CPCI-DIO72

CompactPCI-digital I/O-Board

Hardware Installation
and
Technical Data

Document file:	I:\texte\Doku\MANUALS\CPCI\DIO72\CPI7212H.en9
Date of print:	20.12.2001

PCB-version:	CPCI-DIO72 Rev. 1.1

Changes in the chapters

The changes in the document listed below affect changes in the hardware as well as changes in the description of facts only.

Chapter	Changes versus previous version
5.	Additional types of connector X400.
-	-

Technical details are subject to change without further notice.

N O T E

The information in this document has been carefully checked and is believed to be entirely reliable. **esd** makes no warranty of any kind with regard to the material in this document, and assumes no responsibility for any errors that may appear in this document. **esd** reserves the right to make changes without notice to this, or any of its products, to improve reliability, performance or design.

esd assumes no responsibility for the use of any circuitry other than circuitry which is part of a product of **esd** gmbh.

esd does not convey to the purchaser of the product described herein any license under the patent rights of **esd** gmbh nor the rights of others.

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

Phone: +49-511-372 98-0
Fax: +49-511-372 98-68
E-mail: info@esd-electronics.com
Internet: www.esd-electronics.com

USA / Canada
7667 W. Sample Road
Suite 127
Coral Springs, FL 33065
USA

Phone: +1-800-504-9856
Fax: +1-800-288-8235
E-mail: sales@esd-electronics.com

Contents

This page is intentionally left blank.

1. Overview

1.1 Description of the CPCI-DIO72-Board

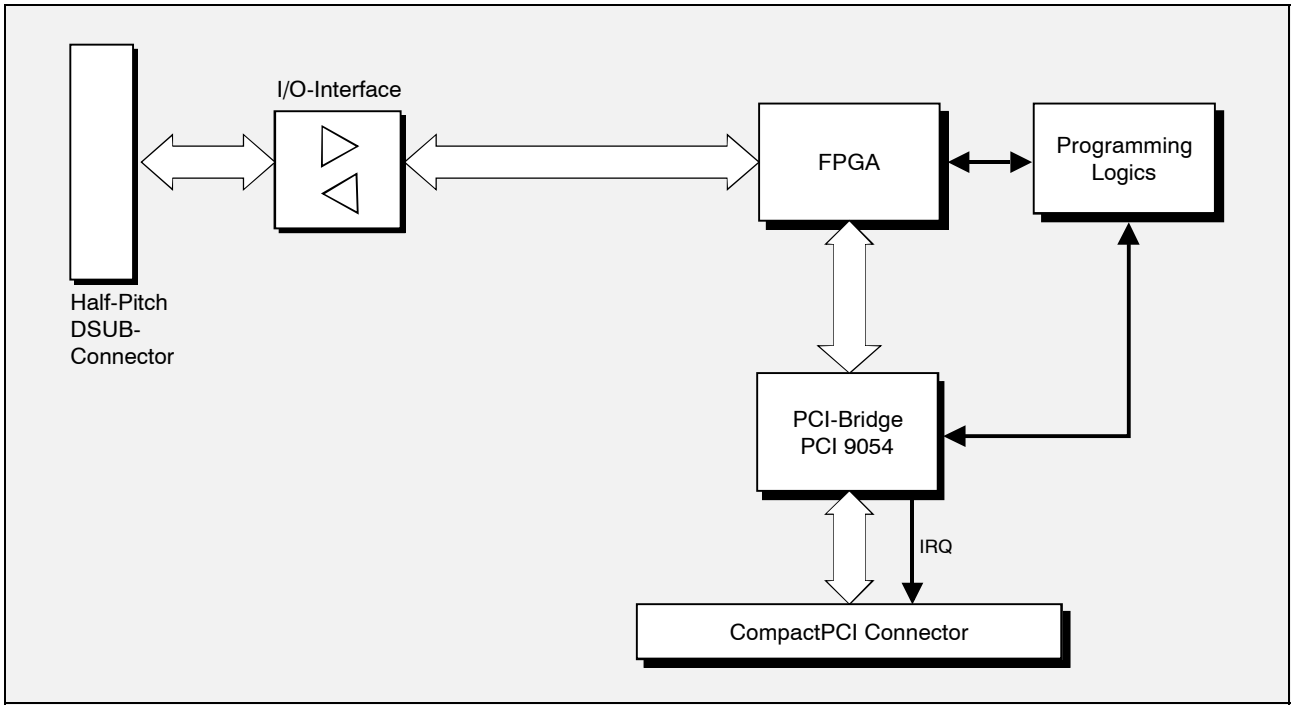


Fig. 1.1: Block-circuit diagram of the CPCI-DIO72-module

The CPCI-DIO72-board is a digital I/O-board which can be universally used in the CompactPCI- system. It mainly consists of the components input circuit, an FPGA as an I/O-controller and a PCI-bridge.

The board is connected to the Compact-PCI-bus via the PCI-Bridge PCI9054 by PLX Technology, which is bus-master capable and is therefore able to initiate a PCI-cycle independently.
An FPGA of the Spartan family by Xilinx is used as a programmable logic component. Very complex functions with a high depth in logic can be realised with this component.

1.2 PCB-View with Connector Designation

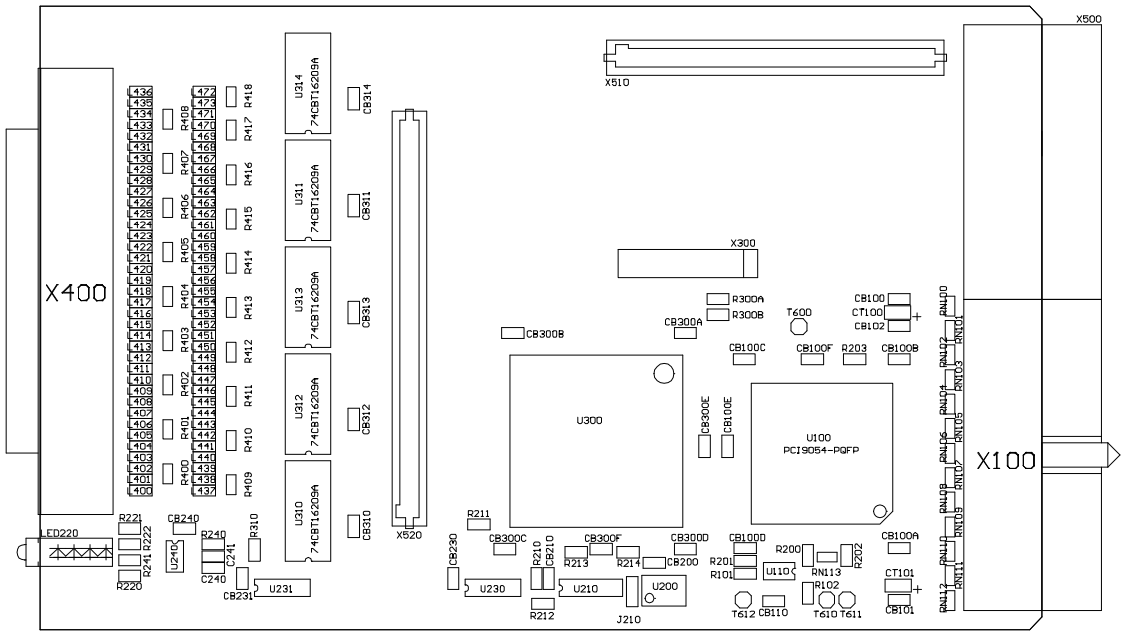


Fig. 1.2: Module view (represented without front panel)

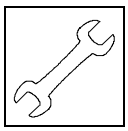
1.3 Switching Inputs and Outputs

For the flexible switching of directions of the 72 channels, bus exchange switches are used on the CPCI-DIO72-board. These components have got FET pass transistors and do therefore not have driver abilities and no predetermined signal direction. A switching of signal directions is therefore not necessary.

The input circuit additionally offers a safety function from positive or negative over voltage caused by internal clamp diodes.
Furthermore, all signals of the 80-pin socket in the front panel of the CPCI-DIO72-board are in high-impedance status as long as the CompactPCI system is not switched on and the FPGA has not been programmed. This high-impedance status can also be triggered by the software.

In each I/O-line a coil and a resistor are connected in series, situated close to the socket. The coil is used to suppress high-frequency disturbances, while the resistor mutes the vibration tendency of the input line.

This page is intentionally left blank.



2. Hardware Installation

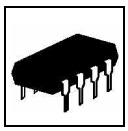
Attention!
Electro-statical discharges can cause damage to electronic components. In order to prevent this, please observe the following steps *before* touching the module to discharge the statical electricity of your body:

- Switch off the power supply of your computer, but leave it still connected to mains.
- Please touch the metal case of your computer now to discharge yourself.
- Furthermore you should avoid touching the module with your clothes, because these might be charged electro-statically as well.

Installation procedure:

1. Switch off your computer and all peripheral devices (such as monitors, printers, etc.).
2. Discharge the electro-statical electricity of your body as described above.
3. Disconnect your computer from mains.
4. Insert the CPCI-DIO72-module into a free CompactPCI-bus slot.
5. Fix the CPCI-DIO72-module with the front-panel screw provided.
6. Connect the I/O-signals to the half-pitch DSUB-connector X400 in the front panel.
7. Connect your computer to mains again.
8. Switch the computer and all other devices on again.
9. End of hardware installation.
10. Continue with the software installation (if you have not installed it already).

This page is intentionally left blank.



3. Summary of Technical Data

3.1 General Technical Data

Ambient temperature	0...50°C, also available for -40 ˚C...+85 ˚C
Humidity	max. 90 %, non-condensing
Power supply	via CompactPCI-bus, nominal voltages: 5 V ±5% 3.3 V ±5%
Connectors	X100 (132-pin post connector) - CompactPCI-board connector X400 (80-pin DSUB Half-Pitch) - angled mini DSUB-socket by AMP
Dimensions	100 mm x 160 mm
Weight	160 g

Table 3.1: General data of the module

3.2 CompactPCI-Bus

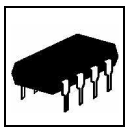
Host bus	PCI-bus in accordance with PCI Local Bus Specification 2.1
PCI-data/address bus	32 bits
Controller	PCI9054 by PLX Technologie
Interrupt	Interrupt signal A
Board dimension	in accordance with CompactPCI-Specification, Rev. 1.0
Connectors	

Table 3.2: CompactPCI-bus data

3.3 Digital Inputs and Outputs

Number of digital inputs and outputs	72 independent input or output channels
Configuration	divided into: 2 x 32 + 8 or 1 x 64 + 8 inputs and outputs
Input voltage	suitable for 3.3 V and 5 V signal voltages
Output voltage	U_{OUT} = 2.4 V (I_{OUT} = 7 mA)
Safety circuit	internal clamp diodes in the driver component
Connector s on board	X400 (80-pin DSUB Half-Pitch) - angled mini DSUB-socket by AMP
Interrupts	IRQ with rising or falling edge of an input
DMA-access	reading the inputs

Table 3.3: Digital inputs and outputs of the CPCI-DIO72-module



3.4 Software Support

Drivers are available as object codes for controlling the board. They offer function to configure and read the inputs and outputs. Furthermore, DMA-accesses to the inputs are supported.
The API will be described in the second part of the manual.

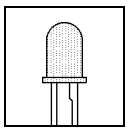
3.5 Order Information

Type	Features	Order No.
CPCI-DIO72	72 digital inputs and outputs, 0...+50 ˚C	I.2303.02
CPCI-DIO72-T	extended temperature range, -40...+85 ˚C	I.2303.03
CPCI-DIO72-RTOS-UH	RTOS-UH-Obj.-licence	I.2303.54
CPCI-DIO72-VxW	VxWorks-Obj.-licence	I.2303.55
CPCI-DIO72-ME *)	English manual	I.2303.21

*) If manual and product are ordered together, the manual is free of charge.

Table 3.5: Order information

This page is intentionally left blank.



4. LED-Display

The module has got four LEDs in the front panel.

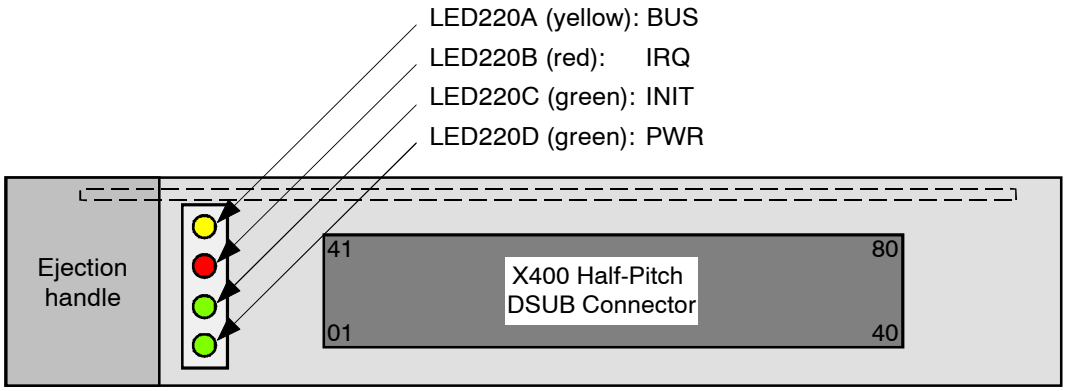


Fig. 4.1: Position and colours of the LEDs

LED	Colour	Name	Display function when LED is on
LED220A	yellow	BUS	Host CPU accesses board via PCI-bus
LED220B	red	IRQ	The board triggers a PCI-bus interrupt
LED220C	green	INIT	FPGA is programmed, board is functioning
LED220D	green	PWR	Power - power is supplied

Table 4.1: Display function of the LEDs

This page is intentionally left blank.

5. Connector Assignment

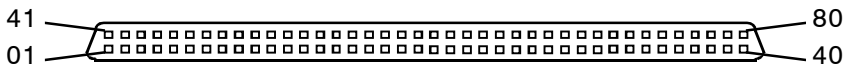
5.1 Assignment of I/O-Connector X400

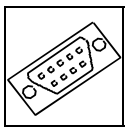
5.1.1 Connector Type

- connector at PCB: 80-pole half pitch male DSUB connector,
  AMP order no.: 749830-8 or 787190-8

- connector at wire: 80-pole half pitch female DSUB connector,
  AMP order no.: 749621-8 or 749111-7

5.1.2 Pin Assignment



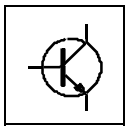


Connector Assignment

5.1.3 Signal Assignment

In addition to the 72 I/O-channels, the external synchronisation signal (LATCH), an output enabling signal (OUTEN) for channels 64-71, the earth potential of the I/O-board is provided here.

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



6. Circuit Diagrams

The PDF-file of this document does not contain the circuit diagrams. The circuit diagrams are shipped on request.

CPCI-DIO72-API

Software Manual

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

N O T E

The information in this document has been carefully checked and is believed to be entirely reliable. **esd** makes no warranty of any kind with regard to the material in this document, and assumes no responsibility for any errors that may appear in this document. **esd** reserves the right to make changes without notice to this, or any of its products, to improve reliability, performance or design.

esd assumes no responsibility for the use of any circuitry other than circuitry which is part of a product of **esd** gmbh.

esd does not convey to the purchaser of the product described herein any license under the patent rights of **esd** gmbh nor the rights of others.

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

Phone: +49-511-372 98-0
Fax: +49-511-372 98-68
E-mail: info@esd-electronics.com
Internet: www.esd-electronics.com

USA / Canada:
esd electronics Inc.
12 Elm Street
Hatfield, MA 01038-0048
USA

Phone: +1-800-732-8006
Fax: +1-800-732-8093
E-mail: us-sales@esd-electronics.com
Internet: www.esd-electronics.us

Manual file:	I:\texte\Doku\MANUALS\CPCI\DIO72\CPI7215S.en9
Manual order number:	I.2303.21
Date of print:	12.12.2005

Software driver	Software order number	described driver version
VxWorks	I.2303.55	not specified
RTOS-UH	I.2303.54	not specified
Linux	I.2303.19	I.2303.55

Changes in the software and/or the documentation

Chapter	Changes in this manual versus previous version
3.3	Configuration of the channel direction changed

This page is intentionally left blank.

Contents

Page

This page is intentionally left blank.

1. Introduction

This manual describes the VxWorks driver software of CompactPCI module CPCI-DIO72.

The first section describes the functions. The second section describes examples for application programming.

1.1 The Linux-Driver Software

A readme file with notes for the driver installation is in the scope of delivery.

Examples for the API-calls can be found in the enclosed file `io72test.c`.

1.2 The Program Interface of the Driver (API)

Below, the individual calls of the program interface will be described. The prototypes of the calls and the required constants are in the header file 'io72api.h'. When using the CPCI-DIO72 board, the header file must always be integrated into the source code. Write and read accesses into registers of the CPCI-DIO72 board are generally 32 bits wide.
All functions of the API convert Little Endianess of the PCI bus and Endianess of the CPU.

Register bits are assigned to channels of the CompactPCI-digital-I/O board as follows:

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

Table 1.1: Bit assignment

2. Starting and Stopping the Driver

In order to **start** the driver

```
io72Start
```

has to be specified.

In order to **stop** the driver

```
io72Stop
```

has to be specified.

3. Description of Functions

3.1 Initialization

io72Open()

Name: **io72Open()** - Generating a handle for read and write operations

Call:

```
int    io72Open
       (
       int          card,        /* CPCI-board number */
       HANDLE       *hnd         /* Output: handle */
       )
```

Description: This function returns a handle for following I/O-calls. Up to 1024 handles can be opened.
Via parameter *card* an index of the desired I/O-board is specified for this function. If the call was successful, a handle is returned to the CPCI-DIO72 board via the variable pointing to *hnd*. A counter in the driver database is increased.

> *io72Open* has to be called before a CPCI-DIO72 board is accessed for the first time. The returned *Handle* is needed for every further access to the CPCI-DIO72 board.

Parameter:
card: Number of CPCI-DIO72 board
**hnd*: A handle is returned after the function has been called successfully.

Return: '0', if the function was successful, or an error code (see table 3.7 on page 24).

Header: io72api.h

io72Close()

Name: **io72Close()** - Closing a handle for read and write accesses

Call:

```
int    io72Close
       (
       HANDLE    hnd              /* specification: handle */
       )
```

Description: *hnd* specifies the handle of a CPCI-DIO72 board to be enabled. The counter of the CPCI-DIO72 board is decremented. The handle is enabled with all assigned resources.

Specification: *hnd:* Handle which was returned by *io72Open.*

Return: '0', if the function was successful, or an error code (see table 3.7 on page 24).

Header: io72api.h

io72Reset()

Name: **io72Reset()** - Reset CPCI-DIO72 board to original status.

Call:

```
int    io72Reset
       (
       HANDLE    hnd              /* specification: handle */
       )
```

Description: The handle of the CPCI-DIO72 board is specified to this function by *hnd*. All registers of the CPCI-DIO72 board are reset to default status. Interrupts are blocked. The timer is stopped. All channels are configured as inputs.

Specification: *hnd:* Handle which was returned by *io72Open*.

Return: '0', if the function was successful, or an error code (see table 3.7 on page 24).

Header: io72api.h

3.2 Configuration

io72Config()

Name: **io72Config ()** - Configuration of the board function

Call:

```
int    io72Config
       (
       HANDLE    hnd,        /* specification: handle */
       int       mode,       /* parameter type */
       int       para        /* parameter value */
       )
```

Description: This function enables the interrupt and configures the sampling rate. *mode* represents the function to be configured. *para* is a configuration parameter and can be taken with *mode* from the following table.

Specification:

hnd: Handle which was returned by *io72Open.*

para: Parameter *para* is evaluated in dependence of parameter *mode.*

mode: *mode* determines, whether in *para* the value for the sampling rate of the digital inputs is specified, or whether the interrupt-controlled reading is enabled.

The following table represents the permissible values of *mode* and *para*:

Parameter		Meaning
mode	*para*	
CONFIG_INTERRUPT	INTERRUPT_ENABLE	Enable interrupt
	INTERRUPT_DISABLE	Disable interrupt
CONFIG_SET_FREQUENCY	Frequenz f in Hz	Set timer frequency.: 0 Hz no sampling 1...100 000 Hz sampling rate (1 Hz ...1 MHz)
CONFG_SET_TIMERVAL	Timerwert N	Set timer frequency by directly writing the timer register. The timer frequency f is calculated in operating mode *slow*: $f = \frac{N}{2^{(20+4)}} \cdot 20 \cdot 10^6 Hz$

Parameter		Meaning
mode	*para*	
CONFIG_SET_INTSOURCE (configure the interrupt source)	INSOURCE_TIMER	Set the timer; several sources can be used simultaneously. In order to do this the parameters have to be set to OR.
	INTSOURCE_EXT	External input
	INTSOURCE_EDGE	Detected edge
	INTSOURCE_NONE	Do not enable interrupt sources, do not enable interrupts
CONFIG_SET_DMASOURCE (configure the DMA-trigger source)	DMASOURCE_TIMER	Timer; several sources can be used simultaneously. In order to do this the parameters have to be set to OR. The maximum timer frequency is 2.5 MHz.
	DMASOURCE_EXT	External output
	DMASOURCE_EDGE	Detected edge
	DMASOURCE_NONE	No DMA-trigger source, do not enable a DMA-trigger
CONFIG_SET_CAPSOURCE (configure capture status for freezing the input data)	CAPSOURCE_TIMER	Timer excess; several sources can be used simultaneously. In order to do this the parameters have to be set to OR.
	CAPSOURCE_EXT	Edge at external synchronisation input
	CAPSOURCE_EDGE	Detected edge
	CAPSOURCE_NONE	Do not freeze inputs
CONFIG_TIMER	TIMER_START	Start timer
	TIMER_STOP	Stop timer
	TIMER_FAST	Increase timer frequency; the timer frequency is increased by factor 16
	TIMER_SLOW	Normal timer function; the frequency corresponds to the value set by CONFIG_SET_FREQUENCY
CONFIG_EXT (configuration of the external trigger connection)	EXT_ACTIVELOW	Input reacts to a falling edge
	EXT_ACTIVEHIGH	Inputs reacts to a rising edge
	EXT_INPUT	External trigger connection is an input
	EXT_OUTPUT	External trigger connection is an output. The capture status configured by means of CONFIG_SET_CAPSOURCE can now be sampled at the external trigger connection.

Table 3.2: Values of the configuration parameters

Return: '0', if the function was successful, or an error code (see page 24).

Header: io72api.h

3.3 Reading the Digital Inputs

io72Write()

Name: **io72Write()** - Writing into outputs or registers of the CPCI-DIO72 board

Call:

```
int    io72Write
       (
       HANDLE    hnd,        /* handle */
       int       mode,       /* inputs to be read */
       void      *buffer     /* pointer to data buffer */
       )
```

Description: This function writes data into outputs as well as into internal registers of the CPCI-DIO72 board.

Specification: *hnd:* Handle which was returned by *io72Open*.

mode: Selection of write access. Configures the data direction of the 72 I/O-channels and the edge detection. Permissible values for *mode* are represented in the following table:

Parameter *mode*	Validity	Meaning
CHANNEL_0_31	RW,INT, DMA	Access to inputs and outputs: 0..31, the bits set in the data represent a high level at the according input or output
CHANNEL_32_63	RW,INT, DMA	Access to inputs and outputs: 32...63
CHANNEL_0_63	RW,INT, DMA	Access to inputs and outputs: 0...63
CHANNEL_64_71	RW	Access to inputs and outputs: 64...71 Valid data is in the eight LSB.
DIRECTION_0_31	RW	Configuration of channel direction for channels: 0...31 Set bit '1' configures channel as output, deleted bit '0' configures channel as input, reading gives the last written value.
DIRECTION_32_63	RW	Configuration of channel direction for channels: 32...63
DIRECTION_0_63	RW	Configuration of channel direction for channels: 0...63
DIRECTION_64_71	RW	Configuration of channel direction for channels: 64...71
EDGE_0_31	RW,INT	When reading, set bits represent a detected edge at inputs 0...31. '1' resets the edge detection bit for this channel, '0' does not have any influence
EDGE_32_63	RW,INT	Detected edges at inputs 32...63

Parameter *mode*	Validity	Meaning
EDGE_0_63	RW,INT	Detected edges at inputs 0...63
LATCH_0_31	R	When the capture status appears, an image of the inputs is written into the 64-bit wide, internal capture register. Read capture register of inputs: 0...31
LATCH_32_63	R	Read capture register of inputs: 32...63
LATCH_0_63	R	Read capture register of inputs: 0...63
EDGE_ENABLES_0_31	RW	Enable edge detection. Bits set in these registers enable the edge detection for inputs 0...31.
EDGE_ENABLES_32_63	RW	Bits set in these registers enable the edge detection for inputs 32...63.
EDGE_ENABLES_0_63	RW	Bits set in these registers enable the edge detection for channels 0...63.
EDGE_IENABLES_0_31	RW	If bits are set in this register, an interrupt is triggered when the according edge appears. The edge detection has to be enabled additionally. Enable interrupts when edges appear at inputs 0...31.
EDGE_IENABLES_32_63	RW	Enable interrupts when edges appear at inputs 32...63.
EDGE_IENABLES_0_63	RW	Enable interrupts when edges appear at inputs 0...63.
EDGE_SELECT_0_31	RW	If bits are set in this register, rising edges are detected. Deleted bits configure the detection of falling edges. This register selects the kind of edge detection for inputs 0...31.
EDGE_SELECT_32_63	RW	This register selects the kind of edge detection for inputs 32...63.
EDGE_SELECT_0_63	RW	This register selects the kind of edge detection for inputs 0...63.

Permissible operations: R... read only, RW... read and write,
INT... interrupt-controlled reading, DMA...reading per DMA

Table 3.3: Values of configuration parameter *mode*

**buffer:* pointer to data buffer

Return: '0', if the function was successful, or an error code (see table 3.7 on page 24).

Header: io72api.h

io72MaskWrite()

Name: **io72MaskWrite() -** Masked writing into outputs or registers of the CPCI-DIO72 board.

Call:

```
int   io72MaskWrite
      (
      HANDLE   hnd,          /* handle */
      int      mode,         /* inputs to be read */
      void     *buffer,      /* pointer to data buffer */
      void     *maskbuffer   /* pointer to masking data buffer */
      )
```

Description: This function writes data into outputs as well as internal registers of the CPCI-DIO72 board. Write accesses only modify the bits of the target which are set in the masking data.

Specification:

hnd:	Handle which was returned by *io72Open*.
mode:	Selects the target for the write access For permissible values refer to table 3.3
**buffer:*	Pointer to data buffer
**maskbuffer:*	Pointer to masking data When register pairs (64 bits) are accessed, *maskbuffer* has to point to an array of two long words as well.

Return: '0', if the function was successful, or an error code (see table 3.7 on page 24).

Header: io72api.h

io72Read()

Name: **io72Read()** - Reading inputs or registers of the CPCI-DIO72 board.

Call:

```
int     io72Read
        (
        HANDLE      hnd,        /* handle */
        int         mode,       /* inputs to the read */
        void        *buffer     /* pointer to data buffer */
        )
```

Description: Via *io72Read* the status of the digital inputs is read transparently.

Specification:

hnd: Handle which was returned by *io72Open*.

mode: Selection of the digital inputs to be read: see table 3.3
Via *mode* you can access a register pair by means of a call.

**buffer:* Pointer to data buffer.
If there is a register pair, two long words (64 bits) are written at the position to which the pointer *buffer* points.

Return: '0', if the function was successful or an error code (see table 3.7 on page 24).

Header: io72api.h

3.4 Interrupt Handling

Io72IntRead()

Name: **Io72IntRead()** - Interrupt-controlled reading of inputs or registers of the CPCI-DIO72 board.

Call:

```
int     Io72IntRead
        (
        HANDLE      hnd,        /* handle */
        int         mode,       /* inputs to be read */
        void        *buffer     /* pointer to data buffer */
        )
```

Description: Via *Io72IntRead* the digital inputs are read interrupt-controlled.
Before calling this function an interrupt source has to be selected and enabled.
The call of *Io72IntRead()* blocks until the interrupt condition occurs and then gives back the inputs or registers, selected by **mode**. The file `io72test.c` contains an example for the use of this function.

Specification:

hnd:	Handle which was returned by *io72Open*.
mode:	Selection of digital inputs to be read. See table 3.3
**buffer:*	Pointer to data buffer

Return: '0', if the function was successful or an error code (see table 3.7 on page 24).
If more than one interrupt was triggered during the enabling of interrupts or between two *Io72IntRead* calls, *Io72IntRead* returns the value:
IO72_ERR_LOST_IRQ. The reading of data is not influenced by this.

Header: io72api.h

3.5 DMA-Handling

The following functions read the data via Direct Memory Access (DMA). By doing this high transfer rates can be achieved without demanding the CPU too much. Larger amounts of data should therefore be read by means of the DMA-functions, provided the application allows the use of them.

io72DmaRead()

Name: **io72DmaRead()** - Reading inputs of the CPCI-DIO72 board via DMA.

Call:

```
int    io72DmaRead
       (
       HANDLE    hnd,        /* handle */
       int       mode,       /* channels to be read */
       void      *buffer,    /* pointer to data buffer */
       int       size        /* number of bytes to be read */
       )
```

Description: By means of *io72DmaRead* the inputs are read via DMA-access. In contrast to *Io72IntRead* not every sampling triggers an interrupt. The data of the digital inputs are directly written into the working memory of the CPU. The desired size of the buffer is specified by means of the parameter *size.* After the last byte has been written into the buffer, an interrupt is triggered.
Each individual DMA-read access (32 or 64 bits) is triggered by the DMA-trigger source configured by means of *io72Config*. The function only returns after the number of bytes specified in *size* has been read.

Specification:

hnd:	Handle which was returned by *io72Open*.
mode:	Selection of the digital inputs to be read: see table 3.3
**buffer:*	Pointer to data buffer.
size:	Number of bytes to be read (buffer has to have at least the size specified in *size*).

Return: '0', if the function was successful, or an error code (see table 3.7 on page 24).
If the individual reading accesses cannot be handled in the required speed, IO72_ERR_LOST_DATA is returned after the entire transfer has been competed.

Header: io72api.h

io72MultiDmaStart()

Name: **io72MultiDmaStart()** - Initialization of a continuous DMA-transfer.

Call:

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

Description: Together with *io72MultiDmaRead* and *io72MultiDmaStop* inputs are read continuously via DMA by means of this function. *Io72MultiDmaStart* initializes this procedure. *Io72MultiDmaStart* configures the DMA-transfer in such a way that these buffer areas are filled with the reading data in succession. If all buffer areas are filled, the first buffer area is used again (chain buffer made of buffer areas).

The following code fragment shows an exemplary call of *io72MultiDmaRead:*

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

...

/* constructing the array of pointers to buffer areas */
for (i=0; i<BUFFER_COUNT; i++)
  {
  /* request buffer area ... */
    buffer = malloc(BUFFER_SIZE);
    /* ... and add to array */
    bufferlist[i] = buffer;
  }

/* further configuration ... (such as timer) */

...

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

...
```

The advantage of *io72MultiDma*-functions compared to the *io72DmaRead*-function is that all input data are stored after *io72MultiDmaStart* has been called, until *io72MultiDmaStop* is called.
(see also *io72MultiDmaRead* and *io72MultiDmaStop*)

Specification:

hnd:	Handle which was returned by *io72Open*.
mode:	Selection of the digital inputs to be read: see table 3.3
***bufferlist*:	is a pointer to an array or pointers to buffer areas into which the data read is to be written.
bufsize:	shows the size of these buffer areas in bytes.
bufcount:	has to show the number of pointers in the array.

Return: '0', if the function was successful, or an error code (see table 3.7 on page 24).

Header: io72api.h

io72MultiDmaRead()

Name: **io72MultiDmaRead()** - Reading inputs of the CPCI-DIO72-board via DMA

Call:

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

Description: This function blocks until a buffer area which has been designed for this by *io72MultiDmaStart* is filled with data. If this is the case, a pointer pointing to the beginning of this filled buffer area is returned at the position pointed to by *buffer*. The handle of a CompactPCI-digital-I/O board is specified to this function by means of *hnd*. Before the first calling of *io72MultiDmaRead* a DMA-event has to be configured via *io72Config*, otherwise this function will block eternally.

Specification: *hnd:* Handle which was returned by *io72Open*.
**buffer*: Pointer to data buffer.

Return: '0', if the function was successful, or an error code (see table 3.7 on page 24). If the individual reading accesses could not be handled with the required speed, IO72_ERR_LOST_DATA is returned. If more than one buffer area has been filled already since the last call, IO72_ERR_LOST_IRQ is returned.

Header: io72api.h

io72MultiDmaStop()

Name: **io72MultiDmaStop()** - Stopping the continuous DMA-transfer.

Call:
```
int    io72MultiDmaStop
       (
       HANDLE      hnd
       )
```

Description: By calling this function a DMA-transfer which was started by means of *io72MultiDmaStart* is stopped. The handle of a CPCI-DIO72-board is specified to this function via *hnd*.

Specification: **hnd:* Handle which was returned by *io72Open*.

Return: '0', if the function was successful, or an error code (see table 3.7 on page 24). If the individual reading accesses could not be handled with the required speed, IO72_ERR_LOST_DATA is returned. If more than one buffer area had been filled already since the last call, IO72_ERR_LOST_IRQ is returned.

Header: io72api.h

io72DmaBufferAlloc()

Name: **io72DmaBufferAlloc()** - Allocate DMA-buffer.

Call:

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

Description: This function must be used to allocate the DMA-buffer. The driver administrates maximum IO72_MAXDMABUFCOUNT of the memory blocks.

> **Note:** The function *io72DmaBufferAlloc* is only available for Linux drivers at present.

Specification:

hnd: handle which is returned by *io72Open*
bufsize: size of DMA-buffer in bytes (max. IO72_MAXDMABUFSIZE bytes).
***buffer*: a pointer to DMA-buffer is returned

Return: '0', if the function was successful, or an error code (see table 3.7 on page 24).

Header: io72api.h

io72DmaBufferFree()

Name: **io72DmaBufferFree()** - Free DMA-buffer.

Call:

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

Description: This function releases the DMA-buffer which is selected by the pointer *buffer*.

> **Note:** The function *io72DmaBufferFree* is only available for Linux drivers at present.

Specification:

hnd: handle which is returned by *io72Open*
bufsize: size of DMA-buffer in bytes (max. IO72_MAXDMABUFSIZE bytes).
**buffer*: pointer to DMA-buffer

Return: '0', if the function was successful, or an error code (see table 3.7 on page 24).

Header: io72api.h

3.6 Auxiliary Functions

io72GetIrqCount()

Name: **io72GetIrqCount()** - Read out interrupt counter.

Call:
```
int    io72GetIrqCount
       (
       HANDLE     hnd
       )
```

Description: All interrupts triggered by a CPCI-DIO72-board are counted by the driver. By means of this function the counter position of this interrupt counter can be requested. The function can be helpful during the application development.

Specification: *hnd:* Handle which was returned by *io72Open.*

Return: The function returns the current status of the interrupt counter.

Header: io72api.h

io72ResetIrqCount()

Name: **io72ResetIrqCount()** - Reset interrupt counter.

Call:
```
int    io72ResetIrqCount
       (
       HANDLE      hnd
       )
```

Description: By means of this function the counter position of the interrupt counter can be reset to 0 (see *io72GetIrqCount*).

Specification: *hnd:* Handle which was returned by *io72Open.*

Return: This function always returns '0'.

Header: io72api.h

3.7 Returned Values and Error Codes of the API-Calls

Returned value	Meaning
'0'	The function was successful.
ENODEV	A non-existing or inoperable CPCI-DIO72-board has been tried to be accessed. This constant is defined in the file 'errno.h' of the standard include files.
EINVAL	One of the specified parameters is impermissible. The reason for this could be that a permissible value range has been exceeded. This constant is defined in the file 'errno.h' of the standard include files.
ENOMEN	The buffer cannot be allocated.
EBUSY	The function cannot run because a resource on the CPCI-DIO72 is already used (e.g.: a DMA-transfer is active).

Table 3.7: Returned values and error codes

4. Operation of Multi-DMA-Calls

Acting for the DMA-calls of the programming interface, the three functions for the continuous DMA-transfer will be explained in this chapter.
The *io72MultiDma . . .* - calls are capable of reading the input data continuously and writing the data successively into several buffer memories.

With applications which require very large amounts of data to be read per DMA, it can be the case that the data are to be divided into various available memory areas.
The *io72MultiDma. . .* - calls can automatically fill various memory areas successively with reading data without the need of starting another DMA-transfer from the programming code.
The driver uses the scatter-gather-DMA-mode of the PCI9054 bridge on the CPCI-DIO72 board for these functions. By means of the API-call *io72MultiDmaStart* various data structures are constructed and initialised first. Doing this, a scatter-gather-descriptor list is constructed in the buffer.

The individual elements of this list are of SG_DESCRIPTOR_BLOCK type:

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

Description of the fields:

pci_address: Points in each element of the list to a target memory to be filled, whose size was determined when *io72MultiDmaStart* was called (please refer to the documentation of this call).

transfer_size: The size of the target memory is written into this field.

local_address: This field points to the FPGA-register in the local address area of the PCI-bridge which is to be read during each individual DMA-cycle. The driver allows here the capture registers LATCHA or LATCHB. This causes the respective inputs of the CPCI-DIO72 board to be read, if an according capture source has been configured.

next: The last field in the elements of the list is a pointer to the following element of the scatter-gather list and is therefore used to link the elements of the list. The length of the list is also determined when *io72MultiDmaStart* is called.
The end of the list refers to the beginning of the list again, so that a chain structure is achieved. Some bits of the 'next' field (mode) have a special function which is not to be explained in further detail.

After the list described above has been assembled, the beginning of the list (chain_start) is written into the configuration register of the PCI-bridge, and the scatter-gather-DMA-transfer is started. The function *io72MultiDmaStart* now returns, and the application program now calls the function *io72MultiDmaRead.* This function then blocks until the PCI-bridge triggers an interrupt and by doing so reports the successful filling of the first memory area.
The bridge begins to fill the following memory area on its own, by reading the required data from the scatter-gather-descriptor list in the working memory.
Each time another memory area has been filled, an interrupt is triggered.
As soon as the entire list has been worked off, the process proceeds with the beginning of the list again (chain structure). This procedure can be aborted any time by calling *io72MultiDmaStop*. The capacity of these API-calls is the indepent processing of the scatter-gather-descriptor list by means of the PCI-bridge.

5. Examples for Application Programming

In order to be able to test the individual functions of the driver, and in order to make it easy to get into the application programming via the programming interface, a few test routines have been created. By means of command line options a particular test routine can be selected by calling `io72test`. The following table gives an overview of the possible options:

Option	Function
`-INT`	Example for the interrupt-controlled reading of inputs
`-DMA`	Example for reading inputs by means of DMA
`-LATCH`	Example for the use of an external synchronisation input
`-EDGE`	Demonstration of the edge detection
`-SGDMA`	Demonstration of the continuous DMA-transfer
`-MASK`	Example for the masked writing onto outputs

Table 5.1: Options for `io72test`

5.1 Test Adapters

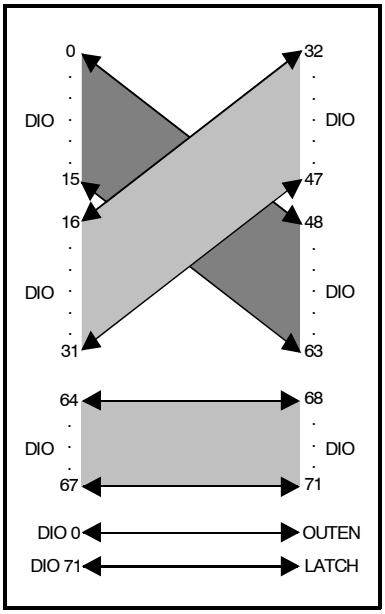


Fig 5.1: Wiring of the test adapter

A correct function of the test routines requires a test adapter, which connects the individual signals of the board to each other, as represented in Fig. 5.1.
The adapter is made of an 80-pin high density connector and a variety of short cable bridges which connect all signals fed to the socket following a pattern.

The special signals OUTEN and LATCH are connected to the I/O-channels so that these signals can be simulated by the test software.

This test adapter is not included in the board's scope of supply.

5.2 Functions of the Driver

At the beginning of the source code a few constants are defined which determine the statuses for the individual example routines (timer frequencies and buffer sizes for the DMA). The individual examples are to be explained briefly below. Before the actual test the test routines always write a test pattern into the 32 channels which were configured as outputs before. In the other channels this test pattern then appears via the test adapter. By doing this, a bit pattern is read in all channels.

5.2.1 io72test -INT

This call of `io72test` configures a timer-controlled, periodical interrupt.
Triggered by an interrupt, inputs 0 to 63 are then read several times in a loop. *io72lntRead()* reads the input data always from the capture registers so that additionally a capture source is configured. The time expired during the reading loop is returned at the end.

5.2.2 io72test -DMA

By this call a memory area is filled with input data. The input data is transferred into the working memory via DMA with each timer impulse.
At the end of the transmission the expired time as well as the first contents of the filled memory area are returned.

5.2.3 io72test -LATCH

This test demonstrates the way the external synchronisation input and the capture registers work. The external synchronisation input is simulated by means of the I/O-line 71.
In the first part of the test an edge is simulated in the synchronisation input and by doing so the current input pattern is transferred into the capture register. In the second part of the test routine a simple task is started which again creates an edge in the synchronisation input.
This edge, however, triggers an interrupt which the first task is waiting for.

5.2.4 io72test -EDGE

The edge detection test consists of two parts as well.
After the edge detection has been configured an edge is first created in an I/O-channel by means of the test adapter. The results are returned.
In the second part of the routine a second task is started, which created several edges in the inputs after a short delay. These edges trigger interrupts which the first task is waiting for. The edge capture register is returned at the end of this test.

5.2.5 io72test -SGDMA

`io72test -SGDMA` demonstrates the DMA-transmission in the scatter-gather-operating mode. First, a few memory areas are requested which are to be filled later. The timer and the capture event are configured. Then a filled memory area is waited for several times in blocking mode. It is helpful to know that the filling of a memory area is waited for twice as often as memory areas have been requested. After all memory areas have been filled, the first is filled again. This is due to the chain structure of the memory areas during scatter-gather DMA, as described in chapter 4.

5.2.6 io72test -MASK

This test demonstrates the way the *io72MaskWrite*-calls of the programming interface are working. Writing accesses in the routine always only mainpulate the bit positions which have been enabled for this by masking data.