
P_SERVER Documentation

Ingo Müller May 2003
Rev 1.7

Berliner Elektronenspeicherring-Gesellschaft
für Synchrotronstrahlung m.b.H.

Copyright

Copyright (c) 1997 by

Berliner Elektronenspeicherring-Gesellschaft

fuer Synchrotronstrahlung m.b.H.,

Berlin, Germany

This software is copyrighted by the BERLINER SPEICHERRING GESELLSCHAFT FUER SYNCHROTRONSTRAHLUNG M.B.H., BERLIN, GERMANY. The following terms apply to all files associated with the software.

BESSY hereby grants permission to use, copy, and modify this software and its documentation for non-commercial educational or research purposes, provided that existing copyright notices are retained in all copies.

The receiver of the software provides BESSY with all enhancements, including complete translations, made by the receiver.

IN NO EVENT SHALL BESSY BE LIABLE TO ANY PARTY FOR DIRECT, INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OF THIS SOFTWARE, ITS DOCUMENTATION, OR ANY DERIVATIVES THEREOF, EVEN IF BESSY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

BESSY SPECIFICALLY DISCLAIMS ANY WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, AND NON-INFRINGEMENT. THIS SOFTWARE IS PROVIDED ON AN "AS IS" BASIS, AND BESSY HAS NO OBLIGATION TO PROVIDE MAINTENANCE, SUPPORT, UPDATES, ENHANCEMENTS, OR MODIFICATIONS

Table of Contents

Chapter 1: System Configuration	5
Introduction	5
Single Card System	5
ISA96 Bus System	5
A/D-Channel Multiplexing	6
Chapter 2: Serial Communication	7
Introduction	7
Commands	7
Chapter 3: CAN Communication	13
Introduction	13
Receiving CAN Messages	13
Transmitting CAN Messages	13
CAN Identifier and NODE-ID	14
IOVAL-Object	14
CONTROL-Object	16
WATCH-Object	18
Chapter 4: User Software Expansion Interface	19
Introduction	19
PS_EXT.H	19
CI_EXT.H	20
Chapter 5: BESSY CAN Adapter	21
Introduction	21
CAN Connector	22
Pin description: 22	
5V Power Connector	22
Pin description: 22	
Serial Ports	22
Pin description: 23	
Status LED's	23
Chapter 6: P_SERVER History and Revision Overview	25
History	25

Chapter 1: System Configuration

Introduction

The P_SERVER running on the Intel386EX controller board from FS-FORTH-Systeme GmbH can be used in various system configurations. Supported cards are the BESSY ISA-Carrier-Board, EuKontroll ADA16-IO8 (4/8 channel), EuKontroll ADA2x16-IO8 (4/8 channel) and EuKontroll IO32(_2). The ADA(2x)16-IO8 cards are operated in interrupt mode. The IO32 cards are polled. For CAN connection the BESSY CAN adapter (including 2 serial ports) is supported.

Single Card System

In this configuration the embedded controller board is located direct on an IO card. With the CAN adapter the IO card can be controlled by CAN bus. The card may be plugged into a device, e.g. a power supply, with the input/output connector and is powered from this side. The CAN adapter attached to the bus connector side. P_SERVER detects the configuration and card type and configures its CAN variables automatically. In a single card system the base address and interrupt of the IO card is fixed and can not be configured.

BASE (BSTA) = 0x300

IRQ = 7

ISA96 Bus System

In this configuration the embedded controller board is located on a carrier board. The carrier board occupies one slot of the ISA96 backplane it shares with one ore more IO cards. P_SERVER supports up to 5 ADA(2x)16-IO8 cards and up to 8 IO32(_2) cards. During the boot up process the P_SERVER searches at first for the

ADA(2x)16-IO8 cards and then for the IO32(_2) cards. After detecting a gap in the ADA(2x)16-IO8 or IO32(_2) address table, the search process changes to the next IO cardtype.

ATTENTION: The maximum count of IO cards is 9 !

Table 1: Card Configuration

Nr.	Cardtype	BASE	IRQ
0	ADA(2x)16	0x310	7
1	ADA(2x)16	0x320	9
2	ADA(2x)16	0x330	12
3	ADA(2x)16	0x340	6
4	ADA(2x)16	0x350	14*
0	IO32(_2)	0x200	-
1	IO32(_2)	0x210	-
2	IO32(_2)	0x220	-
3	IO32(_2)	0x230	-
4	IO32(_2)	0x240	-
5	IO32(_2)	0x250	-
6	IO32(_2)	0x260	-
7	IO32(_2)	0x270	-

* for IRQ 14 it is necessary to use the BESSY IRQ-EXPANDER

A/D-Channel Multiplexing

The ADA(2x)16-IO8 card provides either 4 differential or 8 single-ended analog inputs. An analog multiplexer switches between these channels. The channels have a different priority. For 4 and 8 channel cards the multiplexing differs in the channel sequence.

Channel sequence for 4 channel configuration: {2,0,2,3,2,1,2,3}

Channel sequence for 8 channel configuration: {0,1,2,3,4,5,6,7}

The time between two actual values of different channels is 200 ms.

Chapter 2: *Serial Communication*

Introduction

The P_SERVER can be controlled by RS232 commands. A cyclically called command interpreter handles the commands and expects a preceding <TAB> for each command. The RS232 communication needs no hard-/software handshake and uses the following parameter: 9600 baud / 8 bit / no parity / 1 stopbit.

Commands

Some commands can be used in a long and a short typed form. If a necessary parameter follows, it is marked here by <>. The command and the parameter(s) are separated by a blank (character).

The following commands are supported:

- `help,h,?`
Displays a small help menu. The help menu depends on the actual selected IO card type and includes all practicable commands. The same menu appears if the preceding <TAB> is missing.
- `cardnr,nr <0..n>`
Chooses an IO card for the following card specific commands. This cards is stays selected until a new card is chosen by this command.

Example: `cardnr 2`
Selects the card with number 2
- `terminal,term`
The terminal command chooses one of two communication modes. Terminal mode displays information in a detailed form. This mode is used to control the P_SERVER with a terminal programm like PROCOM or HYPERTERMINAL. The other possible mode is the pc mode.

-
- pc
The pc command chooses one of two communication modes. The pc mode displays information in a very short form. This mode is used to control the P_SERVER with an application e.g. running on a PC.
 - id
This command displays the CAN node ID of this device.
 - busoff
The busoff command forces the CAN controller to disconnect from the CAN bus line. Receiving and transmitting of CAN messages is made impossible. This busoff state is indicated by permanent red and yellow LED light at the CAN adapter.
ATTENTION: This state can not be changed or left by a CAN bus command !
 - buson
The buson command forces the CAN controller to connect to the CAN bus line. Receiving and transmitting of CAN messages is made possible.
 - reset
This command forces a hardware reset on the embedded controller board. In case of an ISA96 bus system, no reset will generated on the bus (RESETDRV).
 - rtcn,sn
Displays the RTC (real-time-clock) serial number. The number is unique and can be used for identification of the used embedded controller board.
 - clear,cl
This command clears all error counters and flags, except for system initialization errors.
 - adastat,as
The adastat command displays status information of the selected ADA16(14)-IO8 card. The following items will be displayed:
CARDTYPE :
Hardware coded IO card information.
ADA_RESET_ERR:
This errorflag is set if a reset of the IO card failed during the initialization phase.
ADA_DIGOUT_ERR:
This errorflag is set if a write to the digital output port fails.
ADA_CONVERSION_ERR:
This errorflag is set if the reading of the analog inputs fails.
ADA_ERR_CODE:
Errorcode defined in *ada_err.h*. These errors are generated in the *bessy_io.lib*.
ADA_ANAIN_INT_ERR:
This is an 8bit error counter for the analog input errors which occur in the interrupt context.

ATTENTION: This command is only practicable if at least one ADA(2x)16-IO8 card exists.
 - io32stat,is
The io32stat command displays status information of the selected IO32(_2) card. The following items will be displayed:

CARDTYPE :

Hardware coded IO card information.

IO32_RESET_ERR:

This errorflag is set if a reset of the IO card failed during the initialization phase.

IO32_DIGOUT_ERR:

This errorflag is set if a write to the digital output port failed.

IO32_DIGIN_ERR:

This errorflag is set if a read of the digital input port failed.

IO32_ERR_CODE:

Errorcode defined in *io32_err.h*. These errors are generated in the *bessy_io.lib*.

ATTENTION: This command is only practicable if at least one IO32 card exists.

- canstat,cs

The canstat command displays CAN related status information. The following items will be displayed:

SCI_OPEN_ERR:

This flag is set if the SCI-function SCI_OPEN returns an error.

SCI_INIT_ERR:

This flag is set if the SCI-function SCI_INIT returns an error.

SCI_SET_CALLB_ERR:

This flag is set if the SCI-function SCI_SET_CALLBACK returns an error.

PS_NODE_ID_ERR:

This flag is set if an invalid CAN node-id is selected by the DIP-switches.

I82_PORTCONF_ERR:

This error flag is set if the configuration of the CAN controller ports failed.

SCI_ERRCODE:

Errorcode defined in *sci_err.h*. These errors are generated in the *can_sub.lib*.

I82527_ERRCODE:

Errorcode defined in *i82_err.h*. These errors are generated in the *can_sub.lib*.

READER_ERR:

This is an error counter which counts the errors generated by the SCI-interrupt handler.

CAN_BUSOFF_ERR:

This error flag is set if the CAN controller goes into busoff state. This occurs if the internal error counter of the CAN controller reaches a limit.

I82_GET_BUSOFF_ERR

This error flag is set if the I82527GET_BUSOFF function returns an error.

I82_BUS_START_ERR:

This error flag is set if the I82527BUS_START function returns an error.

SCI_WRITE_ERR:

This error flag is set if SCI_WRITE fails in the idle loop.

BUSOFF_ERR:

This 8 bit errorcounter is incremented every time the CAN controller goes in busoff state.

- protstat,ps

The protstat command displays lowcal protocol related status information. The following items will be displayed:

LC_READ_PROT_ERR:

This error flag is set if the LOWCAL_READ function returns an error.

LC_WRITE_PROT_ERR:

This error flag is set if the LOWCAL_WRITE function returns an error.

LC_PUT_PROT_ERR:

This error flag is set if the LOWCAL_PUT function returns an error.

LC_INIT_ERR:

This error flag is set if the LOWCAL_INIT_START function returns an error.

LC_INIT_ENTI_ERR:

This error flag is set if the LOWCAL_INIT_ENTITY function returns an error.

LC_INIT_END:

This error flag is set if the LOWCAL_INIT_END function returns an error.

LC_GET_PROT_ERR:

This error flag is set if the LOWCAL_GET function returns an error.

- `anain,ai <0..7,x>`

This command is used to display the values of the analog input ports. The following parameter `<0..7>` selects a single channel, `<x>` selects all channels. It returns the measured value(s) and a LowRes-flag for each channel. The LowRes-flag is only valid for ADA(2x)16-cards. It indicates an overflow on ADC1 and the use of the fast +/-15 bit ADC2.

The range of the measured values is:

-32.767 .. +32767 for ADA16-IO8 cards

-9.227.443 .. +9.194.675 for ADA2x16-IO8 and bipolar configuration

-838.835 .. +9.211.059 for ADA2x16-IO8 and unipolar configuration

Example: `anain 3`

This command displays the value of the analog input 3.

ATTENTION: This command is only practicable if an ADA(2x)16-IO8 card exists and is selected by the `cardnr` command.

- `anaout,ao <0..65535>` for ADA16-IO8 cards
 `<0..16777215>` for ADA2x16-IO8 cards

This command loads the analog output port with the specified value.

Example: `ao 65535`

This command loads the analog output of an ADA16-IO8 card with the maximum value.

ATTENTION: The output voltage depends on the configuration of the ADA(2x)16-IO8 card (0..5V; -5..+5V; 0..10V; -10..+10V).

- `digin,di`

The `digin` command reads the digital input ports. The range of the displayed values depends on the IO card type.

IO32 card : 0..65535 (dec)

IO32_2 card: 2x 0..65535 (dec)

ADA(2x)16 card : 0..255 (dec)

- `digout,do <0..3; 0..255; 0..65535>`

This command loads the digital output with specified value. The valid range is 0..255 for ADA16(14)-IO8 cards , 0..65535 for IO32 cards, 0..3 for IO32_2 cards.

Example: digout 0

This command clears the digital output port.

- aoutrb,ar

This is a read-back command for the analog output. It displays the registered value from the analog output.

ATTENTION: This command is only practicable if an ADA(2x)16-IO8 card exists and is selected by the cardnr command.

- dorb

This is a read-back command for the digital output. It displays the the registered value from the digital output.

- sys

This command displays system hardware information.

- genstat,gs

genstat displays general status information for RS232 user which is also available via CAN bus.

- pmon

This command enables/disables a port access monitor. The monitor function displays all changes at the digital input/output port and the analog output port.

- cmon

This command enables/disables a small CAN bus monitor . The monitor displays only CAN objects which are initialized on this device.

- setbit,sb <0..1; 0..7; 0..15>

Sets one bit of the 8/16-bit digital output port of the ADA(2x)16- or IO32(_2)-card.

- getbit,gb <0..1; 0..7; 0..15>

get the value of one bit of the 8/16-bit digital input port of the ADA(2x)16- or IO32(_2)-card.

- clrbit,cb <0..1; 0..7; 0..15>

Clears one bit of the 8/16-bit digital output port of the ADA(2x)16- or IO32(_2)-card.

- rt

displays the time since the last CPU boot up.

- debug (undocumented now)

- gapcor(undocumented now)

Chapter 3: CAN Communication

Introduction

For CAN communication the P_SERVER uses the LOWCAL and DCANAL (not implemented yet) protocol, both developed by BESSY. LOWCAL supports client-server based connections with different variable types. DCANAL is used to transmit files, tables or datablocks in a secure mode. The P_SERVER supports the Intel I82527 CAN controller in CAN Revision 2.0A mode (11 bit identifier). This controller has 14 buffers for receiving or transmitting CAN messages with different CAN identifiers. With the BESSY CAN adapter the physical layer is according to ISO11898. Four different CAN baudrates are provided: 1Mbit, 500Kbit, 250Kbit, 125Kbit and 66.6Kbit. The baudrate cannot be changed after compilation of source code.

Receiving CAN Messages

If a CAN message is received by the CAN controller an interrupt is generated. The SCI interrupt handler serves this interrupt and calls a callback function. The callback function puts the CAN data into the protocol layer. If a postprocess is defined for this CAN variable, an entry is added to a postprocess queue. This queue is checked and processed in the idle loop.

Transmitting CAN Messages

The transmitting of CAN messages is executed in the idle loop. Here a get-command to the protocol layer delivers the CAN data which has to be sent. This sending is initiated by a preceding protocol put-command (executed in the callback function) or by a protocol write-command (executed somewhere in the application).

CAN Identifier and NODE-ID

According to CAN Revision 2.0A it is possible to use 2048 different CAN identifiers for 2048 CAN objects (COB). The BESSY MLT specification divides the 11 bit identifier in a Node-ID (NID), Channel-ID (CID) and a direction bit (d). The participants on CAN bus could have the Node-ID 1..63 and could use the Channel-IDs 0..12 for both directions. The direction flag (d) is 1 for Read-objects and 0 for Write-objects on the server (see LOWCAL).

Name	CID				d	NID					
Bit of CAN object	10	9	8	7	6	5	4	3	2	1	0
Bit-name	c ₃	c ₂	c ₁	c ₀	d	n ₅	n ₄	n ₃	n ₂	n ₁	n ₀
valid range	0..12					1..63					

The advantage of this COB-splitting is, that every node has both high priority COBs and low priority COBs. The following COBs are used by the P_SERVER:

COB-Name	COB-ID	Object type	Access
WATCH-obj.	Node-ID + 64	LOWCAL multiplexed, write only	Client
CONTROL-obj. (write)	Node-ID + 128	LOWCAL multiplexed, read/write	Server
CONTROL-obj. (read)	Node-ID + 192		
IOVAL-obj. (write)	Node-ID + 256	LOWCAL multiplexed, read/write	Server
IOVAL-obj. (read)	Node-ID + 320		
GAPVAL (read)	Node-ID + 448	defined later	Server

IOVAL-Object

The IOVAL-object is a multiplexed CAN-object with the multiplexer in databyte 0. The total length of this CAN-object is 3 or 4 bytes depending on the used IO cards. If at least one ADA2x16 card is used the length is 4 bytes. This object is used to transmit the IO-data in both directions.

Multiplexer Usage	Number of multiplexer	Range of multiplexer
Card 0	12	0..11
Card 1	12	12..23

Multiplexer Usage	Number of multiplexer	Range of multiplexer
Card 2	12	24..35
Card 3	12	36..47
Card 4	12	48..59
Card 5	12	60..71
Card 6	12	72..83
Card 7	12	84..95
Card 8	12	96..107
not used	20	108..127

The contents of the CAN databytes depend on the multiplexer and are shown in the next table.

Multiplexer	Type	Utilization
(n*12) + 0	2 or 3 byte noarray, sign	Byte 0: A/D channel 0 (lowbyte)
		Byte 1: A/D channel 0 (highbyte)
(n*12) + 1	2 or 3 byte noarray, sign	Byte 0: A/D channel 1 (lowbyte)
		Byte 1: A/D channel 1 (highbyte)
(n*12) + 2	2 or 3 byte noarray, sign	Byte 0: A/D channel 2 (lowbyte)
		Byte 1: A/D channel 2 (highbyte)
(n*12) + 3	2 or 3 byte noarray, sign	Byte 0: A/D channel 3 (lowbyte)
		Byte 1: A/D channel 3 (highbyte)
(n*12) + 4	2 or 3 byte noarray, sign	Byte 0: A/D channel 4 (lowbyte)
		Byte 1: A/D channel 4 (highbyte)
(n*12) + 5	2 or 3 byte noarray, sign	Byte 0: A/D channel 5 (lowbyte)
		Byte 1: A/D channel 5 (highbyte)
(n*12) + 6	2 or 3 byte noarray, sign	Byte 0: A/D channel 6 (lowbyte)
		Byte 1: A/D channel 6 (highbyte)
(n*12) + 7	2 or 3 byte noarray, sign	Byte 0: A/D channel 7 (lowbyte)
		Byte 1: A/D channel 7 (highbyte)
(n*12) + 8	1 or 2 byte noarray, nosign	Byte 0: digital input (lowbyte); (ADA(2x)16: 8bit)
		Byte 1: digital input (highbyte)
(n*12) + 9	1 or 2 byte noarray, nosign	Byte 0: digital output (lowbyte); (ADA(2x)16:8bit)
		Byte 1: digital output (highbyte)
(n*12) + 10	2 byte noarray, nosign	Byte 0: analog output (lowbyte)
		Byte 1: analog output (highbyte)
(n*12) + 11	2 byte noarray, nosign	Byte 0: digital input highword(lowbyte); IO32_2
		Byte 1: digital input highword(highbyte); IO32_2

n = number of IO-card 0..8

CONTROL-Object

The CONTROL-object is a multiplexed CAN-object with the multiplexer in databyte 0. The total length of this CAN-object is 8 bytes. This object is used for controlling, diagnostics and error checking of the P_SERVER.

Multiplexer Usage	Number of multiplexer	Range of multiplexer
CPU	8	0..7
CAN	8	8..15
Card 0	4	16..19
Card 1	4	20..23
Card 2	4	24..27
Card 3	4	28..31
Card 4	4	32..35
Card 5	4	36..39
Card 6	4	40..43
Card 7	4	44..47
Card 8	4	48..51
not used	16	52..67
Misc	4	68..71
not used	56	72..127

The contents of the CAN databytes depend on the multiplexer and are shown in the next table.

Multiplexer	Type	Utilization
0	2 byte array (8bit), nosign	Byte 0: 231d forces a CPU reset
		Byte 1: 024d forces a CPU reset
1	1 byte array (8bit), nosign	clears error flags and counters (if possible)
2	0 byte array (8bit), nosign	ping (red and green2 led are blinking at the CAN adapter for a short while)
3	1 byte array (8bit),nosign	enables /disables gap correction (0 = off (default); 1 = on)
4..7	---	not used
8	6 byte array (8bit), nosign	Byte 0: errors during the init phase of CAN Bit 0: SCI_OPEN_ERR Bit 1: SCI_INIT_ERR Bit 2: SCI_SET_CALLB_ERR Bit 3: --- Bit 4: PS_NODE_ID_ERR Bit 5: I82_PORTCONF_ERR Bit 6/7:---
		Byte 1: SCI_ERRCODE

Multiplexer	Type	Utilization
		Byte 2: I82527_ERRCODE Byte 3: READER_ERR (counter) Byte 4: CAN errorflags Bit 0: CAN_BUSOFF_ERR Bit 1: I82_GET_BUSOFF_ERR Bit 2: I82_BUS_START_ERR Bit 3: SCI_WRITE_ERR Bit 4..7: --- Byte 5: BUSOFF_ERR (counter)
9	1 byte array (8bit), nosign	Byte 0: LOWCAL protocol errorflags Bit 0: LC_READ_PROT_ERR Bit 1: LC_WRITE_PROT_ERR Bit 2: LC_PUT_PROT_ERR Bit 3: LC_INIT_ERR Bit 4: LC_INIT_ENTI_ERR Bit 5: LC_INIT_END_ERR Bit 6: LC_GET_ERR Bit 7: ---
10..15	---	not used
(n*4)+16	5 byte array (8bit), nosign	Byte 0: Cardtype Byte 1: errors during the init phase of IO cards Bit 0..1: --- Bit 2: ADA_RESET_ERR / IO32RESET_ERR Bit 3..7: --- Byte 2: IO-card errorflags Bit 0: ADA_DIGOUT_ERR / IO32_DIGOUT_ERR Bit 1: ADA_ANAOUT_ERR Bit 2: ADA_DIGIN_ERR / IO32_DIGIN_ERR Bit 3: ADA_CONVERSION_ERR Bit 4..7: --- Byte 3: ADA_ERRCODE / IO32_ERRCODE Byte 4: ADA_ANAIN_INTR_ERR (counter) Byte 5: Low resolution flags Bit 0: LowRes flag A/D channel 0 Bit 1: LowRes flag A/D channel 1 Bit 2: LowRes flag A/D channel 2 Bit 3: LowRes flag A/D channel 3 Bit 4: LowRes flag A/D channel 4 Bit 5: LowRes flag A/D channel 5 Bit 6: LowRes flag A/D channel 6 Bit 7: LowRes flag A/D channel 7
(n*4)+17	---	not used
(n*4)+18	---	not used
(n*4)+19	---	not used
52..67	---	not used
68	1 byte array (8bit), nosign	Byte 0: errors during the init phase Bit 0: --- Bit 1: ADA_INI_ERR Bit 2: ADA_SET_CALLB_ERR Bit 3..7: ---

Multiplexer	Type	Utilization
69	2 byte array (8bit), nosign	Byte 0: sum-errorflags Bit 0: GENERAL_CAN_ERR Bit 1: GENERAL_CPU_ERR Bit 2: GENERAL_ADA_ERR Bit 3: GENERAL_MISC_ERR Bit 4: GENERAL_IO32_ERR Bit 5..7: --- Byte 1: not used
70..71	---	not used
72..127	---	not used

n = number of IO card 0..8

WATCH-Object

The WATCH-object is a multiplexed CAN-object with the multiplexer in databyte 0. The total length of this CAN-object is 2 bytes. This object is used to transmit boot/reboot information. In this case P_SERVER is client and sends the information without a preceding request.

Multiplexer Usage	Number of multiplexer	Range of multiplexer
Init Request	1	1
not used	127	1..127

The contents of the CAN databytes depend on the multiplexer and are shown in the next table.

Multiplexer	Type	Utilization
1	1 byte array (8bit), nosign	Byte 0: Init Request after coldboot : 0x10 after warmboot: 0x1A
0; 2..127	---	not used

Chapter 4: User Software Expansion Interface

Introduction

The User Software Expansion Interface (USEI) offers the ability to integrate user specific functions and additional CAN connectivity into the P_SERVER. Therefore some macro-depending includes and function calls exist in the P_SERVER source code. This feature is activated after new compiling, if the user defines a macro named "EXTERN". Then *ps_ext.h* and *ci_ext.h* will be included, some functions will be called and the variable "muxvars" will be modified.

PS_EXT.H

In *ps_ext.h* the following functions have to be declared:

- void `ps_ext_init(void)`
This function is called after the P_SERVER I/O-hardware initialization. It should be used to initialize user specific hardware extensions.
- int `ps_ext_can_init(Byte, int, Entity_type*)`
This function is called at the end of the fill procedure of the `lowcal_array`. The array is used for calling the `lowcal_init_entity`-function which defines the CAN lowcal variables. With this function the user is able to create additional lowcal variables. The first parameter(Byte) is the actual element number of the `lowcal_array`. `lowcal_array` is already filled until this element number. The second parameter(int) is the CAN node ID of the system. This parameter is also necessary for the `lowcal_init_entity`-function. The last parameter(`Entity_type*`) is a pointer to the `lowcal_array`. Return value(int) is the possibly modified element number (last used element) of the `lowcal_array`.
- void `ps_ext_loop_exec(void)`
This function is cyclically called in the `idle_loop`. This loop is always running on P_SERVER. With this function the user is able to control his own hardware or integrate own application code.

In addition to the functions the user has to create the variable “ps_ext_mux” (int). This variable contains the number of additional created lowcal variables in ps_ext_can_init(). The prototypes of additional command interpreter functions have to be located in *ps_ext.h*, too.

This is an example for a possible *ps_ext.h* file:

```
#if defined EXTERN
extern int ps_ext_mux;      /*number of additional used multiplexer*/
void ps_ext_init(void);
int ps_ext_can_init(Byte, int, Entity_type*);
void ps_ext_loop_exec(void);

/* prototypes of command interpreter functions */
void ci_lowcalBufferInfo(int account, char *arg[]);
void ci_testfunction(int account, char *arg[]);
#endif
```

CI_EXT.H

This header file contains the additional commands for the command interpreter. For each command the user has to define a command string and a function which should be executed.

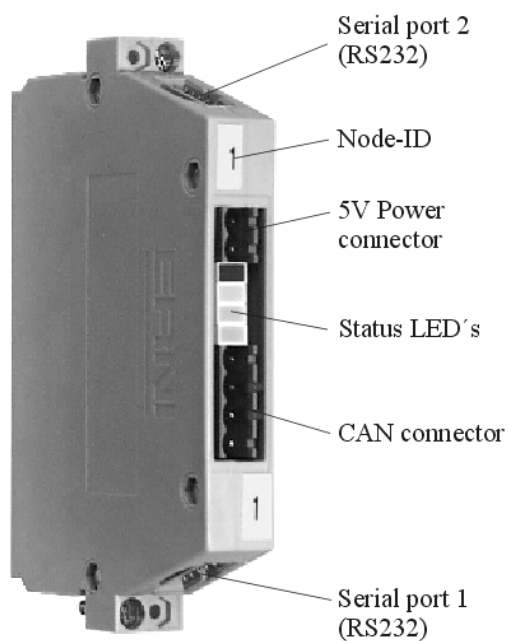
This is an example for a possible *ci_ext.h* file:

```
/*ci_ext.h*/
{"lbinfo", ci_lowcalBufferInfo},
{"test", ci_testfunction},
/*end*/
```

Chapter 5: BESSY CAN Adapter

Introduction

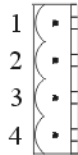
The BESSY CAN Adapter is an interface for CAN, RS232 and 5V power supply developed by BESSY-MLT.



CAN Connector

For CAN communication the physical layer of the CAN bus is according to ISO11898. The CAN connector type is a “MSTB2,5/4-G-5,08” from Phoenix Contact.

Pin description:



Pin	Description
1	CAN-L
2	CAN-H
3	GND
4	(CAN-L)

5V Power Connector

This connector is used for supplying the system from the CAN adapter side (only recommended for test purposes). The CAN connector type is a “MSTB2,5/2-G-5,08” from Phoenix Contact.

Pin description:



Pin	Description
1	GND
2	+5V

Serial Ports

The two serial ports are realized in standard RS232. The serial port 1 is used by P_SERVER for terminal communication. Communication parameters of port 1 are described in the following table. The second serial port is not used in the P_SERVER.

Parameter	Value
baudrate	9600 baud
bits	8
parity	No

Parameter	Value
stopbits	1
handshake	No

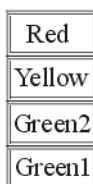
Pin description:



Pin	Description
1	/DCD
2	/DSR
3	RXD
4	/RTS
5	TXD
6	/CTS
7	/DTR
8	/RI
9	GND
10	n.c.

Status LED's

The status LEDs are used to display several different system status information.



Status	LED code
Initialize phase/all channel overflow	Red
Idle-loop	Green1 (blink)
CAN bus traffic	Green2
Internal Error/Warning	Yellow
CAN busoff state	Red (blink)
Ping-command	Red (blink) Green1 (blink)

Chapter 6: *P_SERVER History and Revision Overview*

History

- 01.07.96 I.M. starting with long tests...
- 11.09.96 I.M. removed error in check_general_err();
- 14.01.97 I.M. Multicard-Support 2.00PS/1M
- 07.03.97 I.M. b_reboot_init() implemented, set_boot_info appends now from accessing the analog output; pp_anaout modified; 2.01PS/1M
- 13.03.97 I.M. Bug in "can_init()" removed; sci_set_callback-condition was wrong. Ci_can_bus_stop/start-LED-indicator added; 2.02PS/1M
- 04.06.97 KUN support control of other cards than ada16, io32 added
- 27.06.97 I.M. Now: supports Dallas temperature Sensors ;2.10PS/1M
- 08.10.97 I.M. functions modified to be able to work with allcards=0 :
- get_diginval()
 - ci_print_help()
 - ci_set_cardnr()
 - ci_adastat()
 - ci_io32stat()
 - ci_anaout()
 - ci_anaout_rb()
 - ci_anain()
 - ci_digout()
 - cl_digout_setbit()
 - ci_digout_clrbit()
 - ci_digin()
 - ci_digin_bit() ;2.11PS/1M
- 15.10.97 I.M. function get_advalue() modified. Now an OVERFLOW-Error occurs after 10 overflows ; 2.12PS/1M
- 23.10.97 I.M. IDLELOOP(): when a sci_write-error occurs, the i82527errmess is written to LOWCAL (P_WRITE instead of P_SET); (2.13PS/1M)

-
- 30.10.97 I.M. the support of Dallas temperature sensors is removed with all according functions,variables,includes,... 2.20PS/1M
 - 10.02.98 I.M. some DEFINES, TYPEDEF's and variables(now extern) are moved to pserver.h; 2.21PS/1M
 - 04.05.98 I.M. ci_anain modified: the new parameter "x" displays the value of all channels of one card !
 - 04.05.98 I.M. New function: ci_print_sys; displays system configuration; (2.22PS/1M)
 - 04.05.98 I.M. WARN-led-bug removed in ci_can_bus_start (2.23PS/1M)
 - 29.07.98 I.M New kind of Software Version/Status information;(2.24PS)
 - 30.07.98 I.M Idleloop:after sci_write getdata-field is cleared.(2.25PS)
 - 10.08.98 I.M. Bug removed in ci_print_sys; cardnr changed to var "i" (2.26PS)
 - 17.08.98 I.M. P_SERVER supports now 5 ADA16-cards; changes made in P_SERVER.H and ADA_PAR.H (2.30PS)
 - 20.08.98 I.M. P_SERVER supports now ADA14-cards; p_server.h is modified too ! (2.40PS)
 - 12.10.98 I.M. 3 variables (q_head;q_tail;q_len)added to p_server.h (2.41PS)
 - 20.01.99 I.M. 1 variable (outputaccess) added, for enable/disable writing to the outputs of the I/O-cards. This var. is checked before all write accesses.(2.42PS)
 - 22.01.99 I.M. variable (lock_input) added. This var is used as disable flag for the call of get_input() and command_ interpreter() function.(2.43PS)
 - . --. -- I.M. b_reboot_init() modified. Now the digital ports are always restored int the protocol layer. Send_init_request() modified. Now 0x10 = coldboot and 0x1A = warmboot.
 - 10.05.99 I.M. dcanal protocol added;many new features and modifications, prepared for gap correction(2.50PS)
 - 21.01.00 I.M type of lowcal multiplexer for A/D values changed from nosign to sign; command <digin> returns also HEX values (2.51PS)
 - 31.01.00 I.M. Now with new revision of bessy_io.lib. Older versions could cause I/O-read errors of ADA16-registers(2.60PS)
 - 04.07.00 I.M. Support for ADA2x16 card added. Several functions, types and variables modified. Two serial commands added:
dorb : digital output readback
genstat: general status
mo: monitor function(2.70PS)
 - 13.11.00 I.M. Now with new revision of bessy_io.lib(ADA2x16 part).In version 2.70PS an overflow at boot-time could cause permanent analog-in-errors(2.71PS)
 - 05.02.01 I.M. Now with new revision of bessy_io.lib(ADA16 part) wich supports the newest MACH version(2.72PS)
 - 27.09.01 I.M. Small Bug in global_writecontrol() removed! Now the control-obj. is filled with IO32-card data. Two serial commands added:
-

rt: time since last CPU boot

mon: CAN monitor function on/off

(no command renamed in pmon) (2.73PS)

09.11.01 I.M. Small bug removed in get_advalue(): Now the red LED (overflow on all channels) is only turned off, if the can_bus_off-flag is 0. LED codes changed (2.74PS)

10.07.02 I.M. IO32_2-card support added (2.80PS)

24.03.03 I.M. Small error handling bug in init-phase removed (2.81PS)

