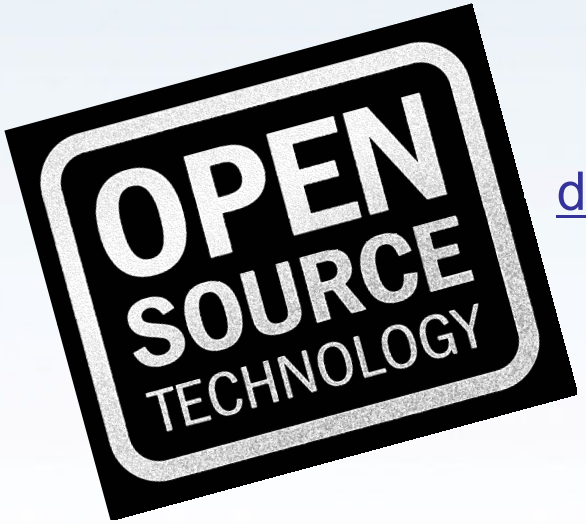
SocketCAN

CAN Driver Interface under Linux



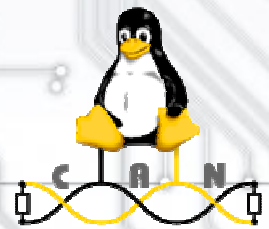
Daniel Krüger
daniel.krueger@systec-electronic.com

SYS TEC electronic GmbH
www.systec-electronic.com

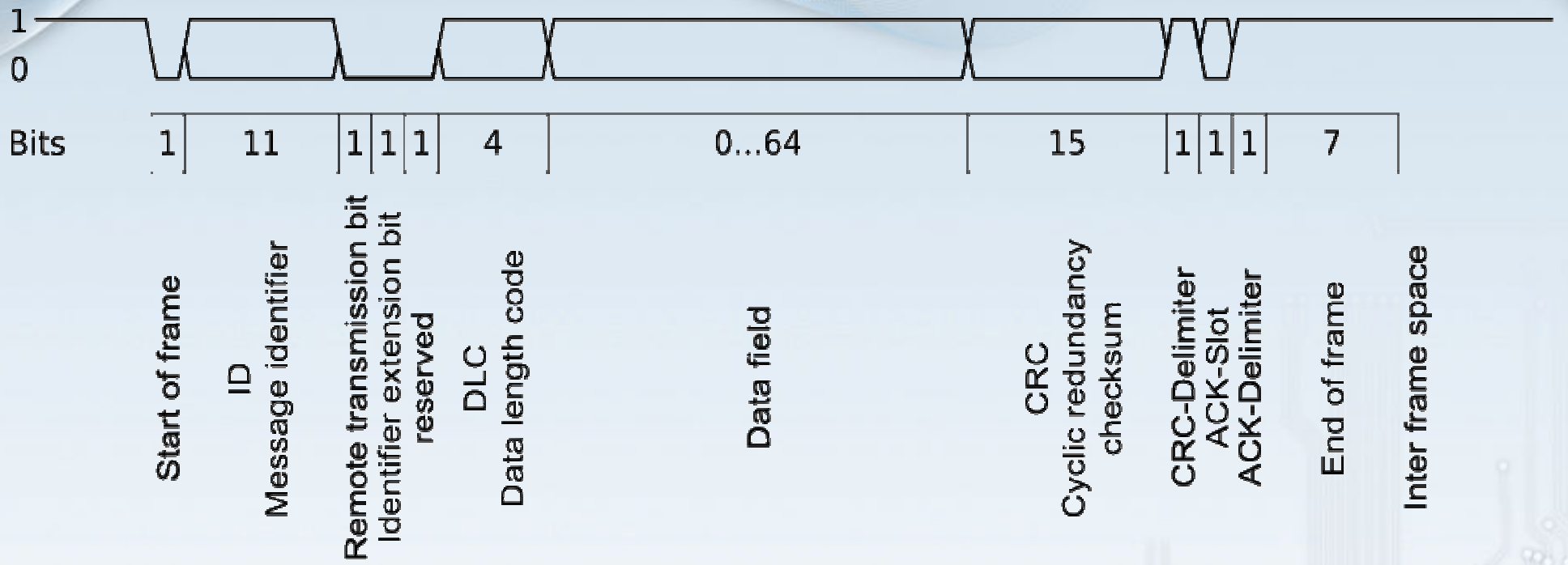
Version 3/19/2012

What is CAN?

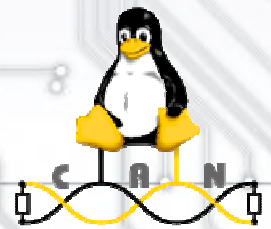
- CAN = Controller Area Network
- Developed by Bosch (starting in 1983)
- Multi master network
- Short broadcast messages (up to 8 Byte)
- Bit rate: up to 1 MBit/s
- Network length up to 5 km
  (depending on the used bit rate)
- Powerful error detection mechanism
  - minimum error rate of $4{,}7 * 10^{-11}$



CAN Standard Frame Format



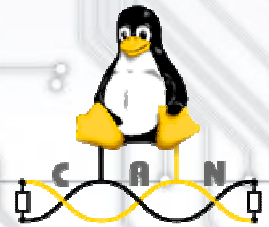
- Dominant bit: logical 0
- Recessive bit: logical 1
- Lowest message identifier wins bus arbitration
- Message identifier: no unique node-ID



SocketCAN

SYS TEC ELECTRONIC

- SocketCAN is the framework for CAN under Linux
- Replaces plenty of vendor-specific CAN APIs
- CAN drivers are network drivers
- Applications receive and transmit CAN messages via BSD Socket API
- Configuration of CAN interfaces: via netlink protocol
- Mainline since Linux 2.6.25



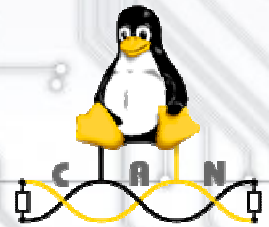
SocketCAN

SYS TEC ELECTRONIC

- linux/can.h

```
struct can_frame {
    u32 can_id;  /* 29 bit CAN_ID + flags */
    u8  can_dlc; /* data length code: 0 .. 8 */
    u8  data[8];
};

#define CAN_EFF_FLAG 0x80000000U
                          /* extended frame format */
#define CAN_RTR_FLAG 0x40000000U
                        /* remote transmission request */
#define CAN_ERR_FLAG 0x20000000U /* error frame */
```

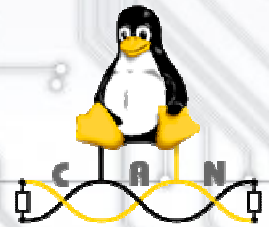
- linux/can/...



Setup CAN channel

SYS TEC ELECTRONIC

- Configure bit rate:
  `$ ip link set can0 type can bitrate 125000`
- Set interface up and running:
  `$ ifconfig can0 up`
- Common pitfall:
  Standard distro kernels do not enable CONFIG_CAN_CALC_BITTIMING
- Bitrate setting
  `$ ip link set can0 type can bitrate 125000`



SocketCAN initialization

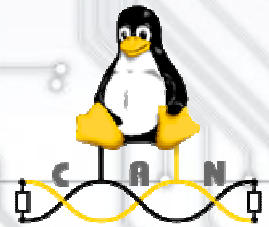
SYS TEC ELECTRONIC

```
int iSock;
struct sockaddr_can addr;


iSock = socket(PF_CAN, SOCK_RAW, CAN_RAW);
addr.can_family = AF_CAN;
addr.can_ifindex = if_nametoindex("can0");
bind(iSock, (struct sockaddr *)&addr,
  sizeof(addr);
```

Send CAN message

SYS TEC ELECTRONIC

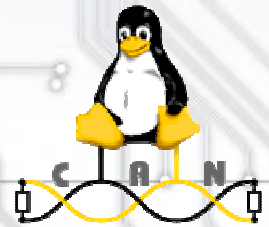
```
struct can_frame frame;


frame.can_id = 0x123;
frame.can_dlc = 1;
frame.data[0] = 0xAB;


nbytes = write(iSock, &frame,
  sizeof(frame));
```



Receive CAN message

SYS TEC ELECTRONIC

```
struct can_frame frame;


nbytes = read(iSock, &frame, sizeof(frame));
if (nbytes > 0) {
   printf("ID=0x%X DLC=%d data[0]=0x%X\n",
      frame.can_id,
      frame.can_dlc,
      frame.data[0]);
}
```



CAN error handling

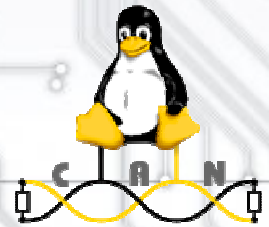
SYS TEC ELECTRONIC

- frame.can_id & CAN_ERR_FLAG
  - frame.can_id & CAN_ERR_BUSOFF
  - frame.can_id & CAN_ERR_ACK
  - frame.can_id & CAN_ERR_RESTARTED
  - frame.can_id & CAN_ERR_CRTL
    - frame.data[1] & (CAN_ERR_CRTL_RX_WARNING | CAN_ERR_CRTL_TX_WARNING)
    - frame.data[1] & (CAN_ERR_CRTL_RX_PASSIVE | CAN_ERR_CRTL_TX_PASSIVE)
    - frame.data[1] & (CAN_ERR_CRTL_RX_OVERFLOW | CAN_ERR_CRTL_TX_OVERFLOW)
  - frame.can_id & CAN_ERR_PROT

CAN bus-off

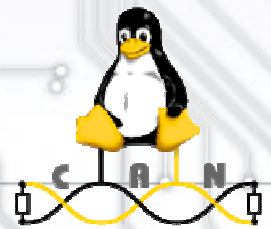
SYS TEC ELECTRONIC

- CAN controller enters bus-off state,
  when internal error counters reach a limit,
  e.g. in case of short-circuit between CAN_H, CAN_L
- Live demo,
  `$ ip -det -stat link show can0`
- Recover from bus-off:
  `$ ip link set can0 type can restart`



Diagnostics

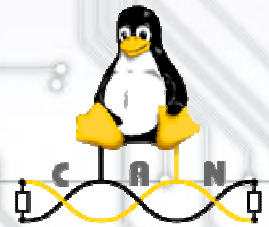
```
$ ip -det -stat link show can0

9: can0: <NOARP,UP,LOWER_UP,ECHO> mtu 16 qdisc pfifo_fast state
   UNKNOWN qlen 10

    link/can 00:02:48:a2:03:00 brd 00:00:00:00:00:00

    can state ERROR-ACTIVE restart-ms 0

    bitrate 1000000 sample-point 0.750

    tq 62 prop-seg 5 phase-seg1 6 phase-seg2 4 sjw 1

    systec_can: tseg1 1..16 tseg2 1..8 sjw 1..4 brp 1..255 brp-inc 1

    clock 48000000

    re-started bus-errors arbit-lost error-warn error-pass bus-off

    0          0          0          0          0          0

    RX: bytes  packets  errors  dropped overrun mcast

    0          0        0       0       0       0

    TX: bytes  packets  errors  dropped carrier collsns

    0          0        0       0       0       0
```



CAN utilities

- Source: https://gitorious.org/linux-can/can-utils
  $ git clone git://gitorious.org/linux-can/can-utils.git

```
$ ./candump can0
  can0  4A2  [4] 27 96 C1 6C
  can0  151  [8] ED 85 FA 65 0D EB C2 4A
  can0  123  [2] AB CD
$ ./cangen can0
$ ./cansend can0 123#abcd
```



Untouched subjects

SYS TEC ELECTRONIC

- Higher layer protocols: CANopen, J1939, DeviceNet
- Extended frame format (29 bit CAN identifier)
- CAN message filtering
- Bit rate calculation
- Error handling: CAN error frames, bus-off recovery
- CAN driver structure

Conclusion

- SocketCAN
  - Uniform and well-defined CAN framework for Linux
- CAN
  - Flexible
  - Powerful
  - Cost-effective
- Future: CAN with Flexible Data-Rate (CAN FD)
  - Higher bit rates
  - Longer data fields (more than 8 Byte)
  - Bus arbitration is identical to classic CAN
  - First CAN FD controllers expected at end of 2012

References

- Specification: CAN 2.0 (by Bosch in 1991)
- Standards: ISO 11898-1:2003, ISO 11898-2:2003
- SocketCAN: https://gitorious.org/linux-can/
- CAN FD: http://www.bosch-semiconductors.de/media/pdf/canliteratur/can_fd.pdf

SYS TEC ELECTRONIC

Thank you for your attention