOCTLs pending, the INTCSR is stored, and a later IOCTL request will return immediately with this result. No interrupts will be lost.

The user task is responsible for enabling mailbox interrupts on the 5933, and selecting the appropriate mailboxes. It is strongly urged that the **IOCTL_ALPHIPCI_SELECT_MAILBOX_FOR_INTERRUPT** IOCTL be used to do this.

Input *none*

Output ***InterruptCause***

Returned INTCSR in the AMCC 5933 at the time of the interrupt.

Return Codes

STATUS_INVALID_PARAMETER	Bad parameter passed.
STATUS_UNSUCCESSFUL	Operation failed because this device not fully map the expected interface.
STATUS_SUCCESS	Operation was successful.

IOCTL_ALPHIPCI_WRITE_AMCC_NVRAM

Description Asks the device driver to write the contents of the supplied buffer into the NVRAM.

Defined in: D:/ALPHIPCI/INCLUDE/DDALPHIP.H

Comments

No confirmation of write is performed.

Although this IOCTL was implemented prior to Revision D of AlphiPci, it did not operate correctly in those revisions.

Input

AmccNvramImage
NVRAM Image to program the AMCC with.

Output

none

Return Codes

STATUS_INVALID_PARAMETER	Bad parameter passed.
STATUS_SUCCESS	Operation was successful.

Developer Notes

In spite of the name, it works on the PLX too. Call with a **PlxNvramImage** structure.

IOCTL_ALPHIPCI_WRITE_PORT

Description

Asks the device driver to write the specified AMCC register as an I/O port.

Defined in: D:/ALPHIPCI/INCLUDE/DDALPHIP.H

Comments

Writes the 32 bit register in the AMCC registers at the specified offset.

Input

PortType
Description of what to write.

Output

none

Return Codes

STATUS_INVALID_PARAMETER	Bad parameter passed.
STATUS_SUCCESS	Operation was successful.

Developer Notes

This IOCTL exists to support very limited functionality when the AMCC registers are I/O mapped. By default (and preference) no board is shippped in this configuration. It is documented for completeness only, and is not intended for use.

Constants and Typedefs

Implementation details of this package.

AllowAmccRegisters constant

ULONG AllowAmccRegisters;

Defined in: D:/ALPHIPCI/INCLUDE/DDALPHIP.H

Allow access to the AMCC PCI Operation Registers.

AllowDualPortRam constant

ULONG AllowDualPortRam;

Defined in: D:/ALPHIPCI/INCLUDE/DDALPHIP.H

Allow access to the Dual Ported RAM.

AllowedAccess_t

Represents the type of access allowed on this device. Any of the following ORed together.

Defined in: D:/ALPHIPCI/INCLUDE/DDALPHIP.H

See Also **AllowAmccRegisters AllowPlxRegisters AllowDualPortRam AllowIp0IdSpace AllowIp0IoSpace AllowIp0MemorySpace AllowIp0All AllowIp1IdSpace AllowIp1IoSpace AllowIp1MemorySpace AllowIp1All AllowIp2IdSpace AllowIp2IoSpace AllowIp2MemorySpace AllowIp2All AllowIp3IdSpace AllowIp3IoSpace AllowIp3MemorySpace AllowIp3All AllowSummitRegisters AllowHostControlRegion**

AllowHostControlRegion constant

ULONG AllowHostControlRegion;

Defined in: D:/ALPHIPCI/INCLUDE/DDALPHIP.H

Allow access to the Direct hardware access area.

AllowIp0All constant

ULONG AllowIp0All;

Defined in: D:/ALPHIPCI/INCLUDE/DDALPHIP.H

Allow access to the IP A Memory Space.

AllowIp0IdSpace constant

ULONG AllowIp0IdSpace;

Defined in: D:/ALPHIPCI/INCLUDE/DDALPHIP.H

Allow access to the IP A ID Space.

AllowIp0IoSpace constant

ULONG AllowIp0IoSpace;

Defined in: D:/ALPHIPCI/INCLUDE/DDALPHIP.H

Allow access to the IP A IO Space.

AllowIp0MemorySpace constant

ULONG AllowIp0MemorySpace;

Defined in: D:/ALPHIPCI/INCLUDE/DDALPHIP.H

Allow access to the IP A Memory Space.

AllowIp1All constant

ULONG AllowIp1All;

Defined in: D:/ALPHIPCI/INCLUDE/DDALPHIP.H

Allow access to the IP B.

AllowIp1IdSpace constant

ULONG AllowIp1IdSpace;

Defined in: D:/ALPHIPCI/INCLUDE/DDALPHIP.H

Allow access to the IP B ID Space.

AllowIp1IoSpace constant

ULONG AllowIp1IoSpace;

Defined in: D:/ALPHIPCI/INCLUDE/DDALPHIP.H

Allow access to the IP B IO Space.

AllowIp1MemorySpace constant

ULONG AllowIp1MemorySpace;

Defined in: D:/ALPHIPCI/INCLUDE/DDALPHIP.H

Allow access to the IP B Memory Space.

AllowIp1Special constant

ULONG AllowIp1Special;

Defined in: D:/ALPHIPCI/INCLUDE/DDALPHIP.H

Allow access to the IP B. Memory shared under A.

AllowIp2All constant

ULONG AllowIp2All;

Defined in: D:/ALPHIPCI/INCLUDE/DDALPHIP.H

Allow access to IP C.

AllowIp2IdSpace constant

ULONG AllowIp2IdSpace;

Defined in: D:/ALPHIPCI/INCLUDE/DDALPHIP.H

Allow access to the IP C ID Space.

AllowIp2IoSpace constant

ULONG AllowIp2IoSpace;

Defined in: D:/ALPHIPCI/INCLUDE/DDALPHIP.H

Allow access to the IP C IO Space.

AllowIp2MemorySpace constant

ULONG AllowIp2MemorySpace;

Defined in: D:/ALPHIPCI/INCLUDE/DDALPHIP.H

Allow access to the IP C Memory Space.

AllowIp2Special constant

ULONG AllowIp2Special;

Defined in: D:/ALPHIPCI/INCLUDE/DDALPHIP.H

Allow access to IP C. Memory shared under A.

AllowIp3All constant

ULONG AllowIp3All;

Defined in: D:/ALPHIPCI/INCLUDE/DDALPHIP.H

Allow access to IP D.

AllowIp3IdSpace constant

ULONG AllowIp3IdSpace;

Defined in: D:/ALPHIPCI/INCLUDE/DDALPHIP.H

Allow access to the IP D ID Space.

AllowIp3IoSpace constant

ULONG AllowIp3IoSpace;

Defined in: D:/ALPHIPCI/INCLUDE/DDALPHIP.H

Allow access to the IP D IO Space.

AllowIp3MemorySpace constant

ULONG AllowIp3MemorySpace;

Defined in: D:/ALPHIPCI/INCLUDE/DDALPHIP.H

Allow access to the IP D Memory Space.

AllowIp3Special constant

ULONG AllowIp3Special;

Defined in: D:/ALPHIPCI/INCLUDE/DDALPHIP.H

Allow access to IP D. Memory shared under A.

AllowPlxRegisters constant

ULONG AllowPlxRegisters;

Defined in: D:/ALPHIPCI/INCLUDE/DDALPHIP.H

Allow access to the PLX PCI Operation Registers.

AllowSummitRegisters constant

ULONG AllowSummitRegisters;

Defined in: D:/ALPHIPCI/INCLUDE/DDALPHIP.H

Allow access to the Summit Registers (1553)

szDriverVersionIdentifier[] constant

char const szDriverVersionIdentifier[];

Defined in: D:/ALPHIPCI/INCLUDE/DDALPHIP.H

This is the version identifier string. If any changes are made to the user interface, this must be changed.

Structures and Enumerations

Non class type data structures and enumerations. Also used to pass data during IOCTL calls to the driver.

AMCC Structure

```
typedef struct {
    volatile ULONG OMB1;
    volatile ULONG OMB2;
    volatile ULONG OMB3;
    volatile ULONG OMB4;
    volatile ULONG IMB1;
    volatile ULONG IMB2;
    volatile ULONG IMB3;
    volatile ULONG IMB4;
    volatile ULONG FIFO;
    volatile ULONG MWAR;
    volatile ULONG MWTC;
    volatile ULONG MRAR;
    volatile ULONG MRTC;
    volatile ULONG MBEF;
    volatile ULONG INTCSR;
    volatile ULONG MCSR;
} AMCC;
```

AMCC S5933 Operation Registers as seen from the PCI bus

Defined in: D:/ALPHIPCI/INCLUDE/DDALPHIP.H

Members

OMB1
Outgoing Mailbox Register 1

OMB2
Outgoing Mailbox Register 2

OMB3
Outgoing Mailbox Register 3

OMB4
Outgoing Mailbox Register 4

IMB1
Incoming Mailbox Register 1

IMB2
Incoming Mailbox Register 2

IMB3
Incoming Mailbox Register 3

IMB4
Incoming Mailbox Register 4

FIFO
FIFO Register port (bidirectional)

MWAR
Master Write Address Register

MWTC
Master Write Transfer Count Register

MRAR
Master Read Address Register

MRTC
Master Read Transfer Count Register

MBEF
Mailbox Empty/Full Status

INTCSR
Interrupt Control/Status Register

MCSR
Bus Master Control/Status Register

AmccNvramImage Structure

```
typedef struct {
    USHORT x_RomSignature;
    UCHAR x_SizeInBlocks;
    ULONG x_EntryPoint;
    UCHAR x_Unused1;
    ULONG m_UserId;
    ULONG m_SizeDualPortRam;
    ULONG m_ClockSpeedDsp;
    ULONG m_ClockSpeed8530;
    ULONG m_BootOption;
    ULONG m_SerialNumber;
    UCHAR m_HardwareRevision[4];
    UCHAR m_ProgrammedLogicRevision[4];
    UCHAR x_Unused2[0x18];
    USHORT m_VendorId;
    USHORT m_DeviceId;
    UCHAR x_Unused3;
    UCHAR m_BusMasterConfig;
    UCHAR x_Unused4[2];
    UCHAR m_RevisionId;
    ULONG m_ClassCode;
    UCHAR m_LatencyTimer;
    UCHAR m_HeaderType;
    UCHAR m_SelfTest;
    ULONG m_Bar[6];
    UCHAR x_Unused5[8];
    ULONG m_ExpansionRom;
    UCHAR x_Unused6[8];
    UCHAR m_IntLine;
    UCHAR m_IntPin;
    UCHAR m_MinGrant;
    UCHAR m_MaxLatency;
} AmccNvramImage;
```

NVRAM Image format. See the AMCC documentation for certain details.

Defined in: D:/ALPHIPCI/INCLUDE/DDALPHIP.H

Members

x_RomSignature
Not used. (0-1)

x_SizeInBlocks
Not used. (2)

x_EntryPoint
Not used. (3-6)

x_Unused1
Not used. (7)

m_UserId
User ID. (8-b)

m_SizeDualPortRam
Size of Dual Port RAM in bytes. (c-f)

m_ClockSpeedDsp
Clock speed of the DSP in Hertz. (10-13)

m_ClockSpeed8530
Clock speed of PCLK at the 8530 in Hertz. (14-17)

m_BootOption
What to do at RESET. (18-1b)

m_SerialNumber
Serial number in decimal. (1c-1f)

m_HardwareRevision[4]
Three character string describing hardware version. (20-23)

m_ProgrammedLogicRevision[4]
Three character string describing FPGA version. (24-27)

x_Unused2[0x18]
Not used. (28-3f)

m_VendorId
Vendor ID. (40-41)

m_DeviceId
Device ID. (42-43)

x_Unused3
Not used. (44)

m_BusMasterConfig
AMCC Special Options. (45)

x_Unused4[2]
Not used. (46-47)

m_RevisionId
Revision ID. (48)

m_ClassCode
Class code. (49-4c with 4c not used)

m_LatencyTimer
Latency timer. (4d)

m_HeaderType
Specific header type. (4e)

m_SelfTest
Self test. (4f)

m_Bar[6]
Base Address Registers. (50-67)

x_Unused5[8]
Not used. (68-6f)

m_ExpansionRom
Expansion ROM Address. (70-73)

x_Unused6[8]
Not used. (74-7b)

m_IntLine
Interrupt line. (7c)

m_IntPin
Interrupt pin. (7d)

m_MinGrant
Minimum grant. (7e)

m_MaxLatency
Maximum latency. (7f)

CardResource Structure

```
typedef struct {
    eTypeOfAccess TypeOfAccess;
} CardResource;
```

Describes the type of access requested.

Defined in: D:/ALPHIPCI/INCLUDE/DDALPHIP.H

Members

TypeOfAccess
Type of access desired. See **eTypeOfAccess**.

DeviceCaps Structure

```
typedef struct {
    USHORT wszTrueName[SIZE_TRUE_NAME];
    AllowedAccess_t AllowedAccess;
    eInterfaceType InterfaceType;
    eProcessorType ProcessorType;
    ULONG ProcessorSpeed;
    ULONG NumBytesDualPortRam;
    ULONG NumIpsAccessibleFromHost;
    ULONG NumIpsTotal;
    ULONG UserIdentifier;
    ULONG BootOption;
    USHORT VendorId;
    USHORT DeviceId;
} DeviceCaps;
```

Represents the device capabilities for this card.

Defined in: D:/ALPHIPCI/INCLUDE/DDALPHIP.H

Members

wszTrueName[SIZE_TRUE_NAME]
NULL terminated UNICODE buffer to receive the true name.

AllowedAccess
Allowed PCI accessible resources. See **AllowedAccess_t**.

InterfaceType
Type of PCI Interface on card. See **eInterfaceType**.

ProcessorType
Type of processor on card. See **eProcessorType**.

ProcessorSpeed
Clock speed given to the DSP.

NumBytesDualPortRam
Size of the dual port RAM.

NumIpsAccessibleFromHost
Number of IPs directly accessible from the host processor.

NumIpsTotal
Total number of IPs on the card.

UserIdentifier
User specified identification.

BootOption
How the board is set to boot.

VendorId
PCI vendor identification.

DeviceId
PCI device identification.

DriverVersionIdentifier Structure

```
typedef struct {
    char VersionIdentifier[SIZE_VERSION_IDENTIFIER];
} DriverVersionIdentifier;
```

Returned version identifier from the driver.

Defined in: D:/ALPHIPCI/INCLUDE/DDALPHIP.H

Members

VersionIdentifier[SIZE_VERSION_IDENTIFIER]
Buffer to receive the version identifier.

eInterfaceType

```
enum eInterfaceType {
    AMCC_5933,
    PLX_9080,
    OHCI_1394
};
```

Type of the PCI interface on the card.

Defined in: D:/ALPHIPCI/INCLUDE/DDALPHIP.H

Members

AMCC_5933
AMCC 5933 interface.

PLX_9080
PLX PCI 9080 interface.

OHCI_1394
1394 interface through host OHCI driver.

eIntType

```
enum eIntType {
    Disable,
    SwMb1_FullFromCard,
    SwMb2_FullFromCard,
    SwMb3_FullFromCard,
    SwMb4_FullFromCard,
    HwMb_FullFromCard,
    SwMb1_EmptyToCard,
    SwMb2_EmptyToCard,
    SwMb3_EmptyToCard,
    SwMb4_EmptyToCard
};
```

Type of mailbox interrupt. Applicable only to AMCC devices.

Defined in: D:/ALPHIPCI/INCLUDE/ALPHIDLL.H

Members

Disable
Mailbox interupts disabled.

SwMb1_FullFromCard
Processor on card has written to (LSB of) Mailbox 1.

SwMb2_FullFromCard
Processor on card has written to (LSB of) Mailbox 2.

SwMb3_FullFromCard
Processor on card has written to (LSB of) Mailbox 3.

SwMb4_FullFromCard
Processor on card has written to (LSB of) Mailbox 4.

HwMb_FullFromCard
Hardware on card has written to (MSB of) Mailbox 4.

SwMb1_EmptyToCard
Processor on card has read from (LSB of) Mailbox 1.

SwMb2_EmptyToCard
Processor on card has read from (LSB of) Mailbox 2.

SwMb3_EmptyToCard
Processor on card has read from (LSB of) Mailbox 3.

SwMb4_EmptyToCard
Processor on card has read from (LSB of) Mailbox 4.

eIoType

```
enum eIoType {
    ReadFileRequest,
    WriteFileRequest,
    WaitForInterruptRequest,
    HostSharedMemory
};
```

Type of I/O Request.

Defined in: D:/ALPHIPCI/INCLUDE/DDALPHIP.H

Members

ReadFileRequest
ReadFile requests

WriteFileRequest
WriteFile requests

WaitForInterruptRequest
IOCTL_ALPHIPCI_WAIT_FOR_INTERRUPT requests.

HostSharedMemory
Shared HOST memory from **IOCTL_ALPHIPCI_SHARE_HOST_MEMORY**.

eProcessorType

```
enum eProcessorType {
    None,
    Single_C31,
    Single_C32
};
```

Presence and type of the processor on the card.

Defined in: D:/ALPHIPCI/INCLUDE/DDALPHIP.H

Members

None
No processor present.

Single_C31
TMS320C31 DSP Processor.

Single_C32
TMS320C32 DSP Processor.

eTypeOfAccess

```
enum eTypeOfAccess {
    AmccRegisters,
    PlxRegisters,
    DualPortRam,
    Ip0IdSpace,
    Ip0IoSpace,
    Ip0MemorySpace,
    Ip1IdSpace,
    Ip1IoSpace,
    Ip1MemorySpace,
    Ip2IdSpace,
    Ip2IoSpace,
    Ip2MemorySpace,
    Ip3IdSpace,
    Ip3IoSpace,
    Ip3MemorySpace,
    SummitRegisters,
    HostControlRegion,
    LastEntry
};
```

Represents the type of access requested.

Defined in: D:/ALPHIPCI/INCLUDE/DDALPHIP.H

Comments

Not all of these are necessarily applicable to the device you are communicating with.

Developer Notes

In hindsight, the HostControlRegion and the SummitRegisters should probably be the same region.

Members

AmccRegisters
AMCC PCI Operation Registers.

PlxRegisters
PLX PCI Operation Registers.

DualPortRam
Dual Ported RAM.

Ip0IdSpace
IP A ID Space.

Ip0IoSpace
IP A IO Space.

Ip0MemorySpace
IP A Memory Space.

Ip1IdSpace
IP B ID Space.

Ip1IoSpace
IP B IO Space.

Ip1MemorySpace
IP B Memory Space.

Ip2IdSpace
IP C ID Space.

Ip2IoSpace
IP C IO Space.

Ip2MemorySpace
IP C Memory Space.

Ip3IdSpace
IP D ID Space.

Ip3IoSpace
IP D IO Space.

Ip3MemorySpace
IP D Memory Space.

SummitRegisters
Summit Registers (1553)

HostControlRegion
Direct access (by HOST) to hardware on the card.

LastEntry
Last Entry.

InterruptCause Structure

```
typedef struct {
    ULONG Value;
} InterruptCause;
```

Represents the value read from the AMCC interrupt control status register during the interrupt routine.

Defined in: D:/ALPHIPCI/INCLUDE/DDALPHIP.H

Members

Value
Value that was in **AMCC**.**INTCSR** at the time of the interrupt.

IntType Structure

```
typedef struct {
    ULONG Type;
} IntType;
```

Type of interrupt.

Defined in: D:/ALPHIPCI/INCLUDE/DDALPHIP.H

Members

Type
Type of interrupt. See **eIntType**.

IoType Structure

```
typedef struct {
    eIoType Type;
} IoType;
```

Type of pending I/O requests to cancel.

Defined in: D:/ALPHIPCI/INCLUDE/DDALPHIP.H

Members

Type
Type of pending I/O requests to cancel. See **eIoType**.

LinearAddress Structure

```
typedef struct {
    PVOID Address;
    ULONG Length;
} LinearAddress;
```

Represents a linear address directly accessible by pointer dereference.

Defined in: D:/ALPHIPCI/INCLUDE/DDALPHIP.H

Members

Address
Linear address.

Length
Length of the mapping.

MailboxStatus Structure

```
typedef struct {
    ULONG Value;
} MailboxStatus;
```

Represents either the mailboxes of interest or the current mailbox status.

Defined in: D:/ALPHIPCI/INCLUDE/DDALPHIP.H

Members

Value
Value.

PhysAddrsOfCardResources Structure

```
typedef struct {
    ULONG PhysAddrAmccRegisters;
    ULONG LengthAmccRegisters;
    ULONG PhysAddrRegion1;
    ULONG LengthRegion1;
    ULONG PhysAddrRegion2;
    ULONG LengthRegion2;
} PhysAddrsOfCardResources;
```

Returns the physical adresses of the card resources.

Defined in: D:/ALPHIPCI/INCLUDE/DDALPHIP.H

Members

PhysAddrAmccRegisters
Physical address of the (memory mapped) AMCC registers.

LengthAmccRegisters
Length of the AMCC registers.

PhysAddrRegion1
Physical address of the first passthrough region (usually region 1).

LengthRegion1
Length of the first passthrough region.

PhysAddrRegion2
Physical address of the second passthrough region (usually region 3).

LengthRegion2
Length of the second passthrough region.

PLX_REGS Structure

```
typedef struct {
    volatile ULONG las0rr;
    volatile ULONG las0ba;
    volatile ULONG barbr;
    volatile ULONG bigend;
    volatile ULONG eromrr;
    volatile ULONG eromba;
    volatile ULONG lbrd0;
    volatile ULONG dmrr;
    volatile ULONG dmlbam;
    volatile ULONG dmlbai;
    volatile ULONG dmpbam;
    volatile ULONG dmcfga;
    volatile ULONG oplfis;
    volatile ULONG oplfim;
    volatile ULONG mbox[8];
    volatile ULONG p2ldbell;
    volatile ULONG l2pdbell;
    volatile ULONG intcsr;
    volatile ULONG cntrl;
    volatile ULONG pcihidr;
    volatile ULONG pcihrev;
    volatile ULONG dmamode0;
    volatile ULONG dmapadr0;
    volatile ULONG dmaladr0;
    volatile ULONG dmasiz0;
    volatile ULONG dmadpr0;
    volatile ULONG dmamode1;
    volatile ULONG dmapadr1;
    volatile ULONG dmaladr1;
    volatile ULONG dmasiz1;
    volatile ULONG dmadpr1;
    volatile ULONG dmacsr;
    volatile ULONG dmaarb;
    volatile ULONG dmathr;
    volatile ULONG mqcr;
    volatile ULONG qbar;
    volatile ULONG ifhpr;
    volatile ULONG iftpr;
    volatile ULONG iphpr;
    volatile ULONG ipfpr;
    volatile ULONG ofhpr;
    volatile ULONG oftpr;
    volatile ULONG ophpr;
    volatile ULONG opfpr;
    volatile ULONG qsr;
    volatile ULONG las1rr;
    volatile ULONG las1ba;
    volatile ULONG lbrd1;
} PLX_REGS;
```

PLX 9080 Operation Registers as seen from the PCI bus.

Defined in: D:/ALPHIPCI/INCLUDE/DDALPHIP.H

Members

las0rr
PCI to local address space 0 range.

las0ba
PCI to local address space 0 base address.

barbr
Mode / Arbitration Register.

bigend
Big/Litle Endian Descriptor Register.

eromrr
Expansion ROM range register.

eromba
Expansion ROM base address.

lbrd0
local address space 0 / EROM descriptor register.

dmrr
Local to PCI range register.

dmlbam
Local to PCI base address memory.

dmlbai
Local to PCI base address I/O or CFG.

dmpbam
Local to PCI remap register for memory.

dmcfga
Local to PCI configuration register for I/O or CFG.

oplfis
Outbound post list FIFO interrupt status register.

oplfim
Outbound post list FIFO interrupt mask register.

mbox[8]
Mailbox registers.

p2ldbell
PCI to local doorbell register.

l2pdbell
Local to PCI doorbell register.

intcsr
Interrupt control / status register.

cntrl
Serial NVRAM, PCI Commands, User IO, Init.

pcihidr
Permanent configuration ID register.

pcihrev
Permanent revision register.

dmamode0
DMA 0 mode register.

dmapadr0
DMA 0 PCI address register.

dmaladr0
DMA 0 local address register.

dmasiz0
DMA 0 size register (BYTES).

dmadpr0
DMA 0 descriptor pointer register.

dmamode1
DMA 1 mode register.

dmapadr1
DMA 1 PCI address register.

dmaladr1
DMA 1 local address register.

dmasiz1
DMA 1 size register (BYTES).

dmadpr1
DMA 1 descriptor pointer register.

dmacsr
DMA 0/1 control / status register.

dmaarb
Mode / Arbitration Register (repeated).

dmathr
DMA threshold register.

mqcr
Messaging queue configuration register.

qbar
Queue base address register.

ifhpr
Inbound free head pointer register.

iftpr
Inbound free tail pointer register.

iphpr
Inbound post head pointer register.

ipfpr
Inbound post tail pointer register.

ofhpr
Outbound free head pointer register.

oftpr
Outbound free tail pointer register.

ophpr
Outbound post head pointer register.

opfpr
Outbound post tail pointer register.

qsr
Queue status / control register

las1rr
PCI to local address space 1 range.

las1ba
PCI to local address space 1 base address.

lbrd1
PCI to local address space 1 descriptor register.

PlxNvramImage Structure

```
typedef struct {
    USHORT m_DeviceId;
    USHORT m_VendorId;
    USHORT m_ClassCode_H;
    USHORT m_ClassCode_L;
    UCHAR m_MinGrant;
    UCHAR m_MaxLatency;
    UCHAR m_IntLine;
    UCHAR m_IntPin;
    USHORT m_MB0_H;
    USHORT m_MB0_L;
    USHORT m_MB1_H;
    USHORT m_MB1_L;
    USHORT m_las0rr_H;
    USHORT m_las0rr_L;
    USHORT m_las0ba_H;
    USHORT m_las0ba_L;
    USHORT m_barbr_H;
    USHORT m_barbr_L;
    USHORT m_bigend_H;
    USHORT m_bigend_L;
    USHORT m_eromrr_H;
    USHORT m_eromrr_L;
    USHORT m_eromba_H;
    USHORT m_eromba_L;
    USHORT m_lbrd0_H;
    USHORT m_lbrd0_L;
    USHORT m_dmrr_H;
    USHORT m_dmrr_L;
    USHORT m_dmlbam_H;
    USHORT m_dmlbam_L;
    USHORT m_dmlbai_H;
    USHORT m_dmlbai_L;
    USHORT m_dmpbam_H;
    USHORT m_dmpbam_L;
    USHORT m_dmcfga_H;
    USHORT m_dmcfga_L;
    USHORT m_SubsysDeviceId;
    USHORT m_SubsysVendorId;
    USHORT m_las1rr_H;
    USHORT m_las1rr_L;
    USHORT m_las1ba_H;
    USHORT m_las1ba_L;
    USHORT m_lbrd1_H;
    USHORT m_lbrd1_L;
    USHORT m_ExpansionRom_H;
    USHORT m_ExpansionRom_L;
    ULONG m_UserId;
    ULONG m_SizeDualPortRam;
    ULONG m_ClockSpeedDsp;
    ULONG m_ClockSpeed8530;
    ULONG m_BootOption;
    ULONG m_SerialNumber;
    UCHAR m_HardwareRevision[4];
    UCHAR m_ProgrammedLogicRevision[4];
} PlxNvramImage;
```

NVRAM Image format. See the PLX documentation for certain details.

Defined in: D:/ALPHIPCI/INCLUDE/DDALPHIP.H

Members

m_DeviceId
Device ID. (0-1)

m_VendorId
Vendor ID. (2-3)

m_ClassCode_H
Class code. (4-5)

m_ClassCode_L
Class code. (6-7)

m_MinGrant
Minimum grant. (8)

m_MaxLatency
Maximum latency. (9)

m_IntLine
Interrupt line. (a)

m_IntPin
Interrupt pin. (b)

m_MB0_H
Mailbox0 Hi Word. (c-d)

m_MB0_L
Mailbox0 Lo Word. (e-f)

m_MB1_H
Mailbox1 Hi Word. (10-11)

m_MB1_L
Mailbox1 Lo Word. (12-13)

m_las0rr_H
PCI to local address space 0 range. (14-15)

m_las0rr_L
PCI to local address space 0 range. (16-17)

m_las0ba_H
PCI to local address space 0 base address. (18-19)

m_las0ba_L
PCI to local address space 0 base address. (1a-1b)

m_barbr_H
Mode / Arbitration Register. (1c-1d)

m_barbr_L
Mode / Arbitration Register. (1e-1f)

m_bigend_H
Big/Litle Endian Descriptor Register. (20-21)

m_bigend_L
Big/Litle Endian Descriptor Register. (22-23)

m_eromrr_H
Expansion ROM range register. (24-25)

m_eromrr_L
Expansion ROM range register. (26-27)

m_eromba_H
Expansion ROM base address. (28-29)

m_eromba_L
Expansion ROM base address. (2a-2b)

m_lbrd0_H
local address space 0 / EROM descriptor register. (2c-2d)

m_lbrd0_L
local address space 0 / EROM descriptor register. (2e-2f)

m_dmrr_H
Local to PCI range register. (30-31)

m_dmrr_L
Local to PCI range register. (32-33)

m_dmlbam_H
Local to PCI base address memory. (34-35)

m_dmlbam_L
Local to PCI base address memory. (36-37)

m_dmlbai_H
Local to PCI base address I/O or CFG. (38-39)

m_dmlbai_L
Local to PCI base address I/O or CFG. (3a-3b)

m_dmpbam_H
Local to PCI remap register for memory. (3c-3d)

m_dmpbam_L
Local to PCI remap register for memory. (3e-3f)

m_dmcfga_H
Local to PCI configuration register for I/O or CFG. (40-41)

m_dmcfga_L
Local to PCI configuration register for I/O or CFG. (42-43)

m_SubsysDeviceId
Device ID. (44-45)

m_SubsysVendorId
Vendor ID. (46-47)

m_las1rr_H
PCI to local address space 1 range. (48-49)

m_las1rr_L
PCI to local address space 1 range. (4a-4b)

m_las1ba_H
PCI to local address space 1 base address. (4c-4d)

m_las1ba_L
PCI to local address space 1 base address. (4e-4f)

m_lbrd1_H
PCI to local address space 1 descriptor register. (50-51)

m_lbrd1_L
PCI to local address space 1 descriptor register. (52-53)

m_ExpansionRom_H
Expansion ROM Address. (54-55)

m_ExpansionRom_L
Expansion ROM Address. (56-57)

m_UserId
User ID. (5a-5b)

m_SizeDualPortRam
Size of Dual Port RAM in bytes. (5e-5f)

m_ClockSpeedDsp
Clock speed of the DSP in Hertz. (62-63)

m_ClockSpeed8530
Clock speed of PCLK at the 8530 in Hertz. (66-67)

m_BootOption
What to do at RESET. (6a-6b)

m_SerialNumber
Serial number in decimal. (6e-6f)

m_HardwareRevision[4]
Three character string describing hardware version. (70-73)

m_ProgrammedLogicRevision[4]
Three character string describing FPGA version. (74-77)

PortType Structure

```
typedef struct {
    USHORT Port;
    ULONG Value;
} PortType;
```

Type of I/O Port Access.

Defined in: D:/ALPHIPCI/INCLUDE/DDALPHIP.H

Members

Port
Offset into AMCC registers to access.

Value
I/O Value Read/Write to Port.