Socket-CAN Device Support for EPICS IOCs*

TECHNISCHE UNIVERSITÄT DARMSTADT

C. Burandt†, U. Bonnes, J. Enders, N. Pietralla

Institut für Kernphysik, Technische Universität Darmstadt, Germany

CAN Bus

The In-house Developed Hardware Family

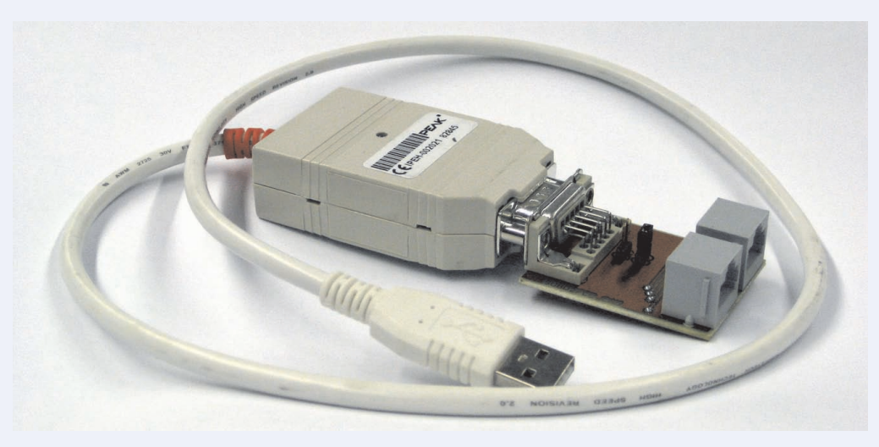
- magnet power supplies, low-level RF, multi-purpose measurement system, ...
- uniform firmware running on microcontrollers for all types of hardware
- connected via CAN bus to PC

CAN Bus Properties as used at the S-DALINAC

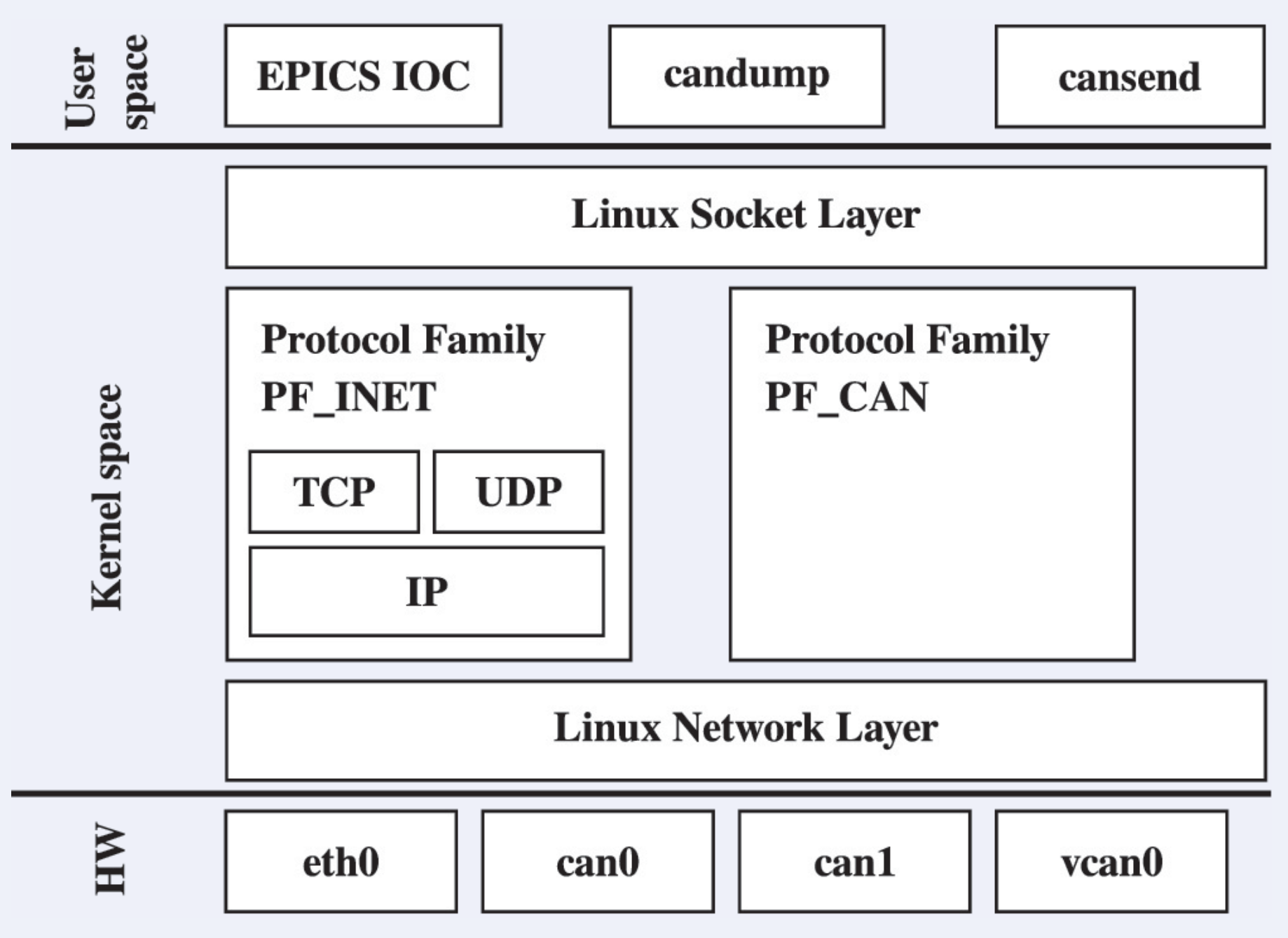
Bit Rate	1 Mbit/s
physical length of bus	$\leq$ 40 m (for bit rates of 1 Mbit/s)
Frame Size	8 byte of user data
Address Format	29 bit (extended frame format)

PC Interface Hardware

- PCI/PCIexpress slot cards used in PCs running the EPICS IOCs
- USB interface for on-site diagnostics



Linux Socket-CAN Network Stack



What it is

- part of the Linux kernel main line since version 2.6.25
- some manufacturers supply their own Socket-CAN device driver
- many CAN interface device drivers already included

How it works

- analogous to internet protocols like TCP/IP (protocol family)
- connection endpoint is represented as a BSD socket

How to use it

- treatment similar to ethernet devices
- setup of virtual CAN device possible

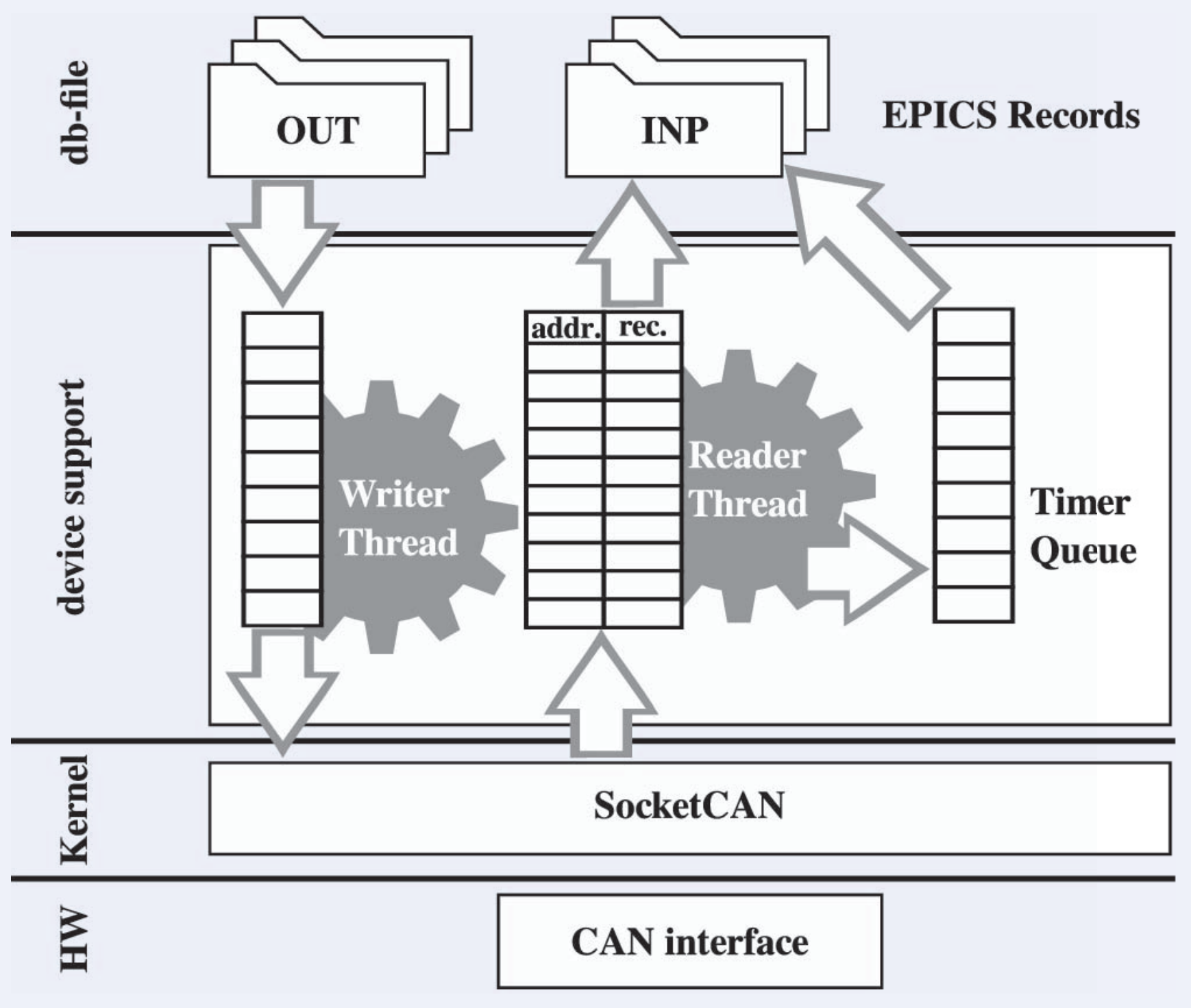
```
# ip link set can0 up type can bitrate 10000000

# modprobe vcan
# ip link add dev canXY type vcan
# ip link set vcan0 up type vcan
```

The tudSocketCan Device Support

Design Overview

- at the S-DALINAC polling is preferred for the in-house developed hardware family
- broken devices can be quickly identified (timeouts are to be defined therefore)
- CAN frames can carry more than one piece of information and thus need to be distributed to multiple input records
- different threads use different tables/queues to handle reading/writing and the timeout behaviour



Software Engineering Features

- approx. 1300 lines of plain C
- build dependencies are limited to header files which are part of the Linux kernel source or EPICS base
- not dependent of the PC CAN interface manufacturer in any way

Supported Record Types

- analog in/out
- long in/out
- binary in/out
- multi binary in/out
- multi binary direct in/out

Not Yet Supported Record Types

- different record types can be implemented in principle
- complex record types like `string in` and `string out` require transmission of multiple CAN frames, which is different from the record types implemented so-far

Using tudSocketCan Device Support

Showcase for a Record Definition

```
record(ai, "rf:rdSupplyVoltADC") {
  field(LINR, "SLOPE")
  field(ESLO, "9.3e-9")
  field(EOFF, "0")
  field(EGU, "V")
  field(SCAN, "I/O Intr")
  field(DTYP, "tudSocketCan")
  field(INP, "@can0 07 01 02 519 02 1 sl 50")
}
```

The template designer needs certain information:

• `@can0`	:	the CAN interface to be used
• `07`	:	marks the CAN frame as being sent from a device (as opposed to being sent to a device)
• `01 02`	:	address of the device
• `519`	:	defines the specific request
• `02`	:	specifies the ADC
• `1`	:	omit the first data byte, when converting the value, since it is the one used to identify the ADC
• `sl`	:	interpret value as signed long
• `50`	:	timeout in seconds

Addressing Scheme

The hardware is organized in crates which are connected to a CAN bus segment.

- an individual number is assigned to each crate.
- the slots are numbered accordingly to their position inside the crate.

Template File

A Perl script allows to use mnemonics like LL08_GET_ADC instead of numerical codes.

```
...
field(INP, "$(can_interface) 07 $(crate)
        $(slot) LL08_GET_ADC 02 1 sl 50")
...
```

A certain CAN frame can be handed to multiple records. The INP fields can either be identical or differ for example in the byte offset.

Only get the additional byte which contains status information of the ADC:

```
...
field(INP, "$(can_interface) 07 $(crate)
        $(slot) LL08_GET_ADC 02 5 uc 50")
...
```

Debugging of Templates

Socket-CAN Tools

The `cansend` and `candump` tools allow to send and receive arbitrary CAN frames.

• `cansend`	:	send single CAN frame
• `candump`	:	dump the traffic of a complete CAN bus segment

Thus, the raw communication can be tested manually. Together with the `caput` and `camonitor` command-line tools the whole communication and conversion chain can be validated.

Example

Request supply voltage by hand and observe the answer with candump and camonitor.

```
# camonitor rf:rdSupplyVoltADC    &
 rf:rdSupplyVoltADC  2012-11-30 09:39:31 0

# candump can0 &

# cansend can0 06044207#02
 can0    6044207  [1] 02
 can0    7044207  [6] 02 53 E9 90 02 22

 rf:rdSupplyVoltADC  2012-11-30 09:39:39 0.61
```

Summary

Experience

- first used for the digital low-level rf control system
- in a production environment for two years now
- growing number of different devices controlled through this device support

Future

- rewrite in C++ programming language is proposed
- C++ STL provides many usefull components which allow for a much cleaner software design
- improve bus error detection and treatment
- support record types which require multiple CAN frames to be transmitted

*Work supported by DFG through CRC 634

†burandt@ikp.tu-darmstadt.de

DFG

SFB 634