Change History Page

Version 1, Revision 0, September 2000, Original document.
Version 1, Revision 1, November 2000, Modifications and additions to cover address the Windows 95/98 kernel driver.
Version 1, Revision 2, June 2001, Update on usage of UnmapDMA
Version 1, Revision 3, April 2002, Update Windows 2000 Kernel Driver
# Table of Contents

1 INTRODUCTION ........................................................................................................................................ 1
  1.1 SYSTEM OVERVIEW .......................................................................................................................... 1
  1.2 KERNEL SERVICES OVERVIEW ......................................................................................................... 2
  1.3 DOCUMENT OVERVIEW .................................................................................................................... 2

2 REFERENCED DOCUMENTS .................................................................................................................... 3

3 INSTALLATION ......................................................................................................................................... 4
  3.1 INSTALLING THE SOFTWARE TO WINDOWS NT .................................................................................. 4
  3.2 INSTALLING THE SOFTWARE TO WINDOWS 95/98/2000 .................................................................. 4
  3.3 SOURCE CODE .................................................................................................................................... 4
  3.4 BUILDTIME CONSIDERATIONS ........................................................................................................... 5
  3.5 RUNTIME CONSIDERATIONS .............................................................................................................. 5

4 API REFERENCE ....................................................................................................................................... 6
  4.1 OPEN .................................................................................................................................................. 6
  4.2 CLOSE ................................................................................................................................................ 7
  4.3 GETDRIVERVERSION ........................................................................................................................... 7
  4.4 MAPBASEREGISTER ............................................................................................................................ 8
  4.5 UNMAPBASEREGISTER ......................................................................................................................... 8
  4.6 MAPDMAMEMORY .............................................................................................................................. 10
  4.7 UNMAPDMAMEMORY .......................................................................................................................... 12
  4.8 GETCONFIGURATIONSPACE ............................................................................................................... 13
  4.9 SETCONFIGURATIONSPACE ................................................................................................................ 14
  4.10 REGISTERINTERRUPT ....................................................................................................................... 15
  4.11 UNREGISTERINTERRUPT .................................................................................................................... 17

5 TEST API .................................................................................................................................................... 19

6 NOTES ....................................................................................................................................................... 20
  6.1 ACRONYMS ..................................................................................................................................... 20
1 INTRODUCTION

This developer’s manual provides all the information necessary to programatically interface with the PCI-DP CY7C09449PV kernel driver services. Following the functional overview, it provides a description of each service and how to use it.

1.1 System Overview

The CY7C09449PV is one of the PCI interface controllers in the Cypress Semiconductor PCI-DP family. The CY7C09449PV provides a PCI master/target interface with direct connections to many popular microprocessors. It provides 128K bits of dual-port SRAM that is used as shared memory between the local microprocessor and the PCI bus. An I20 message unit, complete with message queues and interrupt capability, is also provided. The CY7C09449PV allows the designer to interface an application to the PCI bus in a straight-forward, inexpensive manner.

The CY7C09449PV is composed of number of shared resources that allow effective data movement between the local and the PCI bus. A primary resource within the CY7C09449PV is its 16K bytes dual port memory. This memory is interfaced to both the PCI bus and to a local microprocessor bus. This shared memory can be accessed as a target from both buses at the same for inter-process communication. From either the local or PCI bus the CY7C09449PV can be directed to become a PCI bus master to move data into or out of the internal shared memory as a direct memory access (DMA). The CY7C09449PV can DMA across the PCI bus any number of 32-bit words up to 16K bytes. It uses the full bursting capabilities of the PCI bus for maximum efficiency and can transfer data over the full 32-bit PCI address space.

Four First-In-First-Out (FIFO) storage elements provide another resource to the user. These are accessible from either the PCI bus or the local bus. When the I20 messaging unit functionality of the CY7C09449PV is to be used, the four FIFOs become part of the I20 message unit of the CY7C09449PV. The I20 messaging unit consists of the four FIFO and the I20 system interrupt registers. The shared memory of the CY7C09449PV may be used to store I20 message frame buffers while most of shared memory is still available for general purpose use. Efficient I20 messaging is realized when the local processor uses the CY7C09449PV direct access mechanism. It can be used to retrieve and post I20 messages. It is made very efficient using the CY7C09449PV PCI DMA controller to burst the message frames to other I20 agents.

Inter-process communication is supported by two resources of the CY7C09449PV: the mailbox registers and the arbitration flags. By writing to the mailbox registers, a method is available for the local processor to pass data while causing an interrupt to the host, and vice versa. This is enabled by the interrupt mask located in the CY7C09449PV Operations Registers. The arbitration flags are four pairs of bits that can be used to manage resource allocation and sharing between software and system processes.

The CY7C09449PV includes a interrupt controller. There are separate interrupt mask and command/status registers for the PCI bus and the local bus. The interrupt sources are DMA completion, mailbox, FIFO not empty, FIFO overflow, PCI master abort, PCI
target abort, and there is an external interrupt input pin. This interrupt controller is used to signal interrupts onto the PCI bus and the local bus. The CY7C09449PV interrupt does not perform the interrupt controller function for the PCI bus system. Standard PC PCI systems include this function; embedded systems may need to implement this function.

1.2 Kernel Services Overview

The kernel driver companion of the CY7C09449PV provides an application easy access to the services and attributes available from the PCI bus side of the board. These include interrupt handling for the mailboxes, FIFOs, and DMA. Also included are mapping services to gain access to the board’s operation registers, and the ability to read and write the board’s PCI configuration space.

1.3 Document Overview

This document is comprised of six sections summarized here.

Section 1 provides the introductory information about the PCI-DP CY7C09449PV board and its kernel driver services.

Section 2 refers you to other documents containing information relevant to the subject matter addressed in this document.

Section 3 covers the steps necessary to install the kernel and its accompanying components.

Section 4 is the Application Program Interface (API) reference for all of the services provided by the PCI-DP kernel driver.

Section 5 is a quick reference guide for the kernel driver test program.

Section 6 defines the acronyms and abbreviations used in this document.
2 REFERENCED DOCUMENTS

The following documents are recommended as sources for additional, relevant information.


3 INSTALLATION

This section details how to install the Windows 95/98, Windows NT, and Windows 2000 PCI-DP kernel drivers and their respective development environments. This includes Windows 95/98 and Windows NT open source code for the kernel drivers and the API used to access the drivers.

3.1 Installing the Software to Windows NT

The Windows NT PCI-DP kernel driver software comes in two formats. The first is the expanded version contained on a CD-ROM. The second is the compressed version stored in a ZIP file normally obtained from a Web site. If your copy is a ZIP file, you must first expand its contents, preserving the directory structure, to a predetermined directory before you follow the instructions below.

1. After installing the PCI-DP board, reboot your system.
2. From the Windows NT Control Panel, double click the ‘Add/Remove Programs’ icon.
3. You will see the ‘Add/Remove Program Properties’ dialog box. Click on the ‘Install’ button near the top.
4. You will see a dialog box labeled ‘Install Program From Floppy Disk or CD-ROM’. Place the CD-ROM in its appropriate drive and click ‘Next’.
5. The ‘Command line for installation program’ will read ‘x:\Install.exe’ (where ‘x’ is the appropriate drive letter for the CD-ROM). Click ‘Finish’ to initiate the installation, then follow the setup instructions.

3.2 Installing the Software to Windows 95/98/2000

The PCI-DP kernel driver software comes in two formats. The first is the expanded version contained on a CD-ROM. The second is the compressed version stored in a ZIP file normally obtained from a Web site. If your copy is a ZIP file, you must first expand its contents, preserving the directory structure, to a predetermined directory before you follow the instructions below.

1. After installing the PCI-DP board, reboot your system.
2. Windows 95/98 will automatically detect the PCI-DP board.
3. When Windows 95/98 asks for the driver source, place the CD-ROM in its appropriate drive and select CD-ROM as the source.
4. Follow the installation wizard through the process.

3.3 Source Code

The installation includes Windows 95/98 and Windows NT open source code for both the PCI-DP kernel services API and for the kernel driver itself. You may change either
product in order to tailor it to fit your specific requirements. However, for most applications you will find the existing driver suitable. Cypress Semiconductor assumes you are familiar with the corresponding development environments for these two products. It does not assume any responsibility for modifications made to either product.

The installer prompts you for a home directory in which to install the PCI-DP kernel driver products. The kernel development environments are located in the subdirectory ‘Kernel’. The API development environments are located in the ‘API’ subdirectory.

### 3.4 Buildtime Considerations

In order to compile and link successfully, follow these guidelines. All files mentioned are located in the ‘API’ subdirectory hierarchy.

1. Include the header file ‘PCIDP_IF.h’ in any routine that references a kernel service.
2. Include as part of the link process the file ‘PCIDP_IF.lib’. Although there are four .lib files located in the different development directories, you may select any one of them to link with.

### 3.5 Runtime Considerations

The only runtime consideration is proper placement of the PCI-DP kernel services API DLL file ‘PCIDP_IF.dll’. There are two options you may use to ensure proper placement. With the “set it and forget it” option, you copy the DLL to the Windows NT/2000 base directory (usually ‘WINNT’) under the subdirectory ‘system32’. Windows NT/2000 automatically looks here when searching for DLL files it needs to run an application. The installation package automatically places a copy in this directory for you. The alternative is to place the DLL file in the same directory as the application that utilizes the PCI-DP kernel services API. For Windows 95/98 the DLL may be placed in the ‘system’ subdirectory. The installation package automatically places a copy in this directory for you.
4 API REFERENCE

The PCI-DP kernel driver provides its services to a software application via an API. The API is a specification that describes in detail how the software application may request these services. The API is made up of a set of C language functions that may be invoked from C, C++, or assembly language. Each of the API services returns to the caller a Windows system status code.

Note the keywords ‘out’ and ‘in’ are for decoration only and not a required part of the syntax. They merely indicate the direction of the parameter. If a parameter is preceded by both (i.e., ‘in out’), it means a reference is passed in, and the value pointed to by the reference, may have been modified in return.

4.1 Open

The OpenPCIDP service establishes communication between the host PC and the PCIDP device. This is the first service your application will invoke. The service returns a handle that is required to invoke any of the other services. Based on the board number, it tries to establish communications with the appropriate PCI-DP board. If it is successful, it returns a valid handle to that board. Success or failure is determined by examining the value of the return status. This routine need only be called once per session per board (i.e., the handle is valid for the life of the application using these services).

4.1.1 Syntax

unsigned long OpenPCIDP(
    in unsigned long BoardNumber,
    out unsigned long* PCIDPHandle
);

4.1.2 Parameters

BoardNumber
    The number of the PCI-DP board you wish to open. Board numbering begins with zero.

PCIDPHandle
    A unique PCI-DP board identifier that is used by the other services to identify the board with which you wish to communicate.

4.1.3 Return Status

ERROR_SUCCESS (0)
    Successful completion.

ERROR_BAD_DEVICE (120)
    Invalid board number. No board exists corresponding to the designated board number.
4.1.4 See Also

Section 4.2 Close.

4.1.5 Example

In this example the code is responsible for finding all boards installed on the host PC. It quits once the ERROR_SUCCESS status is no longer detected.

```c
unsigned long Handle[8];
unsigned long BoardNumber = 0;
unsigned long Status = ERROR_SUCCESS;
while(Status == ERROR_SUCCESS && BoardNumber < 8) {
    Status = OpenPCIDP(BoardNumber, &Handle[BoardNumber]);
    BoardNumber ++;
}
```

4.2 Close

The kernel driver requires no close service.

4.3 GetDriverVersion

The GetDriverVersion provides the caller with the current version of the kernel driver.

4.3.1 Syntax

```c
unsigned long GetDriverVersion(
    in unsigned long PCIDPHandle,
    in unsigned long* Version
);
```

4.3.2 Parameters

PCIDPHandle

PCIDP board identifier obtained by calling OpenPCIDP.

Version

Current version of the kernel driver in a longword hex format as follows: YYYYMMDD.

4.3.3 Return Status

ERROR_SUCCESS (0)

Successful completion.

ERROR_INVALID_HANDLE (6)

The PCIDP handle is invalid.
4.3.4 See Also

OpenPCIDP.

4.3.5 Example

```c
unsigned long Handle; // obtained from PCIDPOpen
unsigned long Version;
Status = GetDriverVersion(
    Handle,
    &Version
);
```

4.4 MapBaseRegister

The **MapBaseRegister** service provides mapping to the physical address described by one of the board's PCI base registers. If the address space is memory, the service returns a pointer that can be used to reference the memory directly. If the address space is I/O space, the service returns the I/O address from the base register. The caller must pass this address to an IO access service routine in order to successfully read and write I/O space. You cannot use the address directly like you can with a memory space address.

4.4.1 Syntax

```c
unsigned long MapBaseRegister(
    in unsigned long PCIDPHandle,
    in unsigned long RegisterNumber,
    in out unsigned long* Length,
    out unsigned long** Address,
    out unsigned long* IOAddress,
    out unsigned long* IOSpace
);
```

4.4.2 Parameters

- **PCIDPHandle**
  
  PCIDP board identifier obtained by calling **OpenPCIDP**.

- **RegisterNumber**
  
  PCI configuration space base register. There are a total of six beginning with register zero. Currently, base register zero (0) contains the physical address to the PCI-DP board’s memory space.

- **Length**
  
  Length in bytes of the memory size to map. Returns in bytes the size actually mapped.

- **Address**
  
  Linear address the calling application will use as an unsigned long pointer to the mapped memory space. This parameter is only valid when **IOSpace** is equal to zero (0).
IOAddress
I/O Address. This parameter is only valid when IOSpace is equal to one (1). You must use this address with an IO access service routine to correctly read or write I/O space.

IOSpace
Indicates the type of space mapped by this service. 1 = I/O space, 0 = memory space.

4.4.3 Return Status
ERROR_SUCCESS (0)
Successful completion.

ERROR_ACCESS_DENIED (5)
The mapping request failed.

ERROR_INVALID_HANDLE (6)
The PCIDP handle is invalid.

ERROR_INVALID_PARAMETER (87)
The base register number is out of range. Valid range is between 0 and 5.

ERROR_UNEXP_NET_ERR (59)
The value in the base register is zero, meaning it is unused.

4.4.4 See Also
OpenPCIDP, UnMapBaseRegister.

4.4.5 Example
In this example the code maps to the PCI-DP board’s memory space, including shared memory, then reads the Host Control register (HCTL), and finally reads the first longword of shared memory. The byte offset (or index) for HCTL is 0x04E0. Since the Address parameter points to longwords we need to recalculate this index by dividing by 4. The same is true for the offset to shared memory. Note how Address is defined and how it is passed in the call.

```c
unsigned long Handle;  //obtained from PCIDPOpen
unsigned long Length = 0x8000
unsigned long* Address;
unsigned long IOAddress;
unsigned long IOSpace;
unsigned long Status;
unsigned long HCTL;
unsigned long SM;

Status = MapBaseRegister(
    Handle,
    0,
    &Length,
    &Address,
    &IOAddress,
    &IOSpace
);

if(Status == ERROR_SUCCESS && IOSpace == 0){
```


HCTL = Address[0x04E0/4];  //convert to a longword index
SM = Address[0x4000/4];    //convert to a longword index
}

4.5 **UnMapBaseRegister**

The **UnMapBaseRegister** service is used to un-map from memory space you previously mapped to using **MapBaseRegister** when its parameter IOSpace equaled zero (0). There is no need to un-map from I/O space.

4.5.1 Syntax

```c
unsigned long UnMapBaseRegister(
    in unsigned long PCIDPHandle,
    in unsigned long* Address
);
```

4.5.2 Parameters

- **PCIDPHandle**: PCIDP board identifier obtained by calling **OpenPCIDP**.
- **Address**: Linear address obtained from a previous call to **MapBaseRegister** when its parameter IOSpace equaled zero (0).

4.5.3 Return Status

- **ERROR_SUCCESS (0)**: Successful completion.
- **ERROR_INVALID_HANDLE (6)**: The PCIDP handle is invalid.

4.5.4 See Also

**OpenPCIDP, MapBaseRegister**.

4.5.5 Example

In this example the code un-maps from the PCI-DP board’s memory space, including shared memory. The **Address** was obtained from the example in Section 4.4.5. Note how **Address** is defined and how it is passed in the call.

```c
unsigned long Handle;  //obtained from PCIDPOpen
unsigned long* Address; //obtained from MapBaseRegister
Status = UnMapBaseRegister(
    Handle,
    Address
);
```

4.6 **MapDMAMemory**

The **MapDMAMemory** service returns a pointer that can be used by the application to reference directly the DMA memory space. A physical address describing the DMA
absolute space is also returned and is used by the PCI-DP board during DMA transfers. The DMA memory space is 16K bytes of contiguous memory, located on the host PC, allocated for PCI-DP DMA transactions between shared memory and the host. There is no limit on the number of times one application (with or without multiple threads) or multiple applications may map to this memory space. It is your responsibility to coordinate when and how each entity accesses this space. When you are finished with DMA activity and before you exit the application, you should un-map each instance of DMA mapping using the \texttt{UnMapDMA\_Memory} service.

4.6.1 Syntax

\begin{verbatim}
unsigned long MapDMAMemory(
    in unsigned long PCIDPHandle,
    out unsigned long** LinearAddress,
    out unsigned long* PhysicalAddress
);
\end{verbatim}

4.6.2 Parameters

\begin{itemize}
  \item \textbf{PCIDPHandle}
    PCIDP board identifier obtained by calling \texttt{OpenPCIDP}.
  \item \textbf{LinearAddress}
    Linear address the calling application will use as an unsigned long pointer to the contiguous DMA memory space.
  \item \textbf{AbsoluteAddress}
    Absolute address in the host PC memory where the contiguous DMA memory space is located. This parameter is required by the PCI-DP board to perform DMA transactions.
\end{itemize}

4.6.3 Return Status

\begin{itemize}
  \item \textbf{ERROR\_SUCCESS} (0)
    Successful completion.
  \item \textbf{ERROR\_INVALID\_HANDLE} (6)
    The PCIDP handle is invalid.
  \item \textbf{ERROR\_OUTOFMEMORY} (14)
    Contiguous mapping failed.
\end{itemize}

4.6.4 See Also

OpenPCIDP, UnMapDMA\_Memory, RegisterInterrupt.

4.6.5 Example

In this example the code maps to 16K of allocated and contiguous host PC memory for the purpose of executing DMA transactions with the PCI-DP board. Second, it reads from the first longword and writes to the second longword of that DMA memory. Note that since \texttt{LinearAddress} is a longword pointer, one increment of the index represents 4 bytes. Note how \texttt{LinearAddress} and \texttt{PhysicalAddress} are defined and how they passed in the call. For an example of setting up and
executing a DMA transaction, see the example for the RegisterInterrupt service in Section 4.10.5.

```c
unsigned long Handle;  //obtained from PCIDPOpen
unsigned long* LinearAddress;
unsigned long PhysicalAddress;
unsigned long Status;
unsigned long DMAValue;

Status = MapDMAMemory(
    Handle,
    &LinearAddress,
    &PhysicalAddress
);

if(Status == ERROR_SUCCESS){
    DMAValue = LinearAddress[0];    //reads from first four bytes
    LinearAddress[1] = DMAValue+1;  //writes to second four bytes
}
```

### 4.7 UnMapDMAMemory

The **UnMapDMAMemory** service is used to un-map the DMA memory space you mapped to using **MapDMAMemory**. You should call this service for each instance of DMA mapping you perform within an application.

#### 4.7.1 Syntax

```c
unsigned long UnMapDMAMemory(
    in unsigned long PCIDPHandle,
    in unsigned long* LinearAddress
);
```

#### 4.7.2 Parameters

- **PCIDPHandle**
  - PCIDP board identifier obtained by calling **OpenPCIDP**.
- **LinearAddress**
  - Linear address obtained from a previous call to **MapDMAMemory**.

#### 4.7.3 Return Status

- **ERROR_SUCCESS (0)**
  - Successful completion.
- **ERROR_INVALID_HANDLE (6)**
  - The PCIDP handle is invalid.

#### 4.7.4 See Also

- **OpenPCIDP**, **MapDMAMemory**.

#### 4.7.5 Example

In this example the code un-maps from the host PC contiguous memory space allocated for DMA transactions. The **LinearAddress** was obtained from the
example in Section 4.6.5. Note how LinearAddress is defined and how it is passed in the call.

```c
unsigned long Handle;  //obtained from PCIDPOpen
unsigned long* LinearAddress;  //obtained from MapDMAAddress
Status = UnMapDMAAddress(
    Handle,
    Address
);
```

### 4.8 GetConfigurationSpace

The **GetConfigurationSpace** service returns the current PCI configuration data for the PCI-DP board.

#### 4.8.1 Syntax

```c
unsigned long GetPCIConfigurationSpace(
    in unsigned long PCIDPHandle,
    out PCIDP_CONFIG* PCIConfiguration
);
```

#### 4.8.2 Parameters

**PCIDPHandle**
- PCIDP board identifier obtained by calling **OpenPCIDP**.

**PCIConfiguration**
- A structure of 48 bytes that contain the PCI configuration data defined as follows:

```c
typedef struct _tagPCIDP_CONFIG {
    unsigned short VendorID;
    unsigned short DeviceID;
    unsigned short Command;
    unsigned short Status;
    unsigned char RevisionID;
    unsigned char ProgIF;
    unsigned char SubClass;
    unsigned char BaseClass;
    unsigned char CacheLineSize;
    unsigned char LatencyTimer;
    unsigned char HeaderType;
    unsigned char BIST;
    unsigned long BaseAddresses[6];
    unsigned long CIS;
    unsigned short SubVendorID;
    unsigned short SubSystemID;
    unsigned long ROMBaseAddress;
    unsigned long Reserved[2];
    unsigned char InterruptLine;
    unsigned char InterruptPin;
    unsigned char MIN_GNT;
    unsigned char MAX_LAT;
} PCIDP_CONFIG;
```
4.8.3 Return Status

ERROR_SUCCESS (0)
Successful completion.

ERROR_INVALID_HANDLE (6)
The PCIDP handle is invalid.

4.8.4 See Also

OpenPCIDP, SetPCIConfigurationSpace.

4.8.5 Example

In this example the code fetches the PCI configuration data for the PCI-DP board.

```c
unsigned long Handle; //obtained from PCIDPOpen
PCIDP_CONFIG PCIConfiguration;
Status = GetConfigurationSpace(
    Handle,
    &PCIConfiguration
);
```

4.9 SetConfigurationSpace

The SetConfigurationSpace service updates the PCI configuration data for the PCI-DP board. For that reason it’s best to populate the PCIConfiguration parameter with a call to SetConfigurationSpace, modify the affected configuration data fields, then call SetConfigurationSpace.

4.9.1 Syntax

```c
unsigned long SetPCIConfigurationSpace(
    in unsigned long PCIDPHandle,
    out PCIDP_CONFIG* PCIConfiguration
);
```

4.9.2 Parameters

PCIDPHandle
PCIIDP board identifier obtained by calling OpenPCIDP.

PCIConfiguration
A structure of 48 bytes that contain the PCI configuration data defined in Section 4.8.2.

4.9.3 Return Status

ERROR_SUCCESS (0)
Successful completion.

ERROR_INVALID_HANDLE (6)
The PCIDP handle is invalid.
4.9.4 See Also

OpenPCIDP, GetPCIConfigurationSpace.

4.9.5 Example

In this example the code sets the Command field in a PCICOnfiguration parameter previously obtained from the example in Section 4.8.5, then calls SetPCIConfigurationSpace to update the configuration data of the PCI-DP board.

```c
unsigned long Handle;    //obtained from PCIDPOpen
PCIDP_CONFIG PCICOnfiguration;  //from GetPCIConfigurationSpace
PCICOnfiguration.Command = 7;
Status = SetConfigurationSpace(Handle,
     &PCICOnfiguration
);
```

### 4.10 RegisterInterrupt

The RegisterInterrupt service sets up an interrupt handler that will signal a system event each time the referenced interrupt fires. You must manually reset the system event every time before waiting for it to signal. You may work with the system event as an individual event using the WaitForSingleObject system service or combine it with other system events using the WaitForMultipleObjects system service. When you are through, make sure to unregister the interrupt via the UnRegisterInterrupt service. The same type of interrupt may be registered by more than one thread in one application or across multiple applications. For this reason, each registration is unique and each must be unregistered as a unique and separate action.

#### 4.10.1 Syntax

```c
unsigned long RegisterInterrupt(
    in unsigned long PCIDPHandle,
    in unsigned long InterruptType,
    out void** InterruptID,
    out void** InterruptEvent
);
```

#### 4.10.2 Parameters

- **PCIDPHandle**
  
  PCIDP board identifier obtained by calling OpenPCIDP.

- **InterruptType**

  The PCIDP board interrupt type to enable. Following are the valid interrupt types:
PCIDP_PCIMasterAbort
PCIDP_PCIInterruptAbort
PCIDP_I2OOutboundPostFIFONotEmpty
PCIDP_DMAComplete
PCIDP_LocalToHostExternalSignal
PCIDP_LocalToHostMailbox
PCIDP_I2OPCIFIFOOverflow

**InterruptID**

The handle that identifies this unique interrupt request. It is required in order to properly un-register this interrupt request. Refer to UnRegisterInterrupt for more information.

**InterruptEvent**

The handle of the system event that will signal each time the interrupt fires. You must reset this handle after every wait. See Section 4.10.5 for an example of how this works.

4.10.3 Return Status

ERROR_SUCCESS (0)
Successful completion.

ERROR_INVALID_HANDLE (6)
The PCIDP handle is invalid.

ERROR_INVALID_PARAMETER (87)
The interrupt type is invalid.

ERROR_MR_MID_NOT_FOUND (317)
The interrupt function is not enabled on the board.

4.10.4 See Also

OpenPCIDP, UnRegisterInterrupt.

4.10.5 Example

The code in this example uses an interrupt mechanism to perform a host to local DMA transfer using the PCIDP_DMAComplete interrupt type. Data will be transferred across the PCI bus from the host PC DMA memory to the PCI-DP board shared memory. This example assumes that data has already been written to DMA memory and the operation registers are mapped. Note the length of the transfer is always the desired length in bytes minus 4. Note the index values for OpsAddr which are byte addresses divided by 4 to get a longword index. Note the use of the variable CopsAddress which is a character pointer to the operation registers memory space. Make sure to reset the interrupt event directly after the wait operation, otherwise you will not receive any more signaled events.

```c
unsigned long Handle; //obtained from PCIDPOpen
unsigned long* PhyAddr; //obtained from MapDMA Memory
unsigned long* OpsAddr; //obtained from MapBaseRegister
unsigned char* CopsAddr = (unsigned char*)OpsAddr;
void* InterruptID;
void* InterruptEvent;
```
Status = RegisterInterrupt(
    Handle,
    PCIDP_DMAComplete,
    &InterruptID,
    &InterruptEvent
);
if(Status == ERROR_SUCCESS){
    OpsAddr[0x04B0/4] = 0;  //offset into shared memory
    OpsAddr[0x04B4/4] = PhyAddr;  //physical address of DMA memory
    OpsAddr[0x04B8/4] = 0x3FFC  //length to transfer (0x4000)
    CopsAddr[0x04BC] = 0;  //start DMA transfer (host to local)
    // Use the WaitForSingleObject system service with a 1 second
    // timeout.
    WaitStatus = WaitForSingleObject(InterruptEvt, 1000);
    ResetEvent(InterruptEvt);
    if(WaitStatus == WAIT_TIMEOUT){
        printf("TIMEOUT\n");
    }
}

4.11 UnRegisterInterrupt

The UnRegisterInterrupt service will remove the interrupt handler installed via RegisterInterrupt and the system event InterruptEvent associated with it. Be sure to call this service when you no longer need to monitor the referenced interrupt.

4.11.1 Syntax

    unsigned long UnRegisterInterrupt(
        in unsigned long PCIDPHandle,  
        in void* InterruptID
    );

4.11.2 Parameters

PCIDPHandle
    PCIDP board identifier obtained by calling OpenPCIDP.

InterruptID
    The handle that identifies this unique interrupt request obtained from a previous call to the RegisterInterrupt service.

4.11.3 Return Status

   ERROR_SUCCESS (0)
      Successful completion.
   ERROR_INVALID_HANDLE (6)
      The PCIDP handle is invalid.
4.11.4 See Also

OpenPCIDP, RegisterInterrupt.

4.11.5 Example

In this example the code uses UnRegisterInterrupt service to cancel the interrupt handler established by a previous call to RegisterInterrupt.

```c
unsigned long Handle;  // obtained from PCIDPOpen
tvoid* InterruptID;     // obtained from RegisterInterrupt

Status = UnRegisterInterrupt(
    Handle,
    InterruptID
);
```
5 TEST API

The TestPCIDP application is a convenient way to confirm working communications between the host PC host and one or more PCI-DP Development boards. It exercises each of the services available from the kernel driver API, ensuring all services and the board(s) are working correctly. TestPCIDP can be found in the ‘API\TestPCIDP’ subdirectory of the software installation home directory. Because it is a console application, you must invoke it via an MS-DOS command line in order to review its output. Following is the sequence of events TestPCIDP follows to test the board(s) running under Windows NT. This is an example of a session that PASSED.

The test program found one board, number 0. The first error, Map base register 2 fails because the PCI-DP board does not support map base registers 2 through 5. The second error, Map base register 11 fails because there are only six base registers: 0 – 5. Error 3, Register Interrupt ‘Unknown’ fails because this is an invalid interrupt type. The remaining two tests exercise the interrupt capability of the kernel driver and the board. You will note the DMA transfer moves 161,280,000 bytes in just under 2 seconds.
6  NOTES

6.1  Acronyms

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>API</td>
<td>Application Program Interface</td>
</tr>
<tr>
<td>CD-ROM</td>
<td>Compact Disc Read Only Memory</td>
</tr>
<tr>
<td>DLL</td>
<td>Dynamic Link Library</td>
</tr>
<tr>
<td>DMA</td>
<td>Direct Memory Access</td>
</tr>
<tr>
<td>FIFO</td>
<td>First In First Out</td>
</tr>
<tr>
<td>FTP</td>
<td>File Transfer Protocol</td>
</tr>
<tr>
<td>PC</td>
<td>Personal Computer</td>
</tr>
<tr>
<td>PCI</td>
<td>Peripheral Communications Interface</td>
</tr>
<tr>
<td>PCI-DP</td>
<td>Trademark for that family of PCI interface controllers</td>
</tr>
<tr>
<td>SRAM</td>
<td>Static Random Access Memory</td>
</tr>
</tbody>
</table>