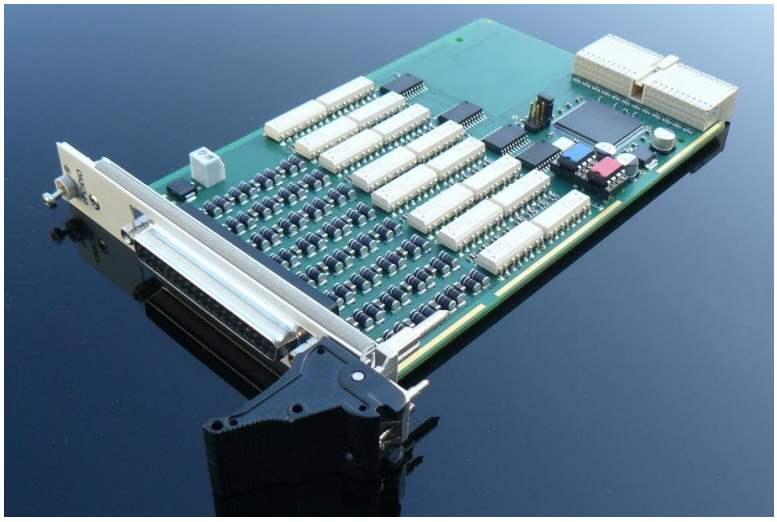
cPCIDIO

Benutzerhandbuch

Revision 3.5

RoHS Compliant
Directive 2005/95/EC



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

Inhaltsverzeichnis

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

Diese Dokumentation wurde zur Beschreibung der Verwendung von Hard- und Software erstellt. Technische Änderungen im Rahmen der Produktverbesserung sind dem Hersteller vorbehalten.

Wichtiger Hinweis!
Wir müssen Sie darauf hinweisen, dass wir keine juristische Verantwortung oder irgendeine Haftung für Folgen die auf Fehlbedienung oder Softwarefehler zurückgehen, übernehmen können. Für die Mitteilung von Fehlern, Anregungen, Verbesserungsvorschlägen, etc. sind wir jederzeit dankbar.

1 Einführung

Sehr geehrte Kundin, sehr geehrter Kunde,

mit dem Kauf der cPCIDIO haben Sie sich für ein technisch hochwertiges Produkt der Firma EBRU GmbH entschieden, das unser Haus im einwandfreien Zustand verlassen hat.

Überprüfen Sie bitte jedoch trotzdem die Vollständigkeit und den Zustand Ihrer Lieferung. Sollten irgendwelche Mängel auftreten bitten wir Sie, uns davon in Kenntnis zu setzen.

Bevor Sie die Karte einbauen, lesen Sie bitte zuerst das Kapitel zur Installation der Karte aufmerksam durch.

1.1 Lieferumfang

Wir sind selbstverständlich bemüht, ein vollständiges Produktpaket auszuliefern. Um sicherzustellen, dass Sie ein vollständiges Paket erhalten haben, haben wir nachfolgend eine Auflistung der im Paket enthaltenen Teile aufgeführt.

- cPCIDIO Baugruppe
- Benutzerhandbuch als *.pdf auf CD-ROM
- Treibersoftware und Demos auf CD-ROM
- Klemmenmodul PCIDIOHM (Optional)
- Verbindungskabel PCIDIOVK1M oder PCIDIOVK2M zum optionalen Klemmenmodul PCIDIOHM (Optional)

1.2 Leistungsumfang

Artikelnummer	I/Os 24V	Timer	Watchdog der Ausgänge	Ext. Versorgung	Erweiterbar
cPCIDIO	32	Ja	Ja	Ja	Nein

Die cPCIDIO Karte bietet in Rechnern mit CompactPCI-Steckplätzen 32 optisch entkoppelte/galvanisch getrennte und für 24VDC optimierte digitale I/Os. Jeder digitale I/O kann je nach Bedarf als digitaler Ausgang oder digitaler Eingang und als Interruptquelle verwendet werden.

Die Kontaktierung der 32 digitalen I/Os der Karte erfolgt über eine 37 polige SUB-D Buchse der 4 TE-Frontblende. Die Steckkarte benötigt einen freien Steckplatz im CompactPCI-System mit Bauhöhe 3 HE und CompactPCI-Anschluss J1 (110 Pin, 32Bit).

Optional steht für die DIN-Normschiene das Klemmenmodul PCIDIOHM mit Käfigzugklemmen zur einfachen Anlagenaufschaltung sowie das entsprechende Verbindungskabel PCIDIOVK1M bzw. PCIDIOVK2M zur Verfügung.

Features

- 32 vom Rechner optisch/galvanisch getrennte und für 24VDC optimierte digitale I/Os
- Jeder I/O als Ein- oder Ausgang und Interruptquelle frei verwendbar
- Isolationsspannung der galvanischen Trennung min. 1500V RMS

Ausgänge

- Max. 1A Ausgangsstrom je Kanal
- Direkter Anschluss ohmscher, kapazitiver und induktiver Lasten
- Dauerkurzschlussfest mit automatischen Wiederanlaufversuchen und Überspannungsschutz
- Kurzschlusserkennung für Diagnosezwecke
- Programmierbare rechnerunabhängige Watchdog der Ausgänge

Eingänge

- Für 24VDC optimierte Schaltschwelle
- Eingangsstrom bei 24VDC ca. 3 mA
- Jeder Eingang ist interruptfähig mit programmierbarer Flanke
- RC-Eingangsfilter und digitales Filter mit 10KHz Grenzfrequenz
- Unbenutzte Eingänge können offen bleiben

Sonstiges

- Programmierbarer 24 Bit 10 MHz Timer mit Interruptfunktion
- 3 zusätzliche Jumper zur Unterscheidung von bis zu 8 Karten innerhalb desselben Systems
- 32 Bit CompactPCI Karte (Universal Card für 5V/33MHz und 3,3V/66MHz Slots) mit 4 TE für 3 HE CompactPCI Systeme
- Externe Versorgung der Ausgangstransistoren und der optischen Entkopplung über die D-SUB Buchse der Frontblende mit 24VDC +/- 30%
- Optionales Klemmenmodul mit Käfigzugklemmen zur einfachen Anlagenaufschaltung
- Umfangreiche Software für Windows 7(64/32 Bit), Vista(64/32 Bit), XP(64/32 Bit), 2K, und MSDOS verfügbar
- Auf Wunsch kundenspezifische Anpassungen und Treiber möglich
- RoHS konform gemäß Direktive 2005/95/EC
- Auch als **softwarekompatible PCI Bus Baugruppe PCIDIO** lieferbar

1.3 Systemanforderung

Die cPCIDIO Karte setzt einen Rechner mit X86 Prozessor oder kompatiblen Rechner voraus.

Als Steckplatz wird ein freier 32 Bit Steckplatz (4 TE) mit 5V/33MHz oder 3,3V/66MHz im CompactPCI-System mit Bauhöhe 3 HE und CompactPCI-Anschluss J1 (110 Pin, 32Bit) benötigt.

1.4 Softwareunterstützung

Den aktuellen Lieferumfang des Softwarepaketes entnehmen Sie bitte den entsprechenden Readme Dateien des Paketes.

Es sind Treiber, Demoprogramme, Inbetriebnahmetools sowie Sourcecode für Windows und DOS erhältlich.

1.5 Notationen in diesem Handbuch

Funktionsnamen	werden in der Beschreibung immer fettkursiv,
<TimesNewRoman>	Parameter in spitzen Klammern,
[]	physikalische Einheiten in eckigen Klammern
TimesNewRoman	Quellcodeausschnitte kursiv dargestellt
🗐	Beschreibung von Treiberfunktionen
🖉	Hinweise auf bestimmte Verwendung bzw. besondere Verwendung
🕮	Tipps zur Verwendung
⇔	Parameter für Funktionen
⇨	Eingabeparameter von Funktionen
⇦	Ausgabeparameter von Funktionen
⮌	Rückgabewert von Funktionen

2 Installation

Bitte lesen Sie vor dem Einbau der Karte das Handbuch Ihres Rechners bzgl. der Installation von Erweiterungskarten genau durch und folgen dann der beschriebenen Installationsprozedur in diesem Handbuch.

> ***Hinweis***
> *Soll ein bereits vorhandener Treiber ersetzt werden, verwenden Sie bitte unbedingt die dazugehörige uninstall.bat des Treiberpaketes vor Installation des neuen Treibers.*

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

Zur Installation unter Windows 7/Vista/XP sind mehrere Installationsschritte durchzuführen.

1. Installieren Sie die gelieferte CompactPCI-Karte in einem freien CompactPCI-Slot ihres Rechners.
   Achtung!! Trennen Sie unbedingt den Rechner vom Versorgungsnetz da einige CPU-Boards trotz vermeintlich abgeschaltetem Rechner Spannung führen.
2. Schalten Sie den Rechner ein und starten Sie Windows.
3. Nun wird automatisch der Hardware-Assistent gestartet, beenden Sie diesen mit Abbruch und öffnen Sie den Explorer (Dateimanager).
4. Wechseln Sie auf das Laufwerk in dem sich das Installationsmedium befindet.
5. Wechseln Sie nun in das Verzeichnis *Win_32(32Bit)* / *Win_64(64Bit)* auf dem Installationsmedium
6. Starten Sie nun das Stapelverarbeitungsprogramm *Install_x32(32Bit)* / *Install_x64(64Bit)*. Die Systemtreiber werden nun installiert. Folgen Sie nun den Anweisungen auf ihrem Rechner. Ist dies abgeschlossen, ist die Karte einsatzbereit installiert.
7. Vor dem Verwenden der Karte sollten sie den Rechner jetzt neu starten.
8. Um die Installation zu testen, können Sie nun die auf dem Installationsmedium vorhandenen Programme zur Demonstration und Inbetriebnahme nutzen.

2.2 Installation unter Windows 2000

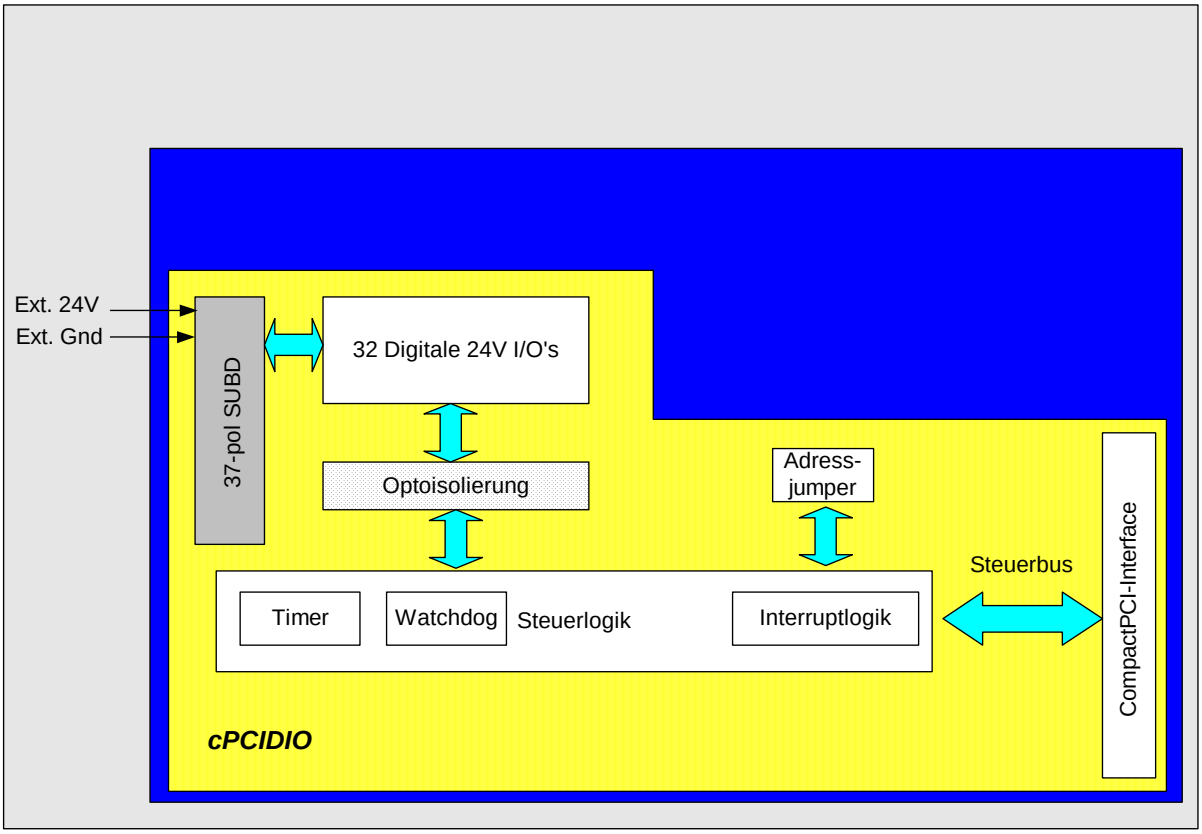
Zur Installation unter Windows 2000 sind mehrere Installationsschritte durchzuführen.

1. Installieren Sie die gelieferte CompactPCI-Karte in einem freien CompactPCI-Slot ihres Rechners.
   Achtung!! Trennen Sie unbedingt den Rechner vom Versorgungsnetz da einige CPU-boards trotz vermeintlich abgeschaltetem Rechner Spannung führen.
2. Schalten Sie den Rechner ein, starten Sie Windows 2000 und melden Sie sich als Administrator an.
3. Nun wird automatisch der Hardware-Assistent gestartet, beenden Sie diesen mit Abbruch und öffnen Sie den Explorer (Dateimanager).
4. Wechseln Sie auf das Installationsmedium in das Verzeichnis *Win32* und führen Sie das Stapelverarbeitungsprogramm *Install_x32* aus.
5. Damit der Treiber im System aktiviert wird, müssen Sie den Rechner jetzt neu starten. Nach dem Neustart ist die Karte einsatzbereit.

3 Hardware

3.1 Blockschaltbild

Untenstehendes Blockschaltbild zeigt den Aufbau der cPCIDIO.



3.2 Generelle Hinweise

Die Karte sowie sämtliche Steckverbinder dürfen nur im spannungslosen Zustand gesteckt werden.

Achten Sie bitte darauf, dass der Slotwinkel der cPCIDIO mit dem Gehäuse des Rechners verschraubt und darüber geerdet ist.

Verwenden Sie für Verbindungen außerhalb des Rechners ausschließlich geschirmte Kabel und stellen Sie die Erdung des Schirmes sicher.

Stellen Sie sicher, dass bei Berührung der Karte und beim Stecken der Anschlusskabel keine statische Entladung über die Steckkarte geführt wird. Achten Sie auf korrekten Sitz der Anschlusskabel, nur so ist eine einwandfreie Funktion der Karte möglich.

3.3 Digitale I/Os

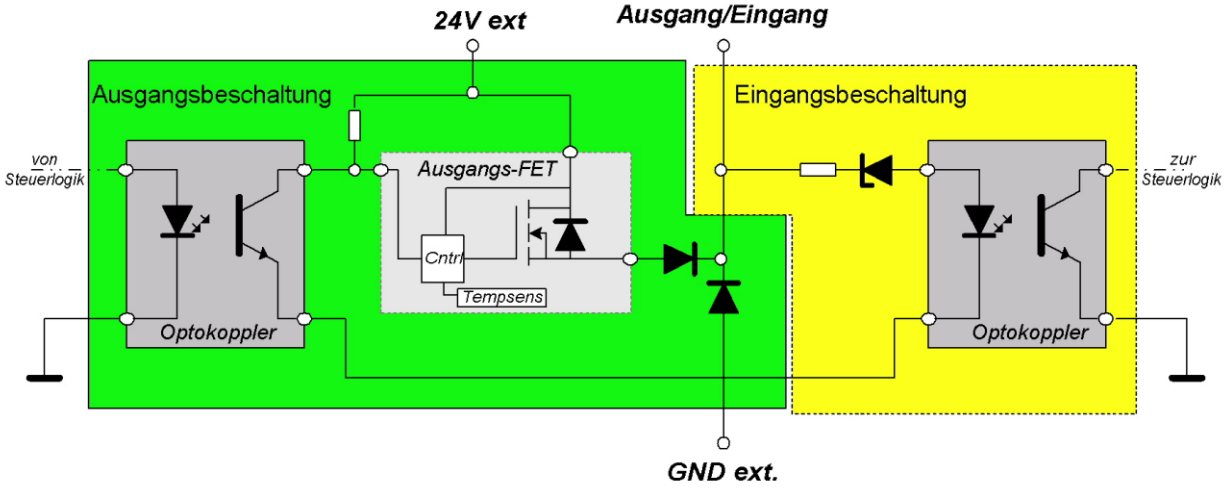
Die cPCIDIO Karte besitzt 32 freikonfigurierbare Ein- und Ausgänge. Alle I/Os können unabhängig voneinander als Eingang oder als rücklesbarer Ausgang konfiguriert werden.

Die Richtung der Ports und deren Interruptfähigkeit wird durch die Software eingestellt.

> ***Hinweis***
> *Sobald ein Port auf ‚1' gesetzt wird, wird der entsprechend zugeordnete Pin auf High (24V) gesetzt und der IO dient als Ausgang.*
>
> ***Ports die als Eingang verwendet werden dürfen nie auf ‚1' gesetzt werden.***
>
> *Wird ein Port gelesen, wird der Level des entsprechend zugeordneten Pins zurückgelesen (‚0' für GND oder ‚1' für 24V). Es können auch Ausgänge zurückgelesen werden.*

Nach dem Einschalten oder einem Reset sind alle Ausgänge auf ‚0' programmiert und die Interruptfunktionalität abgeschaltet.

> ***Tipp***
> *Zur Programmierung lesen sie bitte die entsprechenden Kapitel dieses Handbuchs*



3.3.1 Externe Spannungsversorgung

Die externe Spannungsversorgung der I/Os und der optischen Entkopplung erfolgt über den D-SUB Steckverbinder an den Anschlüssen +24VDC und GND mit 24VDC +/-30%.

> **Tipp**
> *Wir empfehlen dringend, die Spannungsversorgung grundsätzlich an allen zur Verfügung stehenden Anschlüssen auch aufzulegen.*

> **Hinweis**
> *Der Summenstrom darf 5A nicht überschreiten.*

3.3.2 Ausgänge der I/Os

Die optisch entkoppelten und dauerkurzschlussfesten Ausgänge der I/Os sind jeweils mit Power MOSFET Transistoren aufgebaut und verfügen über eine interne Temperaturüberwachung, die im Überlast- und Kurzschlussfall den jeweiligen Ausgang selbsttätig abschaltet und bei Unterschreiten der Schwelltemperatur von 150° der Sperrschicht des MOSFET wieder einschaltet.
Jeder Ausgang verfügt zusätzlich über eine 1A Freilaufdiode nach GND und eine Längsdiode zum Schutz vor Rückströmen. Es können dadurch ohmsche, induktive und kapazitive Lasten direkt angeschlossen werden. Aufgrund der Schutzdiode und des Drain Source Widerstandes des MOSFET liegt der Spannungsabfall am jeweiligen Ausgang lastabhängig zwischen 0,6V und 2V. Das Schaltvermögen und die Schaltfrequenz der Ausgänge sind naturbedingt stark von der Art und Größe der jeweiligen Last abhängig.

Technische Daten Ausgang	
Max. Spitzenstrom	1,2 A
Max. Dauerstrom	1 A
Garantierter Dauerstrom	0,65 A bei rein ohmscher Last
Schaltfrequenz max.	10 KHz
Schaltfrequenz typisch	500 Hz bei 1A und rein ohmscher Last
	1 KHz bei 0,25 A und rein ohmscher Last
Einschaltzeit (typ./max.)	45 us/125 us bei 270 Ohm Last
Ausschaltzeit (typ./max.)	40 us/175 us bei 270 Ohm Last
Min. Isolationsspannung der Optokoppler	5 000 Vrms

3.3.3 Eingänge der I/Os

Für die optische Trennung der Eingänge der I/Os werden Optokoppler eingesetzt, die über die externe Spannungsversorgung an den jeweiligen Steckverbindern versorgt werden. Eine Längsdiode und ein Längswiderstand zur Strombegrenzung stellen die Schaltschwelle ein, ein den Optokopplern nachgeschaltetes RC-Glied und ein digitales Filter dienen der Unterdrückung von Störungen und Transienten.

Technische Daten Eingang	
Schaltschwelle typ. [max.]	12 VDC +/- 10% [12 VDC +/- 30%]
Schaltfrequenz	> 10 KHz
Eingangsstrom	(Uein – 8,2 V – 1,15 V)/ (4700 Ohm)
Max. Eingangsspannung Uein	31,8 VDC
Min. Isolationsspannung der Optokoppler	5 000 Vrms

3.3.4 Kurzschlusserkennung

Die Rücklesbarkeit eines als Ausgang verwendeten I/Os kann zur Kurzschlusserkennung herangezogen werden. Wird nach dem Setzen eines I/Os und einem Delay von mindestens 250µs nebst ggf. vorhandener lastabhängiger Verzögerung der entsprechende I/O zurückgelesen, so muss dieser eine ‚1' zurückliefern. Wird stattdessen eine ‚0' zurückgeliefert, liegt ein externer Kurzschluss vor.

3.3.5 Watchdog der Ausgänge

Die Ausgangstransistoren der Karte können von einer gemeinsamen rechnerunabhängigen Watchdog überwacht werden, deren Timeout Zeit zwischen 26,21ms und 6,68 Sekunden programmierbar ist. Nach dem Einschalten des Rechners oder einem Softwarerest ist die Watchdog disabled.

Wird bei freigegebener Watchdog nicht wenigstens ein Ausgang innerhalb der programmierbaren Timeout Zeit schreibend angesteuert, werden alle Ausgangstransistoren sofort zurückgesetzt.

Nach erfolgter Programmierung kann weder die Timeout Zeit verändert noch die Watchdog wieder disabled werden. Die Watchdog wird nur durch einen Softwarereset der Karte oder Neustart des Computers wieder disabled und der Wert von neuem einstellbar gemacht.

Ob die Watchdog „zugeschlagen" hat, kann mit dem Treiber überprüft werden.

3.3.6 Interruptsteuerung der I/Os

Jeder Eingang kann als separate Interruptquelle genutzt werden. Die Programmierung erfolgt über Funktionen des mitgelieferten Softwaretreibers.

Die einzelnen Interrupts der Eingänge werden per Software konfiguriert, freigegeben und wieder gesperrt.

Für die Eingänge stehen zwei verschiedene Flanken-Konfigurationen zur Verfügung. Es ist möglich, einen Interrupt bei steigender oder fallender Flanke am Eingangskontakt auszulösen.
Nach einem Hardwarereset sind alle Interrupteinstellungen gelöscht und die fallende Flanke als auslösende Flanke als Standardkonfiguration eingestellt.

> ***Hinweis***
> *Weitere Angaben zu den I/O-Interrupts sind in den Kapiteln zur Softwareprogrammierung enthalten.*

3.4 Timer

Auf der cPCIDIO ist ein 24 Bit Timer zur zyklischen Generierung von Interrupts integriert.

Technische Daten Timer	
Auflösung	24 Bit
Kleinstes Intervall	200ns
Größtes Intervall	1,6777217s
Interrupt	Interrupterzeugung bei Nulldurchgang konfigurierbar via Software

Ein Taktzyklus entspricht dabei 100ns, so dass ein maximales Intervall von 1,6777217 Sekunden einstellbar ist.
Bei jedem Nulldurchgang des Timers kann ein Interrupt ausgelöst werden, wenn dieser durch die Software freigegeben und eingestellt ist.

> ***Hinweis***
> *Weitere Angaben zu dem Timer sind in den Kapiteln zur Softwareprogrammierung enthalten.*

3.5 Interruptverwaltung

Die cPCIDIO bietet mehrere Interruptquellen an, die im Interrupt Sharing Verfahren auf der Karte verwaltet werden. Für jeden einzelnen karteninternen Interrupt gibt es Konfigurations-, Freigabe- und Sperrfunktionen.

Für den von der Karte verwendeten PCI Interrupt stehen ebenso Konfigurations-, Freigabe- und Sperrfunktionen zur Verfügung. Außerdem kann der Benutzer an den PCI Interrupt eine eigene Interrupthandler Funktion übergeben.

Nach einem Hardwarereset ist die Interruptfunktionalität zunächst gesperrt und nicht konfiguriert.

> ***Hinweis***
> *Weitere Angaben zu der Interrupt-Programmierung sind in den Kapiteln zur Softwareprogrammierung enthalten.*

3.6 Adressjumper

Zur Kartenunterscheidung mehrerer Karten der cPCIDIO innerhalb desselben Rechners sind auf der cPCIDIO drei Jumper integriert. Damit können acht Karten mit insgesamt maximal 256 I/Os unterschieden werden.

Die Jumper sind auf der Karte per Bestückungsdruck mit S2, S1 und S0 bezeichnet. Die Stellung der Schalter kann mit Hilfe von Softwarefunktionen abgefragt werden.

S2	S1	S0	Karte
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

Der Treiber benutzt die Stellung der Jumper zur Adressierung wenn mehrere Karten in demselben System vorhanden sind.

3.7 Allgemeine Daten

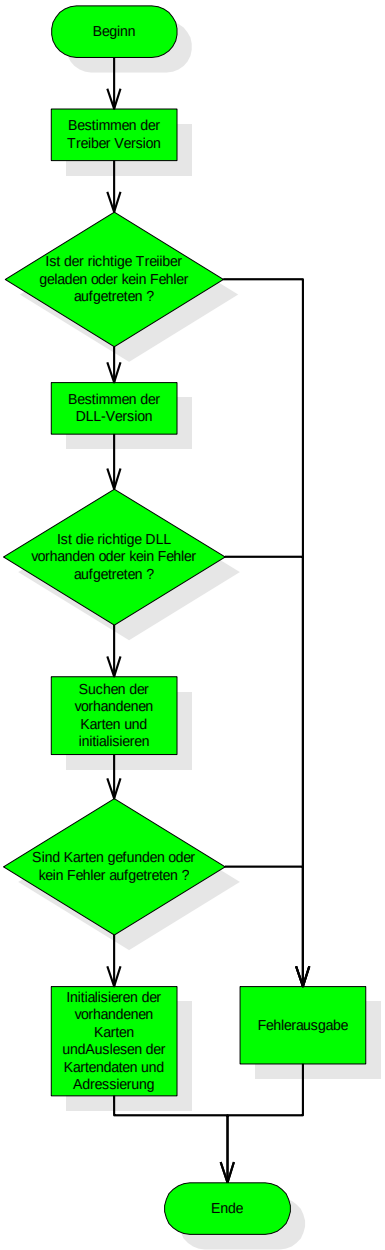
Technische Daten Allgemein	
Abmessungen	3 HE 160mm x 100mm (ohne Stecker und Slotblech) mit 4 TE Frontblende
Anschluss cPCIDIO	37-polige SUBD Buchse
Spannungsversorgung der Ein- und Ausgänge	Externe Spannungsversorgung über den Anschlussstecker
Betriebstemperatur	0..70°C
Lagertemperatur	-40...100°C
Rel. Luftfeuchte	0...90% (nicht kondensierend)

4 Programmierung

In diesem Kapitel soll gezeigt werden wie die cPCIDIO unter der Verwendung der Windows Treiber und der API-Referenz programmiert werden kann. Die Programmierung wird in Form von Ablaufplänen und C-Quellcode Beispielen dargestellt. Alle hier aufgeführten Beispiele dienen ausschließlich der Funktionsdemonstration.

4.1 Initialisieren der cPCIDIO

Dieses Unterkapitel zeigt eine Möglichkeit die im System vorhandenen Karten zu bestimmen und deren Adressierungsdaten auszulesen.

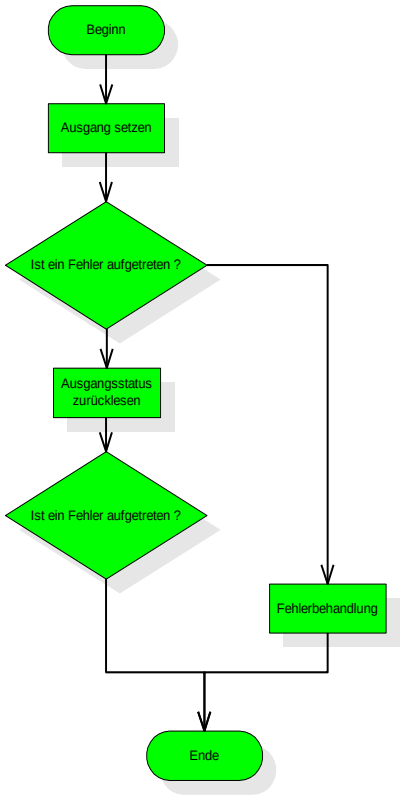



```
...
unsigned int l_uiDriverVersion;  /* Version des installierten Treibers */
unsigned int l_uiDLLRevision;    /* Revision der verwendeten DLL */
PCIDIO_SUMMERY  l_SummeryBuf[4];    /* Puffer für die Adressierungsdaten*/
BOOL l_bReturnValue;                /* Rückgabewert der Funktionen*/
char l_strErrorText[100];           /* Fehlernachricht in Klartext */
int l_iCntCards;                    /* Anzahl gefundener Karten */
...
/* Auslesen der Treiberversion */
l_bReturnValue = pcidioGetDriverVersion(&l_uiDriverVersion);
/* Ist der richtige Treiber installiert ? */
If((l_bRetunValue == TRUE)&&
   (l_uiDriverVersion == ACT_DRIVER_VERSION))
{
  /* Suchen nach vorhandenen Karten */
  l_iCntCards=pcidioGetCountBoards()
  /* Sind Karten vorhanden ? */
  if(l_iCntCards>0)
  {
    /* Initialisieren der vorhandenen Karten */
     l_bReturnValue = pcidioInitCards(&l_iCntCards);
      /* Auslesen der Adressierungsdaten der vorhandenen Karten */
      l_bReturnValue=pcidioGetSummeryOfAllBoards(l_SummeryBuffer);
   }
}
/* Sind Fehler aufgetreten */
if(l_bReturnValue == FALSE)
{
  /* Fehlertext auslesen */
  pcidioGetErrorMsg(l_strErrorTxt);
}
...
```

4.2 Einfache Bedienung der digitale Ausgänge

Als Einstieg in die Verwendung der Karte soll hier die Bedienung der Ausgänge ohne die Verwendung der Watchdog gezeigt werden.

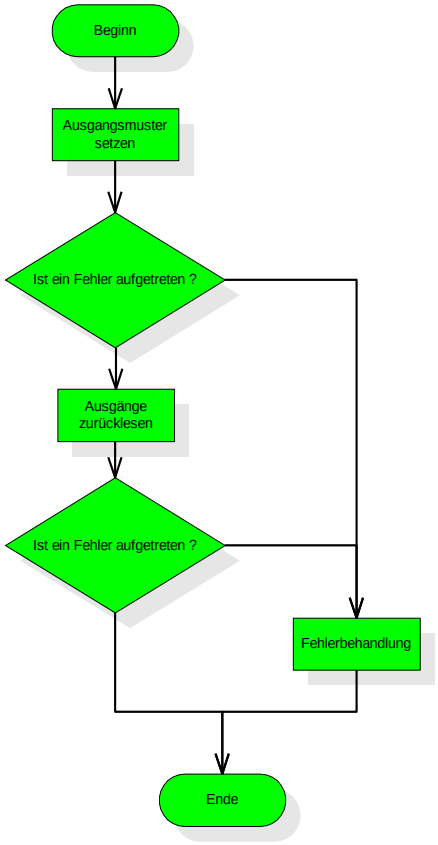
Es gibt grundsätzlich zwei Möglichkeiten die Ausgänge zu bedienen. Die Erste, hier gezeigte Möglichkeit, ist das Setzen eines speziellen Ausgangs.




```
...
unsigned char l_ucActIndex;                 /* Aktuell zu bedienende Karte */
unsigned char l_ucSetOutputState=1;         /*Status des gesetzten Ausgangs */
unsigned int l_ucNewOutputState;            /* Status des zuletzt gesetzten Ausgangs */
unsigned char l_ucChannel;                  /* Kanal der verändert werden soll */
...
/* Übernehmen der Adresse der zu bedienenden Karte aus den Kartendaten */
l_ucActIndex=l_SummeryBuffer[0].BoardNumber;
/* Setzen eines Ausgangs */
l_bReturnValue=pcidioDOSetChannelState(l_ucActIndex, /* Kartenadresse */
                                       l_ucChannel,  /* Zuverändernder Kanal */
                                       l_ucSetOutputState);/*Sollzustand */
/* Funktion erfolgreich abgeschlossen ? */
if(l_bReturnValue==TRUE)
{
 /* Warte min 250µs Schaltdelay  nebst ggf. lastabhängiger Verzögerung (hier 1ms)*/
 Sleep(1);
 /* Lese Kanalzustand zurück */
 l_bReturnValue=pcidioDIGetChannelState(l_ucActIndex,  /* Kartenadresse */
                                        l_ucChannel,    /* Kanal */
                                       &l_ucNewOutputState);/* akt. Zustand */
}
/* Sind Fehler aufgetreten */
if(l_bReturnValue==FALSE)
{
 /* Fehlertext auslesen */
 pcidioGetErrorMsg(l_strErrorTxt);
}
…
```

Hinweis
Die Funktion Sleep() kann durch eine Delay-Funktion ersetzt werden die im Minimum 250µs nebst ggf. lastabhängiger Verzögerung wartet.

Die zweite Möglichkeit die Ausgänge zu bedienen ist das Setzen aller Ausgangskanäle mit einem bestimmten Muster.

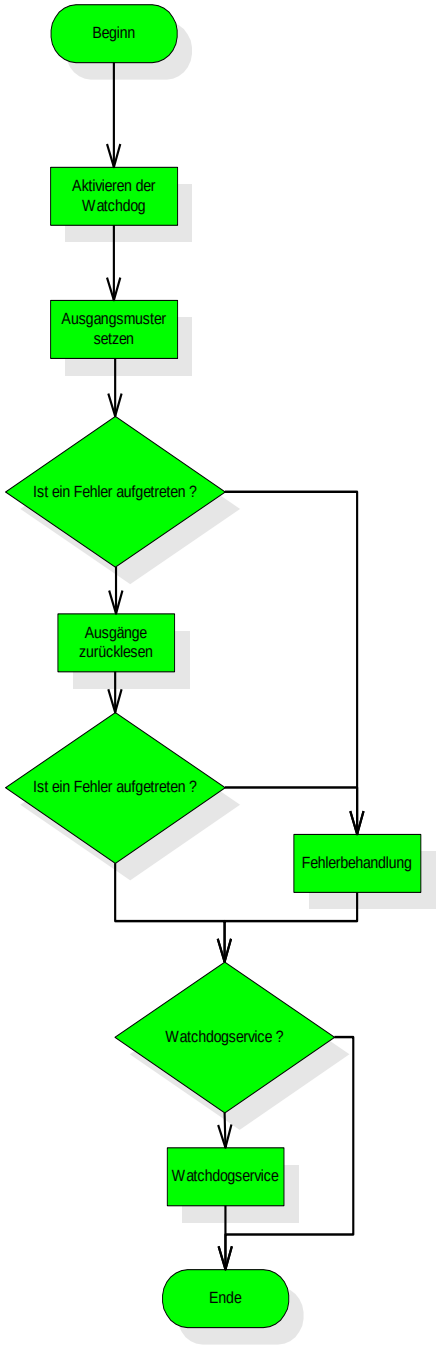



```
...
PCIDIOALLCHANNELS l_OutputStates;        /* Zusetzende Ausgänge */
PCIDIOALLCHANNELS l_NewOutputState;   /* Status der gesetzten Ausgangs */
...
/* Übernehmen der Adresse der zu bedienenden Karte aus den Kartendaten */
l_ucActIndex = l_SummeryBuffer[0].BoardNumber;
/* Setzen des Musters */
l_OutputState.Basis = 0xAAAAAAAA;
l_OutputState.Extension = 0x55555555;
/* Setzen der Ausgänge */
l_bReturnValue = pcidioDOSetState(l_ucActIndex, /* Kartenadresse */
                                  l_OutputState);/*Sollzustand */
/* Funktion erfolgreich abgeschlossen ? */
if(l_bReturnValue == TRUE)
{
 /* Warte min 250µs Schaltdelay nebst ggf. lastabhängiger Verzögerung (hier 1ms)*/
 Sleep(1);
 /* Lese Zustand zurück */
 l_bReturnValue = pcidioDIGetState(l_ucActIndex,  /* Kartenadresse */
                                   &l_NewOutputState);/* akt. Zustand */
}
/* Sind Fehler aufgetreten */
if(l_bReturn == FALSE)
{
 /* Fehlertext auslesen */
 pcidioGetErrorMsg(l_strErrorTxt);
}
…
```

Hinweis
Die Funktion Sleep() kann durch eine Delay-Funktion ersetzt werden die im Minimum 250µs nebst ggf. lastabhängiger Verzögerung wartet.

4.3 Bedienung der digitale Ausgänge mit aktivierter Watchdog

Die zweite Verwendungserläuterung betrifft das Bedienen der Ausgänge mit Aktivierung der Watchdog. In diesem Beispiel werden alle Ausgänge mit einem Muster versehen, das Ansteuern eines einzelnen Ausgangs verläuft analog dazu.




```
...
unsigned char l_ucWatchdogIntervall;        /* Watchdog Intervall */
unsigned char l_ucWatchdogService;          /* Flag für Watchdogservice */
PCIDIOALLCHANNELS l_OutputStates;       /* Zusetzende Ausgänge */
PCIDIOALLCHANNELS l_NewOutputState; /* Status der gesetzten Ausgangs */
...
/* Setzen des Musters */
l_OutputState.Basis=0xAAAAAAAA;
l_OutputState.Extension=0x55555555;
…
/* Watchdog starten */
l_bReturnValue = pcidioSetWatchdogIntervall(l_ucActIndex,
                                            l_ucWatchdogIntervall);
if(l_bReturnValue==TRUE)
{
 /* Setzen der Ausgänge */
 l_bReturnValue=pcidioDOSetState(l_ucActIndex, /* Kartenadresse */
                                 l_OutputState);/*Sollzustand */
 /* Funktion erfolgreich abgeschlossen ? */
 if(l_bReturnValue==TRUE)
 {
  /* Warte min 250µs Schaltdelay (hier 1ms)*/
  Sleep(1);
 /* Lese Zustand zurück */
  l_bReturnValue=pcidioDIGetState(l_ucActIndex,  /* Kartenadresse */
                                  &l_NewOutputState);/* akt. Zustand */
 }
}
/* Sind Fehler aufgetreten */
If(l_bReturnValue==FALSE)
{
 /* Fehlertext auslesen */
 pcidioGetErrorMsg(l_strErrorTxt);
}
…
/* Flag für Watchdogservice setzen nach gewisser Zeit */
l_ucWatchdogService= TRUE;
…
/* Ist Serviceflag gesetzt */
if(l_ucWatchdogService==TRUE)
{
 /*Ausgänge nachsetzen */
 pcidioDOServiceChannel(l_ucActIndex);
}
...
```

Das Flag l_ucWatchdogService wird im Programmablauf auf TRUE gesetzt und abgeprüft. Die Watchdogservice Funktion muss mindestens einmal innerhalb der programmierten

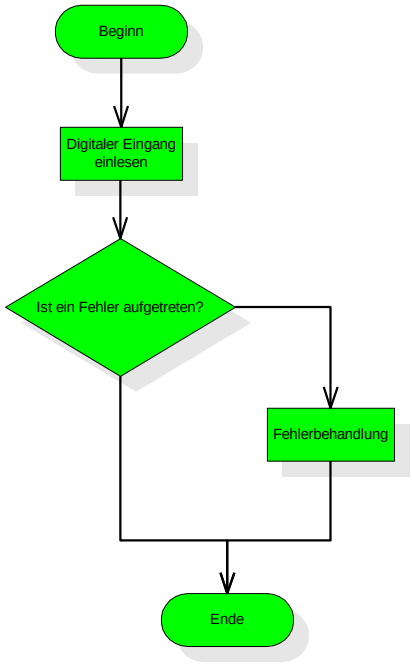
Timeout Zeit vom Benutzerprogramm aufgerufen werden, damit die Ausgänge nicht hardwareseitig zurückgesetzt werden.

Es ist weiterhin möglich, vor dem Aufruf der Watchdogservice Funktion den Status der Watchdog abzufragen. Dadurch kann erkannt werden, ob seit dem letzten Bedienen der Ausgänge die Watchdog zugeschlagen hat.
Ist dies der Fall so, ist im Benutzerprogramm ein Fehler vorhanden, der das rechtzeitige Nachsetzen der Watchdog verhindert.

4.4 Bedienung der digitalen Eingänge

Als nächstes wird nun das Einlesen der digitalen Eingänge demonstriert. Hierzu ist wiederum angemerkt, dass es zwei Möglichkeiten gibt: Den Einzelzugriff auf einen bestimmten Kanal und den Sammelzugriff auf alle Kanäle.

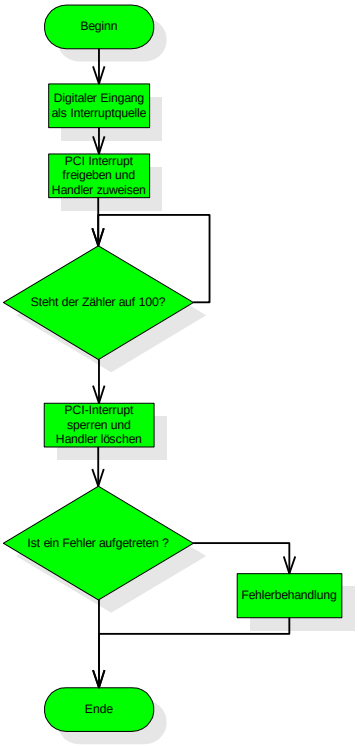
In diesem Kapitel wird als Beispiel der Einzelzugriff behandelt. Der Sammelzugriff erfolgt in gleicher Weise nur mit Hilfe eines anderen Funktionsaufrufs.



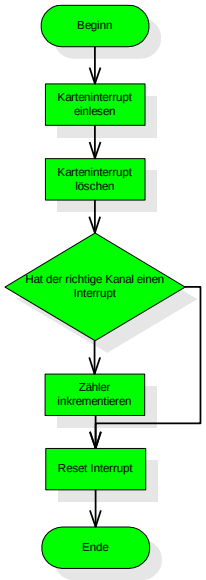

```
...
unsigned char l_ucInputState;
unsigned char l_ucChannel;
...
/* Einlesen des Eingangs */
l_bReturnValue = pcidioDIGetChannelState(l_ucActIndex,l_ucChannel,
                                         &l_ucInputState);
/* Ist ein Fehler aufgetreten ? */
if(l_bReturnValue!=0)
{
  /* Fehlertext auslesen */
  pcidioGetErrorMsg(l_strErrorTxt);
}
...
```

4.5 Digitale Eingänge als Interruptquelle

Diese Unterkapitel beschreibt die Verwendung der Eingangskanäle als Interruptquelle. Dabei wird ein Kanal als Interruptquelle bei steigender Flanke konfiguriert und im Benutzerinterrupthandler bei jeder steigenden Flanke ein Zähler inkrementiert.




```
...
unsigned char g_ucIntCounter;/* Goabaler Zähler */
...
unsigned char l_ucActIndex;
unsigned char l_ucChannel;
...
/* Interrupteingang konfigurieren */
l_bReturnValue = pcidioDISetIrqChannelConfiguration(l_ucActIndex,l_ucChannel,1,1);
if(l_bReturnValue  == TRUE)
{
 /* PCI Interrupt freigeben undHandler übergeben */
 l_bReturnValue= pcidioEnableIRQ(l_ucActIndex,&Inthandler);
 if(l_bReturnValue==TRUE)
 {
   g_ucIntCounter=0;
    while(g_ucIntCounter<100)
    {
      Sleep(1);
    }
 /* PCI-Interrupt sperren */
   l_bReturnValue=pcidioDisableIrq(l_ucActIndex);
 }
}
/* Ist ein Fehler aufgetreten ? */
if(l_bReturnValue ==FALSE)
{
 /* Fehlertext auslesen */
 pcidioGetErrorMsg(l_strErrorTxt);
}
...
```

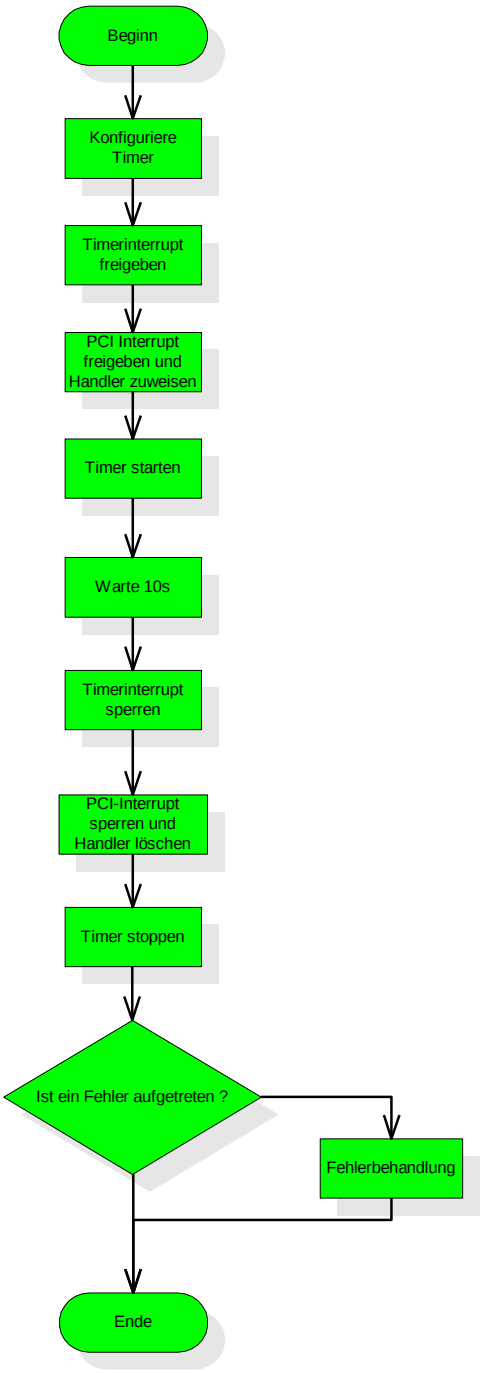


Interrupthandler

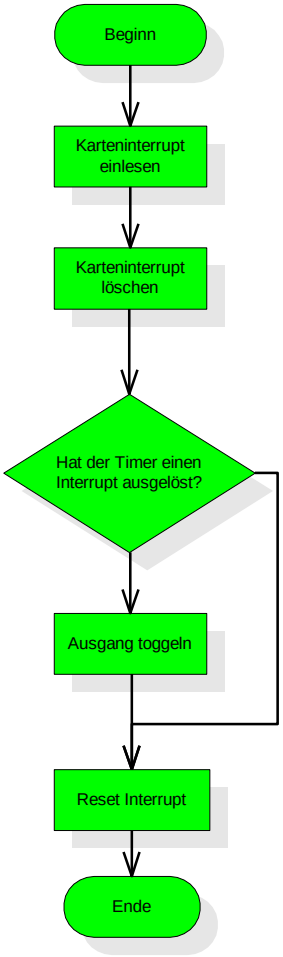
```
PCIDIO_INT_HANDLER  Inthandler(void)
{
  unsigned char l_ucActIndex;
 PCIDIO_INT_STATE   l_IntState;
  ...
/* Auslesen des Karteninterrupts *7
pcidioDIGetIrq (l_ucActIndex,&l_IntState);

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

4.6 Timerbehandlung mit Interruptbedienung

Hier wird nun die Timerprogrammierung mit Interruptbehandlung dargestellt. Im Interrupthandler der Timerprogrammierung wird bei einem Timerinterrupt im Interrupthandler ein Ausgang der cPCDIO umgeschaltet.




```
...
unsigned long l_ulTimerIntervall:
…
/* Timer stoppen */
 pcidioStopTimer(l_ucActIndex);
/* Timer einstellen */
l_bReturnValue =  pcidioSetTimer(l_ucActIndex,
                                 l_ulTimerIntervall)
if(l_bReturnValue == TRUE)
{
   /* Pci-Interrupt im System freigeben und Inthandler setzen */
   l_bReturnValue = pcidioEnableIrq(l_ucActIndex, &IntHandler);
  if(l_bReturnValue == TRUE)
  {
    /* Timerinterrupt auf der Karte freigeben */
    l_bReturnValue = pcidioSetIRQTimer(l_ucActIndex,1);
    if(l_bReturnValue == TRUE)
    {
      /* Starten des Timers */
      l_bReturnValue = pcidioStartTimer(l_ucActIndex);
      if(l_bReturnValue == TRUE)
      {/* 10s  Warten */
        Sleep(10000);
      }
       /* PCI-Interrupt im System sperren */
       l_bReturnValue = pcidioDisableIrq(l_ucActIndex);
      if(l_bReturnValue == TRUE)
      {
       /* Timerinterrupt auf der Karte sperren */
       l_iReturnValue = pcidioSetIRQTimer(l_ucActIndex,0);
       if(l_bReturnValue == TRUE)
       {
         /* Timer stoppen */
         l_bReturnValue = pcidioStopTimer(l_ucActIndex);
       }
    }
   }
  }
}
/* Ist ein Fehler aufgetreten ? */
if(l_bReturnValue == FALSE)
{
 /* Fehlertext auslesen */
 pcidioGetErrorMsg( l_strErrorTxt);
}
…
```



Interrupthandler

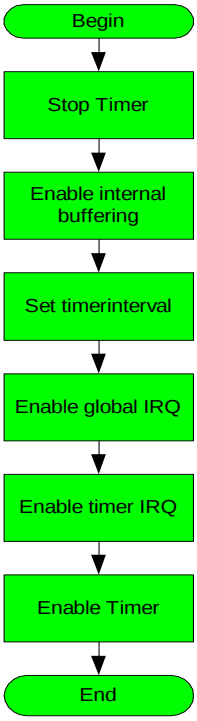
```
void Inthandler(void)
{
  unsigned char l_ucActIndex;
  unsigned char l_ucState;
  PCIDIO_INT_STATE   l_IntState;
   ...
/* Auslesen des Karteninterrupts *7
  pcidioDIGetIrq (l_ucActIndex,&l_IntState);

  if(l_IntState.IRQTIMER & 0x02 !=0)
  {
    /* Ausgangszustand einlesen */
    pcidioDIGetChannelState(l_ucActIndex, 1,&l_ucState);
   /* Ausgang setzen */
    pcidioDOSetChannelState(l_ucActIndex, 1, ~l_ucState);
 }
pcidioResetIRQ(l_ucActIndex)
...
}
```

4.7 Automatische Abtastung der Eingänge

Ab der Treiberversion 3.1 wird ein zeitgesteuertes Einlesen der digitalen Eingänge ermöglicht. Dazu wird ein interner Puffer angelegt, der mit Hilfe des karteninternen Timers als Triggerquelle gefüllt wird. Der intern angelegte Speicher ist ein Ringspeicher. Zu jedem abgelegten Zustand wird zusätzlich ein Zeitstempel in Timerticks abgelegt, der den Abtastzeitpunkt spezifiziert.

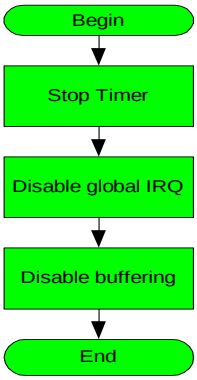
Diese Funktion ermöglicht die Aufnahme von digitalen Eingängen in einem spezifizierten Zeitraster ohne die Applikation damit zu belasten. Da für den Zeitpunkt der Speicherung der karteninterne Timer verwendet wird, ist das Triggerintervall konstant und applikationsunabhängig. Somit lassen sich Signalverläufe zu einer einstellbaren Zeitbasis aufnehmen und in der Applikation weiterverarbeiten.

> ***Hinweis***
> *Die Abtastrate des timergesteuerten Einlesens der digitalen Eingänge ist abhängig von der erzielbaren softwareseitigen Auflösung des Timerinterrupts. Diese Auflösung ist wiederum abhängig vom BIOS, dem PC, dem Betriebssystem und der Konfiguration des Gesamtsystems und muss im Einzelnen vom Anwender geprüft werden.*



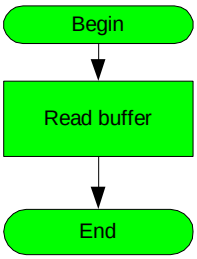
Initialisiere automatische Abtastung

```
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);
}
```



Deinitialisiere automatische Abtastung

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

  // Disable Interrupt
  pcidioDisableIrq(ucActIndex);

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



Lese automatische Abtastung aus

```
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 Treiberkonzept

Der Treiber der cPCIDIO ist für Microsoft Windows 7 (32 Bit und 64 Bit), Vista (32 Bit und 64 Bit), XP (32 Bit und 64 Bit) und 2K (32 Bit) als WDM-Treiber implementiert und besteht aus den Komponenten:

WDM-Treiber kp_pcidi.sys für Windows 7/Vista/XP und 2K
API-DLL pcidio.dll für Visual C++ mit der Aufrufkonvention cdecl und stdcall.

Diese Software Komponenten sind jeweils für 32-Bit Systeme und 64-Bit Systeme getrennt erhältlich.

Bei 64 Bit Systemen ist zusätzlich darauf zu achten, welche Applikation erstellt werden soll. Bei einer 32 Bit Applikation muss die DLL pcidio_32_64.dll und bei einer 64 Bit Applikation die DLL pcidio_64_64.dll mit den entsprechenden Lib-Dateien verwendet werden. Außerdem muss beim Kompilieren für 64 Bit Systeme die Präprozessordirektive KERNEL_64Bit verwendet werden.

Die API nach Außen unterscheidet sich bei den einzelnen Systemen nicht, nur die interne Datenverarbeitung ist unterschiedlich implementiert.

Das Tool PCIDIODemo ist als Quellcode im Lieferumfang enthalten und soll die Programmierung der einzelnen Kartenfunktionen unter Visual C++ darstellen.

Der DOS-Treiber wird in C-Sourcecode zur Verfügung gestellt.

5 API-Referenz

Dieses Kapitel beschreibt die Programmierschnittstelle der cPCIDIO unter Windows, die dem Anwendungsprogrammierer zur Verfügung steht.

> **Tipp**
> *Benutzen Sie bitte ausschließlich die hier dokumentierten Funktionen. Bei Verwendung von undokumentierten Features kann es zur Zerstörung der Karte bzw. der angeschlossenen Hardware kommen bzw. es besteht die Möglichkeit, dass diese Funktionen in der nächsten Version nicht mehr unterstützt werden.*

- Funktionsprototypen:
  In den nachfolgenden Funktionsbeschreibungen werden die Funktionsprototypen für VC++ verwendet.

- Blockierende Funktionsaufrufe
  Bitte beachten Sie, dass die Programmausführung immer erst dann fortgesetzt wird, wenn ein Funktionsaufruf komplett ausgeführt wurde. Dieses nennen wir im weiteren BLOCKING.

- Parameter
  Parameter bei Eingabe und Ausgabe steht für übergebene Variablen, Datenarrays oder Zeiger auf diese.

- Nomenklatur
  Alle Funktionsnamen besitzen den Familienpräfix *“pcidio“* und einen Funktionsgruppenpräfix

““	->	Allgemeine Funktionen
“DI“	->	Digitale Eingänge
“DO“	->	Digitale Ausgänge

  Die Funktionsnamen verwenden weitestgehend “selbstredende“ Bezeichnungen.

  Für die Funktionsdeklaration _cdecl wird kein Postfix verwendet. Für die Funktionsdeklaration _stdcall wird als Postfix StdCall an die Funktionsnamen angehängt.

  Durch diese Maßnahme ist es möglich, auch andere Programmiersprachen einzusetzen, bei denen eine DLL eingebunden werden kann.

5.1 Allgemeine Funktionen

pcidioGetCountBoards

Beschreibung

Liefert die Anzahl der gefundenen cPCIDIO Karten zurück und initialisiert den Treiber entsprechend.

> ***Hinweis***
> *Diese Funktion sollte am Anfang einer Applikation ausgeführt werden damit festgestellt wird, ob überhaupt Karten vorhanden sind.*

⇔ Parameter

⇨ Eingabe

keine

⇦ Ausgabe

keine

Rückgabe

Wurde die Funktion erfolgreich ausgeführt so wird die gefundene Kartenanzahl oder 0 für keine Karte vorhanden zurückgegeben.

pcidioInitCards

Beschreibung

Diese Funktion initialisiert alle im System vorhandenen cPCIDIO Karten und ordnet diese nach der Stellung der Adressjumper ein.

> ***Hinweis***
> *Diese Funktion sollte am Anfang einer Applikation ausgeführt werden, damit alle Karten korrekt initialisiert sind.*

⇔ Parameter

⇨ Eingabe

keine

⇦ Ausgabe

Anzahl der initialisierten Karten

Rückgabe

Wurde die Funktion erfolgreich ausgeführt so wird TRUE zurückgegeben, sonst FALSE.

pcidioDeinitCards

Beschreibung

Diese Funktion deinitialisiert alle im System vorhandenen cPCIDIO Karten und löscht den vom Treiber belegten Speicher.

> ***Hinweis***
> *Diese Funktion sollte am Ende einer Applikation ausgeführt werden, um den vom Treiber belegten Speicher wieder freizugegeben und den Interrupthandler zu löschen.*
>
> *Wir empfehlen, vor Aufruf dieser Funktion die Karte im Benutzerprogramm zusätzlich entweder global zu reseten oder aber wenigstens die Ausgänge zu löschen und die Interruptkonfigurationen zurückzusetzen.*

⇔ Parameter

⇨ Eingabe

keine

⇦ Ausgabe

keine

Rückgabe

keine

pcidioGetSummaryOfAllBoards

Beschreibung

Liefert eine Übersicht über die vorhandenen Karten. Diese Übersicht beinhaltet die Position der Karte im System, die Stellung des Kartenadressierungsjumpers sowie die Basisadresse der Karte im I/O-Bereich und die Kartennummer. Mit Hilfe dieser Daten ist es möglich, die Karten eindeutig im Programm zu verwenden.

⇔ Parameter

⇨ Eingabe

<SummaryBuffer>
Zeiger auf den Datenpuffer für die Übersichtsdaten. Der Zeiger muss auf ein genügend großes Datenarray vom Typ *PCIDIO_SUMMERY* zeigen.

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

<hPCIDIOHandle>
Handle auf die cPCIDIO Karte zur internen Verwendung.

<BoardIndex>
Index für die Adressierung der cPCIDIO Karte. Dieses Element wird für die Adressierung der Karte in allen Funktionen benötigt .

<BoardNumber>
Index für die Adressierung der cPCIDIO Karte. Dieses Element kann alternativ für die Adressierung der Karte in allen Funktionen verwendet werden.

<SlotNumber>
Nummer des Steckplatzes in dem sich die Karte befindet.

<BusNumber>
Nummer des Busses in dem sich die Karte befindet.

<BoardAdressJumper>
Stellung des Adressjumpers auf der Karte direkt zur primären Unterscheidung der einzelnen Karten.

<BoardIOAdress>
Adresse der Karte im I/O-Bereich des Rechnersystems.

> ☝ ***Hinweis***
> *Der Datenpuffer ist vom Applikationsentwickler anzulegen und an die Funktion zu übergeben.*
> 🕮 ***Tip***
> *Der Datenpuffer sollte immer acht Kartenübersichten aufnehmen können.*

⇦ Ausgabe

Der gefüllte Datenpuffer.

⇨ Rückgabe

Wurde die Funktion erfolgreich ausgeführt, so wird TRUE zurückgegeben. Ist während der Ausführung ein Fehler aufgetreten, wird FALSE zurückgegeben.
Mit Hilfe der Funktion *pcidioGetErrorMsg* kann dann der Fehler ermittelt werden.

pcidioGetBoardRevision

Beschreibung

Liefert die Revisionsnummer der Hardware zurück.

⇔ Parameter

⇨ Eingabe

<BoardNumber>
Mit der Funktion p*cidioGetSummaryOfAllBoards* ermittelter Index für die Adressierung der cPCIDIO.

⇦ Ausgabe

<BoardRevision>
Revision der Hardware in hexadezimal z.B. 02h.

Rückgabe

Wurde die Funktion erfolgreich ausgeführt so wird TRUE zurückgegeben. Ist während der Ausführung ein Fehler aufgetreten wird FALSE zurückgegeben.
Mit Hilfe der Funktion *pcidioGetErrorMsg* kann dann der Fehler ermittelt werden.

pcidioGetBoardAddressJumper

Beschreibung

Liefert die Stellung der Adressjumper S0, S1 und S2 zurück.

⇔ Parameter

⇨ Eingabe

<BoardNumber>
Mit der Funktion p*cidioGetSummaryOfAllBoards* ermittelter Index für die Adressierung der cPCIDIO.

⇦ Ausgabe

<BoardAdressJumper>
Stellung der Adressjumper auf der Karte direkt zur primären Unterscheidung der einzelnen Karten. Es werden Werte im Bereich von 0...7 zurückgegeben.

Rückgabe

Wurde die Funktion erfolgreich ausgeführt so wird TRUE zurückgegeben. Ist während der Ausführung ein Fehler aufgetreten wird FALSE zurückgegeben.
Mit Hilfe der Funktion *pcidioGetErrorMsg* kann dann der Fehler ermittelt werden.

pcidioGetBoardConfigurationData

Beschreibung

Liefert die einzelnen Ausbauzustände der adressierten cPCIDIO zurück.

⇔ Parameter

⇨ Eingabe

<BoardNumber>
Mit der Funktion p*cidioGetSummaryOfAllBoards* ermittelter Index für die Adressierung der cPCIDIO.

⇦ Ausgabe

<CntChannel>
Anzahl der verfügbaren digitalen Kanäle. Es wird der Wert 32 zurückgegeben.

⇐ Rückgabe

Wurde die Funktion erfolgreich ausgeführt so wird TRUE zurückgegeben. Ist während der Ausführung ein Fehler aufgetreten wird FALSE zurückgegeben.
Mit Hilfe der Funktion *pcidioGetErrorMsg* kann dann der Fehler ermittelt werden.

pcidioGetDriverVersion

Beschreibung

Liefert die Treiberversion der Karte zurück. Bei der Treiberversion handelt es sich um die Version der eingesetzten Treiber-DLL.

> 🕮 ***Tip***
> *Mit dieser Funktion kann überprüft werden, ob die richtige Treiberversion verwendet wird.*

⇔ Parameter

⇨ Eingabe

keine

⇦ Ausgabe

<DriverVersion>
Version des installierten Treibers.

⇐ Rückgabe

Wurde die Funktion erfolgreich ausgeführt so wird TRUE zurückgegeben. Ist während der Ausführung ein Fehler aufgetreten wird FALSE zurückgegeben.
Mit Hilfe der Funktion *pcidioGetErrorMsg* kann dann der Fehler ermittelt werden.

pcidioGetPCIConfiguration

Beschreibung

Liefert die PCI-Konfigurationsdaten der ausgewählten Karte zurück.

⇔ Parameter

⇨ Eingabe

<BoardNumber>
Mit der Funktion p*cidioGetSummaryOfAllBoards* ermittelter Index für die Adressierung der cPCIDIO.

<PCIConfiguration>
Zeiger auf eine Struktur des Typs *PCIHEADER* die von der Funktion mit Daten gefüllt wird.

```
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>
Beschreibt die Funktionsgruppe, in der die cPCIDIO Karte eingeordnet ist, mit dem hexadezimalen Wert 0004h.
<VendorID>
Beschreibt die Vendor Kennung der cPCIDIO mit dem hexadezimalen Wert 1172h.
<StateReg>
Beschreibt den PCI-Status der cPCIDIO.
<ControlReg>
Kontrollregister für den PCI-Bus
<ClassCode>
Enthält die Kartenklassenbeschreibung mit dem hexadezimalen Wert 118000h.
<RevisionID>
Beschreibt die Revision des FPGAs der Karte, z.B. 02h
<HeaderType>
Beschreibt die Form des PCI-Headers mit dem hexadezimalen Wert 00h.
<BaseAdress>

Beschreibt die Basisadresse (aus BAR0 & 0x0FFFC) der Karte im I/O-Bereich des PC-Systems für die direkte Programmierung.
<SubSysID>
Enthält die (Sub)Systemidentifikation der cPCIDIO mit dem hexadezimalen Wert 1026h.
<SubVenID>
Enthält die (Sub)Vendor Kennung mit dem hexadezimalen Wert EB84h.
<IrqPin>
Enthält den verwendeten PCI-Interruptpin.
<IrqLine>
Enthält die verwendete Interruptnummer.

⇦ Ausgabe

Gefülltes externes Datenarray

⇐ Rückgabe

Wurde die Funktion erfolgreich ausgeführt so wird TRUE zurückgegeben. Ist während der Ausführung ein Fehler aufgetreten wird FALSE zurückgegeben.
Mit Hilfe der Funktion *pcidioGetErrorMsg* kann dann der Fehler ermittelt werden.

pcidioSetTimer

Beschreibung

Setzt das Timerintervall der cPCIDIO mit dem übergebenen Wert.

> ***Hinweis***
> *Das Timerintervall setzt sich zusammen aus*
> *(Übergebener Wert + 1) * 100ns*
> *so dass Zeiten von 200ns bis zu 1,6777217s programmierbar sind.*

⇔ Parameter

⇨ Eingabe

<BoardNumber>
Mit der Funktion p*cidioGetSummaryOfAllBoards* ermittelter Index für die Adressierung der cPCIDIO.

<TimerInterval>
Wert für das Timerintervall in Counts von 1 (200ns) bis 2^{24} = 16777216 (1,6777217s)

⇦ Ausgabe

keine

⇐ Rückgabe

Wurde die Funktion erfolgreich ausgeführt so wird TRUE zurückgegeben. Ist während der Ausführung ein Fehler aufgetreten wird FALSE zurückgegeben.
Mit Hilfe der Funktion *pcidioGetErrorMsg* kann dann der Fehler ermittelt werden.

pcidioStartTimer

Beschreibung

Startet den Timer der cPCIDIO ohne Interruptbearbeitung.

> ***Hinweis***
> *Ist eine Interruptverarbeitung gewünscht so müssen vor dem Aufruf der Funktion die Funktionen pcidioSetIRQTimer und pcidioEnableIrq aufgerufen werden.*

⇔ Parameter

⇨ Eingabe

<BoardNumber>
Mit der Funktion p*cidioGetSummaryOfAllBoards* ermittelte Index für die Adressierung der cPCIDIO.

⇦ Ausgabe

keine

Rückgabe

Wurde die Funktion erfolgreich ausgeführt so wird TRUE zurückgegeben. Ist während der Ausführung ein Fehler aufgetreten wird FALSE zurückgegeben.
Mit Hilfe der Funktion *pcidioGetErrorMsg* kann dann der Fehler ermittelt werden.

pcidioStopTimer

Beschreibung

Stopt den Timer der cPCIDIO ohne den Interrupt zu sperren.

> ***Hinweis***
> *Um die Interruptbearbeitung zu beenden müssen nachdem Aufruf dieser Funktion noch die Funktionen pcdioSetIRQTimer und pcidioDisableIrq aufgerufen werden.*

⇔ Parameter

⇨ Eingabe

<BoardNumber>
Mit der Funktion p*cidioGetSummaryOfAllBoards* ermittelter Index für die Adressierung der cPCIDIO.

⇦ Ausgabe

keine

Rückgabe

Wurde die Funktion erfolgreich ausgeführt so wird TRUE zurückgegeben. Ist während der Ausführung ein Fehler aufgetreten wird FALSE zurückgegeben.
Mit Hilfe der Funktion *pcidioGetErrorMsg* kann dann der Fehler ermittelt werden.

pcidioSetIRQTimer

Beschreibung

Sperrt und gibt den Timerinterrupt für die Verarbeitung frei.

> ***Hinweis***
> *Es wird mit dieser Funktion nicht der PCI-Interrupt selbst global freigegeben bzw. gesperrt. Diese Funktion sperrt oder aktiviert nur lokal den Timerinterrupt in der Karteninterruptsmaske.*

⇔ Parameter

⇨ Eingabe

<BoardNumber>
Mit der Funktion p*cidioGetSummaryOfAllBoards* ermittelte Index für die Adressierung der cPCIDIO.

<CntrlByte>
Kontrollregister für die Steuerung des Timerinterrupts (1->Freigabe, 0->Sperren)

⇦ Ausgabe

keine

Rückgabe

Wurde die Funktion erfolgreich ausgeführt so wird TRUE zurückgegeben. Ist während der Ausführung ein Fehler aufgetreten wird FALSE zurückgegeben.
Mit Hilfe der Funktion *pcidioGetErrorMsg* kann dann der Fehler ermittelt werden.

pcidioSetWatchdogIntervall

Beschreibung

Setzt die Timeout Zeit der Watchdog für die cPCIDIO mit dem übergebenen Wert und startet diese.

> ***Hinweis***
> *Das Setzen der Timeout Zeit hat ein unverzügliches Starten der Hardwarewatchdog zur Folge. Die Watchdog kann nur mit einem Hardwarereset bzw. mit der entsprechenden Softwarefunktion pcidioReset abgeschaltet bzw. die Timeout Zeit wieder neu einstellbar gemacht werden.*

> ***Hinweis***
> *Ist die Watchdog gestartet, muss mindestens einmal innerhalb der Timeout Zeit ein Ausgang angesteuert werden, damit die Ausgänge nicht durch die Watchdog abgeschaltet werden.*

⇔ Parameter

⇨ Eingabe

<*BoardNumber*>
Mit der Funktion p*cidioGetSummaryOfAllBoards* ermittelter Index für die Adressierung der cPCIDIO.

<*WatchdogInterval*>
Wert für die Timeout Zeit der Watchdog in Counts von 1 (26,2144ms)...255 (6,684672s)

⇦ Ausgabe

keine

⇐ Rückgabe

Wurde die Funktion erfolgreich ausgeführt so wird TRUE zurückgegeben. Ist während der Ausführung ein Fehler aufgetreten wird FALSE zurückgegeben.
Mit Hilfe der Funktion *pcidioGetErrorMsg* kann dann der Fehler ermittelt werden.

pcidioGetWatchdogState

Beschreibung

Liefert den Status der Watchdog zurück, ob diese ausgelöst hat.

⇔ Parameter

⇨ Eingabe

<*BoardNumber*>
Mit der Funktion p*cidioGetSummaryOfAllBoards* ermittelte Index für die Adressierung der cPCIDIO.

⇦ Ausgabe

<*WatchdogState*>
Status der Watchdog mit
0 für Watchdog hat nicht ausgelöst und
1 für Watchdog hat ausgelöst.

⇐ Rückgabe

Wurde die Funktion erfolgreich ausgeführt so wird TRUE zurückgegeben. Ist während der Ausführung ein Fehler aufgetreten wird FALSE zurückgegeben.
Mit Hilfe der Funktion *pcidioGetErrorMsg* kann dann der Fehler ermittelt werden.

pcidioReset

Beschreibung

Diese Funktion setzt die Hardware der cPCIDIO komplett in den Defaultzustand zurück.

⇔ Parameter

⇨ Eingabe

<BoardNumber>
Mit der Funktion p*cidioGetSummaryOfAllBoards* ermittelter Index für die Adressierung der cPCIDIO.

⇦ Ausgabe

keine

Rückgabe

Wurde die Funktion erfolgreich ausgeführt so wird TRUE zurückgegeben. Ist während der Ausführung ein Fehler aufgetreten wird FALSE zurückgegeben.
Mit Hilfe der Funktion *pcidioGetErrorMsg* kann dann der Fehler ermittelt werden.

pcidioGetErrorMsg

Beschreibung

Diese Funktion liefert den letzten vorhandenen Fehlertext zurück.

⇔ Parameter

⇨ Eingabe

keine

⇦ Ausgabe

<ErrorMsg>
Zeiger auf ein genügend großes externes vom Programmierer anzulegendes Textfeld (min. 100 Zeichen), in das der String von der Funktion kopiert wird.

Rückgabe

keine

pcidioEnableIrq

Beschreibung

Diese Funktion installiert den benutzerspezifischen Interrupthandler und enabled den PCI-Interrupt global aber nicht die jeweiligen lokalen Masken.

⇔ Parameter

⇨ Eingabe

<BoardNumber>
Mit der Funktion p*cidioGetSummaryOfAllBoards* ermittelter Index für die Adressierung der cPCIDIO.

<IntHandler>
Funktionspointer vom Type *PCIDIO_INT_HANDLER* auf den Interrupthandler des Benutzers.

⇦ Ausgabe

keine

Rückgabe

Wurde die Funktion erfolgreich ausgeführt so wird TRUE zurückgegeben. Ist während der Ausführung ein Fehler aufgetreten wird FALSE zurückgegeben.
Mit Hilfe der Funktion *pcidioGetErrorMsg* kann dann der Fehler ermittelt werden.

pcidioDisableIrq

Beschreibung

Diese Funktion disabled den PCI-Interrupt global aber nicht die jeweiligen lokalen Masken.

⇔ Parameter

⇨ Eingabe

<BoardNumber>
Mit der Funktion p*cidioGetSummaryOfAllBoards* ermittelte Index für die Adressierung der cPCIDIO.

⇦ Ausgabe

keine

Rückgabe

Wurde die Funktion erfolgreich ausgeführt so wird TRUE zurückgegeben. Ist während der Ausführung ein Fehler aufgetreten wird FALSE zurückgegeben.
Mit Hilfe der Funktion *pcidioGetErrorMsg* kann dann der Fehler ermittelt werden.

pcidioGetIrq

Beschreibung

Diese Funktion liefert den Inhalt der karteninternen Interruptregister beim Auftreten des letzten Interrupts.

⇔ Parameter

⇨ Eingabe

<BoardNumber>
Mit der Funktion p*cidioGetSummaryOfAllBoards* ermittelter Index für die Adressierung der cPCIDIO.

<Int_State>
Array für den Registerstatus.

```
typedef PCIDIO_INT_STATE
 {
   BYTE BoardNumber;   /* Kartenzuordnung */
   DWORD IRQIO_1_32; /* Interruptregister der Basiskarte*/
   DWORD RESERVED; /*Reserviert für zukünftige Erweiterungen */
   BYTE IRQTIMER;  /* Interruptregister für den Timer  */
   BYTE PERREG;   /* Globales Interruptregister der Karte */
 }
```

< BoardNumber>
Mit der Funktion p*cidioGetSummaryOfAllBoards* ermittelte Index für die Adressierung der cPCIDIO.
<IRQIO_1_32>
Das Register liefert den Status der jeweiligen Interruptquellen der Eingänge 0 bis 31. Ein gesetztes Bit signalisiert dabei einen anstehenden Interrupt.
<RESERVED>
Reserviert für zukünftige Erweiterungen.
<IRQTIMER>
Die Variable enthält den Status und den Interruptstatus des Timers in den niederen beiden Bitpositionen:

Bit 1	Bit 0	Bedeutung
0	0	Timer läuft nicht und kein Timerinterrupt war ausgelöst
0	1	Timer läuft und kein Timerinterrupt war ausgelöst
1	0	Timer läuft nicht mehr und Timerinterrupt war ausgelöst
1	1	Timer läuft und Timerinterrupt war ausgelöst

<PERREG>
Dieses Register bietet eine weitere Möglichkeit der Ermittlung der Karte als Interruptquelle:

Bit 0: Dieses Bit ist 1, solange wenigstens ein Interrupt der Karte ansteht
Bit 2: Dieses Bit ist 1, solange der Timerinterrupt ansteht
Bit 3: Dieses Bit ist 1, solange wenigstens einer der Interrupts der Eingänge der cPCIDIO ansteht

⇦ Ausgabe

keine

⇦ Rückgabe

Wurde die Funktion erfolgreich ausgeführt so wird TRUE zurückgegeben. Ist während der Ausführung ein Fehler aufgetreten wird FALSE zurückgegeben.
Mit Hilfe der Funktion *pcidioGetErrorMsg* kann dann der Fehler ermittelt werden.

pcidioResetIrq

Beschreibung

Diese Funktion setzt im User Mode den Interrupthandler des Kernelinterrupt der cPCIDIO Karte zurück. Diese Funktion muss im User-Mode am Ende im User-Mode Interrupthandlers aufgerufen werden.

⇔ Parameter

⇨ Eingabe

<*BoardNumber*>
Mit der Funktion p*cidioGetSummaryOfAllBoards* ermittelter Index für die Adressierung der cPCIDIO.

⇦ Ausgabe

keine

⇦ Rückgabe

Wurde die Funktion erfolgreich ausgeführt so wird TRUE zurückgegeben. Ist während der Ausführung ein Fehler aufgetreten wird FALSE zurückgegeben.
Mit Hilfe der Funktion *pcidioGetErrorMsg* kann dann der Fehler ermittelt werden.

5.2 Digital Eingangsfunktionen

pcidioDIGetChannelState

Beschreibung

Liefert den Zustand des übergebenen Kanals zurück.

⇔ Parameter

⇨ Eingabe

<BoardNumber>
Mit der Funktion p*cidioGetSummaryOfAllBoards* ermittelter Index für die Adressierung der cPCIDIO.

<Channel>
Kanalnummer 0...31 des gewünschten Eingangskanal.

⇦ Ausgabe

<ChannelState>
Zustand des Kanals: eine ‚1' entspricht einem High-Pegel und eine ‚0' einem Low-Pegel am entsprechenden I/O Pin des Steckverbinders der cPCIDIO.

Rückgabe

Wurde die Funktion erfolgreich ausgeführt so wird TRUE zurückgegeben. Ist während der Ausführung ein Fehler aufgetreten wird FALSE zurückgegeben.
Mit Hilfe der Funktion *pcidioGetErrorMsg* kann dann der Fehler ermittelt werden.

pcidioDIGetByte

Beschreibung

Diese Funktion ist ab der Treiber-DLL Version 3.0 verfügbar und liefert den Zustand von einer Eingangsgruppe (8 Kanäle) zurück.

⇔ Parameter

⇨ Eingabe

<BoardNumber>
Mit der Funktion p*cidioGetSummaryOfAllBoards* ermittelter Index für die Adressierung der cPCIDIO.

<ByteNumber>
Nummer der 8 kanaligen Eingangsgruppe 0..3, die eingelesen werden soll

⇦ Ausgabe

<State>
Zustand der acht übergebenen Eingangskanäle der ausgewählten Karte.

➲ Rückgabe

Wurde die Funktion erfolgreich ausgeführt so wird TRUE zurückgegeben. Ist während der Ausführung ein Fehler aufgetreten wird FALSE zurückgegeben.
Mit Hilfe der Funktion *pcidioGetErrorMsg* kann dann der Fehler ermittelt werden.

pcidioDIGetState

Beschreibung

Liefert den Zustand aller Eingänge auf einmal zurück.

⇔ Parameter

⇨ Eingabe

<BoardNumber>
Mit der Funktion p*cidioGetSummaryOfAllBoards* ermittelter Index für die Adressierung der cPCIDIO.

⇦ Ausgabe

<AllChannelState>
Zustand aller Eingangskanäle der ausgewählten Karte. Die Variable ist eine Struktur aus zwei 32-Bit langen Werten und jedes Bit steht für einen Kanal.

```
typedef struct PCIDIOALLCHANNELS
 {
   DWORD Basis;          /* Zustand der Eingänge */
   DWORD RESERVED; /*Reserviert für zukünftige Erweiterungen */
 }
```

➲ Rückgabe

Wurde die Funktion erfolgreich ausgeführt so wird TRUE zurückgegeben. Ist während der Ausführung ein Fehler aufgetreten wird FALSE zurückgegeben.
Mit Hilfe der Funktion *pcidioGetErrorMsg* kann dann der Fehler ermittelt werden.

pcidioDISetIrqChannelConfiguration

Beschreibung

Setzt für einen Eingangskanal die Interruptkonfiguration.

> ***Hinweis***
> *Es wird mit dieser Funktion nicht der PCI-Interrupt global freigegeben bzw. gesperrt. Diese Funktion sperrt oder aktiviert nur den jeweiligen lokalen I/O Interrupt in der Karteninterruptmaske.*

⇔ Parameter

⇨ Eingabe

<*BoardNumber*>
Mit der Funktion p*cidioGetSummaryOfAllBoards* ermittelter Index für die Adressierung der cPCIDIO.

<*Channel*>
Kanalnummer 0...31 des gewünschten Eingangskanal.

<ChannelIntEnable>
Gibt mit ‚1' den Eingangskanal als Interruptquelle frei oder sperrt mit ‚0' den Kanal wieder.

<ChannelTrigger>
Legt den Triggerzeitpunkt des Interrupts fest, wobei ‚0' für eine Auslösung bei fallender Flanke und ‚1' für eine Auslösung bei einer steigenden Flanke steht.

⇦ Ausgabe

keine

➲ Rückgabe

Wurde die Funktion erfolgreich ausgeführt so wird TRUE zurückgegeben. Ist während der Ausführung ein Fehler aufgetreten wird FALSE zurückgegeben.
Mit Hilfe der Funktion *pcidioGetErrorMsg* kann dann der Fehler ermittelt werden.

pcidioDIGetIrqChannelConfiguration

🗐 Beschreibung

Liefert die Interruptkonfiguration eines Eingangs zurück.

⇔ Parameter

⇨ Eingabe

<*BoardNumber*>
Mit der Funktion p*cidioGetSummaryOfAllBoards* ermittelter Index für die Adressierung der cPCIDIO.

<*Channel*>
Kanalnummer 0...31 des gewünschten Eingangskanal

⇦ Ausgabe

<*ChannelIntEnable*>
Bei einer ‚1' ist der Eingang als Interruptquelle lokal enabled, bei einer ‚0' dagegen gesperrt.

<*ChannelTrigger*>
Bei einer ‚0' ist für den Eingang die fallende Flanke, bei einer ‚1' die steigende Flanke für den Triggerzeitpunkt ausgewählt.

⇦ Rückgabe

Wurde die Funktion erfolgreich ausgeführt so wird TRUE zurückgegeben. Ist während der Ausführung ein Fehler aufgetreten wird FALSE zurückgegeben.
Mit Hilfe der Funktion *pcidioGetErrorMsg* kann dann der Fehler ermittelt werden.

pcidioDIEnableBuffering

Beschreibung

Diese Funktion gibt das interne Abtasten und Abspeichern der digitalen Eingänge der Karte frei, wenn ein Timerinterrupt des karteninternen Timers auftritt.

⇔ Parameter

⇨ Eingabe

<BoardNumber>
Mit der Funktion p*cidioGetSummaryOfAllBoards* ermittelter Index für die Adressierung der cPCIDIO.

⇦ Ausgabe

keine

⇦ Rückgabe

Wurde die Funktion erfolgreich ausgeführt so wird TRUE zurückgegeben. Ist während der Ausführung ein Fehler aufgetreten wird FALSE zurückgegeben.
Mit Hilfe der Funktion *pcidioGetErrorMsg* kann dann der Fehler ermittelt werden.

pcidioDIDisableBuffering

Beschreibung

Diese Funktion sperrt das interne Abtasten und Abspeichern.

⇔ Parameter

⇨ Eingabe

<BoardNumber>
Mit der Funktion p*cidioGetSummaryOfAllBoards* ermittelter Index für die Adressierung der cPCIDIO.

⇦ Ausgabe

keine

⇦ Rückgabe

Wurde die Funktion erfolgreich ausgeführt so wird TRUE zurückgegeben. Ist während der Ausführung ein Fehler aufgetreten wird FALSE zurückgegeben.
Mit Hilfe der Funktion *pcidioGetErrorMsg* kann dann der Fehler ermittelt werden.

pcidioDIGetBufferedSamples

Beschreibung

Diese Funktion liest die intern gespeicherten Abtastungen der digitalen Eingänge aus.

⇔ Parameter

⇨ Eingabe

<*BoardNumber*>
Mit der Funktion p*cidioGetSummaryOfAllBoards* ermittelter Index für die Adressierung der cPCIDIO.

<bufferSize>
Größe der externen angelegten Speicherstrukturen. Es wird ein Array für die Eingangszustände mit dem Format PCIDIOCHANNELS und ein Array für einen Zeitstempel mit dem Format LARGE_INTEGER benötigt. Beide Arrays müssen die gleiche Länge bufferSize besitzen.

⇦ Ausgabe

<Samples>
Gefülltes Array mit Eingangszuständen die aufgenommen wurden.

<TimeStamp>
Gefülltes Array mit zu den Zeitstempeln korrespondierend mit Eingangszuständen im Samples-Array.

<CounterActSamples>
Anzahl an Samples die in den zurückgegebenen Arrays vorhanden sind.

<BufferOverflow>
Flag welches anzeigt, ob der interne Puffer zwischen zwei Lesevorgängen übergelaufen ist. Es können intern maximal 1000 Eingangszustände zwischengespeichert werden bis es zum Überlauf kommt. Ist dieses Flag 1 so hat ein Überlauf stattgefunden und die zurückgelieferten Daten bei diesem Aufruf sind unvollständig, ist das Flag 0 so sind die alle zurückgegebenen Daten gültig.

➲ Rückgabe

Wurde die Funktion erfolgreich ausgeführt so wird TRUE zurückgegeben. Ist während der Ausführung ein Fehler aufgetreten wird FALSE zurückgegeben.
Mit Hilfe der Funktion *pcidioGetErrorMsg* kann dann der Fehler ermittelt werden.

5.3 Digital Ausgangsfunktionen

pcidioDOSetChannelState

Beschreibung

Setzt den übergebenen Ausgang auf den übergebenen Pegel.

⇔ Parameter

⇨ Eingabe

<BoardNumber>
Mit der Funktion p*cidioGetSummaryOfAllBoards* ermittelter Index für die Adressierung der cPCIDIO.

<Channel>
Kanalnummer 0...31 des gewünschten Ausgangskanal.

<ChannelState>
Zustand des Kanals. ‚1' entspricht einem High-Pegel und ‚0' einem Low-Pegel am entsprechenden I/O Pin des Steckverbinders der cPCIDIO.

⇦ Ausgabe

keine

Rückgabe

Wurde die Funktion erfolgreich ausgeführt so wird TRUE zurückgegeben. Ist während der Ausführung ein Fehler aufgetreten wird FALSE zurückgegeben.
Mit Hilfe der Funktion *pcidioGetErrorMsg* kann dann der Fehler ermittelt werden.

pcidioDOSetByte

Beschreibung

Diese Funktion ist ab der Treiber-DLL Version 3.0 verfügbar und setzt den Zustand der übergebenen 8 kanaligen Ausgangsgruppe.

> ***Hinweis***
> *Die Kanäle, die als Eingang betrieben werden oder nicht benutzt sind, sind immer auf ‚0' zu initialisieren.*

⇔ Parameter

⇨ Eingabe

<BoardNumber>
Mit der Funktion p*cidioGetSummaryOfAllBoards* ermittelter Index für die Adressierung der cPCIDIO.

<ByteNumber>
Nummer der 8 kanaligen Ausgangsgruppe 0..3, die gesetzt werden soll

<State>
Zustand der Ausgangskanäle der ausgewählten Karte.

⇦ Ausgabe
keine

➲ Rückgabe
Wurde die Funktion erfolgreich ausgeführt so wird TRUE zurückgegeben. Ist während der Ausführung ein Fehler aufgetreten wird FALSE zurückgegeben.
Mit Hilfe der Funktion *pcidioGetErrorMsg* kann dann der Fehler ermittelt werden.

pcidioDOSetState

Beschreibung
Setzt den Zustand aller Ausgangskanäle.

> ***Hinweis***
> *Die Kanäle, die als Eingang betrieben werden oder nicht benutzt sind, sind immer auf ‚0' zu initialisieren.*

⇔ Parameter

⇨ Eingabe
<BoardNumber>
Mit der Funktion p*cidioGetSummaryOfAllBoards* ermittelter Index für die Adressierung der cPCIDIO.
<AllChannelState>
Zustand aller Ausgangskanäle der ausgewählten Karte. Die übergebene Struktur enthält für jeweils 32 Kanäle eine Variable. Nicht vorhandene bzw. als Eingang verwendete Kanäle sind immer auf ‚0' zu setzen.

```
typedef struct PCIDIOALLCHANNELS
{
  DWORD Basis;          /* Zustand der Ausgänge */
  DWORD RESERVED;  /* Reserviert für zukünftige Erweiterungen*/
}
```

⇦ Ausgabe
keine

➲ Rückgabe
Wurde die Funktion erfolgreich ausgeführt so wird TRUE zurückgegeben. Ist während der Ausführung ein Fehler aufgetreten wird FALSE zurückgegeben.
Mit Hilfe der Funktion *pcidioGetErrorMsg* kann dann der Fehler ermittelt werden.

pcidioDOServiceChannel

Beschreibung

Setzt die Kanäle für die Watchdogansteuerung nach.

> ***Hinweis***
> *Wird die Watchdog verwendet, sollte diese Funktion mindestens einmal innerhalb der programmierten Timeoutzeit der Watchdog aufgerufen werden, wenn keine andere Ausgangsfunktion verwendet wird.*

⇔ Parameter

⇨ Eingabe

<BoardNumber>
Mit der Funktion p*cidioGetSummaryOfAllBoards* ermittelter Index für die Adressierung der cPCIDIO.

⇦ Ausgabe

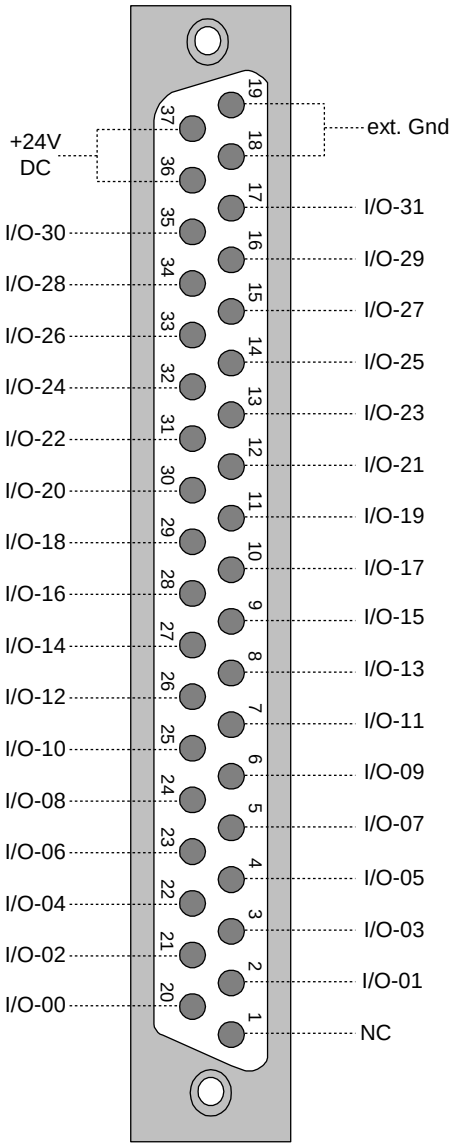
keine

Rückgabe

Wurde die Funktion erfolgreich ausgeführt so wird TRUE zurückgegeben. Ist während der Ausführung ein Fehler aufgetreten wird FALSE zurückgegeben.
Mit Hilfe der Funktion *pcidioGetErrorMsg* kann dann der Fehler ermittelt werden.

Anhang

A Steckerbelegung SUBD-37-Buchse cPCIDIO



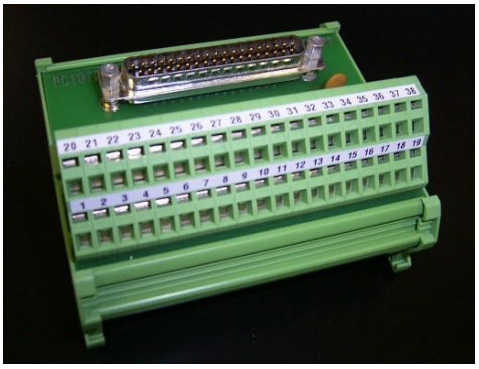
Die externe Spannungsversorgung kann auch intern auf der Karte an einem zweipoligen Steckverbinder mit Schraubklemmenanschluss aufgelegt oder abgegriffen werden:

Beschreibung	Pin
+24V DC	1
GND	2

Pin 1 ist dabei von vorne betrachtet der linke Anschluss der Schraubklemmenblocks. Für die Verdrahtung empfehlen wir die Verwendung einer Litze mit 0,75mm^2 (max. 2,5mm^2).

B Klemmenmodul PCIDIOHM

Zur einfachen Anlagenaufschaltung im Schaltschrank steht für die DIN-Normschiene (TS 35 und TS 32) das optionale Klemmenmodul PCIDIOHM mit Käfigzugklemmen und den Maßen ca. 102,5 mm (Breite) * 90,0 mm (Tiefe) * 60,0 mm (Höhe) zur Verfügung.



Das Modul ist optional auch mit Wandhalterung und Schraubklemmen erhältlich.

Die Anlagenaufschaltung erfolgt an den Käfigzugklemmen mit Litzen 0,25 mm^2 bis 1,5 mm^2 (auch mit Aderendhülsen).

Für die Verbindung zur cPCIDIO über den SUBD Anschluss des Hutschienenmoduls empfehlen wir die Verwendung von geschirmten Verbindungskabeln wie z.B. PCIDIOVK.

Bezeichnung (*)	Kontakt Klemmleiste	Pin SUBD Stecker	Bezeichnung (*)	Kontakt Klemmleiste	Pin SUBD Stecker
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: Steckerbelegung von Klemmleiste und SUBD Stecker von PCIDIOHM

(*) Bezeichnung bei Verwendung eines 1:1 aufgelegten Verbindungskabels zur cPCIDIO
(**) Gehäuse des Steckers über Y-Kondensator auf Klemme 38 der Klemmleiste verdrahtet

Wir empfehlen dringend, die Erde auf Pin 38 auch aufzulegen um Störungen auf dem Verbindungskabel zu vermeiden.

C DOS Treiber

Für den Betrieb der cPCIDIO unter dem Betriebssystem DOS steht ein entsprechender DOS Treiber im Sourcecode zur Verfügung.

Die Initialisierung erfolgt hier über PCI BIOS Erweiterungen und die Kartenansprache über I/O Befehle. Nähere Informationen hierzu sind der entsprechenden Readme Datei und den Quellcodekommentaren des Sourcecodes zu entnehmen.

Für die Programmierung unter DOS ist empfohlen, auch die technische Dokumentation heranzuziehen.

D Artikelnummern

Artikel Nr.	Bezeichnung
cPCIDIO	cPCIDIO Karte mit 32 I/Os, Watchdog, Timer
PCIDIOHM	Optionales Klemmenmodul mit Käfigzugklemmleiste und SUBD Anschluss für die DIN-Normschiene zur einfachen Anlagenaufschaltung im Schaltschrank für 32 I/Os
PCIDIOVK1M	Optionales Verbindungskabel 1m 37-polig 1:1 aufgelegt für die Verbindung zwischen cPCIDIO und Klemmenmodul PCIDIOHM
PCIDIOVK2M	Optionales Verbindungskabel 2m 37-polig 1:1 aufgelegt für die Verbindung zwischen cPCIDIO und Klemmenmodul PCIDIOHM 

E Support

Sollten Sie Fragen zu unserem Produkt haben oder Hilfe benötigen, wenden Sie sich bitte mit entsprechenden ausführlichen Angaben über Ihr Problem an unseren Support, der Ihnen gerne weiter hilft.

Email: support@ebru.de
Tel. 06228/913710 (Mo.-Fr. 8.00-17.00)
Fax 06228/913729

F Kundenspezifische Ausführungen

Als Dienstleister rund um die industrielle Elektronik führen wir für Sie gerne auch kundenspezifische Anpassungen oder Erweiterungen durch. Gerade auch dadurch, dass das PCI Interface gemeinsam mit der gesamten Steuerung von uns in ein FPGA integriert wurde, sind vielfältige Eingriffsmöglichkeiten vorhanden.

G Serviceadresse

Wir hoffen, dass sie diese Serviceadresse nie benötigen werden. Sollte jedoch trotz sorgfältiger Produktion und Prüfung eine Funktionsstörung auftreten, wenden Sie sich bitte an:

EBRU GmbH
In den Kreuzwiesen 21
D-69250 Schönau

Falls Sie Ihre Karte zur Reparatur einschicken, legen Sie bitte eine möglichst ausführliche Fehlerbeschreibung bei. Dadurch ist eine schnellere Bearbeitung möglich.

H Updates

Updates von der Treibersoftware und den Dokumentationen werden im Internet auf unseren Internetseiten www.ebru.de zur Verfügung gestellt.