can4linux — CAN Treiber für passive Karten mit CAN 82c200/SJA1000 Controllern

1. Installation

Entpacken sie den Treiber-Kode can4linux*tgz in einem vorher eingerichteten Verzeichnis, z.B *can4linux*

```
$ mkdir can4linux
$ cd can4linux
$ tar zxvf ../can4linux*tgz
```

Der Treiber wird für jede dafür vorgesehene Hardware speziell übersetzt. Eine Liste der möglichen Interface-Boards enthält das Makefile, das auch den Übersetzungsvorgang steuert. Übersetzen sie den Treiber, z.B:

```
$ make help


$ make TARGET=ATCANMINI_PELICAN
```

Es entsteht im aktuellen Verzeichnis die Datei *Can.o*, die Treiber-Objektdatei. *Can.o*

Erzeugen sie nun Einträge für Gerätedateien. Rufen sie dazu make auf:

```
$ make inodes
```

Die Voreinstellung für die major device number ist 91. Prüfen sie vorher, ob diese Nummer schon im System verwendet wird. (Sie ist jedoch offiziell für CAN reserviert.)

```
$ cat /proc/devices
```

Gehen sie in das Verzeichnis *etc*. Suchen sie sich eine geeignete Konfiguration aus und editieren diese.

In den meisten Fällen können sie die vorhandenen Dateien: *2-at_can_mini.conf* oder *1-cpcpci.conf* verwenden.

Finden sie keine vorkonfigurierten Einträge, welche auf ihre Hardware zutreffen, sollten sie sehr sorgfältig ihre Hardware studieren und einen neuen Eintrag anlegen.

Legen sie eine Konfigurationsdatei, benannt nach ihrem Rechner, an. Den Rechnernamen können sie abfragen mit:

```
$ uname -n
uschi
$ cp 1-cpcpci.conf uschi.conf
```

Viele Einträge sind von der Hardware abhängig. - Vorsicht beim Einsatz selbstentwickelter Hardware.

Die Einträge in der Konfigurationsdatei werden über die entsprechenden Einträge des Treibers im */proc* Dateisystem geschrieben

```
/proc/sys/Can/*
```

Führen sie nun

```
$ make load
```

aus. Diese Anweisung lädt den Treiber Can.o mit dem Betriebssystemkommando `insmod`(1) und überschreibt anschließend die Einträge in */proc/sys/Can/** mit den von ihnen gewählten Werten.

ACHTUNG! Beim Einsatz von PCI Karten verwendet der Treiber für Adressen und Interruptnummern sowie für die Art des Zugriffes Werte, welche er vom BIOS erhält. Diese Werte können dann nicht geändert bzw. überschrieben werden - ignorieren sie einfach diese Fehlermeldung beim Laden der Konfiguration.

2. Test

Gehen sie nun in das Verzeichnis *examples.* Übersetzen sie die Applikationen.

```
$ cd examples
$ make
gcc -Wall -I../src -DUSE_RT_SCHEDULING   -c -o ctest.o ctest.c
gcc   ctest.o   -o ctest
gcc -Wall -I../src -DUSE_RT_SCHEDULING   -c -o baud.o baud.c
gcc   baud.o   -o baud
gcc -Wall -I../src -DUSE_RT_SCHEDULING   -c -o can_send.o can_send.c
gcc   can_send.o   -o can_send
gcc -Wall -I../src -DUSE_RT_SCHEDULING   -c -o acceptance.o acceptance.c
gcc   acceptance.o   -o acceptance
gcc -Wall -I../src -DUSE_RT_SCHEDULING   -c -o noiser.o noiser.c
gcc   noiser.o   -o noiser
gcc -Wall -I../src -DUSE_RT_SCHEDULING   -c -o receive.o receive.c
gcc   receive.o   -o receive
gcc -Wall -I../src -DUSE_RT_SCHEDULING   -c -o transmit.o transmit.c
gcc   transmit.o   -o transmit
gcc -Wall -I../src -DUSE_RT_SCHEDULING   -c -o can_verify.o can_verify.c
gcc   can_verify.o   -o can_verify
```

Bevor sie nun `ctest` aufrufen, beobachten sie die Ausschriften des Treibers in der Datei */var/log/messages.* Dazu öffnen sie sie in einem separatem **xterm**

```
$ tail -f /var/log/messages
```

(Superuser Rechte meist erforderlich.)

Um mehr Meldungen vom Treiber zu erhalten, erhöhen sie vorher den Debuglevel durch Schreiben auf den Eintrag

```
echo 7 > /proc/sys/Can/dbgMask
```

(Superuser Rechte meist erforderlich.)

oder sie benutzen das Skript

```
./debug 7
```

Nach dem Start von `ctest` sehen sie in der Log-Datei:

```
Sep 17 13:13:31 uschi kernel: Can: - :in  can_open
Sep 17 13:13:31 uschi kernel: Can: - :in  CAN_VendorInit
Sep 17 13:13:31 uschi kernel: Can: - :in  Can_RequestIrq
Sep 17 13:13:31 uschi kernel: Can: - :Requested IRQ: 5 @ 0xce2e28c0
Sep 17 13:13:31 uschi kernel: Can: - :out
Sep 17 13:13:31 uschi kernel: Can: - :out
Sep 17 13:13:31 uschi kernel: Can: - :in  Can_WaitInit
Sep 17 13:13:31 uschi kernel: Can: - :out
Sep 17 13:13:31 uschi kernel: Can: - :in  Can_FifoInit
Sep 17 13:13:31 uschi kernel: Can: - :out
Sep 17 13:13:31 uschi kernel: Can: - :in  CAN_ChipReset
Sep 17 13:13:31 uschi kernel: Can: - : INT 0x0
Sep 17 13:13:31 uschi kernel:
Sep 17 13:13:31 uschi kernel: Can: - :status=0x3c mode=0x1
Sep 17 13:13:31 uschi kernel: Can: - :[0] CAN_mode 0x1
Sep 17 13:13:31 uschi kernel:
Sep 17 13:13:31 uschi kernel: Can: - :[0] CAN_mode 0x9
Sep 17 13:13:31 uschi kernel:
Sep 17 13:13:31 uschi kernel: Can: - :[0] CAN_mode 0x9
Sep 17 13:13:31 uschi kernel:
Sep 17 13:13:31 uschi kernel: Can: - :in  CAN_SetTiming
Sep 17 13:13:31 uschi kernel: Can: - :baud[0]=125
Sep 17 13:13:31 uschi kernel: Can: - :tim0=0x3 tim1=0x1c
Sep 17 13:13:31 uschi kernel: Can: - :out
Sep 17 13:13:31 uschi kernel: Can: - :[0] CAN_mode 0x9
Sep 17 13:13:31 uschi kernel:
Sep 17 13:13:31 uschi kernel: Can: - :in  CAN_SetMask
Sep 17 13:13:31 uschi kernel: Can: - :[0] acc=0xffffffff mask=0xffffffff
Sep 17 13:13:31 uschi kernel: Can: - :out
Sep 17 13:13:31 uschi kernel: Can: - :[0] CAN_mode 0x9
Sep 17 13:13:31 uschi kernel:
Sep 17 13:13:31 uschi kernel: Can: - :out
Sep 17 13:13:31 uschi kernel: Can: - :in  CAN_StartChip
```

```
Sep 17 13:13:31 uschi kernel: Can: - :[0] CAN_mode 0x9
Sep 17 13:13:31 uschi kernel:
Sep 17 13:13:31 uschi kernel: Can: - :start mode=0x8
Sep 17 13:13:31 uschi kernel: Can: - :out
Sep 17 13:13:31 uschi kernel:  MODE 0x8, STAT 0x3c, IRQE 0xf,
Sep 17 13:13:31 uschi kernel: Can: - :out
```

Andere Ausschriften deuten auf fehlerhafte *.conf Konfigurationen oder nicht unterstützte Hardware hin.

Starten sie nun `can_send` zum Versenden einer Nachricht mit 8 Datenbyte mit dem Dateninhalt 0x55:

```
$ can_send -D can0 0x555 0x55 0x55 0x55 0x55 0x55 0x55 0x55 0x55
```

Nach Ausführen des Kommandos `can_send` sollte auf dem CAN Bus eine Nachricht erscheinen. Ist der Bus nicht mit einer CAN Gegenstelle abgeschlossen, versucht der CAN Controller diese Message immer wieder abzusetzen, da er kein Acknowledge bekommt. Dies ist sehr gut mit einem Oszillographen zu beobachten. Durch den gewählten Identifier und das Muster der Datenbytes, sollte sehr einfach der kürzeste Signalwechsel erkennbar und ausmessbar sein. Bei 125 kBit/s beträgt die Zeit für den kürzesten Signalwechsel 8 µs.

Setzen sie den Debuglevel des Treibers zurück:

```
./debug 0
```

3. Einträge in /proc/sys/Can

Siehe dazu auch die Beispiele in den Konfigurationsdateien *etc/*.conf*.

Eintrag	pro Kanal	Bedeutung
AccCode	*	CAN Controller Acceptance Code Register
AccMask	*	CAN Controller Acceptance Maske
Base	*	CAN Controller Adresse
Baud	*	Bitrate in kBit/s
IOModel		Ein Buchstabe pro Kanal für IO-Model
IRQ	*	IRQ Nummer
Outc	*	Output Control Register
Overrun	*	Overrun Flag des Kanals
RxErr	*	Anzahl der Rx Fehler
Timeout	*	Timeout Wert
TxErr	*	Anzahl der TX Fehler
dbgMask		globaler DebugLevel
version		Versionsstring
Chipset		vom Treiber unterstützter CAN controller

Für die CAN Bitraten 10,20,40,50,125,250,500,800 und 1000 kBit/s werden die Werte für die Bit-Timing Register des CAN Controllers aus internen Tabellen geladen. Diese Tabellen gelten jedoch nur für eine Taktversorgung des CAN Controllers mit 8 MHz (externer 16 MHz Quartz). Bei abweichenden Werten wird der Inhalt der Bit-Timing Register wie folgt berechnet:

```
BTR0 = (Baud >> 8) && 0xFF
BTR1 =  Baud       && 0xFF
```

Beispiel für die Einstellung von 125 kBit/s für einen SJA1000 Controller mit einer Taktfrequenz von 10 MHz (20MHz), 16 Quanten und einem Abtastpunkt (Sampling Point) bei 87.5%:

```
Berechnete Werte für die Bit-Timing Register:
                                              BTR0 = 0x04
                                              BTR1 = 0x1c


Daraus folgt:
                  Baud = (0x04 << 8) + 0x1c = 0x041c = 1052
```

Die Internet-Seite http://www.port.de/deutsch/canprod/sv_req_form.html bietet ein Formular zur Berechnung der Werte für die Bit-Timing Register.

dbgMask
: globaler DebugLevel

Standard 0 - keine Debugausgaben;
Jedes Bit dieser Maske hat eine bestimmte Bedeutung:

Bit	Bedeutung
0	Flag for setting all options=on
1	log function entries
2	log function exits
3	log branches
4	log data given to functions
5	log interrupts
6	log register info
7	reserved

Outc Das Output Control Register beim Philips 82C200/SJA1000

4. DIL-NET-PC TRM/816 Hardware by SSV embedded Systems

Der Kode wurde von Sven Geggus ⟨`geggus@iitb.fraunhofer.de`⟩. bereitgestellt. Bitte lesen Sie dazu *README.trm816* und auch *trm816/README*.

5. Vorgehen im Fehlerfall

Bei allen Fehlfunktionen des Treibers, z.B. **open**() returns EINVAL - invalid argument -, **dbgMask** Einschalten und */var/log/messages* ansehen.