bitcontrol® dev-can-pcan

QNX® Neutrino Device Driver for Philips 82C200 / SJA1000 CAN Controller

BitCtrl Systems GmbH
Weißenfelser Str. 67
04229 Leipzig

www.bitctrl.de/en

Copyright and Trademark

The documentation reflects the present development stage of the product ***bitcontrol® dev-can-pcan.*** Should you come across any errors or unclear passages in the documentation please contact:

BitCtrl Systems GmbH
Weißenfelser Str. 67
04229 Leipzig, Germany
Tel. +49-341-49067 0
Fax +49-341-49067 15
Email info@bitctrl.de

Liability

BitCtrl Systems GmbH (referred to as BitCtrl in the following) will not accept liability (whether specifically or implicitly) for the software product ***bitcontrol® dev-can-pcan*** and its components. This includes any claims regarding usage and suitability of the software for a specific purpose. BitCtrl will in no way accept liability for coincidental, indirect or consequential damage resulting from misuse or correct usage of the software. This also applies if BitCtrl should be informed prior about such possible damage.
The general terms of business for BitCtrl Systems GmbH will apply. Rights to change the software and documentation accrued thought technical advancements are reserved.

Release Levels

Documentation Version 1.4	Major update due to version 5.x release Released Nov 2020
Documentation Version 1.3	Created by: BitCtrl Systems GmbH Contact: info@bitctrl.de Released: Sep 2009 Copyright © 2008, 2009 by BitCtrl Systems GmbH - Leipzig, Germany

Table of Contents

Glossary

Term	*Meaning*

1 Introduction

The ***bitcontrol® dev-can-pcan*** Device Driver is a QNX® resource manager for CAN (based on Philips 82C200/SJA1000 Stand-alone CAN controller) for PCI/ISA-CAN Cards (1).

The PEAK-System Technik GmbH PCAN-PCI cards always operate in PeliCan (CAN 2.B) mode while the ISA cards always operate in BasicCAN (CAN 2.A) mode. This is important for instance when setting the acceptance mask: With a PCI-card you can use full 29 bits whereas when working with an ISA card you can only use 11 bits.

Attention: Current version 5.x is API-compatible with 4.x, but not compatible with previous versions (3.x or earlier)

The CAN interface is implemented as resource manager in OS QNX® Neutrino. The functionality of the CAN resource manager is comparable to features of an appropriate serial driver under other UNIX operating systems.

In QNX® Neutrino, the CAN resource manager is a regular „user“ task that can be launched and terminated during system runtime. The CAN resource manager operates as special server task having the following characteristics:

- It controls applications (client processes) with the typical performances of a server. For that it uses the interface of the Interprocess Communication with message passing (opening, reading, writing, closing, etc.).
- It integrates itself into the namespace for serial „device drivers“, and supports the name handling according to QNX® Neutrino.
- It is able to react to the defined QNX® Neutrino or POSIX system calls in form of messages and to answer them.
- It supports the typical POSIX C functions like *read()*, *write(), etc.;* whereby the data buffer is put under a special structure.

The structure of name giving for the resource manager in the QNX® file system is as follows:

```
/dev/can/<n>
```

- /dev/can is the name of the registered resource manager
- <n> is the number of the CAN controller in the QNX® Neutrino system

The 82C200/SJA1000 Device Driver features multi-channeling. In addition, it supports interrupt sharing, i.e. more than one controller may use the same interrupt.

2 System Requirements

The following system requirements have to meet for the driver to work:

- Operating system QNX® Neutrino v6.5, v6.6, 7.0 or 7.1
- PCI/ISA-CAN card (1)

3 Installation

For users of QNX versions prior to 6.5: Please see the previous version 4.03 of this document.

The driver is delivered as TAR/GZ archive. Transfer this archive to the target system, e.g. to folder /tmp, using WinSCP or qconn. On a target, open a root console and issue:

```
# tar -C / -xzf /tmp/dev-can-pcanVariant-A.B.C-qnxN_CPU.tgz
```

The routine will install the following files:

/usr/sbin/dev-can-pcan*Variant*	Driver
/usr/bin/caninfo	Info utility
/usr/bin/canread	Test tool to read CAN bus
/usr/bin/canwrite	Test tool to write CAN messages to the bus

In order to develop under the QNX Momentics IDE under Windows/Linux/MAC only the header file "bcan.h" is needed. It comes with a dedicated Momentics project, together with a number of example files. It is a ZIP archive named dev-can-devel.zip. It must be imported in your workspace using “Import -> General -> Existing projects into workspace”.

4 Starting the dev-can-pcan driver

For a successful start of the driver, the following prerequisites have to be complied:

- active QNX Neutrino runtime environment

The online help for this topic can be called at any time by launching the ´use´ program.

```
# use dev-can-pcanVariant
```

dev-can-pcanPCIe [-Options]* [-pci [option[,option ...]]] [-can[n] [option[,option ...]]]

or

devn-can1000 [-Options]* [-isa [option[,option ...]]] [-can[n] [option[,option ...]]]

Options:	
-p number	Priority to run at (default 21, -1 for „don´t change priority“)
-d name	Device prefix to register (default "/dev/can")
-v	Verbose operation (-vv[v..] - debug level)
-t ms	Timeout value in 100 ms. (max is 21474836, 0=disable, default=0)
-r value	Init can output control register with value (default 0xda)
-f	Force driver start even if 82C200 / sja1000 controller is not found

Where:	
-isa[n]	Begin an ISA description specification [for channel n].
port=0xXXX	I/O port of the interface. The default is 0x300
irq=d	Irq of the interface. The default is 12.
mem=0xXXXX	Memory address of the interface. The default is port i/o. (3)
-pci	Begin a PCI description specification.
vid=0xXXXXXX	Vendor id. The default value is 0x001C
did=0xXXXXXX	Device id. The default value is 0x0001
ind=d	specific PCI index (default is zero. Only required when multiple can PCI cards installed)
irq=d	IRQ to use. Default is auto-detect using PCI server API
-can[n]	Begin a Can description specification [for channel n].
baud=dddd	Bit rate in kBits (10,20,50,100,125,250,500,800,1000 default=125)
obufs=dddd	Number of output buffers (min 2, max ..., default=512)
ibufs=dddd	Number of input buffers (min 2, max ..., default=512)
sammod=d	sample mode. 1 - the bus is sampled 3 times, 0 - the bus is sampled once (default - 0)

Examples:

Start with one channel named "/dev/can/0" using defaults.

```
# dev-can-pcanPCIe
```

Start driver with priority 20 and timeout value 10*100ms = 1sec.

```
# dev-can-pcanPCIe -p20 -t 10
```

Start driver to support two channels. First channel is named “/dev/can/0” at ISA address 0x420, Irq 9 and is configured with 512 input and output buffers. Second channel is named /dev/can/1 at ISA address 0x520, Irq 10 and is configured with 512 input and output buffers.

```
# dev-can-pcan -isa0 port=0x420 irq=9 -isa1 port=0x520 irq=10
```

5 Options

-p value

The -p option allows the adjustment of the driver's priority in the command line. Normally, the driver is invoked with priority 51. Valid values range from 1 to 255. A special function has the value -1 „Do not change priority!". This allows the priority control outside of the driver, e.g. by using the program ´nice´: After the driver launch 'nice --2 dev-can-pcanPCIe -p -1', the driver runs with the priority 12 (assuming the terminal where this command is invoked was launched with the standard priority of 10).

-d name

The -d option is only needed if the standard name ´/dev/can´ is in use by other drivers or applications.

-v

The –v option activates the output of debug messages to ´stdout´ and of error messages to ´stderr´. Several ´-v ´ determine the debug level of the driver.

-t value

The -t option specifies the waiting period during reading from the device until at least one telegram has arrived. Exception: Call of the function *open()* with the flag O_NONBLOCK (see page 7). The waiting period is indicated in multiples of 0.1s. Default value is "0", that means – eternally.

-r value

The -r option determines the contents of the „Output Control Register" (OCR) during the initialization of the CAN controller. Among other things, this register defines the output mode of the transmitter (Bi-phase Output Mode, Test Output Mode, Normal Output Mode, Clock Output Mode). For further information, please contact the hardware manufacturer.

-f

The -f option allows starting the driver even if the existence of the controller was not verified successfully.

-isa[n] Begin of ISA - Specification [for channel n]

port =0xXXX	Base I/O – address of the Can controller. Default – address is 0x300. False address specification leads to abort of the driver (exception: -f option).
mem=0xXXXX	Base Memory – address of the Can controller. Default is port I/O. False address specification leads to abort of the driver (exception: -f option).
irq=d	IRQ number. IRQ 2-15 are supported. Default IRQ is 12.

-pci Begin of PCI - Specification

vid =0xXXXXXX	Vendor Id. Default value is 0x001C. False value specification leads to abort of the driver.
did=0xXXXXXX	Device Id. Default value is 0x0001. False value specification leads to abort of the driver.
ind=d	specific PCI index (default is zero. Only required when multiple can pci cards installed)
irq=d	IRQ number to use. Normally the IRQ is read from the PCI configuration, but sometimes this value is not correct. Then it can be overridden with this option.

-can[n] Begin of CAN - Specification [for channel n]

baud =dddd	Baud rate in Kilobits. Valid values are 10, 20, 50, 100, 125, 250, 500,

	800, and 1000. Default Baud rate is 125.
obufs=dddd	Size of the output buffer; number of telegrams. Minimum is 2, default value is 512.
ibufs=dddd	Size of the input buffer; number of telegrams. Minimum is 2, default value is 512.
sammod=d	sample mode. 1 - the bus is sampled 3 times, 0 - the bus is sampled once (default - 0)

When specifying the parameters, no spaces are allowed, e.g. „baud = 125“ is wrong, correct is „baud=125“.

Unspecified parameters are set to the default value.

6 Guidance for Programming

After the driver start, one or more channels are available, which support typical POSIX C – functions like *open(), read(), and write()*. Core of the communication between user application and driver is a telegram (message), which owns a special structure. After the import of dev-can-devel.zip, this structure can be found in the header file *include/bcan.h*

Each telegram consists of a certain number of data bytes and an identifier that determines the priority of the telegram on the one hand. On the other hand, it is the logical address of the telegram, which is 11 (Standard-CAN-Frame) or 29 (Extended-CAN-Frame) Bits long. Information about the type of the telegram is transmitted over an EXT-Bit (stands for Extended-CAN-Frame) in the field ´flags´ (3). The next entry is the length of the data field. Each telegram can transport up to 8 data bytes; however, it may also be telegrams with 0 bytes data length. In this case, the identifier serves as the storage medium.

RTR-Bit in the field ´flags´: CAN uses the so-called „Broadcast Transmission Protocol“, i.e. all users on the bus can hear how others converse. There is no direct possibility to get a telegram from a certain user. For that, a special telegram with a set RTR-Bit (Remote Transmission Request) is necessary. This telegram requests the user to transmit its data to the bus.

6.1 Telegram Structure for Transmitting and Receiving CAN Messages

```
typedef struct {
        uint32_t              flags;       /* message flags     */
        uint32_t              reserved;    /* reserved          */
        uint32_t              id;          /* message id        */
        struct                timespec timestamp;   /* receive timestamp */
        uint16_t              length;      /* message length    */
        uint8_t               data[CAN_MSG_LENGTH]; /* data           */
} canmsg_t;
```

flags -	Flags determine the communication status and the type of communication:	
	MSG_RTR -	The ***R***emote ***T***ranssmission ***R***equest Bit - Application: Transmit / Receive.
	MSG_OVR -	Data Overrun Flag – set Bit means that at least one telegram was lost on receipt. - Application: Only receiving.
	MSG_EXT -	Set Bit means that the telegram is subject to the CAN 2.0B

		specification (Extended-CAN-Frame) - Application: Transmit / Receive.
	MSG_HIP -	High Priority Message. Telegrams of this type have their own FIFO transmit buffer that is processed first. If there is still another telegram in the transmit buffer, the transmission task is aborted in order to perform the transmission of the HIP telegram. Thereby, one or more transmission telegram may be lost. This sort of telegrams are utilised for e.g. the calibration of CAN – modules, which operate without quartz and on which the internal oscillator has to be set to the desired Baud rate from the outside over the CAN-Bus.
reserved -		reserved
id -		Identifier is the logical address of the transmitter / receiver and the arbitration field. A lower address has a higher priority.
timestamp -		Timestamp states when the telegram has arrived. The Driver sets the members of the timespec structure as follows: • tv_sec – the number of seconds since 1970 (GMT) • tv_nsec – the number of nanoseconds expired in the next second. This value increases by some multiple of nanoseconds, based on the system clock's resolution.
length -		Number of bytes in the data field (0 - 8). *CAN_MSG_LENGTH* is 8.
data		User data for transmitting and receiving.

List of supported Functions for the Resource Manager

The detailed description of the functions can be found in the C - library reference and the POSIX.4 documentation.

The description of the functions is based on the assumption that the driver was invoked with the standard –d option.

open()	Opens the file descriptor „dev/can/<n>“ *int open(const char *path, int oflags, ...);* - Supported oflags: O_RDONLY O_WRONLY O_RDWR O_NONBLOCK O_NONBLOCK – has a special meaning.

	Flag not set: Reading from the device blocks the „client“ by the waiting time defined with–t < Timeout >, as long as there is no telegram in the input buffer. If –t is not or set to „0“, the client is blocked until a telegram is received. Flag set: The „client“ gets an immediate response – the requested or smaller number of telegrams. If no telegram is available, the client receives „-1“ and ***errno*** is set to 11 (Resource temporarily unavailable). - Flags not supported: O_APPEND O_CREATE O_EXCL O_NOCTTY O_TRUNC O_DSYNC O_SYNC O_TEMP O_CACHE
close()	Closes the file descriptor for „/dev/can/<n>“.
read()	Reads from CAN device „/dev/can/<n>“. *ssize_t read(int fields, void *buffer, size_t len);* - **len** must be specified in Bytes and not in the number of telegrams. - **len** must be indicated in multiples of sizeof(canmsg_t).
write()	Writes to CAN device „/dev/can/<n>“ *ssize_t write(int fields, void *buffer, size_t len);* - **len** must be specified in Bytes and not in the number of telegrams. - **len** must be indicated in multiples of sizeof(canmsg_t); otherwise, *write()* returns –1 and ***errno*** is set to 22 (Invalid argument).
lseek()	Not supported.
fpathconf()	Not supported.
dup()	Duplicate descriptor.
opendir()	Not supported.
fstat()	File descriptor status.
chmod()	Assign rights.
chown()	Assign owner.
utime()	Not supported.
fcntl()	File control.
readdir()	Not supported.
rewinddir()	Not supported.

stat()	Status.
ionotify()	Event monitoring for a device. *int ionotify(int fd, int action, int flags, const struct sigevent *event)* Supported flags: _NOTIFY_COND_INPUT, _NOTIFY_COND_OUTPUT
select()	Not supported.
Ioctl()	IO control according to POSIX.4 standard: *int ioctl (int fd, long cmd, void *buf),* - Support of the function ***ioctl()*** depends on the manufacturer. - ***ioctl()*** permits the application to only perfom function supported by the driver. devn-can1000 driver supports the call of *ioctl()* with the following parameters: **Get device attributes:** *int ioctl (int fd,* CNGETA*, void *buf),* where fd - descriptor cmd - CNGETA command "Get Can Attributs" buf - pointer on the structure caninfo_s; Additional version 3.10 supports following ioctl() calls: **Stop the Can device:** *int ioctl (int fd, CNSTOP)* where fd - descriptor cmd - CNSTOP command "Can Stop" **Start the Can device:** *int ioctl (int fd, CNSTART)* where fd - descriptor cmd - CNSTART command "Can Start" **Set device acceptance code:** *int ioctl (int fd, CNSETACC, int *code)* where fd - descriptor cmd - *CNSETACC* command "Set Acceptance Code" *code* - pointer to the acceptance code (8 bit MSB) 8 bit MSBit for CAN 2.A [II)] 32 bit MSByte for CAN 2.B **Set device acceptance mask:** *int ioctl (int fd, CNSETACM, int *mask)*

where
fd - descriptor
cmd - *CNSETACM* Kommando "Set Acceptance Mask"
mask - pointer to the acceptance mask (8 bit MSB)
8 bit MSBit for CAN 2.A [II)]
32 bit MSByte for CAN 2.B

Set device acceptance filter mode (PeliCan mode only):

*int ioctl (int fd, CNSETAFM, int *mode)*
where
fd - descriptor
cmd - *CNSETAFM* command "Set Acceptance Mode"
mode - pointer to the acceptance mode
0 – Dual mode
1 – Single mode (after the start of the drivers)

Set device acceptance filter (PeliCan mode only):

*int ioctl (int fd, CNSETACF, void *buf)*
where
fd - descriptor
cmd - *CNSETACF* command "Set Acceptance Filter"
buf - pointer on the structure `canacf_s`;

CAN Message Zeitstempel Format/Quelle bestimmen:

Set CAN Message timestamp format / source

*int ioctl (int fd, CNSETTS, int *format)*
where
fd - descriptor
cmd - *CNSETTS* command "Set Timestamp format"
format - pointer to the format
0 – System time (struct timespec)
1 – CPU cycle counter (uint64_t)

Be careful about wrapping of the cycle counter. Use the following to calculate how many seconds before the cycle counter wraps:

(~(uint64_t)0) /SYSYPAGE_ENTRY(qtime) -> cycles_per_sec

Flush read and write buffers of device:

int ioctl (int fd, CNFLUSH)
where
fd - descriptor
cmd - *CNFLUSH* command "Flush Device"

Get Device Status:

int ioctl (int fd, CNGETSTAT*, void *buf),*
where
fd - descriptor
cmd - CNGETSTAT command "Get Can State"
buf - pointer on the structure canstat_s;

	Get Register status: *int ioctl (int fd,* CNGETR, *void *buf),* where fd - descriptor cmd - CNGETR command "Get Can Registers" buf - pointer on the structure canregs_s;
	Set device bitrate (PeliCan mode only): *int ioctl (int fd, CNSETBTR, int *baud),* wobei fd - Descriptor cmd - *CNSETBTR* Kommando "Set Bit Rate" baud - pointer to the bitrate parameter (in kBits)
devctl()	Device control. (2)

The structure and the command(s) for ***ioctl()*** are declared in the header file include/*bcan.h*.

6.2 caninfo_s - Structure

```
typedef struct caninfo_s {
       uint32_t              chn;          /* channel number            */
       uint32_t              iobase;       /* i/o base addr             */
       uint32_t              irq;          /* hardware interrupt        */
       uint32_t              baud;         /* baud rate                 */
       uint32_t              ocr;          /* output control register   */
       uint32_t              tov;          /* timout  in 100ms          */
       uint32_t              tx_ok;        /* total packets Txd OK      */
       uint32_t              rx_ok;        /* total packets Rxd OK      */
       uint32_t              tx_bad;       /* total packets Txd Bad     */
       uint32_t              rx_err;       /* total Rx errors           */
       uint32_t              rx_sovr;      /* Software FIFO overruns during Rx  */
       uint32_t              rx_hovr;      /* Hardware FIFO overruns during Rx  */
       uint32_t              er_int;       /* error interrupts          */
       uint32_t              er_bus;       /* bus errors                */
       char                  driver_name    [INFO_NAME_MAX];    /* driver name   */
       char                  process_name[INFO_NAME_MAX];       /* process name  */
       char                  driver_vers [INFO_NAME_MAX];       /* driver version*/
       uint32_t              driver_pid;                        /* driver pid    */
       char                  card_name      [INFO_NAME_MAX];    /* card name     */
       char                  contr_name     [INFO_NAME_MAX];    /* controller    */
} caninfo_t;
```

uint32_t	chn;	Channel number.
uint32_t	iobase;	Base I/O address of the CAN controller.
uint32_t	irq;	Hardware interrupt (IRQ number).
uint32_t	baud;	Baud rate.
uint32_t	ocr;	Contents of the OCRs (Output Control Register).
uint32_t	tov;	Waiting period in 100ms (timeout value).

uint32_t	tx_ok;	Altogether successfully transmitted telegrams. *tx_ok* is counted up at runtime when a telegram was sent by the Can controller and at least one bus user signed it as error free.
uint32_t	rx_ok;	Altogether successfully received telegrams. *rx_ok* is counted up at runtime when a telegram was received by the Can controller and fetched by the driver.
uint32_t	tx_bad;	Altogether lost telegrams (not sent). *tx_bad* is counted up at runtime when a transmission task was aborted by the driver. (See also „flag MSG_HIP“, chapter 6 „Guidance to Programming“)
uint32_t	rx_err;	Entire amount of occurred errors during reception. *rx_err = rx_sovr + rx_hovr.*
uint32_t	rx_sovr;	Software FIFO overrun during reception. *rx_sovr* is counted up at runtime when a telegram was received by the Can controller but not stored by the driver because of a full reception buffer. (See also Parameter ***ibufs*** in the Channel specification)
uint32_t	rx_hovr;	Hardware FIFO overrun during reception. *rx_hovr* is counted up at runtime when an incoming telegram could not be received by the Can controller because the controller reception buffer had not yet been emptied and released by the driver.
uint32_t	er_int;	Number of error messages – interrupts. *er_int* is counted up at runtime when an internal controller error counter has reached the value 96.
uint32_t	er_bus;	Number of "bus errors". *er_bus* is counted up at runtime when the controller transmit error counter has reached the value 255.
char	driver_name[INFO_NAME_MAX];	Name of the driver.
char	process_name[INFO_NAME_MAX];	Process name.
char	driver_vers [INFO_NAME_MAX];	Driver version.

uint32_t	driver_pid;	Pid (Process Identifier) of the driver.
char	card_name [INFO_NAME_MAX];	Name of the card (optional).
char	contr_name [INFO_NAME_MAX];	Name of the controller.

6.3 canstat_s - Structure

```
typedef struct canstat_s {
        uint32_t busoff;        /* 1 - Bus off                */
        uint32_t error;         /* 1 - Error                  */
        uint32_t dataovr;       /* 1 - Data Overrun           */
        int32_t rxbt;           /* rx buffers total           */
        int32_t rxbu;           /* rx buffers used            */
        int32_t rxbf;           /* rx buffers free            */
        int32_t txbt;           /* tx buffers total           */
        int32_t txbu;           /* tx buffers used            */
        int32_t txbf;           /* tx buffers free            */
} canstat_t;
```

uint32_t	busoff;	1 – device in the „bussoff" state
uint32_t	error;	1– device in the „error" state
uint32_t	dataovr;	input data overrun
Int32_t	Rxbt	receive buffers total
Int32_t	rxbu;	receive buffers used
int32_t	rxbf;	receive buffers free
Int32_t	Txbt	transmit buffers total
Int32_t	txbu;	transmit buffers used
int32_t	txbf;	transmit buffers free

6.4 canregs_s - Structure

```
typedef struct canregs_s {
        uint32_t cr;   /* control register                    */
        uint32_t sr;   /* status register                     */
        uint32_t isr;  /* interrupt status register  */
        uint32_t ier;  /* interrupt enable register  */
} canregs_t;
```

uint32_t	cr;	Control Register
uint32_t	sr;	status register
uint32_t	isr;	interrupt status register
unt32_t	Ier	interrupt enable register

6.5 canacf_s - Structure

```
struct canacf_s {              /* can acceptance filter */
        uint8_t         mode;  /* 0 - dual filter mode, 1 - single filter mode */
        uint8_t         acr0;  /* acceptance code register 0 */
        uint8_t         acr1;  /* acceptance code register 1 */
        uint8_t         acr2;  /* acceptance code register 2 */
        uint8_t         acr3;  /* acceptance code register 3 */
        uint8_t         amr0;  /* acceptance mask register 0 */
        uint8_t         amr1;  /* acceptance mask register 1 */
        uint8_t         amr2;  /* acceptance mask register 2 */
        uint8_t         amr3;  /* acceptance mask register 3 */
};
```

INFO_NAME_MAX is 64 Bytes long.

The function *ioctl()* returns the value 0 on success, and -1 on error whereby ***errno*** is set to the appropriate value:

EBADF - Parameter „fd“ – wrong file descriptor
EINVAL - Parameter „cmd“ is false.

7 Example

A typical example with read/write:

```
#include <libc.h>
#include <bcan.h>

#define DATA_ID    1234

int main(int argc, char *argv[])
{

      int         fd, ret, len;
      canmsg_t    msg;
      volatile int     done = 0;
      int         myid = 1313;
      char        mydata[CAN_MSG_LENGTH] = "Hi Can!";

      if ((fd = open("/dev/can/0", O_RDWR)) == -1){

            perror("open() failed");
            exit(EXIT_FAILURE) ;
      }

      while(!done){
            ret = read(fd, &msg , 1 * sizeof(canmsg_t));

            switch(ret){
                  case 0:
                        continue;
                  case -1:
                        fprintf(stderr, "read() failed\n");
                        continue;
                  default:
                        break;
            }

            if (msg.flags & MSG_OVR){
                  /*
                  ...
                   strategy for damage limitation
                  ...
                  */
            }
            switch (msg.id){
                  case 0x80:       /* Sync - message.  Send my data */
                        msg.flags  = 0;
                        msg.id     = myid;
                        msg.timestamp = 0;
                        len = strlen(mydata);
                        len = len<CAN_MSG_LENGTH ? len : CAN_MSG_LENGTH;
                        memcpy(&msg.data, &mydata, len);
                        if (write(fd, &msg, 1 * sizeof(canmsg_t)) <= 0){
                          fprintf(stderr, "write() failed\n");
                        }
```

```
                    break;
                case DATA_ID:
                    /*
                        ...
                        Data analysis.
                        ...
                    */
                default:
                    continue;
            }
        }
        return 0;
}
```

(1) Cards tested:

- PC–ISA-Cards:
  - PC-ISA-CAN of PEAK-System Technik GmbH
  - CANISA-DN of Contemporary Control Systems, Inc.
- PC/104-Cards:
  - CAN104-DN of Contemporary Control Systems, Inc.
- PC–PCI-Cards:
  - PCAN-PCIe of PEAK-System Technik GmbH

(2) in preparation

(3) ISA: Driver only supports CAN 2.A specification (11 bit Identifier)