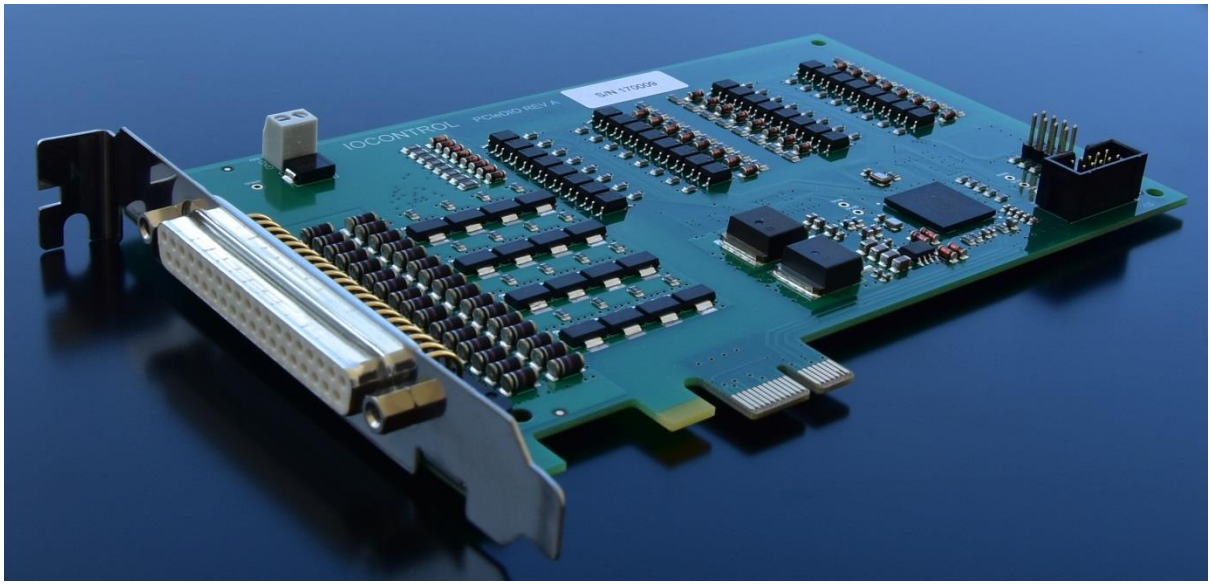
PCIeDIO User Manual

Edition 1.0



IOControl
Email: info@iocontrol.de
Web: www.iocontrol.de

Bruder Electronics Lda
Presa de Moura Lote 29
8400-008 Estômbar
Portugal

All relevant issues have been considered in the preparation of this document. Should you notice an omission or any questionable item in this document, please feel free to notify Bruder Electronics. Regardless of the foregoing statement, Bruder Electronics takes no responsibility for any errors that may appear in this document or for results obtained by the user as a result of using this product. The information in this document is subject to change without notice.

Trademarks and tradenames

IOControl is a tradename of Bruder Electronics Lda. Microsoft and Windows are trademarks of Microsoft Corporation. Other brand and product names are trademarks of their respective holder.

Table of Contents

Document Revisions

Date	Edition	Technical changes
17.06.2017	1.0	None
14.08.2016	Preliminary	

1. General information

The PCIeDIO is a PCI Express x1 add-in card capable of measuring digital inputs and controlling resistive, inductive and capacitive loads in 24V DC industrial applications.

The card provides 32 digital, opto-isolated I/Os operating at 24V DC. Each I/O can either be used as an input or as an output with read-back facility. This versatility makes for multiple configurations; whether an application demands the usage of 16 inputs and 16 outputs, or 24 inputs and 8 outputs, or 3 inputs and 29 outputs, or most extreme, just 32 inputs or 32 outputs, the PCIeDIO fits. Furthermore, as up to 8 cards can be used simultaneously in one PC system, 256 digital I/Os can be operated quasi-parallel.

The card uses latest FPGA technology not only to implement the PCI Express Bridge, but also to use embedded intelligence for providing many more convenient features like an on-board FIFO or programmable interrupts without imposing any load on the PC's CPU. A comprehensive, but easy-to-use software package supports the board on multiple versions and architectures of Windows.

Features

- ✓ 32 digital I/Os optimised for 24V DC, optically isolated from the computer
- ✓ Each I/O freely usable either as an input or as an output with read-back facility
- ✓ Isolation voltage min. 3750Vrms

Outputs

- ✓ High-side Power Switches capable of driving up to 0.65 A per channel (max. 6 A in total for all outputs) and definite power-up/reset at 0V
- ✓ Short circuit protection and recognition, current limitation and thermal shutdown with restart and programmable computer-independent watchdog
- ✓ Direct connection of all types of resistive, capacitive or inductive loads
- ✓ Driver for electromagnetic relays

Inputs

- ✓ Switching threshold optimised for 24V DC and input current approx. 3.69 mA at 24V DC
- ✓ All input signals usable as interrupt events with programmable interrupt triggering edges
- ✓ On-board 8192 * 32 FIFO capable of autonomously recording 8192 samples of all inputs (and outputs with the read-back facility) at a programmable rate
- ✓ Input signals can be applied even if the card is not powered internally and/or externally

Other features

- ✓ Programmable 24 bit, 10 MHz timer with interrupt operation
- ✓ 3 address jumpers for distinguishing multiple cards in the same system, 1 user jumper
- ✓ PCI Express x1 standard height, half-length add-in card compliant to PCI Express Base Specification Rev. 1.1, which can be used in any x1, x4, x8 or x16 PCI Express slot
- ✓ Connection of the 32 I/Os and external power supply via 37-pin DSUB female socket
- ✓ Optional Terminal Block Interface Module PCIeDIOTB available for easy sensor/actuator cabling
- ✓ Comprehensive software support for Windows 10 (32bit/64bit), Windows 8.1/8 (32bit/64bit), Windows 7 (32bit/64bit), Vista (32bit/64bit) and XP (SP3, 32 bit)
- ✓ RoHS 2 compliant according to directive 2011/65/EU

2. Installation

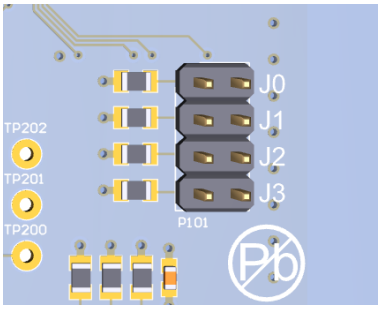
The installation of the card requires a free x1, x4, x8 or x16 PCI Express slot. Be sure to handle the board carefully and avoid any mechanical stress on it. Standard ESD safety precautions must be taken while handling the board, e.g. a wrist strap with earth cable should be used throughout the entire installation.
All external connections to the board should only be made or removed in a powered down state of all its associated components.

2.1 Address jumper settings for running multiple cards

If there is just one card to be operated in a system, no address jumper settings have to be done and this section can be skipped.

If there are multiple cards to be operated in a system, the address jumper settings have to be done properly, as the software driver distinguishes multiple cards in a system by means of these jumpers.

There are 3 address jumpers on the board labelled as J0, J1 and J2 beside the respective jumper (J3 is the user's jumper), which binary code the index of a card according to the table below.



Jumper settings (*)			Index of the board
J2	**J1**	**J0**	
0	0	0	0
0	0	1	1
0	1	0	2
0	1	1	3
1	0	0	4
1	0	1	5
1	1	0	6
1	1	1	7

(*) 0 = no jumper inserted, 1 = jumper inserted

There is no need to use a special numbering sequence, i.e. you can set up any configuration as long as the settings for each card are different. Just be sure to set up different settings for each card as the software driver will refuse to operate if there are cards with identical indices.

2.2 Installation of the hardware

Turn off the computer properly through the operating system. You should additionally disconnect the power cord to avoid any damage to the components of the computer, as independent of the powered off state voltage may be present on the system board as long as the system is plugged into an active AC outlet.

Follow the instructions of your PC manufacturer on how to install a PCI Express expansion board.

After the insertion of the card, press firmly on the card so that the whole PCI Express connector seats properly in the expansion card slot. Make sure that the PCIeDIO is either screwed to the computer casing via the mounting bracket of the card or is otherwise in a secured position, e.g. by means of a slot cover retention latch, etc. Afterwards, reconnect the power cord and turn on the computer.

2.3 Installation of the kernel-mode driver

After the hardware installation of the card has been done, install the driver package using the Windows Device Manager.

Step 1: How to open the Device Manager in different Windows versions

In **Window 10**, select **Settings** from the Start Menu and then **Devices**, then choose the **Device Manager**.

In **Windows 8**, either select the **Control Panel** from the Start Menu or select the **PC Settings** link from the Apps Menu, and then open the **Control Panel**. Depending on if you have selected view by icons or not, either click **System and Maintenance** and then choose **System,** or choose **System** directly if you have selected view by category**,** then click the **Device Manager** button.

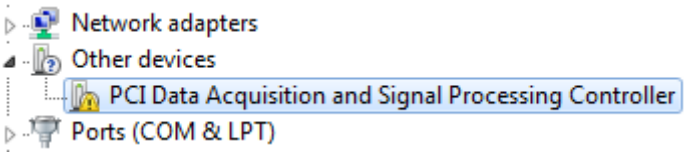
In **Windows 7** and **Windows Vista**, select the **Control Panel** from the Start Menu. Depending on if you have selected view by icons or not, either click **System and Maintenance** and then choose **System,** or choose **System** directly if you have selected view by category**,** then click the **Device Manager** button.

In **Windows XP (SP3)**, select the **Control Panel** from the Start Menu, then click **System**, choose the **Hardware** tab, and then click the **Device Manager** button.

> If you're comfortable with the command-line in Windows, specifically Command Prompt, a really quick and convenient way to start the Device Manager in any supported version of Windows is using the run command “devmgmt.msc”. Just open the Command Prompt, then enter “devmgmt.msc” and press the ENTER button.

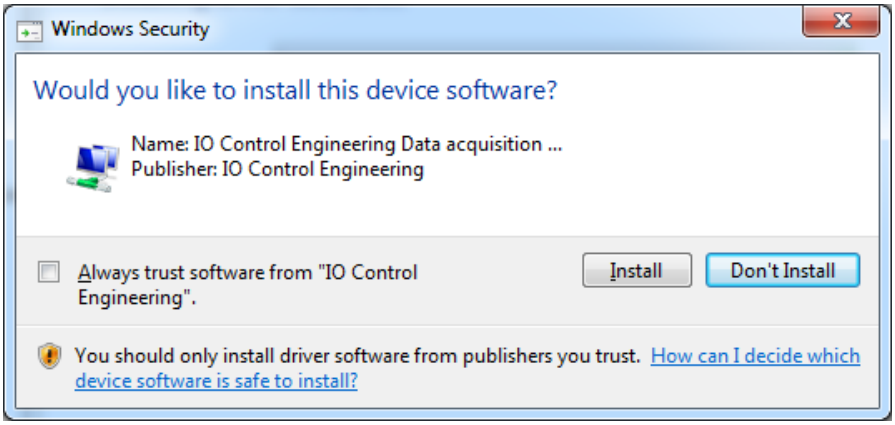
Step 2: Install the card using the Device Manager

With the **Device Manager** now open, double-click on **PCI Data Acquisition and Signal Processing Controller**

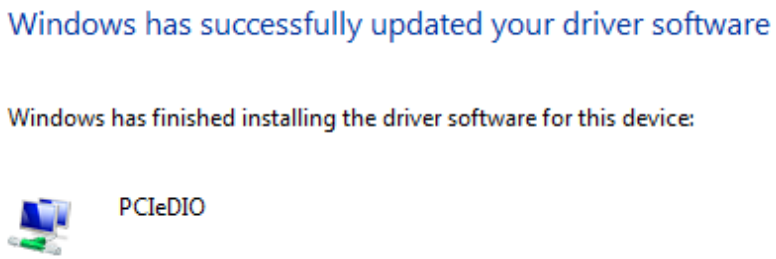


and then select **Update Driver** from the **General tab**. Browse then to your installation medium, select the install folder and then either choose the folder WIN_10 or WIN_8_7_Vista_XP depending on your operation system.

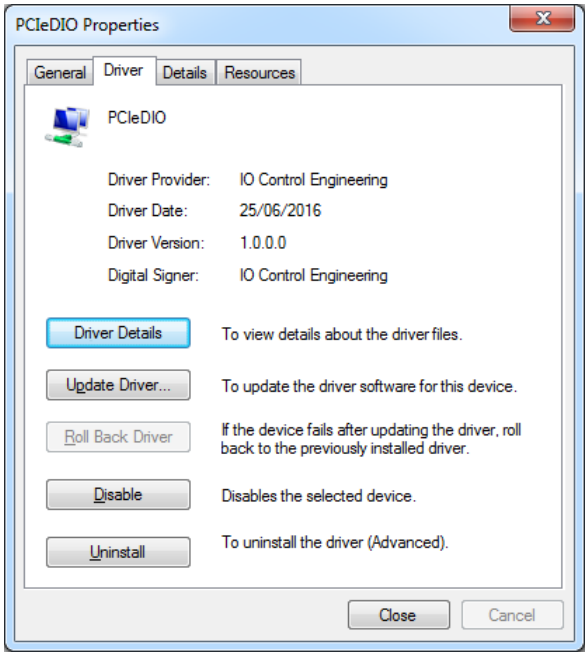
Click **Next**, and then, with the exception of Windows XP, a new window will appear showing a Windows Security message.

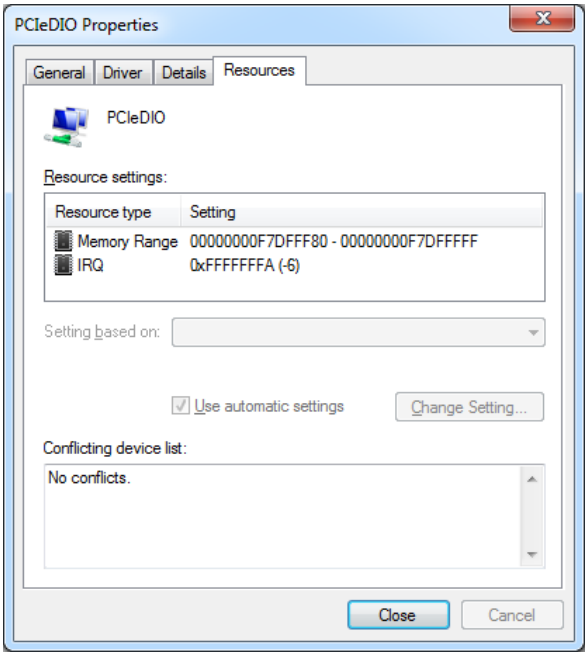


Independent of whether you choose to select 'Always trust software from "IO Control Engineering"' or not, after you have pressed the **Install** button, the kernel-mode driver will install automatically and you will see a similar message as below.



If you close the message above, you can then check the driver settings.





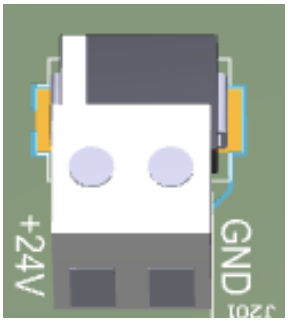
2.4 External power supply

For operating I/Os as outputs, the card has to be externally powered by 24V DC (+/- 30%) to supply the High-Side Power Switches.

If no I/O is to be operated as an output, an external power supply is not required, but a proper ground connection has to be provided as a reference potential to operate I/Os as inputs.

The power rating of the power supply depends primarily on how much current has to be driven simultaneously by all I/Os operated as outputs, which must not exceed 6 A.

The connection of the external power supply to the board can be done via the DSUB connector described in the following chapter and/or the screw terminal block described below via the pins labelled on the board as GND and +24V.

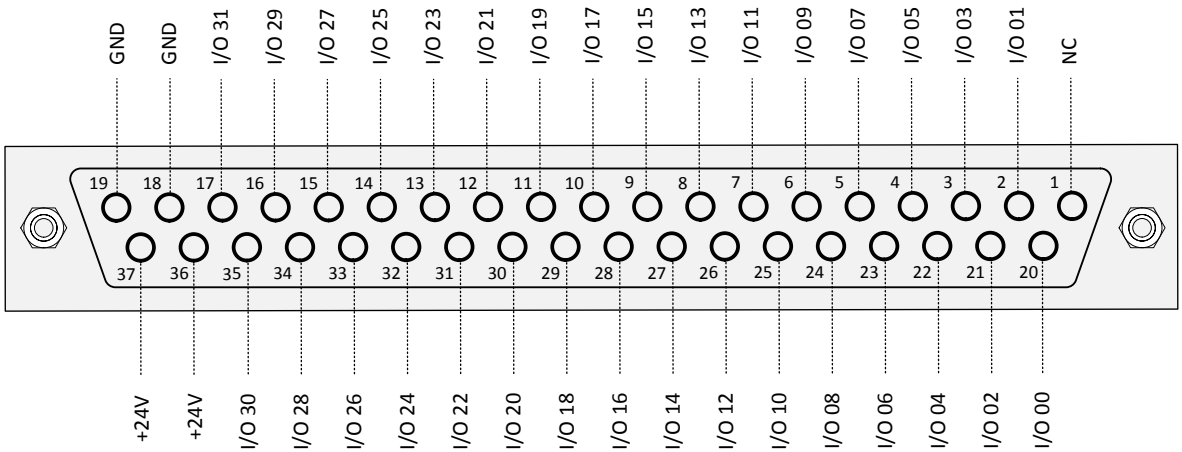


The screw terminal block has a rated current of 6 A. For wiring, we recommend using wires of 0.75 mm^2 (max. 1.0 mm^2). Connect the pin labelled as GND to the ground (0V) of the external power supply, and the pin labelled as +24V to +24V DC.

Connecting the external power supply to both the DSUB socket and the terminal screw block does not increase the maximum current of 6 A the board can handle.

2.5 Pin Assignment 37-pin DSUB Socket

The 32 I/Os, labelled as I/O 00 to I/O 31, are accessible on the 37-pin DSUB female socket of the card. Likewise, the external power supply can be connected to the pins GND and +24V.



Description	Pin	Description	Pin
GND	18 (*)	+24V	36 (**)
GND	19 (*)	+24V	37 (**)
I/O 00	20	I/O 16	28
I/O 01	2	I/O 17	10
I/O 02	21	I/O 18	29
I/O 03	3	I/O 19	11
I/O 04	22	I/O 20	30
I/O 05	4	I/O 21	12
I/O 06	23	I/O 22	31
I/O 07	5	I/O 23	13
I/O 08	24	I/O 24	32
I/O 09	6	I/O 25	14
I/O 10	25	I/O 26	33
I/O 11	7	I/O 27	15
I/O 12	26	I/O 28	34
I/O 13	8	I/O 29	16
I/O 14	27	I/O 30	35
I/O 15	9	I/O 31	17

(*) Both GND pins are internally interconnected and also to the GND pin of the screw terminal block
(**) Both +24V pins are internally interconnected and also to the +24V pin of the screw terminal block

Connect the pins labelled as GND in the table above to the ground (0V) of the external power supply, and the pins labelled as +24V to +24V DC.

If not more than 3 A is to be driven by the I/Os operated as outputs and appropriate wires are used, it is sufficient to connect only one GND pin (either pin 18 or pin 19) and one +24V pin (either pin 36 or pin 37) of the DUSB socket to the external power supply.

If more than 3 A is to be driven by the I/Os operated as outputs, either all power pins of the DSUB socket and appropriate wires must be used and/or the screw terminal block must be used.

Connecting the external power supply to both the DSUB socket and terminal screw block does not increase the maximum current of 6 A the board can handle.

If no I/O is to be operated as an output, but any I/O is to be operated as an input, a proper ground connection to at least one of the pins labelled as GND in the table above and/or to the GND of the terminal screw block has to be provided as a reference potential.

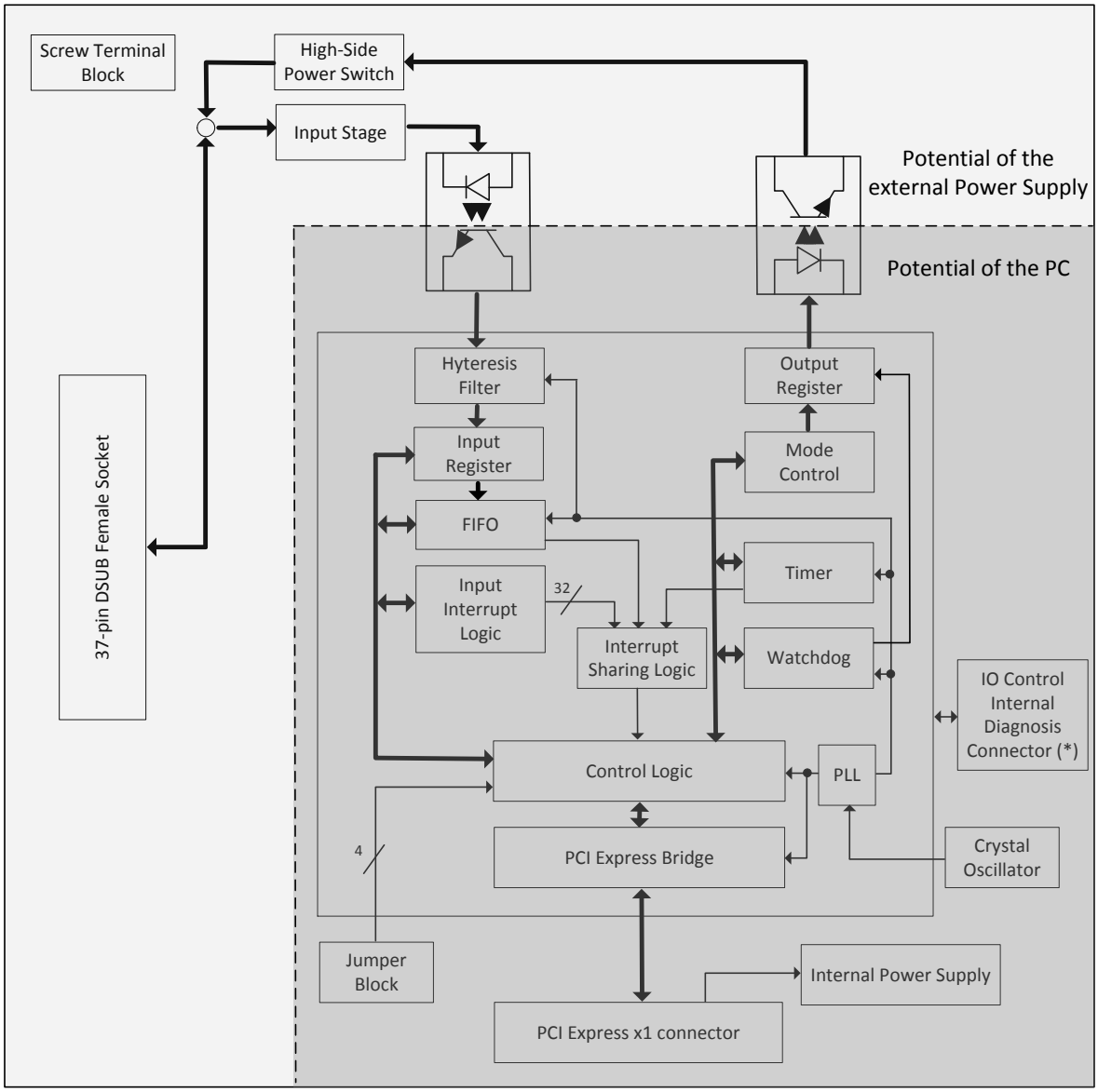
Unused pins may be left open, and any external sources can be left connected when the card is not powered by the PC and/or the external power supply.

3. Overview

3.1 Functional block diagram

The functional block diagram below provides a simplified overview of the layout of the PCIeDIO and the potential-separation between the PC and the I/Os available on the DSUB.

The light-grey shaded area marks the functionalities based on the potential of the external power supply, and the darker shaded area marks the units powered by the PC itself.



(*) The 2*5 low profile header is only for diagnostic purposes used by IO Control during the manufacturing of the card and must by no means be connected by the user

3.2 Mode/Output Enable

By default, each I/O is enabled to be operated as an input or an output, i.e. each I/O can be read as an input and each I/O can be set to a high level.

To prevent an application from accidentally setting an I/O which is used as an input to a high level and hence to protect the user's hardware, each single I/O can be individually configured to be used as an input-only I/O. If an I/O is configured as an input-only I/O, the respective I/O cannot be set to a high level by means of the DLL output functions and therefore will not drive a high level.

3.3 Interrupts

The PCIeDIO supports programmable interrupts, which can be issued by the on-board timer, the on-board FIFO and all 32 I/Os. All sources can be independently programmed and enabled to issue interrupts. Likewise, the interrupt mechanism can be globally enabled and disabled. The DLL provides all necessary functions to set up the interrupt configuration as required by the application.

The PCIeDIO supports both MSI (Message Signaled Interrupts) and Legacy Interrupts. Legacy Interrupts are required as Windows XP does not support MSI interrupts, and are also demanded by the PCI Express Specification.
Starting with Windows Vista, MSI interrupts are supported by all Windows operating systems and the preferred method. During the installation of the card, the correct setup is automatically chosen according to the operation system and proper settings are done.

3.4 Reset

After a hardware reset of the PC, all internal registers of the card are set to their default values, i.e. the timer is stopped and its settings cleared, the contents and settings of the FIFO are cleared, the watchdog of the outputs is disabled, all interrupt sources are disabled locally and globally, all I/Os not driven by an input signal are at a low level, etc. The individual default states are documented in detail in the appendix API Reference.

This default state can as well be initiated by a DLL function, which can be used in the event of an emergency, during the shutdown of the user's application, or just to get the card in a well-defined default state.

3.5 Timer

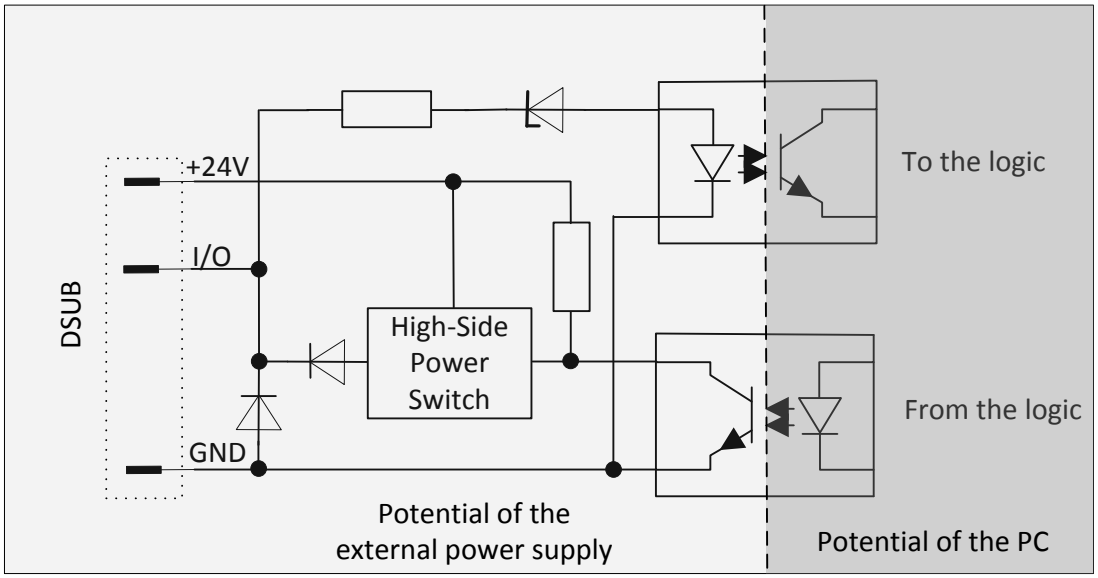
The PCIeDIO features a 24-bit on-board timer for the cyclic generation of interrupts, which can be programmed from 200ns to 1.6777216s in 100ns steps.

The timer uses an on-board computer-independent crystal oscillator. The accuracy of the 10MHz base frequency of the timer is +/- 50ppm, which equals +/-5ps each increment of 100ns. The maximum deviation is therefore +/- 5ps * (interval/100ns).

Please note that the handling of the interrupt by the driver and the operating system adds some unavoidable additional overhead from the time the timer issues an interrupt until the interrupt handler or callback function can recognize it.

4. Operating the I/Os

The diagram below illustrates one I/O in a simplified representation. Each of the 32 I/Os is composed in the same way, and all I/Os share the common external power supply, which is optically isolated from the PC as indicated in the different shadings.



4.1 Operating I/Os as outputs

Each I/O which is not set up as an input-only I/O via the mode/output enable functionality can be used as an output and set independently of one another either to a high level or to a low level.

By writing to the output register using the corresponding functions provided by the DLL, the respective optocoupler, which separates the potential between the output circuit and the PC, controls the output level of the High-Side Power Switch associated with that I/O.

Each I/O used as an output can be read back via its corresponding input channel and therefore also be used for short-circuit recognition.
If the I/O operated as an output is set to a high level and the reading of the corresponding I/O does not return a high level, a short-circuit is present. The switching characteristics of both the output circuit and the input stage have to be taken into account when reading back an output.

4.1.1 Protection functions

Each output circuit features a diode in series to protect its output against reverse currents. It also features a freewheeling recovery diode making resistive, inductive and capacitive loads directly connectable.

Due to the serial protection diode and the on-state resistance of the output circuit, there is a voltage drop at each output, which varies with the load.

The output circuit itself features permanent short-circuit protection, internal current limitation, overload protection and overvoltage protection. In case of an overload or short-circuit, an internal temperature monitoring circuit automatically switches the output of the High-Side Power Switch off, and switches it back on when its junction temperature falls below the thermal overload trip temperature.

4.1.2 Watchdog

The PCIeDIO features a computer-independent hardware-watchdog. When the watchdog is enabled, the watchdog resets all I/Os not used in input-only mode to their default low level if no I/O used as an output is accessed with a successful write instruction within a programmable timeout period. The timeout period can be programmed from 26.2144ms to 6.684672s in 26.2144ms steps with an accuracy of +/- 50ppm.

Each time the watchdog timer elapses, it resets all I/Os which are used as outputs to their default low level. The watchdog timer will not time out if at least one I/O is successfully written within the programmed timeout interval as this automatically restarts the watchdog timer with the programmed watchdog timer interval.

Once enabled, the watchdog will only be disabled and the timeout period made settable again if either the reset functionality of the card is used or a hardware reset of the PC takes place.

4.2 Operating I/Os as inputs

Each I/O can always can be read independently of whether an I/O is operated as an input-only I/O or operated as an output. If an I/O is read, the present state of the I/O pin, either a low or a high level, will be returned.

Besides simply reading the level of an I/O, each I/O can be configured individually to issue interrupts. Each I/O interrupt can be locally enabled or disabled, as well as the interrupt triggering edge, either a low to high transition or a high to low transition, can be individually programmed.

4.2.1 Input circuit

The input stage of each I/O is composed of a serial resistance and a Zener diode. Together with the following optocoupler, which separates the potential between the input circuit and the PC, these parts determine the switching characteristics of the input stage.

Additionally, there is a hysteresis filter implemented in the embedded logic to compensate the switching characteristics of the optocouplers in combination with the following digital evaluation and to prevent any false triggering.

4.2.2 FIFO

The PCIeDIO features a powerful on-board FIFO, which can be used to sample the input levels of all I/Os at a programmable rate. The FIFO is completely implemented in hardware and therefore does not impose any load on the PC's CPU as it operates autonomously.

The FIFO is capable of storing 8192 samples. Each sample contains the levels of all 32 I/Os. The sampling rate can be programmed between 20us to 0.65536s in 10us steps with an accuracy of +/- 50ppm.

Furthermore, an alarm level can be programmed, which means that if a programmable number of samples is stored in the FIFO, an interrupt can be issued.

The FIFO records the levels of all I/Os independent of whether an I/O is used as an input-only I/O or as an output. Care must be taken when evaluating its contents due to the switching characteristics. Especially if an I/O is used as an output, there are two switching characteristics to be taken into account: The switching characteristics of the output circuit and the switching characteristics of its respective input stage.

5. Specifications

5.1 Overview

Physical dimensions (without mounting bracket and connectors)

Height of board	115.15mm / 4.376 inches (standard height)
Length of board	167.65mm / 6.600 inches (half length)

PCI Express Interface

Link width	x1 (can be operated in x1, x4, x8 or x16 PCI Express slots)
Bus specification	PCI Express Base Specification Rev. 1.1
Interrupt support	MSI Interrupts and Legacy Interrupts
Plug & Play	Fully supported
Hot-Plug / Hot-removal	Not supported
Power Consumption	+3.3V: max. 1.2W, +12V : max. 0.5W

Digital I/Os optimised for 24V DC

Number of channels	32 I/Os opto-isolated from the PC
Isolation voltage	3750Vrms (min.), AC, 1 minute, relative humidity ≤ 60%

External connectors

37-pin DSUB	Standard 37-pin DSUB female socket with female screwlocks UNC 4-40
Screw terminal block	M2 screws with test tightening torque/screw 0.3 Nm max. Conductors screw terminal max. section 1.0 mm^2
External power supply V_{EXT}	16.8V to 31.2V DC

Operating and storage conditions

Operating temperature T_A	-20 to 70°C
Storage temperature T_{STG}	-40 to 105°C
Relative humidity	5 to 90% non-condensing

5.2 I/Os operated as inputs

Conditions: T_A = -20 to 70°C				
Characteristics	**Symbol**	**Min**	**Typ.**	**Max**
High-level input voltage	V_{IH}	16.8V	-	31.2V
Low-level input voltage	V_{IL}	0V	-	8.85V
Input resistance	R_{IN}	-	3900 Ω	-
High-level Input current	I_{IN}	1.84mA	3.69mA	5.64mA
Low to High propagation delay	t_{PLH} (*)	-	17us	25us
High to Low propagation delay (incl. storage time)	t_{PHL} (*)	40us	95us	150us
Maximum input frequency (Conditions: duty-cycle 50%)	f_{MAX} (*)	5KHz	-	10KHz

(*) All given data consider the whole input path from the I/O-pin of the DSUB until the valid state is digitally ready to be read by the software or otherwise being internally processed

5.3 I/Os operated as outputs

Conditions: T_A = -20 to 70°C, U_{EXT} = 16.8V to 31.2V				
Characteristics	**Symbol**	**Min**	**Typ.**	**Max**
High-level output voltage (Conditions: I_L <= 0.5A at resistive load, duty-cycle = 100%)	V_{OH}	0.9 x U_{EXT}	0.95 x U_{EXT}	-
Low-level output voltage	V_{OL}	-	-	0V
Max. continuous load current	$I_{L(Max)}$	0.5A	0.6A	0.65A
Repetitive short circuit current limit	$I_{L(SCr)}$	-	0.7A	-
Initial peak short circuit current limit	$I_{L(SCp)}$	-	-	1.2A
Turn-on time (Conditions: R_L=50Ω, V_{OL} to 90% V_{OH})	t_{ON}	-	30us	60us
Turn-off time (Conditions: R_L=50Ω, V_{OH} to 10% V_{OH})	t_{OFF}	-	95us	180us
Inductive load switch-off energy dissipation	E_{AS}	-	5mJ	-

Appendix API Reference

The PCIeDIO provides a comprehensive but easy-to-use software package to support the board on multiple versions of Windows.

Due to the introduction of a new kernel-mode driver signing policy in connection with the release of Windows 10, the installation directory of the driver package contains two directories. The contents differ only regarding the digitally-signed catalog files and the software publishing certificates required during the installation of the driver; the binaries of the driver for the same architecture, 32-bit or 64-bit, are identical for all the supported Windows editions below.

Install directory	**Supported OS**
WIN_10	Windows 10 (32 bit and 64 bit)
WIN_8_7_Vista_XP	Windows 8 / 8.1 (32 bit and 64 bit) Windows 7 (32 bit and 64 bit) Windows Vista (32 bit and 64 bit) Windows XP (SP3, 32 bit)

During the installation of the driver using the Windows Device Manager, the respective directory, either WIN_10 or WIN_8_7_Vista_XP, must be selected conditional on the target system.
Depending on the architecture of the selected target system, either the files from the subdirectory x86 (for 32-bit Windows) or from x64 (for 64-bit Windows) will then be autonomously selected.

The API directory contains the required files to build 32-bit and 64-bit Windows applications.

The DLL pciedioNTx86.dll and its corresponding library pciedioNTx86.lib must be used when 32-bit applications are to be programmed. This is independent of whether the architecture of Windows is 32-bit or if Win32 user-mode applications are to be run on 64-bit Windows (WOW64), as the 64-bit kernel-mode driver supports 32-bit applications as well.

The DLL pciedioNTx64.dll and its corresponding library pciedioNTx64.lib must be used if 64-bit applications are to be programmed.
The interface of both DLLs, as well as their source code, is identical in all architectures.

The header file pciedio.h located in the API directory is independent of the Windows version and architecture, so the file is identical in all architectures. It provides all the function prototypes of the API and some useful constants.

Finally, the Samples directory contains some convenient samples in source code. All samples are ready to be run on all supported Windows versions from XP to Windows 10 in 32-bit and 64-bit architectures.

API internals

The kernel-mode driver pciedio.sys is implemented as a KMDF (WDF) driver; the user-mode DLL is implemented in C/C++.

The API uses the extern “C” modifier and thereby makes the API accessible to executable modules written in C, C++, and other programming languages with multiple tool vendors.

Parameters and return data types of all API functions use only the most basic data types to make for a versatile programming interface. The user’s interface of the API uses DWORD (equivalent to unsigned long – a 32-bit unsigned integer), WORD (equivalent to unsigned short – a 16-bit unsigned integer) and BYTE (equivalent to char – an 8-bit unsigned integer) as these data types are available in almost all programming languages. The return type of all API functions is LONG (equivalent to long – a 32-bit signed integer).

Any programming language which is capable of interfacing to a DLL, using the basic data types stated above and allowing variables, as well as functions if the board’s interrupt facilities shall be used, to be indirectly referenced should be able to build an application for the PCIeDIO card.

With the exception of those functions used to detect, install and uninstall the PCIeDIO card(s) in a system, all functions are ready to be operated in a multithreaded environment, i.e. the API functions can be called from different threads running on the same or different processors within an application or process. The built-in concurrency management of the API and the architecture of the kernel-mode driver guarantee that by no means any malfunction can occur.

If two API functions try to access the same board at the same time, only one of them will operate on the shared resource (e.g. a hardware access to the board) while the operation of the other function will have to wait until the former function has finished its operation on the shared resource; afterwards the latter function can be operated.
If two API functions try to access different boards at the same time, both can be safely operated in parallel if they are called in different threads, as in that case the functions share no resources.

Support and customization

If you have any questions, please feel free to contact us. As a product like the PCIeDIO is always only as good as the customers’ feedback, we also highly encourage you to send us your feedback, positive or negative.

We also love customization and are capable of modifying the hardware and/or software of the PCIeDIO to some extent, so please feel free to present us your needs and we will see what we can do.

A.1 Initialization

Before a board can be accessed, some initializations and information are required.

`PciedioGetNumberOfCards` can be used to detect how many PCIeDIO boards are present in a system without doing any further initializations or allocating resources.

`PciedioOpenCards` not only detects all present PCIeDIO boards in the system, it also initializes all of them and allocates required resources. After a successful call of this function, all present boards are completely accessible. `PciedioCloseCards`, in contrast, is the function which releases all allocated resources.

All boards are addressed by a distinct index, which is determined by the jumper settings of each board. `PcidioGetIndices` returns the information at which index, or jumper settings, a board is present. If you have just one board running, you will not usually use any jumpers and the index would therefore be 0. But you can of course use any jumper settings even if you are running only one board as long as the jumper settings of multiple boards in a system are different; the API handles this appropriately.

If you would like to receive additional information about the board's configuration, you can call `PciedioGetRevisions` to receive the revision of the PCB itself, the FPGA revision, the kernel-mode driver revision and the revision of the DLL.

`PciedioGetPciConfiguration` returns the most important settings of a board's PCI configuration. `PciedioGetAddressRange` returns just the basic address information which is required to associate a particular board in the Windows Device Manager to its index, or jumper settings, if there is more than one board present.

`PciedioGetPresentJumperSettings` returns the present settings of the three address jumpers as well as of the user's jumper.

`PciedioResetCard` makes for resetting a particular board to its default state. This function is extremely convenient if you would like to get a board to its default state, e.g. before closing your application or in case of an emergency. The function directly triggers the hardware reset of the specified board.

`PciedioOpenIrq` is required if you wish to use a particular board's interrupt facilities. It allocates the required resources, creates a thread and installs the user's interrupt service routine. There is no function like 'PciedioCloseIrq', as this is done automatically for all boards that use interrupts when `PciedioCloseCards` is called.

`PciedioEnableGlobalIrq` enables an interrupt of a particular board globally, `PciedioDisableGlobalIrq` disables it globally. All interrupt sources have additional local enable and disable functions, but without enabling the interrupt functionality of a board globally, no interrupt will ever be issued.

PciedioGetNumberOfCards

Description

This function returns the number of PCIeDIO boards present in the system.

As the API supports up to 8 cards simultaneously within a single system, return values of 0 (no card is present) up to 8 (8 cards are present) are possible. If the operation fails, a negative value will be returned indicating an error.

Syntax

```
LONG PciedioGetNumberOfCards(void);
```

Parameters

None

Return value

0: the operation succeeded successfully but no card is present in the system
1: 1 card is present
2: 2 cards are present
3: 3 cards are present
4: 4 cards are present
5: 5 cards are present
6: 6 cards are present
7: 7 cards are present
8: 8 cards are present
-1: unable to receive the handle to the device information set from Windows
-2: unable to receive the interface details from Windows

Remarks

This function can be called prior to PciedioOpenCards

Requirements

DLL Version 1.0 or above

PciedioOpenCards

Description

This function detects and initializes all PCIeDIO cards in the system. It has to be called by the user's application before any other function of the API, with the exception of PciedioGetNumberOfCards, can successfully be called. While and after its operation, no other functions of the API must be performed.

After a successful call of PciedioOpenCards, all required resources are created for all detected cards and all cards are in their definite default state. Besides the additional interrupt functionalities, which have to be initialized separately for each board that shall use interrupts, all functions are ready to be operated.

This function returns the number of PCIeDIO cards present in the system. As the API supports up to 8 cards simultaneously within a single system, return values of 0 (no card is present) up to 8 (8 cards are present) are possible. If the operation fails, a negative value will be returned indicating an error.

Syntax

```
LONG PciedioOpenCards(void);
```

Parameters

None

Return value

0: the operation succeeded successfully but no card is present in the system
1: 1 card is present
2: 2 cards are present
3: 3 cards are present
4: 4 cards are present
5: 5 cards are present
6: 6 cards are present
7: 7 cards are present
8: 8 cards are present
-1: unable to receive the handle to the device information set from Windows
-2: unable to receive the interface details from Windows
-3: unable to create a required file object
-4: unable to create a required semaphore object
-5: jumper settings invalid (two or more cards share the same settings)
-6: the system is already initialized
-7: unable to initialize a card to its default state

Remarks

If the operation fails, all allocated resources by then will be properly released prior return

Requirements

DLL Version 1.0 or above

PciedioCloseCards

Description

This function releases any resources which have been allocated by PciedioOpenCards. While and after its operation, no other functions of the API must be performed.

Syntax

```
LONG PciedioCloseCards(void);
```

Parameters

None

Return value

0: the operation succeeded successfully
-1: error in releasing a resource allocated by PciedioOpenCards

Remarks

This function should be called right before the closing of the user's application to release Windows system resources

Requirements

DLL Version 1.0 or above, and a successful call of the function PciedioOpenCards

PciedioGetIndices

Description

This function provides an easy way to receive the indices of those boards which are present in the system. The indices are required to address a card adequately.
The index of a board corresponds directly to its binary coded jumper settings J2, J1 and J0. Up to 8 PCIeDIO boards can be distinguished by the jumper settings, and 8 PCIeDIO boards are supported by the driver simultaneously. PciedioGetIndices receives the information at which index a board was found and initialized by the function PciedioOpenCards.

Syntax

```
LONG PciedioGetIndices(
  BYTE *indices
);
```

Parameters

`indices [out]`
A pointer to an 8-bit variable. Each bit position of the variable relates to one index. If a bit position returns a '1', a card is present at the corresponding index; if a bit position returns a '0', no card is present at the corresponding index.

Bit position of `indices`	Jumper settings J2 J1 J0	Index of the board
0	0 0 0	0
1	0 0 1	1
2	0 1 0	2
3	0 1 1	3
4	1 0 0	4
5	1 0 1	5
6	1 1 0	6
7	1 1 1	7

Return value

0: the operation succeeded successfully
-1: no board exists in the system

Requirements

DLL Version 1.0 or above, and a successful call of the function PciedioOpenCards

Example

```
BYTE indices;

PciedioGetIndices(&indices);
if(indices & 0x01)    //check bit position 0
  printf("A board is present at index 0\n");
if(indices & 0x40)    //check bit position 6
  printf("A board is present at index 6\n");
```

PciedioGetRevisions

Description

This function receives the revisions of the components of a particular board.

Syntax

```
LONG PciedioGetRevisions(
  BYTE  cardnum,
  BYTE *revpcb,
  BYTE *revfpga,
  BYTE *revkernelhigh,
  BYTE *revkernellow,
  BYTE *revdllhigh,
  BYTE *revdlllow
);
```

Parameters

`cardnum [in]`
The index of the card corresponding to the jumper settings

`revpcb [out]`
A pointer to an 8-bit variable which receives the revision of the PCB

`revfpga [out]`
A pointer to an 8-bit variable which receives the revision of the FPGA

`revkernelhigh [out]`
A pointer to an 8-bit variable which receives the major revision of the kernel-mode device driver

`revkernellow [out]`
A pointer to an 8-bit variable which receives the minor revision of the kernel-mode device driver

`revdllhigh [out]`
A pointer to an 8-bit variable which receives the major revision of the PCIeDIO DLL

`revdlllow [out]`
A pointer to an 8-bit variable which receives the minor revision of the PCIeDIO DLL

Return value

0: the operation succeeded successfully
-1: the board addressed with `cardnum` does not exist or is not initialized

Requirements

DLL Version 1.0 or above, and a successful call of the function PciedioOpenCards

PciedioGetPciConfiguration

Description

This function receives the PCI Express configuration settings of a particular board. The information is read out of a board's PCI Express configuration address space.

Syntax

```
LONG PciedioGetPciConfiguration(
  BYTE                    cardnum,
  struct _PciedioConfig  *pciedioconfig
);
```

Parameters

`cardnum [in]`
The index of the card corresponding to the jumper settings

`pciedioconfig [out]`
A pointer to a structure of type `_PciedioConfig` which receives the PCI Express configuration settings

```
struct _PciedioConfig {
  WORD  vendor;
  WORD  device;
  WORD  subvendor;
  WORD  subsystem;
  BYTE  revision;
  BYTE  baseclass;
  BYTE  subclass;
  BYTE  headertype;
  DWORD address;
  DWORD range;
  BYTE  type;
  BYTE  interruptline;
};
```

with:

`vendor`
The Vendor ID of the card, which is set to 1172_{hex}

`device`
The Device ID of the card, which is set to 0004_{hex}

`subvendor`
The Subsystem Vendor ID of the card, which is set to 2406_{hex}

`subsystem`
The Subsystem ID of the card, which is set to 1507_{hex}

`revision`
The Revision ID of the card's PCI Express interface. The first revision is 1_{hex}

`baseclass`
The base class code of the card, which broadly classifies the type of function the PCIeDIO performs. The value is set to 11_{hex}

`subclass`
The sub-class code of the card, which identifies more specifically the function the PCIeDIO performs. The value is set to 80_{hex}

`headertype`
The Header Type of the card identifies the layout of the second part of the predefined header of the configuration space and is set to 0_{hex}

`address`
The base address of the card in the PC's address space

`range`
The address range used by the the card in bytes

`type`
The type of address space. The PCIeDIO uses the memory-mapped address space and `type` will always receive $2_{decimal}$.

`interruptline`
The Interrupt Line used by the the card. The value in the register is system architecture specific.

Return value

0: the operation succeeded successfully
-1: the board addressed with `cardnum` does not exist or is not initialized

Requirements

DLL Version 1.0 or above, and a successful call of the function PciedioOpenCards

Example

```
struct _PciedioConfig pciedioconfig;

//get the PCI Express configuration of board 0
PciedioGetPciConfiguration(0, &pciedioconfig);
```

PciedioGetAddressRange

Description

This function receives the address space and type used by a particular board. The information can be used to identify a respective board in the Windows Device Manger.

If several boards are running in a system, this function makes for the connection of a respective board identified by its jumper settings to the address range, which is shown in the Windows Device Manager.

Syntax

```
LONG PciedioGetAddressRange(
  BYTE   cardnum,
  DWORD *address,
  DWORD *range,
  BYTE  *type
);
```

Parameters

`cardnum [in]`
The index of the card corresponding to the jumper settings

`address [out]`
A pointer to a 32-bit variable which receives the base address of the board

`range [out]`
A pointer to a 32-bit variable which receives the address range used by the board in bytes

`type [out]`
A pointer to an 8-bit variable which receives the type of address space. The PCIeDIO uses the memory-mapped address space and `type` will always receive $2_{decimal}$

Return value

0: the operation succeeded successfully
-1: the board addressed with `cardnum` does not exist or is not initialized

Requirements

DLL Version 1.0 or above, and a successful call of the function PciedioOpenCards

Example

```
BYTE  cardnum;
DWORD address;
DWORD range;
BYTE  type;
//get the address range of board 2
PciedioGetAddressRange(2, &address, &range, &type);
```

PciedioGetPresentJumperSettings

Description

This function returns the present jumper settings of a particular board.

Syntax

```
LONG PciedioGetPresentJumperSettings(
  BYTE  cardnum,
  BYTE *addressjumpers,
  BYTE *userjumper
);
```

Parameters

`cardnum [in]`

The index of the card corresponding to the jumper settings

`addressjumpers [out]`

A pointer to an 8-bit variable that receives the state of the binary coded address jumpers J2, J1 and J0. If a jumper is inserted, a '1' will be returned in its respective bit position, otherwise a '0', so corresponding decimal values from 0 (no address jumpers are inserted) to 7 (all address jumpers are inserted) are possible.

`userjumper [out]`

A pointer to an 8-bit variable that receives the state of the user's jumper. If a jumper is inserted in J3, a '1' will be returned, otherwise a '0'

Return value

0: the operation succeeded successfully
-1: the board addressed with `cardnum` does not exist or is not initialized
-2: an internal Windows error occurred
-3: an IOCTL error occurred

Remarks

This function returns the present settings of the jumpers and not necessarily the state of the jumpers used during initialisation. Usually, both settings are the same as changing the jumper settings while the board is powered is not recommended.

Requirements

DLL Version 1.0 or above, and a successful call of the function PciedioOpenCards

Example

```
//read the jumper settings of board 0
BYTE address;
BYTE user;
PciedioGetPresentJumperSettings(0, &address, &user);
```

PciedioResetCard

Description

This function resets the hardware of a particular board to its default state, but it does not close the access to the card via the API functions or remove the user's interrupt service routine associated with that board.

After a successful call, the board has set up its default values for all internal hardware registers connected to that board; the hardware of the board is exactly in the same state as it is after a hardware reset of the system or a reboot. Initializations done with PciedioOpenCards or the setup of an optional user's interrupt service routine are not affected by this function as those are not directly related to the hardware of the board, but the hardware settings themselves to enable interrupts etc. have to be repeated.

Syntax

```
LONG PciedioResetCard(
  BYTE cardnum
);
```

Parameters

`cardnum [in]`
The index of the card corresponding to the jumper settings

Return value

0: the operation succeeded successfully
-1: the board addressed with `cardnum` does not exist or is not initialized
-2: an internal Windows error occurred
-3: an IOCTL error occurred

Remarks

This function is especially useful during the closing of the board or in case of an emergency

Requirements

DLL Version 1.0 or above, and a successful call of the function PciedioOpenCards

Example

```
//first do some hardware settings of board 0
PciedioSetIOByte(0, 0x03); //set IO00 and IO01 to a high level

//reset the hardware of board 0
PciedioResetCard(0);          //IO00 and IO01 will be at low level afterwards

//the card is accessible again as soon as PciedioResetCard has returned
PciedioSetIOByte(0, 0x03);//set IO00 and IO01 again to a high level
```

PciedioOpenIrq

Description

This function allocates the required interrupt resources for a particular board, creates the interrupt object and installs the user's interrupt service routine.

It does not enable any local interrupt sources of the board, nor does it enable the interrupt of the board globally.

Syntax

```
LONG PciedioOpenIrq(
  BYTE cardnum,
  void (__cdecl *hIsr)(BYTE  cardnum,
                       DWORD inputs,
                       BYTE  timer,
                       BYTE  fifo
                      )
);
```

Parameters

`cardnum [in]`
The index of the card corresponding to the jumper settings

`hIsr [in]`
A pointer to a function which has the four parameters `cardnum`, `inputs`, `timer` and `fifo` and no return value.

This function is the user's interrupt service routine, which will be called each time a properly enabled interrupt occurs. Each time it is called, the four arguments provide detailed information about the source of the interrupt:

`cardnum`
The index of the card corresponding to the jumper settings which issued the interrupt.

`inputs`
This variable acts as a capture register of the 32 I/O interrupt sources. If one or more I/Os have issued an interrupt, the respective bit positions of `inputs` carry a '1'. If an I/O has not issued an interrupt, its bit position carries a '0'.

Bit 0 relates to IO00, bit 1 to IO01,…, and bit 31 to IO31.

`timer`
If the timer has issued an interrupt, `timer` carries a '1', otherwise it carries a '0'.

`fifo`
If the FIFO has issued an interrupt, `fifo` carries a '1', otherwise it carries a '0'.

Return value

0: the operation succeeded successfully
-1: the board addressed with `cardnum` does not exist or is not initialized
-2: an internal Windows error occurred
-3: an IOCTL error occurred
-4: no user's interrupt service routine was provided
-5: unable to create the interrupt thread
-6: unable to change the priority class of the interrupt thread
-7: an user's interrupt servie routine is already installed for the particular board

Remarks

As the user's interrupt service routine receives in its argument `cardnum` notification which board has issued an interrupt, it is possible to use one interrupt service routine for several boards. If then several boards request interrupts simultaneously, multiple copies of the user's service routine will then be executed in parallel.

Requirements

DLL Version 1.0 or above, and a successful call of the function PciedioOpenCards

Example

```
//the most basic structure of an interrupt servie routine may look like this:
void __cdecl hIsrUser(BYTE  cardnum,
                      DWORD inputs,
                      BYTE  timer,
                      BYTE  fifo
                     )
{ //if several interrupt sources are enabled, they can of course
  //trigger in parallel, so always check all enabled sources
 if(inputs)
   {… //examine which input or inputs have triggered
      //do proper action as required + service the local interrupt using an API function
   }
 if(timer)
   { … //do proper action as required + service the local interrupt using an API function
   }
 if(fifo)
   { … //do proper action as required + service the local interrupt using an API function
   }
}


//allocate the required resources and install the user's interrupt service routine of
//board 0
PciedioOpenIrq(0, &hIsrUser));

//There is a detailed sample "InterruptHandling" in the
//Samples directory of the driver package, which provides a
//deeper insight into the handling of interrupts, and may
//be a good starting point for an own routine
```

PciedioEnableGlobalIrq

Description

This function enables the interrupt of a particular board globally. Without enabling interrupts globally, no interrupt will be issued by the particular board. By default, the global interrupt is disabled. The function does not enable any local interrupt sources of the board.

The interrupt object of the particular board has to be created with PciedioOpenIrq before PciedioEnableGlobalIrq can be called successfully.

Syntax

```
LONG PciedioEnableGlobalIrq(
  BYTE cardnum
);
```

Parameters

`cardnum [in]`
The index of the card corresponding to the jumper settings

Return value

0: the operation succeeded successfully
-1: the board addressed with `cardnum` does not exist or is not initialized
-2: an internal Windows error occurred
-3: an IOCTL error occurred
-4: an interrupt service routine was not set up for the specific card

Remarks

The call of the function would not lead to an error if the interrupt was not disabled before

Requirements

DLL Version 1.0 or above, a successful call of the function PciedioOpenCards and a successful call of the function PciedioOpenIrq of the particular board

Example

```
//enable the global interrupt of board 2
PciedioEnableGlobalIrq(2);
```

PciedioDisableGlobalIrq

Description

This function disables the interrupt of a particular board globally. By default, the global interrupt is disabled. The function does not disable any local interrupt sources of the board.

If during or after the call of PciedioDisableGlobalIrq an interrupt was already issued by the board and is still being on delivery to the user's interrupt service routine, this interrupt will still be delivered to the user's interrupt service routine.

As disabling the global interrupt does not rely on the existence of an interrupt object, this function can be called without the prior creation of the interrupt object with PciedioOpenIrq.

Syntax

```
LONG PciedioDisableGlobalIrq(
  BYTE cardnum
);
```

Parameters

`cardnum [in]`
The index of the card corresponding to the jumper settings

Return value

0: the operation succeeded successfully
-1: the board addressed with `cardnum` does not exist or is not initialized
-2: an internal Windows error occurred
-3: an IOCTL error occurred

Remarks

The call of the function would not lead to an error if the interrupt was not enabled before

Requirements

DLL Version 1.0 or above, and a successful call of the function PciedioOpenCards

Example

```
//disable the global interrupt of board 2
 PciedioDisableGlobalIrq(2);
```

A.2 Configuring I/Os as input-only I/Os

Each single I/O can be individually configured to be used as an input-only I/O. If an I/O is configured as an input-only I/O, the I/O will not drive a high level.

By default, all I/Os are enabled to be operated as input or output, which means that each I/O can be read as an input and each I/O can be set to a high level. But it's highly recommended to configure the I/Os which shall operate as an input as an input-only I/O.

This feature prevents an application from accidentally setting an I/O which is used as an input to a high level and hence protects the user's hardware. It is implemented in two stages: The first stage is a software protection, and the second stage is a pure hardware protection in the FPGA to implement the best security possible.

To configure the mode of multiple I/Os at once, `PciedioSetIOModeByte`, `PciedioSetIOModeWord` and `PciedioSetIOModeDWord` can be used. `PciedioSetIOModeBit` instead sets up the mode of one particular I/O.

The mode which is set up can be read by using the functions `PciedioGetIOModeByte`, `PciedioGetIOModeWord` and `PciedioGetIOModeDWord` for multiple I/Os at once. `PciedioGetIOModeBit` instead reads the mode of one particular I/O.

PciedioSetIOModeByte

Description

This function writes 8-bit data to the specified group of 8 mode bits corresponding to 8 I/Os of a particular board. An I/O can be used as an output if the corresponding bit position carries a '1'; if it carries a '0', the respective I/O is used as input-only I/O and will not drive a high level. By default, all I/Os are enabled as outputs.

Syntax

```
LONG PciedioSetIOModeByte(
  BYTE cardnum,
  BYTE group,
  BYTE val
);
```

Parameters

`cardnum [in]`

The index of the card corresponding to the jumper settings

`group [in]`

The group of 8 I/Os for which the mode bits have to be written

0 : 1st byte = IO07..IO00
1 : 2nd byte = IO15..IO08
2 : 3rd byte = IO23..IO16
3 : 4th byte = IO31..IO24

`val [in]`

The 8-bit data to be written to the stated group

Return value

0: the operation succeeded successfully
-1: the board addressed with `cardnum` does not exist or is not initialized
-2: an internal Windows error occurred
-3: an IOCTL error occurred
-4: the parameter `group` is out of the specified range

Remarks

The 32 mode bits are 1-bit, 8-bit, 16-bit and 32-bit addressable

Requirements

DLL Version 1.0 or above, and a successful call of the function PciedioOpenCards

Example

```
//write the second 8-bit group of mode bits of board 2 with data 3A_hex, so that IO13,
//IO12, IO11 and IO09 can be used as outputs and IO15, IO14, IO10 and IO08 are
//disabled as outputs
PciedioSetIOModeByte(2, 1, 0x3A);
```

PciedioSetIOModeWord

Description

This function writes 16-bit data to the specified group of 16 mode bits corresponding to 16 I/Os of a particular board. An I/O can be used as an output if the corresponding bit position carries a '1'; if it carries a '0', the respective I/O is used as input-only I/O and will not drive a high level. By default, all I/Os are enabled as outputs.

Syntax

```
LONG PciedioSetIOModeWord(
  BYTE cardnum,
  BYTE group,
  WORD val
);
```

Parameters

`cardnum [in]`

The index of the card corresponding to the jumper settings

`group [in]`

The group of 16 I/Os for which the mode bits have to be written
0 : 1st word = IO15..IO00
1 : 2nd word = IO31..IO16

`val [in]`

The 16-bit data to be written to the stated group

Return value

0: the operation succeeded successfully
-1: the board addressed with `cardnum` does not exist or is not initialized
-2: an internal Windows error occurred
-3: an IOCTL error occurred
-4: the parameter `group` is out of the specified range

Remarks

The 32 mode bits are 1-bit, 8-bit, 16-bit and 32-bit addressable

Requirements

DLL Version 1.0 or above, and a successful call of the function PciedioOpenCards

Example

```
//write the first 16-bit group of mode bits of board 1 with data 8112hex, so that IO15,
//IO08, IO04 and IO01 can be used as outputs and all other IOs of the first group are
//disabled as outputs
 PciedioSetIOModeWord(1, 0, 0x8112);
```

PciedioSetIOModeDWord

Description

This function writes 32-bit data to the 32 mode bits corresponding to 32 I/Os of a particular board. An I/O can be used as an output if the corresponding bit position carries a '1'; if it carries a '0', the respective I/O is used as input-only I/O and will not drive a high level. By default, all I/Os are enabled as outputs.

Syntax

```
LONG PciedioSetIOModeDWord(
  BYTE  cardnum,
  DWORD val
);
```

Parameters

`cardnum [in]`
The index of the card corresponding to the jumper settings

`val [in]`
The 32-bit data to be written

Return value

0: the operation succeeded successfully
-1: the board addressed with `cardnum` does not exist or is not initialized
-2: an internal Windows error occurred
-3: an IOCTL error occurred

Remarks

The 32 mode bits are 1-bit, 8-bit, 16-bit and 32-bit addressable

Requirements

DLL Version 1.0 or above, and a successful call of the function PciedioOpenCards

Example

```
//write all 32 IO mode bits of board 1 with data 13018204hex, so that IO28, IO25, IO24,
//IO16, IO15, IO09 and IO02 can be used as outputs and all other IOs are disabled as
//outputs
 PciedioSetIOModeDWord(1, 0x13018204);
```

PciedioSetIOModeBit

Description

This function writes the mode bit of an I/O channel of a particular board and enables or disables the dedicated I/O as an output. By default, all I/Os are enabled as outputs.

Syntax

```
LONG PciedioSetIOModeBit(
  BYTE cardnum,
  BYTE channel,
  BYTE state
);
```

Parameters

`cardnum [in]`
The index of the card corresponding to the jumper settings

`channel [in]`
The channel from 0 to 31, where 0 = IO00, 1 = IO01, 2 = IO02, … , 30 = IO30, and 31 = IO31

`state [in]`
The mode bit of the I/O to be set with
0: the I/O is disabled as an output and is used as an input-only I/O
1: the I/O can be uesd as an output

Return value

0: the operation succeeded successfully
-1: the board addressed with `cardnum` does not exist or is not initialized
-2: an internal Windows error occurred
-3: an IOCTL error occurred
-4: the parameter `channel` is out of the specified range
-5: the parameter `state` is out of the specified range

Remarks

The 32 mode bits are 1-bit, 8-bit, 16-bit and 32-bit addressable

Requirements

DLL Version 1.0 or above, and a successful call of the function PciedioOpenCards

Example

```
PciedioSetIOModeBit(0, 30, 1);//IO30 of board 0 can be used as an output
PciedioSetIOModeBit(2, 5, 0);//IO05 of board 2 is disabled as output
```

PciedioGetIOModeByte

Description

This function reads 8-bit data from the specified I/O group of 8 mode bits corresponding to 8 I/Os of a particular board. If an I/O is enabled as an output, the corresponding bit position will carry a '1'. If an I/O is disabled as an output and used as input-only I/O, the corresponding bit position will carry a '0'. By default, all I/Os are enabled as outputs.

Syntax

```
LONG PciedioGetIOModeByte(
  BYTE  cardnum,
  BYTE  group,
  BYTE *state
);
```

Parameters

`cardnum [in]`

The index of the card corresponding to the jumper settings

`group [in]`

The group of 8 I/Os for which the mode bits have to be read

0 : 1st byte = IO07..IO00
1 : 2nd byte = IO15..IO08
2 : 3rd byte = IO23..IO16
3 : 4th byte = IO31..IO24

`state [out]`

A pointer to an 8-bit variable that receives the state of the mode bits of the specified I/O group

Return value

0: the operation succeeded successfully
-1: the board addressed with `cardnum` does not exist or is not initialized
-2: an internal Windows error occurred
-3: an IOCTL error occurred
-4: the parameter `group` is out of the specified range

Remarks

The 32 mode bits are 1-bit, 8-bit, 16-bit and 32-bit addressable

Requirements

DLL Version 1.0 or above, and a successful call of the function PciedioOpenCards

Example

```
//read the fourth 8-bit group of mode bits of board 2 to the variable state
BYTE state;
PciedioGetIOModeByte(2, 3, &state);
```

PciedioGetIOModeWord

Description

This function reads 16-bit data from the specified I/O group of 16 mode bits corresponding to 16 I/Os of a particular board. If an I/O is enabled as an output, the corresponding bit position will carry a '1'. If an I/O is disabled as an output and used as input-only I/O, the corresponding bit position will carry a '0'. By default, all I/Os are enabled as outputs.

Syntax

```
LONG PciedioGetIOModeWord(
  BYTE  cardnum,
  BYTE  group,
  WORD *state
);
```

Parameters

`cardnum [in]`

The index of the card corresponding to the jumper settings

`group [in]`

The group of 16 I/Os for which the mode bits have to be read
0 : 1st word = IO15..IO00
1 : 2nd word = IO31..IO16

`state [out]`

A pointer to a 16-bit variable that receives the state of the mode bits of the specified I/O group

Return value

0: the operation succeeded successfully
-1: the board addressed with `cardnum` does not exist or is not initialized
-2: an internal Windows error occurred
-3: an IOCTL error occurred
-4: the parameter `group` is out of the specified range

Remarks

The 32 mode bits are 1-bit, 8-bit, 16-bit and 32-bit addressable

Requirements

DLL Version 1.0 or above, and a successful call of the function PciedioOpenCards

Example

```
//read the first 16-bit group of mode bits of board 1 to the variable state
WORD state;
PciedioGetIOModeWord(1, 0, &state);
```

PciedioGetIOModeDWord

Description

This function reads 32-bit data from the 32 mode bits corresponding to 32 I/Os of a particular board. If an I/O is enabled as an output, the corresponding bit position will carry a '1'. If an I/O is disabled as an output and used as input-only I/O, the corresponding bit position will carry a '0'. By default, all I/Os are enabled as outputs.

Syntax

```
LONG PciedioGetIOModeDWord(
  BYTE   cardnum,
  DWORD *state
);
```

Parameters

`cardnum [in]`
The index of the card corresponding to the jumper settings

`state [out]`
A pointer to a 32-bit variable that receives the state of the mode bits of the 32 I/Os

Return value

0: the operation succeeded successfully
-1: the board addressed with `cardnum` does not exist or is not initialized
-2: an internal Windows error occurred
-3: an IOCTL error occurred

Remarks

The 32 mode bits are 1-bit, 8-bit, 16-bit and 32-bit addressable

Requirements

DLL Version 1.0 or above, and a successful call of the function PciedioOpenCards

Example

```
//read the 32 mode bits of board 1 to the variable state
DWORD state;
PciedioGetIOModeDWord(1, &state);
```

PciedioGetIOModeBit

Description

This function reads the mode bit of an I/O channel of a particular board. By default, all I/Os are enabled as outputs.

Syntax

```
LONG PciedioGetIOModeBit(
  BYTE  cardnum,
  BYTE  channel,
  BYTE *state
);
```

Parameters

`cardnum [in]`

The index of the card corresponding to the jumper settings

`channel [in]`

The channel from 0 to 31, where 0 = IO00, 1 = IO01, 2 = IO02, … , 30 = IO30, and 31 = IO31

`state [out]`

A pointer to an 8-bit variable which receives the state of the mode bit of the I/O specified channel

0: the I/O is disabled as an output and used as input-only I/O

1: the I/O can be used as an output

Return value

0: the operation succeeded successfully

-1: the board addressed with `cardnum` does not exist or is not initialized

-2: an internal Windows error occurred

-3: an IOCTL error occurred

-4: the parameter `channel` is out of the specified range

Remarks

The 32 mode bits are 1-bit, 8-bit, 16-bit and 32-bit addressable

Requirements

DLL Version 1.0 or above, and a successful call of the function PciedioOpenCards

Example

```
//get the mode bit of IO30 of board 0
BYTE state;
PciedioGetIOModeBit(0, 30, &state);
```

A.3 Operating I/Os as outputs

Several functions are provided to operate I/Os as outputs, but I/Os which are used as input-only I/Os, must not be set to a high level. Additionally, the usage of the facility for setting up the operating mode of the I/Os is highly recommended.

To write multiple I/Os at once, `PciedioSetIOByte`, `PciedioSetIOWord` and `PciedioSetIODWord` can be used.
To set a particular I/O operated as an output to a high level, `PciedioSetIOBitHigh` can be called while `PciedioSetIOBitLow` sets the I/O to a low level. `PciedioSetIOBit` sets the level of an I/O either high or low according to a passed parameter.

`PciedioSetIOUpdate` is a function which writes all I/Os used as outputs again with the pattern that is currently set. The function is especially convenient to cyclically update the I/Os when the watchdog is used.

To set the desired watchdog timer inverval, `PciedioSetWatchdogInterval` must be uesd. The writing of a non-zero value not only sets the time-out time, it also starts the watchdog timer right afterwards.
`PciedioGetWatchdogSettings` returns the present watchdog timer interval and the information whether the watchdog timer is running or stopped.
`PciedioGetWatchdogState` queries if the watchdog timer has elapsed or timed out at least once since the last call of the function.

PciedioSetIOByte

Description

This function writes 8-bit data to the specified I/O group of 8 I/Os of a particular board. It sets an I/O to a high level if the corresponding bit position carries a '1' and the I/O is not configured as an input-only I/O, or to a low level if it carries a '0'. By default, all I/Os are set to low level.

Syntax

```
LONG PciedioSetIOByte(
  BYTE cardnum,
  BYTE group,
  BYTE val
);
```

Parameters

`cardnum [in]`

The index of the card corresponding to the jumper settings

`group [in]`

The group of 8 I/Os which has to be written

0 : 1st byte = IO07..IO00
1 : 2nd byte = IO15..IO08
2 : 3rd byte = IO23..IO16
3 : 4th byte = IO31..IO24

`val [in]`

The 8-bit data to be written to the stated group

Return value

0: the operation succeeded successfully
-1: the board addressed with `cardnum` does not exist or is not initialized
-2: an internal Windows error occurred
-3: an IOCTL error occurred
-4: the parameter `group` is out of the specified range
-5: at least one I/O which shall be set to a high level is configured as an input-only I/O

Remarks

The 32 I/Os are 1-bit, 8-bit, 16-bit and 32-bit addressable

Requirements

DLL Version 1.0 or above, and a successful call of the function PciedioOpenCards

Example

```
//write the second 8-bit group of board 2 with data 3A_hex, so that IO13, IO12, IO11 and
//IO09 will be set to a high level, and IO15, IO14, IO10 and IO08 to a low level
PciedioSetIOByte(2, 1, 0x3A);
```

PciedioSetIOWord

Description

This function writes 16-bit data to the specified I/O group of 16 I/Os of a particular board. It sets an I/O to a high level if the corresponding bit position carries a '1' and the I/O is not configured as an input-only I/O, or to a low level if it carries a '0'. By default, all I/Os are set to low level.

Syntax

```
LONG PciedioSetIOWord(
  BYTE cardnum,
  BYTE group,
  WORD val
);
```

Parameters

`cardnum [in]`

The index of the card corresponding to the jumper settings

`group [in]`

The group of 16 I/Os which has to be written
0 : 1st word = IO15..IO00
1 : 2nd word = IO31..IO16

`val [in]`

The 16-bit data to be written to the stated group

Return value

0: the operation succeeded successfully
-1: the board addressed with `cardnum` does not exist or is not initialized
-2: an internal Windows error occurred
-3: an IOCTL error occurred
-4: the parameter `group` is out of the specified range
-5: at least one I/O which shall be set to a high level is configured as an input-only I/O

Remarks

The 32 I/Os are 1-bit, 8-bit, 16-bit and 32-bit addressable

Requirements

DLL Version 1.0 or above, and a successful call of the function PciedioOpenCards

Example

```
//write the first 16-bit group of board 1 with data 8112hex, so that IO15, IO08, IO04 and
//IO01 will be set to a high level, and all other IOs of the first group to a low level
PciedioSetIOWord(1, 0, 0x8112);
```

PciedioSetIODWord

Description

This function writes 32-bit data to the I/Os of a particular board. It sets an I/O to a high level if the corresponding bit position carries a '1' and the I/O is not configured as an input-only I/O, or to a low level if it carries a '0'. By default, all I/Os are set to low level.

Syntax

```
LONG PciedioSetIODWord(
  BYTE  cardnum,
  DWORD val
);
```

Parameters

`cardnum [in]`
The index of the card corresponding to the jumper settings

`val [in]`
The 32-bit data to be written to the 32 I/Os

Return value

0: the operation succeeded successfully
-1: the board addressed with `cardnum` does not exist or is not initialized
-2: an internal Windows error occurred
-3: an IOCTL error occurred
-4: at least one I/O which shall be set to a high level is configured as an input-only I/O

Remarks

The 32 I/Os are 1-bit, 8-bit, 16-bit and 32-bit addressable

Requirements

DLL Version 1.0 or above, and a successful call of the function PciedioOpenCards

Example

```
//write all 32 IOs of board 0 with data 13018204hex, so that IO28, IO25, IO24, IO16,
//IO15, IO09 and IO02 will be set to a high level, and all other IOs to a low level
PciedioSetIODWord(1, 0x13018204);
```

PciedioSetIOBitHigh

Description

This function sets one specified I/O of a particular board to a high level if the I/O is not configured as an input-only I/O. By default, all I/Os are set to low level.

Syntax

```
LONG PciedioSetIOBitHigh(
  BYTE cardnum,
  BYTE channel
);
```

Parameters

`cardnum [in]`

The index of the card corresponding to the jumper settings

`channel [in]`

The channel from 0 to 31, where 0 = IO00, 1 = IO01, 2 = IO02, … , 30 = IO30, and 31 = IO31

Return value

0: the operation succeeded successfully
-1: the board addressed with `cardnum` does not exist or is not initialized
-2: an internal Windows error occurred
-3: an IOCTL error occurred
-4: the parameter `channel` is out of the specified range
-5: the I/O which shall be set to a high level is configured as an input-only I/O

Remarks

The 32 I/Os are 1-bit, 8-bit, 16-bit and 32-bit addressable

Requirements

DLL Version 1.0 or above, and a successful call of the function PciedioOpenCards

Example

```
PciedioSetIOBitHigh(0, 23); //set IO23 of board 0 to a high level
```

PciedioSetIOBitLow

Description

This function sets one specified I/O of a particular board to a low level. By default, all I/Os are set to low level.

Syntax

```
LONG PciedioSetIOBitLow(
  BYTE cardnum,
  BYTE channel
);
```

Parameters

`cardnum [in]`

The index of the card corresponding to the jumper settings

`channel [in]`

The channel from 0 to 31, where 0 = IO00, 1 = IO01, 2 = IO02, … , 30 = IO30, and 31 = IO31

Return value

0: the operation succeeded successfully
-1: the board addressed with `cardnum` does not exist or is not initialized
-2: an internal Windows error occurred
-3: an IOCTL error occurred
-4: the parameter `channel` is out of the specified range

Remarks

The 32 I/Os are 1-bit, 8-bit, 16-bit and 32-bit addressable

Requirements

DLL Version 1.0 or above, and a successful call of the function PciedioOpenCards

Example

```
PciedioSetIOBitLow(1, 18); //set IO18 of board 1 to a low level
```

PciedioSetIOBit

Description

This function sets one specified I/O of a particular board to the stated level. If an I/O is configured as an input-only I/O, this function refuses to set the stated I/O to a high level. By default, all I/Os are set to low level.

Syntax

```
LONG PciedioSetIOBit(
  BYTE cardnum,
  BYTE channel,
  BYTE state
);
```

Parameters

`cardnum [in]`
The index of the card corresponding to the jumper settings

`channel [in]`
The channel from 0 to 31, where 0 = IO00, 1 = IO01, 2 = IO02, … , 30 = IO30, and 31 = IO31

`state [in]`
The desired level of the I/O to be set with
0: low level
1: high level

Return value

0: the operation succeeded successfully
-1: the board addressed with `cardnum` does not exist or is not initialized
-2: an internal Windows error occurred
-3: an IOCTL error occurred
-4: the parameter `channel` is out of the specified range
-5: the parameter `state` is out of the specified range
-6: the I/O which shall be set to a high level is configured as an input-only I/O

Remarks

The 32 I/Os are 1-bit, 8-bit, 16-bit and 32-bit addressable

Requirements

DLL Version 1.0 or above, and a successful call of the function PciedioOpenCards

Example

```
PciedioSetIOBit(0, 30, 1); //set IO30 of board 0 to a high level
PciedioSetIOBit(2,  5, 0); //set IO05 of board 2 to a low level
```

PciedioSetIOUpdate

Description

This function updates all 32 I/Os of a particular board according to their last written levels. If the watchdog timer of the I/Os is used, the usage of this function is particularly convenient to prevent the watchdog from elapsing or 'timing out'.

Syntax

```
LONG PciedioSetIOUpdate(
  BYTE cardnum
);
```

Parameters

`cardnum [in]`
The index of the card corresponding to the jumper settings

Return value

0: the operation succeeded successfully
-1: the board addressed with `cardnum` does not exist or is not initialized
-2: an internal Windows error occurred
-3: an IOCTL error occurred

Remarks

The watchdog timer will never time out if at least one I/O of the associated board is written to a high or a low level within the programmed time out time.

Requirements

DLL Version 1.0 or above, and a successful call of the function PciedioOpenCards

Example

```
PciedioSetIOUpdate(2); //update the IOs of board 2
```

PciedioSetWatchdogInterval

Description

This function sets the watchdog timer interval of a particular board in increments of 26.2144 milliseconds and starts the timer right afterwards.

Each time the watchdog timer elapses, it resets all I/Os which are used as outputs to their default low level. The watchdog timer will not time out if at least one I/O of the associated board is successfully written within the programmed time-out interval as this automatically restarts the watchdog timer with the programmed watchdog timer interval.

The resultant watchdog timer interval can be calculated by
watchdog timer interval = `interval` * 26.2144ms
The required `interval` value for a desired watchdog timer interval can be calculated by
`interval` = (watchdog timer interval / 26.2144ms)

The pattern to be written to an I/O to service the watchdog timer does not matter at all; even the writing of a '0' to a single I/O used as input-only I/O services the watchdog timer. The writing of the output register itself is decisive, not the pattern or how many or which channels are written. Therefore the successful call of the functions PciedioSetIOByte, PciedioSetIOWord, PciedioSetIODWord, PciedioSetIOBit, PciedioSetIOBitHigh and PciedioSetIOBitLow all service the watchdog timer appropriately.

Once enabled (default is disabled), the watchdog timer cannot be stopped, nor can the watchdog timer interval be changed. Only a hardware reset of the PC, the API functions PciedioResetCard and PciedioOpenCards disable the watchdog timer and make it configurable again.

The watchdog timer and its protection mechanism itself are completely implemented in hardware and independent of the PC system clock. It can therefore be a powerful and reliable feature to prevent potential damage on the user's actors if the PC should hang or freeze for some reason.

Syntax

```
LONG PciedioSetWatchdogInterval(
  BYTE  cardnum,
  BYTE  interval
);
```

Parameters

`cardnum [in]`
The index of the card corresponding to the jumper settings

`interval [in]`
The 8-bit interval to be set where an interval of 0 must not be porgrammed. The effective programmable range of `interval` is therefore 1 to FF_{hex}

Return value

0: the operation succeeded successfully
-1: the board addressed with `cardnum` does not exist or is not initialized
-2: an internal Windows error occurred
-3: an IOCTL error occurred
-4: the parameter `interval` is 0 and therefore out of the specified range
-5: unable to set an interval because the watchdog timer has already been set up

Remarks

The API function PciedioSetIOUpdate can as well be used to cyclically service the watchdog timer

Requirements

DLL Version 1.0 or above, and a successful call of the function PciedioOpenCards

Example

```
//set interval of board 0 to 39decimal which corresponds to a watchdog timer interval
//of 1.022 seconds
PciedioSetWatchdogInterval(0, 39);
```

PciedioGetWatchdogSettings

Description

This function reads the settings of the watchdog timer of a particular board. It returns the information whether the watchdog timer is running or not, as well as the programmed interval of the watchdog timer.

Syntax

```
LONG PciedioGetWatchdogSettings(
  BYTE  cardnum,
  BYTE *interval,
  BYTE *state
);
```

Parameters

`cardnum [in]`
The index of the card corresponding to the jumper settings

`interval [out]`
A pointer to an 8-bit variable that receives the interval of the watchdog timer as described in `PciedioSetWatchdogInterval`

`state [out]`
A pointer to an 8-bit variable that receives the state of the watchdog timer. A '1' indicates that the watchdog timer is running, a '0' that it is not running

Return value

0: the operation succeeded successfully
-1: the board addressed with `cardnum` does not exist or is not initialized
-2: an internal Windows error occurred
-3: an IOCTL error occurred

Requirements

DLL Version 1.0 or above, and a successful call of the function PciedioOpenCards

Example

```
//read the settings of the watchdog timer of board 1
BYTE interval;
BYTE state;
PciedioGetWatchdogSettings(1, &interval, &state);
```

PciedioGetWatchdogState

Description

This function reads the state of the watchdog timer of a particular board. If the watchdog timer has at least once elapsed since the last call of this function, a '1' will be returned, otherwise a '0'.

Syntax

```
LONG PciedioGetWatchdogState(
  BYTE  cardnum,
  BYTE *state
);
```

Parameters

`cardnum [in]`
The index of the card corresponding to the jumper settings

`state [out]`
A pointer to an 8-bit variable that receives the state of the watchdog timer

Return value

0: the operation succeeded successfully
-1: the board addressed with `cardnum` does not exist or is not initialized
-2: an internal Windows error occurred
-3: an IOCTL error occurred

Requirements

DLL Version 1.0 or above, and a successful call of the function PciedioOpenCards

Example

```
//read the state of the watchdog timer of board 1 to the variable state
BYTE state;
PciedioGetWatchdogState(1, &state);
```

A.4 Operating I/Os as inputs

The following functions can be used independently of whether an I/O is operated as an output or operated as an input-only I/O. This is because of the read-back facility of the outputs: Each output is always likewise an input. It is therefore possible, not only to read back the outputs, but also to use the interrupt facilities of the inputs for the outputs.

Several functions are provided to read the present input level of an I/O. To read multiple I/Os at once, `PciedioGetIOByte`, `PciedioGetIOWord` and `PciedioGetIODWord` can be used. To read the level of a particular I/O, `PciedioGetIOBit` can be called.

Each I/O can issue interrupts, and for each I/O individual settings can be programmed. Interrupts of the I/Os are always edge-triggered, either from high level to low level or from low level to high level, and to set up the triggering edges for multiple I/Os at once, the functions `PciedioSetIOIrqEdgeByte`, `PciedioSetIOIrqEdgeWord` and `PciedioSetIOIrqEdgeDWord` can be used while `PciedioSetIOIrqEdgeBit` makes for the setup of the triggering edge of one particular input. The present settings can also be read back using the corresponding 'Get'-functions.

Each I/O has an individual interrupt enable bit which determines if an I/O can issue an interrupt or not. To enable or disable interrupts for multiple I/Os at once, the API functions `PciedioSetIOIrqEnableByte`, `PciedioSetIOIrqEnableWord` and `PciedioSetIOIrqEnableDWord`, can be used. To enable or disable the interrupt of one particular I/O, `PciedioSetIOIrqEnableBit` can be called. The present settings can also be read back using the corresponding 'Get'-functions.

To program the triggering-edge and the enable bit of a particular I/O at once, `PciedioSetIOIrqConfigBit` can be used, and to read both settings of a particular I/O, `PciedioGetIOIrqConfigBit` can be operated.

If an interrupt of an I/O has occurred, it must be serviced in the user's interrupt service routine. `PciedioServiceIOIrqByte`, `PciedioServiceIOIrqWord` and `PciedioServiceIOIrqDWord` carry out the required tasks after one or more I/O interrupts have occurred while `PciedioServiceIOIrqBit` carries out the required tasks for one particular interrupt.

PciedioGetIOByte

Description

This function reads 8-bit data from the specified I/O group of 8 I/Os of a particular board. If an I/O is at high level, the corresponding bit position will carry a '1'. If an I/O is at low level, the corresponding bit position will carry a '0'.

Syntax

```
LONG PciedioGetIOByte(
  BYTE  cardnum,
  BYTE  group,
  BYTE *state
);
```

Parameters

`cardnum [in]`

The index of the card corresponding to the jumper settings

`group [in]`

The group of 8 I/Os which has to be read

0 : 1st byte = IO07..IO00
1 : 2nd byte = IO15..IO08
2 : 3rd byte = IO23..IO16
3 : 4th byte = IO31..IO24

`state [out]`

A pointer to an 8-bit variable that receives the state of the specified I/O group

Return value

0: the operation succeeded successfully
-1: the board addressed with `cardnum` does not exist or is not initialized
-2: an internal Windows error occurred
-3: an IOCTL error occurred
-4: the parameter `group` is out of the specified range

Remarks

The 32 I/Os are 1-bit, 8-bit, 16-bit and 32-bit addressable

Requirements

DLL Version 1.0 or above, and a successful call of the function PciedioOpenCards

Example

```
//read the fourth 8-bit group of board 2 to the variable state
BYTE state;
PciedioGetIOByte(2, 3, &state);
//bit 0 of state now equals the level of IO24, bit 1 equals the level of IO25, ..., and
//bit 7 equals the level of IO31
```

PciedioGetIOWord

Description

This function reads 16-bit data from the specified I/O group of 16 I/Os of a particular board. If an I/O is at high level, the corresponding bit position will carry a '1'. If an I/O is at low level, the corresponding bit position will carry a '0'.

Syntax

```
LONG PciedioGetIOWord(
  BYTE  cardnum,
  BYTE  group,
  WORD *state
);
```

Parameters

`cardnum [in]`

The index of the card corresponding to the jumper settings

`group [in]`

The group of 16 I/Os which has to be read

0 : 1st word = IO15..IO00

1 : 2nd word = IO31..IO16

`state [out]`

A pointer to a 16-bit variable that receives the state of the specified I/O group

Return value

0: the operation succeeded successfully

-1: the board addressed with `cardnum` does not exist or is not initialized

-2: an internal Windows error occurred

-3: an IOCTL error occurred

-4: the parameter `group` is out of the specified range

Remarks

The 32 I/Os are 1-bit, 8-bit, 16-bit and 32-bit addressable

Requirements

DLL Version 1.0 or above, and a successful call of the function PciedioOpenCards

Example

```
//read the first 16-bit group of board 0 to the variable state
WORD state;
PciedioGetIOWord(0, 0, &state);
//bit 0 of state now equals the level of IO00, bit 1 equals the level of IO01, ..., and
//bit 15 equals the level of IO15
```

PciedioGetIODWord

Description

This function reads 32-bit data from a particular board. If an I/O is at high level, the corresponding bit position will carry a '1'. If an I/O is at low level, the corresponding bit position will carry a '0'.

Syntax

```
LONG PciedioGetIODWord(
  BYTE   cardnum,
  DWORD *state
);
```

Parameters

`cardnum [in]`
The index of the card corresponding to the jumper settings

`state [out]`
A pointer to a 32-bit variable that receives the state of the specified I/O group

Return value

0: the operation succeeded successfully
-1: the board addressed with `cardnum` does not exist or is not initialized
-2: an internal Windows error occurred
-3: an IOCTL error occurred

Remarks

The 32 I/Os are 1-bit, 8-bit, 16-bit and 32-bit addressable

Requirements

DLL Version 1.0 or above, and a successful call of the function PciedioOpenCards

Example

```
//read all 32 IOs of board 0 to the variable state
DWORD state;
PciedioGetIODWord(0, &state);
//bit 0 of state now equals the level of IO00, bit 1 equals the level of IO01, ..., and
//bit 31 equals the level of IO31
```

PciedioGetIOBit

Description

This function reads one specified I/O of a particular board.

Syntax

```
LONG PciedioGetIOBit(
  BYTE  cardnum,
  BYTE  channel,
  BYTE *state
);
```

Parameters

`cardnum [in]`

The index of the card corresponding to the jumper settings

`channel [in]`

The channel from 0 to 31, where 0 = IO00, 1 = IO01, 2 = IO02, … , 30 = IO30, and 31 = IO31

`state [out]`

A pointer to an 8-bit variable that receives the state of the specified I/O. If the specified I/O is at high level, `state` will carry a '1'. If the I/O is at low level, `state` will carry a '0'.

Return value

0: the operation succeeded successfully
-1: the board addressed with `cardnum` does not exist or is not initialized
-2: an internal Windows error occurred
-3: an IOCTL error occurred
-4: the parameter `channel` is out of the specified range

Remarks

The 32 I/Os are 1-bit, 8-bit, 16-bit and 32-bit addressable

Requirements

DLL Version 1.0 or above, and a successful call of the function PciedioOpenCards

Example

```
//read the level of IO25 of board 0 to the variable state
BYTE state;
PciedioGetIOBit(0, 25, &state);
// state (or bit0 of it) now equals the level of the specified channel IO25
```

PciedioSetIOIrqEdgeByte

Description

This function writes 8-bit data to the Interrupt Edge Register related to the specified I/O group of 8 I/Os of a particular board. The Interrupt Edge Register selects the triggering edges at which I/Os issue interrupts if interrupts are or will be enabled. If the corresponding bit position carries a '0' (default), the triggering edge will be set from high level to low level; if it carries a '1', the triggering edge will be set from low level to high level.

Syntax

```
LONG PciedioSetIOIrqEdgeByte(
  BYTE cardnum,
  BYTE group,
  BYTE edge
);
```

Parameters

`cardnum [in]`
The index of the card corresponding to the jumper settings

`group [in]`
The group of 8 I/Os for which the edges have to be set
0 : 1st byte = IO07..IO00
1 : 2nd byte = IO15..IO08
2 : 3rd byte = IO23..IO16
3 : 4th byte = IO31..IO24

`edge [in]`
The 8-bit data to be set for the triggering edges of the stated group

Return value

0: the operation succeeded successfully
-1: the board addressed with `cardnum` does not exist or is not initialized
-2: an internal Windows error occurred
-3: an IOCTL error occurred
-4: the parameter `group` is out of the specified range

Remarks

The Interrupt Edge Register is 1-bit, 8-bit, 16-bit and 32-bit addressable

Requirements

DLL Version 1.0 or above, and a successful call of the function PciedioOpenCards

Example

```
//select the triggering edges for the second 8-bit group of board 2 with 13hex, so that
//IO12, IO09 and IO08 trigger at low level to high level, and IO15, IO14, IO13, IO11 and
//IO10 trigger at high level to low level
PciedioSetIOIrqEdgeByte(2, 1, 0x13);
```

PciedioSetIOIrqEdgeWord

Description

This function writes 16-bit data to the Interrupt Edge Register related to the specified I/O group of 16 I/Os of a particular board. The Interrupt Edge Register selects the triggering edges at which I/Os issue interrupts if interrupts are or will be enabled. If the corresponding bit position carries a '0' (default), the triggering edge will be set from high level to low level; if it carries a '1', the triggering edge will be set from low level to high level.

Syntax

```
LONG PciedioSetIOIrqEdgeWord(
  BYTE cardnum,
  BYTE group,
  WORD edge
);
```

Parameters

`cardnum [in]`
The index of the card corresponding to the jumper settings

`group [in]`
The group of 16 I/Os for which the edges have to be set
0 : 1st word = IO15..IO00
1 : 2nd word = IO31..IO16

`edge [in]`
The 16-bit data to be set for the triggering edges of the stated group

Return value

0: the operation succeeded successfully
-1: the board addressed with `cardnum` does not exist or is not initialized
-2: an internal Windows error occurred
-3: an IOCTL error occurred
-4: the parameter `group` is out of the specified range

Remarks

The Interrupt Edge Register is 1-bit, 8-bit, 16-bit and 32-bit addressable

Requirements

DLL Version 1.0 or above, and a successful call of the function PciedioOpenCards

Example

```
//select the triggering edges for the first 16-bit group of board 1 with 8012hex, so that
//IO15, IO04 and IO01 trigger at low level to high level, and all others of that group
//trigger at high level to low level
PciedioSetIOIrqEdgeWord(1, 0, 0x8012);
```

PciedioSetIOIrqEdgeDWord

Description

This function writes 32-bit data to the Interrupt Edge Register related to the 32 I/Os of a particular board. The Interrupt Edge Register selects the triggering edges at which I/Os issue interrupts if interrupts are or will be enabled. If the corresponding bit position carries a '0' (default), the triggering edge will be set from high level to low level; if it carries a '1', the triggering edge will be set from low level to high level.

Syntax

```
LONG PciedioSetIOIrqEdgeDWord(
  BYTE  cardnum,
  DWORD edge
);
```

Parameters

`cardnum [in]`
The index of the card corresponding to the jumper settings

`edge [in]`
The 32-bit data to be set for the triggering edges of the 32 I/Os

Return value

0: the operation succeeded successfully
-1: the board addressed with `cardnum` does not exist or is not initialized
-2: an internal Windows error occurred
-3: an IOCTL error occurred

Remarks

The Interrupt Edge Register is 1-bit, 8-bit, 16-bit and 32-bit addressable

Requirements

DLL Version 1.0 or above, and a successful call of the function PciedioOpenCards

Example

```
//select the triggering edges for the 32 IOs of board 3 with 80104001hex, so that IO31,
//IO20, IO14 and IO00 trigger at low level to high level, and all others trigger at high
//level to low level
PciedioSetIOIrqEdgeDWord(3, 0x80104001);
```

PciedioSetIOIrqEdgeBit

Description

This function sets the interrupt triggering edge of one dedicated I/O of a particular board. By default, the triggering edges of all 32 I/Os are set to trigger from high level to low level.

Syntax

```
LONG PciedioSetIOIrqEdgeBit(
  BYTE cardnum,
  BYTE channel,
  BYTE edge
);
```

Parameters

`cardnum [in]`
The index of the card corresponding to the jumper settings

`channel [in]`
The channel from 0 to 31, where 0 relates to IO00, 1 to IO01, 2 to IO02, … , 30 to IO30, and 31 to IO31

`edge [in]`
The triggering edge of the stated I/O to be set with
0: high level to low level
1: low level to high level

Return value

0: the operation succeeded successfully
-1: the board addressed with `cardnum` does not exist or is not initialized
-2: an internal Windows error occurred
-3: an IOCTL error occurred
-4: the parameter `channel` is out of the specified range
-5: the parameter `edge` is out of the specified range

Remarks

The Interrupt Edge Register is 1-bit, 8-bit, 16-bit and 32-bit addressable

Requirements

DLL Version 1.0 or above, and a successful call of the function PciedioOpenCards

Example

```
//select the triggering edge for IO10 of board 0 from low level to high level
PciedioSetIOIrqEdgeBit(0, 10, 1);

//select the triggering edge for IO22 of board 0 from high level to low level
PciedioSetIOIrqEdgeBit(0, 22, 0);
```

PciedioGetIOIrqEdgeByte

Description

This function reads 8-bit data from the Interrupt Edge Register related to the specified I/O group of 8 I/Os of a particular board. The Interrupt Edge Register contains the triggering edges at which I/Os issue interrupts if interrupts are or will be enabled. If the corresponding bit position carries a '0' (default), the triggering edge is set from high level to low level; if it carries a '1', the triggering edge is be set from low level to high level.

Syntax

```
LONG PciedioGetIOIrqEdgeByte(
  BYTE  cardnum,
  BYTE  group,
  BYTE *state
);
```

Parameters

`cardnum [in]`
The index of the card corresponding to the jumper settings

`group [in]`
The group of 8 I/Os for which the edge settings have to be read
0 : 1st byte = IO07..IO00
1 : 2nd byte = IO15..IO08
2 : 3rd byte = IO23..IO16
3 : 4th byte = IO31..IO24

`state [out]`
A pointer to an 8-bit variable that receives the edge settings of the specified I/O group

Return value

0: the operation succeeded successfully
-1: the board addressed with `cardnum` does not exist or is not initialized
-2: an internal Windows error occurred
-3: an IOCTL error occurred
-4: the parameter `group` is out of the specified range

Remarks

The Interrupt Edge Register is 1-bit, 8-bit, 16-bit and 32-bit addressable

Requirements

DLL Version 1.0 or above, and a successful call of the function PciedioOpenCards

Example

```
//read the fourth 8-bit group of board 2 to the variable state
BYTE state;
PciedioGetIOIrqEdgeByte(2, 3, &state);
```

PciedioGetIOIrqEdgeWord

Description

This function reads 16-bit data from the Interrupt Edge Register related to the specified I/O group of 16 I/Os of a particular board. The Interrupt Edge Register contains the triggering edges at which I/Os issue interrupts if interrupts are or will be enabled. If the corresponding bit position carries a '0' (default), the triggering edge is set from high level to low level; if it carries a '1', the triggering edge is be set from low level to high level.

Syntax

```
LONG PciedioGetIOIrqEdgeWord(
  BYTE  cardnum,
  BYTE  group,
  WORD *state
);
```

Parameters

`cardnum [in]`
The index of the card corresponding to the jumper settings

`group [in]`
The group of 16 I/Os for which the edge settings have to be read
0 : 1st word = IO15..IO00
1 : 2nd word = IO31..IO16

`state [out]`
A pointer to a 16-bit variable that receives the edge settings of the specified I/O group

Return value

0: the operation succeeded successfully
-1: the board addressed with `cardnum` does not exist or is not initialized
-2: an internal Windows error occurred
-3: an IOCTL error occurred
-4: the parameter `group` is out of the specified range

Remarks

The Interrupt Edge Register is 1-bit, 8-bit, 16-bit and 32-bit addressable

Requirements

DLL Version 1.0 or above, and a successful call of the function PciedioOpenCards

Example

```
//read the second 16-bit group of board 1 to the variable state
WORD state;
PciedioGetIOIrqEdgeWord(1, 1, &state);
```

PciedioGetIOIrqEdgeDWord

Description

This function reads 32-bit data from the Interrupt Edge Register related to the 32 I/Os of a particular board. The Interrupt Edge Register contains the triggering edges at which I/Os issue interrupts if interrupts are or will be enabled. If the corresponding bit position carries a '0' (default), the triggering edge is set from high level to low level; if it carries a '1', the triggering edge is be set from low level to high level.

Syntax

```
LONG PciedioGetIOIrqEdgeDWord(
  BYTE   cardnum,
  DWORD *state
);
```

Parameters

`cardnum [in]`

The index of the card corresponding to the jumper settings

`state [out]`

A pointer to a 32-bit variable that receives the edge settings of the 32 I/Os

Return value

0: the operation succeeded successfully
-1: the board addressed with `cardnum` does not exist or is not initialized
-2: an internal Windows error occurred
-3: an IOCTL error occurred

Remarks

The Interrupt Edge Register is 1-bit, 8-bit, 16-bit and 32-bit addressable

Requirements

DLL Version 1.0 or above, and a successful call of the function PciedioOpenCards

Example

```
//read the edge settings of the 32 IOs of board 1 to the variable state
DWORD state;
PciedioGetIOIrqEdgeDWord(1, &state);
```

PciedioGetIOIrqEdgeBit

Description

This function reads the settings of the interrupt triggering edge of one dedicated I/O of a particular board.

Syntax

```
LONG PciedioGetIOIrqEdgeBit(
  BYTE  cardnum,
  BYTE  channel,
  BYTE *state
);
```

Parameters

`cardnum [in]`

The index of the card corresponding to the jumper settings

`channel [in]`

The channel from 0 to 31, where 0 relates to IO00, 1 to IO01, 2 to IO02, … , 30 to IO30, and 31 to IO31

`state [out]`

A pointer to an 8-bit variable that receives the edge settings of the stated I/O with
0: high level to low level
1: low level to high level

Return value

0: the operation succeeded successfully
-1: the board addressed with `cardnum` does not exist or is not initialized
-2: an internal Windows error occurred
-3: an IOCTL error occurred
-4: the parameter `channel` is out of the specified range

Remarks

The Interrupt Edge Register is 1-bit, 8-bit, 16-bit and 32-bit addressable

Requirements

DLL Version 1.0 or above, and a successful call of the function PciedioOpenCards

Example

```
//read the edge settings of IO12 of board 0 to the variable state
BYTE state;
PciedioGetIOIrqEdgeBit(0, 12, &state);
```

PciedioSetIOIrqEnableByte

Description

This function writes 8-bit data to the Interrupt Enable Register related to the specified I/O group of 8 I/Os of a particular board. The Interrupt Enable Register acts as the local interrupt enable mask for the 32 I/Os. If the corresponding bit position carries a '0' (default), the respective interrupt is disabled; if it carries a '1', the respective interrupt is enabled. The function does not enable or disable the interrupts of a particular board globally.

Syntax

```
LONG PciedioSetIOIrqEnableByte(
  BYTE cardnum,
  BYTE group,
  BYTE endis
);
```

Parameters

`cardnum [in]`
The index of the card corresponding to the jumper settings

`group [in]`
The group of 8 I/Os for which the interrupt enable settings have to be written
0 : 1st byte = IO07..IO00
1 : 2nd byte = IO15..IO08
2 : 3rd byte = IO23..IO16
3 : 4th byte = IO31..IO24

`endis [in]`
The 8-bit data to be written to the local interrupt enable settings of the stated group

Return value

0: the operation succeeded successfully
-1: the board addressed with `cardnum` does not exist or is not initialized
-2: an internal Windows error occurred
-3: an IOCTL error occurred
-4: the parameter `group` is out of the specified range

Remarks

The Interrupt Enable Register is 1-bit, 8-bit, 16-bit and 32-bit addressable

Requirements

DLL Version 1.0 or above, and a successful call of the function PciedioOpenCards

Example

```
//write the interrupt enable settings for the second 8-bit group of board 2 with 23hex,
//so that the interrupts of IO13, IO09 and IO08 are locally enabled and the interrupts of
//IO15, IO14, IO12, IO11 and IO10 are locally disabled
PciedioSetIOIrqEnableByte(2, 1, 0x23);
```

PciedioSetIOIrqEnableWord

Description

This function writes 16-bit data to the Interrupt Enable Register related to the specified I/O group of 16 I/Os of a particular board. The Interrupt Enable Register acts as the local interrupt enable mask for the 32 I/Os. If the corresponding bit position carries a '0' (default), the respective interrupt is disabled; if it carries a '1', the respective interrupt is enabled. The function does not enable or disable the interrupts of a particular board globally.

Syntax

```
LONG PciedioSetIOIrqEnableWord(
  BYTE cardnum,
  BYTE group,
  WORD endis
);
```

Parameters

`cardnum [in]`

The index of the card corresponding to the jumper settings

`group [in]`

The group of 16 I/Os for which the interrupt enable settings have to be written
0 : 1st word = IO15..IO00
1 : 2nd word = IO31..IO16

`endis [in]`

The 16-bit data to be written to the local interrupt enable settings of the stated group

Return value

0: the operation succeeded successfully
-1: the board addressed with `cardnum` does not exist or is not initialized
-2: an internal Windows error occurred
-3: an IOCTL error occurred
-4: the parameter `group` is out of the specified range

Remarks

The Interrupt Enable Register is 1-bit, 8-bit, 16-bit and 32-bit addressable

Requirements

DLL Version 1.0 or above, and a successful call of the function PciedioOpenCards

Example

```
//write the interrupt enable settings for the second 16-bit group of board 2 with
//8040hex, so that the interrupts of IO31 and IO22 are locally enabled and the interrupts
//of all other IOs of the specified group are locally disabled
PciedioSetIOIrqEnableWord(2, 1, 0x8040);
```

PciedioSetIOIrqEnableDWord

Description

This function writes 32-bit data to the Interrupt Enable Register related to the 32 I/Os of a particular board. The Interrupt Enable Register acts as the local interrupt enable mask for the 32 I/Os. If the corresponding bit position carries a '0' (default), the respective interrupt is disabled; if it carries a '1', the respective interrupt is enabled. The function does not enable or disable the interrupts of a particular board globally.

Syntax

```
LONG PciedioSetIOIrqEnableDWord(
  BYTE  cardnum,
  DWORD endis
);
```

Parameters

`cardnum [in]`
The index of the card corresponding to the jumper settings

`endis [in]`
The 32-bit data to be written to the local interrupt enable settings of the 32 I/Os

Return value

0: the operation succeeded successfully
-1: the board addressed with `cardnum` does not exist or is not initialized
-2: an internal Windows error occurred
-3: an IOCTL error occurred

Remarks

The Interrupt Enable Register is 1-bit, 8-bit, 16-bit and 32-bit addressable

Requirements

DLL Version 1.0 or above, and a successful call of the function PciedioOpenCards

Example

```
//write the interrupt enable settings for the 32 IOs of board 1 with 10014002hex, so that
//the interrupts of IO28, IO16, IO14 and IO01 are locally enabled and the interrupts of
//all other IOs are locally disabled
PciedioSetIOIrqEnableDWord(1, 0x10014002);
```

PciedioSetIOIrqEnableBit

Description

This function sets the local interrupt enable/disable bit of one dedicated I/O in the Interrupt Enable Register of a particular board. By default, the local interrupts of all 32 I/Os are disabled. The function does not enable or disable the interrupts of a particular board globally.

Syntax

```
LONG PciedioSetIOIrqEnableBit(
  BYTE cardnum,
  BYTE channel,
  BYTE endis
);
```

Parameters

`cardnum [in]`

The index of the card corresponding to the jumper settings

`channel [in]`

The channel from 0 to 31, where 0 relates to IO00, 1 to IO01, 2 to IO02, … , 30 to IO30, and 31 to IO31

`endis [in]`

The enable/disable settings of the stated I/O to be set with
0: disable the local interrupt for the statetd I/O
1: enable the local interrupt for the stated I/O

Return value

0: the operation succeeded successfully
-1: the board addressed with `cardnum` does not exist or is not initialized
-2: an internal Windows error occurred
-3: an IOCTL error occurred
-4: the parameter `channel` is out of the specified range
-5: the parameter `endis` is out of the specified range

Remarks

The Interrupt Enable Register is 1-bit, 8-bit, 16-bit and 32-bit addressable

Requirements

DLL Version 1.0 or above, and a successful call of the function PciedioOpenCards

Example

```
//enable the interrupt for IO18 of board 0
PciedioSetIOIrqEnableBit(0, 18, 1);

//disable the interrupt for IO28 of board 0
PciedioSetIOIrqEnableBit(0, 28, 0);
```

PciedioGetIOIrqEnableByte

Description

This function reads 8-bit data from the Interrupt Enable Register related to the specified I/O group of 8 I/Os of a particular board. The Interrupt Enable Register acts as the local interrupt enable mask for the 32 I/Os. If the corresponding bit position carries a '0' (default), the respective interrupt is disabled; if it carries a '1', the respective interrupt is enabled.

Syntax

```
LONG PciedioGetIOIrqEnableByte(
  BYTE  cardnum,
  BYTE  group,
  BYTE *state
);
```

Parameters

`cardnum [in]`

The index of the card corresponding to the jumper settings

`group [in]`

The group of 8 I/Os from which the interrupt enable settings have to be read

0 : 1st byte = IO07..IO00
1 : 2nd byte = IO15..IO08
2 : 3rd byte = IO23..IO16
3 : 4th byte = IO31..IO24

`state [out]`

A pointer to an 8-bit variable that receives the local interrupt enable settings of the stated group

Return value

0: the operation succeeded successfully
-1: the board addressed with `cardnum` does not exist or is not initialized
-2: an internal Windows error occurred
-3: an IOCTL error occurred
-4: the parameter `group` is out of the specified range

Remarks

The Interrupt Enable Register is 1-bit, 8-bit, 16-bit and 32-bit addressable

Requirements

DLL Version 1.0 or above, and a successful call of the function PciedioOpenCards

Example

```
//read the interrupt enable settings of the first 8-bit group of board 1
BYTE state;
PciedioGetIOIrqEnableByte(1, 0, &state);
```

PciedioGetIOIrqEnableWord

Description

This function reads 16-bit data from the Interrupt Enable Register related to the specified I/O group of 16 I/Os of a particular board. The Interrupt Enable Register acts as the local interrupt enable mask for the 32 I/Os. If the corresponding bit position carries a '0' (default), the respective interrupt is disabled; if it carries a '1', the respective interrupt is enabled.

Syntax

```
LONG PciedioGetIOIrqEnableWord(
  BYTE  cardnum,
  BYTE  group,
  WORD *state
);
```

Parameters

`cardnum [in]`

The index of the card corresponding to the jumper settings

`group [in]`

The group of 16 I/Os from which the interrupt enable settings have to be read
0 : 1st word = IO15..IO00
1 : 2nd word = IO31..IO16

`state [out]`

A pointer to a 16-bit variable that receives the local interrupt enable settings of the stated group

Return value

0: the operation succeeded successfully
-1: the board addressed with `cardnum` does not exist or is not initialized
-2: an internal Windows error occurred
-3: an IOCTL error occurred
-4: the parameter `group` is out of the specified range

Remarks

The Interrupt Enable Register is 1-bit, 8-bit, 16-bit and 32-bit addressable

Requirements

DLL Version 1.0 or above, and a successful call of the function PciedioOpenCards

Example

```
//read the interrupt enable settings of the second 16-bit group of board 1
WORD state;
PciedioGetIOIrqEnableWord(1, 1, &state);
```

PciedioGetIOIrqEnableDWord

Description

This function reads 32-bit data from the Interrupt Enable Register related to the 32 I/Os of a particular board. The Interrupt Enable Register acts as the local interrupt enable mask for the 32 I/Os. If the corresponding bit position carries a '0' (default), the respective interrupt is disabled; if it carries a '1', the respective interrupt is enabled.

Syntax

```
LONG PciedioGetIOIrqEnableDWord(
  BYTE   cardnum,
  DWORD *state
);
```

Parameters

`cardnum [in]`
The index of the card corresponding to the jumper settings

`state [out]`
A pointer to a 32-bit variable that receives the local interrupt enable settings of the 32 I/Os

Return value

0: the operation succeeded successfully
-1: the board addressed with `cardnum` does not exist or is not initialized
-2: an internal Windows error occurred
-3: an IOCTL error occurred

Remarks

The Interrupt Enable Register is 1-bit, 8-bit, 16-bit and 32-bit addressable

Requirements

DLL Version 1.0 or above, and a successful call of the function PciedioOpenCards

Example

```
//read the interrupt enable settings of the 32 IOs of board 1
DWORD state;
PciedioGetIOIrqEnableDWord(1, &state);
```

PciedioGetIOIrqEnableBit

Description

This function reads the local interrupt enable/disable bit of one dedicated I/O from the Interrupt Enable Register of a particular board. By default, the local interrupts of all 32 I/Os are disabled.

Syntax

```
LONG PciedioGetIOIrqEnableBit(
  BYTE cardnum,
  BYTE channel,
  BYTE *state
);
```

Parameters

`cardnum [in]`
The index of the card corresponding to the jumper settings

`channel [in]`
The channel from 0 to 31, where 0 relates to IO00, 1 to IO01, 2 to IO02, … , 30 to IO30, and 31 to IO31

`state [out]`
A pointer to an 8-bit variable that receives the enable/disable settings of the stated I/O with
0: the local interrupt of the statetd I/O is disabled
1: the local interrupt of the stated I/O is enabled

Return value

0: the operation succeeded successfully
-1: the board addressed with `cardnum` does not exist or is not initialized
-2: an internal Windows error occurred
-3: an IOCTL error occurred
-4: the parameter `channel` is out of the specified range

Remarks

The Interrupt Enable Register is 1-bit, 8-bit, 16-bit and 32-bit addressable

Requirements

DLL Version 1.0 or above, and a successful call of the function PciedioOpenCards

Example

```
//read the interrupt enable settings of IO017 of board 0
PciedioGetIOIrqEnableBit(0, 17);

//read the interrupt enable settings of IO26 of board 0
PciedioGetIOIrqEnableBit(0, 26);
```

PciedioSetIOIrqConfigBit

Description

This function sets the interrupt triggering edge and the local interrupt enable/disable bit of one dedicated I/O of a particular board in one operation. The function does not enable or disable the interrupts of a particular board globally.

By default, the triggering edges of all 32 I/Os are set to trigger from high level to low level and the local interrupt enable/disable bits of all 32 I/Os are disabled.

Syntax

```
LONG PciedioSetIOIrqConfigBit(
  BYTE cardnum,
  BYTE channel,
  BYTE endis,
  BYTE edge
);
```

Parameters

`cardnum [in]`
The index of the card corresponding to the jumper settings

`channel [in]`
The channel from 0 to 31, where 0 relates to IO00, 1 to IO01, 2 to IO02, … , 30 to IO30, and 31 to IO31

`endis [in]`
The enable/disable settings of the stated I/O to be set with
0: disable the interrupt of the statetd I/O
1: enable the interrupt of the stated I/O

`edge [in]`
The triggering edge of the stated I/O to be set with
0: high level to low level
1: low level to high level

Return value

0: the operation succeeded successfully
-1: the board addressed with `cardnum` does not exist or is not initialized
-2: an internal Windows error occurred
-3: an IOCTL error occurred
-4: the parameter `channel` is out of the specified range
-5: the parameter `endis` is out of the specified range
-6: the parameter `edge` is out of the specified range

Remarks

This function can be used to set the triggering edge and the enable/disable bit. The sequence of operation differs depending on the setting of the parameter `endis`:
If an interrupt has to be enabled, the function first sets up the triggering edge and afterwards enables the interrupt.
If an interrupt has to be disabled, this function first disables the interrupt and afterwards sets up the triggering edge.

Requirements

DLL Version 1.0 or above, and a successful call of the function PciedioOpenCards

Example

```
//enable the interrupt of IO 15 of board 0 and select the triggering edge of it from low
//level to high level
PciedioSetIOIrqConfigBit(0, 15, 1, 1);

//disable the interrupt of IO29 of board 1 and select the triggering edge of it from high
//level to low level
PciedioSetIOIrqConfigBit(1, 29, 0, 0);
```

PciedioGetIOIrqConfigBit

Description

This function receives the present settings of the interrupt triggering edge and the local interrupt enable/disable bit of one dedicated I/O of a particular board in one operation.

Syntax

```
LONG PciedioGetIOIrqConfigBit(
  BYTE  cardnum,
  BYTE  channel,
  BYTE *endis,
  BYTE *edge
);
```

Parameters

`cardnum [in]`
The index of the card corresponding to the jumper settings

`channel [in]`
The channel from 0 to 31, where 0 relates to IO00, 1 to IO01, 2 to IO02, … , 30 to IO30, and 31 to IO31

`endis [out]`
A pointer to an 8-bit variable that receives the local interrupt enable/disable settings of the stated I/O with
0: the interrupt of the statetd I/O is disabled
1: the interrupt of the stated I/O is enabled

`edge [out]`
A pointer to an 8-bit variable that receives the interrupt edge settings of the stated I/O with
0: high level to low level
1: low level to high level

Return value

0: the operation succeeded successfully
-1: the board addressed with `cardnum` does not exist or is not initialized
-2: an internal Windows error occurred
-3: an IOCTL error occurred
-4: the parameter `channel` is out of the specified range

Requirements

DLL Version 1.0 or above, and a successful call of the function PciedioOpenCards

Example

```
//receive the present settings of the interrupt enable/disable bit and the interrupt
//triggering edge of IO10 of board 0
BYTE *endis, *edge;
PciedioSetIOIrqConfigBit(0, 10, &endis, &edge);
```

PciedioServiceIOIrqByte

Description

This function carries out the required tasks after one or more I/O interrupts of an 8-bit group of a particular board have occurred. It writes 8-bit data to the Interrupt Service Register related to the stated 8-bit group of I/Os. If the corresponding bit position carries a '1', the interrupt of the respective I/O will be serviced and kept enabled afterwards; if the corresponding bit position carries a '0', no service takes place. The corresponding bit positions of I/Os which have not issued an interrupt must not be set to a '1'.

Syntax

```
LONG PciedioServiceIOIrqByte(
  BYTE cardnum,
  BYTE group,
  BYTE service
);
```

Parameters

`cardnum [in]`
The index of the card corresponding to the jumper settings

`group [in]`
The group of 8 I/Os which receives the service
0 : 1st byte = IO07..IO00
1 : 2nd byte = IO15..IO08
2 : 3rd byte = IO23..IO16
3 : 4th byte = IO31..IO24

`service [in]`
The 8-bit data to be written to the Interrupt Service Register

Return value

0: the operation succeeded successfully
-1: the board addressed with `cardnum` does not exist or is not initialized
-2: an internal Windows error occurred
-3: an IOCTL error occurred
-4: the parameter `group` is out of the specified range

Remarks

This function is usually called within the user's interrupt service routine, which automatically receives notification of the source of the interrupt.

Requirements

DLL Version 1.0 or above, and a successful call of the function PciedioOpenCards

Example

```
//service the interrupts of IO08 and IO14 (both belong to the second group) of board 0
PciedioServiceIOIrqByte(0, 1, 0x41);
```

PciedioServiceIOIrqWord

Description

This function carries out the required tasks after one or more I/O interrupts of a 16-bit group of a particular board have occurred. It writes 16-bit data to the Interrupt Service Register related to the stated 16-bit group of I/Os. If the corresponding bit position carries a '1', the interrupt of the respective I/O will be serviced and kept enabled afterwards; if the corresponding bit position carries a '0', no service takes place. The corresponding bit positions of I/Os which have not issued an interrupt must not be set to a '1'.

Syntax

```
LONG PciedioServiceIOIrqWord(
  BYTE cardnum,
  BYTE group,
  WORD service
);
```

Parameters

`cardnum [in]`
The index of the card corresponding to the jumper settings

`group [in]`
The group of 16 I/Os which receives the service
0 : 1st word = IO15..IO00
1 : 2nd word = IO31..IO16

`service [in]`
The 16-bit data to be written to the Interrupt Service Register

Return value

0: the operation succeeded successfully
-1: the board addressed with `cardnum` does not exist or is not initialized
-2: an internal Windows error occurred
-3: an IOCTL error occurred
-4: the parameter `group` is out of the specified range

Remarks

This function is usually called within the user's interrupt service routine, which automatically receives notification of the source of the interrupt.

Requirements

DLL Version 1.0 or above, and a successful call of the function PciedioOpenCards

Example

```
//service the interrupts of IO14, IO12 and IO02 (all belong to the first group) of board 0
PciedioServiceIOIrqWord(0, 0, 0x5004);
```

PciedioServiceIOIrqDWord

Description

This function carries out the required tasks after one or more I/O interrupts of the 32 I/Os of a particular board have occurred. It writes 32-bit data to the Interrupt Service Register related to the 32 I/Os. If the corresponding bit position carries a '1', the interrupt of the respective I/O will be serviced and kept enabled afterwards; if the corresponding bit position carries a '0', no service takes place. The corresponding bit positions of I/Os which have not issued an interrupt must not be set to a '1'.

Syntax

```
LONG PciedioServiceIOIrqDWord(
  BYTE  cardnum,
  DWORD service
);
```

Parameters

`cardnum [in]`
The index of the card corresponding to the jumper settings

`service [in]`
The 32-bit data to be written to the Interrupt Service Register

Return value

0: the operation succeeded successfully
-1: the board addressed with `cardnum` does not exist or is not initialized
-2: an internal Windows error occurred
-3: an IOCTL error occurred

Remarks

This function is usually called within the user's interrupt service routine, which automatically receives notification of the source of the interrupt.

Requirements

DLL Version 1.0 or above, and a successful call of the function PciedioOpenCards

Example

```
//service the interrupts of IO31, IO16 and IO01 of board 0
PciedioServiceIOIrqDWord(0, 0x80010002);
```

PciedioServiceIOIrqBit

Description

This function carries out the required tasks after an I/O interrupt of a particular board has occurred. It services one dedicated I/O interrupt and keeps it enabled afterwards. It must not be called if the I/O has not issued an interrupt.

Syntax

```
LONG PciedioServiceIOIrqBit(
  BYTE cardnum,
  BYTE channel
);
```

Parameters

`cardnum [in]`
The index of the card corresponding to the jumper settings

`channel [in]`
The channel from 0 to 31, where 0 relates to IO00, 1 to IO01, 2 to IO02, … , 30 to IO30, and 31 to IO31

Return value

0: the operation succeeded successfully
-1: the board addressed with `cardnum` does not exist or is not initialized
-2: an internal Windows error occurred
-3: an IOCTL error occurred
-4: the parameter `channel` is out of the specified range

Remarks

This function is usually called within the user's interrupt service routine, which automatically receives notification of the source of the interrupt.

Requirements

DLL Version 1.0 or above, and a successful call of the function PciedioOpenCards

Example

```
//service the interrupt of IO12 of board 0
PciedioServiceIOIrqIrqBit(0, 12);

//service the interrupt of IO27 of board 2
PciedioServiceIOIrqIrqBit(2, 27);
```

A.5 Operating the on-board FIFO

The on-board FIFO is completely implemented in hardware and records the level of all 32 I/Os at regular, programmable intervals. Because of the read back facility of the outputs, the levels of the I/Os operated as outputs are also stored in the FIFO.

To set up the interval at which the FIFO autonomously samples and stores the level of the I/Os, the function `PciedioSetFifoInterval` must be operated.

`PciedioStartFifo` starts the recording, and `PciedioStopFifo` stops it. To clear the contents of the FIFO without reading it out, `PciedioClearFifo` can be operated.

`PciedioGetFifoNumberOfEntries` queries how many entries are presently stored in the FIFO. One entry corresponds to one set of 32 I/Os sampled simultaneously.
`PciedioGetFifoEntries` reads the number of entries passed as a parameter out of the FIFO and removes these entries from the FIFO.

To receive notification when a specified number of entries is stored in the FIFO, an interrupt can be issued. To set the number of entries where an interrupt will be issued if the number of entries stored in the FIFO is equal or greater than the number of entries specified as the alarm level, `PciedioSetFifoIrqLevel` must be called. `PciedioEnableFifoIrq` enables, and `PciedioDisableFifoIrq` disables the interrupt itself locally.

If the FIFO has issued an interrupt, it must be serviced in the user's interrupt service routine. `PciedioServiceFifoIrq` carries out the required tasks to service the interrupt appropriately.

PciedioSetFifoInterval

Description

This function sets the cycle time of the FIFO timer of a particular board in increments of 10 microseconds. If afterwards the FIFO timer is started, the input states of the 32 I/Os will be sampled and stored in the FIFO according to the cycle time set by this function.

The resultant interval, at which the FIFO timer operates, is always one additional increment higher than programmed.
The resultant cycle time can therefore be calculated by
cycle time = (`interval` + 1) * 10µs
The required `interval` value for a desired cycle time can be calculated by
`interval` = (cycle time / 10µs) – 1

Syntax

```
LONG PciedioSetFifoInterval(
  BYTE cardnum,
  WORD interval
);
```

Parameters

`cardnum [in]`
The index of the card corresponding to the jumper settings

`interval [in]`
The 16-bit interval to be set. As an `interval` of 0 (default) prevents the FIFO timer from running, the effective programmable range of `interval` is 1 to $FFFF_{hex}$

Return value

0: the operation succeeded successfully
-1: the board addressed with `cardnum` does not exist or is not initialized
-2: an internal Windows error occurred
-3: an IOCTL error occurred
-4: the parameter `interval` is 0 and therefore out of the specified range

Remarks

This function neither starts, nor stops the FIFO timer; it keeps a running FIFO timer running (with then the updated interval) and a stopped FIFO timer stopped. Likewise, it does not affect any interrupt settings.

Requirements

DLL Version 1.0 or above, and a successful call of the function PciedioOpenCards

Example

```
//set the interval of the FIFO timer of board 0 to 1 milliseconds
#define FIFO_TIMER_1MS   99    //interval = (1 milliseconds / 10µs) - 1
PciedioSetFifoTimerInterval(0, FIFO_TIMER_1MS);
```

PciedioStartFifo

Description

This function starts the FIFO timer of a particular board. If a FIFO timer interval was programmed to a non-zero value before, the FIFO timer immediately starts running and the input states of the 32 I/Os will be sampled and stored in the FIFO according to the cycle set up before. By default, the FIFO timer is stopped.

Syntax

```
LONG PciedioStartFifo(
  BYTE cardnum
);
```

Parameters

`cardnum [in]`
The index of the card corresponding to the jumper settings

Return value

0: the operation succeeded successfully
-1: the board addressed with `cardnum` does not exist or is not initialized
-2: an internal Windows error occurred
-3: an IOCTL error occurred
-4: the interval of the FIFO timer was not properly set up before

Remarks

It does not lead to an error if this function is called when the FIFO timer is already running. Likewise, this function does not affect any interrupt settings.

Requirements

DLL Version 1.0 or above, and a successful call of the functions PciedioOpenCards and PciedioSetFifoInterval

Example

```
//start the FIFO timer of board 3
PciedioStartFifo(3);
```

PciedioStopFifo

Description

This function stops the FIFO timer of a particular board. By default, the FIFO timer is stopped.

Syntax

```
LONG PciedioStopFifo(
  BYTE cardnum
);
```

Parameters

`cardnum [in]`
The index of the card corresponding to the jumper settings

Return value

0: the operation succeeded successfully
-1: the board addressed with `cardnum` does not exist or is not initialized
-2: an internal Windows error occurred
-3: an IOCTL error occurred

Remarks

It does not lead to an error if this function is called when the FIFO timer is already stopped. Likewise, this function does not affect any interrupt settings.

Requirements

DLL Version 1.0 or above, and a successful call of the function PciedioOpenCards

Example

```
//stop the FIFO timer of board 3
PciedioStopFifo(3);
```

PciedioClearFifo

Description

This function completely clears the contents of the FIFO of a particular board independent if there are any entries stored in it or not.
This function can be called independent of the state of the FIFO timer, i.e. the FIFO timer can be running or it can be stopped. If the FIFO timer is running, the sampling and storing continues right after the call of this function.

Syntax

```
LONG PciedioClearFifo(
  BYTE cardnum
);
```

Parameters

`cardnum [in]`
The index of the card corresponding to the jumper settings

Return value

0: the operation succeeded successfully
-1: the board addressed with `cardnum` does not exist or is not initialized
-2: an internal Windows error occurred
-3: an IOCTL error occurred

Remarks

This function neither starts, nor stops the FIFO timer; it keeps a running FIFO timer running and a stopped FIFO timer stopped. Likewise, it does not affect any interrupt settings.

Requirements

DLL Version 1.0 or above, and a successful call of the function PciedioOpenCards

Example

```
//clear the contents of the FIFO of board 3
PciedioClearFifo(3);
```

PciedioGetFifoNumberOfEntries

Description

This function returns the number of entries currently stored in the FIFO of a particular board, not its contents. This function can be called independent of the state of the FIFO timer, i.e. the FIFO timer can be running or it can be stopped.

Syntax

```
LONG PciedioGetFifoNumberOfEntries(
  BYTE  cardnum,
  WORD *entries
);
```

Parameters

`cardnum [in]`
The index of the card corresponding to the jumper settings

`entries [out]`
A pointer to a 16-bit variable that receives the number of entries currently stored in the FIFO

Return value

0: the operation succeeded successfully
-1: the board addressed with `cardnum` does not exist or is not initialized
-2: an internal Windows error occurred
-3: an IOCTL error occurred

Remarks

This function does not affect the contents of the FIFO, and it neither starts, nor stops the FIFO timer; it keeps a running FIFO timer running and a stopped FIFO timer stopped. Likewise, it does not affect any interrupt settings.

Requirements

DLL Version 1.0 or above, and a successful call of the function PciedioOpenCards

Example

```
//get the number of entries currently stored in the FIFO of board 0
WORD entries;
PciedioGetFifoNumberOfEntries(0, &entries);
```

PciedioGetFifoEntries

Description

This function returns the contents of the FIFO of a particular board. It returns the contents of the specified number of entries in a user provided buffer by removing them from the FIFO. The number of entries to be returned can be specified from 1 to 2048.
This function can be called independent of the state of the FIFO timer, i.e. the FIFO timer can be running or it can be stopped.

Syntax

```
LONG PciedioGetFifoEntries(
  BYTE   cardnum,
  WORD   number,
  DWORD *buffer
);
```

Parameters

`cardnum [in]`
The index of the card corresponding to the jumper settings

`number [in]`
The number of entries to be returned from the FIFO. If the specified number of entries is greater than the number currently stored in the FIFO, the contents of the buffer returned will be partly invalid.
The best practice is using the function PciedioGetNumberOfEntries before and then requesting not more than the number of entries that are actually stored in the FIFO.

`buffer [out]`
A pointer to a field of 32-bit variables that receive the contents of the FIFO. The oldest entry in the FIFO will be stored at the lowest address of the buffer. The user has to make for a `buffer` of the appropriate size to store `number` of entries.

Return value

0: the operation succeeded successfully
-1: the board addressed with `cardnum` does not exist or is not initialized
-2: an internal Windows error occurred
-3: an IOCTL error occurred
-4: the parameter `number` is out of the specified range from 1 to 2048

Remarks

This function neither starts, nor stops the FIFO timer; it keeps a running FIFO timer running and a stopped FIFO timer stopped. Likewise, it does not affect any interrupt settings.

Requirements

DLL Version 1.0 or above, and a successful call of the function PciedioOpenCards

Example

```
//get 100 entries from the FIFO of board 0
LONG buffer[100];
PciedioGetFifoEntries(0, 100, &buffer[0]);
```

PciedioSetFifoIrqLevel

Description

This function sets the alarm level of the FIFO of a particular board. If the FIFO interrupt functionality is set up, an interrupt will be issued if the number of entries stored in the FIFO is equal or greater than the user specified number of entries, which operate as the alarm level. By default, the alarm level is set to 8191 entries.

Syntax

```
LONG PciedioSetFifoIrqLevel(
  BYTE cardnum;
  WORD entries
);
```

Parameters

`cardnum [in]`
The index of the card corresponding to the jumper settings

`entries [in]`
The number of entries which operate as the alarm level. Values up to 8191 are possible.

A value of 0 is basically a valid value, but if the FIFO interrupt was enabled, interrupts would be issued even if there are no entries in the FIFO.

Return value

0: the operation succeeded successfully
-1: the board addressed with `cardnum` does not exist or is not initialized
-2: an internal Windows error occurred
-3: an IOCTL error occurred
-4: the parameter `entries` is out of the specified range from 0 to 8191

Remarks

This function neither starts, nor stops the FIFO timer; it keeps a running FIFO timer running and a stopped FIFO timer stopped. Likewise, it does not affect any interrupt settings.

Requirements

DLL Version 1.0 or above, and a successful call of the function PciedioOpenCards

Example

```
//set the alarm level of the FIFO of board 1 to 1000 entries
PciedioSetFifoIrqLevel(1, 1000);
```

PciedioEnableFifoIrq

Description

This function enables the local FIFO interrupt of a particular board. An interrupt will be issued if the number of entries currently stored in the FIFO is equal or greater than the user specified number of entries, which operate as the alarm level. By default, the FIFO interrupt is disabled.

Before using the function, the alarm level should be set up using the function PciedioSetFifoAlarm.

This function does not enable the interrupts of a particular board globally.

Syntax

```
LONG PciedioEnableFifoIrq(
  BYTE cardnum
);
```

Parameters

`cardnum [in]`
The index of the card corresponding to the jumper settings

Return value

0: the operation succeeded successfully
-1: the board addressed with `cardnum` does not exist or is not initialized
-2: an internal Windows error occurred
-3: an IOCTL error occurred

Remarks

It does not lead to an error if this function is called when the FIFO interrupt is already enabled. Likewise, this function does not affect the present state of the FIFO (‘running’ / ‘stopped’).

Requirements

DLL Version 1.0 or above, and a successful call of the function PciedioOpenCards

Example

```
//enable the FIFO interrupt of board 1
PciedioEnableFifoIrq(1);
```

PciedioDisableFifoIrq

Description

This function disables the local FIFO interrupt of a particular board. By default, the FIFO interrupt is disabled. The function does not disable the interrupts of a particular board globally.

If during or after the call of PciedioDisableFifoIrq an interrupt was already issued by the board and is still being on delivery to the user's interrupt service routine, this interrupt will still be delivered to the user's interrupt service routine.

Syntax

```
LONG PciedioDisableFifoIrq(
  BYTE cardnum
);
```

Parameters

`cardnum [in]`
The index of the card corresponding to the jumper settings

Return value

0: the operation succeeded successfully
-1: the board addressed with `cardnum` does not exist or is not initialized
-2: an internal Windows error occurred
-3: an IOCTL error occurred

Remarks

It does not lead to an error if this function is called when the FIFO interrupt is already disabled. Likewise, this function does not affect the present state of the FIFO ('running' / 'stopped').

Requirements

DLL Version 1.0 or above, and a successful call of the function PciedioOpenCards

Example

```
//disable the FIFO interrupt of board 1
PciedioDisableFifoIrq(1);
```

PciedioServiceFifoIrq

Description

This function carries out the required tasks after a FIFO interrupt of a particular board has occurred. It handles the required hardware settings and enables the FIFO interrupt again afterwards.

This function is usually called within the user's interrupt service routine, which automatically receives notification of the source of the interrupt. It must not be called if the FIFO is not the source of the interrupt.

Syntax

```
LONG PciedioServiceFifoIrq(
  BYTE cardnum
);
```

Parameters

`cardnum [in]`
The index of the card corresponding to the jumper settings

Return value

0: the operation succeeded successfully
-1: the board addressed with `cardnum` does not exist or is not initialized
-2: an internal Windows error occurred
-3: an IOCTL error occurred

Remarks

This function does not affect the present state of the FIFO ('running' / 'stopped')

Requirements

DLL Version 1.0 or above, and a successful call of the function PciedioOpenCards

Example

```
//service the FIFO interrupt of board 2
// please check if the FIFO was in actual fact the source of the interrupt before the call
PciedioServiceFifoIrq(2);
```

A.6 Operating the on-board timer

The on-board timer is completely implemented in hardware and can be used to issue interrupts at regular, programmable intervals.

To set up the interval at which the timer autonomously issues cyclic interrupts, the function `PciedioSetTimerInterval` must be operated. `PciedioStartTimer` starts the timer, and `PciedioStopTimer` stops it.

`PciedioEnableTimerIrq` enables, and `PciedioDisableTimerIrq` disables the timer interrupt itself locally. If the timer has issued an interrupt, it must be serviced in the user's interrupt service routine. `PciedioServiceTimerIrq` carries out the required tasks to service the interrupt appropriately.

`PciedioGetTimerPresentState` returns whether the timer is running or not, whether the timer interrupt is enabled or not as well as the count value of the timer.

PciedioSetTimerInterval

Description

This function sets the interval of the 24-bit timer of a particular board in increments of 100 nanoseconds. If afterwards the timer is started and interrupts are properly enabled, the timer will generate cyclic interrupts based on this programmed interval. The resultant interval, at which the timer operates, is always one additional increment higher than programmed.

The resultant cycle time can therefore be calculated by

cycle time = (`interval` + 1) * 100ns

The required `interval` value for a desired cycle time can be calculated by

`interval` = (cycle time / 100ns) - 1

Syntax

```
LONG PciedioSetTimerInterval(
  BYTE  cardnum,
  DWORD inverval
);
```

Parameters

`cardnum [in]`

The index of the card corresponding to the jumper settings

`interval [in]`

The timer interval which has to be set. The uppermost 8 bits of this 32-bit variable always have to be set to zero. A value of 0 prevents the timer from running (default), so the effective range of `interval` is 1 to $FFFFFF_{hex}$ (200ns to 1.6777216s)

Return value

0: the operation succeeded successfully
-1: the board addressed with `cardnum` does not exist or is not initialized
-2: an internal Windows error occurred
-3: an IOCTL error occurred
-4: the parameter `interval` is out of the specified range (bits 31 to 24 are not zero)

Remarks

This function neither starts, nor stops the timer; it keeps a running timer running (with then the updated interval) and a stopped timer stopped. Likewise, it does not affect any interrupt settings.

Requirements

DLL Version 1.0 or above, and a successful call of the function PciedioOpenCards

Example

```
//set the cycle time of the timer of board 0 to 10 milliseconds
#define TIMER_10MS 99999    //interval = (10 milliseconds / 100ns) - 1
PciedioSetTimerInterval(0, TIMER_10MS);
```

PciedioStartTimer

Description

This function starts the timer of a particular board. If a timer interval was programmed to a non-zero value before, the timer immediately starts running. By default, the timer is stopped.

Syntax

```
LONG PciedioStartTimer(
  BYTE cardnum
);
```

Parameters

`cardnum [in]`
The index of the card corresponding to the jumper settings

Return value

0: the operation succeeded successfully
-1: the board addressed with `cardnum` does not exist or is not initialized
-2: an internal Windows error occurred
-3: an IOCTL error occurred

Remarks

It does not lead to an error if this function is called when the timer is already running. Likewise, this function does not affect any interrupt settings.

Requirements

DLL Version 1.0 or above, and a successful call of the function PciedioOpenCards

Example

```
//start the timer of board 3
PciedioStartTimer(3);
```

PciedioStopTimer

Description

This function stops the timer of a particular board. By default, the timer is stopped.

Syntax

```
LONG PciedioStopTimer(
  BYTE cardnum
);
```

Parameters

`cardnum [in]`
The index of the card corresponding to the jumper settings

Return value

0: the operation succeeded successfully
-1: the board addressed with `cardnum` does not exist or is not initialized
-2: an internal Windows error occurred
-3: an IOCTL error occurred

Remarks

It does not lead to an error if this function is called when the timer is already stopped. Likewise, this function does not affect any interrupt settings.

Requirements

DLL Version 1.0 or above, and a successful call of the function PciedioOpenCards

Example

```
//stop the timer of board 3
PciedioStopTimer(3);
```

PciedioEnableTimerIrq

Description

This function enables the timer interrupt of a particular board locally. By default, the timer interrupt is disabled. The function does not enable the interrupts of a particular board globally.

Syntax

```
LONG PciedioEnableTimerIrq(
  BYTE cardnum
);
```

Parameters

`cardnum [in]`
The index of the card corresponding to the jumper settings

Return value

0: the operation succeeded successfully
-1: the board addressed with `cardnum` does not exist or is not initialized
-2: an internal Windows error occurred
-3: an IOCTL error occurred

Remarks

It does not lead to an error if this function is called when the timer interrupt is already enabled. Likewise, this function does not affect the present state of the timer (‘running’ / ‘stopped’).

Requirements

DLL Version 1.0 or above, and a successful call of the function PciedioOpenCards

Example

```
//enable the timer interrupt of board 1
PciedioEnableTimerIrq(1);
```

PciedioDisableTimerIrq

Description

This function disables the timer interrupt of a particular board locally. By default, the timer interrupt is disabled. The function does not disable the interrupts of a particular board globally.

If during or after the call of PciedioDisableTimerIrq an interrupt was already issued by the board and is still being on delivery to the user's interrupt service routine, this interrupt will still be delivered to the user's interrupt service routine.

Syntax

```
LONG PciedioDisableTimerIrq(
  BYTE cardnum
);
```

Parameters

`cardnum [in]`
The index of the card corresponding to the jumper settings

Return value

0: the operation succeeded successfully
-1: the board addressed with `cardnum` does not exist or is not initialized
-2: an internal Windows error occurred
-3: an IOCTL error occurred

Remarks

It does not lead to an error if this function is called when the timer interrupt is already disabled. Likewise, this function does not affect the present state of the timer ('running' / 'stopped').

Requirements

DLL Version 1.0 or above, and a successful call of the function PciedioOpenCards

Example

```
//disable the timer interrupt of board 1
PciedioDisableTimerIrq(1);
```

PciedioServiceTimerIrq

Description

This function carries out the required tasks after a timer interrupt of a particular board has occurred. It handles the required hardware settings and enables the timer interrupt again afterwards.

This function is usually called within the user's interrupt service routine, which automatically receives notification of the source of the interrupt. It must not be called if the timer is not the source of the interrupt.

Syntax

```
LONG PciedioServiceTimerIrq(
  BYTE cardnum
);
```

Parameters

`cardnum [in]`
The index of the card corresponding to the jumper settings

Return value

0: the operation succeeded successfully
-1: the board addressed with `cardnum` does not exist or is not initialized
-2: an internal Windows error occurred
-3: an IOCTL error occurred

Remarks

This function does not affect the present state of the timer ('running' / 'stopped') or its programmed interval.

Requirements

DLL Version 1.0 or above, and a successful call of the function PciedioOpenCards

Example

```
//service the timer interrupt of board 2
// please check if the timer was in actual fact the source of the interrupt before the call
PciedioServiceTimerIrq(2);
```

PciedioGetTimerPresentState

Description

This function returns the present count value of the timer of a particular board, the state of the timer and its interrupt settings. It can be used to verify the settings made before and to receive the current count value.

Syntax

```
LONG PciedioGetTimerPresentState(
  BYTE   cardnum,
  DWORD *countval,
  BYTE  *start,
  BYTE  *irqen
);
```

Parameters

`cardnum [in]`

The index of the card corresponding to the jumper settings

`countval [out]`

A pointer to a 32-bit variable that receives the present count value of the timer. Each increment equals 100ns. As the timer is actually a cycle counter, it counts from 0 to the `interval` set up with PciedioSetTimerInterval and then restarts countig with 0. This has to be taken inot account when calculating time differences.

`start [out]`

A pointer to an 8-bit variable that receives the present state of the timer. It returns ‘0’ if the timer is stopped and ‘1’ if the timer is running.

`irqen [out]`

A pointer to an 8-bit variable that receives the present state of the local interrupt settings of the timer. It returns ‘0’ if the timer interrupt is disabled and ‘1’ if the timer innterupt is enabled.

Return value

0: the operation succeeded successfully
-1: the board addressed with `cardnum` does not exist or is not initialized
-2: an internal Windows error occurred
-3: an IOCTL error occurred

Remarks

This function neither starts, nor stops the timer or affects the programmed interval; it keeps a running timer running, a stopped timer stopped and the interval unchanged. Likewise, it does not affect any interrupt settings.

Requirements

DLL Version 1.0 or above, and a successful call of the function PciedioOpenCards

Example

```
//get the present state and count value of the timer of board 0
DWORD countval;
BYTE  start;
BYTE  irqen;

PciedioGetTimerPresentState(0, &countval, &start, &irqen);
```