AdvCan Linux Driver User Manual

V1.03

ADVANTECH *Open eAutomation, Boundless Integration*

Contents

1. Introduction

Advcan driver is a migration from can4linux 3.4 and compatible with can4linux 3.4.The application for can4linux driver can run on advcan driver without any modification.

Now it supports following boards.

PCM-3680	2 port Isolated ISA CAN bus Card.
MIC-3680	2 port Isolated PCI CAN bus Card.
PCI-1680	2 port Isolated PCI CAN bus Card.
PCIE-1680	2 port Isolated PCI CAN bus Card.
ADVANTECH GENERAL CAN PORT (1 PORT)	1 port Isolated PCI CAN bus Card.
ADVANTECH GENERAL CAN PORT (2 PORT)	2 port Isolated PCI CAN bus Card.
ADVANTECH GENERAL CAN PORT (4 PORT)	4 port Isolated PCI CAN bus Card.
ADVANTECH GENERAL CAN PORT (1 PORT, support CANopen)	1 port Isolated PCI CAN bus Card and support CANopen.
ADVANTECH GENERAL CAN PORT (2 PORT, support CANopen)	2 port Isolated PCI CAN bus Card and support CANopen.
ADVANTECH GENERAL CAN PORT (4 PORT, support CANopen)	4 port Isolated PCI CAN bus Card and support CANopen.

This driver supports Linux Kernel 2.4.xx, 2.6.xx and newer, Intel x86 hardware platform.

1.1. System Requirement

- Hardware platform: Intel x86
- Kernel version: 2.4.xx, 2.6.xx and newer

1.2. Driver configuration

The driver itself is highly configurable using the /proc interface of the LINUX kernel.

The following listing shows a typical configuration with four boards:

```
$ grep . /proc/sys/dev/Can/*
```

```
/proc/sys/dev/Can/AccCode:  -1       -1      -1      -1      -1
-1      -1      -1

/proc/sys/dev/Can/AccMask:  -1       -1      -1      -1      -1
-1      -1      -1

/proc/sys/dev/Can/Base:     800      672     832     896     684
696     856     872

/proc/sys/dev/Can/Baud:     125      125     125     250     125
125     125     250

/proc/sys/dev/Can/Chipset:  SJA1000

/proc/sys/dev/Can/IOModel:  pppppppp

/proc/sys/dev/Can/IRQ:      5       7       3       5       5
7       3       5

/proc/sys/dev/Can/Outc:     250      250     250     0       250
250     250     0

/proc/sys/dev/Can/Overrun:  0       0       0       0       0
0       0       0

/proc/sys/dev/Can/RxErr:    0       0       0       0       0
0       0       0

/proc/sys/dev/Can/Timeout:  100      100     100     100     100
100     100     100

/proc/sys/dev/Can/TxErr:    0       0       0       0       0
0       0       0

/proc/sys/dev/Can/dbgMask:  0

/proc/sys/dev/Can/version:  advcan-2.6
```

The following sections are describing the sysctl entries.

AccCode/AccMask

contents of the message acceptance mask and acceptance code registers of 82x200/SJA1000 compatible CAN controllers (see **can_ioctl()**).

Base

CAN controllers base address for each board. Depending of the *IOModel* entry that can be a memory or I/O address.
(Don't modify it if device is PCI boards)

Baud

used bit rate for this board in Kbit/s

Chipset

name of the supported CAN chip used with this boards Read only for this version.

IOModel

One letter for each port. Read-only. Read the CAN register access model. The following models are currently supported:

- m - memory access, the registers are directly mapped into memory(PCM-3680,PCI-1680B1, PCIE-1680)
- p - port I/O, 80x86 specific I/O address range (PCI-1680A1,MIC-3680)

IRQ

used IRQ numbers, one value for each board. (don't modify it if device is PCI boards)

Outc

value of the output control register of the CAN controller.

Overrun

counter for overrun conditions in the CAN controller

RxErr

counter for CAN controller rx error conditions

Timeout

time out value for waiting for a successful transmission

TxErr

counter for CAN controller tx error conditions

version

read only entry containing the drivers version number

Please see also at **can_ioctl()** for some additional descriptions.

For initially writing these sysctl entries after loading the driver (or at any time) a shell script utility does exist. It uses a board configuration file that is written over */proc/sys/ dev/Can* .

```
utils/cansetup PCM3680.conf
```

or, like used in the Makefile:

```
CONFIG := advpci.conf


# load host specific CAN configuration
load:
	@echo "Loading etc/$(CONFIG).conf CAN configuration"
	utils/cansetup etc/$(CONFIG).conf
	echo 0 >/proc/sys/Can/dbgMask
```

Example *.conf files are located in the *etc/* directory.

2. AdvCan Data Structures

Here are the data structures with brief descriptions:

canmsg_t	*The CAN message structure*
CanStatusPar_t	*IOCTL generic CAN controller status request parameter structure*
Command_par_t	*IOCTL Command request parameter structure*
ConfigureRTR_par_t	*IOCTL ConfigureRTR request parameter structure*
Send_par_t	*IOCTL Send request parameter structure*

2.1. Canmsg_t Struct Reference

Detailed Description

The CAN message structure.

Used for all data transfers between the application and the driver using read() or write().

Data Fields

int **flags**

flags, indicating or controlling special message properties

```
#define MSG_RTR (1<<0)          /**< RTR Message */
#define MSG_OVR (1<<1)          /**< CAN controller Msg overflow error */
#define MSG_EXT (1<<2)          /**< extended message format */
#define MSG_SELF (1<<3)         /**< message received from own tx */
#define MSG_PASSIVE (1<<4)  /**< controller in error passive */
#define MSG_BUSOFF (1<<5)   /**< controller Bus Off */
#define MSG_BOVR (1<<7)       /**< receive/transmit buffer overflow */
```

int **cob**

CAN object number, used in Full CAN.

unsigned long	**id** *CAN message ID, 4 bytes.*
timeval	**timestamp** *time stamp for received messages*
short int	**length** *number of bytes in the CAN message*
unsigned char	**data** [CAN_MSG_LENGTH] *message data,8 bytes.* *#define CAN_MSG_LENGTH 8 /**< maximum length of a CAN frame */*

2.2.CanStatusPar Struct Reference

Detailed Description

IOCTL generic CAN controller status request parameter structure.

Data Fields

unsigned int	**baud** *actual bit rate*
unsigned int	**status** *CAN controller status register.*
unsigned int	**error_warning_limit** *the error warning limit*
unsigned int	**rx_errors** *content of RX error counter*
unsigned int	**tx_errors** *content of TX error counter*
unsigned int	**error_code** *content of error code register*
unsigned int	**rx_buffer_size** *size of rx buffer*
unsigned int	**rx_buffer_used** *number of messages*

unsigned int	**tx_buffer_size** *size of tx buffer*
unsigned int	**tx_buffer_used** *number of messages*
unsigned long	**retval** *return value*
unsigned int	**type** *CAN controller / driver type.*

2.3. Command_par_t Struct Reference

Detailed Description

IOCTL Command request parameter structure.

Data Fields

int	**cmd** *special driver command* *will be one of them* *# define CMD_START 1* *# define CMD_STOP 2* *# define CMD_RESET 3* *# define CMD_CLEARBUFFERS 4*
int	**target** *special configuration target*
unsigned long	**val1** *parameter for the target*
unsigned long	**val2** *parameter for the target*
int	**error** *return value*
unsigned long	**retval** *return value*

2.4. ConfigureRTR_par Struct Reference

Detailed Description

IOCTL ConfigureRTR request parameter structure.

Data Fields

unsigned	**message** *CAN message ID.*
canmsg_t *	**Tx** *CAN message struct.*
int	**error** *return value for errno*
unsigned long	**retval** *return value*

2.5. Send_par Struct Reference

Detailed Description

IOCTL Send request parameter structure.

Data Fields

canmsg_t *	**Tx** *CAN message struct.*
int	**error** *return value for error*
unsigned long	**retval** *return value*

3. AdvCan Functions

int open(const char *pathname, int flags)

int ioctl(int fd, int request, ...)

ssize_t read(int fd, void *buf, size_t count)

size_t write(int fd, const char *buf, size_t count)

int close(int fd)

int select(int nfds, fd_set * readfds, fd_set * writefds, fd_set * exceptfds, struct timeval *timeout)

3.1. open Function Reference

Functions

int **open**(const char *pathname, int flags);

opens the CAN device

Function Documentation

int open(const char *pathname, int flags);

opens the CAN device for following operations

Parameters:

pathname device pathname, usual /dev/can?

flags is one of **O_RDONLY**, **O_WRONLY** or **O_RDWR** which request opening the file read-only, write-only or read/write, respectively.

The open call is used to "open" the device. Doing a first initialization according the to values in the /proc/sys/Can file system. Additional an ISR function is assigned to the IRQ.

Returns:

open return the new file descriptor, or -1 if an error occurred (in which case, errno is set appropriately).

ERRORS

the following errors can occur

- ENXIO the file is a device special file and no corresponding device exists.
- EINVAL illegal **minor** device number
- EINVAL wrong IO-model format in /proc/sys/Can/IOmodel
- EBUSY IRQ for hardware is not available
- EBUSY I/O region for hardware is not available

3.2.ioctl Function Reference

Functions

int	**ioctl**(int fd, int request, **...**)
	the CAN controllers control interface

Function Documentation

int ioctl(int fd, int request, **...**);

the CAN controllers control interface

Parameters:

fd	The descriptor to change properties
request	special configuration request
...	traditional a *char* *argp

The *ioctl* function manipulates the underlying device parameters of the CAN special device. In particular, many operating characteristics of character CAN driver may be controlled with *ioctl* requests. The argument *fd* must be an open file descriptor.

An ioctl request has encoded in it whether the argument is an **in** parameter or **out** parameter. Macros and defines used in specifying an *ioctl* request are located in the file can4linux.h .

The following *requests* are defined:

- CAN_IOCTL_COMMAND some commands for start, stop and reset the CAN controller chip
- CAN_IOCTL_CONFIG configure some of the device properties like acceptance filtering, bit timings, mode of the output control register or the optional software message filter configuration.
- CAN_IOCTL_STATUS request the CAN controllers status
- CAN_IOCTL_SEND a single message over the *ioctl* interface
- CAN_IOCTL_RECEIVE poll a receive message

The third argument is a parameter structure depending on the request. These are

```
struct Command_par

struct Config_par

struct CanStatusPar

struct ConfigureRTR_par
struct Send_par
```

The structures above are described in **can4linux.h**

The following items are some configuration targets:

Bit Timing

The bit timing can be set using the *ioctl*(CONFIG,..) and the targets are CONF_TIMING or CONF_BTR. CONF_TIMING should be used only for the predefined Bit Rates (given in kbit/s). With CONF_BTR it is possible to set the CAN controllers bit timing registers individually by providing the values in **val1** (BTR0) and **val2** (BTR1).

Setting the bit timing register

advcan provides direct access to the bit timing registers, besides an implicate setting using the *ioctl* CONF_TIMING and fixed values in Kbit/s. In this case ioctl(can_fd, CAN_IOCTL_CONFIG, &cfg); is used with configuration target CONF_BTR. The configuration structure contains two values, *val1* and *val2* . The following relation to the bit timing registers is used regarding the CAN controller:

```
                 val1      val2

SJA1000           BTR0      BTR1
```

Acceptance Filtering

Basic CAN. In the case of using standard identifiers in Basic CAN mode for receiving CAN messages only the low bytes are used to set acceptance code and mask for bits ID.10 ... ID.3

PeliCAN. For acceptance filtering the entries AccCode and AccMask are used like specified in the controllers manual for **Single Filter Configuration** . Both are 4 byte entries. In the case of using standard identifiers for receiving CAN messages also all 4 bytes can be used. In this case two bytes are used for acceptance code and mask for all 11 identifier bits plus additional the first two data bytes. The SJA1000 is working in the **Single Filter** Mode .

Example for extended message format

```
    Bits
mask 31 30 .....         4  3  2  1  0
code
---------------------------------
ID  28 27 .....         1  0  R  +--+-> unused
                              T
                              R


 acccode =  (id << 3) + (rtr << 2)
```

Example for base message format

```
    Bits
```

```
mask  31 30 .....          23 22 21 20 ... 0
code
-------------------------------------------
ID   10  9.....            1  0  R  +--+-> unused
                                 T
                                 R
```

You have to shift the CAN-ID by 5 bits and two bytes to shift them into ACR0 and ACR1 (acceptance code register)

```
 acccode =  (id << 21) + (rtr << 20)
```

In case of the base format match the content of bits 0...20 is of no interest, it can be 0x00000 or 0xFFFFF.

Returns:

On success, zero is returned. On error, -1 is returned, and error is set appropriately.

Example

```
    Config_par_t  cfg;
    volatile Command_par_t cmd;



    cmd.cmd = CMD_STOP;
    ioctl(can_fd, CAN_IOCTL_COMMAND, &cmd);

    cfg.target = CONF_ACCM;
    cfg.val1    = acc_mask;
    ioctl(can_fd, CAN_IOCTL_CONFIG, &cfg);
    cfg.target = CONF_ACCC;
    cfg.val2    = acc_code;
    ioctl(can_fd, CAN_IOCTL_CONFIG, &cfg);

    cmd.cmd = CMD_START;
    ioctl(can_fd, CAN_IOCTL_COMMAND, &cmd);
```

Other CAN_IOCTL_CONFIG configuration targets ,please refer to can4linux.h in appendix and other source code of IOCTL , please refer to “examples” directory in advcan driver source code.

(see **can4linux.h**)

```
CONF_LISTEN_ONLY_MODE   if set switch to listen only mode
```

```
                        (default false)
CONF_SELF_RECEPTION     if set place sent messages back in the rx queue
                        (default false)
CONF_BTR                 configure bit timing directly registers
CONF_TIMESTAMP            if set fill time stamp value in message structure
                        (default true)
CONF_WAKEUP               if set wake up waiting processes (default true)
```

3.3. read Function Reference

Functions

ssize_t	**read**(int fd, void *buf, size_t count)
	the read system call

Function Documentation

ssize_t read(int fd, void *buf, size_t count);

the read system call

Parameters:

fd	The descriptor to read from.
buf	The destination data buffer (array of CAN **canmsg_t**).
count	The number of CAN message to read.

read() attempts to read up to *count* CAN messages(**not bytes!**) from file descriptor fd into the buffer starting at buf. buf must be large enough to hold the “count” times the size of one CAN messages structure canmsg_t.

```
int got;
canmsg_t rx[80];                    // receive buffer for read()


    got = read(can_fd, rx , 80);
    if( got > 0) {
      ...
    } else {
        // read returned with error
        fprintf(stderr, "- Received got = %d\n", got);
        fflush(stderr);
    }
```

ERRORS

the following errors can occur

- EINVAL **buf** points not to an large enough area,

Returns:

On success, the number of CAN messages read is returned (zero indicates end of file). It is not an error if this number is smaller than the messages requested; this may happen for example because fewer messages are actually available right now, or because read() was interrupted by a signal. On error, -1 is returned, and errno is set appropriately.

3.4.write Function Reference

Functions

size_t	**write**(int fd, const char *buf, size_t count)
	write CAN messages to the network

Function Documentation

size_t write(int fd, const char *buf, size_t count);

write CAN messages to the network

Parameters:

fd The descriptor to write to.

buf The data buffer to write (array of CAN **canmsg_t**).

count

The number of CAN message to write.

write writes up to *count* CAN messages**(not bytes!)** to the CAN controller referenced by the file descriptor fd from the buffer starting at buf.

Errors

the following errors can occur

- EBADF fd is not a valid file descriptor or is not open for writing.
- EINVAL fd is attached to an object which is unsuitable for writing.
- EFAULT buf is outside your accessible address space.
- EINTR The call was interrupted by a signal before any data was written.

Returns:

On success, the number of CAN messages written are returned (zero indicates nothing was written). On error, -1 is returned, and error is set appropriately.

3.5. close File Reference

Functions

int	close(int fd)
	close a file descriptor

Function Documentation

int close(int fd);

close a file descriptor

Parameters:

fd The descriptor to close.

close closes a file descriptor, so that it no longer refers to any device and may be reused.

Returns:

close returns zero on success, or -1 if an error occurred.

ERRORS

the following errors can occur

- BADF **fd** isn't a valid open file descriptor

3.6. select Function Reference

Functions

int	select(int nfds, fd_set * readfds, fd_set * writefds, fd_set * exceptfds, struct timeval *timeout)
	the select system call

Function Documentation

int select(int nfds, fd_set * readfds, fd_set * writefds, fd_set * exceptfds, struct timeval *timeout)

the select system call

Parameters:

ndfs	The number of file handle to be watched
readfds	be watched to see if characters become available for reading

writefds	be watched to see if a write will not block
exceptfds	be watched for exceptions
timeout	an upper bound on the amount of time elapsed before select returns

ERRORS

- ERRORS EBADF An invalid file descriptor was given in one of the sets.
- EINTR A non blocked signal was caught.
- EINVAL n is negative or the value contained within timeout is invalid.
- ENOMEM select was unable to allocate memory for internal tables.

4. Examples Reference

These are some simple examples to test the communication between two CAN channels.

- **receive-block**

Block mode to receive message

- **transmit-block**

Block mode to transmit message

- **receive-nonblock**

Non block mode to receive message

- **transmit-nonblock**

Non block mode to transmit message

- **receive-select**

Simple receiving using the select() call to wait for CAN messages

- **transmit-select**

Simple transmit using the select() call

- **send_ioctl**

Simple transmit using the SEND ioctl command

- **baud**

Simple driver test: change the bit rate registers with ioctl()
The change itself stays after program is finished.
You can check it by read
cat /proc/sys/Can/Baud
Before and after using this command

- **acceptance**

Simple driver test: change the message filter with ioctl()
The change itself stays after program is finished.
You can check it by read
cat /proc/sys/Can/AccCode /proc/sys/Can/AccMask
Before and after using this command

- **singlefilter**

In this example, when set accept code to only accept the message which id = 10 and rtr = 0"

Using /dev/can0 device to accept message

Using /dev/can1 to transmit message

- **selfreception**

Self reception example，when you run this example, you should run another receive example to receive the message transmit by selfreception and at the same time, selfreception will also receive the message transmit by itself.

Appendix

can4linux.h

```
/*
 * can4linux.h - can4linux CAN driver module
 *
 * This file is subject to the terms and conditions of the GNU General Public
 * License.  See the file "COPYING" in the main directory of this archive
 * for more details.
 *
 *
 *------------------------------------------------------------------
 *
 *
 *--------------------------------------------------------------------------
 *
 *
 *
 *
 *--------------------------------------------------------------------------
 */


#ifndef __CAN_H
#define __CAN_H

```

```

# define CAN4LINUXVERSION 0x0304 /*(Version 3.3)*/

#ifndef __KERNEL__
#include <sys/time.h>
#endif
 /*---------- the can message structure */

#define CAN_MSG_LENGTH 8
#define MSG_RTR        (1<<0)
#define MSG_OVR        (1<<1)
#define MSG_EXT        (1<<2)
#define MSG_SELF       (1<<3)
#define MSG_PASSIVE    (1<<4)
#define MSG_BUSOFF     (1<<5)
#define MSG_           (1<<6)
#define MSG_BOVR       (1<<7)
#define MSG_ERR_MASK   (MSG_OVR + MSG_PASSIVE + MSG_BUSOFF + MSG_BOVR)

typedef struct {
   int           flags;
   int           cob;
   unsigned  long id;
   struct timeval  timestamp;
   short     int  length;
   unsigned  char data[CAN_MSG_LENGTH];
} canmsg_t;



/* Use 'c' as magic number, follow chapter 6 of LDD3 */
#define CAN4L_IOC_MAGIC 'c'

#define CAN_IOCTL_COMMAND       0
#define CAN_IOCTL_CONFIG        1
```

```
#define CAN_IOCTL_SEND          2
#define CAN_IOCTL_RECEIVE       3
#define CAN_IOCTL_CONFIGURERTR   4
#define CAN_IOCTL_STATUS        5
/*---------- CAN ioctl parameter types */

struct Command_par {
    int cmd;
    int target;
    unsigned long val1;
    unsigned long val2;
    int error;
    unsigned long retval;
};


typedef struct Command_par Command_par_t ;
typedef struct Command_par  Config_par_t ;
typedef struct CanStatusPar {
    unsigned int baud;
    unsigned int status;
    unsigned int error_warning_limit;
    unsigned int rx_errors;
    unsigned int tx_errors;
    unsigned int error_code;
    unsigned int rx_buffer_size;
    unsigned int rx_buffer_used;
    unsigned int tx_buffer_size;
    unsigned int tx_buffer_used;
    unsigned long retval;
    unsigned int type;
} CanStatusPar_t;

#define CAN_TYPE_UNSPEC        0
#define CAN_TYPE_SJA1000       1
#define CAN_TYPE_FlexCAN       2
```

```
#define CAN_TYPE_TouCAN        3
#define CAN_TYPE_82527         4
#define CAN_TYPE_TwinCAN       5
#define CAN_TYPE_BlackFinCAN    6


typedef struct Send_par {
   canmsg_t *Tx;
   int error;
   unsigned long retval;
} Send_par_t ;

typedef struct Receive_par {
   canmsg_t *Rx;
   int error;
   unsigned long retval;
} Receive_par_t ;

typedef struct ConfigureRTR_par {
   unsigned message;
   canmsg_t *Tx;
   int error;
   unsigned long retval;
} ConfigureRTR_par_t ;

# define CMD_START            1
# define CMD_STOP             2
# define CMD_RESET            3
# define CMD_CLEARBUFFERS      4




# define CONF_ACC      0      /* mask and code */
# define CONF_ACCM     1      /* mask only */
# define CONF_ACCC     2      /* code only */
# define CONF_TIMING    3       /* bit timing */
# define CONF_OMODE     4       /* output control register */
# define CONF_FILTER    5
# define CONF_FENABLE   6
# define CONF_FDISABLE  7
# define CONF_LISTEN_ONLY_MODE  8       /* for SJA1000 PeliCAN */
# define CONF_SELF_RECEPTION    9       /* */
# define CONF_BTR              10      /* set direct bit timing registers
                                        (SJA1000)  */
# define CONF_TIMESTAMP        11      /* use TS in received messages */
# define CONF_WAKEUP           12      /* wake up processes */

#endif  /* __CAN_H */
```