Linux/RT-Linux CAN Driver (LinCAN)

Linux/RT-Linux CAN Driver (LinCAN)

by Pavel Pisa

Published June 2005

Table of Contents

List of Figures

Preface

LinCAN is a Linux kernel module that implements a CAN driver capable of working with multiple cards, even with different chips and IO methods. Each communication object can be accessed from multiple applications concurrently. It supports RT-Linux, 2.2, 2.4, and 2.6 with fully implemented select, poll, fasync, O_NONBLOCK, and O_SYNC semantics and multithreaded read/write capabilities. It works with the common Intel i82527, Philips 82c200, and Philips SJA1000 (in standard and PeliCAN mode) CAN controllers. It is part of a set of CAN/CANopen related components developed as part of OCERA framework.

Chapter 1. Linux/RT-Linux CAN Driver (LinCAN)

The LINCAN is an implementation of the Linux device driver supporting more CAN controller chips and many CAN interface boards. Its implementation has long history already. The OCERA version of the driver adds new features, continuous enhancements and reimplementation of structure of the driver. Most important feature is that driver supports multiple open of one communication object from more Linux and even RT-Linux applications and threads. The usage of the driver is tightly coupled to the virtual CAN API interface component which hides driver low level interface to the application programmers.

1.1. LinCAN Summary

1.1.1. Summary

Name of the component
    Linux CAN Driver (LINCAN)
Author
    Pavel Pisa
    Arnaud Westenberg
    Tomasz Motylewski

Maintainer
    Pavel Pisa
LinCAN Internet resources
    http://www.ocera.org OCERA project home page
    http://sourceforge.net/projects/ocera OCERA SourceForge project page. The OCERA CVS relative path to LinCAN driver sources is
    `ocera/components/comm/can/lincan`.
    http://cmp.felk.cvut.cz/~pisa/can local testing directory

Reviewer
    The previous driver versions were tested by more users. The actual version has been tested at CTU by more OCERA developers, by Unicontrols and by BFAD GmbH, which use pre-OCERA and current version of the driver in their products.
    List of the cards tested with latest version of the driver:
    • PC104 Advantech PCM3680 dual channel board on 2.4 RT-Linux enabled kernel
    • PiKRON ISA card on 2.4.and 2.6 Linux kernels
    • BfaD DIMM PC card on 2.4 RT-Linux enabled kernel
    • KVASER pcican-q on 2.6 Linux kernel and on 2.4 RT-Linux enabled kernel
    • virtual board tested on all systems as well

Supported layers

    • High-level available
    Linux device interface available for soft real-time Linux only and for mixed-mode RT-Linux/Linux driver compilation

    • Low-level available

RT-Linux device is registered only for mixed-mode RT-Linux/Linux driver compilation. The driver messages transmition and reception runs in hard real-time threads in such case.

Version

lincan-0.3

Status

Beta

Dependencies

The driver requires CAN interface hardware for access to real CAN bus.

Driver can be used even without hardware if a virtual board is configured. This setup is useful for testing of interworking of other CAN components.

Linux kernels from 2.2.x, 2.4.x and 2.6.x series are fully supported.

The RT-Linux version 3.2 or OCERA RT-Linux enabled system is required for hard real-time use.

The RT-Linux version requires RT-Linux `malloc`, which is part of OCERA RT-Linux version and can be downloaded for older RT-Linux versions .

The use of VCA API library is suggested for seamless application transitions between driver kinds and versions.

Supported hardware (some not tested)

- Advantech PC-104 PCM3680 dual channel board
- PiKRON ISA card
- BfaD DIMM PC card
- KVASER PCIcan-Q, PCIcan-D, PCIcan-S
- KVASER PCcan-Q, PCcan-D, PCcan-S, PCcan-F
- MPL AG PIP5, PIP6, PIP7, PIP8
- NSI PC-104 board CAN104
- Contemporary Controls PC-104 board CAN104
- Arcom Control Systems PC-104 board AIM104CAN
- IXXAT ISA board PC-I03
- SECO PC-104 board M436
- Board support template sources for yet unsupported hardware
- Virtual board

Release date

February 2004

1.2. LinCAN Driver Description

1.2.1. Introduction

The LinCAN driver is the loadable module for the Linux kernel which implements CAN driver. The driver communicates and controls one or more CAN controllers chips. Each chip/CAN interface is represented to the applications as one or more CAN message objects accessible as character devices. The application can open the character device and use `read`/`write` system calls for CAN messages transmission or reception through the connected message object. The parameters of the message object can be modified by the `IOCTL` system call. The closing of the character device releases resources allocated by

the application. The present version of the driver supports three most common CAN controllers:

- Intel i82527 chips
- Philips 82c200 chips
- Philips SJA1000 chips in standard and PeliCAN mode

The intelligent CAN/CANopen cards should be supported by in the near future. One of such cards is P-CAN series of cards produced by Unicontrols. The driver contains support for more than ten CAN cards basic types with different combinations of the above mentioned chips. Not all card types are held by OCERA members, but CTU has and tested more SJA1000 type cards and will test some i82527 cards in near future.

1.3. LinCAN Driver System Level API

1.3.1. Device Files and Message Structure

Each driver is a subsystem which has no direct application level API. The operating system is responsible for user space calls transformation into driver functions calls or dispatch routines invocations. The CAN driver is implemented as a character device with the standard device node names `/dev/can0`, `/dev/can1`, etc. The application program communicates with the driver through the standard system low level input/output primitives (`open`, `close`, `read`, `write`, `select` and `ioctl`). The CAN driver convention of usage of these functions is described in the next subsection.

The `read` and `write` functions need to transfer one or more CAN messages. The structure `canmsg_t` is defined for this purpose and is defined in include file `can/can.h`. The `canmsg_t` structure has next fields:

```
struct canmsg_t {
    int flags;
    int cob;
    unsigned long   id;
    canmsg_tstamp_t timestamp;
    unsigned short  length;
    unsigned char   data[CAN_MSG_LENGTH];
} PACKED;
```

flags

> The flags field holds information about message type. The bit `MSG_RTR` marks remote transmission request messages. Writing of such message into the CAN message object handle results in transmission of the RTR message. The RTR message can be received by the read call if no buffer with corresponding ID is pre-filled in the driver. The bit `MSG_EXT` indicates that the message with extended (bit 29 set) ID will be send or was received. The bit `MSG_OVR` is intended for fast indication of the reception message queue overfill. The transmitted messages could be distributed back to the local clients after transmition to the CAN bus. Such messages are marked by `MSG_LOCAL` bit.

cob

> The field reserved for a holding message communication object number. It could be used for serialization of received messages from more message object into one message queue in the future.

id

> CAN message ID.

timestamp

> The field intended for storing of the message reception time.

length

> The number of the data bytes send or received in the CAN message. The number of data load bytes is from 0 to 8.

data

   The byte array holding message data.

As was mentioned above, direct communication with the driver through system calls is not encouraged because this interface is partially system dependent and cannot be ported to all environments. The suggested alternative is to use OCERA provided VCA library which defines the portable and clean interface to the CAN driver implementation.

The other issue is addition of the support for new CAN interface boards and CAN controller chips. The subsection Board Support Functions describes template functions, which needs to be implemented for newly supported board. The template of board support can be found in the file `src/template.c`.

The other task for more brave souls is addition of the support for the unsupported chip type. The source supporting the SJA1000 chip in the PeliCAN mode can serve as an example. The full source of this chip support is stored in the file `src/sja1000p.c`. The subsection Chip Support Functions describes basic functions necessary for the new chip support.

1.3.2. CAN Driver File Operations

open

Name

`open` — message communication object open system call

Synopsis

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

Arguments

pathname

   The path to driver device node is specified there. The conventional device names for Linux CAN driver are `/dev/can0`, `/dev/can1`, etc.

flags

   flags modifying style of open call. The standard `O_RDWR` mode should be used for CAN device. The mode `O_NOBLOCK` can be used with driver as well. This mode results in immediate return of `read` and `write`.

Description

Returns negative number in the case of error. Returns the file descriptor for named CAN message object in other cases.

close

Name

`close` — message communication object close system call

Synopsis

```
int close (int fd);
```

Arguments

fd

> file descriptor to opened can message communication object

Description

Returns negative number in the case of error.

read

Name

read — reads received CAN messages from message object

Synopsis

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

Arguments

fd

> file descriptor to opened can message communication object

buf

> pointer to array of canmsg_t structures.

count

> size of message array buffer in number of bytes

Description

Returns negative value in the case of error else returns number of read bytes which is multiple of canmsg_t structure size.

write

Name

write — writes CAN messages to message object for transmission

Synopsis

```
ssize_t write(int fd, const void * buf, size_t count);
```

Arguments

fd

> file descriptor to opened can message communication object

buf

> pointer to array of canmsg_t structures.

count

size of message array buffer in number of bytes. The parameter informs driver about number of messages prepared for transmission and should be multiple of `canmsg_t` structure size.

Description

Returns negative value in the case of error else returns number of bytes successfully stored into message object transmission queue. The positive returned number is multiple of `canmsg_t` structure size.

struct canfilt_t

Name

`struct canfilt_t` — structure for acceptance filter setup

Synopsis

```
struct canfilt_t {
  int flags;
  int queid;
  int cob;
  unsigned long id;
  unsigned long mask;
};
```

Members

flags

message flags

`MSG_RTR` .. message is Remote Transmission Request,

`MSG_EXT` .. message with extended ID,

`MSG_OVR` .. indication of queue overflow condition,

`MSG_LOCAL` .. message originates from this node.

there are corresponding mask bits `MSG_RTR_MASK`, `MSG_EXT_MASK`, `MSG_LOCAL_MASK`.

`MSG_PROCESSLOCAL` enables local messages processing in the combination with global setting

queid

CAN queue identification in the case of the multiple queues per one user (open instance)

cob

communication object number (not used)

id

selected required value of cared ID id bits

mask

select bits significant for the comparison;

1 .. take care about corresponding ID bit,

0 .. don't care

IOCTL CANQUE_FILTER

Name

IOCTL CANQUE_FILTER — Sets acceptance filter for CAN queue connected to client state

Synopsis

```
int ioctl(int fd, int command = CANQUE_FILTER, struct canfilt_t * filt);
```

Arguments

fd

    file descriptor to opened can message communication object

command

    Denotes CAN queue filter command, CANQUE_FILTER

filt

    pointer to the canfilt_t structure.

Description

The CANQUE_FILTER IOCTL invocation sets acceptance mask of associated canqueue to specified parameters. Actual version of the driver changes filter of the default reception queue. The filed queid should be initialized to zero to support compatibility with future driver versions.

The call returns negative value in the case of error.

IOCTL CANQUE_FLUSH

Name

IOCTL CANQUE_FLUSH — Flushes messages from receiption CAN queue

Synopsis

```
int ioctl(int fd, int command = CANQUE_FLUSH, int queid);
```

Arguments

fd

    file descriptor to opened can message communication object

command

    Denotes CAN queue flush command, CANQUE_FLUSH

queid

    Should be initialized to zero to support compatibility with future driver versions

Description

The call flushes all messages from the CAN queue.

The call returns negative value in the case of error.

1.4. LinCAN Driver Architecture

The LinCAN provides simultaneous queued communication for more concurrent running applications.
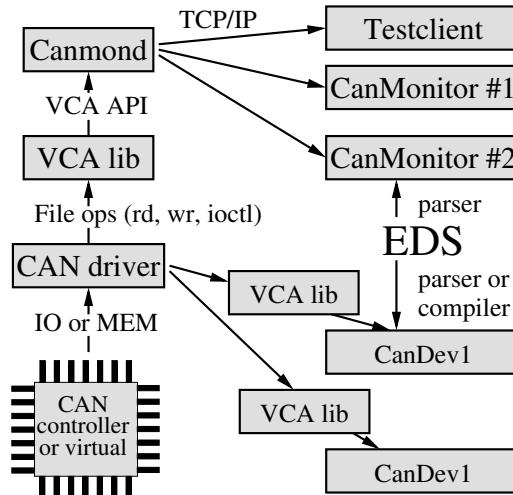
Figure 1-1. LinCAN architecture

Even each of communication object can be used by one or more applications, which connects to the communication object internal representation by means of CAN FIFO queues. This enables to build complex systems based even on card and chips, which provides only one communication objects (for example SJA1000).

The driver can be configured to provide virtual CAN board (software emulated message object) to test CAN components on the Linux system without hardware required to connect to the real CAN bus. The example configuration of the CAN network components connected to one real or virtual communication object of LinCAN driver is shown in figure Figure 1-1. The communication object is used by the CAN monitor daemon and two CANopen devices implemented by OCERA CanDev component. The actual system dependent driver API is hidden to applications under VCA library. The CAN monitor daemon translates CAN messages to TCP/IP network for Java based platform independent CAN monitor and C based test client.

Each communication object is represented as character device file. The devices can be opened and closed by applications in blocking or non-blocking mode. LinCAN client application state, chip and object configurations are controlled by IOCTL system call. One or more CAN messages can be sent or received through write/read system calls. The data read from or written to the driver are formed from sequence of fixed size structures representing CAN messages.

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

The LinCAN driver version 0.2 has rewritten infrastructure based on message FIFOs organized into oriented edges between chip drivers (`structure chip_t`) message objects representations (`structure msgobj_t`) and open device file instances state (`structure canuser_t`). The complete relationship between CAN hardware representation and open instances is illustrated in the figure Figure 1-4.

The message FIFO (`structure canque_fifo_t`) initialization code allocates configurable number of slots capable to hold one message.
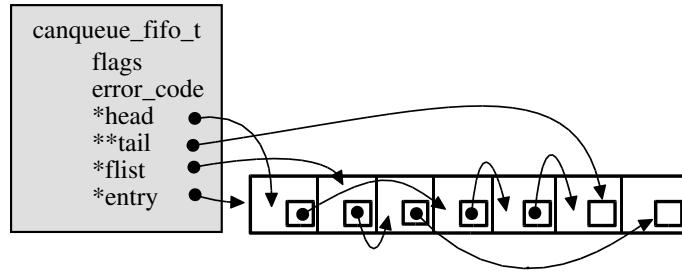
Figure 1-2. LinCAN message FIFO implementation

The all slots are linked to the free list after initialization. The slot can be requested by FIFO input side by function `canque_fifo_get_inslot`. The slot is filled by message data and is linked into FIFO queue by function `canque_fifo_put_inslot`. If previously requested slot is not successfully filled by data, it can be released by `canque_fifo_abort_inslot`. The output side of the FIFO tests presence of ready slots by function `canque_fifo_test_outslot`. If the slot is returned by this function, it is processed and released by function `canque_fifo_free_outslot`. The processing can be postponed in the case of bus error or higher priority message processing request by `canque_fifo_again_outslot` function. All these functions are optimized to be fast and short, which enables to synchronize them by spin-lock semaphores and guarantee atomic nature of them. The FIFO implementation is illustrated in the figure Figure 1-2.
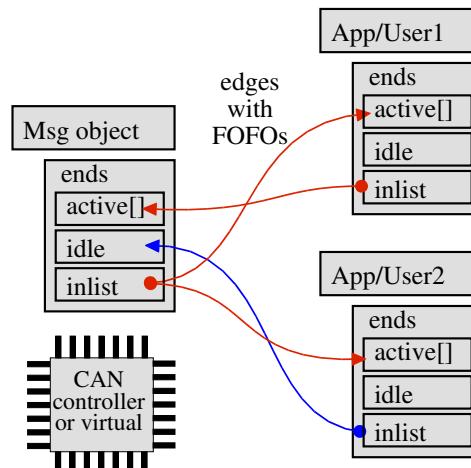
Figure 1-3. LinCAN driver message flow graph edges

The low level message FIFOs are wrapped by CAN edges structures (`canque_edge_t`), which are used for message passing between all components of the driver. The actual version of LinCAN driver uses oriented edges to connect Linux and RT-Linux clients/users with chips and communication objects. Each entity, which is able to hold edge ends, has to be equipped by `canque_ends_t` structure. The input ends of edges/FIFOs are held on `inlist`. The inactive/empty out ends of the edges are held on a `idle` list and active out ends are held on a `active` list corresponding to the edge priority. The `canque_fifo_test_outslot` function can determine by examination of active lists if there is message to accept/process. This concept makes possible to use same type of edges for outgoing and incoming directions. The concept of edges can be even used for message filtering by priority or acceptance masks. It is prepared for future targeting messages to predefined message objects according to their priority or type and for redundant and fault tolerant message distribution into more CAN buses. Message concentration, virtual nodes and other special processing can be implemented above this concept as well. The example of interconnection of one communication object with two users/open instances is illustrated in the picture Figure 1-3. Three edges/FIFOs are in the active state and one edge/FIFO is empty in the shown example.
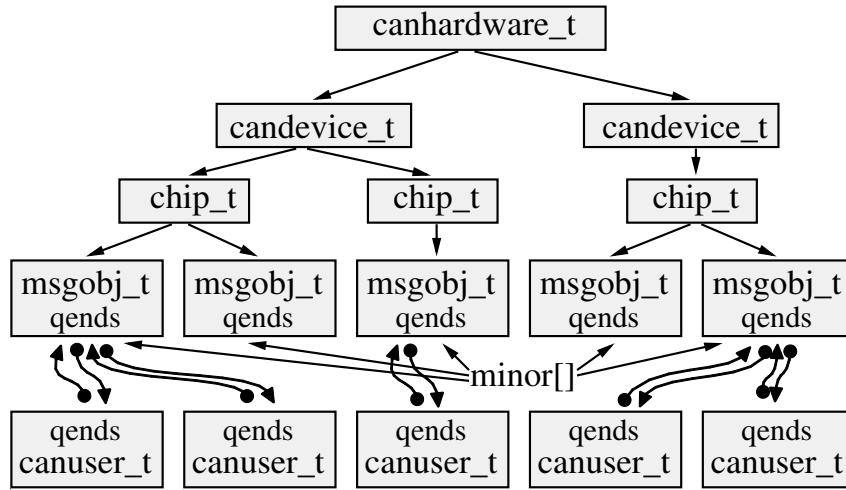
Figure 1-4. CAN hardware model in the LinCAN driver

The figure Figure 1-4 is example of object inside LinCAN driver representing system with two boards, three chips and more communication objects. Some of these objects are used by one or more applications. The object open instances are represented as `canuser_t` structures.

1.5. Driver History and Implementation Issues

The development of the CAN drivers for Linux has long history. We have been faced before two basic alternatives, start new project from scratch or use some other project as basis of our development. The first approach could lead faster to more simple and clean internal architecture but it would mean to introduce new driver with probably incompatible interface unusable for already existing applications. The support of many types of cards is thing which takes long time as well. More existing projects aimed to development of a Linux CAN driver has been analyzed:

Original LDDK CAN driver project

> The driver project aborted on the kernel evolution and LDDK concept. The LDDK tried to prepare infrastructure for development of the kernel version independent character device drivers written in meta code. The goal was top-ranking, but it proves, that well written "C" language driver can be more portable than the LDDK complex infrastructure.

can4linux-0.9 by PORT GmbH

> This is version of the above LDDK driver maintained by Port GmbH. The card type is hard compiled into the driver by selected defines and only Philips 82c200 chips are supported.

CanFestival

> The big advantage of this driver is an integrated support for the RT-Linux, but driver implementation is highly coupled to one card. Some concepts of the driver are interesting but the driver has the hard-coded number of message queues.

can-0.7.1 by Arnaud Westenberg

> This driver has its roots in the LDDK project as well. The original LDDK concept has been eliminated in the driver source and necessary adaptation of the driver for the different Linux kernel versions is achieved by the controllable number of defines and conditional compilation. There is more independent contributors. The main advantages of the driver are support of many cards working in parallel, IO and memory space chip connection support and more cards of different types can be selected at module load time. There exist more users and applications compatible with the driver interface. Disadvantages of the original version of this driver

are non-optimal infrastructure, non-portable make system and lack of the select support.

The responsible OCERA developers selected the can-0.7.1 driver as a base of their development for next reasons:

- Best support for more cards in system
- Supports for many types of cards
- The internal abstraction of the peripheral access method and the chip support

The most important features added by OCERA development team are:

- Added the select system call support
- The support for our memory mapped ISA card added, which proved simplicity of addition of the support for new type of CAN cards
- Added devfs support
- Revised and bug-fixed the IRQ support in the first phase
- Added support for 2.6.x kernels
- Rebuilt the make system to compile options fully follow the running kernel options, cross-compilation still possible when the kernel location and compiler is specified. The driver checked with more 2.2.x, 2.4.x and 2.6.x kernels and hardware configurations.
- Cleaned-up synchronization required to support 2.6.x SMP kernels and enhanced 2.4.x kernels performance
- The deeper rebuilt of the driver infrastructure to enable porting to more systems (most important RT-Linux). The naive FIFO implementation has been replaced by robust CAN queues, edges and ends framework. The big advantage of continuous development is ability to keep compatibility with many cards and applications
- The infrastructure rewrite enabled to support multiple opening of the single minor device
- Support for individual queues message acceptance filters added
- The driver setup functions modified to enable PCI and USB hardware hot-swapping and PnP recognition in the future
- Added support for KVASER PCI cards family
- Added support for virtual can board for more CAN/CANopen components interworking testing on single computer without real CAN hardware.
- The conditional compilation mode for Linux/RT-Linux support has been added. The driver manipulates with chips and boards from RT-Linux hard real-time worker threads in that compilation mode. The POSIX device file interface is provided for RT-Linux threads in parallel to the standard Linux device interface.
- Work on support for first of intelligent CAN/CANopen cards has been started

The possible future enhancements

- Cleanup and enhance RTR processing. Add some support for emulated RTR processing for SJA1000 chips
- Enhance clients API to gain full advantages of possibility to connect more CAN queues with different priorities to the one user state structure
- Add support for more CAN cards and chips (82C900 comes to mind)
- Add support for XILINX FPGA based board in development at CTU. There already exists VHDL source for the chip core, connect it to PC-104 bus and LinCAN driver
- Do next steps in the PCI cards support cleanup and add Linux 2.6.x sysfs support

1.6. LinCAN Driver Internals

1.6.1. Basic Driver Data Structures

struct canhardware_t

Name

struct canhardware_t — structure representing pointers to all CAN boards

Synopsis

```
struct canhardware_t {
  int nr_boards;
  struct rtr_id * rtr_queue;
  can_spinlock_t rtr_lock;
  struct candevice_t * candevice[MAX_HW_CARDS];
};
```

Members

nr_boards

   number of present boards

rtr_queue

   RTR - remote transmission request queue (expect some changes there)

rtr_lock

   locking for RTR queue

candevice[MAX_HW_CARDS]

   array of pointers to CAN devices/boards

struct candevice_t

Name

struct candevice_t — CAN device/board structure

Synopsis

```
struct candevice_t {
  char * hwname;
  int candev_idx;
  unsigned long io_addr;
  unsigned long res_addr;
  unsigned long dev_base_addr;
  unsigned int flags;
  int nr_all_chips;
  int nr_82527_chips;
  int nr_sja1000_chips;
  struct canchip_t * chip[MAX_HW_CHIPS];
  struct hwspecops_t * hwspecops;
  struct canhardware_t * hosthardware_p;
  union sysdevptr;
};
```

Members

hwname

   text string with board type

candev_idx

> board index in canhardware_t.candevice[]

io_addr

> IO/physical MEM address

res_addr

> optional reset register port

dev_base_addr

> CPU translated IO/virtual MEM address

flags

> board flags: `PROGRAMMABLE_IRQ` .. interrupt number can be programmed into board

nr_all_chips

> number of chips present on the board

nr_82527_chips

> number of Intel 8257 chips

nr_sja1000_chips

> number of Philips SJA100 chips

chip[MAX_HW_CHIPS]

> array of pointers to the chip structures

hwspecops

> pointer to board specific operations

hosthardware_p

> pointer to the root hardware structure

sysdevptr

> union reserved for pointer to bus specific device structure (case *pcidev* is used for PCI devices)

Description

The structure represent configuration and state of associated board. The driver infrastructure prepares this structure and calls board type specific `board_register` function. The board support provided register function fills right function pointers in *hwspecops* structure. Then driver setup calls functions `init_hw_data`, `init_chip_data`, `init_chip_data`, `init_obj_data` and `program_irq`. Function `init_hw_data` and `init_chip_data` have to specify number and types of connected chips or objects respectively. The use of *nr_all_chips* is preferred over use of fields *nr_82527_chips* and *nr_sja1000_chips* in the board non-specific functions. The *io_addr* and *dev_base_addr* is filled from module parameters to the same value. The request_io function can fix-up *dev_base_addr* field if virtual address is different than bus address.

struct canchip_t

Name

struct canchip_t — CAN chip state and type information

Synopsis

```
struct canchip_t {
    char * chip_type;
    int chip_idx;
    int chip_irq;
```

```
    unsigned long chip_base_addr;
    unsigned int flags;
    long clock;
    long baudrate;
    void (* write_register) (unsigned data,unsigned long address);
    unsigned (* read_register) (unsigned long address);
    void * chip_data;
    unsigned short sja_cdr_reg;
    unsigned short sja_ocr_reg;
    unsigned short int_cpu_reg;
    unsigned short int_clk_reg;
    unsigned short int_bus_reg;
    struct msgobj_t * msgobj[MAX_MSGOBJS];
    struct chipspecops_t * chipspecops;
    struct candevice_t * hostdevice;
    int max_objects;
    can_spinlock_t chip_lock;
#ifdef CAN_WITH_RTL
  pthread_t worker_thread;
  unsigned long pend_flags;
#endif
};
```

Members

chip_type

   text string describing chip type

chip_idx

   index of the chip in candevice_t.chip[] array

chip_irq

   chip interrupt number if any

chip_base_addr

   chip base address in the CPU IO or virtual memory space

flags

   chip flags: `CHIP_CONFIGURED` .. chip is configured, `CHIP_SEGMENTED` .. access to
   the chip is segmented (mainly for i82527 chips)

clock

   chip base clock frequency in Hz

baudrate

   selected chip baudrate in Hz

write_register

   write chip register function copy

read_register

   read chip register function copy

chip_data

   pointer for optional chip specific data extension

sja_cdr_reg

   SJA specific register - holds hardware specific options for the Clock Divider
   register. Options defined in the sja1000.h file: `CDR_CLKOUT_MASK`, `CDR_CLK_OFF`,
   `CDR_RXINPEN`, `CDR_CBP`, `CDR_PELICAN`

sja_ocr_reg

   SJA specific register - hold hardware specific options for the Output Control reg-
   ister. Options defined in the sja1000.h file: `OCR_MODE_BIPHASE`, `OCR_MODE_TEST`,
   `OCR_MODE_NORMAL`, `OCR_MODE_CLOCK`, `OCR_TX0_LH`, `OCR_TX1_ZZ`.

int_cpu_reg

   Intel specific register - holds hardware specific options for the CPU Interface
   register. Options defined in the i82527.h file: `iCPU_CEN`, `iCPU_MUX`, `iCPU_SLP`,
   `iCPU_PWD`, `iCPU_DMC`, `iCPU_DSC`, `iCPU_RST`.

int_clk_reg

   Intel specific register - holds hardware specific options for the Clock Out register. Options defined in the i82527.h file: `iCLK_CD0`, `iCLK_CD1`, `iCLK_CD2`, `iCLK_CD3`, `iCLK_SL0`, `iCLK_SL1`.

int_bus_reg

   Intel specific register - holds hardware specific options for the Bus Configuration register. Options defined in the i82527.h file: `iBUS_DR0`, `iBUS_DR1`, `iBUS_DT1`, `iBUS_POL`, `iBUS_CBY`.

msgobj[MAX_MSGOBJS]

   array of pointers to individual communication objects

chipspecops

   pointer to the set of chip specific object filled by `init_chip_data` function

hostdevice

   pointer to chip hosting board

max_objects

   maximal number of communication objects connected to this chip

chip_lock

   reserved for synchronization of the chip supporting routines (not used in the current driver version)

worker_thread

   chip worker thread ID (RT-Linux specific field)

pend_flags

   holds information about pending interrupt and `tx_wake` operations (RT-Linux specific field). Masks values: `MSGOBJ_TX_REQUEST` .. some of the message objects requires `tx_wake` call, `MSGOBJ_IRQ_REQUEST` .. chip interrupt processing required `MSGOBJ_WORKER_WAKE` .. marks, that worker thread should be waked for some of above reasons

Description

The fields *write_register* and *read_register* are copied from corresponding fields from *hwspecops* structure (chip->hostdevice->hwspecops->write_register and chip->hostdevice->hwspecops->read_register) to speedup `can_write_reg` and `can_read_reg` functions.

struct msgobj_t

Name

`struct msgobj_t` — structure holding communication object state

Synopsis

```
struct msgobj_t {
  unsigned long obj_base_addr;
  unsigned int minor;
  unsigned int object;
  unsigned long obj_flags;
  int ret;
  struct canque_ends_t * qends;
  struct canque_edge_t * tx_qedge;
  struct canque_slot_t * tx_slot;
  int tx_retry_cnt;
  struct timer_list tx_timeout;
  struct canmsg_t rx_msg;
  struct canchip_t * hostchip;
```

```
        unsigned long rx_preconfig_id;
        atomic_t obj_used;
        struct list_head obj_users;
    };
```

Members

obj_base_addr

minor

    associated device minor number

object

    object number in canchip_t structure +1

obj_flags

    message object specific flags. Masks values: `MSGOBJ_TX_REQUEST` .. the message
    object requests TX activation `MSGOBJ_TX_LOCK` .. some IRQ routine or callback on
    some CPU is running inside TX activation processing code

ret

    field holding status of the last Tx operation

qends

    pointer to message object corresponding ends structure

tx_qedge

    edge corresponding to transmitted message

tx_slot

    slot holding transmitted message, slot is taken from `canque_test_outslot` call
    and is freed by `canque_free_outslot` or rescheduled `canque_again_outslot`

tx_retry_cnt

    transmission attempt counter

tx_timeout

    can be used by chip driver to check for the transmission timeout

rx_msg

    temporary   storage   to   hold   received   messages   before   calling   to
    `canque_filter_msg2edges`

hostchip

    pointer to the &canchip_t structure this object belongs to

rx_preconfig_id

    place to store RX message identifier for some chip types that reuse same object for
    TX

obj_used

    counter of users (associated file structures for Linux userspace clients) of this object

obj_users

    list of user structures of type &canuser_t.

struct canuser_t

Name

`struct canuser_t` — structure holding CAN user/client state

Synopsis

```
struct canuser_t {
  unsigned long flags;
  struct list_head peers;
  struct canque_ends_t * qends;
  struct msgobj_t * msgobj;
  struct canque_edge_t * rx_edge0;
  union {} userinfo;
  int magic;
};
```

Members

flags

  used to distinguish Linux/RT-Linux type

peers

  for connection into list of object users

qends

  pointer to the ends structure corresponding for this user

msgobj

  communication object the user is connected to

rx_edge0

  default receive queue for filter IOCTL

userinfo

  stores user context specific information. The field `fileinfo`.file holds pointer to
  open device file state structure for the Linux user-space client applications

magic

  magic number to check consistency when pointer is retrieved from file private field

struct hwspecops_t

Name

struct hwspecops_t — hardware/board specific operations

Synopsis

```
struct hwspecops_t {
  int (* request_io) (struct candevice_t *candev);
  int (* release_io) (struct candevice_t *candev);
  int (* reset) (struct candevice_t *candev);
  int (* init_hw_data) (struct candevice_t *candev);
  int (* init_chip_data) (struct candevice_t *candev, int chipnr);
  int (* init_obj_data) (struct canchip_t *chip, int objnr);
  int (* program_irq) (struct candevice_t *candev);
  void (* write_register) (unsigned data,unsigned long address);
  unsigned (* read_register) (unsigned long address);
};
```

Members

request_io

  reserve io or memory range for can board

release_io

  free reserved io memory range

reset

  hardware reset routine

init_hw_data

    called to initialize &candevice_t structure, mainly *res_add*, *nr_all_chips*, *nr_82527_chips*, *nr_sja1000_chips* and *flags* fields

init_chip_data

    called initialize each &canchip_t structure, mainly *chip_type*, *chip_base_addr*, *clock* and chip specific registers. It is responsible to setup &canchip_t->*chipspecops* functions for non-standard chip types (type other than "i82527", "sja1000" or "sja1000p")

init_obj_data

    called initialize each &msgobj_t structure, mainly *obj_base_addr* field.

program_irq

    program interrupt generation hardware of the board if flag `PROGRAMMABLE_IRQ` is present for specified device/board

write_register

    low level write register routine

read_register

    low level read register routine

struct chipspecops_t

Name

`struct chipspecops_t` — can controller chip specific operations

Synopsis

```
struct chipspecops_t {
  int (* chip_config) (struct canchip_t *chip);
  int (* baud_rate) (struct canchip_t *chip, int rate, int clock, int sjw,int sampl_pt, int flags);
  int (* standard_mask) (struct canchip_t *chip, unsigned short code,unsigned short mask);
  int (* extended_mask) (struct canchip_t *chip, unsigned long code,unsigned long mask);
  int (* message15_mask) (struct canchip_t *chip, unsigned long code,unsigned long mask);
  int (* clear_objects) (struct canchip_t *chip);
  int (* config_irqs) (struct canchip_t *chip, short irqs);
  int (* pre_read_config) (struct canchip_t *chip, struct msgobj_t *obj);
  int (* pre_write_config) (struct canchip_t *chip, struct msgobj_t *obj,struct canmsg_t *msg);
  int (* send_msg) (struct canchip_t *chip, struct msgobj_t *obj,struct canmsg_t *msg);
  int (* remote_request) (struct canchip_t *chip, struct msgobj_t *obj);
  int (* check_tx_stat) (struct canchip_t *chip);
  int (* wakeup_tx) (struct canchip_t *chip, struct msgobj_t *obj);
  int (* filtch_rq) (struct canchip_t *chip, struct msgobj_t *obj);
  int (* enable_configuration) (struct canchip_t *chip);
  int (* disable_configuration) (struct canchip_t *chip);
  int (* set_btregs) (struct canchip_t *chip, unsigned short btr0,unsigned short btr1);
  int (* attach_to_chip) (struct canchip_t *chip);
  int (* release_chip) (struct canchip_t *chip);
  int (* start_chip) (struct canchip_t *chip);
  int (* stop_chip) (struct canchip_t *chip);
  int (* irq_handler) (int irq, struct canchip_t *chip);
  int (* irq_accept) (int irq, struct canchip_t *chip);
};
```

Members

chip_config

    CAN chip configuration

baud_rate

    set communication parameters

standard_mask

    setup of mask for message filtering

extended_mask

    setup of extended mask for message filtering

message15_mask

    set mask of i82527 message object 15

clear_objects

    clears state of all message object residing in chip

config_irqs

    tunes chip hardware interrupt delivery

pre_read_config

    prepares message object for message reception

pre_write_config

    prepares message object for message transmission

send_msg

    initiate message transmission

remote_request

    configures message object and asks for RTR message

check_tx_stat

    checks state of transmission engine

wakeup_tx

    wakeup TX processing

filtch_rq

    optional routine for propagation of outgoing edges filters to HW

enable_configuration

    enable chip configuration mode

disable_configuration

    disable chip configuration mode

set_btregs

    configures bitrate registers

attach_to_chip

    attaches to the chip, setups registers and possibly state informations

release_chip

    called before chip structure removal if `CHIP_ATTACHED` is set

start_chip

    starts chip message processing

stop_chip

    stops chip message processing

irq_handler

    interrupt service routine

irq_accept

    optional fast irq accept routine responsible for blocking further interrupts

1.6.2. Board Support Functions

The functions, which should be implemented for each supported board, are described in the next section. The functions are prefixed by boardname. The prefix *template* has been selected for next description.

template_request_io

Name

`template_request_io` — reserve io or memory range for can board

Synopsis

`int **template_request_io** (struct candevice_t * *candev*);`

Arguments

candev

> pointer to candevice/board which asks for io. Field *io_addr* of *candev* is used in most cases to define start of the range

Description

The function `template_request_io` is used to reserve the io-memory. If your hardware uses a dedicated memory range as hardware control registers you will have to add the code to reserve this memory as well. `IO_RANGE` is the io-memory range that gets reserved, please adjust according your hardware. Example: #define IO_RANGE 0x100 for i82527 chips or #define IO_RANGE 0x20 for sja1000 chips in basic CAN mode.

Return Value

The function returns zero on success or `-ENODEV` on failure

File

src/template.c

template_release_io

Name

`template_release_io` — free reserved io memory range

Synopsis

`int **template_release_io** (struct candevice_t * *candev*);`

Arguments

candev

> pointer to candevice/board which releases io

Description

The function `template_release_io` is used to free reserved io-memory. In case you have reserved more io memory, don't forget to free it here. IO_RANGE is the io-memory range that gets released, please adjust according your hardware. Example: #define IO_RANGE 0x100 for i82527 chips or #define IO_RANGE 0x20 for sja1000 chips in basic CAN mode.

Return Value

The function always returns zero

File

src/template.c

template_reset

Name

`template_reset` — hardware reset routine

Synopsis

```
int template_reset (struct candevice_t * candev);
```

Arguments

candev

    Pointer to candevice/board structure

Description

The function `template_reset` is used to give a hardware reset. This is rather hardware specific so I haven't included example code. Don't forget to check the reset status of the chip before returning.

Return Value

The function returns zero on success or `-ENODEV` on failure

File

src/template.c

template_init_hw_data

Name

`template_init_hw_data` — Initialize hardware cards

Synopsis

```
int template_init_hw_data (struct candevice_t * candev);
```

Arguments

candev

    Pointer to candevice/board structure

Description

The function `template_init_hw_data` is used to initialize the hardware structure containing information about the installed CAN-board. `RESET_ADDR` represents the io-address of the hardware reset register. `NR_82527` represents the number of Intel 82527 chips on the board. `NR_SJA1000` represents the number of Philips sja1000 chips on the board. The flags entry can currently only be `CANDEV_PROGRAMMABLE_IRQ` to indicate that the hardware uses programmable interrupts.

Return Value

The function always returns zero

File

src/template.c

template_init_chip_data

Name

`template_init_chip_data` — Initialize chips

Synopsis

```
int template_init_chip_data (struct candevice_t * candev, int chipnr);
```

Arguments

candev

>   Pointer to candevice/board structure

chipnr

>   Number of the CAN chip on the hardware card

Description

The function `template_init_chip_data` is used to initialize the hardware structure containing information about the CAN chips. `CHIP_TYPE` represents the type of CAN chip. `CHIP_TYPE` can be "i82527" or "sja1000". The *chip_base_addr* entry represents the start of the 'official' memory map of the installed chip. It's likely that this is the same as the *io_addr* argument supplied at module loading time. The *clock* entry holds the chip clock value in Hz. The entry *sja_cdr_reg* holds hardware specific options for the Clock Divider register. Options defined in the `sja1000`.h file: `sjaCDR_CLKOUT_MASK`, `sjaCDR_CLK_OFF`, `sjaCDR_RXINPEN`, `sjaCDR_CBP`, `sjaCDR_PELICAN` The entry *sja_ocr_reg* holds hardware specific options for the Output Control register. Options defined in the `sja1000`.h file: `sjaOCR_MODE_BIPHASE`, `sjaOCR_MODE_TEST`, `sjaOCR_MODE_NORMAL`, `sjaOCR_MODE_CLOCK`, `sjaOCR_TX0_LH`, `sjaOCR_TX1_ZZ`. The entry *int_clk_reg* holds hardware specific options for the Clock Out register. Options defined in the `i82527`.h file: `iCLK_CD0`, `iCLK_CD1`, `iCLK_CD2`, `iCLK_CD3`, `iCLK_SL0`, `iCLK_SL1`. The entry *int_bus_reg* holds hardware specific options for the Bus Configuration register. Options defined in the `i82527`.h file: `iBUS_DR0`, `iBUS_DR1`, `iBUS_DT1`, `iBUS_POL`, `iBUS_CBY`. The entry *int_cpu_reg* holds hardware specific options for the cpu interface register. Options defined in the `i82527`.h file: `iCPU_CEN`, `iCPU_MUX`, `iCPU_SLP`, `iCPU_PWD`, `iCPU_DMC`, `iCPU_DSC`, `iCPU_RST`.

Return Value

The function always returns zero

File

src/template.c

template_init_obj_data

Name

`template_init_obj_data` — Initialize message buffers

Synopsis

```
int template_init_obj_data (struct canchip_t * chip, int objnr);
```

Arguments

chip

>   Pointer to chip specific structure

objnr

>   Number of the message buffer

Description

The function `template_init_obj_data` is used to initialize the hardware structure containing information about the different message objects on the CAN chip. In case of the sja1000 there's only one message object but on the i82527 chip there are 15. The code below is for a i82527 chip and initializes the object base addresses The entry *obj_base_addr* represents the first memory address of the message object. In case of the sja1000 *obj_base_addr* is taken the same as the chips base address. Unless the hardware uses a segmented memory map, flags can be set zero.

Return Value

The function always returns zero

File

src/template.c

template_program_irq

Name

`template_program_irq` — program interrupts

Synopsis

```
int template_program_irq (struct candevice_t * candev);
```

Arguments

candev

> Pointer to candevice/board structure

Description

The function `template_program_irq` is used for hardware that uses programmable interrupts. If your hardware doesn't use programmable interrupts you should not set the *candevices_t*->flags entry to `CANDEV_PROGRAMMABLE_IRQ` and leave this function unedited. Again this function is hardware specific so there's no example code.

Return value

The function returns zero on success or `-ENODEV` on failure

File

src/template.c

template_write_register

Name

`template_write_register` — Low level write register routine

Synopsis

```
void template_write_register (unsigned data, unsigned long address);
```

Arguments

data

> data to be written

address

> memory address to write to

Description

The function `template_write_register` is used to write to hardware registers on the CAN chip. You should only have to edit this function if your hardware uses some specific write process.

Return Value

The function does not return a value

File

src/template.c

template_read_register

Name

`template_read_register` — Low level read register routine

Synopsis

```
unsigned template_read_register (unsigned long address);
```

Arguments

address

 memory address to read from

Description

The function `template_read_register` is used to read from hardware registers on the CAN chip. You should only have to edit this function if your hardware uses some specific read process.

Return Value

The function returns the value stored in *address*

File

src/template.c

1.6.3. Chip Support Functions

The controller chip specific functions are described in the next section. The functions should be prefixed by chip type. Because documentation of chip functions has been retrieved from the actual SJA1000 PeliCAN support, the function prefix is *sja1000p*.

sja1000p_enable_configuration

Name

sja1000p_enable_configuration — enable chip configuration mode

Synopsis

```
int sja1000p_enable_configuration (struct canchip_t * chip);
```

Arguments

chip

 pointer to chip state structure

sja1000p_disable_configuration

Name

sja1000p_disable_configuration — disable chip configuration mode

Synopsis

```
int sja1000p_disable_configuration (struct canchip_t * chip);
```

Arguments

chip

> pointer to chip state structure

sja1000p_chip_config

Name

`sja1000p_chip_config` — can chip configuration

Synopsis

```
int sja1000p_chip_config (struct canchip_t * chip);
```

Arguments

chip

> pointer to chip state structure

Description

This function configures chip and prepares it for message transmission and reception. The function resets chip, resets mask for acceptance of all messages by call to `sja1000p_extended_mask` function and then computes and sets baudrate with use of function `sja1000p_baud_rate`.

Return Value

negative value reports error.

File

src/sja1000p.c

sja1000p_extended_mask

Name

`sja1000p_extended_mask` — setup of extended mask for message filtering

Synopsis

```
int sja1000p_extended_mask (struct canchip_t * chip, unsigned long code, unsigned long mask);
```

Arguments

chip

> pointer to chip state structure

code

> can message acceptance code

mask

> can message acceptance mask

Return Value

negative value reports error.

File

src/sja1000p.c

sja1000p_baud_rate

Name

`sja1000p_baud_rate` — set communication parameters.

Synopsis

```
int sja1000p_baud_rate (struct canchip_t * chip, int rate, int clock, int sjw, int sampl_pt,
int flags);
```

Arguments

chip
    pointer to chip state structure
rate
    baud rate in Hz
clock
    frequency of sja1000 clock in Hz (ISA osc is 14318000)
sjw
    synchronization jump width (0-3) prescaled clock cycles
sampl_pt
    sample point in % (0-100) sets (TSEG1+1)/(TSEG1+TSEG2+2) ratio
flags
    fields `BTR1_SAM`, `OCMODE`, `OCPOL`, `OCTP`, `OCTN`, `CLK_OFF`, `CBP`

Return Value

negative value reports error.

File

src/sja1000p.c

sja1000p_read

Name

`sja1000p_read` — reads and distributes one or more received messages

Synopsis

```
void sja1000p_read (struct canchip_t * chip, struct msgobj_t * obj);
```

Arguments

chip

> pointer to chip state structure

obj

> pinter to CAN message queue information

File

src/sja1000p.c

sja1000p_pre_read_config

Name

`sja1000p_pre_read_config` — prepares message object for message reception

Synopsis

```
int sja1000p_pre_read_config (struct canchip_t * chip, struct msgobj_t * obj);
```

Arguments

chip

> pointer to chip state structure

obj

> pointer to message object state structure

Return Value

negative value reports error. Positive value indicates immediate reception of message.

File

src/sja1000p.c

sja1000p_pre_write_config

Name

`sja1000p_pre_write_config` — prepares message object for message transmission

Synopsis

```
int sja1000p_pre_write_config (struct canchip_t * chip, struct msgobj_t * obj, struct canmsg_t * msg);
```

Arguments

chip

> pointer to chip state structure

obj

> pointer to message object state structure

msg

   pointer to CAN message

Description

This function prepares selected message object for future initiation of message transmission by `sja1000p_send_msg` function. The CAN message data and message ID are transfered from *msg* slot into chip buffer in this function.

Return Value

negative value reports error.

File

src/sja1000p.c

sja1000p_send_msg

Name

`sja1000p_send_msg` — initiate message transmission

Synopsis

```
int sja1000p_send_msg (struct canchip_t * chip, struct msgobj_t * obj, struct canmsg_t * msg);
```

Arguments

chip

   pointer to chip state structure

obj

   pointer to message object state structure

msg

   pointer to CAN message

Description

This function is called after `sja1000p_pre_write_config` function, which prepares data in chip buffer.

Return Value

negative value reports error.

File

src/sja1000p.c

sja1000p_check_tx_stat

Name

`sja1000p_check_tx_stat` — checks state of transmission engine

Synopsis

int **sja1000p_check_tx_stat** (struct canchip_t * *chip*);

Arguments

chip

>   pointer to chip state structure

Return Value

negative value reports error. Positive return value indicates transmission under way status. Zero value indicates finishing of all issued transmission requests.

File

src/sja1000p.c

sja1000p_set_btregs

Name

sja1000p_set_btregs — configures bitrate registers

Synopsis

int **sja1000p_set_btregs** (struct canchip_t * *chip*, unsigned short *btr0*, unsigned short *btr1*);

Arguments

chip

>   pointer to chip state structure

btr0

>   bitrate register 0

btr1

>   bitrate register 1

Return Value

negative value reports error.

File

src/sja1000p.c

sja1000p_start_chip

Name

sja1000p_start_chip — starts chip message processing

Synopsis

int **sja1000p_start_chip** (struct canchip_t * *chip*);

Arguments

chip

   pointer to chip state structure

Return Value

negative value reports error.

File

src/sja1000p.c

sja1000p_stop_chip

Name

`sja1000p_stop_chip` — stops chip message processing

Synopsis

```
int sja1000p_stop_chip (struct canchip_t * chip);
```

Arguments

chip

   pointer to chip state structure

Return Value

negative value reports error.

File

src/sja1000p.c

sja1000p_attach_to_chip

Name

`sja1000p_attach_to_chip` — attaches to the chip, setups registers and state

Synopsis

```
int sja1000p_attach_to_chip (struct canchip_t * chip);
```

Arguments

chip

   pointer to chip state structure

Return Value

negative value reports error.

File

src/sja1000p.c

sja1000p_release_chip

Name

sja1000p_release_chip — called before chip structure removal if CHIP_ATTACHED is set

Synopsis

int **sja1000p_release_chip** (struct canchip_t * *chip*);

Arguments

chip

pointer to chip state structure

Return Value

negative value reports error.

File

src/sja1000p.c

sja1000p_remote_request

Name

sja1000p_remote_request — configures message object and asks for RTR message

Synopsis

int **sja1000p_remote_request** (struct canchip_t * *chip*, struct msgobj_t * *obj*);

Arguments

chip

pointer to chip state structure

obj

pointer to message object structure

Return Value

negative value reports error.

File

src/sja1000p.c

sja1000p_standard_mask

Name

`sja1000p_standard_mask` — setup of mask for message filtering

Synopsis

```
int sja1000p_standard_mask (struct canchip_t * chip, unsigned short code, unsigned short mask);
```

Arguments

chip

 pointer to chip state structure

code

 can message acceptance code

mask

 can message acceptance mask

Return Value

negative value reports error.

File

src/sja1000p.c

sja1000p_clear_objects

Name

`sja1000p_clear_objects` — clears state of all message object residing in chip

Synopsis

```
int sja1000p_clear_objects (struct canchip_t * chip);
```

Arguments

chip

 pointer to chip state structure

Return Value

negative value reports error.

File

src/sja1000p.c

sja1000p_config_irqs

Name

`sja1000p_config_irqs` — tunes chip hardware interrupt delivery

Synopsis

```
int sja1000p_config_irqs (struct canchip_t * chip, short irqs);
```

Arguments

chip

>   pointer to chip state structure

irqs

>   requested chip IRQ configuration

Return Value

negative value reports error.

File

src/sja1000p.c

sja1000p_irq_write_handler

Name

`sja1000p_irq_write_handler` — part of ISR code responsible for transmit events

Synopsis

```
void sja1000p_irq_write_handler (struct canchip_t * chip, struct msgobj_t * obj);
```

Arguments

chip

>   pointer to chip state structure

obj

>   pointer to attached queue description

Description

The main purpose of this function is to read message from attached queues and transfer message contents into CAN controller chip. This subroutine is called by `sja1000p_irq_write_handler` for transmit events.

File

src/sja1000p.c

sja1000p_irq_handler

Name

`sja1000p_irq_handler` — interrupt service routine

Synopsis

```
int sja1000p_irq_handler (int irq, struct canchip_t * chip);
```

Arguments

irq

> interrupt vector number, this value is system specific

chip

> pointer to chip state structure

Description

Interrupt handler is activated when state of CAN controller chip changes, there is message to be read or there is more space for new messages or error occurs. The receive events results in reading of the message from CAN controller chip and distribution of message through attached message queues.

File

src/sja1000p.c

sja1000p_wakeup_tx

Name

`sja1000p_wakeup_tx` — wakeups TX processing

Synopsis

```
int sja1000p_wakeup_tx (struct canchip_t * chip, struct msgobj_t * obj);
```

Arguments

chip

> pointer to chip state structure

obj

> pointer to message object structure

Description

Function is responsible for initiating message transmition. It is responsible for clearing of object TX_REQUEST flag

Return Value

negative value reports error.

File

src/sja1000p.c

sja1000p_fill_chipspecops

Name

`sja1000p_fill_chipspecops` — fills chip specific operations

Synopsis

```
int sja1000p_fill_chipspecops (struct canchip_t * chip);
```

Arguments

chip

> pointer to chip representation structure

Description

The function fills chip specific operations for sja1000 (PeliCAN) chip.

Return Value

returns negative number in the case of fail

1.6.4. CAN Queues Common Structures and Functions

This part of the driver implements basic CAN queues infrastructure. It is written as much generic as possible and then specialization for each category of CAN queues clients is implemented in separate subsystem. The only synchronization mechanism required from target system are spin-lock synchronization and atomic bit manipulation. Locked sections are narrowed to the short operations. Even can message 8 bytes movement is excluded from the locked sections of the code.

struct canque_slot_t

Name

struct `canque_slot_t` — one CAN message slot in the CAN FIFO queue

Synopsis

```
struct canque_slot_t {
  struct canque_slot_t * next;
  unsigned long slot_flags;
  struct canmsg_t msg;
};
```

Members

next

> pointer to the next/younger slot

slot_flags

> space for flags and optional command describing action associated with slot data

msg

>   space for one CAN message

Description

This structure is used to store CAN messages in the CAN FIFO queue.

struct canque_fifo_t

Name

struct `canque_fifo_t` — CAN FIFO queue representation

Synopsis

```
struct canque_fifo_t {
  unsigned long fifo_flags;
  unsigned long error_code;
  struct canque_slot_t * head;
  struct canque_slot_t ** tail;
  struct canque_slot_t * flist;
  struct canque_slot_t * entry;
  can_spinlock_t fifo_lock;
  int slotsnr;
};
```

Members

fifo_flags

>   this field holds global flags describing state of the FIFO. `CAN_FIFOF_ERROR` is set when some error condition occurs. `CAN_FIFOF_ERR2BLOCK` defines, that error should lead to the FIFO block state. `CAN_FIFOF_BLOCK` state blocks insertion of the next messages. `CAN_FIFOF_OVERRUN` attempt to acquire new slot, when FIFO is full. `CAN_FIFOF_FULL` indicates FIFO full state. `CAN_FIFOF_EMPTY` indicates no allocated slot in the FIFO. `CAN_FIFOF_DEAD` condition indication. Used when FIFO is beeing destroyed.

error_code

>   futher description of error condition

head

>   pointer to the FIFO head, oldest slot

tail

>   pointer to the location, where pointer to newly inserted slot should be added

flist

>   pointer to list of the free slots associated with queue

entry

>   pointer to the memory allocated for the list slots.

fifo_lock

>   the lock to ensure atomicity of slot manipulation operations.

slotsnr

>   number of allocated slots

Description

This structure represents CAN FIFO queue. It is implemented as a single linked list of slots prepared for processing. The empty slots are stored in single linked list (`flist`).

canque_fifo_get_inslot

Name

canque_fifo_get_inslot — allocate slot for the input of one CAN message

Synopsis

int **canque_fifo_get_inslot** (struct canque_fifo_t * *fifo*, struct canque_slot_t ** *slotp*, int *cmd*);

Arguments

fifo
> pointer to the FIFO structure

slotp
> pointer to location to store pointer to the allocated slot.

cmd
> optional command associated with allocated slot.

Return Value

The function returns negative value if there is no free slot in the FIFO queue.

canque_fifo_put_inslot

Name

canque_fifo_put_inslot — releases slot to further processing

Synopsis

int **canque_fifo_put_inslot** (struct canque_fifo_t * *fifo*, struct canque_slot_t * *slot*);

Arguments

fifo
> pointer to the FIFO structure

slot
> pointer to the slot previously acquired by canque_fifo_get_inslot.

Return Value

The nonzero return value indicates, that the queue was empty before call to the function. The caller should wake-up output side of the queue.

canque_fifo_abort_inslot

Name

canque_fifo_abort_inslot — release and abort slot

Synopsis

```
int canque_fifo_abort_inslot (struct canque_fifo_t * fifo, struct canque_slot_t * slot);
```

Arguments

fifo

   pointer to the FIFO structure

slot

   pointer to the slot previously acquired by `canque_fifo_get_inslot`.

Return Value

The nonzero value indicates, that fifo was full

canque_fifo_test_outslot

Name

`canque_fifo_test_outslot` — test and get ready slot from the FIFO

Synopsis

```
int canque_fifo_test_outslot (struct canque_fifo_t * fifo, struct canque_slot_t ** slotp);
```

Arguments

fifo

   pointer to the FIFO structure

slotp

   pointer to location to store pointer to the oldest slot from the FIFO.

Return Value

The negative value indicates, that queue is empty. The positive or zero value represents command stored into slot by the call to the function `canque_fifo_get_inslot`. The successfully acquired FIFO output slot has to be released by the call `canque_fifo_free_outslot` or `canque_fifo_again_outslot`.

canque_fifo_free_outslot

Name

`canque_fifo_free_outslot` — free processed FIFO slot

Synopsis

```
int canque_fifo_free_outslot (struct canque_fifo_t * fifo, struct canque_slot_t * slot);
```

Arguments

fifo

   pointer to the FIFO structure

slot

    pointer to the slot previously acquired by `canque_fifo_test_outslot`.

Return Value

The returned value informs about FIFO state change. The mask `CAN_FIFOF_FULL` indicates, that the FIFO was full before the function call. The mask `CAN_FIFOF_EMPTY` informs, that last ready slot has been processed.

canque_fifo_again_outslot

Name

`canque_fifo_again_outslot` — interrupt and postpone processing of the slot

Synopsis

```
int canque_fifo_again_outslot (struct canque_fifo_t * fifo, struct canque_slot_t * slot);
```

Arguments

fifo

    pointer to the FIFO structure

slot

    pointer to the slot previously acquired by `canque_fifo_test_outslot`.

Return Value

The function cannot fail..

struct canque_edge_t

Name

`struct canque_edge_t` — CAN message delivery subsystem graph edge

Synopsis

```
struct canque_edge_t {
  struct canque_fifo_t fifo;
  unsigned long filtid;
  unsigned long filtmask;
  struct list_head inpeers;
  struct list_head outpeers;
  struct list_head activepeers;
  struct canque_ends_t * inends;
  struct canque_ends_t * outends;
  atomic_t edge_used;
  int edge_prio;
  int edge_num;
#ifdef CAN_WITH_RTL
  struct list_head pending_peers;
  unsigned long pending_inops;
  unsigned long pending_outops;
#endif
};
```

Members

fifo

    place where primitive `struct` canque_fifo_t FIFO is located.

filtid

    the possible CAN message identifiers filter.

filtmask

    the filter mask, the comparison considers only `filtid` bits corresponding to set bits in the `filtmask` field.

inpeers

    the lists of all peers FIFOs connected by their input side (`inends`) to the same terminal (`struct` canque_ends_t).

outpeers

    the lists of all peers FIFOs connected by their output side (`outends`) to the same terminal (`struct` canque_ends_t).

activepeers

    the lists of peers FIFOs connected by their output side (`outends`) to the same terminal (`struct` canque_ends_t) with same priority and active state.

inends

    the pointer to the FIFO input side terminal (`struct` canque_ends_t).

outends

    the pointer to the FIFO output side terminal (`struct` canque_ends_t).

edge_used

    the atomic usage counter, mainly used for safe destruction of the edge.

edge_prio

    the assigned queue priority from the range 0 to `CANQUEUE_PRIO_NR-1`

edge_num

    edge sequential number intended for debugging purposes only

pending_peers

    edges with pending delayed events (RTL->Linux calls)

pending_inops

    bitmask of pending operations

pending_outops

    bitmask of pending operations

Description

This structure represents one direction connection from messages source (`inends`) to message consumer (`outends`) fifo ends hub. The edge contains &struct canque_fifo_t for message fifo implementation.

struct canque_ends_t

Name

`struct canque_ends_t` — CAN message delivery subsystem graph vertex (FIFO ends)

Synopsis

```
struct canque_ends_t {
  unsigned long ends_flags;
  struct list_head active[CANQUEUE_PRIO_NR];
  struct list_head idle;
  struct list_head inlist;
  struct list_head outlist;
  can_spinlock_t ends_lock;
  void (* notify) (struct canque_ends_t *qends, struct canque_edge_t *qedge, int what);
  void * context;
  union {} endinfo;
  struct list_head dead_peers;
};
```

Members

ends_flags

 this field holds flags describing state of the ENDS structure.

active[CANQUEUE_PRIO_NR]

 the array of the lists of active edges directed to the ends structure with ready messages. The array is indexed by the edges priorities.

idle

 the list of the edges directed to the ends structure with empty FIFOs.

inlist

 the list of outgoing edges input sides.

outlist

 the list of all incoming edges output sides. Each of there edges is listed on one of `active` or `idle` lists.

ends_lock

 the lock synchronizing operations between threads accessing same ends structure.

notify

 pointer to notify procedure. The next state changes are notified. `CANQUEUE_NOTIFY_EMPTY` (out->in call) - all slots are processed by FIFO out side. `CANQUEUE_NOTIFY_SPACE` (out->in call) - full state negated => there is space for new message. `CANQUEUE_NOTIFY_PROC` (in->out call) - empty state negated => out side is requested to process slots. `CANQUEUE_NOTIFY_NOUSR` (both) - notify, that the last user has released the edge usage called with some lock to prevent edge disappear. `CANQUEUE_NOTIFY_DEAD` (both) - edge is in progress of deletion. `CANQUEUE_NOTIFY_ATACH` (both) - new edge has been attached to end. `CANQUEUE_NOTIFY_FILTCH` (out->in call) - edge filter rules changed `CANQUEUE_NOTIFY_ERROR` (out->in call) - error in messages processing.

context

 space to store ends user specific information

endinfo

 space to store some other ends usage specific informations mainly for waking-up by the notify calls.

dead_peers

 used to chain ends wanting for postponed destruction

Description

Structure represents place to connect edges to for CAN communication entity. The zero, one or more incoming and outgoing edges can be connected to this structure.

canque_notify_inends

Name

canque_notify_inends — request to send notification to the input ends

Synopsis

void **canque_notify_inends** (struct canque_edge_t * *qedge*, int *what*);

Arguments

qedge

   pointer to the edge structure

what

   notification type

canque_notify_outends

Name

canque_notify_outends — request to send notification to the output ends

Synopsis

void **canque_notify_outends** (struct canque_edge_t * *qedge*, int *what*);

Arguments

qedge

   pointer to the edge structure

what

   notification type

canque_notify_bothends

Name

canque_notify_bothends — request to send notification to the both ends

Synopsis

void **canque_notify_bothends** (struct canque_edge_t * *qedge*, int *what*);

Arguments

qedge

   pointer to the edge structure

what

   notification type

canque_activate_edge

Name

canque_activate_edge — mark output end of the edge as active

Synopsis

void **canque_activate_edge** (struct canque_ends_t * *inends*, struct canque_edge_t * *qedge*);

Arguments

inends

　　input side of the edge

qedge

　　pointer to the edge structure

Description

Function call moves output side of the edge from idle onto active edges list. This function has to be called with edge reference count held. that is same as for most of other edge functions.

canque_filtid2internal

Name

canque_filtid2internal — converts message ID and filter flags into internal format

Synopsis

unsigned int **canque_filtid2internal** (unsigned long *id*, int *filtflags*);

Arguments

id

　　CAN message 11 or 29 bit identifier

filtflags

　　CAN message flags

Description

This function maps message ID and MSG_RTR, MSG_EXT and MSG_LOCAL into one 32 bit number

canque_edge_incref

Name

canque_edge_incref — increments edge reference count

Synopsis

```
void canque_edge_incref (struct canque_edge_t * edge);
```

Arguments

edge

> pointer to the edge structure

canque_edge_decref

Name

canque_edge_decref — decrements edge reference count

Synopsis

```
void canque_edge_decref (struct canque_edge_t * edge);
```

Arguments

edge

> pointer to the edge structure

Description

This function has to be called without lock held for both ends of edge. If reference count drops to 0, function `canque_edge_do_dead` is called.

canque_fifo_flush_slots

Name

canque_fifo_flush_slots — free all ready slots from the FIFO

Synopsis

```
int canque_fifo_flush_slots (struct canque_fifo_t * fifo);
```

Arguments

fifo

> pointer to the FIFO structure

Description

The caller should be prepared to handle situations, when some slots are held by input or output side slots processing. These slots cannot be flushed or their processing interrupted.

Return Value

The nonzero value indicates, that queue has not been empty before the function call.

canque_fifo_init_slots

Name

canque_fifo_init_slots — initializes slot chain of one CAN FIFO

Synopsis

int **canque_fifo_init_slots** (struct canque_fifo_t * *fifo*);

Arguments

fifo

   pointer to the FIFO structure

Return Value

The negative value indicates, that there is no memory to allocate space for the requested number of the slots.

canque_get_inslot

Name

canque_get_inslot — finds one outgoing edge and allocates slot from it

Synopsis

int **canque_get_inslot** (struct canque_ends_t * *qends*, struct canque_edge_t ** *qedgep*, struct canque_slot_t ** *slotp*, int *cmd*);

Arguments

qends

   ends structure belonging to calling communication object

qedgep

   place to store pointer to found edge

slotp

   place to store pointer to allocated slot

cmd

   command type for slot

Description

Function looks for the first non-blocked outgoing edge in *qends* structure and tries to allocate slot from it.

Return Value

If there is no usable edge or there is no free slot in edge negative value is returned.

canque_get_inslot4id

Name

`canque_get_inslot4id` — finds best outgoing edge and slot for given ID

Synopsis

```
int canque_get_inslot4id (struct canque_ends_t * qends, struct canque_edge_t ** qedgep,
struct canque_slot_t ** slotp, int cmd, unsigned long id, int prio);
```

Arguments

qends

  ends structure belonging to calling communication object

qedgep

  place to store pointer to found edge

slotp

  place to store pointer to allocated slot

cmd

  command type for slot

id

  communication ID of message to send into edge

prio

  optional priority of message

Description

Function looks for the non-blocked outgoing edge accepting messages with given ID. If edge is found, slot is allocated from that edge. The edges with non-zero mask are preferred over edges open to all messages. If more edges with mask accepts given message ID, the edge with highest priority below or equal to required priority is selected.

Return Value

If there is no usable edge or there is no free slot in edge negative value is returned.

canque_put_inslot

Name

`canque_put_inslot` — schedules filled slot for processing

Synopsis

```
int canque_put_inslot (struct canque_ends_t * qends, struct canque_edge_t * qedge, struct
canque_slot_t * slot);
```

Arguments

qends

  ends structure belonging to calling communication object

qedge

  edge slot belong to

slot

> pointer to the prepared slot

Description

Puts slot previously acquired by `canque_get_inslot` or `canque_get_inslot4id` function call into FIFO queue and activates edge processing if needed.

Return Value

Positive value informs, that activation of output end has been necessary

canque_abort_inslot

Name

`canque_abort_inslot` — aborts preparation of the message in the slot

Synopsis

```
int canque_abort_inslot (struct canque_ends_t * qends, struct canque_edge_t * qedge, struct
canque_slot_t * slot);
```

Arguments

qends

> ends structure belonging to calling communication object

qedge

> edge slot belong to

slot

> pointer to the previously allocated slot

Description

Frees slot previously acquired by `canque_get_inslot` or `canque_get_inslot4id` function call. Used when message copying into slot fails.

Return Value

Positive value informs, that queue full state has been negated.

canque_filter_msg2edges

Name

`canque_filter_msg2edges` — sends message into all edges which accept its ID

Synopsis

```
int canque_filter_msg2edges (struct canque_ends_t * qends, struct canmsg_t * msg);
```

Arguments

qends

    ends structure belonging to calling communication object

msg

    pointer to CAN message

Description

Sends message to all outgoing edges connected to the given ends, which accepts message communication ID.

Return Value

Returns number of edges message has been send to

canque_test_outslot

Name

`canque_test_outslot` — test and retrieve ready slot for given ends

Synopsis

```
int canque_test_outslot (struct canque_ends_t * qends, struct canque_edge_t ** qedgep, struct canque_slot_t ** slotp);
```

Arguments

qends

    ends structure belonging to calling communication object

qedgep

    place to store pointer to found edge

slotp

    place to store pointer to received slot

Description

Function takes highest priority active incoming edge and retrieves oldest ready slot from it.

Return Value

Negative value informs, that there is no ready output slot for given ends. Positive value is equal to the command slot has been allocated by the input side.

canque_free_outslot

Name

`canque_free_outslot` — frees processed output slot

Synopsis

```
int canque_free_outslot (struct canque_ends_t * qends, struct canque_edge_t * qedge, struct
canque_slot_t * slot);
```

Arguments

qends

    ends structure belonging to calling communication object

qedge

    edge slot belong to

slot

    pointer to the processed slot

Description

Function releases processed slot previously acquired by `canque_test_outslot` function call.

Return Value

Return value informs if input side has been notified to know about change of edge state

canque_again_outslot

Name

canque_again_outslot — reschedule output slot to process it again later

Synopsis

```
int canque_again_outslot (struct canque_ends_t * qends, struct canque_edge_t * qedge, struct
canque_slot_t * slot);
```

Arguments

qends

    ends structure belonging to calling communication object

qedge

    edge slot belong to

slot

    pointer to the slot for re-processing

Description

Function reschedules slot previously acquired by `canque_test_outslot` function call for second time processing.

Return Value

Function cannot fail.

canque_set_filt

Name

canque_set_filt — sets filter for specified edge

Synopsis

```
int canque_set_filt (struct canque_edge_t * qedge, unsigned long filtid, unsigned long
filtmask, int filtflags);
```

Arguments

qedge

   pointer to the edge

filtid

   ID to set for the edge

filtmask

   mask used for ID match check

filtflags

   required filer flags

Return Value

Negative value is returned if edge is in the process of delete.

canque_flush

Name

canque_flush — fluesh all ready slots in the edge

Synopsis

```
int canque_flush (struct canque_edge_t * qedge);
```

Arguments

qedge

   pointer to the edge

Description

Tries to flush all allocated slots from the edge, but there could exist some slots associ-
ated to edge which are processed by input or output side and cannot be flushed at this
moment.

Return Value

The nonzero value indicates, that queue has not been empty before the function call.

canqueue_ends_init_gen

Name

canqueue_ends_init_gen — subsystem independent routine to initialize ends state

Synopsis

int **canqueue_ends_init_gen** (struct canque_ends_t * *qends*);

Arguments

qends
    pointer to the ends structure

Return Value

Cannot fail.

canqueue_connect_edge

Name

canqueue_connect_edge — connect edge between two communication entities

Synopsis

int **canqueue_connect_edge** (struct canque_edge_t * *qedge*, struct canque_ends_t * *inends*,
struct canque_ends_t * *outends*);

Arguments

qedge
    pointer to edge
inends
    pointer to ends the input of the edge should be connected to
outends
    pointer to ends the output of the edge should be connected to

Return Value

Negative value informs about failed operation.

canqueue_disconnect_edge

Name

canqueue_disconnect_edge — disconnect edge from communicating entities

Synopsis

int **canqueue_disconnect_edge** (struct canque_edge_t * *qedge*);

Arguments

qedge

    pointer to edge

Return Value

Negative value means, that edge is used by somebody other and cannot be disconnected. Operation has to be delayed.

canqueue_block_inlist

Name

`canqueue_block_inlist` — block slot allocation of all outgoing edges of specified ends

Synopsis

```
void canqueue_block_inlist (struct canque_ends_t * qends);
```

Arguments

qends

    pointer to ends structure

canqueue_block_outlist

Name

`canqueue_block_outlist` — block slot allocation of all incoming edges of specified ends

Synopsis

```
void canqueue_block_outlist (struct canque_ends_t * qends);
```

Arguments

qends

    pointer to ends structure

canqueue_ends_kill_inlist

Name

`canqueue_ends_kill_inlist` — sends request to die to all outgoing edges

Synopsis

```
int canqueue_ends_kill_inlist (struct canque_ends_t * qends, int send_rest);
```

Arguments

qends

pointer to ends structure

send_rest

select, whether already allocated slots should be processed by FIFO output side

Return Value

Non-zero value means, that not all edges could be immediately disconnected and that ends structure memory release has to be delayed

canqueue_ends_kill_outlist

Name

`canqueue_ends_kill_outlist` — sends request to die to all incoming edges

Synopsis

int **canqueue_ends_kill_outlist** (struct canque_ends_t * *qends*);

Arguments

qends

pointer to ends structure

Return Value

Non-zero value means, that not all edges could be immediately disconnected and that ends structure memory release has to be delayed

canqueue_ends_filt_conjuction

Name

`canqueue_ends_filt_conjuction` — computes conjunction of incoming edges filters filters

Synopsis

int **canqueue_ends_filt_conjuction** (struct canque_ends_t * *qends*, struct canfilt_t * *filt*);

Arguments

qends

pointer to ends structure

filt

pointer the filter structure filled by computed filters conjunction

Return Value

Number of incoming edges

canqueue_ends_flush_inlist

Name

canqueue_ends_flush_inlist — flushes all messages in incoming edges

Synopsis

int **canqueue_ends_flush_inlist** (struct canque_ends_t * *qends*);

Arguments

qends

   pointer to ends structure

Return Value

Negative value informs about unsuccessful result

canqueue_ends_flush_outlist

Name

canqueue_ends_flush_outlist — flushes all messages in outgoing edges

Synopsis

int **canqueue_ends_flush_outlist** (struct canque_ends_t * *qends*);

Arguments

qends

   pointer to ends structure

Return Value

Negative value informs about unsuccessful result

1.6.5. CAN Queues Kernel Specific Functions

canqueue_notify_kern

Name

canqueue_notify_kern — notification callback handler for Linux userspace clients

Synopsis

void **canqueue_notify_kern** (struct canque_ends_t * *qends*, struct canque_edge_t * *qedge*, int *what*);

Arguments

qends

pointer to the callback side ends structure

qedge

edge which invoked notification

what

notification type

Description

The notification event is handled directly by call of this function except case, when called from RT-Linux context in mixed mode Linux/RT-Linux compilation. It is not possible to directly call Linux kernel synchronization primitives in such case. The notification request is postponed and signaled by *pending_inops* flags by call `canqueue_rtl2lin_check_and_pend` function. The edge reference count is increased until until all pending notifications are processed.

canqueue_ends_init_kern

Name

`canqueue_ends_init_kern` — Linux userspace clients specific ends initialization

Synopsis

```
int canqueue_ends_init_kern (struct canque_ends_t * qends);
```

Arguments

qends

pointer to the callback side ends structure

canque_get_inslot4id_wait_kern

Name

`canque_get_inslot4id_wait_kern` — find or wait for best outgoing edge and slot for given ID

Synopsis

```
int canque_get_inslot4id_wait_kern (struct canque_ends_t * qends, struct canque_edge_t **
qedgep, struct canque_slot_t ** slotp, int cmd, unsigned long id, int prio);
```

Arguments

qends

ends structure belonging to calling communication object

qedgep

place to store pointer to found edge

slotp

    place to store pointer to allocated slot

cmd

    command type for slot

id

    communication ID of message to send into edge

prio

    optional priority of message

Description

Same as `canque_get_inslot4id`, except, that it waits for free slot in case, that queue is full. Function is specific for Linux userspace clients.

Return Value

If there is no usable edge negative value is returned.

canque_get_outslot_wait_kern

Name

`canque_get_outslot_wait_kern` — receive or wait for ready slot for given ends

Synopsis

int **canque_get_outslot_wait_kern** (struct canque_ends_t * *qends*, struct canque_edge_t ** *qedgep*, struct canque_slot_t ** *slotp*);

Arguments

qends

    ends structure belonging to calling communication object

qedgep

    place to store pointer to found edge

slotp

    place to store pointer to received slot

Description

The same as `canque_test_outslot`, except it waits in the case, that there is no ready slot for given ends. Function is specific for Linux userspace clients.

Return Value

Negative value informs, that there is no ready output slot for given ends. Positive value is equal to the command slot has been allocated by the input side.

canque_sync_wait_kern

Name

`canque_sync_wait_kern` — wait for all slots processing

Synopsis

int **canque_sync_wait_kern** (struct canque_ends_t * *qends*, struct canque_edge_t * *qedge*);

Arguments

qends

> ends structure belonging to calling communication object

qedge

> pointer to edge

Description

Functions waits for ends transition into empty state.

Return Value

Positive value indicates, that edge empty state has been reached. Negative or zero value informs about interrupted wait or other problem.

canque_fifo_init_kern

Name

canque_fifo_init_kern — initialize one CAN FIFO

Synopsis

int **canque_fifo_init_kern** (struct canque_fifo_t * *fifo*, int *slotsnr*);

Arguments

fifo

> pointer to the FIFO structure

slotsnr

> number of requested slots

Return Value

The negative value indicates, that there is no memory to allocate space for the requested number of the slots.

canque_fifo_done_kern

Name

canque_fifo_done_kern — frees slots allocated for CAN FIFO

Synopsis

int **canque_fifo_done_kern** (struct canque_fifo_t * *fifo*);

Arguments

fifo

    pointer to the FIFO structure

canque_new_edge_kern

Name

canque_new_edge_kern — allocate new edge structure in the Linux kernel context

Synopsis

struct canque_edge_t * **canque_new_edge_kern** (int *slotsnr*);

Arguments

slotsnr

    required number of slots in the newly allocated edge structure

Return Value

Returns pointer to allocated slot structure or NULL if there is not enough memory to process operation.

canqueue_ends_dispose_kern

Name

canqueue_ends_dispose_kern — finalizing of the ends structure for Linux kernel clients

Synopsis

int **canqueue_ends_dispose_kern** (struct canque_ends_t * *qends*, int *sync*);

Arguments

qends

    pointer to ends structure

sync

    flag indicating, that user wants to wait for processing of all remaining messages

Return Value

Function should be designed such way to not fail.

1.6.6. CAN Queues RT-Linux Specific Functions

canqueue_rtl2lin_check_and_pend

Name

canqueue_rtl2lin_check_and_pend — postpones edge notification if called from RT-Linux

Synopsis

int **canqueue_rtl2lin_check_and_pend** (struct canque_ends_t * *qends*, struct canque_edge_t * *qedge*, int *what*);

Arguments

qends

   notification target ends

qedge

   edge delivering notification

what

   notification type

Return Value

if called from Linux context, returns 0 and lefts notification processing on caller responsibility. If called from RT-Linux contexts, schedules postponed event delivery and returns 1

canque_get_inslot4id_wait_rtl

Name

canque_get_inslot4id_wait_rtl — find or wait for best outgoing edge and slot for given ID

Synopsis

int **canque_get_inslot4id_wait_rtl** (struct canque_ends_t * *qends*, struct canque_edge_t ** *qedgep*, struct canque_slot_t ** *slotp*, int *cmd*, unsigned long *id*, int *prio*);

Arguments

qends

   ends structure belonging to calling communication object

qedgep

   place to store pointer to found edge

slotp

   place to store pointer to allocated slot

cmd

   command type for slot

id

   communication ID of message to send into edge

prio

> optional priority of message

Description

Same as `canque_get_inslot4id`, except, that it waits for free slot in case, that queue is full. Function is specific for Linux userspace clients.

Return Value

If there is no usable edge negative value is returned.

canque_get_outslot_wait_rtl

Name

`canque_get_outslot_wait_rtl` — receive or wait for ready slot for given ends

Synopsis

```
int canque_get_outslot_wait_rtl (struct canque_ends_t * qends, struct canque_edge_t **
qedgep, struct canque_slot_t ** slotp);
```

Arguments

qends

> ends structure belonging to calling communication object

qedgep

> place to store pointer to found edge

slotp

> place to store pointer to received slot

Description

The same as `canque_test_outslot`, except it waits in the case, that there is no ready slot for given ends. Function is specific for Linux userspace clients.

Return Value

Negative value informs, that there is no ready output slot for given ends. Positive value is equal to the command slot has been allocated by the input side.

canque_sync_wait_rtl

Name

`canque_sync_wait_rtl` — wait for all slots processing

Synopsis

```
int canque_sync_wait_rtl (struct canque_ends_t * qends, struct canque_edge_t * qedge);
```

Arguments

qends

    ends structure belonging to calling communication object

qedge

    pointer to edge

Description

Functions waits for ends transition into empty state.

Return Value

Positive value indicates, that edge empty state has been reached. Negative or zero value informs about interrupted wait or other problem.

canque_fifo_init_rtl

Name

canque_fifo_init_rtl — initialize one CAN FIFO

Synopsis

```
int canque_fifo_init_rtl (struct canque_fifo_t * fifo, int slotsnr);
```

Arguments

fifo

    pointer to the FIFO structure

slotsnr

    number of requested slots

Return Value

The negative value indicates, that there is no memory to allocate space for the requested number of the slots.

canque_fifo_done_rtl

Name

canque_fifo_done_rtl — frees slots allocated for CAN FIFO

Synopsis

```
int canque_fifo_done_rtl (struct canque_fifo_t * fifo);
```

Arguments

fifo

    pointer to the FIFO structure

canque_new_edge_rtl

Name

`canque_new_edge_rtl` — allocate new edge structure in the RT-Linux context

Synopsis

`struct canque_edge_t * ` **`canque_new_edge_rtl`** ` (int ` *`slotsnr`* `);`

Arguments

`slotsnr`

    required number of slots in the newly allocated edge structure

Return Value

Returns pointer to allocated slot structure or `NULL` if there is not enough memory to process operation.

canqueue_notify_rtl

Name

`canqueue_notify_rtl` — notification callback handler for Linux userspace clients

Synopsis

`void ` **`canqueue_notify_rtl`** ` (struct canque_ends_t * ` *`qends`* `, struct canque_edge_t * ` *`qedge`* `, int ` *`what`* `);`

Arguments

`qends`

    pointer to the callback side ends structure

`qedge`

    edge which invoked notification

`what`

    notification type

canqueue_ends_init_rtl

Name

`canqueue_ends_init_rtl` — RT-Linux clients specific ends initialization

Synopsis

`int ` **`canqueue_ends_init_rtl`** ` (struct canque_ends_t * ` *`qends`* `);`

Arguments

qends

> pointer to the callback side ends structure

canqueue_ends_dispose_rtl

Name

canqueue_ends_dispose_rtl — finalizing of the ends structure for Linux kernel clients

Synopsis

```
int canqueue_ends_dispose_rtl (struct canque_ends_t * qends, int sync);
```

Arguments

qends

> pointer to ends structure

sync

> flag indicating, that user wants to wait for processing of all remaining messages

Return Value

Function should be designed such way to not fail.

canqueue_rtl_initialize

Name

canqueue_rtl_initialize — initialization of global RT-Linux specific features

Synopsis

```
void canqueue_rtl_initialize ( void);
```

Arguments

void

> no arguments

canqueue_rtl_done

Name

canqueue_rtl_done — finalization of glopal RT-Linux specific features

Synopsis

```
void canqueue_rtl_done ( void);
```

Arguments

void

>   no arguments

1.6.7. CAN Queues CAN Chips Specific Functions

canqueue_notify_chip

Name

`canqueue_notify_chip` — notification callback handler for CAN chips ends of queues

Synopsis

```
void canqueue_notify_chip (struct canque_ends_t * qends, struct canque_edge_t * qedge, int
what);
```

Arguments

qends

>   pointer to the callback side ends structure

qedge

>   edge which invoked notification

what

>   notification type

Description

This function has to deal with more possible cases. It can be called from the kernel or interrupt context for Linux only compilation of driver. The function can be called from kernel context or RT-Linux thread context for mixed mode Linux/RT-Linux compilation.

canqueue_ends_init_chip

Name

`canqueue_ends_init_chip` — CAN chip specific ends initialization

Synopsis

```
int canqueue_ends_init_chip (struct canque_ends_t * qends, struct canchip_t * chip, struct
msgobj_t * obj);
```

Arguments

qends

>   pointer to the ends structure

chip

   pointer to the corresponding CAN chip structure

obj

   pointer to the corresponding message object structure

canqueue_ends_done_chip

Name

`canqueue_ends_done_chip` — finalizing of the ends structure for CAN chips

Synopsis

```
int canqueue_ends_done_chip (struct canque_ends_t * qends);
```

Arguments

qends

   pointer to ends structure

Return Value

Function should be designed such way to not fail.

1.6.8. CAN Boards and Chip Setup specific Functions

can_checked_malloc

Name

`can_checked_malloc` — memory allocation with registering of requested blocks

Synopsis

```
void * can_checked_malloc (size_t size);
```

Arguments

size

   size of the requested block

Description

The function is used in the driver initialization phase to catch possible memory leaks for future driver finalization or case, that driver initialization fail.

Return Value

pointer to the allocated memory or NULL in the case of fail

can_checked_free

Name

can_checked_free — free memory allocated by can_checked_malloc

Synopsis

int **can_checked_free** (void * *address_p*);

Arguments

address_p

    pointer to the memory block

can_del_mem_list

Name

can_del_mem_list — check for stale memory allocations at driver finalization

Synopsis

int **can_del_mem_list** (*void*);

Arguments

void

    no arguments

Description

Checks, if there are still some memory blocks allocated and releases memory occupied by such blocks back to the system

can_request_io_region

Name

can_request_io_region — request IO space region

Synopsis

int **can_request_io_region** (unsigned long *start*, unsigned long *n*, const char * *name*);

Arguments

start

    the first IO port address

n

    number of the consecutive IO port addresses

name

    name/label for the requested region

Description

The function hides system specific implementation of the feature.

Return Value

returns positive value (1) in the case, that region could be reserved for the driver. Returns zero (0) if there is collision with other driver or region cannot be taken for some other reason.

can_release_io_region

Name

`can_release_io_region` — release IO space region

Synopsis

void **can_release_io_region** (unsigned long *start*, unsigned long *n*);

Arguments

start

    the first IO port address

n

    number of the consecutive IO port addresses

can_request_mem_region

Name

`can_request_mem_region` — request memory space region

Synopsis

int **can_request_mem_region** (unsigned long *start*, unsigned long *n*, const char * *name*);

Arguments

start

    the first memory port physical address

n

    number of the consecutive memory port addresses

name

    name/label for the requested region

Description

The function hides system specific implementation of the feature.

Return Value

returns positive value (1) in the case, that region could be reserved for the driver. Returns zero (0) if there is collision with other driver or region cannot be taken for some other reason.

can_release_mem_region

Name

can_release_mem_region — release memory space region

Synopsis

void **can_release_mem_region** (unsigned long *start*, unsigned long *n*);

Arguments

start
    the first memory port physical address

n
    number of the consecutive memory port addresses

can_base_addr_fixup

Name

can_base_addr_fixup — relocates board physical memory addresses to the CPU accessible ones

Synopsis

int **can_base_addr_fixup** (struct candevice_t * *candev*, unsigned long *new_base*);

Arguments

candev
    pointer to the previously filled device/board, chips and message objects structures

new_base
    candev new base address

Description

This function adapts base addresses of all structures of one board to the new board base address. It is required for translation between physical and virtual address mappings. This function is prepared to simplify board specific xxx_request_io function for memory mapped devices.

can_check_dev_taken

Name

can_check_dev_taken — checks if bus device description is already taken by driver

Synopsis

int **can_check_dev_taken** (void * *anydev*);

Arguments

anydev

    pointer to bus specific Linux device description

Returns

Returns 1 if device is already used by LinCAN driver, 0 otherwise.

register_obj_struct

Name

register_obj_struct — registers message object into global array

Synopsis

int **register_obj_struct** (struct msgobj_t * *obj*, int *minorbase*);

Arguments

obj

    the initialized message object being registered

minorbase

    wanted minor number, if (-1) automatically selected

Return Value

returns negative number in the case of fail

register_chip_struct

Name

register_chip_struct — registers chip into global array

Synopsis

int **register_chip_struct** (struct canchip_t * *chip*, int *minorbase*);

Arguments

chip

the initialized chip structure being registered

minorbase

wanted minor number base, if (-1) automatically selected

Return Value

returns negative number in the case of fail

init_hw_struct

Name

init_hw_struct — initializes driver hardware description structures

Synopsis

int **init_hw_struct** (*void*);

Arguments

void

no arguments

Description

The function init_hw_struct is used to initialize the hardware structure.

Return Value

returns negative number in the case of fail

init_device_struct

Name

init_device_struct — initializes single CAN device/board

Synopsis

int **init_device_struct** (int *card*, int * *chan_param_idx_p*, int * *irq_param_idx_p*);

Arguments

card

index into *hardware_p* HW description

chan_param_idx_p

pointer to the index into arrays of the CAN channel parameters

irq_param_idx_p

pointer to the index into arrays of the per CAN channel IRQ parameters

Description

The function builds representation of the one board from parameters provided

in the module parameters arrays

hw[card] .. hardware type, *io*[card] .. base IO address, *baudrate*[chan_param_idx] .. per channel baudrate, *minor*[chan_param_idx] .. optional specification of requested channel minor base, *irq*[irq_param_idx] .. one or more board/chips IRQ parameters. The indexes are advanced after consumed parameters if the registration is successful.

The hardware specific operations of the device/board are initialized by call to `init_hwspecops` function. Then board data are initialized by board specific `init_hw_data` function. Then chips and objects representation is build by `init_chip_struct` function. If all above steps are successful, chips and message objects are registered into global arrays.

Return Value

returns negative number in the case of fail

init_chip_struct

Name

`init_chip_struct` — initializes one CAN chip structure

Synopsis

```
int init_chip_struct (struct candevice_t * candev, int chipnr, int irq, long baudrate);
```

Arguments

candev

   pointer to the corresponding CAN device/board

chipnr

   index of the chip in the corresponding device/board structure

irq

   chip IRQ number or (-1) if not appropriate

baudrate

   baudrate in the units of 1Bd

Description

Chip structure is allocated and chip specific operations are filled by call to board specific `init_chip_data` which calls chip specific `fill_chipspecops`. The message objects are generated by calls to `init_obj_struct` function.

Return Value

returns negative number in the case of fail

init_obj_struct

Name

`init_obj_struct` — initializes one CAN message object structure

Synopsis

`int` **`init_obj_struct`** `(struct candevice_t * candev, struct canchip_t * hostchip, int objnr);`

Arguments

candev

 pointer to the corresponding CAN device/board

hostchip

 pointer to the chip containing this object

objnr

 index of the builded object in the chip structure

Description

The function initializes message object structure and allocates and initializes CAN queue chip ends structure.

Return Value

returns negative number in the case of fail

init_hwspecops

Name

`init_hwspecops` — finds and initializes board/device specific operations

Synopsis

`int` **`init_hwspecops`** `(struct candevice_t * candev, int * irqnum_p);`

Arguments

candev

 pointer to the corresponding CAN device/board

irqnum_p

 optional pointer to the number of interrupts required by board

Description

The function searches board *hwname* in the list of supported boards types. The board type specific `board_register` function is used to initialize *hwspecops* operations.

Return Value

returns negative number in the case of fail

can_default_irq_dispatch

Name

can_default_irq_dispatch — the first level interrupt dispatch handler

Synopsis

can_irqreturn_t **can_default_irq_dispatch** (int *irq*, void * *dev_id*, struct pt_regs * *regs*);

Arguments

irq

> interrupt vector number, this value is system specific

dev_id

> driver private pointer registered at time of request_irq call. The CAN driver uses this pointer to store relationship of interrupt to chip state structure - *struct* canchip_t

regs

> system dependent value pointing to registers stored in exception frame

File

src/setup.c

can_chip_setup_irq

Name

can_chip_setup_irq — attaches chip to the system interrupt processing

Synopsis

int **can_chip_setup_irq** (struct canchip_t * *chip*);

Arguments

chip

> pointer to CAN chip structure

Return Value

returns negative number in the case of fail

can_chip_free_irq

Name

can_chip_free_irq — unregisters chip interrupt handler from the system

Synopsis

```
void can_chip_free_irq (struct canchip_t * chip);
```

Arguments

chip

   pointer to CAN chip structure

1.6.9. CAN Boards and Chip Finalization Functions

msgobj_done

Name

msgobj_done — destroys one CAN message object

Synopsis

```
void msgobj_done (struct msgobj_t * obj);
```

Arguments

obj

   pointer to CAN message object structure

canchip_done

Name

canchip_done — destroys one CAN chip representation

Synopsis

```
void canchip_done (struct canchip_t * chip);
```

Arguments

chip

   pointer to CAN chip structure

candevice_done

Name

candevice_done — destroys representation of one CAN device/board

Synopsis

```
void candevice_done (struct candevice_t * candev);
```

Arguments

candev

> pointer to CAN device/board structure

canhardware_done

Name

canhardware_done — destroys representation of all CAN devices/boards

Synopsis

```
void canhardware_done (struct canhardware_t * canhw);
```

Arguments

canhw

> pointer to the root of all CAN hardware representation

1.7. LinCAN Usage Information

1.7.1. Installation Prerequisites

The next basic conditions are necessary for the LinCAN driver usage

- some of supported types of CAN interface boards (high or low speed). Not required for *virtual* board setup.
- cables and at least one device compatible with the board or the second computer with an another CAN interface board. Not required for *virtual* board setup. Even more clients can communicate each with another if *process local* is enabled for real chip driver.
- working Linux system with any recent 2.6.x, 2.4.x or 2.2.x kernel (successfully tested on 2.4.18, 2.4.22, 2.2.19, 2.2.20, 2.2.22, 2.6.0 kernels) or working setup for kernel cross-compilation
- installed native and or target specific development tools (GCC and binutils) and pre-configured kernel sources corresponding to the running kernel or intended target for cross-compilation

Every non-archaic Linux distribution should provide good starting point for the LinCAN driver installation.

If mixed mode compilation for Linux/RT-Linux is required, additional conditions has to be fulfilled:

- RT-Linux version 3.2 or higher is required and RT-Linux enabled Linux kernel sources and configuration has to be prepared. The recommended is use of OCERA Linux/RT-Linux release (http://www.ocera.org).

- RT-Linux real-time `malloc` support. It is already included in the OCERA release. It can be downloaded from OCERA web site for older RT-Linux releases as well (http://www.ocera.org/download/components/index.html).

The RT-Linux specific Makefiles infrastructure is not distributed with the current standard LinCAN distribution yet. Please, download full OCERA-CAN package or retrieve sources from CVS by next command:

```
cvs -d:pserver:anonymous@cvs.ocera.sourceforge.net:/cvsroot/ocera login
cvs -z3 -d:pserver:anonymous@cvs.ocera.sourceforge.net:/cvsroot/ocera co ocera/components/comm/can
```

1.7.2. Quick Installation Instructions

Change current directory into the LinCAN driver source root directory

```
cd lincan-dir
```

invoke make utility. Just type '**make**' at the command line and driver should compile without errors

```
make
```

If there is problem with compilation, look at first lines produced by 'make' command or store make output in file. More about possible problems and more complex compilation examples is in the next subsection.

Install built LinCAN driver object file (`can.o`) into Linux kernel loadable module directory (`/lib/modules/2.x.y/kernel/drivers/char`). This and next commands needs root privileges to proceed successfully.

```
make install
```

If device filesystem (devfs) is not used on the computer, device nodes have to be created manually.

```
mknod -m666 /dev/can0 c 91 0
mknod -m666 /dev/can1 c 91 1
...
mknod -m666 /dev/can7 c 97 7
```

The parameters, IO address and interrupt line of inserted CAN interface card need to be determined and configured. The manual driver load can be invoked from the command line with parameters similar to example below

```
insmod can.o hw=pip5 irq=4 io=0x8000
```

This commands loads module with selected one card support for PIP5 board type with IO port base address `0x8000` and interrupt line 4. The full description of module parameters is in the next subsection. If module starts correctly utilities from `utils` subdirectory can be used to test CAN message interchange with device or another computer. The parameters should be written into file `/etc/modules.conf` for subsequent module startup by modprobe command.

Line added to file `/etc/modules.conf` follows

```
options can hw=pip5 irq=4 io=0x8000
```

The module dependencies should be updated by command

```
depmod -a
```

The driver can be now stopped and started by simple **modprobe** command

```
modprobe -r can modprobe can
```

1.7.3. Installation instructions

The LinCAN make solutions tries to fully automate native kernel out of tree module compilation. Make system recurses through kernel `Makefile` to achieve selection of right preprocessor, compiler and linker directives. The description of make targets after make invocation in driver top directory follows

lincan-drv/Makefile (all)

> LinCAN driver top makefile

lincan-drv/src/Makefile (default or all -> make_this_module)

> Needs to resolve target system kernel sources location. This can be selected manually by uncommenting the `Makefile` definition **KERNEL_LOCATION=/usr/src/linux-2.2.22**. The default behavior is to find the running kernel version and look for path to sources of found kernel version in `/lib/modules/2.x.y/build` directory. If no such directory exists, older version of kernel is assumed and makefile tries the `/usr/src/linux` directory.

lib/modules/2.*x.y*/build/Makefile SUBDIRS=.../lincan-drv/src (modules)

> The kernel supplied `Makefile` is responsible for defining of right defines for preprocessor, compiler and linker. If the Linux kernel is cross-compiled, Linux kernel sources root `Makefile` needs be edited before Linux kernel compilation. The variable CROSS_COMPILE should contain development tool-chain prefix, for example **arm-linux-**. The Linux kernel make process recurses back into LinCAN driver `src/Makefile`.

lincan-drv/src/Makefile (modules)

> This pass starts real LinCAN driver build actions.

If there is problem with automatic build process, the next commands can help to diagnose the problem.

```
make clean make >make.out 2>&1
```

The first lines of file `make.out` indicates auto-detected values and can help with resolving of possible problems.

```
make -C src default ;
make -C utils default ;
make[1]: /scripts/pathdown.sh: Command not found
make[1]: Entering directory '/usr/src/can-0.7.1-pi3.4/src'
echo >.supported_cards.h echo \#define ENABLE_CARD_pip 1 >>.supported_cards.h ; ...
Linux kernel version 2.4.19
echo Linux kernel sources /lib/modules/2.4.19/build
Linux kernel sources /lib/modules/2.4.19/build
echo Module target can.o
Module target can.o
echo Module objects proc.o pip.o pccan.o smartcan.o nsi.o ...
make[2]: Entering directory '/usr/src/linux-2.4.19'
```

The driver size can be decreased by restricting of number of supported types of boards. This can be done by editing of definition for SUPPORTED_CARDS variable.

There is complete description of driver supported parameters.

```
insmod can.o hw='your hardware' irq='irq number' io='io address' <more options>
```

The more values can be specified for *hw*, *irq* and *io* parameters if more cards is used. Values are separated by commas in such case. The *hw* argument can be one of:

- `pip5`, for the PIP5 computer by MPL AG
- `pip6`, for the PIP6 computer by MPL AG
- `pip7`, for the PIP7 computer by MPL AG
- `pip8`, for the PIP8 computer by MPL AG
- `pccan-q`, for the PCcan-Q ISA card by KVASER
- `pccan-f`, for the PCcan-F ISA card by KVASER

- `pccan-s`, for the PCcan-S ISA card by KVASER
- `pccan-d`, for the PCcan-D ISA card by KVASER
- `pcican-q`, for the PCIcan-Q PCI card by KVASER (4x SJA1000)
- `pcican-d`, for the PCIcan-D PCI card by KVASER (2x SJA1000)
- `pcican-s`, for the PCIcan-S PCI card by KVASER (1x SJA1000)
- `nsican`, for the CAN104 PC/104 card by NSI
- `cc104`, for the CAN104 PC/104 card by Contemporary Controls
- `aim104`, for the AIM104CAN PC/104 card by Arcom Control Systems
- `pc-i03`, for the PC-I03 ISA card by IXXAT
- `pcm3680`, for the PCM-3680 PC/104 card by Advantech
- `m437`, for the M436 PC/104 card by SECO
- `bfadcan` for sja1000 CAN embedded card made by BFAD GmbH
- `pikronisa` for ISA memory mapped sja1000 CAN card made by PiKRON Ltd.
- `template`, for yet unsupported hardware (you need to edit `src/template.c`)
- `virtual`, virtual/dummy board support for testing of driver and software devices and applications

The lists of values for board hardware type (*hw*) and board base IO address (*io*) parameters have to contain same number of values. If the value of *io* has no meaning for specified hardware type (`virtual` or PCI board), it has to be substituted by `0`.

The number of required *irq* values per board is variable. The `virtual` and PCI board demands no value, most of the other boards requires one *irq* value per each chip/channel.

The *<more options>* can be one or more of:

- *major=<nr>*, major specifies the major number of the driver. Default value is 91
- *minor=<nr>*, you can specify which base minor number the driver should use for each can channel/chip. Consecutive numbers are taken in the case, that chip supports more communication objects. The values for channels are separated by comas
- *extended=[1/0]*, enables automatic switching to extended format if ID>2047, selects extended frames reception for i82527
- *pelican=[1/0]*, unused parameter, PeliCAN used by default for sja1000p chips now
- *baudrate=<nr>*, baudrate for each channel in step of 1kBd
- *clock_freq=<nr>*, the frequency of the CAN quartz for BfaD board
- *stdmask=<nr>*, default standard mask for some (i82527) chips
- *extmask=<nr>*, default extended mask for some (i82527) chips
- *mo15mask=<nr>*, sets the mask for message object 15 (i82527 only)
- *processlocal=<nr>*, select post-processing/loop-back of transmitted messages

  0 .. disabled

  1 .. can be enabled by application by FIFO filter setup

  2 .. enabled by default

- *can_rtl_priority=<nr>*, select priority of chip worker thread for driver compiled with RT-Linux support

Actual list of supported CAN module parameters and short description can be reached by invocation of the command

```
modinfo can
```
.

1.7.4. Simple Utilities

The simple test utilities can be found in the `utils` subdirectory of the LinCAN driver source subtree. These utilities can be used as base for user programs directly communicating with the LinCAN driver. We do not suggest to build applications directly dependent on the driver operating system specific interface. We suggest to use the VCA API library for communication with the driver which brings higher level of system interface abstraction and ensures compatibility with the future versions of LinCAN driver and RT-Linux driver clone versions. The actual low level RT-Linux API to LinCAN driver closely matches `open/close`, `read/write` and `ioctl` interface. Only `select` cannot be provided directly by RT-Linux API.

The basic utilities provided with LinCAN driver are:

rxtx

the simple utility to receive or send message which guides user through operation, the message type, the message ID and the message contents by simple prompts

send

even more simplistic message sending program

readburst

the utility for continuous messages reception and printing of the message contents. This utility can be used as an example of the `select` system call usage.

sendburst

the periodic message generator. Each message is filled by the constant pattern and the message sequence number. This utility can be used for throughput and message drops tests.

can-proxy

the simple TCP/IP to CAN proxy. The proxy receives simple commands from IP datagrams and processes command sending and state manipulations. Received messages are packed into IP datagrams and send back to the client.

readburst

Name

`readburst` — the utility for continuous messages reception and printing of the message contents

Synopsis

readburst   [-d *candev*] [-m *mask*] [-i *id*] [-f *flags*] [-w *sec*] [-p *prefix*][-V][-h]

Description

The utility **readburst** can be used to monitor or log CAN messages received by one CAN message communication object. Even outgoing transmitted messages can be logged if *process local* is globally or explicitly enabled.

OPTIONS

-d --device

This options selects **readburst** target CAN device. If the option is not specified, default device name /dev/can0 is used.

-m --mask

This option enables to change default mask accepting all messages to the specified CAN message id mask. The hexadecimal value has to be prefixed by prefix `0x`. Numeric value without any prefix is considered as decimal one.

```
-i --id
```
This option specifies CAN message identifier in the acceptance mask. The accepted CAN messages are then printed by **readburst** command. Only bits corresponding to the non-zero bits of acceptance mask are compared. Hexadecimal value has to be prefixed by any prefix `0x`. Numeric value without prefix is considered as decimal one.

```
-f --flags
```
Specification of modifiers flags of reception CAN queur. Hexadecimal value has to be prefixed by prefix `0x`. Numeric value without any prefix is considered as decimal one.

Bit name	Bit number	Mask	Description
`MSG_RTR`	0	0x1	Receive RTR or non-RTR messages
`MSG_EXT`	2	0x4	Receive extended/standard messages
`MSG_LOCAL`	3	0x8	Receive local or external messages
`MSG_RTR_MASK`	8	0x100	Take care about `MSG_RTR` bit else RTR and non-RTR messages are accepted
`MSG_EXT_MASK`	10	0x400	Take care about `MSG_EXT` bit else extended and standard messages are accepted
`MSG_LOCAL_MASK`	11	0x800	Take care about `MSG_LOCAL` bit else both local and external messages are accepted
`MSG_PROCESSLOCAL`	9	0x200	Enable processing of the local messages if not explicitly enabled globally or disabled globally.

```
-w --wait
```
The number of second the **readburst** waits in the `select` call.

```
-p --prefix
```
The *prefix* string can is added at beginning of each printed line. The format specifies `%s` could be used to add device name into prefix.

```
-V --version
```
Print command version.

```
-h --help
```
Print command usage information

sendburst

Name

`sendburst` — the utility for continuous messages reception and printing of the message contents

Synopsis

sendburst [-d *candev*] [-i *id*] [-s] [-f *flags*] [-w *sec*] [-b *blocksize*] [-c *count*] [-p *prefix*] [-V] [-h]

Description

The utility **sendburst** generates blocks of messages with specified CAN message ID. The burst block of *blocksize* messages is generated and pushed into can device. If *count* is specified, the command stops and exits after *count* of message blocks send.

OPTIONS

`-d --device`

> This options selects **sendburst** target CAN device. If the option is not specified, default device name `/dev/can0` is used.

`-i --id`

> This option specifies which CAN message ID is used for transmitted blocks of messages. Hexadecimal value has to be prefixed by prefix `0x`. Numeric value without any prefix is considered as decimal one.

`-f --flags`

> Specification of modifiers flags of the send message. Hexadecimal value has to be prefixed by prefix `0x`. Numeric value without prefix is considered as decimal one.

Bit name	Bit number	Mask	Description
MSG_RTR	0	0x1	Generate RTR messages if specified
MSG_EXT	2	0x4	Use extended messages identifiers if specified

`-s --sync`

> Open device in the synchronous mode. The `send` and `close` blocks until message is sent to to CAN bus.

`-w --wait`

> The number of second the **sendburst** waits between sending burst blocks.

`-b --block`

> The number of messages in the one burst block. Default value is `10`.

`-c --count`

> The number of block send after command invocation. If specified, command finishes and returns after specified number of blocks. If unspecified, the **sendburst** runs for infinite time.

`-p --prefix`

> The *prefix* string can is added at beginning of each printed line. The format specifies `%s` could be used to add device name into prefix.

`-V --version`

> Print command version.

`-h --help`

> Print command usage information