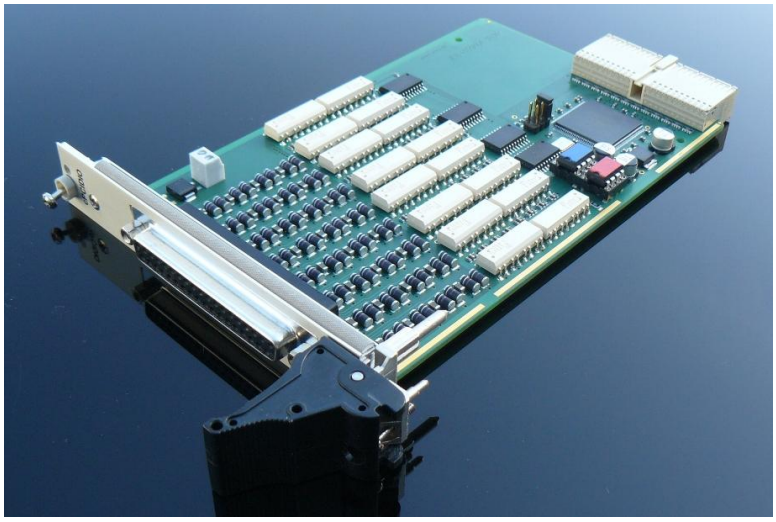
cPCIDIO

User Manual

Revision 3.5

RoHS Compliant
Directive 2005/95/EC



EBRU® GmbH • In den Kreuzwiesen 21 • D-69250 Schoenau • Germany
Tel:+49 6228 913710 • Fax:+49 6228 913729
www.ebru.de • info@ebru.de

Table of Contents

1 Introduction 4

1.1 Scope of Delivery 4
1.2 Description 4
1.3 System Requirements 5
1.4 Software Support 6
1.5 Notation in this Manual 6

2 Installation 7

2.1 Installation under Windows 7/Vista/XP (32/64 Bit) 7
2.2 Installation under Windows 2000 7

3 Hardware 8

3.1 Block Diagram 8
3.2 General Notes 8
3.3 Digital I/Os 9
3.3.1 External Power Supply 9
3.3.2 Outputs of the I/Os 10
3.3.3 Inputs of the I/Os 10
3.3.4 Short-Circuit Recognition 11
3.3.5 Outputs Watchdog 11
3.3.6 Controlling I/Os with Interrupts 11
3.4 Timer 12
3.5 Managing Interrupts 12
3.6 Address Jumper 12
3.7 General Data 13

4 Programming 14

4.1 Initialising the cPCIDIO 14
4.2 Simple Operation of the Digital Outputs 15
4.3 Operation of the Digital Outputs with Watchdog Enabled 17
4.4 Operation of the Digital Inputs 18
4.5 Digital Inputs as Interrupt Source 19
4.6 Handling the Timer with Interrupt Operation 20
4.7 Buffered sampling of the digital inputs 22
4.8 Driver Concept 24

5 API Reference 25

5.1 General Functions 26
pcidioGetCountBoards 26
pcidioInitCards 26
pcidioDeinitCards 27
pcidioGetSummaryOfAllBoards 27
pcidioGetBoardRevision 29
pcidioGetBoardAddressJumper 29
pcidioGetBoardConfigurationData 30
pcidioGetDriverVersion 30
pcidioGetPCIConfiguration 31

pcidioSetTimer ... 32
pcidioStartTimer ... 33
pcidioStopTimer ... 33
pcidioSetIRQTimer ... 34
pcidioSetWatchdogIntervall ... 34
pcidioGetWatchdogState ... 35
pcidioReset ... 36
pcidioGetErrorMsg ... 36
pcidioEnableIrq ... 36
pcidioDisableIrq ... 37
pcidioGetIrq ... 37
pcidioResetIrq ... 39
5.2 Digital Input Functions ... 40
pcidioDIGetChannelState ... 40
pcidioDIGetByte ... 40
pcidioDIGetState ... 41
pcidioDISetIrqChannelConfiguration ... 41
pcidioDIGetIrqChannelConfiguration ... 42
pcidioDIEnableBuffering ... 43
pcidioDIDisableBuffering ... 43
pcidioDIGetBufferedSamples ... 43
5.3 Digital Output Functions ... 45
pcidioDOSetChannelState ... 45
pcidioDOSetByte ... 45
pcidioDOSetState ... 46
pcidioDOServiceChannel ... 47

Appendix ... 48

A Pin Assignment 37-pin D-SUB Socket cPCIDIO ... 48
B PCIDIOHM Clamp Module ... 49
C DOS Driver ... 50
D Item Numbers ... 50
E Support ... 50
F Customised Models ... 50
G Service Address ... 51
H Updates ... 51

Published by ***EBRU*** ® GmbH, In den Kreuzwiesen 21, D-69250 Schönau, www.ebru.de

This documentation has been created to describe the use of hardware and software. The manufacturer may make technical changes for improving the product.

Important notice!
We must draw your attention to the fact that we cannot carry legal responsibility or assume any liability for consequences of incorrect use or software errors.

We always appreciate every notification of errors and all comments and suggestions for improvement etc.

1 Introduction

Dear Customer,

In purchasing the cPCIDIO, you have chosen a high-quality, technical product from EBRU GmbH that was in perfect condition when it left our factory.

Nevertheless, please check the completeness and condition of the package. Should anything be missing or defective, please notify us immediately.

Before you install the card, carefully read through the chapter on installation.

1.1 Scope of Delivery

We make every effort to deliver a complete product package. So that you can make sure you have received a complete package, we have listed the parts contained in the package below.

- cPCIDIO board
- User manual as *.pdf file on CD-ROM
- Driver software and demos on CD-ROM
- PCIDIOHM Clamp Module (optional)
- PCIDIOVK1M or PCIDIOVK2M connector cable for optional Clamp Module PCIDIOHM (optional)

1.2 Description

Item N°	I/Os 24V	Timer	Watchdog for outputs	Ext. supply	Extendable
cPCIDIO	32	Yes	Yes	Yes	Nein

The cPCIDIO offers in CompactPCI systems 32 optically isolated digital I/Os, optimised for 24VDC. Each digital I/O can be used as digital input or output and interrupt source as needed.

The 32 digital I/Os on the cPCIDIO card are connected to the card's 4 HP slot bracket by a 37-pin D-SUB socket. The card requires one free slot in a CompactPCI system with 3U form factor and CompactPCI connector J1 (110-pin, 32Bit).

Optionally, the PCIDIOHM Clamp Module with spring force connection for user friendly wiring and the corresponding PCIDIOVK1M or PCIDIOVK2M connector cable are available for the DIN standard rail.

Features
- 32 digital I/Os optically/galvanically isolated from the computer and optimised for 24VDC
- Each I/O freely usable as an input or output and interrupt source
- Galvanic isolation voltage min. 1500Vrms

Outputs
- Max. 1A output current per channel
- Direct connection of resistive, capacitive or inductive loads
- Permanently short-circuit proof with automatic restart attempts and overvoltage protection
- Short-circuit recognition for diagnostic purposes
- Programmable, computer-independent watchdog for the outputs

Inputs
- Switching threshold optimised for 24VDC
- Input current at 24VDC approx. 3 mA
- Each input can trigger interrupts, with programmable edge
- RC input filter and digital filter with 10KHz cut-off frequency
- Unused inputs can be left open

Other features
- Programmable 24 bit, 10 MHz timer with interrupt operation
- 3 additional jumpers for distinguishing up to 8 cards within the same system
- 32 Bit CompactPCI short card (Universal Card for 5V/33MHz and 3.3V/66MHz slots) with 4 HP for 3U form factor CompactPCI systems
- External supply of output transistors and the optical isolation via the connector of the respective card with 24VDC +/- 30%
- Optional clamp module with spring force connection for user-friendly wiring
- Comprehensive software for Windows 7(64/32 Bit), Vista (64/32 Bit), XP (64/32 Bit), 2K and DOS included
- Customised modifications and drivers possible upon request
- RoHS compliant according to Directive 2002/95/EC
- Also available as **software-compatible PCI Bus board PCIDIO**

1.3 System Requirements

The cPCIDIO board requires a computer with a X86 processor or compatible computer.

The card requires one free 32 Bit slot (4 HP) with 5V/33MHz or 3.3V/66MHz in the CompactPCI system with 3U form factor and CompactPCI connector J1 (110-pin, 32Bit).

1.4 Software Support

Please read the Readme files in the current software package to find out what software is included.

There are drivers, demo programs, tools for commencing use and source code available for Windows and DOS.

1.5 Notation in this Manual

Function names	are always bold and italicised,
<TimesNewRoman>	parameters are in angle brackets,
[]	physical units are in square brackets
TimesNewRoman	source code segments are italicised
🗐	description of driver functions
☝	notes on specific or special use
🕮	tips for use
⇔	parameters for functions
⇨	function input parameters
⇦	function output parameters
⇦	function return values

2 Installation

Before installing the card, carefully read your computer's manual on the installation of expansion cards and then follow the installation procedure described in this manual. Make sure the slot bracket of the card is screwed onto and earthed through the computer case.

> ***Note***
> *If an already present driver has to be replaced, please use in any case the corresponding uninstall.bat of the driver package before installation of a new driver.*

2.1 Installation under Windows 7/Vista/XP (32/64 Bit)

1. Install the CompactPCI card into a free CompactPCI slot in your computer. Careful!! You must disconnect your computer from the mains supply since some CPU-boards are still powered even if the computer is supposed to be switched off.
2. Switch the computer on and launch Windows.
3. The hardware wizard will launch automatically. Close the wizard by clicking 'Cancel' as the driver cannot be installed using the wizard and open the file manager (e.g. Explorer).
4. Change to the drive where the installation files are located
5. Change to the directory *Win_32 (32 Bit) / Win_64 (64Bit)*
6. Run the installation routine *Install_x32 (32Bit) / Install_x64 (64 Bit)*. Now the driver installation is accomplished automatically. Follow the instructions of the operating system around the installation. The card is ready to go when the installation has finished.
7. Before using the card you should reboot your computer.
8. To test the installation you can now use the sample programs for demonstration and start-up.

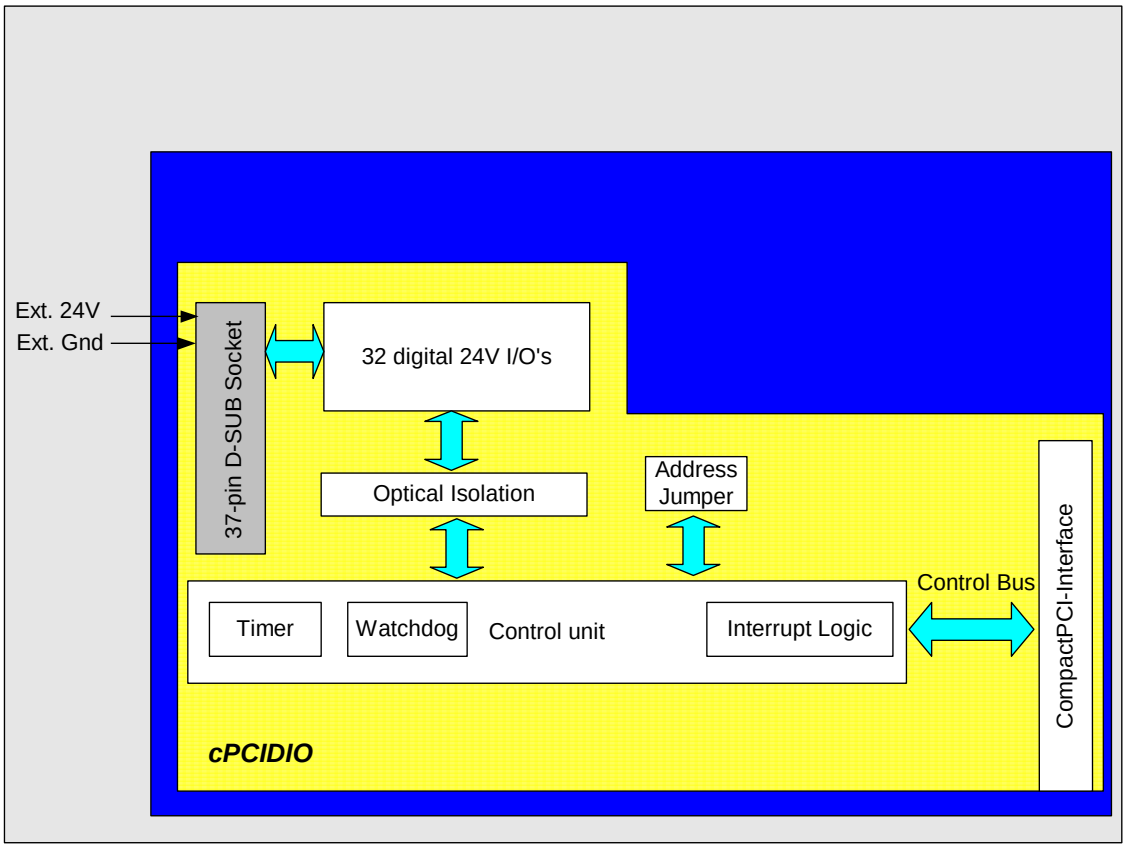
2.2 Installation under Windows 2000

1. Install the CompactPCI card into a free CompactPCI slot in your computer. Careful!! You must disconnect your computer from the mains supply since some CPU-boards are still powered even if the computer is supposed to be switched off.
2. Switch the computer on, launch Windows 2000 and log in as "Administrator".
3. The hardware wizard will launch automatically. Close the wizard by clicking 'Cancel' as the driver cannot be installed using the wizard and open the file manager (e.g. Explorer).
4. Change to the drive where the installation files are located.
5. Change to the directory *Win32.*
6. Run the installation routine *Install_x32*. Now the driver installation is accomplished automatically. Follow the instructions of the operating system around the installation.
7. After the installation has finished you must reboot your computer.
8. After reboot the card is ready to go. To test the installation you can now use the sample programs for demonstration and start-up.

3 Hardware

3.1 Block Diagram

The block diagram below shows the layout of the cPCIDIO.



3.2 General Notes

The card and all connectors may only be connected when powered off.

Make sure the slot bracket of the card is screwed onto and earthed through the computer case. For connections outside the computer, use exclusively shielded cable and make sure the shielding is earthed.

Make sure when touching the card or connecting the connector cable that no static discharge can occur over the card. Make sure the connector cable is properly inserted, otherwise the card may not function properly.

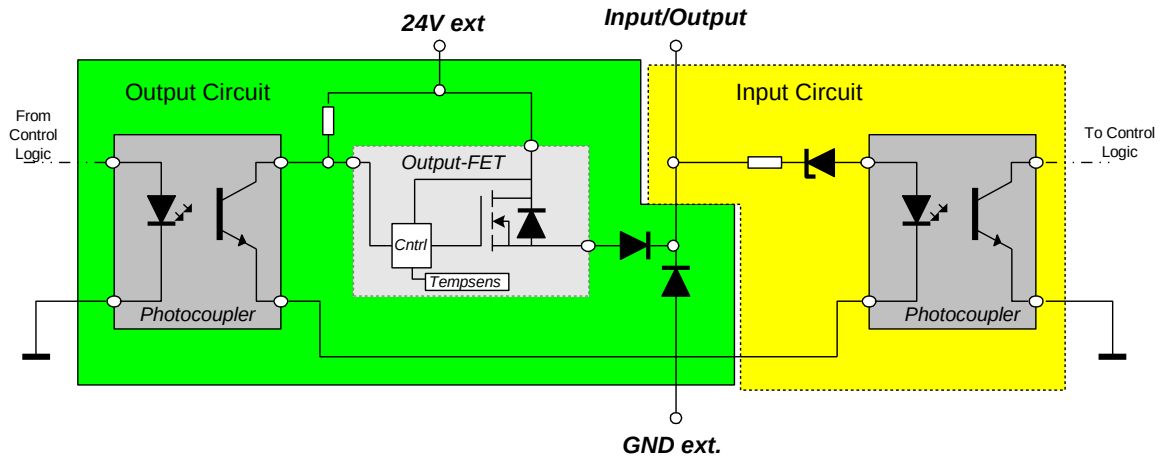
3.3 Digital I/Os

The cPCIDIO card has 32 freely configurable inputs and outputs. All I/Os can be configured independently of one another as input or as output with read back. The software sets the direction of the ports and their interrupt functionality.

> ***Note***
> *As soon as a port is set to '1', the correspondingly allocated pin is set to High (24V) and the IO serves as output.*
>
> ***Ports used as input may never be set to '1'.***
>
> *If a port is read, the level of the correspondingly allocated pin is read back ('0' for GND or '1' for 24V). Outputs can also be read back.*

After switching on the computer or after a reset, all outputs are programmed to '0' and the interrupt functionality is deactivated.

> ***Tip***
> *For programming, read the corresponding chapter of this manual*



3.3.1 External Power Supply

The external power supply of the I/Os and the optical isolation is provided by the D-SUB connector to the +24VDC and GND with 24VDC +/- 30% connections.

> **Tip**
> *We urgently recommend you also apply the power supply to all available connections on principle.*

> **Note**
> *Do not exceed the total current of 5A on the card.*

3.3.2 Outputs of the I/Os

The optically isolated and permanently short-circuit-proof outputs of the I/Os are each built with Power MOSFET transistors and have internal temperature monitoring that automatically switches off the respective output in the case of overload or short circuit, and switches it back on when the MOSFET depletion layer temperature falls below the 150°C threshold.
Each output also has a 1A freewheeling recovery diode between the respective output and GND and a diode in series to protect against reverse currents. Thus, resistive, inductive and capacitive loads can be connected directly. Due to the diode protection and the drain source resistance of the MOSFET, the voltage drop at each output is between 0.6V and 2V, depending on load. The load switching capabilities and switching frequency of the outputs are by nature strongly dependent on the type and size of the respective load.

Technical Data Output	
Max. peak current	1.2 A
Max. continuous current	1 A
Guaranteed continuous current	0.65 A at purely resistive load
Switching frequency, max.	10 KHz
Switching frequency, typical	500 Hz at 1A and purely resistive load
	1 KHz at 0.25 A and purely resistive load
Turn on time (typ./max.)	45 us/125 us at 270 Ohm load
Turn off time (typ./max.)	40 us/175 us at 270 Ohm load
Min. isolation voltage of optocoupler	5 000 Vrms

3.3.3 Inputs of the I/Os

Optocouplers are implemented to optically isolate the inputs of the I/Os and are supplied with power from the external power supply on the respective male connectors. A diode in series and a resistor in series for limiting the current set the switching threshold; a downstream RC-network and a digital filter serve to suppress interference and transients.

.

Technical Data Input	
Switching threshold typ. [max]	12 VDC +/- 10% [12 VDC +/-30%]
Switching frequency	> 10 KHz
Input current	(Vin – 8.2 V – 1.15 V)/ (4700 Ohm)
Max. input voltage Vin	31.8 VDC
Min. isolation voltage of optocoupler	5 000 Vrms

3.3.4 Short-Circuit Recognition

The read-back of an I/O used as output can be exploited for short circuit recognition.

If, after setting an I/O and a delay of at least 250µs in addition to any load-dependent delay, the corresponding I/O is read back, it must return a '1'. If a '0' is returned instead, then an external short-circuit exists.

3.3.5 Outputs Watchdog

The output transistors on the card can be monitored by a common, computer-independent watchdog, the timeout period can be programmed to be between 26.21ms and 6.68 seconds. After switching on the computer or restarting the software, the watchdog will be disabled.

When the watchdog is enabled, if there is not at least one output on the cPCIDIO card accessed with a write within the programmed timeout period, then all output transistors will be immediately reset.

After programming, the timeout cannot be changed and the watchdog cannot be disabled again. The watchdog will only be disabled and the value made settable again after a software card reset or after rebooting the computer.

Whether the watchdog has tripped can be checked with the driver.

3.3.6 Controlling I/Os with Interrupts

Each input can be used as a separate interrupt source. They are programmed by functions of the supplied software driver.

The individual interrupts of the inputs are configured, enabled and disabled again by software.

There are two different edge configurations available for the inputs. It is possible to trigger an interrupt upon a rising or a falling edge at the input contact.

After a hardware reset, all interruption settings are deleted and the falling edge set as default triggering edge.

> ***Note***
> *Further details on I/O interrupts can be found in the chapters on software programming.*

3.4 Timer

There is a 24 bit timer integrated on the cPCIDIO card to cyclically generate interrupts.

Technical Data Timer	
Resolution	24 Bit
Smallest interval	200ns
Largest interval	1.6777217s
Interrupt	Interrupt creation upon zero-crossing, configurable by software

One clock cycle is 100ns, so a maximum interval of 1.6777217 seconds can be set.

An interrupt can be triggered upon every zero-crossing of the timer, if so enabled and set by the software.

> ***Note***
> *Further details on the timer can be found in the chapters on software programming.*

3.5 Managing Interrupts

The cPCIDIO offers several interrupt sources that are managed by interrupt sharing on the card. There are configuration, enabling and disabling functions for each of the individual card-internal interrupts.

There are also configuration, enabling and disabling functions available for the PCI interrupt used by the card. Also, the user can transfer an interrupt handler function of his own to the PCI interrupt.

After a hardware reset, the interrupt functionality is disabled and not configured.

> ***Note***
> *Further details on interrupt programming can be found in the chapters on software programming.*

3.6 Address Jumper

So as to distinguish multiple cards of the cPCIDIO within the same computer, there are three jumpers integrated on the cPCIDIO card. That way, eight cards, with a total of maximum 256 I/Os can be distinguished.

The jumpers are labelled S2, S1 and S0 on the card. The position of the switches can be queried using software functions.

The driver uses the position of the jumper for addressing if several cards are present.

S2	S1	S0	Card
0	0	0	1
0	0	1	2
0	1	0	3
0	1	1	4
1	0	0	5
1	0	1	6
1	1	0	7
1	1	1	8

3.7 General Data

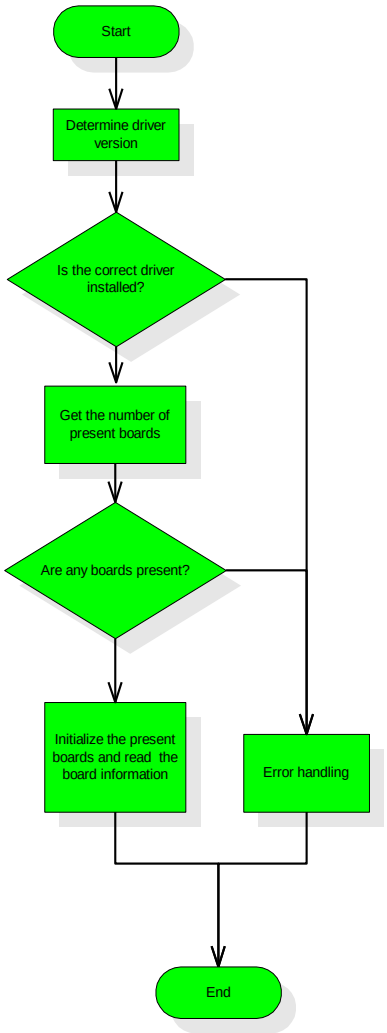
Technical Data General	
Dimensions	3U 160mm x 100mm (without connector and slot bracket) with 4 HP slot bracket
cPCIDIO card connection	37-pin D-SUB Socket
Power supply of inputs and outputs	External power supply over the connector
Operating temperature	0..70°C
Storage temperature	-40...100°C
Rel. humidity	0...90% (non-condensing)

4 Programming

This chapter shows how the cPCIDIO card can be programmed using the Windows driver and the API reference. The programming is illustrated in the form of flow charts and C source code. All examples listed here are exclusively for the purpose of demonstrating the functions.

4.1 Initialising the cPCIDIO

This subchapter shows a way to specify the cards present in the system and to read their addressing data.

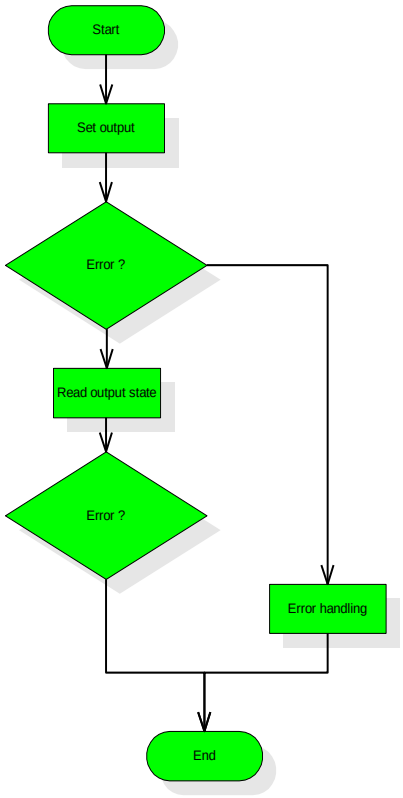



```
...
unsigned int l_uiDriverVersion;          /* version of the installed driver*/
PCIDIO_SUMMERY  l_SummeryBuf[4];         /* buffer for the board information*/
BOOL l_bReturnValue;                     /* return value of the functions*/
char l_strErrorText[100];                /* error message*/
int l_iCntCards;                         /* number of present boards */
...
/* get driver version */
l_bReturnValue = pcidioGetDriverVersion(&l_uiDriverVersion);
/* Is the correct driver installed?*/
If((l_bRetunValue == TRUE)&&
  (l_uiDriverVersion == ACT_DRIVER_VERSION))
{
  /*get  the number of  present boards*/
  l_iCntCards=pcidioGetCountBoards()
  /*are boards present ? */
  if(l_iCntCards>0)
  {
    /*initialize the present boards*/
     l_bReturnValue = pcidioInitCards(&l_iCntCards);
      /* read the board informations from all present boards*/
      l_bReturnValue=pcidioGetSummeryOfAllBoards(l_SummeryBuffer);
   }
}
/*error ?*/
if(l_bReturnValue == FALSE)
{
  /*get error message*/
  pcidioGetErrorMsg(l_strErrorTxt);
}
...
```

4.2 Simple Operation of the Digital Outputs

As an introduction to using the card, operation of the outputs without employing the watch-dog will be described here.

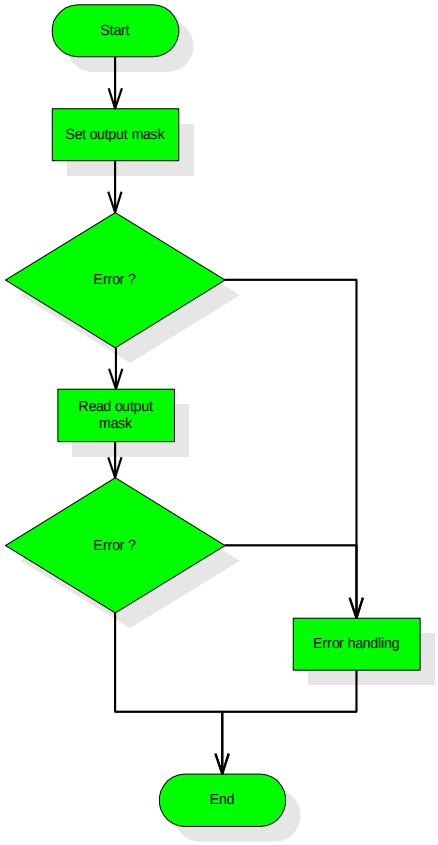
There are principally two ways to operate the outputs. The first way, shown here, is to set a special output.




```
...
unsigned char l_ucActIndex;                 /*board address*/
unsigned char l_ucSetOutputState=1;         /*state of the output*/
unsigned int l_ucNewOutputState;            /*new state of the output*/
unsigned char l_ucChannel;                  /*output channel to change*/
...
/*take the address of the current board from the board information*/
l_ucActIndex=l_SummeryBuffer[0].BoardNumber;
/*set an output */
l_bReturnValue=pcidioDOSetChannelState(l_ucActIndex, /*board address*/
                                       l_ucChannel,  /*output to change*/
                                       l_ucSetOutputState);/*output state*/
/*error ?*/
if(l_bReturnValue==TRUE)
{
 /* wait min 250µs due to the switching delay of the output transistors and input filter (if*/
 /*necessary with load-sensitive delay(here 1ms))*/
 Sleep(1);
 /*read the state of the output */
 l_bReturnValue=pcidioDIGetChannelState(l_ucActIndex,  /*board address number */
                                        l_ucChannel,    /*output to read*/
                                       &l_ucNewOutputState);/*new state*/
}
/*error ?*/
if(l_bReturnValue==FALSE)
{
 /*get error message */
 pcidioGetErrorMsg(l_strErrorTxt);
}
…
```

Note
The function Sleep() can be replaced by a Delay function that waits minimum 250µs in addition to any load-dependent delay.

The second way to operate the outputs is to set all output channels with a certain pattern.

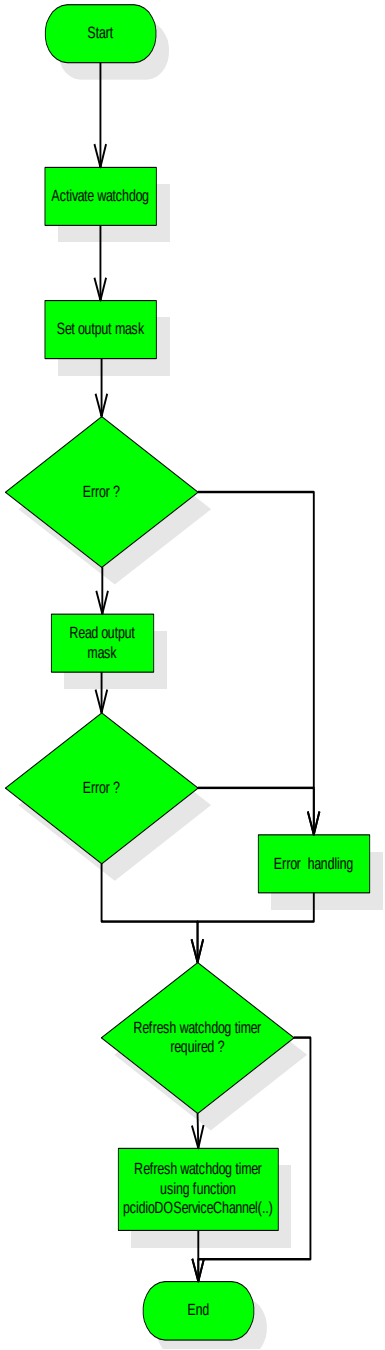



```
...
PCIDIOALLCHANNELS l_OutputStates;        /*state of the outputs */
PCIDIOALLCHANNELS l_NewOutputState;   /*new state of the outputs*/
...
/* take the address number of the current board from the board information*/
l_ucActIndex = l_SummeryBuffer[0].BoardNumber;
/*set the mask of the outputs*/
l_OutputState.Basis = 0xAAAAAAAA;
l_OutputState.Extension = 0x55555555;
/*set the outputs*/
l_bReturnValue = pcidioDOSetState(l_ucActIndex, /*board address number*/
                                  l_OutputState);/*output state mask*/
/*error ?*/
if(l_bReturnValue == TRUE)
{
/* wait min 250µs due to the switching delay of the output transistors and input filter (if*/
/*necessary with load-sensitive delay(here 1ms))*/
  Sleep(1);
  /*read state of the outputs*/
  l_bReturnValue = pcidioDIGetState(l_ucActIndex,  /*board address number*/
                                    &l_NewOutputState); /*output state mask*/
}
/*error ?*/
if(l_bReturn == FALSE)
{
  /*get error message*/
  pcidioGetErrorMsg(l_strErrorTxt);
}
…
```

Note
The function Sleep() can be replaced by a Delay function that waits minimum 250µs in addition to any load-dependent delay.

4.3 Operation of the Digital Outputs with Watchdog Enabled

Now we shall describe the operation of the outputs with the watchdog enabled. In this example, all outputs will be given a pattern; the activation of an individual output is done in a similar manner.




```
...
unsigned char l_ucWatchdogIntervall;            /*watchdog interval*/
unsigned char l_ucWatchdogService;              /*flag for watchdogservice*/
PCIDIOALLCHANNELS l_OutputStates;               /*state of the outputs*/
PCIDIOALLCHANNELS l_NewOutputState;             /*new state of the outputs*/
...
/*set the output mask*/
l_OutputState.Basis=0xAAAAAAAA;
l_OutputState.Extension=0x55555555;
…
/*start watchdog*/
l_bReturnValue = pcidioSetWatchdogIntervall(l_ucActIndex,
                                            l_ucWatchdogIntervall);
if(l_bReturnValue==TRUE)
{
 /*set outputs*/
 l_bReturnValue=pcidioDOSetState(l_ucActIndex, /*board address number*/
                                 l_OutputState);/*state of the outputs*/
 /*function returns with no error?*/
 if(l_bReturnValue==TRUE)
 {
/* wait min 250µs due to the switching delay of the output transistors and input filter (if*/
/*necessary with load-sensitive delay(here 1ms))*/
   Sleep(1);
  /*read state of the outputs*/
   l_bReturnValue=pcidioDIGetState(l_ucActIndex,  /*board address number*/
                                   &l_NewOutputState);/*new state of the outputs*/
 }
}
/*error ?*/
If(l_bReturnValue==FALSE)
{
 /*get error message*/
 pcidioGetErrorMsg(l_strErrorTxt);
}
…
/*flag for the watchdogservice*/
l_ucWatchdogService= TRUE;
…
/*is the watchdogserviceflag set*/
if(l_ucWatchdogService==TRUE)
{
 /*set outputs again*/
 pcidioDOServiceChannel(l_ucActIndex);
}
...
```

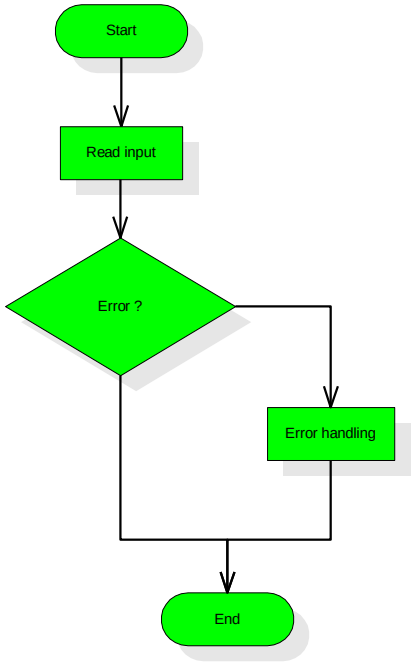
The flag l_ucWatchdogService is set in the program sequence to TRUE and checked. The program must call the watchdog service function at least once within the programmed timeout period, otherwise the outputs will be reset by the hardware.

It is still possible to query the status of the watchdog before calling the watchdog service function. That way it can be recognised whether the watchdog has tripped since the last operation of the outputs.
Should that be the case, an error has occurred in the user program that prevented the timely service of the watchdog.

4.4 Operation of the Digital Inputs

Next, we shall demonstrate the read-in of the digital inputs. Again, it must be noted here that there are two options: individually accessing a certain channel and collectively accessing all channels.

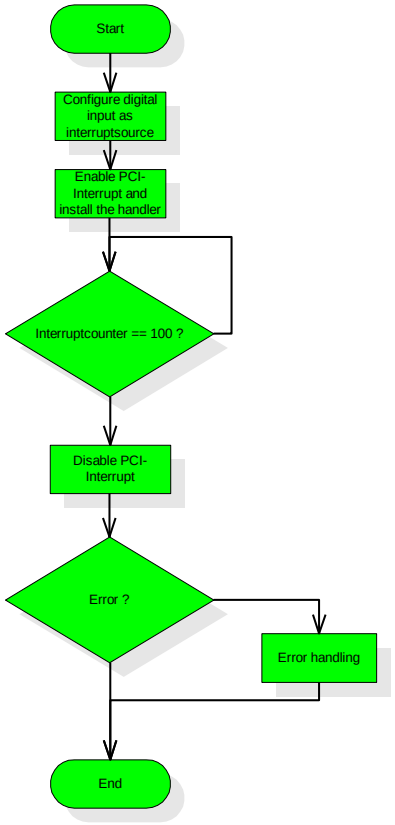
This chapter deals with individual access by way of example. Collective access follows the same method, except with the help of another function call.



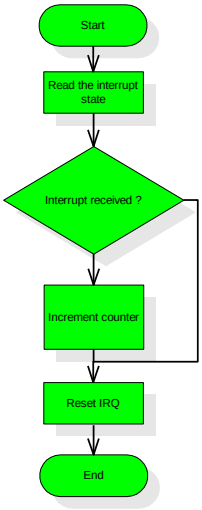

```
...
unsigned char l_ucInputState;     /*state of the input*/
unsigned char l_ucChannel;        /*number of the input*/
...
/* read the input */
l_bReturnValue = pcidioDIGetChannelState(l_ucActIndex , /*board address number*/
                                         l_ucChannel , /*number of the input*/
                                        &l_ucInputState);  /*state of the input*/
/*error ?*/
if(l_bReturnValue!=0)
{
  /*get error message */
  pcidioGetErrorMsg(l_strErrorTxt);
}
...
```

4.5 Digital Inputs as Interrupt Source

This subchapter describes the use of an input channel as interrupt source. A channel is configured as interrupt source upon rising edge and a counter in the user interrupt handler is incremented upon every rising edge.




```
...
unsigned char g_ucIntCounter;/*global interrupt counter*/
...
unsigned char l_ucActIndex;
unsigned char l_ucChannel;
...
/*configure digital input as interrupt source*/
l_bReturnValue = pcidioDISetIrqChannelConfiguration(l_ucActIndex,
                                                    l_ucChannel,
                                                    1,
                                                    1);

if(l_bReturnValue  == TRUE)
{
 /*enable PCI Interrupt and install interrupt handler*/
 l_bReturnValue= pcidioEnableIRQ(l_ucActIndex,
                                 &Inthandler);
 if(l_bReturnValue==TRUE)
 {
   g_ucIntCounter=0;
   while(g_ucIntCounter<100)
  {
    Sleep(1);
  }
 /*disable PCI-Interrupt*/
  l_bReturnValue=pcidioDisableIrq(l_ucActIndex);
 }
}
/*error ?*/
if(l_bReturnValue ==FALSE)
{
 /* get error message */
 pcidioGetErrorMsg(l_strErrorTxt);
}
...
```

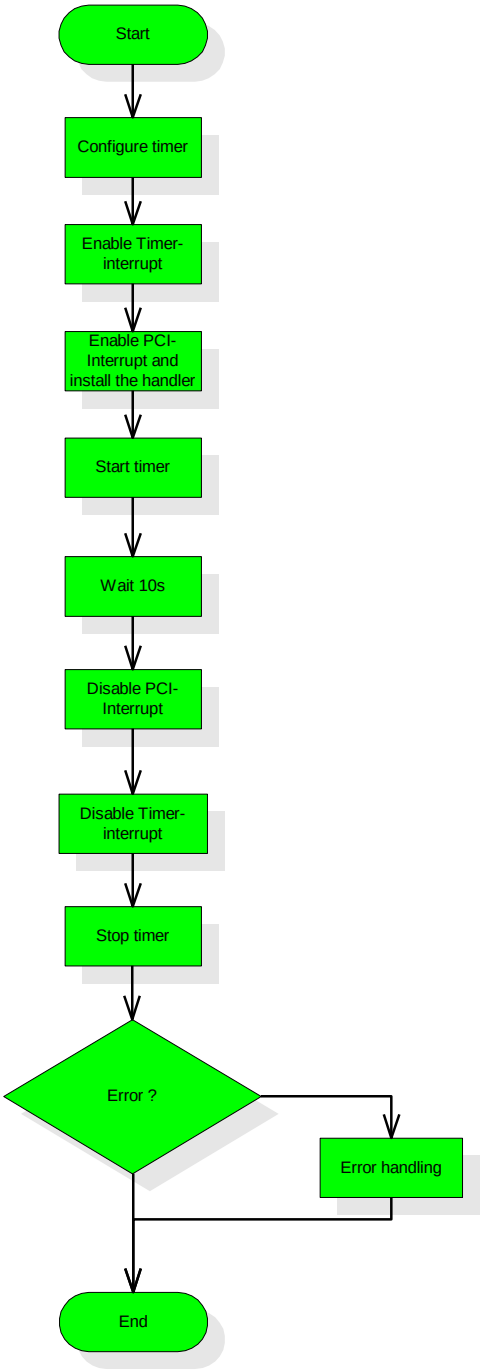


Interrupthandler

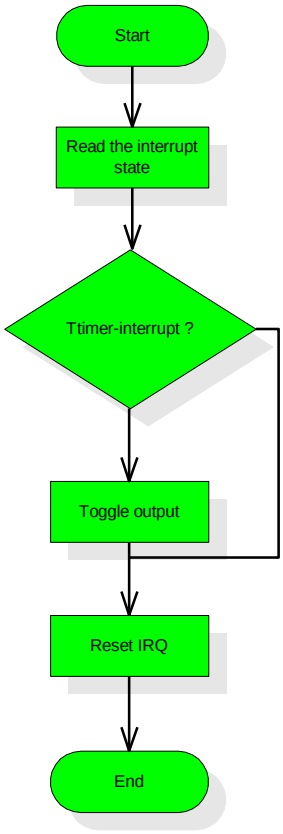
```
PCIDIO_INT_HANDLER  Inthandler(void)
{
  unsigned char l_ucActIndex;
 PCIDIO_INT_STATE   l_IntState;
  ...
/*read the interrupt state*/
pcidioDIGetIrq (l_ucActIndex,
                &l_IntState);

 if(l_IntState.IRQIO_1_32!=0)
{
  g_ucIntCounter++;
}
pcidioResetIRQ(l_ucActIndex)

...
}
```

4.6 Handling the Timer with Interrupt Operation

Here we describe timer programming with interrupt handling. In the interrupt handler of the timer programming, an output of the cPCIDIO will be switched in the interrupthandler upon a timer interrupt.




```
...
unsigned long l_ulTimerIntervall:
…
/*stop timer*/
 pcidioStopTimer(l_ucActIndex);
/*configure timer*/
l_bReturnValue =  pcidioSetTimer(l_ucActIndex,
                                 l_ulTimerIntervall)
if(l_bReturnValue == TRUE)
{
   /*enable PCI-Interrupt and install interrupt handler*/
   l_bReturnValue = pcidioEnableIrq(l_ucActIndex,
                                    &IntHandler);
  if(l_bReturnValue == TRUE)
  {
   /*enable timer-interrupt*/
   l_bReturnValue = pcidioSetIRQTimer(l_ucActIndex,1);
   if(l_bReturnValue == TRUE)
   {
    /*start timer*/
    l_bReturnValue = pcidioStartTimer(l_ucActIndex);
    if(l_bReturnValue == TRUE)
    { /*wait 10s*/
      Sleep(10000);
    }
     /*disable PCI-Interrupt*/
     l_bReturnValue = pcidioDisableIrq(l_ucActIndex);
    if(l_bReturnValue == TRUE)
    {
     /*disable timer-interrupt */
     l_iReturnValue = pcidioSetIRQTimer(l_ucActIndex,0);
     if(l_bReturnValue == TRUE)
     {
       /*stop timer*/
       l_bReturnValue = pcidioStopTimer(l_ucActIndex);
     }
   }
  }
  }
}
/*error ?*/
if(l_bReturnValue == FALSE)
{
 /* get error message */
 pcidioGetErrorMsg( l_strErrorTxt);
}
…
```



Interrupthandler

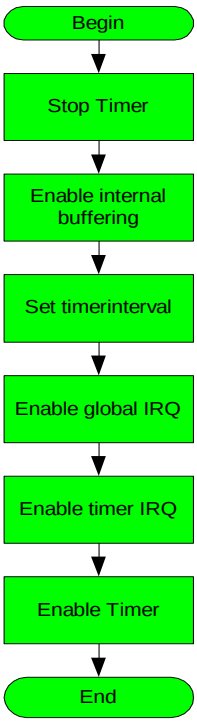
```
void Inthandler(void)
{
 unsigned char l_ucActIndex;
 unsigned char l_ucState;
 PCIDIO_INT_STATE   l_IntState;
  ...
 /*read the interrupt state*/
 pcidioDIGetIrq (l_ucActIndex,&l_IntState);

 if(l_IntState.IRQTIMER & 0x02 !=0)
 {
   /*read output*/
   pcidioDIGetChannelState(l_ucActIndex, 1,&l_ucState);
  /* toggle state and set output*/
   pcidioDOSetChannelState(l_ucActIndex, 1, ~l_ucState);
 }
 pcidioResetIRQ(l_ucActIndex)

  ...
}
```

4.7 Buffered sampling of the digital inputs

Starting with driver revision 3.1, a time-controlled buffered sampling of the digital inputs is applicable. An internal buffer organized as a ring memory is filled with the help of the on board timer of the base card used as a trigger source. To each sample a corresponding timestamp in timer-ticks is also saved.

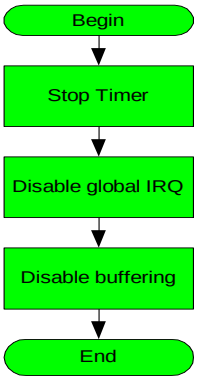
This function allows a buffered sampling of the digital inputs with a programmable sampling rate without any further interaction of the application. The sampling rate is constant as the on board timer of the base card is used as the trigger source and therefore is also independent of the application.

> ***Note***
> *The sampling rate of the time-controlled sampling of the digital inputs depends on the achievable service rate of the timer interrupt by the Software. The service rate depends again on the BIOS, the PC, the Operating System and the configuration of the system and has to be confirmed by the user.*



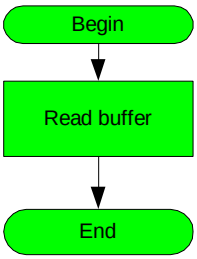
Enable the buffered sampling

```
void InitInputSample(void)
{
  // Stop card timer
  pcidioStopTimer(ucActIndex);

  // Enable internal buffering
  pcidioDIEnableBuffering(ucActIndex);

// Set Timerintervall to 1ms
pcidioSetTimer(ucActIndex,9999);

// Enable global  card interrupt and set dummy interrupt routine
pcidioEnableIrq(ucActIndex,PCIDIO_IntHandlerRoutine1);

//Enable timerinterrupt
pcidioSetIRQTimer(ucActIndex,1);

// Start timer
pcidioStartTimer(ucActIndex);
}
```



Disable the buffered sampling

```
void DeinitInputSample(void)
{
  // Stop card timer
  pcidioStopTimer(ucActIndex);

  // Disable Interrupt
  pcidioDisableIrq(ucActIndex);

  // Disable internal buffering
  pcidioDIDisableBuffering(ucActIndex);
}
```



Read out the buffer

```
void ReadInputSample(void)
{
   DWORD CountActSamples=0;
   DWORD BufferOverflow=0;
   DWORD bufferSize=1000;
   PCIDIOCHANNELS Samples[1000];
   LARGE_INTEGER TimeStamp[1000];

   // Read input samples buffer
   pcidioDIGetBufferedSamplesStdCall(ucActIndex, bufferSize, Samples,
                                     TimeStamp, &CountActSamples,  &BufferOverflow);
}
```

4.8 Driver Concept

The Windows 7 (32 Bit and 64 Bit), Vista (32 Bit and 64 Bit), XP (32 Bit and 64 Bit) and 2K (32 Bit) driver for the cPCIDIO is implemented as WDM driver and comprises the following components:

WDM-driver kp_pcidi.sys for Windows 7/Vista/XP und 2K
API-DLL pcidio.dll for Visual C++ with cdecl and stdcall declarations.

These Software components are available for 32 Bit and 64 Bit systems.

As for 64 Bit systems the programmer has to take care which application has to be developed. For a 32 Bit application the DLL pcidio_32_64.dll and for a 64 Bit application the DLL pcidio_64_64.dll must be used along with the respective Lib-files. During compilation for 64 Bit systems the preprocessor directive KERNEL_64Bit must be used.

The API does not differ for the several systems, only the internal data processing differs in its implementation.

The tool PCIDIODEMO is included as source code and is for supporting and displaying the programming of the individual card functions under Visual C++.

For DOS applications, there is a driver included as C source code.

5 API Reference

This chapter describes the programming interface of the cPCIDIO under Windows, which the application programmer has available.

> **Tip**
> *Only use the functions documented here. Using undocumented features can destroy the card or the hardware connected to it. Also it can happen that these functions will no longer be supported in the next version.*

- Function prototypes:
  In the following function descriptions, the function prototypes for VC++ will be used.

- Blocking function calls
  Please note that program execution will only be continued if a function call has been executed completely. We shall refer to this below as BLOCKING.

- Parameters
  Parameter upon input and output stands for assigned variables, data arrays or pointers to these.

- Nomenclature
  All function names have the family prefix *"pcidio"* and a function group prefix

  "" -> General Functions
  "DI" -> Digital Inputs
  "DO" -> Digital Outputs

  For the most part, the function names use "self-explanatory" descriptions.

  There is no postfix used for the function declaration _cdecl. The function declaration _stdcall has the postfix StdCall added to its name.

  This makes it also possible to use other programming languages with which a DLL can be integrated.

5.1 General Functions

pcidioGetCountBoards

Description

Returns the number of cPCIDIO cards found present in the system and initialises the driver accordingly.

> ***Note***
> *This function should be run at the beginning of an application so as to determine whether there are any cards present at all.*

⇔ Parameter

⇨ Input

none

⇦ Output

none

Return

If the function was run successfully, it returns the number of cards found or 0 for no cards.

pcidioInitCards

Description

This function initialises all cPCIDIO cards present in the system and arranges them according to the position of the address jumper.

> ***Note***
> *This function should be run at the beginning of an application so that all cards are initialised correctly.*

⇔ Parameter

⇨ Input

none

⇦ Output

Number of initialised cards

Return

If the function was run successfully, it returns TRUE, otherwise FALSE.

pcidioDeinitCards

Description

This function deinitialises all cPCIDIO cards present in the system and deletes the memory occupied by the driver.

> ***Note***
> *This function should be run at the end of an application in order to free up the memory occupied by the driver and to delete the interrupt handler.*
>
> *Before calling this function, we recommend additionally either to globally reset the card in the user program or at least to delete the outputs and reset the interrupt configurations.*

⇔ Parameter

⇨ Input

none

⇦ Output

none

Return

none

pcidioGetSummaryOfAllBoards

Description

Gives an overview of the cards present. This overview contains the location of the card in the system, the position of the card addressing jumper and the basis address of the card in the I/O range and the card number. Given this data, it is possible to use the cards uniquely in the program.

⇔ Parameter

⇨ Input

<*SummaryBuffer*>
Pointer to the data buffer for the overview data. The pointer must show a sufficiently large data array of *PCIDIO_SUMMERY* type.

```
typedef struct PCIDIO_SUMMERYSummerybuffer
 {
   PCIDIO_HANDLE hPCIDIOHandle;
   BYTE BoardIndex;
   BYTE BoardNumber;
   BYTE SlotNumber;
   BYTE BUSNumer;
   BYTE BoardAdressJumper;
   DWORD BoardIOAdress;
 };
```

<hPCIDIOHandle>
Handle on the cPCIDIO card for internal use.

<BoardIndex>
Index for addressing the cPCIDIO card. This element is required for addressing the card in all functions.

<BoardNumber>
Index for addressing the cPCIDIO card. This element can be used alternatively for addressing the card in all functions.

<SlotNumber>
Number of the slot in which the card is located.

<BusNumber>
Number of the bus on which the card is located.

<BoardAdressJumper>
Position of the address jumper on the card specifically for primary distinction of the individual cards.

<BoardIOAdress>
Address of the card in the I/O range of the computer system.

> ***Note***
> *The data buffer must be applied by the application developer and passed on to the function.*
> ***Tip***
> *The data buffer should always be able to accept eight card overviews.*

⇦ Output

The filled data buffer.

Return

If the function was run successfully, it returns TRUE. If an error has occurred while running, it returns FALSE.
The error can be identified using the function *pcidioGetErrorMsg*.

pcidioGetBoardRevision

Description

Returns the revision number of the hardware.

⇔ Parameter

⇨ Input

<BoardNumber>
cPCIDIO addressing index, detected by the function *pcidioGetSummaryOfAllBoards*.

⇦ Output

<BoardRevision>
Revision of the hardware in hexadecimal, e.g. 02h.

Return

If the function was executed successfully, it returns TRUE. If an error has occurred while running, it returns FALSE.
The error can be identified using the function *pcidioGetErrorMsg*.

pcidioGetBoardAddressJumper

Description

Returns the position of the address jumpers S0, S1 and S2.

⇔ Parameter

⇨ Input

<BoardNumber>
cPCIDIO addressing index, detected by the function *pcidioGetSummaryOfAllBoards.*

⇦ Output

<BoardAdressJumper>
Position of the address jumper on the card specifically for primary distinction of the individual cards. It returns values in the range of 0...7.

Return

If the function was executed successfully, it returns TRUE. If an error has occurred while running, it returns FALSE.
The error can be identified using the function *pcidioGetErrorMsg*.

pcidioGetBoardConfigurationData

Description

Returns the individual configurations of the addressed cPCIDIOs.

⇔ Parameter

⇨ Input

<BoardNumber>
cPCIDIO addressing index, detected by the function *pcidioGetSummaryOfAllBoards.*

⇦ Output

<CntChannel>
Number of available digital channels. Always 32 is returned.

Return

If the function was executed successfully, it returns TRUE. If an error has occurred while running, it returns FALSE.
The error can be identified using the function *pcidioGetErrorMsg*.

pcidioGetDriverVersion

Description

Returns the card driver version.

> **Tip**
> *With this function, you can check whether the correct driver version is being used or not.*

⇔ Parameter

⇨ Input

none

⇦ Output

<DriverVersion>
Version of installed driver.

Return

If the function was executed successfully, it returns TRUE. If an error has occurred while running, it returns FALSE.
The error can be identified using the function *pcidioGetErrorMsg*.

pcidioGetPCIConfiguration

Description

Returns the PCI configuration data of the selected card.

⇔ Parameter

⇨ Input

<BoardNumber>
cPCIDIO addressing index, detected by the function *pcidioGetSummaryOfAllBoards.*

<PCIConfiguration>
Pointer to a structure of the type *PCIHEADER*, which the function fills with data.

```
typedef struct PCIHEADER
 {
   unsigned int DeviceID;
   unsigned int VendorID;
   unsigned int StateReg;
   unsigned int ControlReg;
   unsigned long ClassCode;
   unsigned char RevisionID;
   unsigned char HeaderType;
   unsigned long BaseAdress;
   unsigned int SubsysID;
   unsigned int SubVenID;
   unsigned char IrqPin;
   unsigned char IrqLine;
 };
```

<DeviceID>
Describes the function group in which the cPCIDIO card is arranged, with the hexadecimal value 0004h.
<VendorID>
Describes the vendor ID of the cPCIDIO with the hexadecimal value 1172h.
<StateReg>
Describes the PCI status of the cPCIDIO.
<ControlReg>
Control register for the PCI bus
<ClassCode>
Contains the card class description with the hexadecimal value 118000h.
<RevisionID>
Describes the revision of the card FPGA, e.g. 02h
<HeaderType>
Describes the type of the PCI header with the hexadecimal value 00h.
<BaseAdress>
Describes the base address (form BAR0 & 0x0FFFC) of the card in the I/O range of the PC system for direct programming.

<SubSysID>
Contains the (sub)system identification of the cPCIDIO with the hexadecimal value 1026h.
<SubVenID>
Contains the (sub)vendor identification with the hexadecimal value EB84h.
<IrqPin>
Contains the PCI interrupt pin used.
<IrqLine>
Contains the interrupt number used.

⇦ Output

Filled, external data array

Return

If the function was executed successfully, it returns TRUE. If an error has occurred while running, it returns FALSE.
The error can be identified using the function *pcidioGetErrorMsg*.

pcidioSetTimer

Description

Sets the timer interval of the cPCIDIO with the assigned value.

> ***Note***
> *The timer interval is derived as*
> *(assigned value + 1) * 100ns*
> *which means times of 200ns to 1.6777217s can be programmed.*

⇔ Parameter

⇨ Input

<BoardNumber>
cPCIDIO addressing index, detected by the function *pcidioGetSummaryOfAllBoards.*

<TimerInterval>
Value for the timer interval in counts from 1 (200ns) to 2^{24} = 16777216 (1.6777217s)

⇦ Output

none

Return

If the function was executed successfully, it returns TRUE. If an error has occurred while running, it returns FALSE.
The error can be identified using the function *pcidioGetErrorMsg*.

pcidioStartTimer

Description

Starts the timer of the cPCIDIO without interrupt handling.

> ***Note***
> *If an interrupt handle is desired, then the functions pcidioSetIRQTimer and pcidioEnableIrq must be called before the function is called.*

⇔ Parameter

⇨ Input

<BoardNumber>
cPCIDIO addressing index, detected by the function *pcidioGetSummaryOfAllBoards*.

⇦ Output

none

Return

If the function was executed successfully, it returns TRUE. If an error has occurred while running, it returns FALSE.
The error can be identified using the function *pcidioGetErrorMsg*.

pcidioStopTimer

Description

Stops the timer of the cPCIDIO without blocking the interrupt.

> ***Note***
> *In order to terminate the interrupt handle, the functions pcdioSetIRQTimer and pcidioDisableIrq must also be called after calling this function.*

⇔ Parameter

⇨ Input

<BoardNumber>
cPCIDIO addressing index, detected by the function *pcidioGetSummaryOfAllBoards.*

⇦ Output

none

Return

If the function was executed successfully, it returns TRUE. If an error has occurred while running, it returns FALSE.
The error can be identified using the function *pcidioGetErrorMsg*.

pcidioSetIRQTimer

Description

Blocks and releases the timer interrupt for processing.

> ***Note***
> *The PCI interrupt itself will not be globally released or blocked with this function. This function blocks or activates the timer interrupt only locally in the card interrupts mask.*

⇔ Parameter

⇨ Input

<BoardNumber>
cPCIDIO addressing index, detected by the function *pcidioGetSummaryOfAllBoards*.

<CntrlByte>
Control register for controlling the timer interrupt (1->Release, 0->Block)

⇦ Output

none

Return

If the function was executed successfully, it returns TRUE. If an error has occurred while running, it returns FALSE.
The error can be identified using the function *pcidioGetErrorMsg*.

pcidioSetWatchdogIntervall

Description

Sets the timeout period for the cPCIDIO watchdog to the assigned value and starts it.

> ***Note***
> *Setting the timeout period immediately starts the hardware watchdog. The watchdog can only be disabled or the timeout period made settable again by a hardware reset or the appropriate software function pcidioReset.*
> ***Note***
> *If the watchdog is enabled, an output must be accessed at least once within the timeout period, otherwise the watchdog will switch off the output.*

⇔ Parameter

⇨ Input

<BoardNumber>
cPCIDIO addressing index, detected by the function *pcidioGetSummaryOfAllBoards.*

<WatchdogInterval>
Value for the timeout period of the watchdog in counts from 1 (26.2144ms)...255 (6.684672s)

⇦ Output

none

➲ Return

If the function was executed successfully, it returns TRUE. If an error has occurred while running, it returns FALSE.
The error can be identified using the function *pcidioGetErrorMsg*.

pcidioGetWatchdogState

Description

Returns the watchdog status, whether it has tripped.

⇔ Parameter

⇨ Input

<BoardNumber>
cPCIDIO addressing index, detected by the function *pcidioGetSummaryOfAllBoards*.

⇦ Output

<WatchdogState>
Watchdog status, with
0 for watchdog has not tripped and
1 for watchdog has tripped.

➲ Return

If the function was executed successfully, it returns TRUE. If an error has occurred while running, it returns FALSE.
The error can be identified using the function *pcidioGetErrorMsg*.

pcidioReset

Description

This function resets the cPCIDIO hardware, including any present extension cards, completely to its default state.

⇔ Parameter

⇨ Input

<BoardNumber>
cPCIDIO addressing index, detected by the function *pcidioGetSummaryOfAllBoards.*

⇦ Output

none

Return

If the function was executed successfully, it returns TRUE. If an error has occurred while running, it returns FALSE.
The error can be identified using the function *pcidioGetErrorMsg*.

pcidioGetErrorMsg

Description

This function returns the last given error message.

⇔ Parameter

⇨ Input

none

⇦ Output

<ErrorMsg>
Pointer to a sufficiently large external text field (min. 100 characters), to be created by the programmer, into which the string from the function is copied.

Return

none

pcidioEnableIrq

Description

This function installs the user-specific interrupt handler and enables the PCI interrupt globally but not the respective local masks.

⇔ Parameter

⇨ Input

<BoardNumber>
cPCIDIO addressing index, detected by the function *pcidioGetSummaryOfAllBoards.*

<IntHandler>
Function pointer of type *PCIDIO_INT_HANDLER* on the user interrupt handler.

⇦ Output

none

Return

If the function was executed successfully, it returns TRUE. If an error has occurred while running, it returns FALSE.
The error can be identified using the function *pcidioGetErrorMsg*.

pcidioDisableIrq

Description

This function disables the PCI interrupt globally, but not the respective local masks.

⇔ Parameter

⇨ Input

<BoardNumber>
cPCIDIO addressing index, detected by the function *pcidioGetSummaryOfAllBoards*.

⇦ Output

none

Return

If the function was executed successfully, it returns TRUE. If an error has occurred while running, it returns FALSE.
The error can be identified using the function *pcidioGetErrorMsg*.

pcidioGetIrq

Description

This function returns the content of the card's internal interrupt register upon appearance of the final interrupt.

⇔ Parameter

⇨ Input

<BoardNumber>
cPCIDIO addressing index, detected by the function *pcidioGetSummaryOfAllBoards.*.

<Int_State>
Array for the register status.

```
typedef PCIDIO_INT_STATE
{
  BYTE BoardNumber;   /* Card allocation */
  DWORD IRQIO_1_32; /* Interrupt register of the base card */
  DWORD RESERVED; /* Reserved for future extensions */
  BYTE IRQTIMER;  /* Interrupt register for the timer  */
  BYTE PERREG;   /* Global interrupt register of the card */
}
```

< BoardNumber>
cPCIDIO addressing index, detected by the function *pcidioGetSummaryOfAllBoards*.
<IRQIO_1_32>
The register returns the status of the respective interrupt sources of inputs 0 to 31. A set bit indicates a pending interrupt.
<RESERVED>
Reserved for future extensions.
<IRQTIMER>
The variable contains the status and the interrupt status of the timer in the two lowest bit positions:

Bit 1	Bit 0	Significance
0	0	Timer is not running and no timer interrupt was triggered
0	1	Timer is running and no timer interrupt was triggered
1	0	Timer is no longer running and timer interrupt was triggered
1	1	Timer is running and timer interrupt was triggered

<PERREG>
This register offers another method of determining the card as interrupt source:

Bit 0: This bit is 1 if at least one interrupt of the card is pending
Bit 2: This bit is 1 if the timer interrupt is pending
Bit 3: This bit is 1 if at least one of the interrupts of the base card inputs is pending

⇦ Output

none

⇐ Return

If the function was executed successfully, it returns TRUE. If an error has occurred while running, it returns FALSE.
The error can be identified using the function *pcidioGetErrorMsg*.

pcidioResetIrq

Description

This function resets in the user-specific mode the kernel mode interrupt handler. This function must be called at end of the user-specific interrupt handler.

⇔ Parameter

⇨ Input

<BoardNumber>
cPCIDIO addressing index, detected by the function *pcidioGetSummaryOfAllBoards.*

⇦ Output

none

Return

If the function was executed successfully, it returns TRUE. If an error has occurred while running, it returns FALSE.
The error can be identified using the function *pcidioGetErrorMsg*.

5.2 Digital Input Functions

pcidioDIGetChannelState

Description

Returns the input state of the assigned channel.

⇔ Parameter

⇨ Input

<BoardNumber>
cPCIDIO addressing index, detected by the function *pcidioGetSummaryOfAllBoards.*

<Channel>
Channel number 0...31 of the desired input channel.

⇦ Output

<ChannelState>
The status of the channel: a '1' signifies a high level and '0' a low level on the corresponding I/O pin of the cPCIDIO connector.

Return

If the function was executed successfully, it returns TRUE. If an error has occurred while running, it returns FALSE.
The error can be identified using the function *pcidioGetErrorMsg*.

pcidioDIGetByte

Description

This function is available starting from the driver DLL version 3.0 and returns the input state of the assigned 8 channel group.

⇔ Parameter

⇨ Input

<BoardNumber>
cPCIDIO addressing index, detected by the function *pcidioGetSummaryOfAllBoards.*

<bytenumber>
Number of the input group 0..3

⇦ Output

<state>
The state of the 8 selected input channels of the selected card.

⮌ Return

If the function was executed successfully, it returns TRUE. If an error has occurred while running, it returns FALSE.
The error can be identified using the function *pcidioGetErrorMsg*.

pcidioDIGetState

Description

Returns the state of all inputs at once.

⇔ Parameter

⇨ Input

<BoardNumber>
cPCIDIO addressing index, detected by the function *pcidioGetSummaryOfAllBoards.*

⇦ Output

<AllChannelState>
Status of all input channels of the selected card. The variable is a structure of two 32-bit-long values and each bit stands for a channel.

```
typedef struct PCIDIOALLCHANNELS
 {
   DWORD Basis;          /* Status of the base card inputs */
   DWORD RESERVED;  /* Reserved for future extensions */
 }
```

⮌ Return

If the function was executed successfully, it returns TRUE. If an error has occurred while running, it returns FALSE.
The error can be identified using the function *pcidioGetErrorMsg*.

pcidioDISetIrqChannelConfiguration

Description

sets the interrupt configuration for an input channel.

> ***Note***
> *The PCI interrupt itself will not be globally released or blocked with this function. This function only blocks or activates the respective local I/O interrupt in the card interrupts mask.*

⇔ Parameter

⇨ Input

<BoardNumber>
cPCIDIO addressing index, detected by the function *pcidioGetSummaryOfAllBoards.*

<*Channel*>
Channel number 0...31 of the desired input channel.

<ChannelIntEnable>
Releases input channel as interrupt source with '1' or blocks the channel again with '0'.

<ChannelTrigger>
Defines the interrupt trigger time, where '0' stands for a trigger on a falling edge and '1' stands for a trigger on a rising edge.

⇦ Output

none

Return

If the function was executed successfully, it returns TRUE. If an error has occurred while running, it returns FALSE.
The error can be identified using the function *pcidioGetErrorMsg*.

pcidioDIGetIrqChannelConfiguration

Description

Returns the interrupt configuration of an input.

⇔ Parameter

⇨ Input

<*BoardNumber*>
cPCIDIO addressing index, detected by the function *pcidioGetSummaryOfAllBoards.*

<*Channel*>
Channel number 0...31 of the desired input channel

⇦ Output

<*ChannelIntEnable*>
If '1', the input is locally enabled as interrupt source, if '0', the input is locally disabled as interrupt source.

<*ChannelTrigger*>
If '0', the falling edge will be selected as trigger time for the input and if '1', the rising edge will be selected.

Return

If the function was executed successfully, it returns TRUE. If an error has occurred while running, it returns FALSE.
The error can be identified using the function *pcidioGetErrorMsg*.

pcidioDIEnableBuffering

Description

This function enables the internal buffering of the digitals inputs of the card. A sample is taken each time the interrupt of the internal timer of the card triggers.

⇔ Parameter

⇨ Input

<BoardNumber>
cPCIDIO addressing index, detected by the function *pcidioGetSummaryOfAllBoards*

⇦ Output

None

Return

If the function was executed successfully, it returns TRUE. If an error has occurred while running, it returns FALSE.
The error can be identified using the function *pcidioGetErrorMsg*.

pcidioDIDisableBuffering

Description

This function disables the internal buffering.

⇔ Parameter

⇨ Input

<BoardNumber>
cPCIDIO addressing index, detected by the function *pcidioGetSummaryOfAllBoards*

⇦ Output

none

Return

If the function was executed successfully, it returns TRUE. If an error has occurred while running, it returns FALSE.
The error can be identified using the function *pcidioGetErrorMsg*.

pcidioDIGetBufferedSamples

Description

This function does a read out of the buffered samples of the digital inputs.

⇔ Parameter

⇨ Input

<*BoardNumber*>
cPCIDIO addressing index, detected by the function *pcidioGetSummaryOfAllBoards*

<bufferSize>
Size of the external allocated memory. For the digital inputs an array according to the format PCIDIOCHANNELS is required, for the timestamp an array according to the format LARGE_INTEGER is required. Both Arrays must have the same length bufferSize.

⇦ Output

<Samples>
An array filled with the samples of the digital inputs.

<TimeStamp>
An array filled with the timestamps corresponding to the samples of the digital inputs in the array Samples.

<CounterActSamples>
Number of samples in the returned arrays.

<BufferOverflow>
A flag to notify if a buffer overrun between to subsequent read outs of the buffer occurred. The internal buffer can save up to 1000 samples until a buffer overrun occurs. If the flag is 1, an overrun occurred and the returned data are incomplete. If the flag is 0, the returned data are valid.

⇨ Return

If the function was executed successfully, it returns TRUE. If an error has occurred while running, it returns FALSE.
The error can be identified using the function *pcidioGetErrorMsg*.

5.3 Digital Output Functions

pcidioDOSetChannelState

Description

Sets the assigned output to the assigned level.

⇔ Parameter

⇨ Input

<BoardNumber>
cPCIDIO addressing index, detected by the function *pcidioGetSummaryOfAllBoards.*

<Channel>
Channel number 0...31 of the desired output channel.

<ChannelState>
Status of the channel. '1' signifies a high level and '0' a low level on the corresponding I/O pin of the PCIDIO connector.

⇦ Output

none

Return

If the function was executed successfully, it returns TRUE. If an error has occurred while running, it returns FALSE.
The error can be identified using the function *pcidioGetErrorMsg*.

pcidioDOSetByte

Description

This function available starting from the driver DLL version 3.0 sets the output level of the assigned 8 channel output group to the assigned levels.

> ***Note***
> *Channels that are operated as inputs or are not used must always be initialised to '0'.*

⇔ Parameter

⇨ Input

<BoardNumber>
cPCIDIO addressing index, detected by the function *pcidioGetSummaryOfAllBoards.*

<bytenumber>
Number of the output group 0..3

<state>
Output level to set the output channels of the selected card.

⇦ Output

none

⇐ Return

If the function was executed successfully, it returns TRUE. If an error has occurred while running, it returns FALSE.
The error can be identified using the function *pcidioGetErrorMsg*.

pcidioDOSetState

Description

Sets the state of all output channels.

> ***Note***
> *Channels that are operated as inputs or are not used must always be initialised to '0'.*

⇔ Parameter

⇨ Input

<BoardNumber>
cPCIDIO addressing index, detected by the function *pcidioGetSummaryOfAllBoards.*

<AllChannelState>
Status of all output channels of the selected card. The assigned structure contains one variable for each of 32 channels. Nonexistent channels or channels used as inputs must always be set to '0'.

```
typedef struct PCIDIOALLCHANNELS
{
  DWORD Basis;          /* Status of base card outputs */
  DWORD RESERVED;  /*Reserved for future extensions */
}
```

⇦ Output

none

⇐ Return

If the function was executed successfully, it returns TRUE. If an error has occurred while running, it returns FALSE.
The error can be identified using the function *pcidioGetErrorMsg*.

pcidioDOServiceChannel

Description

Updates the status of all output channels as programmed with the functions pcidio-DOSetChannelState or pcidioDOSetState.

> ***Note***
> *If the watchdog is used, this program must be called at least once within the watchdog's programmed timeout period if no other output function is being used.*

⇔ Parameter

⇨ Input

<BoardNumber>
cPCIDIO addressing index, detected by the function *pcidioGetSummaryOfAllBoards.*

⇦ Output

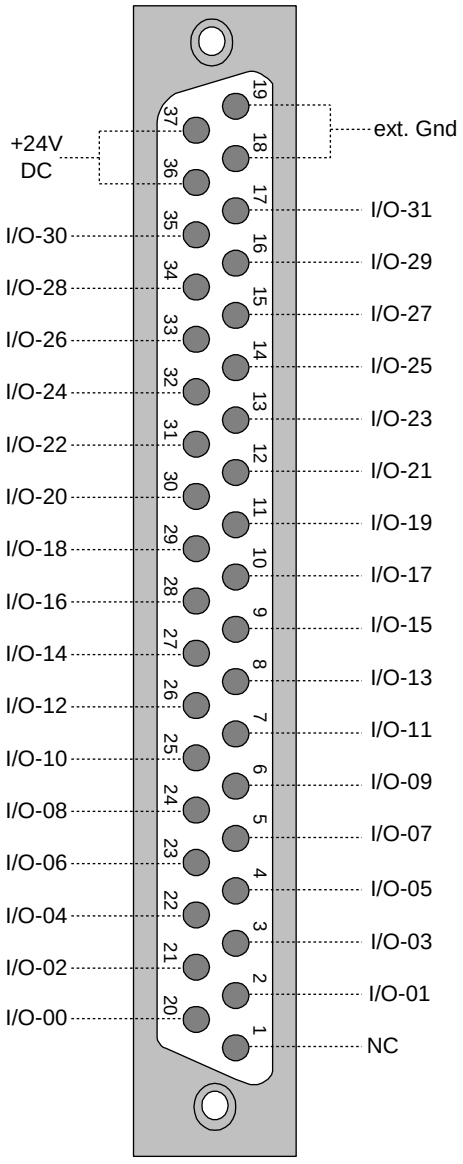
none

Return

If the function was executed successfully, it returns TRUE. If an error has occurred while running, it returns FALSE.
The error can be identified using the function *pcidioGetErrorMsg*.

Appendix

A Pin Assignment 37-pin D-SUB Socket cPCIDIO



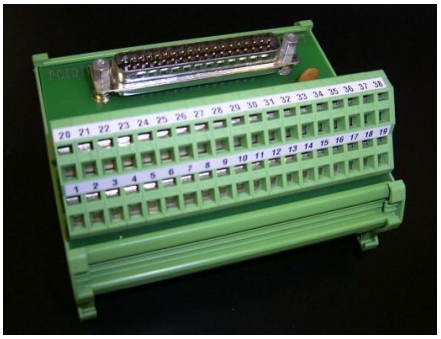
The external power supply can also be connected internal on the card on a separate screw-type terminal block.

Description	Pin
+24V DC	1
GND	2

When looking from the front, pin 1 is the left connector of the screw-type terminal block. For wiring, we recommend using a wire of 0.75mm^2 (max. 2.5mm^2).

B PCIDIOHM Clamp Module

For user-friendly system wiring in the switch cabinet, there is the terminal block PCIDIOHM with spring-cage connections and approx. dimensions 102.5 mm (width) * 90.0 mm (depth) * 60.0 mm (height) available for the DIN-Rail (TS 35 and TS 32).



The module is optionally available with wall mount and screw-type terminals.

System wiring is done on the spring force connections with 0.25 mm^2 to 1.5 mm^2 wires (also with ferrules).

We recommend using shielded connector cables such as PCIDIOVK for connecting the cPCIDIO to the D-SUB connection of the DIN-rail module.

Name (*)	Contact terminal strip	Pin D-SUB male connector	Name (*)	Contact terminal strip	Pin D-SUB male connector
Not connected	1	1	I/O-13	8	8
GND	18	18	I/O-14	27	27
GND	19	19	I/O-15	9	9
+24V	36	36	I/O-16	28	28
+24V	37	37	I/O-17	10	10
			I/O-18	29	29
			I/O-19	11	11
I/O-00	20	20	I/O-20	30	30
I/O-01	2	2	I/O-21	12	12
I/O-02	21	21	I/O-22	31	31
I/O-03	3	3	I/O-23	13	13
I/O-04	22	22	I/O-24	32	32
I/O-05	4	4	I/O-25	14	14
I/O-06	23	23	I/O-26	33	33
I/O-07	5	5	I/O-27	15	15
I/O-08	24	24	I/O-28	34	34
I/O-09	6	6	I/O-29	16	16
I/O-10	25	25	I/O-30	35	35
I/O-11	7	7	I/O-31	17	17
I/O-12	26	26	Erde	38	**(**)**

Tabelle: Connector assignment of terminal strip and D-SUB socket of PCIDIOHM

(*) Name when using a 1:1 wired connector cable to the cPCIDIO
(**) Casing of the connector wired over Y-condenser to clamp 38 of the terminal strip

We urgently recommend earthing pin 38 as well in order to prevent interference on the connector cable.

C DOS Driver

There is a suitable DOS driver in the source code available for operating the cPCIDIO under MS-DOS.

The initialisation is done over PCI BIOS extensions and the card is communicated with over I/O commands. You can find more information on this in the corresponding Readme file and the source code comments.

D Item Numbers

Item N°	Description
cPCIDIO	cPCIDIO card with 32 I/Os, Watchdog, Timer
PCIDIOHM	Optional Clamp Module with spring force connections and D-SUB connection for the DIN standard rail for user-friendly wiring in the switch cabinet for 32 I/Os
PCIDIOVK1M	Optional connector cable, 1m 37 pin, 1:1 wired, for connecting cPCIDIO to DIN Clamp Module PCIDIOHM
PCIDIOVK2M	Optional connector cable, 2m 37 pin, 1:1 wired, for connecting cPCIDIO to DIN Clamp Module PCIDIOHM 

E Support

Should you have any questions on our product or need assistance, get in touch with our support team, giving an exact description of your problem, and we will gladly help you out.

E-mail: support@ebru.de
Tel. +49 6228 91 37 10 (Mo.-Fr. 8:00 am – 5:00 pm)
Fax +49 6228 91 37 29

F Customised Models

As an all-round industrial electronics service provider, we will gladly conduct custom modifications or extensions for you. Because we have integrated the PCI interface into an FPGA together with the entire control, there exist many possibilities for development.

G Service Address

We hope you will never have any need for this service address. Nevertheless, should any malfunction occur despite careful production and controlling, please write to:

EBRU GmbH
In den Kreuzwiesen 21
D-69250 Schoenau
Germany

Should your send your card in for repair, please include as detailed a description of the malfunction as possible. That way, we can handle your specific case much more quickly.

H Updates

We provide driver software and documentation updates on our website at www.ebru.de.