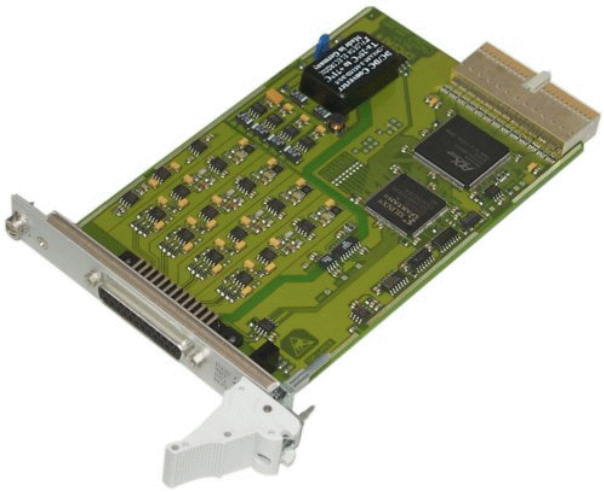
CPCI-AI4

4 analoge Eingänge, 16 Bit



Hardware-Handbuch

Dokument-Datei:	M:\texte\Doku\MANUALS\CPCI\AI4\cpci-ai4_20h.ma9
Datum des Ausdrucks:	18.02.2003

Platinenversion:	CPCI-AI4 Rev. 1.2

Änderungen in den Kapiteln

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

Kapitel	Änderungen gegenüber Vorversion
-	Handbuch der neuen Platinen-Revision angepasst.
-	

Weitere technische Änderungen vorbehalten.

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

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

esd electronic system design gmbh
Vahrenwalder Str. 207
30165 Hannover

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

Inhalt

Diese Seite ist bewusst unbedruckt.

1. Übersicht

1.1 Beschreibung der CPCI-AI4

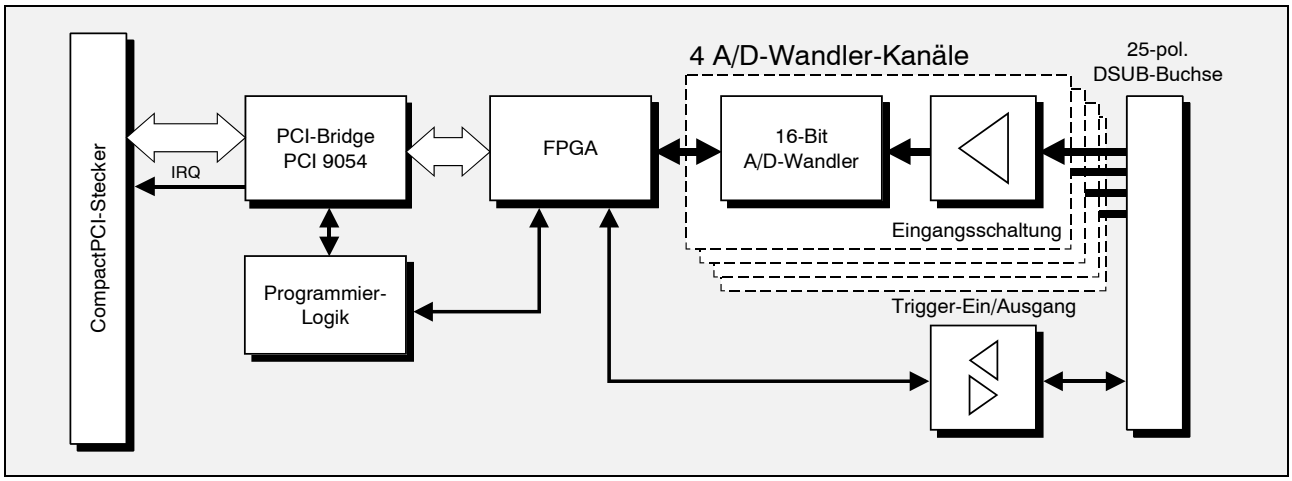


Abb. 1.1.1: Blockschaltbild der CPCI-AI4

Die CPCI-AI4 ist eine CompactPCI- Karte im Euro-Format. Sie besitzt vier analoge Eingänge mit einer Auflösung von 16 Bits. Der Eingangsspannungsbereich beträgt ±10 V.
Die vier A/D-Wandler werden von einem FPGA gesteuert, der automatisch über die CompactPCI- Bus-Software initialisiert wird.

Die Eingänge werden kontinuierlich gewandelt. Der Trigger wird über einen DDFS Timer (Direct Digital Frequency Synthesis) generiert, der in weiten Bereichen frei programmierbar ist.

Alternativ kann über ein externes Trigger-Signal die Wandlung gestartet werden. Das Triggerport ist bidirektional, so dass die A/D-Wandlung weiterer CPCI-AI4-Karten gleichzeitig gestartet werden kann.

1.2 Platinenansicht mit Steckerbezeichnung

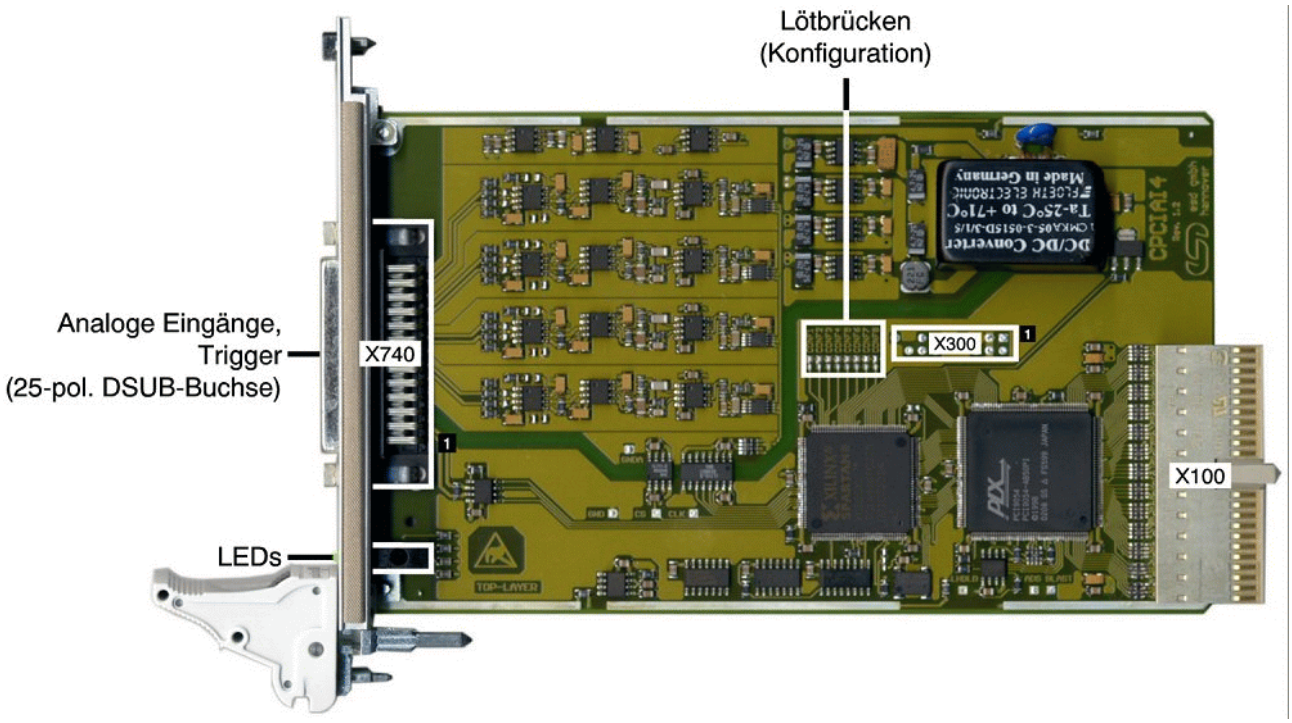


Abb. 1.2.1: Ansicht der Platine

1.3 Zusammenfassung der technischen Daten

1.3.1 Allgemeine technische Daten

Umgebungstemperatur	0...50 ˚C, optional auch für -40 ˚C...+85 ˚C erhältlich
Luftfeuchtigkeit	max. 90 %, nicht kondensierend
Versorgungsspannung	3.3 V und 5 V
Steckverbinder	X100 (132-pol. Pfostenstecker) - CompactPCI-Board-Connector X740 (25-pol. DSUB-Buchse) - analoge Eingänge und Trigger X300 (16-pol. Pfostenstecker) - nur für FPGA-Testzwecke bestückt
Status LEDs	4
Abmessungen	100 mm x 160 mm (ohne Frontplatte und ohne Stecker)
Gewicht	< 200 g

Tabelle 1.3.1: Allgemeine Daten der CPCI-AI4

1.3.2 CompactPCI-Interface

Host-Bus	PCI-Bus gemäß PCI Local Bus Specification 2.1
PCI-Daten/Adressbus	32 Bit
Controller	PCI9054
Interrupt	Interrupt-Signal A
Board-Dimension	gemäß CompactPCI-Specification, Rev. 1.0
Steckverbinder	
Stecker-Kodierung	keine Farbkodierung Universal Board (3.3 V oder 5 V Signalspannung)

Tabelle 1.3.2: CompactPCI-Interface

1.3.3 A/D-Konverter Einheit

Anzahl der Kanäle	4
A/D-Konverter Auflösung	16 Bits
Eingangsspannung	±10 V
Konvertierungsrate	bis zu 96000 Wandlungen/s (4 Kanäle synchron)
Trigger Ein- / Ausgang	RS485-kompatibler Transceiver, parallele Triggerung von verschiedenen Karten möglich

Tabelle 1.3.3: Analoge Eingänge und Triggerports des CPCI-AI4-Moduls

1.3.4 Software-Unterstützung

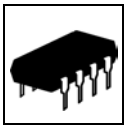
Zur Ansteuerung der Karte ist ein VxWorks-Treiber als Object-Code lieferbar. Er wird in einem eigenen Handbuch beschrieben. Weitere Treiber sind auf Anfrage lieferbar.

1.3.5 Bestellhinweise

Typ	Eigenschaften	Bestell-Nr.
CPCI-AI4	4 analoge Eingänge, 16 Bits 0...+50 ˚C	I.2302.02
CPCI-AI4-T	erweiterter Temperaturbereich -40...+85 ˚C	I.2302.03
CPCI-AI4-VxW	VxWorks-Objekt-Lizenz	I.2302.55
CPCI-AI4-MD *)	Anwenderhandbuch in deutsch	I.2302.20

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

Tabelle 1.3.4: Bestellhinweise



2. Beschreibung der Baugruppen

2.1 Frontplatte und LED-Anzeige

Das Modul ist mit vier LEDs in der Frontplatte versehen.

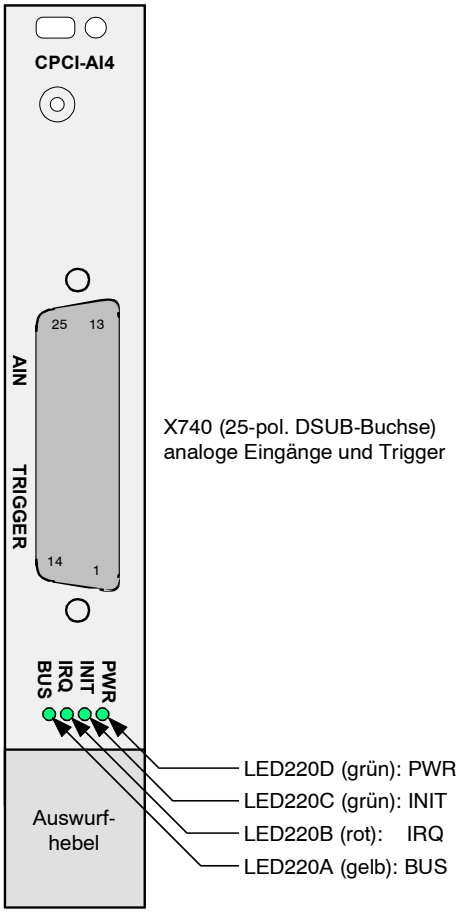


Abb. 2.1.1: Position und Farben der LEDs

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

Tabelle 2.1.1: Anzeigefunktionen der LEDs

2.2 Schaltung der analogen Eingänge

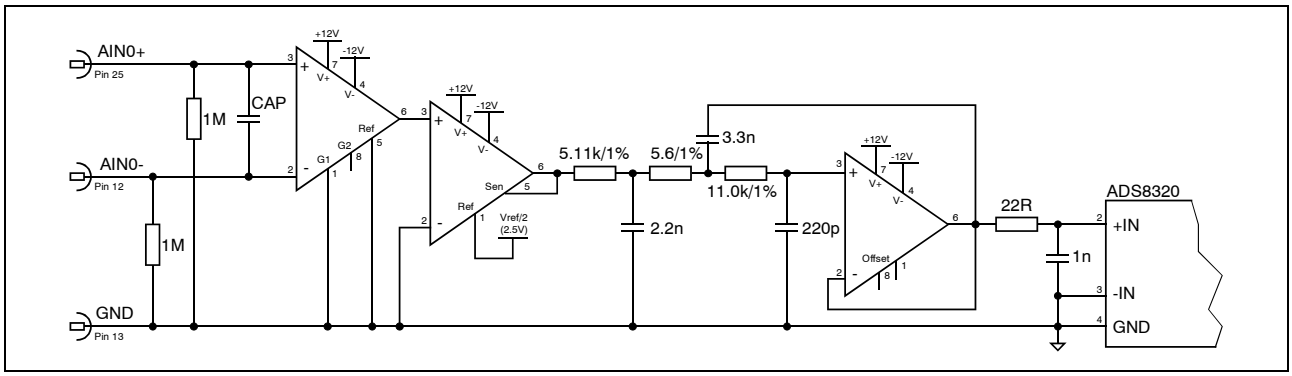


Abb. 2.2.1: Schaltung der analogen Eingänge (Beispiel: Kanal 1)

2.3 Schaltung des Trigger-Ports

Das Trigger-Port ist mit einem RS485-kompatiblen Treiberbaustein ausgestatttet.

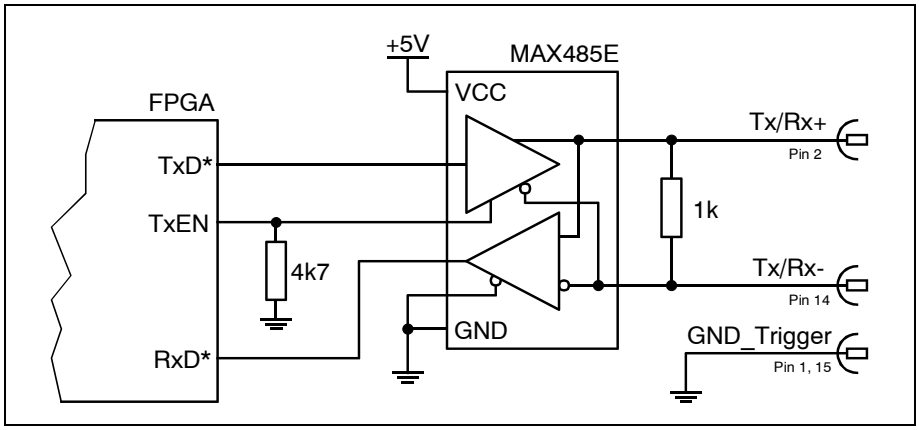


Abb. 2.3.1: Schaltung des Trigger-Ports

2.4 Konfigurationslötbrücken

Die acht Konfigurationslötbrücken dürfen vom Anwender nicht verändert werden.

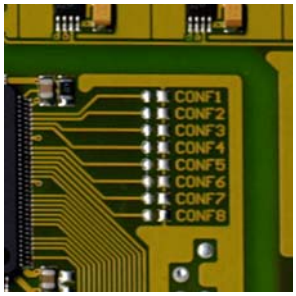


Abb. 2.4.1: Konfigurationslötbrücken CONF1...CONF8

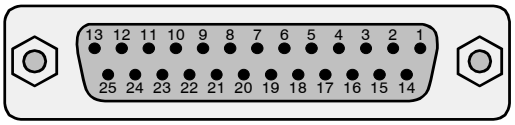
3. Steckerbelegung

3.1 Analoge Eingänge und Trigger auf X740

Gerätestecker: 25-pol. DSUB, Buchsenkontakte,
z.B. Hersteller Harting, Best.-Nr.: 09 66 352 661 3

Leitungsstecker: 25-pol.DSUB, Stiftkontakte

Pin Zuordnung:

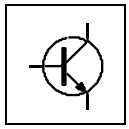


Pin Belegung:

Pin-Nr.	Signal
1	GND_Trigger
2	Rx/Tx+ (Trigger)
3	-
4	-
5	-
6	AIN3+
7	GND
8	AIN2+
9	GND
10	AIN1+
11	GND
12	AIN0+
13	GND

Pin-Nr.	Signal
14	Rx/Tx- (Trigger)
15	GND_Trigger
16	-
17	-
18	GND
19	AIN3-
20	GND
21	AIN2-
22	GND
23	AIN1-
24	GND
25	AIN0-

- ... nicht angeschlossen



4. Stromlaufpläne

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

CPCI-AI4

4 analoge Eingänge, 16 Bit

Software-Handbuch

Handbuch-Datei:	M:\texte\Doku\MANUALS\CPCI\AI4\cpci-ai4_11s.ma9
Handbuch-Bestellnummer:	C.2302.20
Datum der Druck-vorlagenerstellung:	18.02.2003

Software-Bestellnummer:	C.2302.55

Änderungen in der Software und/oder der Dokumentation

Änderung in diesem Handbuch gegenüber der Vorversion	Änderung in der Software	Änderung in der Dokumentation
Handbuch neu erstellt.	x	x
-	-	-

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

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

esd electronic system design gmbh
Vahrenwalder Str. 207
30165 Hannover

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

Inhalt

Seite

Diese Seite ist bewusst unbedruckt.

1. Einleitung

Dieses Handbuch beschreibt die VxWorks-Treibersoftware des CompactPCI-Moduls CPCI-AI4.

Im ersten Abschnitt werden die Funktionen beschrieben. Im zweiten Abschnitt ist ein Beispielprogramm angefügt, in dem die Funktionen Verwendung finden.

2. Starten und Beenden des Treibers

Zum **Starten** des Treibers ist

```
ai4Start
```

einzugeben.

Zum **Stoppen** des Treibers ist

```
ai4Stop
```

einzugeben.

3. Funktionsbeschreibungen

3.1 Konfiguration

ai4Open()

Name: **ai4Open()** - Erzeugen eines Handles für Lese- und Schreiboperationen

Aufruf:
```
int     ai4Open
        (
        int         card,       /* CPCI-Karten-Nummer */
        HANDLE      *hnd        /* Output:  Handle */
        )
```

Beschreibung: Diese Funktion gibt ein Handle für nachfolgende I/O-Aufrufe zurück. Es können maximal 1024 Handles geöffnet werden.

> *ai4Open* ist als **erster** Funktionsaufruf auszuführen, da alle anderen Funktionen das *Handle* benötigen !

Eingabe: *card:* Nummer der CPCI-AI4-Karte

Rückgabe: Nach erfolgreichem Funktionsaufruf wird ein Handle zurückgegeben.

Header: ai4api.h

---

ai4Close()

Name: **ai4Close()** - Schließen eines Handles für Lese- und Schreibzugriffe

Aufruf:
```
int     ai4Close
        (
        HANDLE      hnd         /* Eingabe: Handle */
        )
```

Beschreibung: Diese Funktion gibt ein Handle mit allen belegten Ressourcen frei.

Eingabe: *hnd:* Handle, das von *ai4Open* zurückgegeben wurde.

Rückgabe: Bestätigung / Fehlermeldungen

Header: ai4api.h

ai4Config()

Name: **ai4Config ()** - Initialisierung der Karte

Aufruf:

```
int     ai4Config
        (
        HANDLE      hnd,        /* Eingabe: Handle */
        int         mode,       /* Parameter-Typ */
        int         para        /* Parameter-Wert */
        )
```

Beschreibung: Diese Funktion dient zur Freigabe des Interrupts und zur Einstellung der Abtastrate.

Eingabe:

hnd: Handle, das von *ai4Open* zurückgegeben wurde.

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

mode	*para*	Bedeutung
1	0	Interrupt gesperrt (disabled)
	1	Interrupt freigegeben (enabled)
10	0	keine Abtastung
	1...96000	Abtastrate in Hz (1 Hz ...96 kHz)

Tabelle 3.1.1: Werte der Konfigurationsparameter

Rückgabe: Bestätigung / Fehlermeldungen

Header: ai4api.h

3.2 Lesen der Analogdaten

Datenformat

Die CPCI-AI4 kann Spannungen zwischen -10V und +10V mit einer Auflösung von 16 Bit messen. Die Kodierung der Messwerte ist wie folgt:

Spannung	Messwert [Hex]
-10 V	8000
:	:
0	0
:	:
+10 V - 1 LSB	7FFF

Tabelle 3.2.1: Kodierung der Messwerte

ai4Read()

Name: **ai4Read()** - Lesen des A/D-Daten-Buffers

Aufruf:

```
int     ai4Read
        (
        HANDLE      hnd,        /* Handle */
        int         mode,       /* zu lesende Kanäle */
        void        *buffer     /* Pointer auf Daten-Buffer */
        )
```

Beschreibung: Mit *ai4Read* wird der Daten-Buffer der Analogdaten transparent gelesen. Er enthält die zuletzt gewandelten Daten.
Diese Funktion führt nicht zum Starten einer A/D-Wandlung! Dies ist nur mit den Funktionen *ai4IntRead* und *ai4DmaRead* möglich (siehe folgende Kapitel).

Eingabe:

hnd: Handle, das von *ai4Open* zurückgegeben wurde.
mode: Auswahl der zu lesenden A/D-Kanäle:
0 alle vier Kanäle werden gelesen
1 nur Kanal 0 und 1 werden gelesen
2 nur Kanal 2 und 3 werden gelesen
**buffer:* Pointer auf Daten-Buffer

Rückgabe: Bestätigung / Fehlermeldungen

Header: ai4api.h

ai4IntRead()

Name: **ai4IntRead()** - Interrupt-gesteuertes Lesen der A/D-Daten

Aufruf:

```
int     ai4IntRead
        (
        HANDLE      hnd,        /* Handle */
        int         mode,       /* zu lesende Kanäle */
        void        *buffer     /* Pointer auf Daten-Buffer */
        )
```

Beschreibung: Mit *ai4IntRead* werden die Analogdaten Interrupt-gesteuert gelesen. Vor dem Lesen muss die Karte zunächst mit der gewünschten Abtastfrequenz konfiguriert und der Interrupt freigeschaltet werden.
Die Konfiguration wird mit der Funktion *ai4Config* durchgeführt (siehe Seite 5). Es wird ein Timer initialisiert, der die Wandlung startet und, nachdem die Daten im Buffer abgelegt sind, einen Interrupt auslöst.
Die Funktion liest die Daten aus dem Buffer, wenn der Interrupt empfangen wurde.

Eingabe:

hnd: Handle, das von *ai4Open* zurückgegeben wurde.
mode: Auswahl der zu lesenden A/D-Kanäle:
- 0 alle vier Kanäle werden gelesen
- 1 nur Kanal 0 und 1 werden gelesen
- 2 nur Kanal 2 und 3 werden gelesen

**buffer:* Pointer auf Daten-Buffer

Rückgabe: Bestätigung / Fehlermeldungen

Header: ai4api.h

ai4DmaRead()

Name: **ai4DmaRead()** - Interrupt-gesteuertes Lesen der A/D-Daten

Aufruf:

```
int     ai4DmaRead
        (
        HANDLE      hnd,        /* Handle */
        int         mode,       /* zu lesende Kanäle */
        void        *buffer,    /* Pointer auf Daten-Buffer */
        int         size        /* Anzahl der zu lesenden Bytes */
        )
```

Beschreibung: Mit *ai4DmaRead* werden die Analogdaten per DMA-Zugriff gelesen. Im Gegensatz zu *ai4IntRead* wird nicht nach jeder Wandlung ein Interrupt ausgelöst. Die Analogdaten werden zunächst in einem Buffer abgelegt. Die gewünschte Größe des Buffers wird über den Parameter *size* angegeben. Nachdem das letzte Byte in den Buffer geschrieben ist, wird ein Interrupt ausgelöst.

Vor dem Lesen muss die Karte zunächst mit der gewünschten Abtastfrequenz konfiguriert werden.
Die Konfiguration wird mit der Funktion *ai4Config* durchgeführt. Es wird ein Timer initialisiert, der die Wandlung startet und, nachdem die Daten im Buffer abgelegt sind, einen Interrupt auslöst.

Eingabe:

hnd: Handle, das von *ai4Open* zurückgegeben wurde.
mode: Auswahl der zu lesenden A/D-Kanäle:
- 0 alle vier Kanäle werden gelesen
- 1 nur Kanal 0 und 1 werden gelesen
- 2 nur Kanal 2 und 3 werden gelesen

**buffer:* Pointer auf Daten-Buffer
size: Anzahl der zu lesenden Bytes (Buffer muss mindestens die Größe haben, die in *size* angegeben ist)

Rückgabe: Bestätigung / Fehlermeldungen

Header: ai4api.h

4. Beispielprogramm

```
/*****************************************************************************/
/*                        Testprogram for esd CPCIAI4                        */
/*                                                                           */
/*              (c) electronic system design gmbh hannover 1999              */
/*****************************************************************************/
/*
 * ai4test.c - Testprogram for the esd cpciai4-board
 *
 * History:
 *
 *  $Log: ai4test.c,v $
 *  Revision 1.3  1999/08/03 15:42:31  stefan
 *  api changed to handle instead of card-no
 *  check for fpga-booting included
 *  fpga-image-file in cvs included
 *
 *  Revision 1.2  1999/07/30 12:33:01  stefan
 *  first release version
 *
 *  Revision 1.1  1999/07/22 08:21:56  stefan
 *  added to cvs tree
 *
 *
 */

/* Includes */

#include <vxWorks.h>
#include <sysLib.h>
#include <intLib.h>
#include <iv.h>
#include <semLib.h>
#include <vxLib.h>
#include <taskLib.h>
#include <vmLib.h>
#include <tickLib.h>

#include <errno.h>
#include <string.h>
#include <stdio.h>

#include "ai4api.h"


#define BUFFER_MAX 1000000



/*
 * ai4ReadTest:
 *
 */
int ai4ReadTest(void)
{
  int status;
  unsigned short buffer[4];
  HANDLE hnd;

  status = ai4Open(0, &hnd);
  printf("ai4Open returns %d\n", status);
  if (status != 0)
    {
      printf("Aborted!\n");
      return -1;
    }
```

```
  status = ai4Read(hnd, CHANNEL_0123, buffer);
  printf("value=0x%04x 0x%04x 0x%04x 0x%04x\n",
         buffer[0], buffer[1], buffer[2], buffer[3]);

  status = ai4Close(hnd);
  return 0;
}


/*
 * ai4IntReadTest:
 *
 */
int ai4IntReadTest(void)
{
  int status;
  int i;
  unsigned short buffer[4];
  HANDLE hnd;

  status = ai4Open(0, &hnd);
  printf("ai4Open returns %d\n", status);
  if (status != 0)
    {
      printf("Aborted!\n");
      return -1;
    }

  /* set 1khz sample-frequency... */
  status = ai4Config(hnd, CONFIG_SET_FREQUENCY, 1000);
  /* enable interrupt on timer... */
  status = ai4Config(hnd, CONFIG_SET_IRQ, ON);

  for (i=0; i<10; i++)
    {
      status = ai4IntRead(hnd, CHANNEL_0123, buffer);
      printf("value=0x%04x 0x%04x 0x%04x 0x%04x\n",
             buffer[0], buffer[1], buffer[2], buffer[3]);
    }

  /* disable interrupt on timer... */
  status = ai4Config(hnd, CONFIG_SET_IRQ, OFF);

  status = ai4Close(hnd);
  return 0;
}


/*
 * ai4DmaReadTest:
 *
 */
int ai4DmaReadTest(void)
{
  int status;
  int i, l;
  int t1, t2;
  unsigned long *buffer;
  unsigned short *ptr;
  HANDLE hnd;

  status = ai4Open(0, &hnd);
  printf("ai4Open returns %d\n", status);
  if (status != 0)
    {
      printf("Aborted!\n");
      return -1;
    }
```

```
  buffer = (unsigned long *)malloc(BUFFER_MAX*8);

  /* fill buffer... */
  for (i=0; i<BUFFER_MAX; i++)
    buffer[i] = 0;

  /* set 1khz sample-frequency... */
  status = ai4Config(hnd, CONFIG_SET_FREQUENCY, 1000);

  t1 = (int)tickGet();
  /* read 1000 * 8 bytes via dma to buffer... */
  status = ai4DmaRead(hnd, CHANNEL_0123, buffer, 1000*8);
  t2 = (int)tickGet();

  printf("ai4DmaRead returns %d; duration: %f seconds\n", status,
         (double)(t2-t1)/(double)sysClkRateGet());

  ptr = (unsigned short *)buffer;

  for (l=0; l<7; l++)
    {
      printf("%02x: ", l*0x10);
      for (i=0; i<8; i++)
        printf("%04x ", *ptr++);
      printf("\n");
    }
  printf("\n");

  for (i=0; i<BUFFER_MAX; i++)
    {
      if (buffer[i] == 0)
        {
          printf("End of DMA at Index %d\n", i);
          break;
        }
    }

  free(buffer);

  status = ai4Close(hnd);
  return 0;
}
```