# CPCI-DIO72

## CompactPCI-digital I/O-Karte

Hardware-Installation
und
technische Daten

| | |
|---|---|
| **Dokument-Datei:** | I:\texte\Doku\MANUALS\CPCI\DIO72\CPI7212H.ma9 |
| **Datum des Ausdrucks:** | 20.12.2001 |

| | |
|---|---|
| **Platinenversion:** | CPCI-DIO72 Rev. 1.1 |

**Änderungen in den Kapiteln**

Die hier aufgeführten Änderungen im Dokument betreffen sowohl Änderungen in der Hardware als auch reine Änderungen in der Beschreibung der Sachverhalte.

| Kapitel | Änderungen gegenüber Vorversion |
|---|---|
| 5. | Weitere Stecker-Typen für X400. |
| - | |

Weitere technische Änderungen vorbehalten.

Der Inhalt dieses Handbuches wurde mit größter Sorgfalt erarbeitet und geprüft. **esd** übernimmt jedoch keine Verantwortung für Schäden, die aus Fehlern in der Dokumentation resultieren könnten. Insbesondere Beschreibungen und technische Daten sind keine zugesicherten Eigenschaften im rechtlichen Sinne.

**esd** hat das Recht, Änderungen am beschriebenen Produkt oder an der Dokumentation ohne vorherige Ankündigung vorzunehmen, wenn sie aus Gründen der Zuverlässigkeit oder Qualitätssicherung vorgenommen werden oder dem technischen Fortschritt dienen.



**esd electronic system design gmbh**
Vahrenwalder Str. 207
30165 Hannover

Tel.: 0511/372 98-0
FAX : 0511/372 98-68
E-Mail: info@esd.electronics.com
Internet: www.esd-electronics.com

# Inhalt

Diese Seite ist bewußt unbedruckt.

# 1. Übersicht

## 1.1 Beschreibung der CPCI-DIO72-Karte

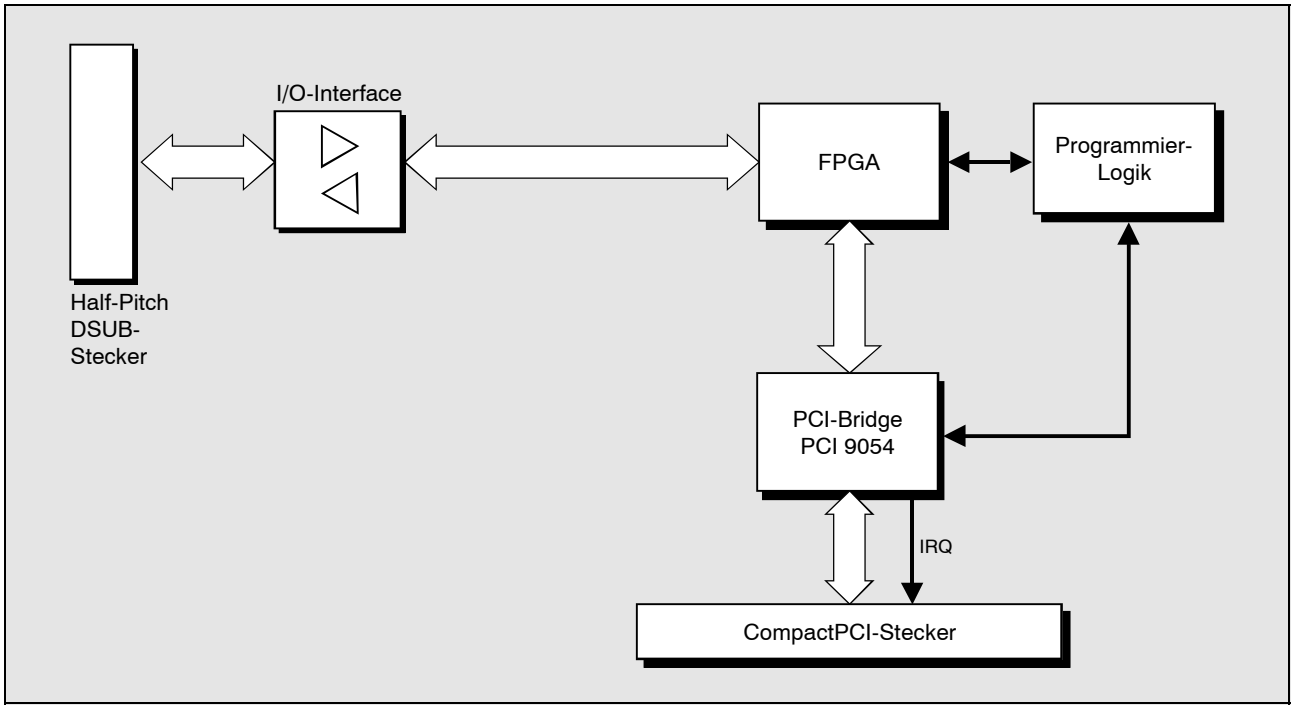



**Abb. 1.1:** Blockschaltbild des CPCI-DIO72-Moduls

Die CPCI-DIO72-Karte ist eine universell einsetzbare digitale I/O-Karte für den Betrieb im CompactPCI- System. Sie setzt sich im wesentlichen aus den Komponenten Eingangsschaltung, einem FPGA als I/O-Controller und einer PCI-Bridge zusammen.

Die Anbindung an den CompactPCI-Bus wird durch die PCI-Bridge PCI9054 von PLX Technology realisiert, die Busmasterfähigkeit besitzt und damit in der Lage ist, selbständig Initiator eines PCI-Zyklus zu sein.
Als programmierbarer Logikbaustein wird ein FPGA der Spartan Familie von der Firma Xilinx eingesetzt, mit dem sich sehr komplexe Funktionen mit hoher Logiktiefe realisieren lassen.

## 1.2 Platinenansicht mit Steckerbezeichnung

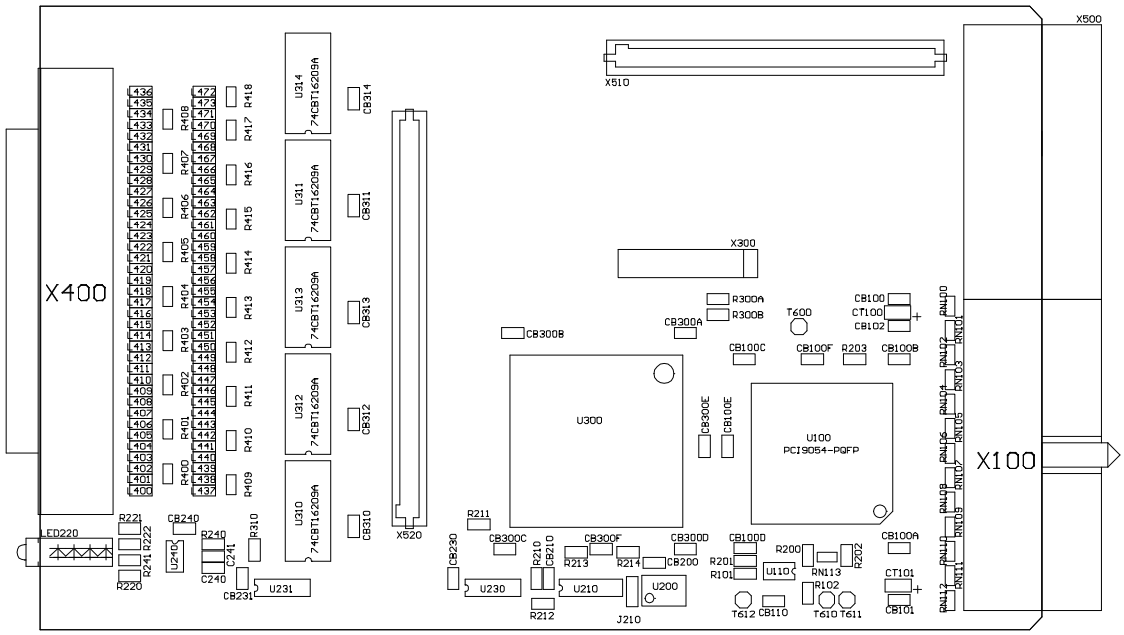



**Abb. 1.2:** Ansicht des Moduls (Darstellung ohne Frontplatte)

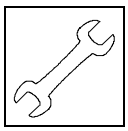
## 1.3 Schaltung der Ein- und Ausgänge

Für die flexible Richtungsschaltung der 72 Kanäle werden auf der CPCI-DIO72-Karte Bus-Exchange-Switches eingesetzt. Diese Bausteine enthalten FET Pass-Transistoren und besitzen somit keine Treiberfähigkeit und keine vorgegebene Signalrichtung. Eine Umschaltung der Signalrichtung kann daher entfallen.

Die Eingangsschaltung bietet zudem eine Schutzfunktion gegen positive oder negative Überspannung durch interne Clamp-Dioden.
Zudem sind alle Signale der 80-poligen Buchse auf der Frontplatte der CPCI-DIO72-Karte in hochohmigem Zustand, solange das CompactPCI-System nicht eingeschaltet ist und das FPGA noch nicht programmiert ist. Dieser hochohmige Zustand kann auch von der Software ausgelöst werden.

In jede I/O-Leitung ist kurz vor der Buchse eine Reihenschaltung aus einer Spule und einem Widerstand geschaltet. Die Spule dient dabei dem Unterdrücken von hochfrequenten Störungen und der Widerstand dämpft die Schwingneigung der Eingangsleitung.

Diese Seite ist bewußt unbedruckt.



# 2. Hardware-Installation

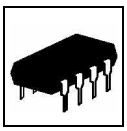
**Achtung !**
Elektrostatische Entladungen können Schäden an elektronischen Bauteilen verursachen. Um dies zu verhindern, führen Sie bitte *vor* dem Berühren des Moduls die folgenden Schritte aus, um die statische Elektrizität Ihres Körpers zu entladen:

- ➽ Schalten Sie die Versorgungsspannung Ihres Rechners aus, aber lassen Sie vorerst den Netzstecker noch in der Steckdose.
- ➽ Jetzt berühren Sie bitte das Metallgehäuse des Rechners um sich zu entladen.
- ➽ Im Weiteren sollten Sie es außerdem vermeiden, das Modul mit Ihrer Kleidung zu berühren, da diese ebenfalls elektrostatisch aufgeladen sein kann.

**Vorgehensweise zur Installation:**

1. Schalten Sie Ihren Rechner und alle angeschlossenen Peripheriegeräte (Monitor, Drucker etc.) aus.
2. Führen Sie die Entladung der elektrostatischen Elektrizität Ihres Körpers wie oben beschrieben aus.
3. Ziehen Sie das Netzkabel des Rechners aus der Steckdose.
4. Stecken Sie das CPCI-DIO72-Modul in einen freien CompactPCI-Bus-Steckplatz.
5. Fixieren Sie das CPCI-DIO72-Modul mit der hierfür vorgesehenen Schraube der Frontplatte.
6. Schließen Sie die I/O-Signale an den Half-Pitch DSUB-Stecker X400 in der Frontplatte an.
7. Schließen Sie die Spannungsversorgung des Rechners wieder an.
8. Schalten Sie den Rechner und alle anderen Geräte wieder an.
9. Ende der Hardware-Installation.
10. Fahren Sie mit der Software-Installation fort (sofern nicht bereits erfolgt).

Diese Seite ist bewußt unbedruckt.



# 3. Zusammenfassung der technischen Daten

## 3.1 Allgemeine technische Daten

| | |
|---|---|
| Umgebungstemperatur | 0...50°C,<br>auch für -40 ˚C...+85 ˚C erhältlich |
| Luftfeuchtigkeit | max. 90 %, nicht kondensierend |
| Versorgungsspannung | über CompactPCI-Bus,<br>Nennspannungen: 5 V ±5%<br>3,3 V ±5% |
| Steckverbinder | X100 (132-pol. Pfostenstecker) - CompactPCI-Board-Connector<br>X400 (80-pol. DSUB Half-Pitch) - Abgewinkelte Mini DSUB-Buchse von AMP |
| Abmessungen | 100 mm x 160 mm |
| Gewicht | 160 g |

**Tabelle 3.1:** Allgemeine Daten des Moduls

## 3.2 CompactPCI Bus

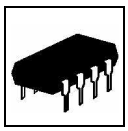
| | |
|---|---|
| Host-Bus | PCI-Bus gemäß PCI Local Bus Specification 2.1 |
| PCI-Daten/Adressbus | 32 Bit |
| Controller | PCI9054 von PLX Technologie |
| Interrupt | Interrupt-Signal A |
| Board-Dimension | gemäß CompactPCI-Specification, Rev. 1.0 |
| Steckverbinder | |

**Tabelle 3.2:** CompactPCI-Bus-Daten

## 3.3 Digitale Ein- und Ausgänge

| | |
|---|---|
| Anzahl der digitalen Ein- und Ausgänge | 72 unabhängige Ein- oder Ausgangskanäle |
| Konfiguration | unterteilt in: 2 x 32 + 8 oder<br>1 x 64 + 8 Ein- und Ausgänge |
| Eingangsspannung | geeignet für 3,3 V und 5 V Signalspannungen |
| Ausgangsspannung | $U_{AUS}$ = 2,4 V ($I_{AUS}$ = 7 mA) |
| Schutzschaltung | interne Clamp-Dioden in Treiberbaustein |
| Steckverbinder auf der Platine | X400 (80-pol. DSUB Half-Pitch) - Abgewinkelte Mini DSUB-Buchse von AMP |
| Interrupts | IRQ bei steigender oder fallender Flanke eines Eingangs |
| DMA-Zugriff | Lesen der Eingänge |

**Tabelle 3.3:** Digitale Ein- und Ausgänge des CPCI-DIO72-Moduls



## 3.4 Software-Unterstützung

Zur Ansteuerung der Karte sind Treiber als Object-Code lieferbar. Sie bieten Funktionen zur Konfiguration und zum Lesen der Ein-/Ausgänge. Außerdem werden DMA-Zugriffe auf die Eingänge unterstützt.
Die API wird im zweiten Teil des Handbuches beschrieben.

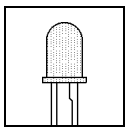
## 3.5 Bestellhinweise

| Typ | Eigenschaften | Bestell-Nr. |
|---|---|---|
| CPCI-DIO72 | 72 digitale Ein- und Ausgänge<br>0...+50 ˚C | I.2303.02 |
| CPCI-DIO72-T | erweiterter Temperaturbereich<br>-40...+85 ˚C | I.2303.03 |
| CPCI-DIO72-RTOS-UH | RTOS-UH-Obj.-Lizenz | I.2303.54 |
| CPCI-DIO72-VxW | VxWorks-Obj.-Lizenz | I.2303.55 |
| CPCI-DIO72-MD *) | Anwenderhandbuch in deutsch | I.2303.20 |

*) Wird das Handbuch gemeinsam mit dem Produkt bestellt, so wird es kostenlos mitgeliefert.

**Tabelle 3.5:** Bestellhinweise

Diese Seite ist bewußt unbedruckt.



# 4. LED-Anzeige

Das Modul ist mit vier LEDs in der Frontplatte versehen.

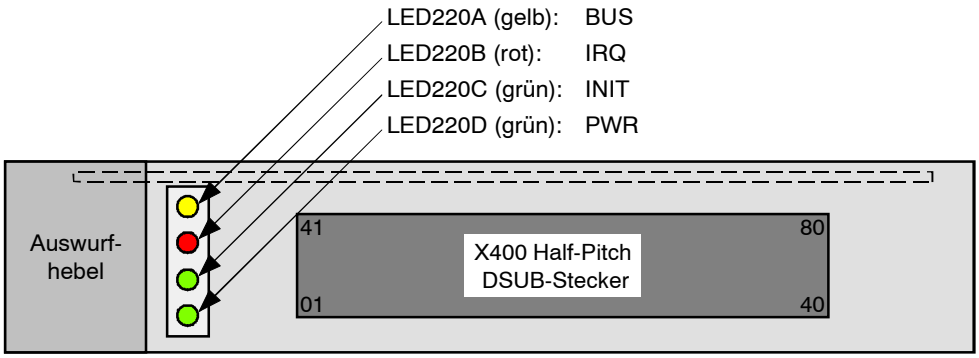
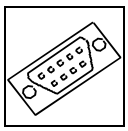



**Abb. 4.1:** Position und Farben der LEDs

| LED | Farbe | Name | Anzeigefunktion bei leuchtender LED |
|---|---|---|---|
| LED220A | gelb | BUS | Host-CPU greift über PCI-Bus auf die Karte zu |
| LED220B | rot | IRQ | Die Karte löst einen PCI-Bus-Interrupt aus |
| LED220C | grün | INIT | FPGA ist programmiert, Karte ist funktionsfähig |
| LED220D | grün | PWR | Power - Versorgungsspannung liegt an |

**Tabelle 4.1:** Anzeigefunktionen der LEDs

Diese Seite ist bewußt unbedruckt.



# 5. Steckerbelegung

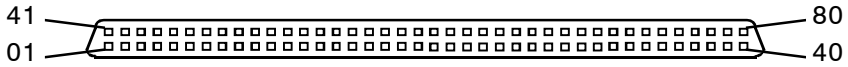
## 5.1 Belegung des I/O-Steckers X400

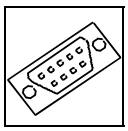
### 5.1.1 Bauform

- auf Platine: 80-poliger Half-Pitch DSUB-Stecker mit Stiftkontakten,
  AMP Best.Nr.: 749830-8 oder 787190-8

- Gegenstecker an Leitung: 80-poliger Half-Pitch DSUB-Stecker mit Buchsenkontakten,
  AMP Best.Nr.: 749621-8 oder 749111-7

### 5.1.2 Pin-Belegung



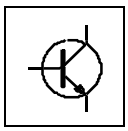


**Steckerbelegung**

### 5.1.3 Signale

Neben den 72 I/O-Kanälen, einem externen Synchronisationssignal (LATCH) und einem Ausgangsfreigabesignal (OUTEN) für die Kanäle 64-71 wird dort auch das Massepotential der I/O-Karte bereitgestellt.

| Signal | Pin | | Signal |
|---|---|---|---|
| GND | 1 | 41 | I/O 37 |
| I/O 00 | 2 | 42 | I/O 38 |
| I/O 01 | 3 | 43 | I/O 39 |
| I/O 02 | 4 | 44 | I/O 40 |
| I/O 03 | 5 | 45 | I/O 41 |
| I/O 04 | 6 | 46 | I/O 42 |
| I/O 05 | 7 | 47 | I/O 43 |
| I/O 06 | 8 | 48 | I/O 44 |
| I/O 07 | 9 | 49 | I/O 45 |
| I/O 08 | 10 | 50 | I/O 46 |
| I/O 09 | 11 | 51 | I/O 47 |
| I/O 10 | 12 | 52 | GND |
| I/O 11 | 13 | 53 | I/O 48 |
| I/O 12 | 14 | 54 | I/O 49 |
| I/O 13 | 15 | 55 | I/O 50 |
| I/O 14 | 16 | 56 | I/O 51 |
| I/O 15 | 17 | 57 | I/O 52 |
| GND | 18 | 58 | I/O 53 |
| I/O 16 | 19 | 59 | I/O 54 |
| I/O 17 | 20 | 60 | I/O 55 |
| I/O 18 | 21 | 61 | I/O 56 |
| I/O 19 | 22 | 62 | I/O 57 |
| I/O 20 | 23 | 63 | I/O 58 |
| I/O 21 | 24 | 64 | I/O 59 |
| I/O 22 | 25 | 65 | I/O 60 |
| I/O 23 | 26 | 66 | I/O 61 |
| I/O 24 | 27 | 67 | I/O 62 |
| I/O 25 | 28 | 68 | I/O 63 |
| I/O 26 | 29 | 69 | GND |
| I/O 27 | 30 | 70 | LATCH |
| I/O 28 | 31 | 71 | I/O 64 |
| I/O 29 | 32 | 72 | I/O 65 |
| I/O 30 | 33 | 73 | I/O 66 |
| I/O 31 | 34 | 74 | I/O 67 |
| GND | 35 | 75 | I/O 68 |
| I/O 32 | 36 | 76 | I/O 69 |
| I/O 33 | 37 | 77 | I/O 70 |
| I/O 34 | 38 | 78 | I/O 71 |
| I/O 35 | 39 | 79 | OUTEN |
| I/O 36 | 40 | 80 | GND |



# 6. Stromlaufpläne

Die Stromlaufpläne sind in der PDF-Datei dieses Dokumentes nicht enthalten. Sie werden auf Anfrage ausgeliefert.

# CPCI-DIO72-API

## Software-Handbuch



esd electronic system design gmbh
Vahrenwalder Str. 207 • 30165 Hannover • Germany
www.esd-electronics.com • Fax: 0511/37 29 8-68
Phone: 0511/37 29 80 • International: +49-5 11-37 29 80

Der Inhalt dieses Handbuches wurde mit größter Sorgfalt erarbeitet und geprüft. **esd** übernimmt jedoch keine Verantwortung für Schäden, die aus Fehlern in der Dokumentation resultieren könnten. Insbesondere Beschreibungen und technische Daten sind keine zugesicherten Eigenschaften im rechtlichen Sinne.

**esd** hat das Recht, Änderungen am beschriebenen Produkt oder an der Dokumentation ohne vorherige Ankündigung vorzunehmen, wenn sie aus Gründen der Zuverlässigkeit oder Qualitätssicherung vorgenommen werden oder dem technischen Fortschritt dienen.



**esd electronic system design gmbh**
Vahrenwalder Str. 207
30165 Hannover

Tel.: 0511/372 98-0
FAX : 0511/372 98-68
E-Mail: info@esd-electronics.com
Internet: www.esd-electronics.com

| | |
|---|---|
| Handbuch-Datei: | I:\texte\Doku\MANUALS\CPCI\DIO72\CPI7215S.ma9 |
| Handbuch-Bestellnummer: | I.2303.20 |
| Datum der Druck-vorlagenerstellung: | 12.12.2005 |

| Software-Treiber | Bestellnummer | Beschriebene Treiber-Version |
|---|---|---|
| VxWorks | I.2303.55 | keine Angabe |
| RTOS-UH | I.2303.54 | keine Angabe |
| Linux | I.2303.19 | V.1.2.0 |

**Änderungen in der Software und/oder der Dokumentation**

| Kapitel | Änderung in diesem Handbuch gegenüber der Vorversion |
|---|---|
| 3.3 | Konfiguration der Kanalrichtung geändert. |

Diese Seite ist bewußt unbedruckt.

## Inhalt

Seite

Diese Seite ist bewußt unbedruckt.

# 1. Einleitung

Dieses Handbuch beschreibt die Treiber-Software des CompactPCI-Moduls CPCI-DIO72.

Im ersten Abschnitt werden die Funktionen beschrieben. Im zweiten Abschnitt werden Beispiele für die Anwendungsprogrammierung beschrieben.

## 1.1 Hinweise zur Linux-Treiber-Software

Im Lieferumfang der Linux-Treiber-Software ist eine readme-Datei mit Hinweisen zur Treiber-Installation enthalten.

In der ebenfalls mitgelieferten Datei `io72test.c` finden sich Beispiele für die API-Aufrufe.

## 1.2 Die Programmierschnittstelle des Treibers (API)

Im folgenden werden die einzelnen Aufrufe der Programmierschnittstelle beschrieben. Die Prototypen der Aufrufe sowie die nötigen Konstanten sind in der Headerdatei 'io72api.h' enthalten. Sie muß bei Verwendung der CPCI-DIO72-Karte stets in den Quellcode eingebunden sein. Bei Schreibe- und Lesezugriffen auf Register in der CPCI-DIO72-Karte wird grundsätzlich 32 Bit breit zugegriffen.
Alle Funktionen der API führen eine Konvertierung von Little Endianess des PCI Busses und der Endianess der CPU durch.

Es gilt die folgende Zuordnung zwischen Registerbits und den Kanälen der CPCI-DIO72-Karte:

| Bitposition | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | MSB | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | LSB |
| Kanäle 0-31 | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| Kanäle 32-63 | 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| Kanäle 64-71 | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | 71 | 70 | 69 | 68 | 67 | 65 | 64 | 63 |

**Tabelle 1.2:** Bitzuordnung

# 2. Starten und Beenden des Treibers

Zum **Starten** des Treibers ist

```
io72Start
```

einzugeben.

Zum **Stoppen** des Treibers ist

```
io72Stop
```

einzugeben.

# 3. Funktionsbeschreibungen

## 3.1 Initialisierung

### io72Open()

**Name:** **io72Open()** - Erzeugen eines Handles für Lese- und Schreiboperationen

**Aufruf:**

```
int     io72Open
        (
        int         card,       /* CPCI-Karten-Nummer */
        HANDLE      *hnd        /* Output:  Handle */
        )
```

**Beschreibung:** Diese Funktion gibt ein Handle für nachfolgende I/O-Aufrufe zurück. Es können maximal 1024 Handles geöffnet werden.
Durch den Parameter *card* wird dieser Funktion ein Index der gewünschten I/O-Karte übergeben. Bei Erfolg wird in der Variablen, auf die *hnd* zeigt, ein Handle der CPCI-DIO72-Karte zurückgegeben. Ein Nutzungszähler in der Treiberdatenbank wird erhöht.

> *io72Open* ist vor dem ersten Zugriff auf eine CPCI-DIO72-Karte aufzurufen. Das zurückgegebene *Handle* wird als Referenz für jeden weiteren Zugriff auf die CPCI-DIO72-Karte benötigt.

**Parameter:**

| | |
|---|---|
| *card:* | Nummer der CPCI-DIO72-Karte |
| **hnd*: | Nach erfolgreichem Funktionsaufruf wird ein Handle zurückgegeben. |

**Rückgabe:** '0' im Falle eines korrekten Ablaufs oder einen Fehlercode (siehe Tabelle 3.7 auf Seite 24).

**Header:** io72api.h

# io72Close()

**Name:** **io72Close()** - Schließen eines Handles für Lese- und Schreibzugriffe

**Aufruf:**

```
int    io72Close
       (
       HANDLE    hnd          /* Eingabe: Handle */
       )
```

**Beschreibung:** Dieser Funktion wird mit *hnd* das Handle einer freizugebenden CPCI-DIO72-Karte übergeben. Der Nutzungszähler der CPCI-DIO72-Karte wird dekrementiert. Das Handle wird mit allen belegten Ressourcen freigegeben.

**Eingabe:** *hnd:* Handle, das von *io72Open* zurückgegeben wurde.

**Rückgabe:** '0' im Falle eines korrekten Ablaufs oder einen Fehlercode (siehe Tabelle 3.7 auf Seite 24).

**Header:** io72api.h

# io72Reset()

**Name:** **io72Reset()** - CPCI-DIO72-Karte in Grundzustand setzen

**Aufruf:**
```
int    io72Reset
       (
       HANDLE    hnd              /* Eingabe: Handle */
       )
```

**Beschreibung:** Dieser Funktion wird mit *hnd* das Handle der CPCI-DIO72-Karte übergeben. Alle Register der CPCI-DIO72-Karte werden in einen Default-Zustand gebracht. Interrupts werden gesperrt. Der Timer wird gestoppt. Alle Kanäle werden als Eingänge konfiguriert.

**Eingabe:** *hnd:* Handle, das von *io72Open* zurückgegeben wurde.

**Rückgabe:** '0' im Falle eines korrekten Ablaufs oder einen Fehlercode (siehe Tabelle 3.7 auf Seite 24).

**Header:** io72api.h

## 3.2 Konfiguration

# io72Config()

**Name:** **io72Config ()** - Konfiguration der Kartenfunktion

**Aufruf:**

```
int    io72Config
       (
       HANDLE    hnd,        /* Eingabe: Handle */
       int       mode,       /* Parameter-Typ */
       int       para        /* Parameter-Wert */
       )
```

**Beschreibung:** Diese Funktion dient zur Freigabe des Interrupts und zur Einstellung der Abtastrate. *mode* gibt die zu konfigurierende Funktion an. *para* ist ein Konfigurationsparameter und ist zusammen mit *mode* der folgenden Tabelle zu entnehmen.

**Eingabe:**

*hnd:* Handle, das von *io72Open* zurückgegeben wurde.

*para*: Der Parameter *para* wird in Abhängigkeit vom Parameter *mode* ausgewertet.

*mode:* *mode* bestimmt, ob in *para* der Wert für die Abtastrate der digitalen Eingänge oder die Freigabe des Interrupts für das Interrupt-gesteuerte Lesen übergeben wird.
Die folgende Tabelle zeigt die zulässigen Werte von *mode* und *para*:

| **Parameter** | | **Bedeutung** |
|---|---|---|
| *mode* | *para* | |
| CONFIG_INTERRUPT | INTERRUPT_ENABLE | Interrupt freigegeben |
| | INTERRUPT_DISABLE | Interrupt gesperrt |
| CONFIG_SET_FREQUENCY | Frequenz f in Hz | Frequenz des Timers setzen.:<br>0 Hz keine Abtastung<br>1...100 000 Hz Abtastrate (1 Hz ...1 MHz) |
| CONFG_SET_TIMERVAL | Timerwert N | Frequenz des Timers einstellen durch direktes Schreiben des Timerregisters. Die Timerfrequenz f berechnet sich in der Betriebsart *slow* zu:<br>$f = \frac{N}{2^{(20+4)}} \cdot 20 \cdot 10^6 Hz$ |

| Parameter | | Bedeutung |
|---|---|---|
| *mode* | *para* | |
| CONFIG_SET_INTSOURCE<br><br>(Festlegen der Interruptquelle) | INSOURCE_TIMER | Festlegen des Timers; mehrere Quellen gleichzeitig sind möglich. Die Parameter müssen dafür 'verodert' werden. |
| | INTSOURCE_EXT | Externer Eingang |
| | INTSOURCE_EDGE | Erkannte Flanke |
| | INTSOURCE_NONE | Keine Interruptquelle freigeben, keine Interrupts zulassen |
| CONFIG_SET_DMASOURCE<br><br>(Festlegen der DMA-Triggerquelle) | DMASOURCE_TIMER | Timer; Mehrere Quellen gleichzeitig sind möglich. Die Parameter müssen dafür 'verodert' werden. Die maximale Timerfrequenz beträgt 2,5 MHz. |
| | DMASOURCE_EXT | Externer Ausgang |
| | DMASOURCE_EDGE | Erkannte Flanke |
| | DMASOURCE_NONE | Keine DMA-Triggerquelle, keinen DMA zulassen |
| CONFIG_SET_CAPSOURCE (Auffangbedingung für das Einfrieren der Eingangsdaten festlegen.) | CAPSOURCE_TIMER | Timerüberlauf; Mehrere Quellen gleichzeitig sind möglich. Die Parameter müssen dafür 'verodert' werden. |
| | CAPSOURCE_EXT | Flanke an externen Synchronisationseingang |
| | CAPSOURCE_EDGE | Erkannte Flanke |
| | CAPSOURCE_NONE | Kein Einfrieren der Eingänge |
| CONFIG_TIMER | TIMER_START | Timer starten |
| | TIMER_STOP | Timer stoppen |
| | TIMER_FAST | Timerfrequenz erhöhen; die Timerfrequenz wird um den Faktor 16 erhöht |
| | TIMER_SLOW | Normale Timerfunktion; die Frequenz stimmt überein mit dem von CONFIG_SET_FREQUENCY gesetzten Wert |
| CONFIG_EXT<br><br>(Konfiguration des externen Triggeranschlusses.) | EXT_ACTIVELOW | Eingang reagiert auf eine fallende Flanke. |
| | EXT_ACTIVEHIGH | Eingang reagiert auf eine steigende Flanke. |
| | EXT_INPUT | Externer Triggeranschluß ist ein Eingang |
| | EXT_OUTPUT | Externer Triggeranschluß ist ein Ausgang. Die mit CONFIG_SET_CAPSOURCE konfigurierte Capturebedingung ist nun am externen Triggeranschluß abgreifbar. |

**Tabelle 3.2:** Werte der Konfigurationsparameter

**Rückgabe:** '0' im Falle eines korrekten Ablaufs oder einen Fehlercode (siehe Seite 24).

**Header:** io72api.h

## 3.3 Lesen der digitalen Eingänge

### io72Write()

**Name:** **io72Write()** - Schreiben auf Ausgänge oder Register der CPCI-DIO72-Karte

**Aufruf:**

```
int     io72Write
        (
        HANDLE      hnd,        /* Handle */
        int         mode,       /* zu lesende Eingänge */
        void        *buffer     /* Pointer auf Daten-Buffer */
        )
```

**Beschreibung:** Diese Funktion dient dem Schreiben von Daten auf die Ausgänge sowie interne Register der CPCI-DIO72-Karte.

**Eingabe:**

*hnd:* Handle, das von *io72Open* zurückgegeben wurde.

*mode:* Auswahl des Schreibzugriffes. Konfiguriert die Datenrichtung der 72 I/O-Kanäle und die Flankenerkennung. Zulässige Werte für *mode* sind in der folgenden Tabelle dargestellt.

| Parameter *mode* | Gültigkeit | Bedeutung |
|---|---|---|
| CHANNEL_0_31 | RW,INT, DMA | Zugriff auf die Ein-und Ausgänge: 0..31, Die gesetzten Bits in den Daten repräsentieren einen High-Pegel am entsprechenden Ein- oder Ausgang |
| CHANNEL_32_63 | RW,INT, DMA | Zugriff auf die Ein- und Ausgänge: 32...63 |
| CHANNEL_0_63 | RW,INT, DMA | Zugriff auf die Ein- und Ausgänge: 0...63 |
| CHANNEL_64_71 | RW | Zugriff auf die Ein- und Ausgänge: 64...71 Gültige Daten befinden sich in den unteren 8 Bit. |
| DIRECTION_0_31 | RW | Konfiguration der Kanalrichtung für Kanäle: 0...31 Gesetztes Bit '1' konfiguriert Kanal als Ausgang, gelöschtes Bit '0' konfiguriert Kanal als Eingang, Lesen liefert den zuletzt geschriebenen Wert. |
| DIRECTION_32_63 | RW | Konfiguration der Kanalrichtung für Kanäle: 32...63 |
| DIRECTION_0_63 | RW | Konfiguration der Kanalrichtung für Kanäle; 0...63 |
| DIRECTION_64_71 | RW | Konfiguration der Kanalrichtung für Kanäle: 64...71 |
| EDGE_0_31 | RW,INT | Beim Lesen zeigen gesetzte Bits eine erkannte Flanke an den Eingängen 0...31 an. '1' setzt das Flankenerkennungsbit für diesen Kanal zurück, '0' hat keine Auswirkung |

| | | |
|---|---|---|
| EDGE_32_63 | RW,INT | Erkannte Flanken an den Eingängen 32...63 |
| **Parameter *mode*** | **Gültigkeit** | **Bedeutung** |
| EDGE_0_63 | RW,INT | Erkannte Flanken an den Eingängen 0...63 |
| LATCH_0_31 | R | Beim Auftreten der Auffangbedingung wird ein Abbild der Eingänge in das 64 Bit breite, interne Auffangregister geschrieben.<br>Auffangregister der Eingänge lesen: 0...31 |
| LATCH_32_63 | R | Auffangregister der Eingänge lesen: 32...63 |
| LATCH_0_63 | R | Auffangregister der Eingänge lesen: 0...63 |
| EDGE_ENABLES_0_31 | RW | Flankenerkennung freigeben.<br>Gesetzte Bits in diesen Registern geben die Flankenerkennung für die Eingänge 0...31 frei. |
| EDGE_ENABLES_32_63 | RW | Gesetzte Bits in diesen Registern geben die Flankenerkennung für die Eingänge 32...63 frei. |
| EDGE_ENABLES_0_63 | RW | Gesetzte Bits in diesen Registern geben die Flankenerkennung für die Eingänge 0...63 frei. |
| EDGE_IENABLES_0_31 | RW | Bei gesetzten Bits in diesem Register wird ein Interrupt bei Auftreten der entsprechenden Flanke ausgelöst. Die Flankenerkennung muß zusätzlich freigegeben werden.<br>Interrupts bei auftretenden Flanken an den Eingängen 0...31 freigeben |
| EDGE_IENABLES_32_63 | RW | Interrupts bei auftretenden Flanken an den Eingängen 32...63 freigeben |
| EDGE_IENABLES_0_63 | RW | Interrupts bei auftretenden Flanken an den Eingängen 0...63 freigeben |
| EDGE_SELECT_0_31 | RW | Bei gesetzten Bits in diesem Register werden steigende Flanken erkannt. Gelöschte Bits konfigurieren die Erkennung von fallenden Flanken. Dieses Register wählt die Art der Flankenerkennung für die Eingänge 0...31 aus |
| EDGE_SELECT_32_63 | RW | Dieses Register wählt die Art der Flanken-erkennung für die Eingänge 32...63 aus |
| EDGE_SELECT_0_63 | RW | Dieses Register wählt die Art der Flanken-Erkennung für die Eingänge 0...63 aus |

**Tabelle 3.3:** Werte des Konfigurationsparameters *mode*

Zulässige Operationen: R... nur Lesen, RW... Lesen und Schreiben,
INT... Interruptgestütztes Lesen, DMA... Lesen per DMA

**buffer:* Pointer auf Daten-Buffer

**Rückgabe:** ‘0’ im Falle eines korrekten Ablaufs oder einen Fehlercode (siehe Tabelle 3.7 auf Seite 24).

**Header:** io72api.h

# io72MaskWrite()

**Name:** **io72MaskWrite()** - Maskiertes Schreiben auf Ausgänge oder Register der CPCI-DIO72-Karte.

**Aufruf:**

```
int   io72MaskWrite
      (
      HANDLE   hnd,        /* Handle */
      int      mode,       /* zu lesende Eingänge */
      void     *buffer,    /* Pointer auf Daten-Buffer */
      void     *maskbuffer /* Pointer auf Maskierungsdaten-Buffer */
      )
```

**Beschreibung:** Diese Funktion dient dem Schreiben von Daten auf Ausgänge sowie interne Register der CPCI-DIO72-Karte. Bei den Schreibzugriffen werden nur die Bits im Ziel modifiziert, die in den Maskierungsdaten gesetzt sind.

**Eingabe:**

| | |
|---|---|
| *hnd:* | Handle, das von *io72Open* zurückgegeben wurde. |
| *mode:* | Wählt das Ziel des Schreibzugriffes aus<br>Zulässige Werte siehe Tabelle 3.3 |
| **buffer:* | Pointer auf Daten-Buffer |
| **maskbuffer:* | Pointer auf Markierungsdaten<br>Bei Zugriff auf Registerpaare (64 Bit) muß *maskbuffer* ebenfalls auf ein Array von zwei Langworten zeigen. |

**Rückgabe:** ‘0’ im Falle eines korrekten Ablaufs oder einen Fehlercode (siehe Tabelle 3.7 auf Seite 24).

**Header:** io72api.h

# io72Read()

**Name:** **io72Read()** - Lesen von Eingängen oder Registern der CPCI-DIO72-Karte.

**Aufruf:**

```
int     io72Read
        (
        HANDLE      hnd,        /* Handle */
        int         mode,       /* zu lesende Eingänge */
        void        *buffer     /* Pointer auf Daten-Buffer */
        )
```

**Beschreibung:** Mit *io72Read* wird der Zustand der digitalen Eingänge transparent gelesen.

**Eingabe:**

*hnd:* Handle, das von *io72Open* zurückgegeben wurde.

*mode:* Auswahl der zu lesenden digitalen Eingänge: siehe Tabelle 3.3
Durch *mode* kann mit einem Aufruf auf ein Registerpaar zugegriffen werden.

**buffer:* Pointer auf Daten-Buffer
Bei einem Registerpaar werden auf die Stelle, auf die *buffer* zeigt zwei Langworte (64 Bit) geschrieben.

**Rückgabe:** '0' im Falle eines korrekten Ablaufs oder einen Fehlercode (siehe Tabelle 3.7 auf Seite 24).

**Header:** io72api.h

## 3.4 Interrupt Behandlung

# Io72IntRead()

**Name:** **Io72IntRead()** - Interrupt-gesteuertes Lesen von Eingängen oder Registern der CPCI-DIO72-Karte.

**Aufruf:**

```
int    Io72IntRead
       (
       HANDLE    hnd,        /* Handle */
       int       mode,       /* zu lesende Eingänge */
       void      *buffer     /* Pointer auf Daten-Buffer */
       )
```

**Beschreibung:** Mit *Io72IntRead* werden die digitalen Eingänge Interrupt-gesteuert gelesen.
Vor dem Aufruf muß eine Interruptquelle durch die Funktion *io72Config* ausgewählt und freigeschaltet werden. DEr Aufruf von io*72IntRead()* blockiert dann, bis die Interruptbedingung auftritt und gibt die Eingänge oder Register, die durch **mode** ausgewählt werden zurück
Die Beispieldatei `io72test.c` enthält ein Beispiel für die Nutzung dieser Funktion.

**Eingabe:**

| | |
|---|---|
| *hnd:* | Handle, das von *io72Open* zurückgegeben wurde. |
| *mode:* | Auswahl der zu lesenden digitalen Eingänge<br>Siehe Tabelle 3.3 |
| **buffer:* | Pointer auf Daten-Buffer |

**Rückgabe:** '0' im Falle eines korrekten Ablaufs oder einen Fehlercode (siehe Tabelle 3.7 auf Seite 24).
Ist zwischen der Freigabe von Interrupts oder zwischen zwei *Io72IntRead*-Aufrufen mehr als ein Interrupt aufgetreten, gibt *Io72IntRead* den Wert: IO72_ERR_LOST_IRQ zurück. Das Lesen der Daten wird dabei nicht beeinflußt.

**Header:** io72api.h

## 3.5 DMA-Behandlung

Die folgenden Funktionen lesen die Daten per Direct Memory Access (DMA). So können hohe Transferraten erreicht werden, ohne das die CPU dabei stark belastet wird. Für das Lesen größerer Datenmengen sollten daher die DMA-Funktionen gewählt werden, wenn die Anwendung dies zuläßt.

### io72DmaRead()

**Name:** **io72DmaRead()** - Lesen von Eingängen der CPCI-DIO72-Karte per DMA.

**Aufruf:**

```
int     io72DmaRead
        (
        HANDLE  hnd,        /* Handle */
        int     mode,       /* zu lesende Kanäle */
        void    *buffer,    /* Pointer auf Daten-Buffer */
        int     size        /* Anzahl der zu lesenden Bytes */
        )
```

**Beschreibung:** Mit *io72DmaRead* werden die Eingänge per DMA-Zugriff gelesen. Im Gegensatz zu *Io72IntRead* wird nicht nach jeder Abtastung ein Interrupt ausgelöst. Die Daten der digitalen Eingänge werden direkt in den Arbeitsspeicher der CPU geschrieben. Die gewünschte Größe des Buffers wird über den Parameter *size* angegeben. Nachdem das letzte Byte in den Buffer geschrieben ist, wird ein Interrupt ausgelöst.
Jeder einzelne DMA-Lesezugriff (32 oder 64 Bit) wird durch die mit *io72Config* konfigurierte DMA-Triggerquelle ausgelöst. Die Funktion kehrt erst zurück, wenn die in *size* angegebene Anzahl von Bytes gelesen worden ist.

**Eingabe:**

| | |
|---|---|
| *hnd:* | Handle, das von *io72Open* zurückgegeben wurde. |
| *mode:* | Auswahl der zu lesenden digitalen Eingänge:<br>Siehe Tabelle 3.3 |
| **buffer:* | Pointer auf Daten-Buffer |
| *size:* | Anzahl der zu lesenden Bytes (Buffer muß mindestens die Größe haben, die in *size* angegeben ist) |

**Rückgabe:** '0' im Falle eines korrekten Ablaufs oder einen Fehlercode (siehe Tabelle 3.7 auf Seite 24).
Können die einzelnen Lesezugriffe nicht in der geforderten Geschwindigkeit bearbeitet werden, so wird IO72_ERR_LOST_DATA nach Beendigung des kompletten Transfers zurückgegeben.

**Header:** io72api.h

# io72MultiDmaStart()

**Name:** **io72MultiDmaStart()** - Initialisierung eines kontinuierlichen DMA-Transfers.

**Aufruf:**

```
int    io72MultiDmaStart
       (
       HANDLE      hnd,
       int         mode,
       void        **bufferlist,
       int         bufsize,
       int         bufcount
       )
```

**Beschreibung:** Diese Funktion dient zusammen mit *io72MultiDmaRead* und *io72MultiDmaStop* dem kontinuierlichem Lesen der Eingänge per DMA. *Io72MultiDmaStart* initialisiert diesen Vorgang. *Io72MultiDmaStart* konfiguriert den DMA Transfer so, daß diese Speicherbereiche nacheinander mit den Lesedaten gefüllt werden. Sind alle Speicherbereiche gefüllt, so wird beim ersten Speicherbereich fortgesetzt (Ringbuffer aus Speicherbereichen).

Das folgende Codefragment zeigt einen beispielhaften Aufruf von *io72MultiDmaRead:*

```
void *buffer;
void* bufferlist[BUFFER_COUNT];
int ret,i;

...

/* Aufbau des Arrays von Zeigern auf Speicherbereiche */
for (i=0 ; i<BUFFER_COUNT; i++)
  {
  /* Speicherbereich anfordern ... */
    buffer = malloc(BUFFER_SIZE);
    /* ... und dem Array hinzufuegen */
    bufferlist[i] = buffer;
  }

/* weitere Konfiguration ... (z.B. Timer) */

...

/* starten */
ret =
io72MultiDmaStart(hnd,...,bufferlist,BUFFER_SIZE,BUFFER_COUNT);

...
```

Der Vorteil der *io72MultiDma*-Funktionen gegenüber der Funktion *io72DmaRead* ist, daß alle Eingangsdaten nach dem Aufruf von *io72MultiDmaStart* bis zum Aufruf von *io72MultiDmaStop* gespeichert werden.
(siehe auch *io72MultiDmaRead* und *io72MultiDmaStop*)

**Eingabe:**

| | |
|---|---|
| *hnd:* | Handle, das von *io72Open* zurückgegeben wurde. |
| *mode:* | Auswahl der zu lesenden digitalen Eingänge:<br>Siehe Tabelle 3.3 |
| ***bufferlist*: | ist ein Zeiger auf ein Array von Zeigern auf Speicherbereiche, in die die gelesenen Daten geschrieben werden sollen. |
| *bufsize:* | gibt die Größe dieser Speicherbereiche in Bytes an. |
| *bufcount*: | muß die Anzahl der Zeiger in dem Array angeben. |

**Rückgabe:** '0' im Falle eines korrekten Ablaufs oder einen Fehlercode (siehe Tabelle 3.7 auf Seite 24).

**Header:** io72api.h

# io72MultiDmaRead()

**Name:** **io72MultiDmaRead()** - Lesen von Eingängen der CPCI-DIO72-Karte per DMA

**Aufruf:**

```
int    io72MultiDmaRead
       (
       HANDLE      hnd,
       void        *buffer
       )
```

**Beschreibung:** Diese Funktion blockiert solange bis ein Speicherbereich, der mit *io72MultiDmaStart* dafür vorgesehen wurde, mit Daten gefüllt ist. Ist dies der Fall, so wird an der Stelle, auf die *buffer* zeigt, ein Zeiger, der auf den Beginn dieses gefüllten Speicherbereiches anzeigt, zurückgegeben. Dieser Funktion wird mit *hnd* das Handle einer CompactPCI-digital-I/O Karte übergeben. Vor dem ersten Aufruf von *io72MultiDmaRead* muß mit *io72Config* ein DMA Ereignis konfiguriert werden. Andernfalls blockiert diese Funktion.

**Eingabe:**

| | |
|---|---|
| *hnd:* | Handle, das von *io72Open* zurückgegeben wurde. |
| **buffer*: | Pointer auf Datenbuffer |

**Rückgabe:** '0' im Falle eines korrekten Ablaufs oder einen Fehlercode (siehe Tabelle 3.7 auf Seite 24). Konnten die einzelnen Lesezugriffe nicht in der geforderten Geschwindigkeit bearbeitet werden, so wird IO72_ERR_LOST_DATA zurückgegeben. Wurde bereits mehr als ein Speicherbereich seit dem letzten Aufruf gefüllt, so wird IO72_ERR_LOST_IRQ zurückgegeben.

**Header:** io72api.h

# io72MultiDmaStop()

**Name:** **io72MultiDmaStop()** - Beenden des kontinuierlichen DMA Transfers

**Aufruf:**
```
int    io72MultiDmaStop
       (
       HANDLE          hnd
       )
```

**Beschreibung:** Durch den Aufruf dieser Funktion wird ein DMA Transfer abgebrochen, der mit *io72MultiDmaStart* gestartet wurde. Dieser Funktion wird mit *hnd* das Handle einer CPCI-DIO72-Karte übergeben.

**Eingabe:** **hnd:* Handle, das von *io72Open* zurückgegeben wurde.

**Rückgabe:** '0' im Falle eines korrekten Ablaufs oder einen Fehlercode (siehe Tabelle 3.7 auf Seite 24). Konnten die einzelnen Lesezugriffe nicht in der geforderten Geschwindigkeit bearbeitet werden, so wird IO72_ERR_LOST_DATA zurückgegeben. Wurde bereits mehr als ein Speicherbereich seit dem letzten Aufruf gefüllt, so wird IO72_ERR_LOST_IRQ zurückgegeben.

**Header:** io72api.h

# io72DmaBufferAlloc()

**Name:** **io72DmaBufferAlloc()** - Allokieren des DMA-Speichers.

**Aufruf:**

```
int    io72DmaBufferAlloc
       (
       HANDLE      hnd,
       int         bufsize,
       void        **buffer
       )
```

**Beschreibung:** Diese Funktion muß zum Allokieren von DMA-Speicher benutzt werden. Der Treiber kann maximal IO72_MAXDMABUFCOUNT dieser Speicherblöcke verwalten.

> **Hinweis:** Die Funktion *io72DmaBufferAlloc* ist zur Zeit nur für Linux-Treiber verfügbar.

**Eingabe:**

| | |
|---|---|
| *hnd:* | Handle, das von *io72Open* zurückgegeben wurde. |
| *bufsize:* | Größe des DMA-Speichers in Bytes (max. IO72_MAXDMABUFSIZE Bytes). |
| ***buffer*: | hier wird ein Zeiger auf den DMA-Speicherbereich zurückgegeben |

**Rückgabe:** ‘0’ im Falle eines korrekten Ablaufs oder einen Fehlercode (siehe Tabelle 3.7 auf Seite 24).

**Header:** io72api.h

# io72DmaBufferFree()

**Name:** **io72DmaBufferFree()** - Freigeben des DMA-Speichers.

**Aufruf:**

```
int     io72DmaBufferFree
        (
        HANDLE      hnd,
        int         bufsize,
        void        *buffer
        )
```

**Beschreibung:** Diese Funktion gibt den DMA-Speicher, auf den *buffer* zeigt, wieder frei.

> **Hinweis:** Die Funktion *io72DmaBufferFree* ist zur Zeit nur für Linux-Treiber verfügbar.

**Eingabe:**

| | |
|---|---|
| *hnd:* | Handle, das von *io72Open* zurückgegeben wurde. |
| *bufsize:* | Größe des DMA-Speichers in Bytes (max. IO72_MAXDMABUFSIZE Bytes). |
| **buffer*: | Zeiger auf den DMA-Speicherbereich |

**Rückgabe:** ‘0’ im Falle eines korrekten Ablaufs oder einen Fehlercode (siehe Tabelle 3.7 auf Seite 24).

**Header:** io72api.h

## 3.6 Hilfsfunktionen

# io72GetIrqCount()

**Name:** **io72GetIrqCount()** - Interruptzähler auslesen

**Aufruf:**
```
int    io72GetIrqCount
       (
       HANDLE          hnd
       )
```

**Beschreibung:** Alle von einer CPCI-DIO72-Karte erzeugten Interrupts werden vom Treiber mitgezählt. Mit dieser Funktion kann der Zählerstand dieses Interruptzählers abgefragt werden. Die Funktion kann während der Anwendungsentwicklung hilfreich sein.

**Eingabe:** *hnd:* Handle, das von *io72Open* zurückgegeben wurde.

**Rückgabe:** Die Funktion liefert den aktuellen Stand des Interruptzählers zurück.

**Header:** io72api.h

# io72ResetIrqCount()

**Name:** **io72ResetIrqCount()** - Interruptzähler zurücksetzen

**Aufruf:**
```
int    io72ResetIrqCount
       (
       HANDLE         hnd
       )
```

**Beschreibung:** Mit dieser Funktion kann der Zählerstand des Interruptzählers auf den Wert ‘0’ gesetzt werden (siehe *io72GetIrqCount*).

**Eingabe:** *hnd:* Handle, das von *io72Open* zurückgegeben wurde.

**Rückgabe:** Die Funktion liefert stets den Wert ‘0’ zurück.

**Header:** io72api.h

## 3.7 Rückgabewerte und Fehlercodes der API Aufrufe

| Rückgabewert | Bedeutung |
|---|---|
| '0' | Die Funktion ist erfolgreich ausgeführt worden. |
| ENODEV | Es wurde versucht auf eine nicht existierende oder nicht funktionsfähige CPCI-DIO72-Karte zuzugreifen. Diese Konstante ist in der Datei 'errno.h' der Standard-Include-Dateien definiert. |
| EINVAL | Einer der übergebenen Parameter ist unzulässig. Grund hierfür kann das Verlassen eines zulässigen Wertebereichs sein. Diese Konstante ist in der Datei 'errno.h' der Standard-Include-Dateien definiert. |
| ENOMEN | Der Speicher kann nicht allokiert werden. |
| EBUSY | Die Funktion kann nicht ausgeführt werden, weil eine Resource auf der CPCI-DIO72 bereits benutzt wird (z.B.: es läuft gerade ein DMA-Transfer). |

**Tabelle 3.7:** Rückgabewerte und Fehlercodes

# 4. Funktionsweise der Multi-DMA-Aufrufe

Stellvertretend für die DMA-Aufrufe der Programmierschnittstelle sollen in diesem Kapitel die drei Funktionen für den kontinuierlichen DMA-Transfer erläutert werden.
Die *io72MultiDma . . .* - Aufrufe sind in der Lage, die Eingangsdaten kontinuierlich zu lesen und die Daten nacheinander in mehrere Pufferspeicher zu schreiben.

In Anwendungen, die das Lesen von sehr großen Datenmengen per DMA erfordern, kann der Fall auftreten, daß die Daten auf mehrere freie Speicherbereiche aufgeteilt werden müssen.
Die *io72MultiDma. . .* - Aufrufe können automatisch mehrere Speicherbereiche nacheinander mit Lesedaten füllen, ohne daß aus dem Programmcode heraus ein erneuter DMA Transfer gestartet werden muß.
Der Treiber verwendet für diese Funktionen den Scatter-Gather-DMA Mode der PCI9054 Bridge auf der CPCI-DIO72-1/O-Karte. Durch den API-Aufruf *io72MultiDmaStart* werden zunächst einige Datenstrukturen aufgebaut und initialisiert. Es wird dabei eine Scatter-Gather-Descriptor-Liste im Speicher aufgebaut.

Die einzelnen Elemente dieser Liste sind vom Typ SG_DESCRIPTOR_BLOCK:

```
struct SG_DESCRIPTOR_BLOCK
{
    unsigned long    pci_address;
    unsigned long    local_adress;
    unsigned long    transfer_size;
    unsigned long    next;
};
```

Beschreibung der Felder:

pci_address: Zeigt in jedem Element der Liste auf einen der zu füllenden Zielspeicher, dessen Größe beim Aufruf von *io72MultiDmaStart* festgelegt wurde (siehe Dokumentation zu diesem Aufruf).

transfer_size: Die Größe des Zielspeichers wird in dieses Feld geschrieben.

local_address: Dieses Feld zeigt auf das FPGA-Register im lokalen Adreßraum der PCI-Bridge, das bei jedem einzelnen DMA-Zyklus gelesen werden soll. Der Treiber läßt hier die Auffangregister LATCHA oder LATCHB zu. Das führt zu einem Lesen der jeweiligen Eingänge der CPCI-DIO72-Karte, wenn eine entsprechende Auffangbedingung konfiguriert ist.

next: Das letzte Feld der Listenelemente ist ein Zeiger auf das nächste Element der Scatter-Gather-Liste und dient so dem Verketten der Listenelemente. Die Länge der Liste wird ebenfalls beim Aufruf von *io72MultiDmaStart* festgelegt.
Das Ende der Liste verweist wieder auf den Anfang der Liste, so daß eine Ringstruktur entsteht. Einige Bits des 'next'-Feldes (mode) haben eine Sonderfunktion, die nicht näher erläutert werden soll.

Nachdem die beschriebene Liste aufgebaut ist, wird der Anfang der Liste (chain_start) in ein Konfigurationsregister der PCI-Bridge geschrieben und der Scatter-Gather-DMA-Transfer gestartet. Die Funktion *io72MultiDmaStart* kehrt jetzt zurück, und das Anwendungsprogramm ruft als nächstes die Funktion *io72MultiDmaRead* auf. Diese Funktion blockiert dann solange, bis die PCI-Bridge einen Interrupt auslöst und dadurch das erfolgreiche Füllen des ersten Speicherbereiches meldet.

Die Bridge beginnt selbständig mit dem Füllen des nächsten Speicherbereiches, indem sie die nötigen Daten aus der Scatter-Gather-Descriptor-Liste im Arbeitsspeicher liest.

Jedesmal, wenn ein weiterer Speicherbereich gefüllt ist, wird wieder ein Interrupt ausgelöst.
Ist die gesamte Liste abgearbeitet, so wird wieder am Anfang der Liste fortgefahren (Ringstruktur). Dieser Vorgang kann jederzeit durch den Aufruf von *io72MultiDmaStop* abgebrochen werden. Die Leistungsfähigkeit dieser API-Aufrufe liegt im selbständigen Abarbeiten der Scatter-Gather-Descriptor-Liste durch die PCl-Bridge.

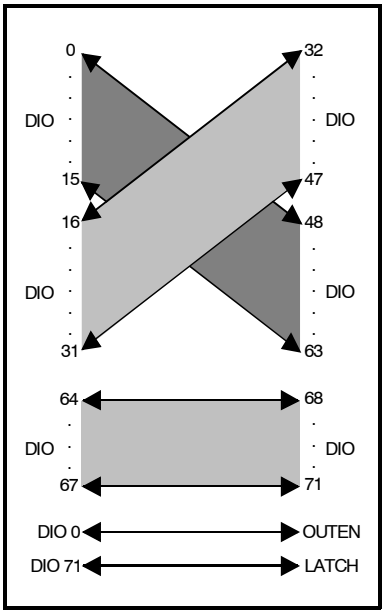
# 5. Beispiele für die Anwendungsprogrammierung

Um die einzelnen Funktionen des Treibers und der Hardware testen zu können, und um einen einfachen Einstieg in die Anwendungsprogrammierung mit der Programmierschnittstelle zu ermöglichen, sind einige Testroutinen erstellt worden.
Über Kommandozeilenoptionen kann beim Aufruf von `io72test` eine bestimmte Testroutine ausgewählt werden. Die folgende Tabelle zeigt eine Übersicht über die möglichen Optionen.

| Option | Funktion |
|---|---|
| `-INT` | Beispiel für das interruptgestützte Lesen der Eingänge |
| `-DMA` | Beispiel für das Lesen der Eingänge per DMA |
| `-LATCH` | Beispiel für die Verwendung des externen Synchronisationseingangs |
| `-EDGE` | Demonstration der Flankenerkennung |
| `-SGDMA` | Demonstration des kontinuierlichen DMA Transfers |
| `-MASK` | Beispiel für das maskierte Schreiben auf Ausgänge |

**Tabelle 5.1:** Optionen für `io72test`

## 5.1 Testadapter




Für eine korrekte Funktion der Testroutinen muß jedoch ein Testadapter aufgesteckt sein, der wie in Abb. 5.1 beschrieben, die einzelnen Signale der Karte miteinander verbindet.
Der Adapter besteht aus einem 80-poligen High-Density-Stecker und einer Vielzahl kurzer Kabelbrücken, die alle auf die Buchse geführten Signale nach einem Muster verbinden.

Die Sondersignale OUTEN und LATCH sind mit den I/O-Kanälen verbunden, so daß diese Signale von der Testsoftware simuliert werden können.

Dieser Testadapter gehört nicht zum Lieferumfang der Karte.

**Abb 5.1:** Verdrahtung des Testadapters

## 5.2 Funktionen des Treibers

Zu Beginn des Quellcodes werden einige Konstanten definiert, die die Bedingungen für die einzelnen Beispielroutinen festlegen (Timerfrequenzen und Buffergrößen für den DMA). Die einzelnen Beispiele sollen nun kurz erläutert werden. Die Testroutinen schreiben vor dem eigentlichen Test stets ein Testmuster auf 32 Kanäle, die zuvor als Ausgang konfiguriert werden. Auf den anderen Kanälen erscheint dieses Testmuster dann über den Testadapter. Auf diese Art werden wird auf allen Kanälen ein Bitmuster gelesen.

### 5.2.1 io72test -INT

Bei diesem Aufruf von `io72test` wird ein timergesteuerter, periodischer Interrupt konfiguriert.
Ausgelöst durch einen Interrupt werden die Eingänge 0 bis 63 dann einige Male in einer Schleife gelesen. *io72lntRead()* liest die Eingangsdaten stets aus den Auffangregistern, so daß zusätzlich eine Auffangbedingung (Capturesource) eingestellt wird. Die während der Leseschleife verstrichene Zeitspanne wird am Ende ausgegeben.

### 5.2.2 io72test -DMA

Durch diesen Aufruf wird ein Speicherbereich mit Eingangsdaten gefüllt. Die Eingangsdaten werden bei jedem Timerimpuls per DMA in den Arbeitsspeicher transferiert.
Am Ende der Übertragung werden die verstrichene Zeitspanne sowie die ersten Inhalte des gefüllten Speicherbereiches ausgegeben.

### 5.2.3 io72test -LATCH

Dieser Test demonstriert die Arbeitsweise des externen Synchronisationseingangs und der Auffangregister. Der externe Synchronisationseingang wird über die I/O-Leitung 71 simuliert.
Im ersten Teil des Tests wird eine Flanke am Synchronisationseingang simuliert und dadurch das aktuelle Eingangsmuster in das Auffangregister übernommen. Im zweiten Teil der Testroutine wird eine einfache Task gestartet, welche erneut eine Flanke auf dem Synchronisationseingang erzeugt.
Beim Auftreten dieser Flanke wird jedoch ein Interrupt erzeugt, auf den die erste Task wartet.

### 5.2.4 io72test -EDGE

Der Test der Flankenerkennung besteht ebenfalls aus zwei Teilen.
Nachdem die Flankenerkennung konfiguriert ist, wird zuerst wird über den Testadapter eine Flanke auf einem I/O-Kanal erzeugt. Die Ergebnisse werden ausgegeben.
Im zweiten Teil der Routine wird eine zweite Task gestartet, die nach einer kurzen Wartezeit mehrere Flanken auf den Eingängen erzeugt. Diese Flanken lösen Interrupts aus, auf die die erste Task wartet.
Das Flankenauffangregister wird am Ende dieses Tests ausgegeben.

### 5.2.5 io72test -SGDMA

`io72test -SGDMA` demonstriert die DMA-Übertragung in der Scatter-Gather-Betriebsart. Zunächst werden einige Speicherbereiche angefordert, die dann später gefüllt werden sollen. Der Timer und das Auffangereignis werden konfiguriert. In einer Schleife wird dann mehrmals auf einen gefüllten Speicherbereich blockierend gewartet. Nach Füllung aller Speicherbereiche wird erneut mit dem ersten begonnen. Dies liegt an der in Kapitel 4 beschriebenen Ringstruktur der Speicherbereiche beim Scatter-Gather-DMA.

### 5.2.6 io72test -MASK

Dieser Test demonstriert die Funktionsweise des *io72MaskWrite*-Aufrufes der Programmierschnittstelle. Bei den Schreibzugriffen in der Routine werden stets nur die Bitpositionen manipuliert, die durch die Maskierungsdaten dazu freigegeben sind.