LINUX Deviceserver für die Transientenrekorder TRC2

Lothar Steffen

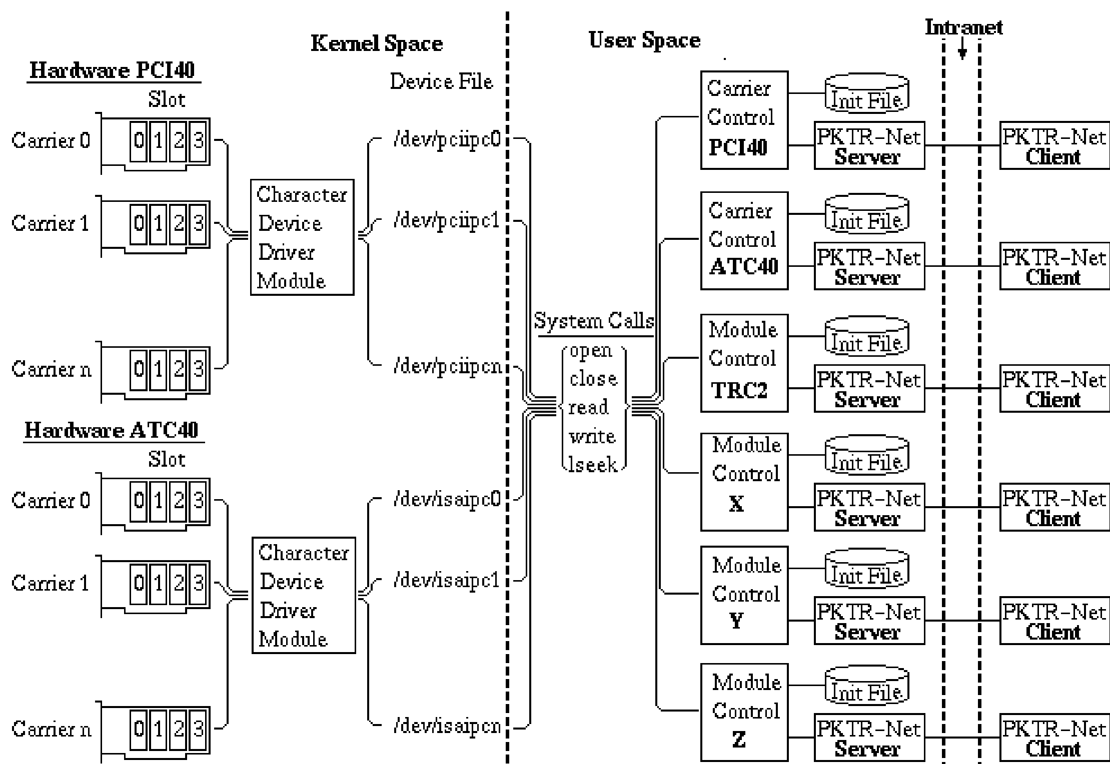
DESY-FEB

22-Dec-1999.

Einführung

IndustryPack-Module, kurz IP-Module genannt, sind zigarettenschachtel große Elektronikgeräte mit genormten Anschlüssen. Während einige Computersysteme direkte Steckmöglichkeiten für IP-Module besitzen (z.B. VME-Rechner) benötigt man für andere Systeme Adapter. Solche Adapter, die eigentlich IP-Carrier heißen, gibt es unter anderem für den PCI und den ISA-Bus. Ein Adapter besitzt mehrere Steckplätze für IP-Module. Am häufigsten sind vier Steckplätze, sogenannte Slots anzutreffen.

Aber wie muß nun die Software aussehen die die IP-Module bedient. Es gibt mehrere verschiedene IP-Carrier und es gibt diverse IP-Module, folglich sind viele Kombinationen denkbar. Einen Treiber zu schreiben, der nur eine Kombination von IP-Carrier und IP-Modul unterstützt erscheint wenig sinnvoll. Viel gescheiter ist es, sich mit der Softwarestruktur an die Hardware anzulehnen. Das bedeutet, für jeden IP-Carrier den man einsetzen will benötigt man einen Treiber und für jedes IP-Modul benötigt man einen weiteren Treiber, so daß immer zwei Treiber zusammenarbeiten müssen wenn man die Hardware ansprechen will. Das folgende Schaubild zeigt die Struktur der Software:



Für den IP-Carrier existiert ein Treiber in Form eines Kernelmoduls. Ein Kernelmodul bewegt sich als Bestandteil des Betriebssystems im sogenannten "Kernel Space", dem Adreßraum des Betriebssystems und hat somit Zugriffsrechte auf die Hardware und sonstige Speicherbereiche die Anwenderprogrammen nicht zur Verfügung stehen. Die Schnittstelle zum Anwenderprogramm ist eine Gerätedatei (Device File). Der

Speicherbereich eines IP-Carriers steht als Datei zur Verfügung. Innerhalb dieses Speicherbereiches kann nun an beliebiger Stelle gelesen oder geschrieben werden. Ein Kernelmodul ist in der Lage mehrere IP-Carrier zu bedienen. Pro IP-Carrier gibt es allerdings jeweils eine Gerätedatei. Greift man also z.B. über die Gerätedatei "/dev/pciipc0" auf die Hardware zu, dann stellt das Kernelmodul die Verbindung zum "Carrier 0" her. Für die anderen Gerätedateien gilt sinngemäß dasselbe. Ein Kernelmodul für einen IP-Carrier ist so universell wie möglich zu gestalten, damit auch IP-Module die es noch zu erfinden gilt damit betrieben werden können. Die übrigen hardwarespezifischen Programmteile laufen als ganz normale Anwenderprogramme im "User Space" des Computers. Im Schaubild sind diese Programme mit "Carrier Control" oder "Module Control" beschriftet. Für jeden Hardwaretyp gibt es ein separates Programm. Im Schaubild sind zwei Carrier Control Programme für die IP-Carrier Boards der Typen PCI40 und ATC40 sowie ein Module Control Programm für das IP-Modul vom Typ TRC2 dargestellt. Weiterhin enthält das Schaubild drei Module Control Programme für IP-Module der Typen X, Y und Z. Diese drei Angaben bezeichnen keine konkrete Hardware sondern sollen nur die Modularität des Softwarekonzepts unterstreichen. Falls in Zukunft ein weiterer IP-Modul Typ eingesetzt werden soll, dann gilt es für dieses neue IP-Modul ein Module Control Programm zu schreiben. Die übrige Software kann unverändert bleiben.

Alle Carrier- und Module Control Programme benötigen Daten über die Hardwareausstattung des Computers. Die Programme müssen wissen hinter welchen Gerätedateien sich Hardware befindet für die sie zuständig sind. Einige Hardwarekomponenten müssen außerdem vor Gebrauch initialisiert werden. Die nötigen Angaben erhalten die Programme aus Initialisierungsdateien die im Schaubild mit "Init File" gekennzeichnet sind. Die "Carrier Control"-Programme bedienen die Konfigurationsregister der IP-Carrier Steckkarten. Die Initialisierungdatei enthält ausschließlich Angaben über die Gerätedateien und die Steuerregister der Carrier. Angaben über die Slots und deren Bestückung mit IP-Modulen sind nicht in der Datei enthalten. Die "Module Control"-Programme bedienen die IP-Module. Die Initialisierungsdatei enthält alle Angaben die zum Zugriff und zur Steuerung des IP-Modultyps notwendig sind. Pro IP-Modultyp gibt es ein separates Module Control Programm. Der Zugriff auf die Gerätedateien erfolgt mit Dateioperationen aus der Standardbibliothek. Das sind die Funktionen "open", "close", "read", "write" und "lseek" die auch im Schaubild angegeben sind.

Ein weiteres Problem ist die Interruptbehandlung. Ein IP-Modul kann zwei Interrupts erzeugen. Bei vier IP-Modulen pro Carrier macht das 8 Interrupts zuzüglich der Interrupts die der Carrier selbst erzeugt. Der Carrier faßt die ganzen Interrupts zu einem Sammelinterrupt zusammen, denn in einem PC sind Interrupts Mangelware und erfordern einen sparsamen Umgang. Jeder Carrier bekommt vom Betriebssystem nur eine Interruptnummer zugewiesen. Wenn ein Carrier einen Interrupt erzeugt wird dies dem Kernelmodul mitgeteilt. Das Kernelmodul muß nun entscheiden was zu geschehen hat. Das ist aber nicht so einfach, weil das Kernelmodul keinerlei Informationen über die Aufgaben und speziellen Eigenschaften der IP-Module besitzt. Diese Informationen sind nicht im Kernelspace des Rechners vorhanden sondern nur im Userspace. Bei einem Kernelmodul, daß einen Treiber für eine konkrete Steckkarte darstellt, z.B. eine Ethernetkarte wäre das kein Problem weil alle benötigten Informationen über die

Hardware während der Softwareentwicklung bekannt sind und in den Quellcode für das Kernelmodul einfließen können. In unseren Fall gilt es einen Weg zu finden die Interruptmeldung an ein Programm weiterzureichen das im Userspace läuft. Die Antwort lautet "kerneld". Dabei handelt es sich um ein Dämonprogramm das vom Kernel Messages bekommt und unter anderem auch Programme starten. Über diesen Weg ist die Interruptverteilung realisiert. Es gibt ein eigenständiges Programm das von kerneld gestartet wird und nichts weiter macht als den "Carrier Control" und "Modul Control" Programmen die Interrupts zu melden und dann selber zu enden.

Ein Programm hat eine gewisse Aufgabe zu erfüllen. Dazu ist es notwendig, daß ein Programm Aufträge annimmt und Ergebnisse mitteilt. Dieser Sachverhalt wird allgemein mit dem Begriff "Schnittstelle" umschrieben. Die Carrier Control und Module Control Programme sind Bestandteil eines übergeordneten Softwarepakets, dem Archivsystem. Das Archivsystem stellt nur geringe Anforderungen an die Schnittstelle. Die Programme lesen beim Start die Initialisierungsdatei, nehmen die erforderlichen Einstellungen vor und befinden sich danach sozusagen in einer Endlosschleife, in der sie Meßdaten aus den IP-Modulen auslesen, zwischenspeichern und in vereinbarter Form dem Archivsystem zur Verfügung stellen. Das ist schon alles. Aber außer dem geschilderten Normalbetrieb gibt es ja auch die Inbetriebnahmephase, Maschinenwartungspausen und die Störungbeseitigung bzw. Fehlersuche die wesentlich höhere Anforderungen an eine Benutzerschnittstelle mit sich bringen. Neben dem Normalbetrieb der ja fast vollautomatisch abläuft gibt es noch die Möglichkeit einen manuellen Betrieb zu machen. Neben der eigentlichen Deviceserversoftware existiert eine Kommandoshell mit der das Programm manuell gesteuert werden kann. Die Kommandoshell ermöglicht gezielten Zugriff auf die Hardware. Die Parameter aller Hardwarekomponenten die bei Normalbetrieb aus der Initialisierungsdatei stammen können angezeigt, erzeugt, verändert und gelöscht werden. Das Einlesen und Abspeichern einer Initialisierungsdatei ist jederzeit möglich. Der Meßbetrieb läßt sich manuell steuern und die gemessenen Daten können unter Zuhilfenahme eines Plotprogramms direkt betrachtet werden. Der Sourcecode ist so gestaltet, daß drei Programme erzeugt werden können, der netzwerkfähige Deviceserver, die netzwerkfähige Kommandoshell und ein nicht netzwerkfähiges Programm das sowohl Deviceserver und Kommandoshell enthält und nur manuellen Betrieb gestattet. Ein als Java-Applet geschriebenes Pendent zu der Kommandoshell ist geplant.

2 Installation

Das Softwarepaket heisst "**ip-driver.tar.gz"** und liegt als "tar"-Archiv vor das mit gzip komprimiert ist. Zunächst einmal überlegt man sich in welchem Verzeichnis die Software installiert werden soll. Anbieten tut sich das Verzeichnis "/usr/local" oder ein "/home"-Verzeichnis eines hierfür extra eingerichteten Benutzers. Im folgenden wird von einer Installation in "/usr/local" ausgegangen.

```
rechnername:~# cp ip-driver.tar.gz /usr/local
```

```
rechnername:~# cd /usr/local
```

Jetzt muß die Datenkompression mit gunzip rückgängig gemacht werden. Das geschieht mit dem Shellkommando:

```
rechnername:/usr/local# gunzip ip-driver.tar.gz
```

Man erhält eine Datei namens "ip-driver.tar" die mit dem Kommando

```
rechnername:/usr/local# tar -xvf ip-driver.tar
```

ausgepackt wird. In dem aktuellen Verzeichnis wird ein neues Unterverzeichnis mit dem Namen "devicedriver" angelegt. Darin befindet sich die gesamte Software.

2.1 Datei- und Verzeichnisstruktur

Das Verzeichnis "devicedriver" hat folgende Struktur:

Die einzelnen Unterverzeichnisse enthalten weitere Unterverzeichnisse. An dieser Stelle soll erst einmal ein grober Überblick gegeben werden damit klar wird an welcher Stelle man was suchen muß.

atc40
Software für die ATC40 IP-Carrier Steckkarte. In diesem Verzeichnis findet man sowohl den Gerätetreiber (Kernelmodul) als auch die Bediensoftware (Userprogramm).

doc
Dokumentation im HTML-Format. Die Hauptseite mit dem Inhaltsverzeichnis heißt "index.html".

general
Software allgemeiner Art die sich keiner Hardwarekomponente zuordnen läßt. Das können Hilfsmittel sein, die dazu dienen dem Anwender das Leben zu erleichtern indem sie Programme die zusammen arbeiten müssen gemeinsam starten usw..

pci40
Software für die PCI40 IP-Carrier Steckkarte. In diesem Verzeichnis findet man sowohl den Gerätetreiber (Kernelmodul) als auch die Bediensoftware (Userprogramm).

pktrnet
Softwarebibliothek für das PKTR-Netz.

scripts
Tools für den automatischen Programmstart.

test
Spielwiese zum Ausprobieren neuer Ideen.

trc2
Software für die TRC2 IP-Module

atc40

Dieses Verzeichnis enthält die Software für die ATC40 IP-Carrier Steckkarte. Zur Zeit befinden sich hier noch Dateien die der Hersteller mitgeliefert hat. Der Linuxtreiber für die ATC40 ist noch nicht fertig. Die Struktur der Unterverzeichnisse ist vorläufiger Art.

doc

Hier findet man Informationen über das gesamte Paket. Die Dokumente liegen als HTML-Dateien vor. Die Hauptseite mit dem Inhaltsverzeichnis heißt "index.html".

general

Alle Komponenten des Pakets die sich keiner Hardwarekomponente zuordnen lassen sind im Verzeichnis "general" untergebracht. In erster Linie sind das Tools die den Komfort erhöhen indem sie den Programmstart mehrerer Einzelprogramme zusammenfassen. Ein Teil der Interruptsteuerung ist hier ebenfalls untergebracht. Die Treiber fuer die IP-Carrier, die als Kernelmodul realisiert sind, starten im Falle eines Interrupts ein ganz normales Anwenderprogramm mittels kerneld, dass die Aufgabe hat alle betroffenen Prozesse zu unterrichten. Die Interruptuebermittlung an die Empfaenger erfolgt mit dem PKTR-Netz Protokoll, weil alle betroffenen Prozesse ohnehin PKTR-Netz Server sind.

user/bin

Makefile
Projektsteuerung.

client.log
Logdatei.

interrupt_targets.txt
Tabelle mit PKTR-Netz Servern und Properties die im Falle eines Interrupts angerufen werden muessen.

IPModuleInterrupt
Ausfuehrbares Programm, das von kerneld im Falle eines Interrupts gestartet wird, die Liste "interrupt_targets.txt" abarbeitet und endet.

user/src

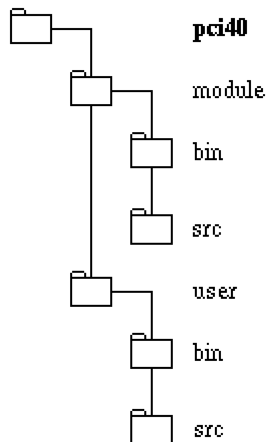
Makefile
Projektsteuerung.

interrupt.c
Sourcecode des Programmes "IPModuleInterrupt".

pktrnet.h
Header. Enthaelt eine Auflistung der benoetigten (#include) Sourcecodedateien des PKTR-Netzes.

pci40

Die Software für die PCI40 IP-Carrier Steckkarte ist hier untergebracht. Die Verzeichnisstruktur verdeutlicht die Unterteilung der Software in das Kernelmodul und das Benutzerprogramm.



Die Unterverzeichnisse "module/bin" und "module/src" enthalten folgende Dateien.

module/bin

Makefile
Projektsteuerung

pci40.o
Diese Datei ist das eigentliche Kernelmodul. Kernelmodule werden zum Zeitpunkt des Installierens dynamisch gelinkt und müssen daher als Objektdatei vorliegen.

pci40_load
Ausführbares Script. Installiert das Kernelmodul "`pci40.o`" und richtet die zugehörigen Gerätedateien ein.

pci40_unload
Ausführbares Script. Löscht das Kernelmodul "`pci40.o`" und entfernt die zugehörigen Gerätedateien.

module/src

Makefile
Projektsteuerung

pci40.h
Headerdatei.

pci40ipc.c
Sourcecode des Kernelmoduls. Diese Datei wird vom Compiler bearbeitet. Das Resultat ist die Datei "`pci40.o`" im Verzeichnis module/bin. Es existieren zwei Versionen des Kernelmoduls - `pci40ipc_int.c` und `pci40ipc_noint.c`. Wenn man sich entschieden hat mit welcher Version man arbeiten möchte, dann kopiert man die entsprechende Datei nach `pci40ipc.c` und startet den Compiler, der die Objektdatei erzeugt.

pci40ipc_int.c
Sourcecode des Kernelmoduls. Diese Version ist gewissermaßen die Vollversion des Kernelmoduls. Sie beinhaltet die Unterstützung für mehrere PCI40 IP-Carrier Steckkarten und für Interrupts (daher der Namenszusatz "`_int`").

pci40ipc_int_old.c
Sourcecode des Kernelmoduls. Diese Version ist von der Funktion her die gleiche wie pci40ipc_int.c, allerdings eine Entwicklungsstufe älter.

pci40ipc_noint.c
Sourcecode des Kernelmoduls. Diese Sparversion ist die erste Version die während der Entwicklungsphase zur Verfügung stand. Sie verzichtet auf vieles was nicht unbedingt zum Betrieb nötig ist. Es kann nur ein PCI40 IP-Carrier angesteuert werden, und zwar derjenige der zuerst vom Rechner-BIOS registriert wurde. Interrupts werden gar nicht unterstützt worauf auch der Namenszusatz "`_noint`" hinweist.

sysdep.h
System-Headerdatei. Relikt aus der Entwicklungsphase. Es ist zu prüfen ob es nicht ohne diese Datei geht.

Die Unterverzeichnisse "user/bin" und "user/src" enthalten alle Dateien des Benutzerprogrammes.

user/bin

Makefile
Projektsteuerung.

client.log
Logdatei.

cshosts.csv
Steuerungsdatei für das Pktr-Netz.

eqpdbase.csv
Steuerungsdatei für das Pktr-Netz.

exports.csv
Steuerungsdatei für das Pktr-Netz.

fecaddr.csv
Steuerungsdatei für das Pktr-Netz.

fecid.csv
Steuerungsdatei für das Pktr-Netz.

herans.csv
Steuerungsdatei für das Pktr-Netz.

pci40ipc.ini
Initialisierung.

pci40shell1
Ausführbares Programm das mittels Kommandoshell gestartet werden kann. Abgesetzte Bedieneroberfläche für die manuelle Steuerung des Deviceservers. Gedacht als Servicetool für das Archivsystem. Dieses Programm startet pci40shell2. Zwischen pci40shell1 und pci40shell2 werden zwei Pipes zur Datenübermittlung installiert.

pci40shell2
Ausführbares Programm, das von pci40shell1 gestartet wird. Ein manueller Start mittels Kommandoshell hat keinen Sinn und sollte deshalb unterbleiben. pci40shell2 stellt die Verbindung zum PKTR-Netz her. pci40shell1 und pci40shell2 stellen zusammengenommen das Clientprogramm dar.

pci40srv1
Ausführbares Programm das mittels Kommandoshell gestartet werden kann. Dieses Programm startet pci40srv2. Zwischen pci40srv1 und pci40srv2 werden zwei Pipes zur Datenübermittlung installiert. pci40srv1 stellt die Verbindung zum PKTR-Netz her.

pci40srv2
Ausführbares Programm, das von pci40srv1 gestartet wird. Ein manueller Start mittels Kommandoshell hat keinen Sinn und sollte deshalb unterbleiben. pci40srv2 stellt die Verbindung zu den PCI40-IP-Carriern her.

pci40srv1 und pci40srv2 stellen zusammengenommen den Deviceserver dar.

server.log
Logdatei.

user/src

Makefile
Projektsteuerung.

cln_eqm.c
Sourcecode. Programmanbindung an das PKTR-Netz als Client.

ipdevice.c
Sourcecode. Zugriff auf die Gerätedateien.

ipdevice.h
Header. Enthält Angaben über die Adreßbereiche der Slots innerhalb der Gerätedateien.

ipdevtest.c
Sourcecode. Dient nur zu Testzwecken. Im Unterschied zu der Datei ipdevice.c findet kein Zugriff auf die Gerätedateien statt.

memory.c
Sourcecode. Dynamische Speicherverwaltung.

memory.h
Header. Definition der Datenstrukturen für die Erfassung und Verwaltung der Hardwareparameter.

pci40usr.c
Sourcecode. Komplexere Programmfunktionen wie Initialisieren aller IP-Carrier, scannen aller Slots nach IP-Modulen.

pci40usr.h
Header. Enthält die Adressen der Steuerregister der PCI40 IP-Carrier Steckkarte.

pktrnet.h
Header. Enthält eine Auflistung der benötigten (#include) Sourcecodedateien des PKTR-Netzes.

readini.c
Sourcecode. Einlesen und Auswerten der Initialisierungsdatei.

serv_eqm.c
Sourcecode. Programmanbindung an das PKTR-Netz als Server.

userinterface.c
Sourcecode. Interface zur Bedieneroberfläche oder zum PKTR-Netz, je nachdem welche Programmversion erstellt wird.

usershell.c
Sourcecode. Bedienerobefläche.

Die ausführbaren Programme setzen sich aus folgenden Sourcecodedateien zusammen.

	pci40local	**pci40client**	**pci40server**
cln_eqm.c		X	
ipdevice.c	X		X
ipdevice.h	X		X
ipdevtest.h			
memory.c	X		X
memory.h	X		X
pci40usr.c	X		X
pci40usr.h	X		X

pktrnet.h	X		X
readini.c	X		X
serv_eqm.c			X
userinterface.c	X		X
usershell.c	X	X	

pktrnet

Die Programmbibliotheken, die benötigt werden um die Netzanbindung ans PKTR-Netz zu ermöglichen befinden sich hier an zentraler Stelle für das ganze Paket. Eigentlich gehören diese Dateien in die "src"-Verzeichnisse der einzelnen Komponenten, aber dann wären die Dateien mehrmals vorhanden.

scripts

Hier stehen die Shellscripts die das tägliche Leben erleichtern. Der Start von Programmen mit diversen Parametern wird so vereinfacht.

rc.TRC2srv
Dieses Script kann manuell, automatisch beim Booten oder regelmässig (z.B. durch crontab) ausgeführt werden. Es prüft ob der Server für das PKTR-Netz läuft, falls nicht wird er gestartet.

test

Dies ist sozusagen die Spielwiese des Systems. Ideen für die Lösung von Detailproblemen können hier mit Beispielprogrammen ausprobiert werden. Dadurch bleiben die eigentlichen Programme von Spielereien verschont und die Innovationen behalten ihren Bezug zum Gesamtprojekt.

trc2

Hier befindet sich die Software für die Steuerung der TRC2-IP-Module.

user/bin

***.bat**
Batchdatei, die vom Programm trc2shell1 ausgeführt werden kann.

***.ini**
Initialisierungsdatei. Enhält alle Angaben ueber die Hardwareausstattung des Systems sowie die Parameter.

Makefile
Projektsteuerung.

client.log
Logdatei.

cshosts.csv
Steuerungsdatei für das Pktr-Netz.

eqpdbase.csv
Steuerungsdatei für das Pktr-Netz.

exports.csv
Steuerungsdatei für das Pktr-Netz.

fecaddr.csv
Steuerungsdatei für das Pktr-Netz.

fecid.csv
Steuerungsdatei für das Pktr-Netz.

herans.csv
Steuerungsdatei für das Pktr-Netz.

server.log
Logdatei.

trc2shell1
Ausführbares Programm.

trc2shell2
Ausführbares Programm.

trc2srv1
Ausführbares Programm.

trc2srv2
Ausführbares Programm.

trc2.ini
Initialisierungsdatei die beim Programmstart automatisch gelesen wird. Initialisierungsdateien mit anderem Namen können nur manuell über die eine Benutzeroberfläche benutzt werden.

user/src

Makefile
Projektsteuerung

automatic.c
Sourcecode für trc2srv3.

cln_eqm.c
Sourcecode für trc2shell2. Anbindung an das PKTR-Netz.

extended_io.c
Sourcecode für trc2srv1, trc2srv2 und trc2srv3.

extended_io.h
Headerdatei für trc2srv1, trc2srv2 und trc2srv3.

filedata.c
Sourcecode für trc2srv2. Messwerte eines Kanales in eine lokale Datei schreiben.

ipdevice-new.c
Sourcecode für die nächste Version von trc2srv2. Wird zur Zeit noch nicht eingesetzt.

ipdevice.c
Sourcecode für trc2srv2. Zugriff auf die Gerätedatei.

ipdevice.h
Headerdatei für trc2srv2. Adressen der Gerätedatei.

ipdevtest.c
Sourcecode für trc2srv2. **ipdevtest.c** ersetzt alle Funktionen der Datei **ipdevice.c** durch Dummyfunktionen ohne Zugriff auf die Gerätedateien. Diese Datei wird nur benötigt wenn das Programm als Testversion erstellt wird. Die Testversion ist ohne besondere Hardware auf jedem Linuxrechner lauffähig. Nicht mal das Kernelmodul wird benötigt.

local.h
Headerdatei für trc2shell1. Siehe Beschreibung von **mode.h**.

memory.c
Sourcecode für trc2srv2. Anlegen, verwalten und abbauen der dynamischen Speicherstrukturen für die Hardwareparameter.

memory.h
Headerdatei für trc2srv2. Definition der Datensatzstrukturen.

mode.h
Headerdatei für trc2shell1. Das Programm trc2shell1 (Kommandooberfläche) muss während der Compilierung gesagt bekommen ob es das Programm trc2shell2 (Client für das Pktr-Netz) oder das Programm trc2srv2 (Zugriff auf Transientenrecorderhardware) starten soll. Dazu wird die Datei **mode.h** ausgewertet. Vor der Compilierung wird entweder die Datei **local.h** oder die Datei **remote.h** nach **mode.h** kopiert.

pktrnet.h
Headerdatei für trc2srv2. Definition der Datenstrukturen die über das PKTR-Netz übertragen werden.

readini.c
Sourcecode für trc2srv2. Einlesen und Abspeichern der Initialisierungsdatei.

remote.h
Headerdatei für trc2shell1. Siehe Beschreibung von **mode.h**.

serv_eqm.c
Sourcecode für trc2srv1. Anbindung an das PKTR-Netz.

trc2.c
Sourcecode für trc2srv2. Funktionen für den Zugriff auf die Register des TRC2-IP-Moduls.

trc2.h
Headerdatei für trc2srv2. Adressen und sonstige Parameter des TRC2-IP-Moduls.

userinterface.c
Sourcecode für trc2srv2. Enthält die gesamte Schnittstelle zu den Programmteilen trc2srv1 und trc2shell1 sowie das Hauptprogramm (main).

usershell.c
Sourcecode für trc2shell1. Enthält die gesamte Kommandoshell.

2.2 Compilieren des Sourcecodes

Bei den folgenden Beispielen wird wieder davon ausgegangen, dass die Software im Verzeichnis "/usr/local" installiert ist. Nach der Installation wechselt man in das Verzeichnis "**devicedriver**".

```
rechnername:~# cd /usr/local/devicedriver
```

In diesem Verzeichnis befindet sich das ranghöchste Makefile des Softwarepaketes. Die Compilierung des Sourcecodes wird von hier gesteuert. Zwei Alternativen stehen zur Auswahl. Mit

```
rechnername:/usr/local/devicedriver# make local
```

erzeugt der Compiler das Kernelmodul sowie Programmversionen der Carrier-Controlprogramme und IP-Modul-Controlprogramme ohne Anbindung an das PKTR-Netz. Für den lokalen Einsatz im Entwicklungslabor ist dies die richtige Wahl. Benötigt man allerdings die netzwerkfähigen Programmversionen, z.B. für den Einsatz als FrontEnd in der Maschine, dann muss man

```
rechnername:/usr/local/devicedriver# make net
```

eingeben. Der Compiler erzeugt jetzt das Kernelmodul sowie die netzwerkfähigen Programmversionen der Carrier- und IP-Modul-Controlprogramme. Zwischen zwei Compilerläufen ist es ratsam mit

```
rechnername:/usr/local/devicedriver# make clean
```

alle überflüssigen Dateien zu löschen. Dazu zählen unter anderem von Editorprogrammen angelegte Backups und vom Compiler erzeugte Objektdateien.

3.1 ATC40 IP-Carrier Steckkarte
- noch zu schreiben -

3.2 PCI40 IP-Carrier Steckkarte
Hierbei handelt es sich um einen IP-Carrier für den PCI-Bus. Bis zu 4 IP-Module einfacher Größe können auf die PCI40-Steckkarte gesteckt werden.

Konfiguration
Die PCI40-Steckkarte besitzt keine Jumper. Die gesamte Konfiguration geschieht mittels Software.

Status und Controlregister (lt. Hersteller)

- **CNTL0**
  Offset 0x00000500 = control register 0 [CNTL0]

Bit	Bitname	Definition
D0	CLKA	0 = 8MHz, 1 = 32 Mhz for IP slot A
D1	CLKB	0 = 8MHz, 1 = 32 Mhz for IP slot B
D2	CLKC	0 = 8MHz, 1 = 32 Mhz for IP slot C
D3	CLKD	0 = 8MHz, 1 = 32 Mhz for IP slot D
D4	CLR_AUTO	0 = clear, 1 = enable bus error timer interrupt
D5	AUTO_ACK	0 = disable, 1 = enable bus error timer
D6	INTEN	0 = disable interrupts, 1 = enable interrupts
D7	INTSET	0 = turn off, 1 = force local interrupt [INTEN = 1]

CNTL1
Offset 0x00000600 = control register 1 [CNTL1]

Bit	Bitname	Definition
D0	IRQA0	0 = no interrupt, 1 = interrupt pending
D1	IRQA1	0 = no interrupt, 1 = interrupt pending
D2	IRQB0	0 = no interrupt, 1 = interrupt pending
D3	IRQB1	0 = no interrupt, 1 = interrupt pending
D4	IRQC0	0 = no interrupt, 1 = interrupt pending
D5	IRQC1	0 = no interrupt, 1 = interrupt pending
D6	IRQD0	0 = no interrupt, 1 = interrupt pending
D7	IRQD1	0 = no interrupt, 1 = interrupt pending

- CNTL2
  [read only] offset 0x00000700 = control register 2 [CNTL2]

Bit	Bitname	Definition
D0	X	unused
D1	X	unused
D2	X	unused
D3	X	unused
D4	X	unused
D5	X	unused
D6	Auto_Int_Set	0 = no interrupt, 1 = bus error interrupt pending
D7	LINT	0 = no interrupt, 1 = interrupt pending to PLX

Interrupts

Das PCI40-Board bildet alle Interrupts auf das INTA# Signal der PCI-Backplane ab. Weil die PCI Interrupts shared sind kann ein Interrupt von einer PCI-Steckkarte oder vom Motherboard kommen. Der PCI40-Treiber muß zuerst feststellen ob der Interrupt vom PCI40-Board stammt indem er das CNTL0 Register des PCI40 liest. Kommt der Interrupt vom PCI40 dann bestimmt man die Quelle indem man die CNTL1 Register liest und die Bits überprüft.

Auf den Interrupt-Adreßraum eines jeden Slots kann man jederzeit zugreifen. Normalerweise wird die Interrupt Service Routine auf den Interrupt-Adreßraum zugreifen um die lokale Ursache des Interrupts zu bestimmen.

CNTL0-Bits

CLR_AUTO
Das CLR_AUTO-Steuerbit wird benutzt um einen Interrupt durch die Bus-Error-Timer-Funktion zu ermöglichen und später zu löschen. Wenn das AUTO_ACK-Bit, das CLR_AUTO-Bit und das INTEN-Bit alle auf "1" gesetzt sind, wird ein Interrupt erzeugt wenn die Software auf eine Speicheradresse zugreift die nicht antwortet (also nicht existiert weil z.B. der IP-Slot frei ist). Die Software muß das CNTL0 und das CNTL1-Register lesen um die Ursache des Interrupts festzustellen. Das Setzen des CLR_AUTO-Bit auf "0" löscht den Bus-Error-Timer-Interrupt. Das Bit muß auf "1" zurückgesetzt werden um den AUTO_ACK Interrupt zu ermöglichen. Mit INTEN gesperrt (disable) und CLR_AUTO freigegeben (enable) kann AUTO_ACK benutzt werden um den Zustand nach jedem Zugriff abzufragen.

INTEN
Das INTEN-Bit ist ein Interruptmaskenbit, welches logisch mit der Interruptquelle verUNDet wird. Wenn es auf "1" gesetzt ist wird eine Anfrage an das PLX weitergereicht. Wenn es auf "0" gesetzt ist bleibt die Anfrage im Raum stehen wird aber vom PLX geblockt. Wenn INTEN auf "1" gesetzt ist und ein IP-Modul einen Interrupt meldet wird das PLX einen INTA#-Interrupt auf dem PCI-Bus erzeugen. Das PLX muß so eingestellt sein daß es Interrupts durchläßt.

INTSET
Das INTSET-Bit wird zusammen mit dem INTEN-Bit zur Erzeugung eines Interrupts benutzt. Die Interruptquelle ist innerhalb des ALTERA Steuerungs-PLD. Wenn INTEN auf "1" gesetzt ist bewirkt das Setzen von INTSET auf "1" eine Interruptmeldung an das PLX. Wenn Interrupts vom lokalen Bus freigegeben sind wird der INTA# des PCI-Busses aktiviert. Dies ist eine nützliche Einrichtung für die Fehlersuche.

3.3 Transienten Recorder IP-Module

Die Transienten Recorder IP-Module unterstützen 8 Frontend-Kanäle. Ein IP-Modul hat eine Flachkabelverbindung zu einem abgesetztem Gerät, das sich DTU (Digital Trigger Unit) nennt. Die DTU besitzt die Anschlüsse und Bedienelemente die sich aus Platzgründen nicht am IP-Modul unterbringen lassen, als da sind Steckanschlüsse für acht Tastköpfe sowie Schalter und Steckanschlüsse für Timingsignale. Die DTU übernehmen außerdem die Spannungsversorgung für die Tastköpfe. Es handelt sich um aktive Tastköpfe mit integrierter Elektronik, deren Eigenschaften vom IP-Modul verändert werden können. Die Datenübertragung zwischen Tastkopf und IP-Modul findet auf seriellem Wege statt. Das IP-Modul unterstützt alle Standard IP-Funktionen bis auf DMA. Die IP-Busfrequenz beträgt 8MHz.

Jeder Kanal hat eine Speichergröße von 8192 Worten (1 Wort = 16 Bit).

3.3.1 Betriebsarten

Die IP-Module befinden sich in einem der folgenden Betriebsmodi.

Data taking
Das Stop Signal ist nicht aktiv (logisch L). Ein Trigger startet einen Lesezyklus aller 8 Kanäle. Die Triggerquelle ist entweder intern oder extern. Der interne Trigger erfolgt alle 10.5µs, das ergibt eine Abtastfrequenz von 95238.095 Hz. Die seriellen Daten kommen von der Frontend-Elektronik mit einer Taktrate von 2MHz, werden in parallele Daten konvertiert und im Speicher abgelegt. Der Datenspeicher ist als Ringspeicherstruktur ausgelegt. Sobald das Ende des Speichers erreicht ist, werden die ältesten Daten überschrieben. In diesem Betriebsmodus kann der Datenspeicher nicht ausgelesen werden. Allerdings kann man auf einen Buffer zugreifen und somit den zuletzt gemessenen Wert auslesen. Die Software kann Trigger und Stopsignale sperren und freigeben sowie den Betriebsmodus wechseln.

Stop transition
Vorausgesetzt das Modul befindet sich in normalem **Data taking** Modus und ein Stopsignal trifft ein oder die Stopbedingung wird erfüllt, dann nimmt das IP-Modul noch eine vordefinierte Anzahl von Meßwerten. Diese Zahl muß vor der Messung im **Software Control** Betriebsmodus in das Register cy_post_reg geschrieben werden. Am Ausgangspin ioSTOP_SYN wird der Stopzustand der Außenwelt mitgeteilt. Damit kann man mehrere IP-Module miteinander synchronisieren. Wenn die Messung beendet ist wird ein Interrupt erzeugt (IP Interrupt 0). Die Stopbedingung, die bereits erwähnt wurde, ergibt sich aus der Verknüpfung der Meßdaten mit den Inhalten mehrerer Register. Zur Anwendung kommt folgende Formel:

```
condition = ((data[i] AND mask[i]) XOR xor[i])
operator[i] level[i]
```

wobei

i	**Nummer des Kanales**
data[]	Daten vom Tastkopf
mask[]	Inhalt des 16 Bit Registers mask
xor[]	Inhalt des 16 Bit Registers xor
operator[]	`= < <= = !=`, im 16 Bit Register config
level[]	Inhalt des 16 Bit Registers level

Data read-out
Der Datenspeicher kann von der Software ausgelesen werden. Im 16 Bit Register rx_address steht die Nummer (0-8191) des letzten Meßwertes.

Software Control
In diesem Betriebsmodus lassen sich die meisten Einstellungen machen. Die Tastköpfe können konfiguriert werden. Ein einzelner Lesezyklus kann ausgelöst werden. Die letzten beiden Eigenschaften sind mit einem gemeinsamen Interrupt verbunden (IP Interrupt 1).

Der Abtastzeitpunkt der seriellen Eingangsdaten kann relativ zur ioCLK verschoben werden um Zeitverzögerungen zu kompensieren die durch unterschiedliche Kabellängen der Tastköpfe hervorgerufen werden. Damit das funktioniert muß die Frontend-

Elektronik in der Lage sein genau definierte Pulsketten zum IP-Modul zu senden. Nach einem Reset befindet sich das IP-Modul in folgendem Zustand:

- Die Timing Eingänge sind gesperrt.
- Die Anzahl der Lesezyklen ist auf 0 gesetzt.
- Die Interrupts sind gesperrt.
- Die Pointer zeigen auf den Speicheranfang, aber der Speicherinhalt ist nicht gelöscht.

Jeder Initialisierungszyklus wird erkannt. Alle IO Übertragungszyklen erfolgen so schnell wie möglich bis auf das Lesen der letzten Meßdaten (rx_dio_sel), das bis zu 9.5µs dauern kann. Das IP-Modul verwendet einen wait state auf alle Speicherzugriffe.

3.3.2 Programmierung

3.3.2.1 Übersicht: IO Adressen

Zugriff	Größe	Adresse						Erforderlicher Betriebsmodus	Registername
		A6	A5	A4	A3	A2	A1		
write	16 Bit	0	1	0	0	0	0	Software Control	rx_trigger
read	16 Bit	0	0	1	C	C	C	beliebig	rx_dio_sel (CCC)
write	16 Bit	0	0	1	C	C	C	Software Control	tx_write (CCC)
read/write	8 Bit	0	0	0	0	1	0	beliebig	control_word
write	8 Bit	0	0	0	0	0	0	beliebig	intrpt_vec0
write	8 Bit	0	0	0	0	0	1	beliebig	intrpt_vec1
read	16 Bit	0	0	0	0	1	1	beliebig	rx_address
read	8 Bit	0	0	0	1	0	0	beliebig	status
write	16 Bit	0	1	1	0	0	0	Data Taking	cy_sw_stop
write	16 Bit	0	1	1	0	0	1	Software Control	cy_post_reg
write	16 Bit	0	1	1	0	1	0	Software Control	clock_shift
write	16 Bit	1	0	0	C	C	C	Software Control	mask (CCC)
write	16 Bit	1	0	1	C	C	C	Software Control	level (CCC)
write	16 Bit	1	1	0	C	C	C	Software Control	xor (CCC)
write	16 Bit	1	1	1	C	C	C	Software Control	config (CCC)

3.3.2.2 Register:

rx_trigger:
Startet einen einzelnen Lesezyklus im Betriebsmodus Software Control, ähnlich einem Puls am Eingang ioTRIGGER. Das ist aber nur möglich wenn der Trigger freigegeben ist. Dazu muß im **control_word** das Bit 5 (trigger enable) auf 1 gesetzt werden. **rx_trigger** ist in erster Linie für Hardwaretests während der Inbetriebnahme oder Wartung gedacht. Für den eigentlichen Meßbetrieb wird diese Funktion nicht benötigt.

rx_dio_sel (CCC):
Liest den letzten Meßwert des Kanals CCC aus einem Zwischenspeicher.

tx_write (CCC):
Sendet 16 Bit zur Frontend-Elektronik des Kanales CCC. Diese Funktion dient der Konfiguration der Tastköpfe. Die Bedeutung der 16 Bit ist von der Art des Tastkopfes abhängig. Außerdem sollte man daran denken, daß die Datenübertragung seriell stattfindet. Es kommt nicht so sehr darauf an, daß auch wirklich 16 Bit gesendet werden. Es können durchaus weniger sein. Aber die Reihenfolge ist sehr wichtig. Am besten geht man so vor, daß man die Konfigurationsdaten in eine 16 Bit Integervariable schreibt und soweit nach links verschiebt daß keine unbenutzten Bits mehr links von den Konfigurationsdaten stehen.

control_word:
Dieses Register dient der Steuerung des IP-Modul. Die einzelnen Bits haben folgende Bedeutung:

Bit	Bedeutung
D7	mode (1)
D6	mode (0)
D5	trigger enable (1)
D4	stop enable (1)
D3	interrupt0 enable (1) post cycles
D2	interrupt1 enable (1) tx & rx
D1	trigger source (1=ext, 0=int)
D0	io_stop_syn enable (1)

mode
Hiermit wird der Betriebsmodus vorgewählt und so müssen die Bits gesetzt werden:

mode 1	mode 0	Betriebsmodus
1	1	data taking (DT)
1	0	stop transition (ST)
0	1	data read-out (DR)
0	0	software control (SW)

Für einen reibungslosen Betrieb ist zu beachten, daß nicht beliebig von einem Betriebsmodus in einen anderen gewechselt werden kann. Nur folgende Wechsel sind erlaubt:

- data taking (DT) - data read-out (DR)
- data read-out (DR) - software control (SW)
- software control (SW) - data taking (DT)

Für einen reibungslosen Betrieb ist es ratsam diesen Wechsel des Betriebsmodus in zwei Schritten durchzuführen.

- software control (SW) - stop transition (ST)
- stop transition (ST) - data taking (DT)
- software control (SW) - data read-out (DR)

trigger enable
0 - IP-Modul reagiert nicht auf Triggerpulse

1 - IP-Modul regiert auf Triggerpulse

Dieses Bit wirkt auf alle Triggerarten.

interner Trigger

externer Trigger am Pin io_TRIGGER

Einzeltrigger durch Schreibzugriff auf das Register rx_trigger.

stop enable
0 - IP-Modul reagiert nicht auf Stopsignale

1 - IP-Modul reagiert auf Stopsignale

Dieses Bit wirkt auf alle Arten des Stopsignals.

externes Stopsignal am Pin ioSTOP

interne Stopbedingung die aus den Meßwerten generiert wird.

Softwarestop durch Schreibzugriff auf das Register cy_sw_stop.

interrupt0 enable
0 - Interrupt 0 wird nicht erzeugt

1 - Interrupt 0 wird erzeugt wenn die Erfassung der Meßwerte beendet wurde weil eine Stopbedingung eingetreten ist und die Anzahl der Meßwerte nach dem Stopsignal, die im Register cy_post_reg steht, aufgezeichnet wurden.

interrupt1 enable
0 - Interrupt 1 wird nicht erzeugt

1 - Interrupt 1 wird erzeugt wenn

nach einem Schreibzugriff auf das Register rx_trigger das IP-Modul einen Meßwert komplett eingelesen hat und der serielle Empfänger für den nächsten Meßwert bereit ist. Alternativ könnte man auch das Statusregister auslesen und prüfen ob das Bit D5 (rx_rdy) gesetzt ist.

nach einem Schreibzugriff auf das Register tx_write das IP-Modul die Konfigurationsdaten an einen Tastkopf gesendet hat und der serielle Sender bereit ist den nächsten Tastkopf zu konfigurieren. Alternativ könnte man auch das Statusregister auslesen und prüfen ob das Bit D4 (tx_rdy) gesetzt ist.

trigger source
0 - interner Trigger

1 - externer Trigger

io_stop_syn enable
0 - IP-Modul gibt keine Stopsignale am Pin ioSTOP_SYN aus.

1 - IP-Modul gibt Stopsignale am Pin ioSTOP_SYN aus.

Beim Auslesen dieses Registers erhält man nur die Daten zurückgeliefert die man selber geschrieben hat. Eine Reaktion des IP-Moduls läßt sich nicht daraus ableiten. Wenn man Software entwickelt die mit dem IP-Modul zusammen arbeiten soll kann man das control_word Register gut für die ersten Schritte benutzen weil es das einzige Register ist deren Inhalt man schreiben und lesen kann.

intrpt_vec0:
intrpt_vec1:
Vor der Aufnahme des Meßbetriebs muß auf diese beiden Register ein Schreibzugriff erfolgen. Dadurch wird die Interruptunterstützung des IP-Moduls initialisiert. Der Wert der geschrieben wird, stellt den Interruptvektor dar. Diesem Interruptvektor kommt allerdings nur eine Bedeutung zu wenn das IP-Modul auf einen VME-Rechner gesteckt wird. Setzt man das IP-Modul mittels IP-Carrier in einem PC ein, dann funktioniert die Interruptbehandlung nach einem anderen Prinzip. Der Wert der in die Register geschrieben wird ist dann ohne Belang.

rx_address:
Enthält die Nummer des zuletzt gespeicherten Meßwertes (0 - 8191).

status:
Zeigt den augenblicklichen Betriebszustand des IP-Moduls

Bit	Bedeutung	Zustand nach einem Reset
D7	mode (0)	0
D6	mode (1)	0
D5	rx_rdy receiver ready (1)	1
D4	tx_rdy transmitter ready (1)	1
D3	ioSTOP input	0
D2	ioTRIGGER input	0
D1		0
D0		0

cy_sw_stop:
Stoppt die Aufzeichnung von Meßwerten. Dieses Register ist gleichbedeutend wie ein Puls am Eingang ioSTOP

cy_post_reg:
Anzahl der Meßwerte die nach dem Eintreten der Stopbedingung noch aufgezeichnet werden sollen.

clockshift:
Zeitverzögerung

Bit	Shift Bit	Kanal
D15	sh1	7
D14	sh0	
D13	sh1	6
D12	sh0	
D11	sh1	5
D10	sh0	
D9	sh1	4
D8	sh0	
D7	sh1	3
D6	sh0	
D5	sh1	2
D4	sh0	

<table>
<tr><td>D3</td><td>sh1</td><td rowspan="2">1</td></tr>
<tr><td>D2</td><td>sh0</td></tr>
<tr><td>D1</td><td>sh1</td><td rowspan="2">0</td></tr>
<tr><td>D0</td><td>sh0</td></tr>
</table>

Die Bedeutung der Shift Bits geht aus folgender Tabelle hervor:

Shift Bit		Zeitverzögerung
sh1	sh0	
0	0	0 ns
0	1	156 ns
1	0	312 ns
1	1	467 ns

mask (CCC):
Parameter für die interne Stopbedingung.

level (CCC):
Parameter für die interne Stopbedingung.

xor (CCC):
Parameter für die interne Stopbedingung.

config (CCC):
Parameter für die interne Stopbedingung.

Wert	Bedeutung
0	=
1	<
2	
3	=
4	<=
5	!=
6	disable
7	disable

3.3.2.3 Zugriff auf den Datenspeicher

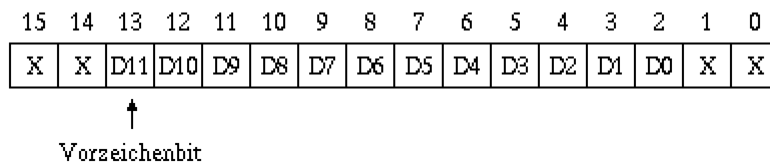
Die Meßwerte aller Kanäle stehen hintereinander im Speicher. Als erstes kommen die 8192 Meßwerte des Kanales 0 und als letztes die 8192 Meßwerte des Kanales 7. Zum Auslesen der Werte benötigt man die Adressen der Speicherplätze. Die Ermittlung setzt etwas Rechnerei voraus. Kompliziert wird die Sache dadurch, daß die IP-Module eine wortweise Organisation des Speichers besitzen, aber einige Rechnersysteme eine byteweise Adressierung praktizieren. Byteadressen und Wortadressen dürfen nicht durcheinander gebracht werden. Angenommen, das IP-Modul hat die Datennahme beendet und steht im Betriebsmodus data read-out DR. Nun muß der Speicher ausgelesen werden. Wir lesen das Register rx_address aus und erfahren welcher Meßwert zuletzt geschrieben wurde. Das Register rx_address enthält einen Wert im Bereich von 0 bis 8191, also gewissermaßen einen Offset in wortweiser Adressierung, man könnte auch Meßwertnummer dazu sagen. Um die Adresse innerhalb des IP-Modulspeichers zu erhalten müssen wir die Anzahl der Meßwerte pro Kanal (8192) multipliziert mit der Kanalnummer (0 bis 7) hinzuaddieren.

IPModulMemory(16 Bit) = Meßwertnummer + Anzahl der Meßwerte pro Kanal * Kanalnummer

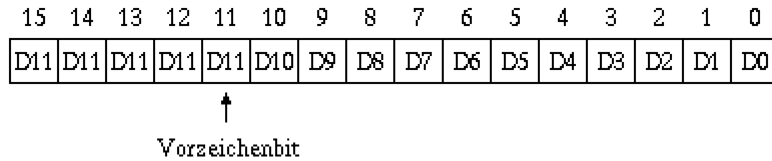
Besitzt das System das die IP-Module steuert eine byteweise Adressierung dann muß dieser Wert mit 2 multipliziert werden. Dieser Fall ist z.B. gegeben wenn das IP-Modul mittels IP-Carrier Steckkarte in einem Standard PC eingesetzt wird.

IPModulMemory(8 Bit) = 2 * (Meßwertnummer + Anzahl der Meßwerte pro Kanal * Kanalnummer)

Hat man die richtige Adresse ermittelt und ausgelesen, dann müssen die 16Bit Daten noch in ein Standarddatenformat umgewandelt werden. Die AD-Wandler liefern nur 12Bit-Daten inklusive Vorzeichenbit. Also enthalten 4 der 16Bit keine Information. Außerdem steht das Vorzeichenbit an der falschen Stelle. Das folgende Bild zeigt den Zustand einer 16Bit-Variablen (signed short integer) nach dem Auslesen des IP-Moduls. Die relevanten Daten befinden sich in den Bits 2 bis 13. Der Zustand der Bits 0,1,14 und 15 ist undefiniert.



Der Inhalt der 16Bit-Variablen muß um zwei Bit nach rechts verschoben werden. Die Daten befinden sich dann in den Bits 0 bis 11. Die Bits 12 bis 15 müssen auf den gleichen Wert gesetzt werden wie das Vorzeichenbit 11. Wenn das geschehen ist sieht die 16Bit-Variable so aus.



Die Daten entsprechen jetzt einem Standarddatenformat (signed short integer) und sind fertig für die Ablieferung an das Archivsystem.

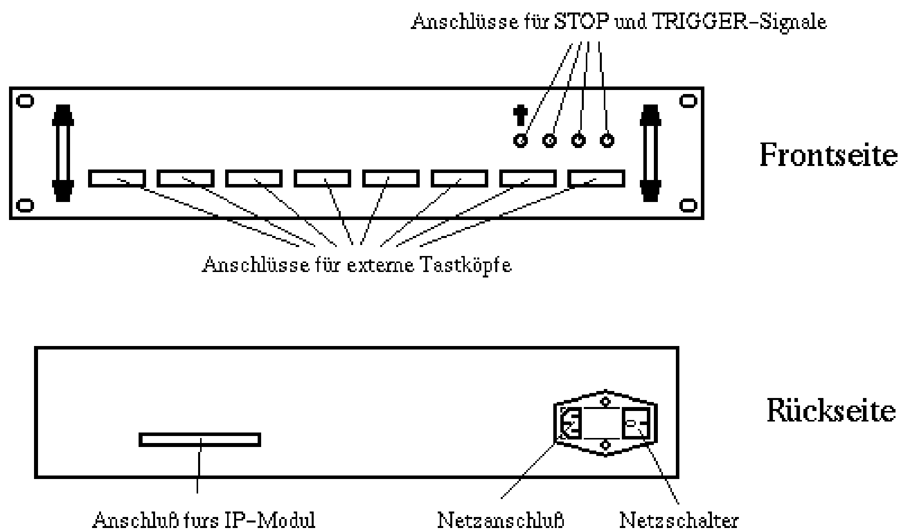
3.3.3 Anschlüsse

Alle Ein- und Ausgangssignale sind mit RS422-Treiber bestückt. Die Pulsbreite der externen Trigger liegt zwischen 500ns und 5µs.

output			input		
Pin	Signal	Bemerkung	Pin	Signal	Bemerkung
1	ground		26	ground	
2	-		27	-	
3	ioCLK_		28	ioTRIGGER_	
4	ioCLK		29	ioTRIGGER	
5	ioSTOP_SYN_		30	ioSTOP_	
6	ioSTOP_SYN		31	ioSTOP	
7	ioDtoFE_(0)		32	ioDtoIP_(0)	
8	ioDtoFE(0)		33	ioDtoIP(0)	
9	ioDtoFE_(1)		34	ioDtoIP_(1)	
10	ioDtoFE(1)		35	ioDtoIP(1)	
11	ioDtoFE_(2)		36	ioDtoIP_(2)	
12	ioDtoFE(2)		37	ioDtoIP(2)	
13	ioDtoFE_(3)		38	ioDtoIP_(3)	
14	ioDtoFE(3)		39	ioDtoIP(3)	
15	ioDtoFE_(4)		40	ioDtoIP_(4)	
16	ioDtoFE(4)		41	ioDtoIP(4)	
17	ioDtoFE_(5)		42	ioDtoIP_(5)	
18	ioDtoFE(5)		43	ioDtoIP(5)	
19	ioDtoFE_(6)		44	ioDtoIP_(6)	

20	ioDtoFE(6)		45	ioDtoIP(6)	
21	ioDtoFE_(7)		46	ioDtoIP_(7)	
22	ioDtoFE(7)		47	ioDtoIP(7)	
23	-		48	-	
24	-		49	-	
25	ground		50	ground	

3.4 Device and Trigger Unit DTU



- noch zu schreiben -

3.5 Tastköpfe

3.5.1 Analoger Tastkopf

3.5.2 Digitaler Tastkopf

- noch zu schreiben -

4.1 Kernelmodule

- noch zu schreiben -

4.1.1 ATC40 IP-Carrier

- noch zu schreiben -

4.1.2 PCI40 IP-Carrier

- noch zu schreiben -

Carrier Control Programme

ATC40 IP-Carrier

4.2.1.1 Struktur

- noch zu schreiben -

4.2.1.2 Properties

- noch zu schreiben -

4.2.1.3 Kommandosyntax

- noch zu schreiben -

PCI40 IP-Carrier

4.2.2.1 Struktur

- noch zu schreiben -

4.2.2.2 Properties

Der pci40-Server ist nur mit einer Property für das PKTR-Netz ausgestattet.

COMMAND

- Eingangsdaten
    - Datenformat: CF_TEXT
    - Datengröße: variabel

- Ausgangsdaten
    - Datenformat: CF_TEXT
    - Datengröße: variabel

Diese Property dient als Sammelbecken für eine ganze Reihe von Kommandos. Die Steuerung des Servers kann in vollem Umfang über diese Property abgewickelt werden. Als Kommandos dienen die Funktionsnamen des C-Sourcecodes. Zur Übermittlung eines Kommandos wird eine Zeichenkette erstellt die den Funktionsnamen gefolgt von den Parametern enthält. Als Trennzeichen dient das Komma.

Kommando,Parameter 1,Parameter 2, ... Parameter n

Beispiel:

get_device_parameter,/dev/pciip0

Jetzt folgt eine Auflistung aller Kommandos mit den möglichen Parametern und Rückgabewerten.

IP-Carrier initialisieren carrier_init

`carrier_init` schreibt die Konfigurationsdaten, die sich in den dynamischen Speicherstrukturen befinden, in die IP-Carrier. Es können nur alle IP-Carrier zusammen konfiguriert werden. Eine gezielte Auswahl eines IP-Carriers ist nicht möglich.

Prototyp und Parameter

`carrier_init`

Rückgabewert

"ok" bei fehlerfreier Ausführung, sonst "error".

Verweise

IP-Carrier nach IP-Modulen abtasten carrier_scan

`carrier_scan` greift auf die ID-Bereiche der IP-Module zu und wertet sie aus.

Prototyp und Parameter

`carrier_scan`

Rückgabewert

"ok" bei fehlerfreier Ausführung, sonst "error".

Verweise

Gerätedatei löschen delete_device

`delete_device` löscht eine Gerätedatei aus der Liste.

Prototyp und Parameter

`delete_device,device`

`device`:	Name der Gerätedatei z.B. /dev/pciip0

Rückgabewert

"ok" bei fehlerfreier Ausführung, sonst "error".

Verweise

`do_device`

AUTO_ACK-Bit setzen do_auto_ack

`do_auto_ack` setzt das AUTO_ACK-Bit des Control-Registers 0.

Prototyp und Parameter

`do_auto_ack,device,value`

`device`:	Name der Gerätedatei z.B. /dev/pciip0
`value`:	gewünschter Zustand des AUTO_ACK-Bit, mögliche Angaben: enable disable

Rückgabewert

"ok" bei fehlerfreier Ausführung, sonst "error".

Verweise

Busfrequenz für Slot A setzen do_clka

`do_clka` setzt die Busfrequenz des IP-Busses.

Prototyp und Parameter

`do_clka,device,value`

`device`:	Name der Gerätedatei z.B. /dev/pciip0
`value`:	Busfrequenz des IP-Busses. Mögliche Angaben 32MHz 8MHz

Rückgabewert

"ok" bei fehlerfreier Ausführung, sonst "error".

Verweise

Busfrequenz für Slot B setzen do_clkb

`do_clkb` setzt die Busfrequenz des IP-Busses.

Prototyp und Parameter

do_clkb,device,value

`device`:	Name der Gerätedatei z.B. /dev/pciip0
`value`:	Busfrequenz des IP-Busses. Mögliche Angaben 32MHz 8MHz

Rückgabewert

"ok" bei fehlerfreier Ausführung, sonst "error".

Verweise

Busfrequenz für Slot C setzen do_clkc

`do_clkc` setzt die Busfrequenz des IP-Busses.

Prototyp und Parameter

`do_clkc,device,value`

`device`:	Name der Gerätedatei z.B. /dev/pciip0
`value`:	Busfrequenz des IP-Busses. Mögliche Angaben 32MHz 8MHz

Rückgabewert

"ok" bei fehlerfreier Ausführung, sonst "error".

Verweise

Busfrequenz für Slot D setzen do_clkd

`do_clkd` setzt die Busfrequenz des IP-Busses.

Prototyp und Parameter

do_clkd,device,value

device:	Name der Gerätedatei z.B. /dev/pciip0
value:	Busfrequenz des IP-Busses. Mögliche Angaben 32MHz 8MH

Rückgabewert

"ok" bei fehlerfreier Ausführung, sonst "error".

Verweise

CLR_AUTO-Bit setzen do_clr_auto

`do_clr_auto` setzt das CLR_AUTO-Bit des Control-Registers 0.

Prototyp und Parameter

`do_clr_auto,device,value`

device:	Name der Gerätedatei z.B. /dev/pciip0
value:	gewünschter Zustand des CLR_AUTO-Bit, mögliche Angaben: enable clear

Rückgabewert

"ok" bei fehlerfreier Ausführung, sonst "error".

Verweise

Gerätedatei hinzufügen do_device

`do_device` legt einen Eintrag für eine neue Gerätedatei an.

Prototyp und Parameter

`do_device,device`

device:	Name der Gerätedatei z.B. /dev/pciip0

Rückgabewert

"ok" bei fehlerfreier Ausführung, sonst "error".

Verweise

`delete_device`

Slot bzw. IP-Modul hinzufügen do_slot

`do_slot` legt einen Eintrag für ein neues IP-Modul bzw. Slot an.

Prototyp und Parameter

`do_slot,device,slot`

`device`:	Name der Gerätedatei z.B. /dev/pciip0
`slot`:	Steckplatz des IP-Moduls (A,B,C oder D)

Rückgabewert

"ok" bei fehlerfreier Ausführung, sonst "error".

Verweise

`delete_slot`

INTEN-Bit setzen — do_inten

`do_inten` setzt das INTEN-Bit des Control-Registers 0.

Prototyp und Parameter

`do_inten,device,value`

`device`:	Name der Gerätedatei z.B. /dev/pciip0
`value`:	gewünschter Zustand des INTEN-Bit, mögliche Angaben: enable disable

Rückgabewert

"ok" bei fehlerfreier Ausführung, sonst "error".

Verweise

INTSET-Bit setzen — do_intset

`do_intset` setzt das INTSET-Bit des Control-Registers 0.

Prototyp und Parameter

`do_intset,device,value`

`device`:	Name der Gerätedatei z.B. /dev/pciip0
`value`:	gewünschter Zustand des INTSET-Bit, mögliche Angaben: enable disable

Rückgabewert

"ok" bei fehlerfreier Ausführung, sonst "error".

Verweise

Zustand des AUTO_ACK-Bit ermitteln get_auto_ack

`get_auto_ack` ermittelt den Zustand des AUTO_ACK-Bits des Control-Registers 0.

Prototyp und Parameter

`get_auto_ack,device`

`device`:	Name der Gerätedatei z.B. /dev/pciip0

Rückgabewert

"enable" oder "disable" bei fehlerfreier Ausführung, sonst "error".

Verweise

Busfrequenz von Slot A ermitteln get_clka

`get_clka` ermittelt die Busfrequenz von Slot A.

Prototyp und Parameter

`get_clka,device`

device:	Name der Gerätedatei z.B. /dev/pciip0

Rückgabewert

Busfrequenz (8MHz oder 32MHz) bei fehlerfreier Ausführung, sonst "error".

Verweise

Busfrequenz von Slot B ermitteln get_clkb

`get_clkb` ermittelt die Busfrequenz von Slot B.

Prototyp und Parameter

`get_clkb,device`

`device`:	Name der Gerätedatei z.B. /dev/pciip0

Rückgabewert

Busfrequenz (8MHz oder 32MHz) bei fehlerfreier Ausführung, sonst "error".

Verweise

Busfrequenz von Slot C ermitteln get_clkc

`get_clkc` ermittelt die Busfrequenz von Slot C.

Prototyp und Parameter

`get_clkc,device`

`device`:	Name der Gerätedatei z.B. /dev/pciip0

Rückgabewert

Busfrequenz (8MHz oder 32MHz) bei fehlerfreier Ausführung, sonst "error".

Verweise

Busfrequenz von Slot D ermitteln — get_clkd

`get_clkd` ermittelt die Busfrequenz von Slot D.

Prototyp und Parameter

`get_clkd,device`

`device`:	Name der Gerätedatei z.B. /dev/pciip0

Rückgabewert

Busfrequenz (8MHz oder 32MHz) bei fehlerfreier Ausführung, sonst "error".

Verweise

Zustand des CLR_AUTO-Bit ermitteln — get_clr_auto

`get_clr_auto` ermittelt den Zustand des CLR_AUTO-Bit.

Prototyp und Parameter

`get_clr_auto,device`

`device`:	Name der Gerätedatei z.B. /dev/pciip0

Rückgabewert

"enable" oder "clear" bei fehlerfreier Ausführung, sonst "error".

Verweise

Control-Register 0 eines IP-Carriers auslesen — get_cntl0

`get_cntl0` ermittelt den Zustand des Control-Registers 0.

Prototyp und Parameter

`get_cntl0,device`

`device`:	Name der Gerätedatei z.B. /dev/pciip0

Rückgabewert

Zurückgegeben wird eine Zeichenkette mit dem Inhalt des Control-Registers 0 in hexadezimaler Darstellung. Im Fehlerfall wird "error" zurückgegeben.

Verweise

Control-Register 1 eines IP-Carriers auslesen get_cntl1

`get_cntl1` ermittelt den Zustand des Control-Registers 1.

Prototyp und Parameter

`get_cntl1,device`

device:	Name der Gerätedatei z.B. /dev/pciip0

Rückgabewert

Zurückgegeben wird eine Zeichenkette mit dem Inhalt des Control-Registers 1 in hexadezimaler Darstellung. Im Fehlerfall wird "error" zurückgegeben.

Verweise

Control-Register 2 eines IP-Carriers auslesen get_cntl2

`get_cntl2` ermittelt den Zustand des Control-Registers 2.

Prototyp und Parameter

`get_cntl2,device`

device:	Name der Gerätedatei z.B. /dev/pciip0

Rückgabewert

Zurückgegeben wird eine Zeichenkette mit dem Inhalt des Control-Registers 2 in hexadezimaler Darstellung. Im Fehlerfall wird "error" zurückgegeben.

Verweise

Gerätedatei ermitteln get_device

`get_device` ermittelt den Namen der Gerätedatei.

Prototyp und Parameter

get_device,device

device:	Name der Gerätedatei z.B. /dev/pciip0

Rückgabewert

Name der Gerätedatei bei fehlerfreier Ausführung, sonst "error".

Verweise

Auflistung aller Gerätedateien ermitteln get_device_list

get_device_list erstellt eine Auflistung aller Gerätedateien.

Prototyp und Parameter

`get_device_list`

Rückgabewert

Zurückgegeben wird eine Zeichenkette mit den Namen aller Gerätedateien, durch Komma getrennt. Im Fehlerfall wird "error" zurückgeliefert..

Verweise

`do_device, delete_device`

Parameter einer Gerätedatei ermitteln get_device_parameter

`get_device_parameter` ermittelt die Einstellung aller Bits des Control-Registers 0.

Prototyp und Parameter

`get_device_pointer,device`

`device:`	Name der Gerätedatei z.B. /dev/pciip0

Rückgabewert

Zurückgegeben wird eine Zeichenkette mit den Ergebnissen der Funktionen

- `get_clka`
- `get_clkb`
- `get_clkc`
- `get_clkd`
- `get_clr_auto`
- `get_auto_ack`
- `get_inten`
- `get_intset`

Die einzelnen Angaben sind durch Kommas getrennt. Im Fehlerfall, wenn z.B. die Gerätedatei nicht im Speicher bekannt ist, wird "error" zurückgegeben.

Verweise

`do_clka, do_clkb, d0_clkc, do_clkd, get_clka, get_clkb, get_clkc, get_clkd`

Existenz einer Gerätedatei prüfen get_device_pointer

`get_device_pointer` durchsucht den Speicher nach der genannten Gerätedatei.

Prototyp und Parameter

`get_device_pointer,device`

`device`:	Name der Gerätedatei z.B. /dev/pciip0

Rückgabewert

"ok" bei fehlerfreier Ausführung, sonst "error".

Verweise

Zustand des INTEN-Bit ermitteln get_inten

`get_inten` ermittelt den Zustand des INTEN-Bits des Control-Registers 0.

Prototyp und Parameter

get_inten,device

`device`:	Name der Gerätedatei z.B. /dev/pciip0

Rückgabewert

"enable" oder "disable" bei fehlerfreier Ausführung, sonst "error".

Verweise

Zustand des INTSET-Bit ermitteln get_intset

`get_intset` ermittelt den Zustand des INTSET-Bits des Control-Registers 0.

Prototyp und Parameter

`get_intset,device`

`device`:	Name der Gerätedatei z.B. /dev/pciip0

Rückgabewert

"enable" oder "disable" bei fehlerfreier Ausführung, sonst "error".

Verweise

Steckplatz des IP-Moduls ermitteln get_slot

`get_slot` ermittelt den Steckplatz des IP-Moduls.

Prototyp und Parameter

`get_slot,device,slot`

`device:`	Name der Gerätedatei z.B. /dev/pciip0
`slot:`	Steckplatz des IP-Moduls (A,B,C oder D)

Rückgabewert

Steckplatz des IP-Moduls bei fehlerfreier Ausführung, sonst "error".

Verweise

Auflistung aller IP-Module einer Gerätedatei erstellen get_slot_list

`get_slot_list` erstellt eine Liste aller IP-Module die zu der genannten Gerätedatei gehören..

Prototyp und Parameter

`get_slot_list,device`

`device:`	Name der Gerätedatei z.B. /dev/pciip0

Rückgabewert

Zurückgegebeb wird eine Zeichenkette mit den Bezeichnungen der belegten Slots, durch Komma getrennt. Im Fehlerfall wird "error" zurückgeliefert..

Verweise

`do_slot, delete_slot`

ID-Bereich des IP-Moduls auslesen get_slot_parameter

`get_slot_parameter` liest den ID-Bereich eines IP-Moduls. Die Software unterscheidet selbsttätig zwischen dem "ID PROM Data Format I" und dem "ID PROM Data Format II" und stellt aus den Angaben eine einheitliche Liste zusammen.

Prototyp und Parameter

`get_slot_parameter,device,slot`

`device:`	Name der Gerätedatei z.B. /dev/pciip0
`slot:`	Steckplatz des IP-Moduls (A,B,C oder D)

Rückgabewert

Zurückgeliefert wird eine Zeichenkette die folgende Angaben enthält.

- IP_module ASCII
- Manufacturer ID
- Model Number

- Revision
- Reserved
- Driver ID
- 16-bit Flags
- Number of bytes used
- CRC

Die Angaben sind durch Komma voneinander getrennt. Im Fehlerfall wird "error" zurückgegeben.

Verweise

get_slot_pointer

`get_slot_pointer`

Prototyp und Parameter

`get_slot_pointer,device,slot`

`device`:	Name der Gerätedatei z.B. /dev/pciip0
`slot`:	Steckplatz des IP-Moduls (A,B,C oder D)

Rückgabewert

"ok" bei fehlerfreier Ausführung, sonst "error".

Verweise

Server beenden — quit

`quit` beendet die Programmausführung des Servers.

Prototyp und Parameter

`quit`

keine Parameter

Rückgabewert

keiner

Verweise

Allgemeiner Lesezugriff auf Gerätedateien — read_general

`read_general` liest beliebige Daten aus einer Gerätedatei. Dabei kann es sich sowohl um Register des Carriers als auch um Speicherbereiche von IP-Modulen handeln. Hierbei handelt es sich um eine elementare Funktion die keinen Komfort bietet. Die angegebenen Parameter werden benutzt ohne sie vorher zu checken. Wenn man also eine Adresse angibt an der sich kein Speicher befindet dann stürzt der Rechner ab!

Prototyp und Parameter

`read_general,device,address,count`

`device`:	Name der Gerätedatei
`address`:	Adresse innerhalb der Gerätedatei in hexadezimaler Schreibweise.
`count`:	Anzahl der Bytes die gelesen werden sollen, maximal 64. Wenn es sich um eine gerade Zahl handelt findet wortweiser Zugriff statt. Bei einer ungeraden Zahl findet byteweiser Zugriff statt.

Rückgabewert

Zurückgeliefert wird eine Zeichenkette die mit Steuerzeichen für eine Ausgabe im Textmodus durchsetzt ist. In der ersten Zeile steht, zur Kontrolle, die Anzahl der gelesenen Bytes. Die übrigen Zeilen enthalten bis zu acht Bytes in hexadezimaler Darstellung. Die einzelnen Bytes sind hier durch Leerzeichen voneinander getrennt und nicht durch Kommas. Die Rückgabedaten könnten z.B. so aussehen:

```
no of bytes read : 18
0x00 0x01 0x02 0x03 0x04 0x05 0x06 0x07
0x08 0x09 0x0A 0x0B 0x0C 0x0D 0x0E 0x0F
0x10 0x11
```

Im Fehlerfall wird "error" zurückgeliefert.

Verweise

`write_general`

Initialisierungsdatei lesen — read_init

`read_init` liest die Initialisierungsdaten aus einer Datei.

Prototyp und Parameter

`read_init,filename`

`filename`:	Pfad und Name der Initialisierungsdatei

Rückgabewert

"ok" bei fehlerfreier Ausführung, sonst "error".

Verweise

`carrier_init, write_init`

Allgemeiner Schreibzugriff auf Gerätedateien — write_general

`write_general` schreibt beliebige Daten in eine Gerätedatei. Dabei kann es sich sowohl um Register des Carriers als auch um Speicherbereiche von IP-Modulen handeln. Hierbei

handelt es sich um eine elementare Funktion die keinen Komfort bietet. Die angegebenen Parameter werden benutzt ohne sie vorher zu checken. Wenn man also eine Adresse angibt an der sich kein Speicher befindet dann stürzt der Rechner ab!

Prototyp und Parameter

`write_general,device,address,count,value`

`device`:	Name der Gerätedatei
`address`:	Adresse innerhalb der Gerätedatei in hexadezimaler Schreibweise.
`count`:	Anzahl der Bytes die geschrieben werden sollen, maximal 64. Wenn es sich um eine gerade Zahl handelt findet wortweiser Zugriff statt. Bei einer ungeraden Zahl findet byteweiser Zugriff statt.
`value`:	Zeichenkette mit den zu schreibenden hexadezimalen Daten in ASCII-Darstellung. Für die Darstellung eines Bytes werden zwei Schriftzeichen benötigt. Führende "0x" dürfen nicht geschrieben werden. Eine solche Zeichenkette entsteht wenn die Daten über die Tastatur eingegeben werden. Die Software wandelt Schritt für Schritt die beiden Schriftzeichen eines Bytes in eine echte hexadezimale Zahl um, die in einem Byte Platz hat, bevor der Zugriff auf die Hardware erfolgt.

Rückgabewert

Zurückgeliefert wird eine Zeichenkette mit der Anzahl der geschriebenen Bytes.

Beispiel: no of bytes written : 4

Im Fehlerfall wird "error" zurückgeliefert.

Verweise

read_general

Initialisierungdatei schreiben write_init

`write_init` schreibt die Initialisierungsdaten in eine Datei.

Prototyp und Parameter

`do_carrier_init`

`filename`:	Pfad und Name der Initialsierungsdatei

Rückgabewert

"ok" bei fehlerfreier Ausführung, sonst "error".

Verweise

`carrier_init, read_init`

4.2.2.3 Kommandosyntax

Die Programmbedienung orientiert sich an den Kommandointerpretern wie sie von MS-DOS oder UNIX bekannt sind. Die Baumstruktur der Hardware ist im Programm abgebildet. An oberster Stelle in der Hierarchie stehen die Gerätedateien der IP-Carrier. Direkt darunter sind die einzelnen Slots der IP-Carrier angeordnet, die die IP-Module aufnehmen. Für die Reise durch die Hardwareebenen und die Manipulation der Angaben werden die gleichen Kommandos verwendet mit denen man in einem Computerdateisystem die anstehenden Arbeiten verrichtet. Sowohl die UNIX-Kommandos als auch die entsprechenden MS-DOS Versionen sind implementiert.

Prompt

Als Prompt findet der Programmname gefolgt von ">" Verwendung.

```
pci40:>
```

Der Prompt verändert sein Aussehen wenn man in den Hardwareebenen rauf und runterwechselt. In dem folgenden Fall zeigt der Prompt an, daß wir in die Gerätedatei "/dev/pciip0" gewechselt sind.

```
pci40:/dev/pciip0:>
```

Die Doppelpunkte dienen dazu die Hardwareebenen voneinander zu trennen. Die bekannten Trennzeichen "\" oder "/" kommen dafür leider nicht in Frage, weil sie Bestandteil des Gerätedateinamens sind.

Kommandos

jetzt folgt eine alphabetische Auflistung aller verfügbaren Kommandos mit Angabe von Beispielen.

cat

Die Parameter einzelner Einträge einer Hardwareebene kann man sich auf die gleiche Weise anzeigen lassen wie den Inhalt einer Datei. Die von UNIX bekannten Kommandos "cat" und "more" und das von MS-DOS bekannte Kommando "type" dienen dazu.

Wendet man dieses Kommando auf eine Gerätedatei an, dann bekommt man die Einstellung des Control Registers 0 angezeigt.

```
pci40:>more /dev/pciip0
device: /dev/pciip0
------------------------------
control register 0
   D0, CLKA : 8MHz
   D1, CLKB : 8MHz
   D2, CLKC : 8MHz
   D3, CLKD : 8MHz
   D4, CLR_AUTO (bus error timer interrupt) : clear
   D5, AUTO_ACK (bus error timer)           : enable
   D6, INTEN    (interrupts)                : disable
   D7, INTSET   (local interrupt [INTEN=1]) : disable
Bei einem Slot wird der Inhalt des ID-Bereiches dargestellt.
```

```
pci40:/dev/pciip0:>more A
device: /dev/pciip0, slot: A
------------------------------
   ip-module ASCII     : IPAC
   Manufacturer ID     : 15 (unregistered)
   Model Number        : 17
   Revision            : 1
   Reserved            : 0
   Driver ID           : 0
   16-bit Flags        : 0
   Number of bytes used: 12
   CRC                 : 224
```

Es ist noch zu erwähnen, daß man sich in der richtigen Hardwareebene befinden muß wenn man sich die Einstellungen ansehen will. Den Inhalt des Control Registers 0 eines Carriers bekommt man nur dann angezeigt wenn man sich in der höchsten Ebene der Hardware befindet, d. h. im Prompt darf kein Gerätedateinamen enthalten sein. Ähnliches gilt für die ID-Bereiche von IP-Modulen. Der Prompt muß eine Gerätedatei aber er darf keine Slotbezeichnung enthalten. Folgendes Kommando führt nicht zu dem gewünschten Ergebnis:

```
pci40:/dev/pciip0>cat /dev/pciip0
```

cd

Dieser von MS-DOS und UNIX gleichermaßen benutzte Befehl dient dazu die Hardwareebenen zu wechseln, analog den Verzeichnisebenen eines Dateisystems. Als Parameter muß die gewünschte Gerätedatei angegeben werden um tiefer in die Hardware einzudringen. Den umgekehrten Weg beschreitet man mit dem bekannten Parameter "..", wobei aber zu beachten ist, daß zwischen "cd" und ".." ein Leerzeichen steht, wie es unter UNIX üblich ist. Das MS-DOS-Kuriosum "cd.." ohne Leerzeichen nach dem "cd" wird nicht akzeptiert. Der Parameter ".", der auf das aktuelle Verzeichnis verweist wird ebenfalls akzeptiert. In einem Punkt unterscheidet sich der Befehl allerdings von seinem Vorbild. Man kann nämlich immer nur eine Ebene rauf oder runterschalten. Ein Überspringen einer Ebene, indem man einen entsprechenden Parameter eingibt ist nicht möglich.

```
pci40:>cd /dev/pciip0

pci40:/dev/pciip0:>cd ..

pci40:>
```

del

Hardwarekomponenten aus der Liste löschen. Dazu dienen die UNIX-Kommandos "rmdir" und "rm" und das MS-DOS Kommando "del".

dir

Damit man sich nicht das gesamte Hardwaregebilde merken muß kann man die Einträge der einzelnen Ebenen anzeigen lassen. Von MS-DOS ist der Befehl "dir" bekannt, das UNIX-Pendent lautet "ls".

```
pci40:>ls
available devices:
------------------
   /dev/pciip0
   /dev/pciip1
pci40:>cd /dev/pciip0
pci40:/dev/pciip0:>dir
available slots:
----------------
   A
   B
   C
   D
pci40:/dev/pciip0:>
```

init

Initialisierungsdaten bzw. Hardwaredaten in die Hardware schreiben. Dieser Schritt muß nach dem Einlesen einer *.ini-Datei und nach jeder Änderung der Einstellungen erfolgen. Die Software beschreibt nur die Register der IP-Carrier. Die IP-Module bleiben unbehelligt.

ls

Siehe "dir".

md

Neue Hardwarekomponenten in die Liste einfügen. Das geschieht mit den Kommandos mit denen man in einem Dateisystem neue Verzeichnisse erzeugt. In UNIX ist das der Befehl "mkdir" und in MS-DOS "md".

mkdir

Siehe "md".

more

Siehe "cat".

read

Lesen von Daten. Aus der Sicht der Software können verschiedene Arten von Daten von verschiedenen Quellen gelesen werden. Man hat die Wahl zwischen drei Möglichkeiten.

Option	Beschreibung
`-b [Datei]`	Datei als Batchdatei benutzen
`-batch [Datei]`	
`-i [Datei]`	Datei als Initialisierungsdatei benutzen
`-init [Datei]`	

-m [Register]	Registerinhalt der Hardware lesen
-modul [Register]	

Batchdatei lesen

```
pci40:>read -b <Dateiname>.bat
```

Initialisierungsdatei lesen

Der Dateiname kann frei gewählt werden, allerdings ist die Erweiterung ".ini" wichtig. Wenn die angegebene Datei nicht mit der Erweiterung ".ini" endet geht das Programm von einem Schreibfehler des Anwenders aus und versucht die Datei "pci40ipc.ini" zu lesen.

```
pci40:>read -i <Dateiname>.ini
```

Registerinhalt lesen

Vier verschiedene Registernamen werden akzeptiert.

- `cntl0`
- `cntl1`
- `cntl2`
- `general`

`cntl0, cntl1, cntl2` sind Control Register des Carriers.

Gibt man als Registernamen "general" an, dann bekommt man einen allgemeinen Zugriff auf die Gerätedatei. Mit dieser Methode kann man beliebig im gesamten Speicherbereich, den der Gerätetreiber zur Verfügung stellt, schreiben und lesen. Natürlich muß man die richtige Adresse und Registergröße wissen.

```
pci40:>read -m general
address [hex] : 0500
no of bytes [dezimal] : 1
no of bytes read : 1
0x30
pci40:>
```

Bei der Angabe der Anzahl der Bytes gibt es noch etwas zu beachten. In der IP-Welt gibt es Register auf die wortweise zugegriffen werden muß während andere byteweisen Zugriff erlauben. Der Gerätetreiber ist nun so programmiert, daß wortweiser Zugriff immer dann stattfindet wenn die Anzahl der zu transportierenden Bytes gerade ist, ansonsten findet byteweiser Zugriff statt.

rm

Siehe "del".

rmdir

Siehe "del".

scan

ID-Prom Bereiche der IP-Slots abtasten. IP-Module müssen sich, laut Spezifikation, über den ID-Prom Bereich identifizieren. Zur Zeit gibt es drei unterschiedliche Formen des ID-Prom Bereiches:

- IPAC (Module nach älterer Spezifikation mit 8MHz Bustakt)
- IPAH (Module nach älterer Spezifikation mit 32MHz Bustakt)
- VITAII (Module nach neuerer Spezifiaktion mit 8MHz oder 32MHz Bustakt)

Die Software erkennt alle drei Formen. Beim Abtasten der entsprechenden Speicherbereiche ist der Bus-Error-Timer eingeschaltet. Die Inhalte der ersten zwei Worte werden dargestellt.

```
pci40:>scan
device=/dev/pciip0, slot=A, ID: 0x0049, 0x0050
device=/dev/pciip0, slot=B, ID: 0x0049, 0x0050
device=/dev/pciip0, slot=C, ID: 0x0049, 0x0050
device=/dev/pciip0, slot=D, ID: 0x0049, 0x0050
device=/dev/pciip1, slot=A, ID: 0x0049, 0x0050
device=/dev/pciip1, slot=B, ID: 0x0049, 0x0050
device=/dev/pciip1, slot=C, ID: 0x0049, 0x0050
device=/dev/pciip1, slot=D, ID: 0x0049, 0x0050
```

In dem Beispiel sind alle Slots mit Modulen nach älterer Spezifikation bestückt. Wenn ein Slot als bestückt erkannt wird, wird ein Listeneintrag im Speicher angelegt und die Daten des kompletten ID-Prom Bereiches in den Speicher geschrieben. Diese Daten kann man mit dem Kommando "more" abrufen.

type

Siehe "cat".

write

Schreiben von Daten

Aus der Sicht der Software können verschiedene Arten von Daten gelesen und geschrieben werden. Auch die Quelle oder das Ziel können unterschiedlich sein. Da gibt es zunächst einmal die Einstellungs-, Konfigurations- oder Initialisierungsdaten der Hardware. Diese Daten können als gewöhnliche Datei vorliegen oder sich im Speicher des Programmes befinden. Man benötigt zunächst einmal die Möglichkeit diese Daten aus der Datei in den Speicher zu lesen. Dazu dient der Befehl:

```
pci40:>read -i <Dateiname>.ini
```

Der Dateiname kann frei gewählt werden, allerdings ist die Erweiterung ".ini" wichtig. Wenn die angegebene Datei nicht mit der Erweiterung ".ini" endet geht das Programm

von einem Schreibfehler des Anwenders aus und versucht die Datei "pci40ipc.ini" zu lesen. Änderungen in der Hardware lassen sich mit dem Befehl:

```
pci40:>write -i <Dateiname>.ini
```

in eine Datei zurückschreiben. Die Behandlung des Dateinamen erfolgt hier genauso wie bei der Read-Funktion. Die nächste Sort von Daten sind die Registerinhalte der Hardware. Wenn man den Parameter "-m" benutzt werden die Register angesprochen.

`-b oder -batch`	Datei als Batchdatei benutzen
`-i oder -init`	Datei als Initialisierungsdatei benutzen
`-m oder -modul`	Registerinhalte lesen/schreiben

Vier verschiedene Registernamen werden akzeptiert.

- `cntl0`
- `cntl1`
- `cntl2`
- `general`

Gibt man als Registernamen "general" an, dann bekommt einen allgemeinen Zugriff auf die Gerätedatei. Mit dieser Methode kann man beliebig im gesamten Speicherbereich, den der Gerätetreiber zur Vefügung stellt, schreiben und lesen. Natürlich muß man die richtige Adresse und Registergröße wissen.

```
pci40:>read -m general
address [hex] : 0500
no of bytes [dezimal] : 1
readdata : IOaddr = 0x00000500
readdata : buf[0] = 0x30
no of bytes read : 1
0x30
pci40:>
```

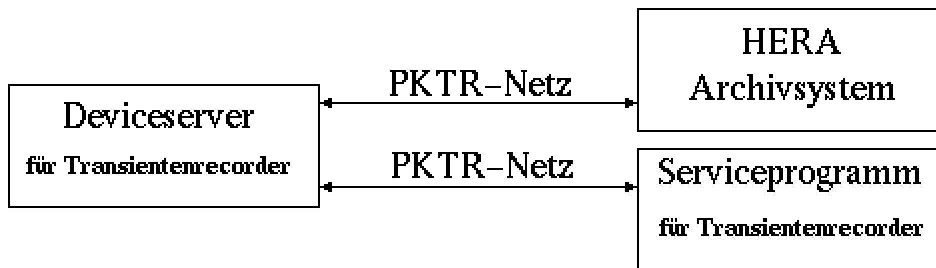
Bei der Angabe der Anzahl der Bytes gibt es noch etwas zu beachten. In der IP-Welt gibt es Register auf die wortweise zugegriffen werden muß während andere byteweisen Zugriff erlauben. Der Gerätetreiber ist nun so programmiert, daß wortweiser Zugriff immer dann stattfindet wenn die Anzahl der zu transportierenden Bytes gerade ist, ansonsten findet byteweiser Zugriff statt.

IP-Modul Kontrollprogramme

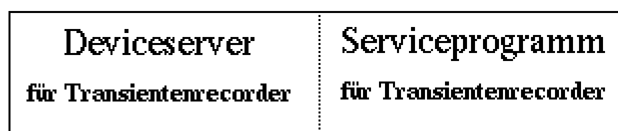
TRC2 Transientenrecorder

4.3.1.1 Struktur

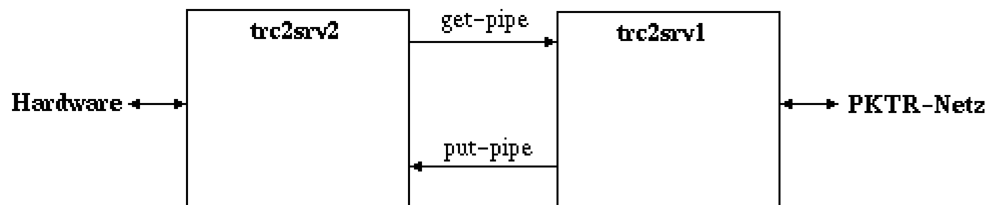
Die Software setzt sich aus mehreren Programmen zusammen. Da gibt es zunächst einmal den Deviceserver, der die Hardware steuert. Ein Serviceprogramm dient zur Steuerung dieses Deviceservers. Das ganze muss zusammenarbeiten mit dem HERA-Archivsystem. Das ergibt folgendes Bild:



Die Kommunikation zwischen den Programmen erfolgt über das PKTR-Netz. Dadurch können die einzelnen Programme sowohl auf unterschiedlichen Maschinen laufen als auch verschiedene Prozesse auf einer Maschine bilden. Im Falle des Serviceprogramms dürfte es häufig vorkommen, dass es sich zusammen mit dem Deviceserver auf einer Maschine befindet. Es ist sogar vorgesehen den Deviceserver und das Serviceprogramm zu einem Programm zusammen zu linken, ohne Anbindung an das PKTR-Netz.



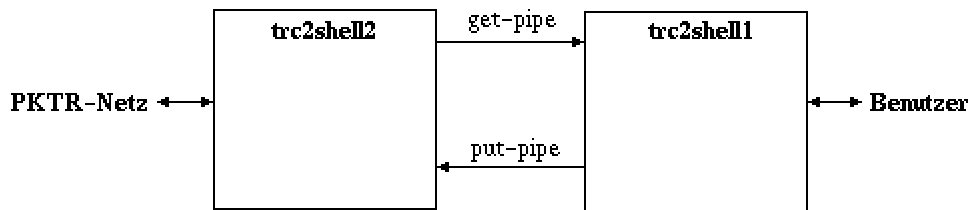
Die Modularität der Software wird durch Aufteilung der Programme auf mehrere Prozesse erreicht. Im Falle des Deviceservers ist das besonders wichtig, weil er als Bindeglied zwischen der Hardware auf der einen Seite und dem Archivsystem auf der anderen Seite steht. Die Hardware generiert Interrupts auf die der Deviceserver reagieren muss und das PKTR-Netz erwartet ebenfalls Reaktionen innerhalb gewisser Zeitspannen. Der Deviceserver besteht also aus zwei Prozessen dem Teil der das PKTR_Netz bedient und dem Teil der die Hardware bedient. Die Kommunikation zwischen den beiden Teilen geschieht über Pipes.



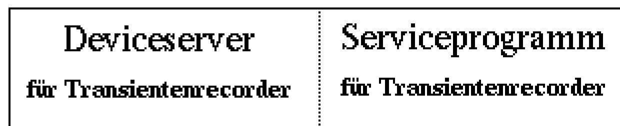
Das Programm startet mit dem Prozess **trc2srv1**. Dieser Prozess richtet die Pipes ein und erzeugt eine Semaphorgruppe. Anschliessend wird **trc2srv2**als Kindprozess gestartet.

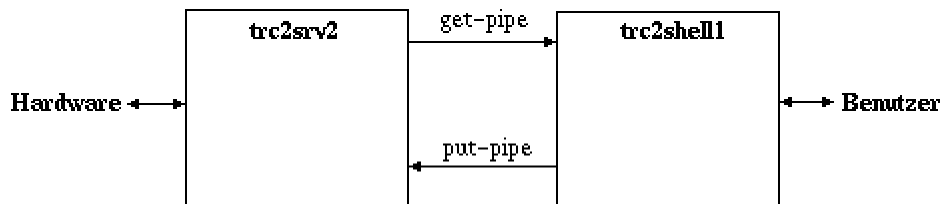
Die notwendigen Angaben zu den Pipes und Semaphoren werden als Parameter übergeben.

Das Serviceprogramm besteht ebenfalls aus zwei Prozessen, die mittels Pipes miteinander in Verbindung stehen. Der Programmstart erfolgt auf die gleiche Art und Weise wie beim Deviceserver. Der Prozess **trc2shell1** richtet direkt nach dem Start die Pipes und Semaphoren ein und startet anschliessend **trc2shell2** als Kindprozess, wobei die Angaben über Pipes und Semaphoren als Parameter an den Kindprozess übergeben werden.



Für den bereits erwähnten Fall, dass Deviceserver und Serviceprogramm nicht über das PKTR-Netz in Verbindung stehen sondern zu einem Prozess zusammen gelinkt sind ergibt sich folgende Struktur.

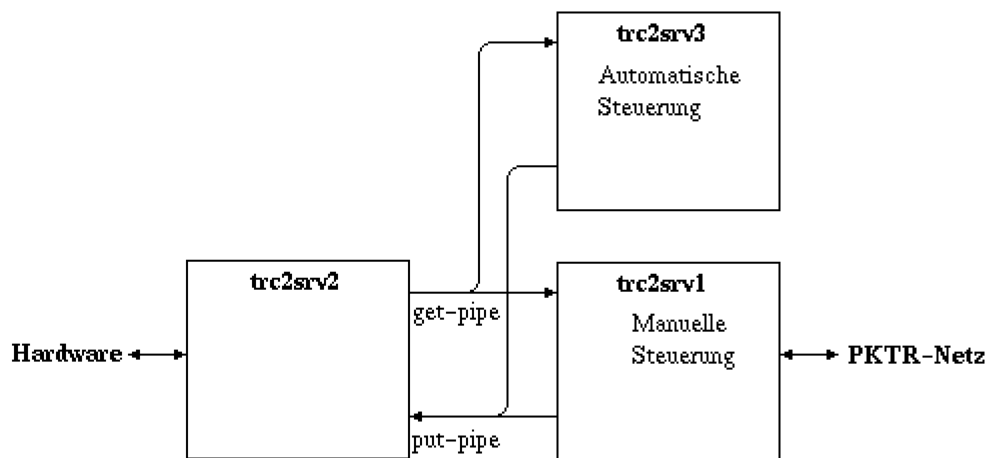




Da diese Programmversion völlig auf die PKTR-Netzanbindung verzichtet sind die Programmteile **trc2srv1** und **trc2shell2** überflüssig. Das Programm startet mit dem Prozess **trc2shell1**, der die Pipes und Semaphoren einrichtet und **trc2srv2** als Kindprozess startet.

Automatischer Serverbetrieb

Der Deviceserver hat die Aufgabe ständig Messwerte aufzuzeichnen. Beim Eintreten eines Ereignisses wird die Datennahme kurz unterbrochen um ein Speicherabbild der Messwerte anzulegen. Dieser automatische Programmablauf ist als Endlosschleife realisiert. Der erste Versuch, die automatische Steuerung in kleine Stücke zu zerteilen und zusammen mit der PKTR-Netz Software in einer grossen Schleife auszuführen, wie es früher bei MS-DOS FECs gemacht wurde, schlug fehl weil die PKTR-Netz Software die Programmausführung bis zu 20 Sekunden unterbrochen hat. Der nächste Versuch führte dann aber zum Erfolg. Die automatische Steuerung ist in einen separaten Prozess **trc2srv3** ausgelagert. Der Zugriff auf den Prozess **trc2srv2** erfolgt über dieselben Pipes die auch der Prozess **trc2srv1** benutzt.



Damit die gemeinsame Nutzung der Pipes keinen Datensalat ergibt muss man ein paar Vorkehrungen treffen. Die erste Massnahme sind Datensatzsperren. Wenn der Prozess **trc2srv1** oder **trc2srv3** ein Kommando an den Prozess **trc2srv2** geben will versucht er zuerst auf der put-pipe eine Schreibsperre einzurichten die es nur ihm alleine gestattet die Pipe zu beschreiben. Dann wird ein Kommando auf die put-pipe geschrieben. Der Prozess **trc2srv2** liest die put-pipe, bearbeitet das Kommando und schreibt das Ergebnis auf die get-pipe. Von dort wird es vom aufrufenden Prozess gelesen und die Schreibsperre wieder aufgehoben. Dieser Ablauf wiederholt sich für jedes Kommando.

4.3.1.2 Properties

Der trc2-Deviceserver ist mit fünf Properties für das PKTR-Netz ausgestattet.

- DATA
- HEADER
- DATAREADY
- SAVEDATA

- COMMAND

Die Properties erwarten unterschiedliche Parameter die im Folgenden erläutert werden.

DATA

- Eingangsdaten
  - keine (im Datenbereich). Der Kanalidentifikator wird mittels "devName" übertragen.
- Ausgangsdaten
  - Datenformat: CF_SHORT
  - Datengröße: 8192 Integerwerte (16-Bit)

Für die Identifikation des Kanals werden folgende vier Angaben benötigt.

- Gerätedatei (z.B. "/dev/pciip0")
- IP-Modul bzw. Slot (z.B. "A")
- Kanalnummer (z.B. "0")
- Bitnummer (z.B. "4")

Das aufrufende Programm setzt aus diesen vier Angaben und dem Servernamen einen String zusammen und übergibt ihn als Parameter "devName". Als Abtrennung zum Servernamen dient der Schrägstrich (`/`). Die Kanalangaben sind untereinander durch Punkte getrennt. Zahlenwerten ist die Raute vorangestellt (`#`). Beispiel:

```
TRC2SRV/#0.A.#0.#4
```

Die PKTR-Netz Software entfernt den Servernamen und den Schrägstrich (`TRC2SRV/`) und übergibt der Equipmentfunktion den Teilstring (`#0.A.#0.#4`) im Parameter "devName". Die Equipmentfunktion entfernt die Trenn- und Sonderzeichen (`.#`) und erhält so wieder die vier Angaben zur Kanalidentifikation.

Der Deviceserver schickt als Antwort die 8192 Messwerte als Rohdaten, d.h. so wie sie vom AD-Wandler kommen, allerdings umgewandelt in das Standarddatenformat "signed short integer". Diese Daten sind noch nicht mit einer physikalischen Einheit behaftet.

HEADER

- Eingangsdaten
  - keine (im Datenbereich). Der Kanalidentifikator wird mittels "devName" übertragen.
- Ausgangsdaten
  - Datenformat: CF_STRUCT
  - Datengröße: sizeof(LOCALHEADER)

Für die Eingangsdaten gilt dasselbe wie bei der Property DATA.

Der Deviceserver schickt als Antwort den lokalen Datenkopf der für die Auswertung der Meßwerte benötigt wird.

DATAREADY

- Eingangsdaten
    - keine
- Ausgangsdaten
    - Datenformat: CF_SHORT
    - Datengrösse: sizeof(short)

Mit einem Aufruf von DATAREADY kann man feststellen ob der Deviceserver Daten besitzt die noch nicht archiviert sind. Als Antwort dient ein short integer Wert. 1 bedeutet, es liegen Daten vor. -1 bedeutet, es liegen keine Daten vor.

SAVEDATA

- Eingangsdaten
    - keine
- Ausgangsdaten
    - keine

Beim Aufruf dieser Property wird lediglich ein Flag gesetzt. Daraufhin beendet der Deviceserver die Datennahme und macht die Daten versandfertig. Diese Property stellt somit eine von mehreren Möglichkeiten dar einen Stop bzw. Trigger auszulösen.

COMMAND

- Eingangsdaten
    - Datenformat: CF_TEXT
    - Datengröße: variabel
- Ausgangsdaten
    - Datenformat: CF_TEXT
    - Datengröße: variabel

Diese Property dient als Sammelbecken für eine ganze Reihe von Kommandos. Die Steuerung des Deviceservers kann in vollem Umfang über diese Property abgewickelt werden. Als Kommandos dienen die Funktionsnamen des C-Sourcecodes. Zur Übermittlung eines Kommandos wird eine Zeichenkette erstellt die den Funktionsnamen gefolgt von den Parametern enthält. Als Trennzeichen dient das Komma.

Kommando,Parameter 1,Parameter 2, ... Parameter n

Beispiel:

do_probe,/dev/pciip0,A,5,analog

Jetzt folgt eine Auflistung aller Kommandos mit den möglichen Parametern und Rückgabewerten

clock_shift	config	create_testdata
cy_post_reg	cy_sw_stop	dataready
delete_channel	delete_device	delete_slot
do_bandwidth	do_channel	do_channelname
do_clockshift	do_device	do_egu
do_eguhifactor	do_egulofactor	do_int_trg_div
do_interrupt0	do_interrupt1	do_ioSTOP
do_io_stop_syn	do_level	do_levelbit
do_mask	do_modebit	do_operator
do_post_trigger_cycles	do_probe	do_range
do_sampling_rate	do_slot	do_testvoltage
do_trigger_source	do_xor	get_bandwidth
get_bit_list	get_bit_parameter	get_ch_ipdata
get_channel	get_channel_list	get_channelname
get_channel_parameter	get_channel_pointer	get_device_parameter
get_device_pointer	get_io_stop_syn	get_clockshift
get_device	get_device_list	get_egu
get_eguhifactor	get_egulofactor	get_hardware_stop
get_interrupt0	get_interrupt1	get_ioSTOP
get_ioSTOP_input	get_ioTRIGGER_input	get_ipdata
get_level	get_levelbit	get_lheader
get_mask	get_mode	get_modebit
get_operator	get_post_trigger_cycles	get_probe
get_range	get_receiver_ready	get_sampling_rate
get_slot	get_slot_list	get_slot_parameter
get_slot_pointer	get_testvoltage	get_transmitter_ready

get_trigger_source	get_value	get_xor
intrpt_vec0	intrpt_vec1	level
mask	module_init	quit
read_control_word	read_general	read_init
rx_address	rx_dio_sel	rx_trigger
set_channel	set_device	set_slot
show_memory	start_all_datataking	start_datataking
status	stop_all_datataking	stop_datataking
tx_write	write_control_word	write_data_file
write_general	write_hexdata_file	write_init
xor		

Clockshift-Register schreiben *clock_shift*

clock_shift() schreibt ein 16-Bit Wort in das "clockshift"-Register.

Prototyp und Parameter

`clock_shift,device,slot,value`

`device`:	Name der Gerätedatei z.B. /dev/pciip0
`slot`:	Steckplatz des IP-Moduls (A,B,C oder D)
`value`:	16-Bit-Integer in hexadezimaler Darstellung.

Rückgabewert

"ok" bei fehlerfreier Ausführung, sonst "error".

Verweise

`do_clockshift(), get_clockshift()`

Config-Register schreiben *config*

`config()` schreibt ein 16-Bit Wort in das "config"-Register.

Prototyp und Parameter

`config,device,slot,value`

`device`:	Name der Gerätedatei z.B. /dev/pciip0
`slot`:	Steckplatz des IP-Moduls (A,B,C oder D)
`value`:	16-Bit-Integer in hexadezimaler Darstellung.

Rückgabewert

"ok" bei fehlerfreier Ausführung, sonst "error".

Verweise

`do_config(), get_config()`

Testdaten erzeugen *create_testdata*

`create_testdata()` erzeugt Testdaten für einen Kanal. Die Daten werden ohne Zuhilfenahme von Hardware generiert und dienen ausschließlich Servicezwecken.

Prototyp und Parameter

```
create_testdata,device,slot,channel
```

`device`:	Name der Gerätedatei z.B. /dev/pciip0
`slot`:	Steckplatz des IP-Moduls (A,B,C oder D)
`channel`:	Kanalnummer (0 bis 7)

Rückgabewert

"ok" bei fehlerfreier Ausführung, sonst "error".

Nachlauf einstellen *cy_post_reg*

`cy_post_reg()` setzt die Anzahl der Meßwerte die nach dem Ende der Datennahme noch aufgezeichnet werden sollen.

Prototyp und Parameter

```
cy_post_reg,device,slot,value
```

`device`:	Name der Gerätedatei z.B. /dev/pciip0
`slot`:	Steckplatz des IP-Moduls (A,B,C oder D)
`value`:	16-Bit-Integer in dezimaler Darstellung. Der Wert muß im Bereich von 0 bis 8191 liegen.

Rückgabewert

"ok" bei fehlerfreier Ausführung, sonst "error".

Verweise

`do_post_trigger_cycles(), get_post_trigger_cycles()`

Datennahme beenden *cy_sw_stop*

`cy_sw_stop()` stoppt die Aufzeichnung von Meßwerten.

Prototyp und Parameter

```
cy_sw_stop,device,slot
```

`device:`	Name der Gerätedatei z.B. /dev/pciip0
`slot:`	Steckplatz des IP-Moduls (A,B,C oder D)

Rückgabewert

"ok" bei fehlerfreier Ausführung, sonst "error".

Verweise

`stop_datataking()`

Archivierung von Daten auslösen *dataready*

`dataready()` teilt mit, ob Daten vorliegen die archiviert werden müssen.

Prototyp und Parameter

```
dataready, par
```

`par`	0

Rückgabewert

"ok" wenn Daten vorliegen die archiviert werden müssen, sonst "error".

Verweise

Kanal bzw. Tastkopf löschen *delete_channel*

`delete_channel()` löscht einen Kanal bzw. Tastkopf aus der Liste.

Prototyp und Parameter

```
delete_channel,device,slot,channel
```

`device:`	Name der Gerätedatei z.B. /dev/pciip0
`slot:`	Steckplatz des IP-Moduls (A,B,C oder D)
`channel:`	Kanalnummer (0 bis 7)

Rückgabewert

"ok" bei fehlerfreier Ausführung, sonst "error".

Verweise

`do_channel()`

Gerätedatei löschen *delete_device*

`delete_device()` löscht eine Gerätedatei aus der Liste. Es dürfen keine untergeordnete Datenstrukturen mehr vorhanden sein. Alle Kanäle und Slots dieser Gerätedatei müssen vorher gelöscht werden.

Prototyp und Parameter

`delete_device,device`

`device:`	Name der Gerätedatei z.B. /dev/pciip0

Rückgabewert

"ok" bei fehlerfreier Ausführung, sonst "error".

Verweise

`do_device()`

Slot bzw. IP-Modul löschen *delete_slot*

`delete_slot()` löscht einen Steckplatz für ein IP-Modul aus der Liste. Es dürfen keine untergeordnete Datenstrukturen mehr vorhanden sein. Alle Kanäle dieses Slots müssen vorher gelöscht werden.

Prototyp und Parameter

`delete_slot,device,slot`

`device:`	Name der Gerätedatei z.B. /dev/pciip0
`slot:`	Steckplatz des IP-Moduls (A,B,C oder D)

Rückgabewert

"ok" bei fehlerfreier Ausführung, sonst "error".

Verweise

`do_slot()`

Bandbreite einstellen *do_bandwidth*

`do_bandwidth()` stellt die Bandbreite eines analogen Tastkopfes ein.

Prototyp und Parameter

`do_bandwidth,device,slot,channel,value`

`device:`	Name der Gerätedatei z.B. /dev/pciip0
`slot:`	Steckplatz des IP-Moduls (A,B,C oder D)
`channel:`	Kanalnummer (0 bis 7)
`value:`	Maßzahl und Einheit. Folgende Angaben werden akzeptiert: 200kHz 100kHz 25kHz 10kHz 1kHz Die Angabe der Einheit ist nicht zwingend notwendig

Rückgabewert

"ok" bei fehlerfreier Ausführung, sonst "error".

Verweise

`get_bandwidth()`

Kanal hinzufügen *do_channel*

`do_channel()` legt einen Eintrag für einen neuen Kanal bzw. Tastkopf an.

Prototyp und Parameter

`do_channel,device,slot,channel`

`device:`	align="left"Name der Gerätedatei z.B. /dev/pciip0
`slot:`	Steckplatz des IP-Moduls (A,B,C oder D)
`channel:`	Kanalnummer (0 bis 7)

Rückgabewert

"ok" bei fehlerfreier Ausführung, sonst "error".

Verweise

`delete_channel()`

Logischen Kanalnamen setzen *do_channelname*

do_channelname() weist einem Kanal einen Namen zu. Jeder analoge Kanal kann einen Namen zugewiesen bekommen, bei digitalen Tastköpfen kann sogar jedes Bit einen eigenen Namen bekommen.

Prototyp und Parameter

`do_channelname,device,slot,channel,bit,value`

`device:`	Name der Gerätedatei z.B. /dev/pciip0

`slot:`	Steckplatz des IP-Moduls (A,B,C oder D)
`channel:`	Kanalnummer (0 bis 7)
`bit:`	Bitnummer, 0 bis 11 bei digitalen Tastköpfen, 0 bei analogen Tastköpfen.
`value:`	logischer Kanalname, maximal 31 Schriftzeichen lang.

Rückgabewert

"ok" bei fehlerfreier Ausführung, sonst "error".

Verweise

`get_channelname()`

clockshift setzen *do_clockshift*

`do_clockshift()` setzt die Zeitverzögerung für die seriellen Signale eines Kanales.

Prototyp und Paramete

`do_clockshift,device,slot,channel,value`

`device:`	Name der Gerätedatei z.B. /dev/pciip0
`slot:`	Steckplatz des IP-Moduls (A,B,C oder D)
`channel:`	Kanalnummer (0 bis 7)
`value:`	16-Bit Integer in dezimaler Darstellung.

Rückgabewert

"ok" bei fehlerfreier Ausführung, sonst "error".

Verweise

`get_clockshift()`

Gerätedatei hinzufügen *do_device*

`do_device()` legt einen Eintrag für eine neue Gerätedatei an.

Prototyp und Parameter

`do_device,device`

`device:`	Name der Gerätedatei z.B. /dev/pciip0

Rückgabewert

"ok" bei fehlerfreier Ausführung, sonst "error".

Verweise

`delete_device()`

Physikalische Einheit setzen *do_egu*

`do_egu()` weist einem Kanal eine physikalische Einheit zu. Jeder analoge Kanal kann eine physikalische Einheit zugewiesen bekommen, bei digitalen Tastköpfen kann sogar jedes Bit eine eigene physikalische Einheit bekommen. Der Kommandoname *egu* ist die Abkürzung der englischen Bezeichnung *engineering unit*. Der Deviceserver liefert die physikalische Einheit an das Archivsystem ab. Wenn die physikalische Einheit fehlt hat das keine gravierenden Auswirkungen. Betrachtungsprogramme können dann den Meßwerten halt keine Einheit zuweisen.

Prototyp und Parameter

`do_egu,device,slot,channel,bit,value`

device:	Name der Gerätedatei z.B. /dev/pciip0
slot:	Steckplatz des IP-Moduls (A,B,C oder D)
channel:	Kanalnummer (0 bis 7)
bit:	Bitnummer, 0 bis 11 bei digitalen Tastköpfen, 0 bei analogen Tastköpfen.
value:	physikalische Einheit, maximal 7 Schriftzeichen lang.

Rückgabewert

"ok" bei fehlerfreier Ausführung, sonst "error".

Verweise

`get_egu()`

Umrechnungsfaktor für positive Meßwerte *do_eguhifactor*

`do_eguhifactor()` weist einem Kanal einen Umrechnungsfaktor für positive Meßwerte zu. Jeder analoge Kanal kann einen Faktor zugewiesen bekommen, bei digitalen Tastköpfen kann sogar jedes Bit einen eigenen Faktor bekommen. Der Deviceserver selber benutzt den Umrechnungsfaktor nicht sondern liefert ihn lediglich an das Archivsystem ab. Wenn dieser Faktor fehlt kann das Betrachtungsprogramm die Meßergebnisse des Tastkopfes nicht in physikalische Werte umrechnen.

Prototyp und Parameter

`do_eguhifactor,device,slot,channel,bit,value`

`device:`	Name der Gerätedatei z.B. /dev/pciip0
`slot:`	Steckplatz des IP-Moduls (A,B,C oder D)
`channel:`	Kanalnummer (0 bis 7)
`bit:`	Bitnummer, 0 bis 11 bei digitalen Tastköpfen, 0 bei analogen Tastköpfen.
`value:`	Umrechnungsfaktor für negative Meßwerte. Die Software wandelt

diesen Wert in eine Fließkommazahl des Datenformats *double* um.

Rückgabewert

"ok" bei fehlerfreier Ausführung, sonst "error".

Verweise

`get_eguhifactor()`

Umrechnungsfaktor für negative Meßwerte *do_egulofactor*

`do_egulofactor()` weist einem Kanal einen Umrechnungsfaktor für negative Meßwerte zu. Jeder analoge Kanal kann einen Faktor zugewiesen bekommen, bei digitalen Tastköpfen kann sogar jedes Bit einen eigenen Faktor bekommen. Der Deviceserver selber benutzt den Umrechnungsfaktor nicht sondern liefert ihn lediglich an das Archivsystem ab. Wenn dieser Faktor fehlt kann das Betrachtungsprogramm die Meßergebnisse des Tastkopfes nicht in physikalische Werte umrechnen.

Prototyp und Parameter

`do_egulofactor,device,slot,channel,bit,value`

`device:`	Name der Gerätedatei z.B. /dev/pciip0
`slot:`	Steckplatz des IP-Moduls (A,B,C oder D)
`channel:`	Kanalnummer (0 bis 7)
`bit:`	Bitnummer, 0 bis 11 bei digitalen Tastköpfen, 0 bei analogen Tastköpfen.
`value:`	Umrechnungsfaktor für negative Meßwerte. Die Software wandelt diesen Wert in eine Fließkommazahl des Datenformats *double* um.

Rückgabewert

"ok" bei fehlerfreier Ausführung, sonst "error".

Verweise

`get_egulofactor()`

Unterteilung des internen Triggers ermitteln *do_int_trg_div*

`do_int_trg_div()` ermittelt die Unterteilung des internen Triggers anhand der gewünschten Abtastfrequenz (sampling_rate). Das Ergebnis kann direkt in das Register *int_trg_div* geschrieben werden. Die Funktion stellt den internen Trigger so ein, dass die gewünschte Abtastfrequenz auf jeden Fall erreicht wird. Ausserdem wird die tatsächliche Abtastfrequenz berechnet und anstelle der gewünschten gespeichert.

Prototyp und Parameter

`do_int_trg_div,device,slot`

`device:`	Name der Gerätedatei z.B. /dev/pciip0
`slot:`	Steckplatz des IP-Moduls (A,B,C oder D)

Rückgabewert

Unterteilung des Triggers bei fehlerfreier Ausführung (positiver Wert), sonst -1.

Verweise

`do_sampling_rate(), get_sampling_rate()`

Interrupt 0 aktivieren *do_interrupt0*

`do_interrupt0()` aktiviert oder deaktiviert die Interruptbehandlung des Interrupt0 des IP-Moduls.

Prototyp und Parameter

`do_interrupt0,device,slot,value`

`device:`	Name der Gerätedatei z.B. /dev/pciip0
`slot:`	Steckplatz des IP-Moduls (A,B,C oder D)
`value:`	gewünschter Zustand der Interruptbehandlung. Folgende Angaben werden akzeptiert: • enable • disable

Rückgabewert

"ok" bei fehlerfreier Ausführung, sonst "error".

Verweise

`get_interrupt0()`

Interrupt 1 aktivieren *do_interrupt1*

`do_interrupt1()` aktiviert oder deaktiviert die Interruptbehandlung des Interrupt1 des IP-Moduls.

Prototyp und Parameter

`do_interrupt1,device,slot,value`

`device:`	Name der Gerätedatei z.B. /dev/pciip0
`slot:`	Steckplatz des IP-Moduls (A,B,C oder D)

`value:`	Gewünschter Zustand der Interruptbehandlung. Folgende Angaben werden akzeptiert: • enable • disable

Rückgabewert

"ok" bei fehlerfreier Ausführung, sonst "error".

Verweise

`get_interrupt1()`

ioSTOP Signal Freigabe *do_ioSTOP*

`do_ioSTOP()` legt fest ob ein Signal am Eingang ioSTOP ausgewertet wird oder nicht.

Prototyp und Parameter

`do_ioSTOP,device,slot,value`

`device:`	Name der Gerätedatei z.B. /dev/pciip0
`slot:`	Steckplatz des IP-Moduls (A,B,C oder D)
`value:`	ioSTOP Eingang freigeben. Folgende Angaben werden akzeptiert: • enable • disable

Rückgabewert

"ok" bei fehlerfreier Ausführung, sonst "error".

Verweise

Stopsynchronisation ein/ausschalten *do_io_stop_syn*

`do_io_stop_syn()` legt fest ob eine erkannte Stopbedingung auf einen Ausgangspin gegeben wird um anderen TRC2-IP-Modulen zur Verfügung zu stehen. Das kommt insbesondere dann in Betracht wenn ein TRC2-IP-Modul die Stopbedingung aus den Meßwerten generiert.

Prototyp und Parameter

`do_io_stop_syn,device,slot,value`

`device:`	Name der Gerätedatei z.B. /dev/pciip0
`slot:`	Steckplatz des IP-Moduls (A,B,C oder D)

	Stopbedingung ausgeben. Folgende Angaben werden akzeptiert:
*val*ue:	• enable • disable

Rückgabewert

"ok" bei fehlerfreier Ausführung, sonst "error".

Verweise

`get_io_stop_syn()`

Level setzen — *do_level*

`do_level()` setzt den Operanden *level* auf den übergebenen Wert.

Prototyp und Parameter

`do_level,device,slot,channel,value`

device:	Name der Gerätedatei z.B. /dev/pciip0
slot:	Steckplatz des IP-Moduls (A,B,C oder D)
channel:	Kanalnummer (0 bis 7)
value:	16-Bit Integer. Wenn die Angabe mit *0x* beginnt wird hexadezimale Darstellung angenommen sonst wird dezimale Darstellung vorausgesetzt.

Rückgabewert

"ok" bei fehlerfreier Ausführung, sonst "error".

Verweise

`do_mask(), do_xor(), do_operator(), get_mask(), get_xor(), get_operator(), get_level()`

Meßbereich einstellen — *do_levelbit*

`do_levelbit()` stellt den Meßbereich eines digitalen Tastkopfes ein.

Prototyp und Parameter

`do_levelbit,device,slot,channel,bit,value`

device:	Name der Gerätedatei z.B. /dev/pciip0
slot:	Steckplatz des IP-Moduls (A,B,C oder D)
channel:	Kanalnummer (0 bis 7)
bit:	Bitnummer (0 bis 11)

`value:`	Maßzahl und Einheit. Folgende Angaben werden akzeptiert: • 24V • 5V Die Angabe der Einheit ist nicht zwingend notwendig

Rückgabewert

"ok" bei fehlerfreier Ausführung, sonst "error".

Verweise

`get_levelbit()`

Mask setzen *do_mask*

`do_mask()` setzt den Operanden *mask* Abbruchbedingung auf den übergebenen Wert.

Prototyp und Parameter

`do_mask,device,slot,channel,value`

`device:`	align="left"Name der Gerätedatei z.B. /dev/pciip0
`slot:`	Steckplatz des IP-Moduls (A,B,C oder D)
`channel:`	Kanalnummer (0 bis 7)
`value:`	16-Bit Integer. Wenn die Angabe mit *0x* beginnt wird hexadezimale Darstellung angenommen sonst wird dezimale Darstellung vorausgesetzt.

Rückgabewert

"ok" bei fehlerfreier Ausführung, sonst "error".

Verweise

`do_xor(), do_operator(), do_level(), get_mask(), get_xor(), get_operator(), get_level()`

Betriebsmodus einstellen *do_modebit*

`do_modebit()` stellt den Betriebsmodus eines digitalen Tastkopfes ein. Zwei Betriebsmodi stehen zur Wahl:

1. Betriebsmodus ***edge:*** Der Tastkopf liefert ein Highsignal wenn seit dem letzten Meßvorgang das Eingangssignal seinen Zustand geändert hat, also eine Flanke aufgetreten ist. Wenn das Eingangssignal unverändert geblieben ist liefert der Tastkopf ein Lowsignal.

2. Betriebsmodus ***sample:*** Der Tastkopf liefert ein Highsignal wenn das Eingangssignal zum Zeitpunkt der Abtastung Highpegel führt und ein Lowsignal wenn das Eingangssignal zum Abtastzeitpunkt Lowpegel führt.

Prototyp und Parameter

`do_modebit,device,slot,channel,bit,value`

`device:`	Name der Gerätedatei z.B. /dev/pciip0
`slot:`	Steckplatz des IP-Moduls (A,B,C oder D)
`channel:`	Kanalnummer (0 bis 7)
`bit:`	Bitnummer (0 bis 11)
`value:`	Betriebsmodus. Folgende Angaben werden akzeptiert: • edge • sample

Rückgabewert

"ok" bei fehlerfreier Ausführung, sonst "error".

Verweise

get_modebit()

Operator setzen *do_operator*

`do_operator()` setzt den Vergleichsoperator der Abbruchbedingung.

Prototyp und Parameter

do_operator,device,slot,channel.value

`device:`	Name der Gerätedatei z.B. /dev/pciip0
`slot:`	Steckplatz des IP-Moduls (A,B,C oder D)
`channel:`	Kanalnummer (0 bis 7)
value:	Vergleichsoperator (=,<,,=,<=,!=,DISABLE)

Rückgabewert

"ok" bei fehlerfreier Ausführung, sonst "error".

Verweise

`do_mask(), do_xor(), do_level(), get_mask(), get_xor(), get_operator(), get_level()`

Nachlauf einstellen do_post_trigger_cycles

`do_post_trigger_cycles()` stellt die Anzahl der Meßwerte ein, die nach dem Ende des Meßbetriebs noch aufgezeichnet werden sollen.

Prototyp und Parameter

`do_post_trigger_cycles,device,slot,value`

`device:`	Name der Gerätedatei z.B. /dev/pciip0
`slot:`	Steckplatz des IP-Moduls (A,B,C oder D)
`value:`	Anzahl der Meßwerte die nach dem Ende der Messung noch aufgezeichnet werden sollen. Es muß eine Zahl zwischen 0 und 8191 sein.

Rückgabewert

"ok" bei fehlerfreier Ausführung, sonst "error".

Verweise

`get_post_trigger_cycles()`

Art des Tastkopfes festlegen *do_probe*

`do_probe()` spezifiziert die Art des Tastkopfes für einen Kanal.

Prototyp und Parameter

`do_probe,device,slot,channel,value`

`device:`	Name der Gerätedatei z.B. /dev/pciip0
`slot:`	Steckplatz des IP-Moduls (A,B,C oder D)
`channel:`	Kanalnummer (0 bis 7)
`value:`	Art des Tastkopfes (analog oder digital)

Rückgabewert

"ok" bei fehlerfreier Ausführung, sonst "error".

Verweise

`get_probe()`

Meßbereich einstellen *do_range*

`do_range()` stellt den Meßbereich eines analogen Tastkopfes ein.

Prototyp und Parameter

`do_range,device,slot,channel,value`

`device:`	Name der Gerätedatei z.B. /dev/pciip0
`slot:`	Steckplatz des IP-Moduls (A,B,C oder D)
`channel:`	Kanalnummer (0 bis 7)
`value:`	Maßzahl und Einheit. Folgende Angaben werden akzeptiert: • 30V • 10V • 1V • 0.1V • 100mV Die Angabe der Einheit ist nicht zwingend notwendig

Rückgabewert

"ok" bei fehlerfreier Ausführung, sonst "error".

Verweise

`get_range()`

Abtastrate einstellen — *do_sampling_rate*

`do_sampling_rate()` stellt die Abtastrate für ein IP-Modul ein. Allerdings hat das keine Auswirkung auf den Meßbetrieb sondern dient nur zur Information. Die Abtastrate mit der gemessen wird kann man mit diesem Kommando nicht ändern, dennoch ist dieses Kommando sehr wichtig. Die eingestellte Zahl wird an das Archivsystem weitergegeben und zusammen mit den Meßwerten archiviert. Wenn diese Zahl falsch ist oder fehlt, kann man später, bei einer Auswertung der Daten, keine Aussage über die Zeitabstände zwischen den Meßwerten machen.

Prototyp und Parameter

`do_sampling_rate,device,slot,value`

`device:`	Name der Gerätedatei z.B. /dev/pciip0
`slot:`	Steckplatz des IP-Moduls (A,B,C oder D)
`value:`	Abtastrate in Hz. Die Zahl wird programmintern in eine Fließkommazahl des Typs *double* umgewandelt. Die Angabe der Einheit ist nicht zwingend notwendig.

Rückgabewert

"ok" bei fehlerfreier Ausführung, sonst "error".

Verweise

`get_sampling_rate()`

Slot bzw. IP-Modul hinzufügen *do_slot*

`do_slot()` legt einen Eintrag für ein neues IP-Modul bzw. Slot an.

Prototyp und Parameter

`do_slot,device,slot`

`device:`	Name der Gerätedatei z.B. /dev/pciip0
`slot:`	Steckplatz des IP-Moduls (A,B,C oder D)

Rückgabewert

"ok" bei fehlerfreier Ausführung, sonst "error".

Verweise

`delete_slot()`

Testspannung ein- oder ausschalten *do_testvoltage*

`do_testvoltage()` schaltet die Testspannung eines analogen Tastkopfes ein- oder aus.

Prototyp und Parameter

`do_testvoltage,device,slot,channel,value`

`device:`	Name der Gerätedatei z.B. /dev/pciip0
`slot:`	Steckplatz des IP-Moduls (A,B,C oder D)
`channel:`	Kanalnummer (0 bis 7)
`value:`	gewünschter Zustand der Testspannung. Folgende Angaben werden akzeptiert: • on • off

Rückgabewert

"ok" bei fehlerfreier Ausführung, sonst "error".

Verweise

`get_testvoltage()`

Triggerquelle einstellen *do_trigger_source*

`do_trigger_source()` legt fest, ob das IP-Modul mit internen oder externen Trigger arbeitet.

Prototyp und Parameter

`do_trigger_source,device,slot,value`

`device:`	Name der Gerätedatei z.B. /dev/pciip0
`slot:`	Steckplatz des IP-Moduls (A,B,C oder D)
`value:`	Triggerquelle. Folgende Angaben werden akzeptiert: • intern • extern

Rückgabewert

"ok" bei fehlerfreier Ausführung, sonst "error".

Verweise

`get_trigger_source()`

XOR setzen *do_xor*

`do_xor()` setzt den Operanden *xor* auf den übergebenen Wert.

Prototyp und Parameter

`do_xor,device,slot,channel,value`

`device:`	Name der Gerätedatei z.B. /dev/pciip0
`slot:`	Steckplatz des IP-Moduls (A,B,C oder D)
`channel:`	Kanalnummer (0 bis 7)
`value:`	16-Bit Integer. Wenn die Angabe mit *0x* beginnt wird hexadezimale Darstellung angenommen sonst wird dezimale Darstellung vorausgesetzt.

Rückgabewert

"ok" bei fehlerfreier Ausführung, sonst "error".

Verweise

`do_mask(), do_operator(), do_level(), get_mask(), get_xor(),`
`get_operator(), get_level()`

Bandbreite eines analogen Tastkopfes ermitteln *get_bandwidth*

`get_bandwidth()` ermittelt die eingestellte Bandbreite eines analogen Tastkopfes.

Prototyp und Parameter

`get_bandwidth,device,slot,channel`

`device:`	Name der Gerätedatei z.B. /dev/pciip0
`slot:`	Steckplatz des IP-Moduls (A,B,C oder D)
`channel:`	Kanalnummer (0 bis 7)

Rückgabewert

Bandbreite in dezimaler Darstellung bei fehlerfreier Ausführung, sonst "channel not found".

Verweise

`do_bandwidth()`

get_bit_list

`get_bit_list()`

Prototyp und Parameter

`get_bit_list,device,slot,channel`

`device:`	Name der Gerätedatei z.B. /dev/pciip0
`slot:`	Steckplatz des IP-Moduls (A,B,C oder D)
`channel:`	Kanalnummer (0 bis 7)

Rückgabewert

`"channel not found".`

get_bit_parameter

`get_bit_parameter()`

Prototyp und Parameter

`get_bit_parameter,device,slot,channel,bit`

`device:`	Name der Gerätedatei z.B. /dev/pciip0
`slot:`	Steckplatz des IP-Moduls (A,B,C oder D)
`channel:`	Kanalnummer (0 bis 7)
`bit:`	Bitnummer, 0 bis 11 bei digitalen Tastköpfen, 0 bei analogen Tastköpfen.

Rückgabewert

`"channel not found".`

Verweise

`do_channelname(), get_channelname(), do_egu(), get_egu(), do_egulofactor(), get_egulofactor(), do_eguhifactor(), get_eguhifactor()`

Daten aus IP-Modul auslesen *get_ch_ipdata*

`get_ch_ipdata()` liest bei jedem Aufruf die Daten eines Kanals aus. Die Funktion muss für jeden Kanal einmal aufgerufen werden. Die Weiterschaltung der Kanäle geschieht automatisch innerhalb der Funktion. Nach Bearbeitung des letzten Kanals wird "error" zurückgegeben. Ein erneuter Aufruf bearbeitet dann wieder den ersten Kanal.

Prototyp und Parameter

```
get_ch_ipdata
```

Rückgabewert

"ok" wenn noch weitere Kanäle existieren die noch nicht ausgelesen wurden. "error" wenn alle Kanäle bearbeitet wurden.

Verweise

`get_ipdata()`

Kanalnummer ermitteln *get_channel*

`get_channel()` ermittelt die Kanalnummer.

Prototyp und Parameter

```
get_channel,device,slot,channel
```

`device:`	Name der Gerätedatei z.B. /dev/pciip0
`slot:`	Steckplatz des IP-Moduls (A,B,C oder D)
`channel:`	Kanalnummer (0 bis 7)

Rückgabewert

Kanalnummer bei fehlerfreier Ausführung, sonst "error".

get_channel_list

`get_channel_list()`

Prototyp und Parameter

```
get_channel_list,device,slot
```

`device:`	Name der Gerätedatei z.B. /dev/pciip0

`slot:`	Steckplatz des IP-Moduls (A,B,C oder D)

Rückgabewert

"channel not found".

Verweise

`do_channel(), delete_channel()`

Logischen Kanalnamen ermitteln *get_channelname*

`get_channelname()` ermittelt den logischen Namen eines Kanals.

Prototyp und Parameter

`get_channelname,device,slot,channel,bit`

`device:`	Name der Gerätedatei z.B. /dev/pciip0
`slot:`	Steckplatz des IP-Moduls (A,B,C oder D)
`channel:`	Kanalnummer (0 bis 7)
`bit:`	Bitnummer, 0 bis 11 bei digitalen Tastköpfen, 0 bei analogen Tastköpfen.

Rückgabewert

Logischer Kanalname bei fehlerfreier Ausführung, sonst "channel not found".

Verweise

`do_channelname()`

get_channel_parameter

`get_channel_parameter()`

Prototyp und Parameter

`get_channel_parameter,device,slot,channel`

`device:`	Name der Gerätedatei z.B. /dev/pciip0
`slot:`	Steckplatz des IP-Moduls (A,B,C oder D)
`channel:`	Kanalnummer (0 bis 7)

Rückgabewert

"channel not found".

Verweise

`do_channel(), delete_channel(), do_channelname(), get_channelname(), do_egu(), get_egu(), do_egulofactor(), get_egulofactor(), do_eguhifactor(), get_eguhifactor(), do_probe(), get_probe()`

get_channel_pointer

`get_channel_pointer()`

Prototyp und Parameter

`get_channel_pointer,device,slot,channel,bit`

`device:`	Name der Gerätedatei z.B. /dev/pciip0
`slot:`	Steckplatz des IP-Moduls (A,B,C oder D)
`channel:`	Kanalnummer (0 bis 7)

Rückgabewert

"channel not found".

get_device_parameter

`get_device_parameter()`

Prototyp und Parameter

get_device_parameter,device

`device:`	Name der Gerätedatei z.B. /dev/pciip0

Rückgabewert

"channel not found".

Existenz einer Gerätedatei prüfen — *get_device_pointer*

`get_device_pointer()` durchsucht den Speicher nach der genannten Gerätedatei.

Prototyp und Parameter

`get_device_pointer,device`

`device:`	Name der Gerätedatei z.B. /dev/pciip0

Rückgabewert

"ok" bei fehlerfreier Ausführung, sonst "error".

get_io_stop_syn

`get_io_stop_syn()` ermittelt ob die Stopbedingung ausgegeben wird.

Prototyp und Parameter

`get_io_stop_syn,device,slot`

`device:`	Name der Gerätedatei z.B. /dev/pciip0
`slot:`	Steckplatz des IP-Moduls (A,B,C oder D)

Rückgabewert

"enable" oder "disable" bei fehlerfreier Ausführung, sonst "slot not found".

Verweise

`do_io_stop_syn()`

clockshift abfragen *get_clockshift*

`get_clockshift()` die eingestellte Zeitverzögerung für einen Kanal.

Prototyp und Parameter

`get_clockshift,device,slot,channel`

`device:`	Name der Gerätedatei z.B. /dev/pciip0
`slot:`	Steckplatz des IP-Moduls (A,B,C oder D)
`channel:`	Kanalnummer (0 bis 7)

Rückgabewert

Zeitverzögerung in dezimaler Darstellung in Nanosekunden bei fehlerfreier Ausführung, sonst "channel not found".

Verweise

`do_clockshift()`

Gerätedatei ermitteln *get_device*

`get_device()` ermittelt den Namen der Gerätedatei.

Prototyp und Parameter

get_device,device

`device:`	Name der Gerätedatei z.B. /dev/pciip0

Rückgabewert

Name der Gerätedatei bei fehlerfreier Ausführung, sonst "error".

get_device_list

`get_device_list()`

Prototyp und Parameter

get_device_list

Rückgabewert

"channel not found".

Verweise

`do_device(), delete_device()`

Physikalische Einheit ermitteln *get_egu*

`get_egu()`

Prototyp und Parameter

`get_egu,device,slot,channel,bit`

`device:`	Name der Gerätedatei z.B. /dev/pciip0
`slot:`	Steckplatz des IP-Moduls (A,B,C oder D)
`channel:`	Kanalnummer (0 bis 7)
`bit:`	Bitnummer, 0 bis 11 bei digitalen Tastköpfen, 0 bei analogen Tastköpfen.

Rückgabewert

"channel not found".

Verweise

`do_egu()`

get_eguhifactor

`get_eguhifactor()`

Prototyp und Parameter

`get_eguhifactor,device,slot,channel,bit`

`device:`	Name der Gerätedatei z.B. /dev/pciip0
`slot:`	Steckplatz des IP-Moduls (A,B,C oder D)
`channel:`	Kanalnummer (0 bis 7)
`bit:`	Bitnummer, 0 bis 11 bei digitalen Tastköpfen, 0 bei analogen Tastköpfen.

Rückgabewert

"channel not found".

Verweise

`do_eguhifactor()`

get_egulofactor

`get_egulofactor()`

Prototyp und Parameter

`get_egulofactor,device,slot,channel,bit`

`device:`	Name der Gerätedatei z.B. /dev/pciip0
`slot:`	Steckplatz des IP-Moduls (A,B,C oder D)
`channel:`	Kanalnummer (0 bis 7)
`bit:`	Bitnummer, 0 bis 11 bei digitalen Tastköpfen, 0 bei analogen Tastköpfen.

Rückgabewert

"channel not found".

Verweise

`do_egulofactor()`

Hardwarestop ermitteln *get_hardware_stop*

`get_hardware_stop()` stellt fest ob ein IP-Modul den Betriebsmodus von DT (data taking) nach DR (data read out) gewechselt hat.

Prototyp

`get_hardware_stop`

Rückgabewert

"ok" wenn ein IP-Modul gestoppt ist, sonst "error".

Verweise

Interrupt0 Freigabe ermitteln *get_interrupt0*

`get_interrupt0()` ermittelt die Freigabe des Interrupt0.

Prototyp und Parameter

`get_interrupt0,device,slot`

`device:`	Name der Gerätedatei z.B. /dev/pciip0

slot: Steckplatz des IP-Moduls (A,B,C oder D)

Rückgabewert

"enable" oder "disable" bei fehlerfreier Ausführung, sonst "slot not found".

Verweise

`do_interrupt0(), intrpt_vec0()`

Interrupt1 Freigabe ermitteln *get_interrupt1*

`get_interrupt1()` ermittelt die Freigabe des Interrupt0.

Prototyp und Parameter

`get_interrupt1,device,slot`

device: Name der Gerätedatei z.B. /dev/pciip0

slot: Steckplatz des IP-Moduls (A,B,C oder D)

Rückgabewert

"enable" oder "disable" bei fehlerfreier Ausführung, sonst "slot not found".

Verweise

`do_interrupt1(), intrpt_vec1()`

get_ioSTOP

`get_ioSTOP`

Prototyp und Parameter

`get_ioSTOP,device,slot`

device: Name der Gerätedatei z.B. /dev/pciip0

slot: Steckplatz des IP-Moduls (A,B,C oder D)

Rückgabewert

"enable", "disable" bei fehlerfreier Ausführung, sonst "slot not found".

==
=================== Nachbesserung! ===============
==
=========

Verweise

Zustand des ioSTOP Eingangs ermitteln *get_ioSTOP_input*

`get_ioSTOP_input` ermittelt den Zustand des ioSTOP Eingangs. Dazu wird das "status"-Register ausgelesen und Bit 3 ausgewertet.

Prototyp und Parameter

`get_ioSTOP_input,device,slot`

`device:`	Name der Gerätedatei z.B. /dev/pciip0
`slot:`	Steckplatz des IP-Moduls (A,B,C oder D)

Rückgabewert

"ok" bei fehlerfreier Ausführung, sonst "error".

=== Nachbesserung! ================
===

Verweise

`status()`

Zustand des ioTRIGGER Eingangs ermitteln *get_ioTRIGGER_input*

`get_ioTRIGGER_input()` ermittelt den Zustand des ioTRIGGER Eingangs. Dazu wird das "status"-Register ausgelesen und Bit 2 ausgewertet.

Prototyp und Parameter

`get_ioTRIGGER_input,device,slot`

`device:`	Name der Gerätedatei z.B. /dev/pciip0
`slot:`	Steckplatz des IP-Moduls (A,B,C oder D)

Rückgabewert

"ok" bei fehlerfreier Ausführung, sonst "error".

=== Nachbesserung! ================
===

Verweise

`status()`

IP-Speicher auslesen *get_ipdata*

`get_ipdata()` kopiert die Daten aus dem Memorybereich des TRC2-IP-Moduls in dem Hauptspeicher des Deviceservers.

Prototyp und Parameter

get_ipdata,device,slot,channel,bit

`device:`	Name der Gerätedatei z.B. /dev/pciip0
`slot:`	Steckplatz des IP-Moduls (A,B,C oder D)
`channel:`	Kanalnummer (0 bis 7)
`bit:`	Bitnummer, 0 bis 11 bei digitalen Tastköpfen, 0 bei analogen Tastköpfen.

Rückgabewert

"ok" bei fehlerfreier Ausführung, sonst "error".

Verweise

`rx_address(), get_ch_ipdata()`

Level ermitteln *get_level*

`get_level()` ermittelt den Wert "level" der Stopbedingung.

Prototyp und Parameter

`get_level,device,slot,channel`

`device:`	Name der Gerätedatei z.B. /dev/pciip0
`slot:`	Steckplatz des IP-Moduls (A,B,C oder D)
`channel:`	Kanalnummer (0 bis 7)

Rückgabewert

Größe des Wertes "level" bei fehlerfreier Ausführung, sonst "channel not found".

Verweise

`do_level()`

Meßbereich eines digitalen Tastkopfes ermitteln *get_levelbit*

`get_levelbit()` ermittelt den Meßbereich eines digitalen Tastkopfes.

Prototyp und Parameter

`get_levelbit,device,slot,channel,bit`

`device:`	Name der Gerätedatei z.B. /dev/pciip0
`slot:`	Steckplatz des IP-Moduls (A,B,C oder D)
`channel:`	Kanalnummer (0 bis 7)
`bit:`	Bitnummer, 0 bis 11 bei digitalen Tastköpfen, 0 bei analogen Tastköpfen.

Rückgabewert

Meßbereich in dezimaler Darstellung in Nanosekunden bei fehlerfreier Ausführung, sonst "channel not found".

Verweise

`do_levelbit()`

get_lheader

`get_lheader()`

Prototyp und Parameter

`get_lheader,device,slot,channel,bit`

`device:`	Name der Gerätedatei z.B. /dev/pciip0
`slot:`	Steckplatz des IP-Moduls (A,B,C oder D)
`channel:`	Kanalnummer (0 bis 7)
`bit:`	Bitnummer, 0 bis 11 bei digitalen Tastköpfen, 0 bei analogen Tastköpfen.

Rückgabewert

"channel not found".

Verweise

`get_ipdata()`

Mask ermitteln *get_mask*

`get_mask()` ermittelt den Wert "mask" der Stopbedingung.

Prototyp und Parameter

`get_mask,device,slot,channel`

`device:`	Name der Gerätedatei z.B. /dev/pciip0
`slot:`	Steckplatz des IP-Moduls (A,B,C oder D)
`channel:`	Kanalnummer (0 bis 7)

Rückgabewert

Größe des Wertes "mask" bei fehlerfreier Ausführung, sonst "channel not found".

Verweise

`do_mask()`

Betriebsmodus ermitteln *get_mode*

`get_mode()` ermittelt den Betriebsmodus eines TRC2-IP-Moduls. Dazu wird das "status"-Register ausgelesen und die Bits 6 und 7 ausgewertet.

Prototyp und Parameter

`get_mode,device,slot`

`device:`	Name der Gerätedatei z.B. /dev/pciip0
`slot:`	Steckplatz des IP-Moduls (A,B,C oder D)

Rückgabewert

"ok" bei fehlerfreier Ausführung, sonst "error".

==
================== Nachbesserung! ===============
==
=========

Verweise

`status()`

Betriebsart eines digitalen Tastkopfes ermitteln *get_modebit*

`get_modebit()` ermittelt die Betriebsart eines digitalen Tastkopfes.

Prototyp und Parameter

`get_modebit,device,slot,channel,bit`

`device:`	Name der Gerätedatei z.B. /dev/pciip0
`slot:`	Steckplatz des IP-Moduls (A,B,C oder D)
`channel:`	Kanalnummer (0 bis 7)
`bit:`	Bitnummer, 0 bis 11 bei digitalen Tastköpfen, 0 bei analogen Tastköpfen.

Rückgabewert

Betriebsart bei fehlerfreier Ausführung, sonst "channel not found".

Verweise

`do_modebit()`

Operator ermitteln *get_operator*

`get_operator()` ermittelt den Wert "config" der Stopbedingung.

Prototyp und Parameter

`get_operator,device,slot,channel`

`device:`	Name der Gerätedatei z.B. /dev/pciip0
`slot:`	Steckplatz des IP-Moduls (A,B,C oder D)
`channel:`	Kanalnummer (0 bis 7)

Rückgabewert

"=","<","","=","<=","!=","DISABLE" bei fehlerfreier Ausführung, sonst "channel not found".

Verweise

`do_operator()`

Nachlauf ermitteln get_post_trigger_cycles

`get_post_trigger_cycles()` ermittelt den Nachlauf eines Kanals.

Prototyp und Parameter

`get_post_trigger_cycles,device,slot`

`device:`	Name der Gerätedatei z.B. /dev/pciip0
`slot:`	Steckplatz des IP-Moduls (A,B,C oder D)

Rückgabewert

Nachlauf in dezimaler Darstellung bei fehlerfreier Ausführung, sonst "channel not found".

Verweise

`cy_post_reg(), do_post_trigger_cycles()`

Art des Tastkopfes ermitteln *get_probe*

`get_probe()` ermittelt die Art des Tastkopfes.

Prototyp und Parameter

`get_probe,device,slot,channel`

`device:`	Name der Gerätedatei z.B. /dev/pciip0
`slot:`	Steckplatz des IP-Moduls (A,B,C oder D)
`channel:`	Kanalnummer (0 bis 7)

Rückgabewert

Art des Tastkopfes bei fehlerfreier Ausführung, sonst "error".

Verweise

`do_probe()`

Meßbereich eines analogen Tastkopfes ermitteln *get_range*

`get_range()` ermittelt den Meßbereich eines analogen Tastkopfes.

Prototyp und Parameter

get_range,device,slot,channel

`device:`	Name der Gerätedatei z.B. /dev/pciip0
`slot:`	Steckplatz des IP-Moduls (A,B,C oder D)
`channel:`	Kanalnummer (0 bis 7)

Rückgabewert

Meßbereich in dezimaler Darstellung bei fehlerfreier Ausführung, sonst "channel not found".

Verweise

`do_range()`

Empfangsbereitschaft ermitteln *get_receiver_ready*

`get_receiver_ready()` ermittelt ob das TRC2-IP-Modul empfangsbereit ist. Dazu wird das "status"-Register ausgelesen und Bit 5 ausgewertet.

Prototyp und Parameter

`get_receiver_ready,device,slot`

`device:`	Name der Gerätedatei z.B. /dev/pciip0
`slot:`	Steckplatz des IP-Moduls (A,B,C oder D)

Rückgabewert

"ok" bei fehlerfreier Ausführung, sonst "error".

==
================== Nachbesserung! ===============

==
=========

Verweise

`status()`

Abtastrate ermitteln *get_sampling_rate*

get_sampling_rate() ermittelt die Abtastrate eines TRC2-IP-Moduls.

Prototyp und Parameter

`get_sampling_rate,device,slot`

`device:`	Name der Gerätedatei z.B. /dev/pciip0
`slot:`	Steckplatz des IP-Moduls (A,B,C oder D)

Rückgabewert

Abtastrate in dezimaler Darstellung bei fehlerfreier Ausführung, sonst "channel not found".

Verweise

`do_sampling_rate()`

Steckplatz des IP-Moduls ermitteln *get_slot*

`get_slot()` ermittelt den Steckplatz des IP-Moduls.

Prototyp und Parameter

`get_slot,device,slot`

`device:`	Name der Gerätedatei z.B. /dev/pciip0
`slot:`	Steckplatz des IP-Moduls (A,B,C oder D)

Rückgabewert

Steckplatz des IP-Moduls bei fehlerfreier Ausführung, sonst "error".

get_slot_list

`get_slot_list()`

Prototyp und Parameter

`get_slot_list,device`

`device:`	Name der Gerätedatei z.B. /dev/pciip0

Rückgabewert

"channel not found".

Verweise

`do_slot(), delete_slot()`

get_slot_parameter

`get_slot_parameter()`

Prototyp und Parameter

`get_slot_parameter,device,slot`

`device:`	Name der Gerätedatei z.B. /dev/pciip0
`slot:`	Steckplatz des IP-Moduls (A,B,C oder D)

Rückgabewert

"channel not found".

Verweise

`do_post_trigger_cycles(), get_post_trigger_cycles(), do_sampling_rate(), get_sampling_rate(), get_mode(), do_clockshift(), get_clockshift()`

get_slot_pointer

`get_slot_pointer()`

Prototyp und Parameter

`get_slot_pointer,device,slot`

`device:`	Name der Gerätedatei z.B. /dev/pciip0
`slot:`	Steckplatz des IP-Moduls (A,B,C oder D)

Rückgabewert

"channel not found".

Zustand der Testspannung ermitteln *get_testvoltage*

`get_testvoltage()` ermittelt ob die Testspannung eingeschaltet ist.

Prototyp und Parameter

`get_testvoltage,device,slot,channel`

`device:`	Name der Gerätedatei z.B. /dev/pciip0

`slot:`	Steckplatz des IP-Moduls (A,B,C oder D)
`channel:`	Kanalnummer (0 bis 7)

Rückgabewert

"on" oder "off" bei fehlerfreier Ausführung, sonst "channel not found".

Verweise

`do_testvoltage()`

Sendebereitschaft ermitteln *get_transmitter_ready*

`get_transmitter_ready()` ermittelt ob das TRC2-IP-Modul sendebereit ist. Dazu wird das "status"-Register ausgelesem und Bit 4 ausgewertet.

Prototyp und Parameter

`get_transmitter_ready,device,slot`

`device:`	Name der Gerätedatei z.B. /dev/pciip0
`slot:`	Steckplatz des IP-Moduls (A,B,C oder D)

Rückgabewert

"ok" bei fehlerfreier Ausführung, sonst "error".

==
=================== Nachbesserung! ===============
==
=========

Verweise

`status()`

Triggerquelle ermitteln *get_trigger_source*

`get_trigger_source()` ermittelt ob der Kanal intern oder extern getriggert wird.

Prototyp und Parameter

`get_trigger_source,device,slot`

`device:`	Name der Gerätedatei z.B. /dev/pciip0
`slot:`	Steckplatz des IP-Moduls (A,B,C oder D)

Rückgabewert

"extern" oder "intern" bei fehlerfreier Ausführung, sonst "slot not found".

Verweise

`do_trigger_source()`

get_value

`get_value()`

Prototyp und Parameter

`get_value,device,slot,channel`

`device:`	Name der Gerätedatei z.B. /dev/pciip0
`slot:`	Steckplatz des IP-Moduls (A,B,C oder D)
`channel:`	Kanalnummer (0 bis 7)

Rückgabewert

"channel not found".

Verweise

`rx_dio_sel()`

Xor ermitteln *get_xor*

`get_xor()` ermittelt den Wert "xor" der Stopbedingung.

Prototyp und Parameter

`get_xor,device,slot,channel`

`device:`	Name der Gerätedatei z.B. /dev/pciip0
`slot:`	Steckplatz des IP-Moduls (A,B,C oder D)
`channel:`	Kanalnummer (0 bis 7)

Rückgabewert

Größe des Wertes "xor" bei fehlerfreier Ausführung, sonst "channel not found".

Verweise

do_xor()

Interruptvektor0 schreiben *intrpt_vec0*

`intrpt_vec0()` schreibt den Vektor in das "intrpt_vec0"-Register.

Prototyp und Parameter

`intrpt_vec0,device,slot,value`

`device:`	Name der Gerätedatei z.B. /dev/pciip0
`slot:`	Steckplatz des IP-Moduls (A,B,C oder D)
`value:`	8-Bit-Wert der in das Register geschrieben werden soll. Der Wert muß in hexadezimaler Darstellung vorliegen, wobei die führenden Zeichen *0x* weggelassen werden können.

Rückgabewert

"ok" bei fehlerfreier Ausführung, sonst "error".

Verweise

`do_interrupt0(), get_interrupt0, intrpt_vec1(), do_interrupt1(), get_interrupt1()`

Interruptvektor1 schreiben *intrpt_vec1*

`intrpt_vec1()` schreibt den Vektor in das "intrpt_vec1"-Register.

Prototyp und Parameter

```
intrpt_vec1,device,slot,value
```

`device:`	Name der Gerätedatei z.B. /dev/pciip0
`slot:`	Steckplatz des IP-Moduls (A,B,C oder D)
`value:`	8-Bit-Wert der in das Register geschrieben werden soll. Der Wert muß in hexadezimaler Darstellung vorliegen, wobei die führenden Zeichen *0x* weggelassen werden können.

Rückgabewert

"ok" bei fehlerfreier Ausführung, sonst "error".

Verweise

`do_interrupt0(), get_interrupt0, intrpt_vec0, do_interrupt1(), get_interrupt1()`

Level-Register schreiben *level*

`level()` schreibt ein 16-Bit Wort in das "level"-Register.

Prototyp und Parameter

```
level,device,slot,value
```

`device:`	Name der Gerätedatei z.B. /dev/pciip0
`slot:`	Steckplatz des IP-Moduls (A,B,C oder D)
`value:`	16-Bit-Integer in hexadezimaler Darstellung.

Rückgabewert

"ok" bei fehlerfreier Ausführung, sonst "error".

Verweise

`do_level(), get_level()`

Mask-Register schreiben *mask*

`mask()` schreibt ein 16-Bit Wort in das "mask"-Register.

Prototyp und Parameter

`mask,device,slot,value`

`device:`	Name der Gerätedatei z.B. /dev/pciip0
`slot:`	Steckplatz des IP-Moduls (A,B,C oder D)
`value:`	16-Bit-Integer in hexadezimaler Darstellung.

Rückgabewert

"ok" bei fehlerfreier Ausführung, sonst "error".

Verweise

`do_mask(), get_mask()`

TRC2-IP-Module initialisieren *module_init*

`module_init()` schreibt die Konfigurationsdaten, die sich in den dynamischen Speicherstrukturen befinden, in die IP-Module. Es können nur alle IP-Module zusammen konfiguriert werden. Eine gezielte Auswahl eines IP-Moduls ist nicht möglich.

Prototyp

`module_init`

Rückgabewert

"ok" bei fehlerfreier Ausführung, sonst "error".

Verweise

`read_init(), write_init()`

Server beenden *quit*

`quit()` beendet die Programmausführung des Deviceservers.

Prototyp und Parameter

`quit`

keine Parameter

Rückgabewert

keiner

Verweise

Register "control-word" auslesen *read_control_word*

`read_control_word()` liest das "control_word"-Register aus.

Prototyp und Parameter

`read_control_word,device,slot`

`device:`	Name der Gerätedatei z.B. /dev/pciip0
`slot:`	Steckplatz des IP-Moduls (A,B,C oder D)

Rückgabewert

Die Zeichenkette "ok" gefolgt von 8 Bits durch Komma getrennt. Das höchstwertigste Bit folgt direkt hinter den "ok". Beispiel:

ok,1,1,0,0,1,0,0,0

Im Fehlerfall wird die Zeichenkette "error" zurückgegeben.

Verweise

`write_control_word()`

Allgemeiner Lesezugriff auf Gerätedateien *read_general*

`read_general()` liest beliebige Daten aus einer Gerätedatei. Dabei kann es sich sowohl um Register des Carriers als auch um Speicherbereiche von IP-Modulen handeln. Hierbei handelt es sich um eine elementare Funktion die keinen Komfort bietet. Die angegebenen Parameter werden benutzt ohne sie vorher zu checken. Wenn man also eine Adresse angibt an der sich kein Speicher befindet dann stürzt der Rechner ab.

Prototyp und Parameter

`read_general,device,addr,count`

`device:`	Name der Gerätedatei z.B. /dev/pciip0
`addr:`	Adresse innerhalb der Gerätedatei in hexadezimaler Schreibweise.
`count:`	Anzahl der Bytes die gelesen werden sollen, maximal 64. Wenn es sich um eine gerade Zahl handelt findet wortweiser Zugriff statt. Bei einer ungeraden Zahl findet byteweiser Zugriff statt.

Rückgabewert

Zurückgeliefert wird eine Zeichenkette die mit Steuerzeichen für eine Ausgabe im Textmodus durchsetzt ist. In der ersten Zeile steht, zur Kontrolle, die Anzahl der gelesenen Bytes. Die übrigen Zeilen enthalten bis zu acht Bytes in hexadezimaler Darstellung. Die einzelnen Bytes sind hier durch Leerzeichen voneinander getrennt und nicht durch Kommas. Die Rückgabedaten könnten z.B. so aussehen:

```
no of bytes read : 18
0x00 0x01 0x02 0x03 0x04 0x05 0x06 0x07
0x08 0x09 0x0A 0x0B 0x0C 0x0D 0x0E 0x0F
0x10 0x11
```

Im Fehlerfall wird "error" zurückgeliefert.

Verweise

`write_general()`

Initialisierungsdatei lesen *read_init*

`read_init()` liest die Initialisierungsdaten aus einer Datei.

Prototyp und Parameter

`read_init,filename`

`filename:`	Pfad und Name der Initialisierungsdatei

Rückgabewert

"ok" bei fehlerfreier Ausführung, sonst "error".

Verweise

`module_init(), write_init()`

Nummer des letzten Meßwertes lesen *rx_address*

`rx_address()` liest die Nummer des zuletzt aufgezeichneten Meßwertes. Die Nummer ist gleichbedeutend mit dem Adreßoffset bei wortweiser Adressierung.

Prototyp und Parameter

`rx_address,device,slot`

device:	Name der Gerätedatei z.B. /dev/pciip0
slot:	Steckplatz des IP-Moduls (A,B,C oder D)

Rückgabewert

Die Zeichenkette der Form:

`ok,0xXXXX`

XXXX ist die Meßwertnummer in hexadezimaler Form. Die Meßwertnummer muß im Bereich von 0(dezimal) bis 8191(dezimal) liegen. Im Fehlerfall wird die Zeichenkette "error" zurückgegeben.

Verweise

`get_ipdata()`

Letzten Meßwert eines Kanals lesen *rx_dio_sel*

`rx_dio_sel()` liest den letzten Meßwert eines Kanals. Dieses Kommando macht einen Zugriff auf einen Zwischenspeicher im TRC2-IP-Modul. Ein Zugriff auf den Memorybereich des IP-Moduls findet nicht statt.

Prototyp und Parameter

`rx_dio_sel,device,slot,channel`

device:	Name der Gerätedatei z.B. /dev/pciip0
slot:	Steckplatz des IP-Moduls (A,B,C oder D)
channel:	Kanalnummer (0 bis 7)

Rückgabewert

Eine Zeichenkette der Form:

`rx_dio_sel = 0xXXXX`

XXXX ist der Inhalt des Zwischenspeichers in hexadezimaler Form. Dieser 16-Bit-Wert ist direkt dem IP-Modul entnommen, d.h. es handelt sich nicht um ein Standarddatenformat. Die Umwandlung in einen *signed short integer* ist in der Dokumentation des TRC2-IP-Moduls beschrieben.

Verweise

`rx_trigger()`

Einmaligen Lesezyklus ausführen *rx_trigger*

`rx_trigger()` führt einen einmaligen Lesezyklus aus. Im IP-Modul befindet sich dann ein Meßwert der abgerufen werden kann.

Prototyp und Parameter

rx_trigger,device,slot

`device:`	Name der Gerätedatei z.B. /dev/pciip0
`slot:`	Steckplatz des IP-Moduls (A,B,C oder D)

Rückgabewert

"ok" bei fehlerfreier Ausführung, sonst "error".

Verweise

`rx_dio_sel()`

set_channel

`set_channel()`

Prototyp und Parameter

`set_channel,device,slot,channel`

`device:`	Name der Gerätedatei z.B. /dev/pciip0
`slot:`	Steckplatz des IP-Moduls (A,B,C oder D)
`channel:`	Kanalnummer (0 bis 7)

Rückgabewert

"channel not found".

set_device

`set_device()`

Prototyp und Parameter

`set_device,device`

`device:`	Name der Gerätedatei z.B. /dev/pciip0

Rückgabewert

"channel not found".

set_slot

`set_slot()`

Prototyp und Parameter

`set_slot,device,slot`

`device:`	Name der Gerätedatei z.B. /dev/pciip0
`slot:`	Steckplatz des IP-Moduls (A,B,C oder D)

Rückgabewert

"channel not found".

show_memory

`show_memory()`

Prototyp und Parameter

`show_memory`

Rückgabewert

"channel not found".

`Meßbetrieb starten` **`start_all_datataking`**

`start_all_datataking()` startet die Datenahme aller IP-Module..

Prototyp

`start_all_datataking`

Rückgabewert

"ok" bei fehlerfreier Ausführung, sonst "error".

Verweise

`start_datataking(), stop_all_datataking(), stop_datataking()`

Meßbetrieb starten *start_datataking*

`start_datataking()` startet die Datenahme eines IP-Moduls.

Prototyp und Parameter

`start_datataking,device,slot`

`device:`	Name der Gerätedatei z.B. /dev/pciip0
`slot:`	Steckplatz des IP-Moduls (A,B,C oder D)

Rückgabewert

"ok" bei fehlerfreier Ausführung, sonst "error".

Verweise

`start_all_datataking(), stop_all_datataking(), stop_datataking()`

Statusregister auslesen *status*

`status()` liest das "status"-Register aus.

Prototyp und Parameter

status,device,slot

`device:`	Name der Gerätedatei z.B. /dev/pciip0
`slot:`	Steckplatz des IP-Moduls (A,B,C oder D)

Rückgabewert

Die Zeichenkette "ok" gefolgt von 8 Bits durch Komma getrennt. Das höchstwertigste Bit folgt direkt hinter den "ok". Beispiel:

`ok,1,1,0,0,1,0,0,0`

Im Fehlerfall wird die Zeichenkette "error" zurückgegeben.

Verweise

`get_mode()`

Meßbetrieb stoppen *stop_all_datataking*

`stop_all_datataking()` stoppt die Datennahme aller IP-Module..

Prototyp

`stop_all_datataking`

Rückgabewert

"ok" bei fehlerfreier Ausführung, sonst "error".

Verweise

`start_all_datataking(), start_datataking(), stop_datataking()`

Meßbetrieb stoppen *stop_datataking*

`stop_datataking()` stoppt die Datennahme eines IP-Moduls.

Prototyp und Parameter

stop_datataking,device,slot

`device:`	Name der Gerätedatei z.B. /dev/pciip0
`slot:`	Steckplatz des IP-Moduls (A,B,C oder D)

Rückgabewert

"ok" bei fehlerfreier Ausführung, sonst "error".

Verweise

`start_all_datataking(), start_datataking(), stop_all_datataking()`

Tastkopf konfigurieren *tx_write*

`tx_write()` schreibt einen 16-Bit-Wert in einen Tastkopf.

Prototyp und Parameter

`tx_write,device,slot,channel,value`

device:	Name der Gerätedatei z.B. /dev/pciip0
slot:	Steckplatz des IP-Moduls (A,B,C oder D)
channel:	Kanalnummer (0 bis 7)
value:	16-Bit-Wert der in den Tastkopf geschrieben werden soll. Der Wert muß in hexadezimaler Darstellung vorliegen, wobei die führenden Zeichen *0x* weggelassen werden können. Die signifikanten Bits müssen rechtsbündig angeordnet sein, da der Deviceserver sie vor dem Absenden nach links verschiebt und zwar bei analogen Tastköpfen um 2 Bit und bei digitalen Tastköpfen um 9 Bit.

Rückgabewert

"ok" bei fehlerfreier Ausführung, sonst "error".

Verweise

`do_range(), do_bandwidth, do_testvoltage()`

Register "control-word" beschreiben *write_control_word*

write_control_word() schreibt in das "control_word"-Register.

Prototyp und Parameter

```
write_control_word,device,slot,value
```

device:	Name der Gerätedatei z.B. /dev/pciip0
slot:	Steckplatz des IP-Moduls (A,B,C oder D)
value:	8-Bit-Wert der in das Register geschrieben werden soll. Der Wert muß in hexadezimaler Darstellung vorliegen, wobei die führenden Zeichen *0x* weggelassen werden können.

Rückgabewert

"ok" bei fehlerfreier Ausführung, sonst "error".

Verweise

`read_control_word()`

Analoge Meßwerte in Datei schreiben *write_data_file*

`write_data_file()` schreibt die Meßwerte eines analogen Kanales in eine Textdatei des Formats **.csv*. Diese Datei wird auf dem Deviceserver angelegt und dient eher Servicezwecken. Der normale Meßbetrieb wird anders abgewickelt. Angelegt werden zwei Spalten und 8192 Zeilen. Die erste Spalte enthält die vergangene Zeit seit dem ersten Meßwert, wobei das Stopsignal und der Nachlauf (post_trigger_cycles) unberücksichtigt bleiben. Die erste Zeitangabe ist also immer 0. Die zweite Spalte enthält

den Meßwert in physikalischer Einheit als reiner Zahlenwert ohne Angabe der Einheit, damit Plotprogramme diese Datei direkt verarbeiten können. Ein Auszug aus so einer Datei sieht so aus:

```
0.00000,6.54874
0.00800,7.01093
0.01600,7.24012
0.80000,12.4407
```

Prototyp und Parameter

`write_data_file,device,slot,channel,filename`

`device:`	Name der Gerätedatei z.B. /dev/pciip0
`slot:`	Steckplatz des IP-Moduls (A,B,C oder D)
`channel:`	Kanalnummer (0 bis 7)
`filename:`	Dateiname mit Pfadangabe. Falls eine Datei dieses Namens bereits existiert wird sie überschrieben.

Rückgabewert

"ok" bei fehlerfreier Ausführung, sonst "error".

Verweise

`write_hexdata_file()`

Allgemeiner Schreibzugriff auf Gerätedateien — *write_general*

`write_general()` schreibt beliebige Daten in eine Gerätedatei. Dabei kann es sich sowohl um Register des Carriers als auch um Speicherbereiche von IP-Modulen handeln. Hierbei handelt es sich um eine elementare Funktion die keinen Komfort bietet. Die angegebenen Parameter werden benutzt ohne sie vorher zu checken. Wenn man also eine Adresse angibt an der sich kein Speicher befindet dann stürzt der Rechner ab!

Prototyp und Parameter

`write_general,device,addr,count,value`

`device:`	Name der Gerätedatei z.B. /dev/pciip0
`addr:`	Adresse innerhalb der Gerätedatei in hexadezimaler Schreibweise.
`count:`	Anzahl der Bytes die geschrieben werden sollen, maximal 64. Wenn es sich um eine gerade Zahl handelt findet wortweiser Zugriff statt. Bei einer ungeraden Zahl findet byteweiser Zugriff statt.
`value:`	Zeichenkette mit den zu schreibenden hexadezimalen Daten in ASCII-Darstellung. Für die Darstellung eines Bytes werden

zwei Schriftzeichen benötigt. Führende "0x" dürfen nicht geschrieben werden. Eine solche Zeichenkette entsteht wenn die Daten über die Tastatur eingegeben werden. Die Software wandelt Schritt für Schritt die beiden Schriftzeichen eines Bytes in eine echte hexadezimale Zahl um, die in einem Byte Platz hat, bevor der Zugriff auf die Hardware erfolgt.

Rückgabewert

Zurückgeliefert wird eine Zeichenkette mit der Anzahl der geschriebenen Bytes.

Beispiel:

```
no of bytes written : 4
```

Im Fehlerfall wird "error" zurückgeliefert.

Verweise

`read_general()`

Rohdaten in Datei schreiben *write_hexdata_file*

`write_hexdata_file()` schreibt die Meßwerte eines Kanales in eine Textdatei des Formats **.csv*. Diese Datei wird auf dem Deviceserver angelegt und dient eher Servicezwecken. Der normale Meßbetrieb wird anders abgewickelt. Angelegt werden zwei Spalten und 8192 Zeilen. Die erste Spalte enthält die Nummer des Meßwertes (0 - 8191). Die zweite Spalte enthält den Meßwert in hexadezimaler Darstellung. Ausgegeben werden nur die niederwertigsten 12Bit. Ein Auszug aus so einer Datei sieht so aus:

```
0000,0x7FF
0001,0x02A
0002,0x00F


8191,0xF00
```

Prototyp und Parameter

`write_hexdata_file,device,slot,channel,filename`

`device:`	Name der Gerätedatei z.B. /dev/pciip0
`slot:`	Steckplatz des IP-Moduls (A,B,C oder D)
`channel:`	Kanalnummer (0 bis 7)
`filename:`	Dateiname mit Pfadangabe. Falls eine Datei dieses Namens bereits existiert wird sie überschrieben.

Rückgabewert

"ok" bei fehlerfreier Ausführung, sonst "error".

Verweise

`write_data_file()`

Initialisierungsdatei schreiben *write_init*

`write_init()` schreibt die Intialisierungsdaten in eine Datei.

Prototyp und Parameter

`write_init,filename`

`filename:`	`Pfad und Name der Initialsierungsdatei`

Rückgabewert

"ok" bei fehlerfreier Ausführung, sonst "error".

Verweise

`module_init(), read_init()`

Xor-Register *xor*

`xor()` schreibt ein 16-Bit Wort in das "xor"-Register.

Prototyp und Parameter

`xor,device,slot,value`

`device:`	Name der Gerätedatei z.B. /dev/pciip0
`slot:`	Steckplatz des IP-Moduls (A,B,C oder D)
`value:`	16-Bit-Integer in hexadezimaler Darstellung.

Rückgabewert

"ok" bei fehlerfreier Ausführung, sonst "error".

Verweise

`do_xor(), get_xor()`

4.3.1.3 Kommandosyntax

Die Programmbedienung orientiert sich an den Kommandointerpretern wie sie von MS-DOS oder UNIX bekannt sind. An einem Prompt können eine Reihe von Befehlen eingegeben und zur Ausführung gebracht werden. Die Konfiguration der Hardware macht sich die Baumstruktur der Hardware zu Nutze. An oberster Stelle in der Hierarchie stehen die Gerätedateien der IP-Carrier. Direkt darunter sind die einzelnen Slots der IP-Carrier angeordnet, die die IP-Module aufnehmen. Die nächste Ebene ist nun speziell auf die TRC2-Module ausgerichtet und besteht aus den einzelnen Kanälen der TRC2-Module, bzw. den Tastköpfen. Für den Fall, daß es sich um einen digitalen Tastkopf

handelt gibt es eine vierte und letzte Ebene - das Bit, oder auch digitalen Kanal. Für die Reise durch die Hardwareebenen und die Manipulation der Angaben werden die gleichen Kommandos verwendet mit denen man in einem Computerdateisystem die anstehenden Arbeiten verrichtet. Sowohl die UNIX-Kommandos auch auch die entsprechenden MS-DOS Versionen sind implementiert. Soweit so gut. Starten wir in das Programm:

Prompt

Als Prompt findet der Programmname gefolgt von "" Verwendung.

```
trc2:
```

Der Prompt verändert sein Aussehen wenn man in den Hardwareebenen rauf und runterwechselt. In dem folgenden Fall zeigt der Prompt an, daß wir in die Gerätedatei "/dev/pciip" gewechselt haben.

```
trc2:/dev/pciip:
```

Wenn man noch eine Ebene tiefergeht und einen Slot anwählt nimmt der Prompt folgendes Aussehen an:

```
trc2:/dev/pciip:A:
```

Die Doppelpunkte dienen dazu die Hardwareebenen voneinander zutrennen. Die bekannten Trennzeichen "\" oder "/" kommen dafür leider nicht Frage, weil sie Bestandteil des Gerätedateinamens sind.

Kommandos

Jetzt folgt eine Auflistung aller verfügbaren Kommandos mit Angabe von Beispielen.

cd

Dieser von MS-DOS und UNIX gleichermaßen benutzte Befehl dient dazu die Hardwareebenen zu wechseln, analog den Verzeichnisebenen eines Dateisystems. Als Parameter muß die gewünschte Gerätedatei, der gewünschte Slot oder der gewünschte Kanal angegeben werden um tiefer in die Hardware einzudringen. Den umgekehrten Weg beschreitet man mit dem bekannten Parameter "..", wobei aber darauf zu achten ist, daß zwischen "cd" und ".." ein Leerzeichen steht, wie es unter UNIX üblich ist. Das MS-DOS-Kuriosum "cd.." ohne Leerzeichen nach dem "cd" wird nicht akzeptiert. Der Parameter ".", der auf das aktuelle Verzeichnis verweist wird ebenfalls akzeptiert. In einem Punkt unterscheidet sich der Befehl allerdings von seinem Vorbild. Man kann nämlich immer nur eine Ebene rauf oder runterschalten. Ein Überspringen einer Ebene, indem man einen entsprechenden Parameter eingibt ist nicht möglich.

```
trc2:cd /dev/pciip
trc2:/dev/pciip:cd A
trc2:/dev/pciip:A:cd 4
trc2:/dev/pciip:A:4:cd ..
trc2:/dev/pciip:A:cd ..
trc2:/dev/pciip:cd ..
```

```
trc2:
```

ls, dir

Damit man sich nicht das gesamte Hardwaregebilde merken muß kann man sich die Einträge der einzelnen Ebenen anzeigen lassen. Von MS-DOS ist der Befehl "dir" bekannt, das UNIX-Pendent lautet "ls".

```
trc2:ls
available devices:
------------------
   /dev/pciip
trc2:cd /dev/pciip
trc2:/dev/pciip:dir
available slots:
----------------
   A
trc2:/dev/pciip:cd A
trc2:/dev/pciip:A:ls
available channels:
-------------------
   3   analog probe
   4   digital probe
trc2:/dev/pciip:A:cd 4
trc2:/dev/pciip:A:4:dir
available bits:
---------------
   0
   1
   2
   3
   4
   5
   6
   7
   8
   9
   10
   11
trc2:/dev/pciip:A:4:
cat, more, type
```

Die Parameter einzelner Einträge einer Hardwareebene kann man sich auf die gleiche Weise anzeigen lassen wie den Inhalt einer Datei. Die von UNIX bekannten Kommandos "cat" und "more" und das von MS-DOS bekannte Kommando "type" dienen dazu.

Der Vollständigkeit halber kann man sich auch die Eigenschaften der Gerätedatei ansehen obwohl es da nichts zu sehen gibt.

```
trc2:ls
available devices:
------------------
   /dev/pciip
trc2:more /dev/pciip
device: /dev/pciip
-----------------------------
```

```
nothing to do
trc2:
```

Die IP-Module vom Typ TRC2 haben als Parameter nur die Anzahl der Meßwerte nach dem Stopsignal und die Abtastrate. Die Abtastrate von 95238.093750 Hz, die im folgenden Beispiel angegeben ist, kommt durch den internen Trigger des IP-Moduls zustande. Das IP-Modul liest, intern getriggert, alle 10.5us einen neuen Meßwert.

```
trc2:/dev/pciip:ls

available slots:
----------------
   D
trc2:/dev/pciip:more D
device: /dev/pciip, slot: D
------------------------------
   post_trigger_cycles  : 6000
   sampling_rate        : 95238.093750

trc2:/dev/pciip:
```

Die Kanäle haben insgesamt unterschiedliche Parameter je nachdem ob es sich um einen analogen oder digitalen Kanal handelt. Das Beispiel zeigt einen analogen Kanal.

```
trc2:/dev/pciip:D:ls
available channels:
-------------------
   0 (analog probe)
trc2:/dev/pciip:D:more 0
device: /dev/pciip, slot: D, channel: 0
------------------------------
   channelname          : testanalog
   range                : 30V
   bandwidth            : 200kHz
   testvoltage          : off
   engineering unit     : V
   eng.unit low factor  : 1.000000
   eng.unit high factor : 1.000000

trc2:/dev/pciip:D:
```

Ein digitaler Kanal teilt sich in einzelne Bit auf. Die Parameter können für jedes Bit getrennt eingestellt werden. Gemeinsame Parameter, die für alle Bits gelten gibt es nur bei analogen Tastköpfen. Wenn man trotzdem versucht sich die Parameter eines digitalen Tastkopfes anzusehen macht das Programm auf diesen Sachverhalt aufmerksam.

```
trc2:/dev/pciip:D:ls
available channels:
-------------------
   0 (analog probe)
   1 (digital probe)
trc2:/dev/pciip:D:more 1
channel is not connected to an analog probe
trc2:/dev/pciip:D:
```

Die Parameter digitaler Tastköpfe sind erst zugänglich wenn man sich auf der Bitebene befindet.

```
trc2:/dev/pciip:D:ls
```

```
available channels:
-------------------
   0 (analog probe)
   1 (digital probe)
trc2:/dev/pciip:D:cd 1
trc2:/dev/pciip:D:1:ls
available bits:
---------------
   0
   1
   2
   3
   4
   5
   6
   7
   8
   9
   10
   11
trc2:/dev/pciip:D:1:more 0
device: /dev/pciip, slot: D, channel: 1, bit: 0
-------------------------------
   channelname          :
   level                : 24V
   mode                 : sample
   engineering unit     : V
   eng.unit low factor  : 1.000000
   eng.unit high factor : 1.000000

trc2:/dev/pciip:D:1:
```

edit

Das Ändern von Parameter geschieht mit dem Kommando "edit", so als wenn man eine Datei editieren möchte. Das Programm stellt nacheinander alle Parameter und deren augenblicklichen Wert dar. Der Wert jedes Parameters kann neu eingegeben werden oder durch einfaches Betätigen der ENTER-Taste unverändert bleiben. Bei Parametern die nicht frei gestaltet werden können, wie z.B. eine Meßbereichsangabe die nur wenige Werte zuläßt, werden die möglichen Einstellungen angezeigt. Nach der Bearbeitung eines Parameters wird der neue Wert angezeigt. Die folgenden Zeilen zeigen die "edit" Funktion am Beispiel eines Analogkanales.

```
trc2:/dev/pciip:A:ls
available channels:
-------------------
   3   analog probe
   4   digital probe
trc2:/dev/pciip:A:edit 3
edit analog probe
device = /dev/pciip, slot = A, channel = 3
channelname = testanalog | insert new name (or ENTER for unchange):
labortest
channelname = labortest -- ok
```

```
range = 10V | 30V, 10V, 1V, 100mV : 100mV
range = 100mV -- ok
bandwidth = 200kHz | 200kHz, 100kHz, 25kHz, 10kHz, 1kHz : 25kHz
bandwidth = 25kHz -- ok
testvoltage = off | on, off : <ENTER
testvoltage = off -- unchanged
engineering_unit = V | : mA
engineering_unit = mA -- ok
engineering unit low factor = 1.000000 | : 2.5
engineering unit low factor = 2.500000 -- ok
engineering unit high factor = 1.000000 | : 2.5
engineering unit high factor = 2.500000 -- ok
trc2:/dev/pciip:A:
```

mkdir, md

Nun will man nicht nur vorhandene Einträge ansehen und verändern, sondern auch neue anlegen. Das geschieht mit den Kommandos mit denen man in einem Dateisystem neue Verzeichnisse erzeugt. In UNIX ist das der Befehl "mkdir" und in MS-DOS "md".

rmdir, rm, del

Wenn man Einträge anlegen kann muß man sie auch wieder löschen können. Dazu dienen die UNIX-Kommandos "rmdir" und "rm" und das MS-DOS Kommando "del".

read

Lesen von Daten.

write

Schreiben von Daten.

Aus der Sicht des Transientenrecorderprogrammes können verschiedene Arten von Daten gelesen und schrieben werden. Auch die Quelle oder Ziel können unterschiedlich sein. Da gibt es zunächst einmal die Einstellungs-, Konfigurations- oder Initialisierungsdaten der Hardware. Diese Daten können als gewöhnliche Datei vorliegen oder sich im Speicher des Programmes befinden. Man benötigt zunächst einmal die Möglichkeit diese Daten aus der Datei in den Speicher zu lesen. Dazu dient der Befehl:

```
trc2:read -i <Dateiname.ini
```

Der Dateiname kann frei gewählt werden, allerdings ist die Erweiterung ".ini" wichtig. Wenn die angegebene Datei nicht mit der Erweiterung ".ini" endet geht das Programm von einem Schreibfehler des Anwenders aus und versucht die Datei "trc2.ini" zu lesen. Mit der Edit-Funktion geänderte Hardwareeinstellungen lassen sich mit dem Befehl

```
trc2:write -i <Dateiname.ini
```

in eine Datei zurückschreiben. Die Behandlung des Dateinamens erfolgt hier genauso wie bei der Read-Funktion. Als nächstes wären die Meßwerte zu nennen. Dieses Programm ist eigentlich nur ein Glied in einer ganzen Kette von Programmen die alle dazu dienen

transiente Daten zu verwalten und darzustellen. Dieses Programm ist ein sogenannter Deviceserver, dessen Aufgabe es ist die Hardware zu bedienen und die Daten auf Anforderung des Archivservers in einem Standardformat bereitzustellen. Die Archivierung der Daten gehört zu den Aufgaben des Archivservers. Dennoch bietet dieses Deviceserverprogramm die Möglichkeit Meßwerte in eine Datei zu schreiben. Damit soll allerdings nicht dem Archivserver Konkurrenz gemacht werden. Diese Schreibfunktion soll nur bei Inbetriebnahmeproblemen und bei der Fehlersuche Hilfestellung leisten. In so einem Fall ist es nämlich wenig sinnvoll das globale Archiv mit Daten zu überschwemmen die keine Informationen über die Beschleuniger enthalten. Mit dem Befehl

```
trc2:write -d <Dateiname
```

können die Meßwerte eines Kanales als physikalischer Wert in Klartext gefolgt von einer Zeilenendemarkierung in eine Datei geschrieben werden. Damit erhält man eine Datei die eine Tabelle enthält, bestehend aus zwei Spalten und 8192 Zeilen, im "*.csv" Format. In der ersten Spalte steht die Zeitinformation und in der zweiten Spalte die Meßwerte. Die Zeitinformation in der ersten Spalte kann direkt als Zeitachse für die graphische Darstellung verwendet werden. Die Zahlen geben die Zeit in Sekunden an, die seit der Messung des erstes Wertes vergangen ist. Dieses Format wird von vielen Datenbanken und Tabellenkalkulationen verstanden. Allerdings sollte man bedenken, das Unixsysteme eine andere Zeilenendemarkierung verwenden als Windows. Dies ist ein generelles Problem wenn man Dateien hin und her kopiert. Es gibt diverse Programme die Abhilfe schaffen, z.B. dos2unix und unix2dos die in vielen Linux-Distributionen enthalten sind. Die Programme dos2unix und unix2dos liegen als C-Quellcode vor und müßten auch auf Windows portierbar sein. Ein Test mit Excel hat ergeben, daß Excel mit der Unix-Zeilenendemarkierung keine Probleme hat, dafür kann Excel aber nur max. 4000 Werte graphisch darstellen. Die nächste Sorte von Daten sind Registerinhalte eines IP-Moduls. Wenn man den Parameter "-m" benutzt werden die Register angesprochen.

`-b oder -batch`	Datei als Batchdatei benutzen
`-i oder -init`	Datei als Initialisierungsdatei benutzen
`-d oder -data`	Meßwerte eines Kanals in eine Datei schreiben
`-m oder -modul`	Registerinhalt eines IP-Moduls lesen/schreiben

Bei den Registern gibt es die Möglichkeit den Registernamen direkt anzugeben. Das Programm fragt dann nicht nach Adresse und Größe. Folgende Registernamen werden akzeptiert:

- **read**

```
rx_dio_sel
control_word
rx_address
status
```

- **write**

```
rx_trigger
tx_write
control_word
```

```
      intrpt_vec0
      intrpt_vec1
      cy_sw_stop
      cy_post_reg
      clock_shift
      mask
      level
      xor
      config
```

Die Praxis sieht so aus:

```
trc2:read -m status
available devices:
------------------
   /dev/pciip
insert device : /dev/pciip
available slots:
----------------
   D
insert slot : D
ok, status = 0x30
  D7  mode(1) = 0
  D6  mode(0) = 0
  D5  receiver ready = 1
  D4  transmitter ready = 1
  D3  ioSTOP input = 0
  D2  ioTRIGGER input = 0
  D1  = 0
  D0  = 0
trc2:
```

Weil das Programm mehrere IP-Module bedienen kann muß es zunächst feststellen welches IP-Modul gemeint ist. Diese Abfragen nach device und slot können auf die Dauer lästig werden und lassen sich vermeiden indem man in der Hardwarehierarchie ein paar Ebenen tiefer geht und so die Auswahl vorweg nimmt.

```
trc2:cd /dev/pciip
trc2:/dev/pciip:cd D
trc2:/dev/pciip:D:read -m status
readdata : Ioaddr = 0x00004008
readdata : buf[0] = 0x30
ok, status = 0x30

trc2:/dev/pciip:D:
```

Gibt man als Registernamen "general" an dann bekommt einen allgemeinen Zugriff auf die Gerätedatei. Mit dieser Methode kann man beliebig im gesamten Speicherbereich, den der Gerätetreiber zur Verfügung stellt, schreiben und lesen. Natürlich muß man die richtige Adresse und Registergröße wissen. Auch die Register des IP-Carriers lassen so auslesen und beschreiben.

```
Trc2:/dev/pciip:D:read -m general
address [hex] : 4008
no of bytes [decimal] : 1
readdata : Ioaddr = 0x00004008
```

```
readdata : buf[0] = 0x30
no of bytes read : 1
0x30

trc2:/dev/pciip:D:
```

Bei der Angabe der Anzahl der Bytes gibt es noch etwas zu beachten. In der IP Welt gibt es Register auf die wortweise zugegriffen werden muß während andere byteweisen Zugriff erlauben. Der Gerätetreiber ist nun so programmiert, daß wortweiser Zugriff immer dann stattfindet wenn die Anzahl der zu transportierenden Bytes gerade ist, ansonsten findet byteweiser Zugriff statt.

init

Initialisierungsdaten bzw. Hardwaredaten in die Hardware (TRC2-Modul) schreiben. Dieser Schritt muß nach dem Einlesen einer *.ini Datei und nach jeder Änderung der Einstellungen mit "edit" erfolgen.

start

Datennahme starten.

stop

Datennahme stoppen.

get_data

Meßwerte aus den TRC2-IP-Modulen auslesen und in den Speicher schreiben.

last_value

Letzten Meßwert eines Kanals aus dem TRC2-IP-Modul auslesen, in die physikalische Einheit umrechnen und in der Standardausgabe anzeigen.

Serviceprogramme

4.4.1 Kommandoshell

- noch zu schreiben -

4.4.2 Java-Applet

- noch zu schreiben -

5. Andere Betriebssysteme

- noch zu schreiben -

Prüfung der DTU

Prüfvorschrift "Tastkopfanschluss"

Die einwandfreie Funktion der Eingangs- und Ausgangstreiber soll festgestellt werden. Dazu ist es notwendig Daten vom Tastkopf zu lesen und zu schreiben mit folgender Vorgehensweise:

Analogen Tastkopf auf den zu prüfenden Kanal stecken.

5V Gleichspannung anlegen.

Tastkopf auf 30V-Messbereich einstellen.

Messwert einlesen.

3V Gleichspannung anlegen.

Tastkopf auf 10V-Messbereich einstellen.

Messwert einlesen.

Wenn die eingelesenen Messwerte mit den angelegten Spannungen übereinstimmen ist die Prüfung erfolgreich verlaufen.

Prüfvorschrift "TRIGGER"

Impulsgenerator an "TRIGGER IN" anschliessen.

Schalter "TRIGGER IN" in Stellung "DIRECT" bringen.

IP-Modul auf externen Trigger einstellen.

Messung starten.

Register "rx_address" auslesen und merken.

Positiven TTL-Impuls auslösen.

Register "rx_address" erneut auslesen. Wenn der Registerinhalt sich um den Wert 1 vergrössert hat ist der Anschluss "TRIGGER IN" in Ordnung. Ein Überlauf des Registerinhalts von 0x1FFF (dezimal 8191) nach 0x0000 ist auch zulässig.

Schalter "TRIGGER IN" in Stellung ":110" bringen.

Register "rx_address" auslesen und merken.

Positiven TTL-Impuls 110 mal auslösen.

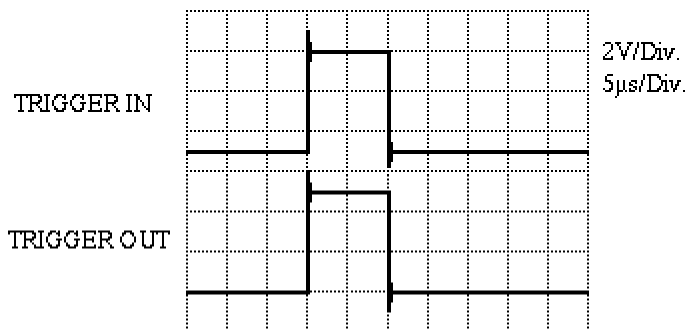
Register "rx_address" erneut auslesen. Wenn der Registerinhalt sich um den Wert 1 vergrössert hat ist der Schalter "TRIGGER IN" in Ordnung. Ein Überlauf des Registerinhalts von 0x1FFF (dezimal 8191) nach 0x0000 ist auch zulässig.

Schalter "TRIGGER IN" in Stellung "DIRECT" bringen.

Positiven TTL-Impuls auslösen. Wenn der TTL-Impuls am Anschluss "TRIGGER OUT" erscheint ist der Anschluss "TRIGGER OUT" in Ordnung.

Hinweise zur Funktion:

Die Triggersignale sind high-aktiv. Wenn der Schalter "TRIGGER IN" in Stellung "DIRECT" steht wird das Signal, dass am Eingang "TRIGGER IN" anliegt unverändert am Ausgang "TRIGGER OUT" ausgegeben. Eine Bearbeitung des Signals findet nicht statt.



Wenn der Schalter "TRIGGER IN" in Stellung ":110" steht wird der Ausgang "TRIGGER OUT" mit dem Eintreffen jedes 110. Impulses am Eingang "TRIGGER OUT" auf High-Signal (5V) geschaltet und mit dem Eintreffen des nächsten Impulses wieder auf Low-Signal (0V) zurückgesetzt.

Prüfvorschrift "STOP"

Impulsgenerator an "STOP IN" anschliessen.

IP-Modul auf internen Trigger einstellen.

IP-Modul auf ioStop=enable einstellen.

IP-Modul auf io_stop_syn=enable einstellen.

Messung starten. Das IP-Modul befindet sich jetzt im Betriebsmodus "Data taking". Die beiden Bits D6 und D7 des Statusregisters sind auf 1 gesetzt.

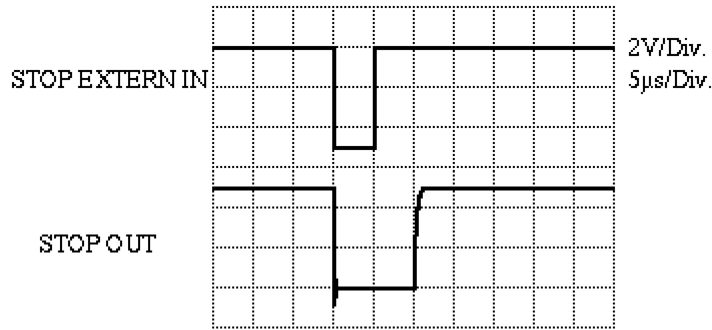
Negativen TTL-Impuls auslösen.

Wenn der TTL-Impuls am Anschluss "STOP EXTERN OUT" erscheint ist der Anschluss "STOP EXTERN OUT" in Ordnung.

Statusregister auslesen. Wenn im Statusregister das Bit D6 weiterhin auf 1 gesetzt ist und das Bit D7 jetzt auf 0 gewechselt hat ist der Anschluss "STOP OUT" in Ordnung.

Hinweise zur Funktion:

Die Stopsignale sind low-aktiv. Das Signal "STOP OUT" wird vom IP-Kärtchen gebildet, daher ist die Impulsbreite (ca. 10 Mikrosekunden) des Signals am Ausgang "STOP OUT" unabhängig von der Breite des Eingangsimpulses am Anschluss "STOP EXTERN IN".



Prüfung des analogen Tastkopfes

Gleichspannung im Bereich 10V bis 30V anlegen.

Tastkopf auf Messbereich=30V, Bandbreite=200kHz und Testvoltage=off einstellen.

Messwert einlesen. Der Messwert muss mit der angelegten Spannung übereinstimmen.

Tastkopf auf Messbereich=10V, Bandbreite=200kHz und Testvoltage=off einstellen.

Messwert einlesen. Der 10V-Bereich wird überschritten. Der Messwert muss genau 10.00V betragen.

Gleichspannung im Bereich 1V bis 10V anlegen.

Messwert einlesen. Der Messwert muss mit der angelegten Spannung übereinstimmen.

Tastkopf auf Messbereich=1V, Bandbreite=200kHz und Testvoltage=off einstellen.

Messwert einlesen. Der 1V-Bereich wird überschritten. Der Messwert muss genau 1.00V betragen.

Gleichspannung im Bereich 100mV bis 1V anlegen.

Messwert einlesen. Der Messwert muss mit der angelegten Spannung übereinstimmen.

Tastkopf auf Messbereich=100mV, Bandbreite=200kHz und Testvoltage=off einstellen.

Messwert einlesen. Der 100mV-Bereich wird überschritten. Der Messwert muss genau 0.100V betragen.

Gleichspannung im Bereich 0 bis 100mV anlegen.

Messwert einlesen. Der Messwert muss mit der angelegten Spannung übereinstimmen.

Punkte 1 bis 15 mit negativen Spannungen wiederholen.

Prüfung des digitalen Tastkopfes

11V Gleichspannung anlegen.

Tastkopf auf Level=24V und Mode=sample einstellen.

Messwert zum ersten Mal einlesen. Das Bit muss 1 sein.

Messwert ein zweites Mal einlesen. Das Bit muss 1 sein.

5V Gleichspannung anlegen.

Messwert zum ersten Mal einlesen. Das Bit muss 0 sein.

Messwert ein zweites Mal einlesen. Das Bit muss 0 sein.

Tastkopf auf Level=5V und Mode=sample einstellen.

Messwert zum ersten Mal einlesen. Das Bit muss 1 sein.

Messwert ein zweites Mal einlesen. Das Bit muss 1 sein.

0V Gleichspannung anlegen.

Messwert zum ersten Mal einlesen. Das Bit muss 0 sein.

Messwert ein zweites Mal einlesen. Das Bit muss 0 sein.

Tastkopf auf Level=5V und Mode=edge einstellen.

Messwert zum ersten Mal einlesen. Das Bit muss 0 sein.

Messwert ein zweites Mal einlesen. Das Bit muss 0 sein.

5V Gleichspannung anlegen.

Messwert zum ersten Mal einlesen. Das Bit muss 1 sein.

Messwert ein zweites Mal einlesen. Das Bit muss 0 sein.

0V Gleichspannung anlegen.

Messwert zum ersten Mal einlesen. Das Bit muss 1 sein.

Messwert ein zweites Mal einlesen. Das Bit muss 0 sein.

5V Gleichspannung nur kurz anlegen.

Messwert zum ersten Mal einlesen. Das Bit muss 1 sein.

Messwert ein zweites Mal einlesen. Das Bit muss 0 sein.

Tastkopf auf Level=24V und Mode=edge einstellen.

Messwert zum ersten Mal einlesen. Das Bit muss 0 sein.

Messwert ein zweites Mal einlesen. Das Bit muss 0 sein.

11V Gleichspannung anlegen.

Messwert zum ersten Mal einlesen. Das Bit muss 1 sein.

Messwert ein zweites Mal einlesen. Das Bit muss 0 sein.

5V Gleichspannung anlegen.

Messwert zum ersten Mal einlesen. Das Bit muss 1 sein.

Messwert ein zweites Mal einlesen. Das Bit muss 0 sein.

11V Gleichspannung nur kurz anlegen.

Messwert zum ersten Mal einlesen. Das Bit muss 1 sein.

Messwert ein zweites Mal einlesen. Das Bit muss 0 sein.