

Installation und Evaluation von MeeGo

Kommunikation mit dem CAN-Bus

Installation and evaluation of MeeGo

Communication with CAN bus

Sven Killig

Projektarbeit zum Diplom-Fernstudium Informatik

Betreuer: Prof. Dr. Jörn Schneider

Schwabach, 3.10.2011

Kurzfassung

Mit dem proTRon III und AERIS entwickelte die FH Trier bereits in der Vergangenheit eigene Fahrzeuge zur Teilnahme am Shell Eco-Marathon. Für die Neuentwicklung proTRon evolution ist geplant, auch ein eigenes In-Vehicle Infotainment (IVI)-System zu entwickeln. In der vorliegenden Arbeit dokumentiere ich zwei dafür notwendige Aspekte: Zum einen die Installation des Betriebssystems MeeGo auf embedded-Hardware, zum anderen die Kommunikation mit dem CAN-Bus auf dieser.

Inhaltsverzeichnis

| | | |
|----------|---|-----------|
| 1 | Meego..... | 1 |
| 1.1 | Einführung | 1 |
| 1.2 | Test auf einem PC | 1 |
| 1.3 | Erstellung des root filesystems und Booten auf dem QSB..... | 1 |
| 1.3.1 | i.MX53 | 1 |
| 1.3.2 | QSB | 2 |
| 1.3.3 | MIC..... | 2 |
| 1.3.4 | Qemu | 3 |
| 1.3.5 | Device mapper | 3 |
| 1.3.6 | Bootstrapping | 3 |
| 1.3.7 | Zielort | 3 |
| 1.3.8 | Erstellen des Dateisystems | 3 |
| 1.3.9 | NFS | 4 |
| 1.3.10 | uboot..... | 4 |
| 1.4 | Installation von Software für andere OSs | 4 |
| 1.4.1 | OpenICE | 4 |
| 1.4.2 | Web-Applikationen | 5 |
| 1.4.3 | Myriad Alien Dalvik | 5 |
| 1.5 | Fazit..... | 5 |
| 2 | CAN-Bus | 6 |
| 2.1 | Einführung | 6 |
| 2.2 | Physische Schicht..... | 6 |
| 2.3 | Logische Schicht | 6 |
| 2.4 | SocketCAN | 6 |
| 2.5 | FlexCAN | 6 |
| 2.6 | USB-CAN-Controller..... | 7 |
| 2.6.1 | PCAN-USB | 7 |
| 2.6.2 | Tiny-CAN I..... | 8 |
| 2.6.3 | Funktionstest | 8 |
| 2.7 | SPI-CAN-Controller | 9 |
| 2.8 | Paralleler Betrieb mit zwei OSs | 9 |
| 2.8.1 | Motivation..... | 9 |
| 2.8.2 | Übergabe..... | 10 |
| 2.8.3 | Sharing | 10 |
| 2.9 | Fazit..... | 10 |
| | Literatur..... | 11 |
| | Anhang | 12 |

Abbildungsverzeichnis

| | |
|------------------------------------|---|
| Abbildung 1: QSB..... | 2 |
| Abbildung 2: PCAN-USB..... | 7 |
| Abbildung 3: Tiny-CAN I..... | 8 |
| Abbildung 4: CAN-Testschleife..... | 9 |

Bildquellen 1-3: Hersteller

Tabellenverzeichnis

| | |
|---|---|
| Tabelle 1: Vergleich Meego/OpenICE..... | 5 |
|---|---|

1 Meego

1.1 Einführung

Meego ist ein Open Source-Betriebssystem für mobile Geräte und den Embedded-Bereich. Es entstand 2010 durch den Zusammenschluß von Moblin, das erstmals 2007 von Intel der Öffentlichkeit vorgestellt wurde, und Maemo, das seit 2005 von Nokia entwickelt wurde.

Es gliedert sich in das Core OS, das auf einem Linux-Kernel und dem rpm-Paketsystem basiert, und verschiedene GUIs (User Experiences, "UX"), von denen sich diese Arbeit mit der IVI UX beschäftigen wird. Diese wurde von der GENIVI Alliance, einem Zusammenschluß einiger Automobil- (u.a. BMW, GM, PSA Peugeot Citroën) und IT-Konzerne (u.a. Intel) als Standard-Plattform ausgewählt. Als GUI-Framework kommt Qt zum Einsatz.

1.2 Test auf einem PC

Um sich einen ersten Eindruck von Meego zu verschaffen, stehen auf meego.com .iso images für Intel-basierte PCs bereit, die auf eine CD oder einen USB-Stick geschrieben werden können. Mit der aktuellen Version 1.2 hatte ich auf meinen Systemen allerdings Probleme, ins GUI zu booten.

Bereits hier steht die 3D-Hardware-Beschleunigung nur mit Intel GMA500-GPUs und erst nach Installation eines Treibers zu Verfügung.

1.3 Erstellung des root filesystems und Booten auf dem QSB

1.3.1 i.MX53

Freescales i.MX53-Familie ist eine Reihe ARM-basierter System On Chip (SoC)-Modelle für die Bereiche Automotive, Consumer und Industrial.

1.3.2 QSB

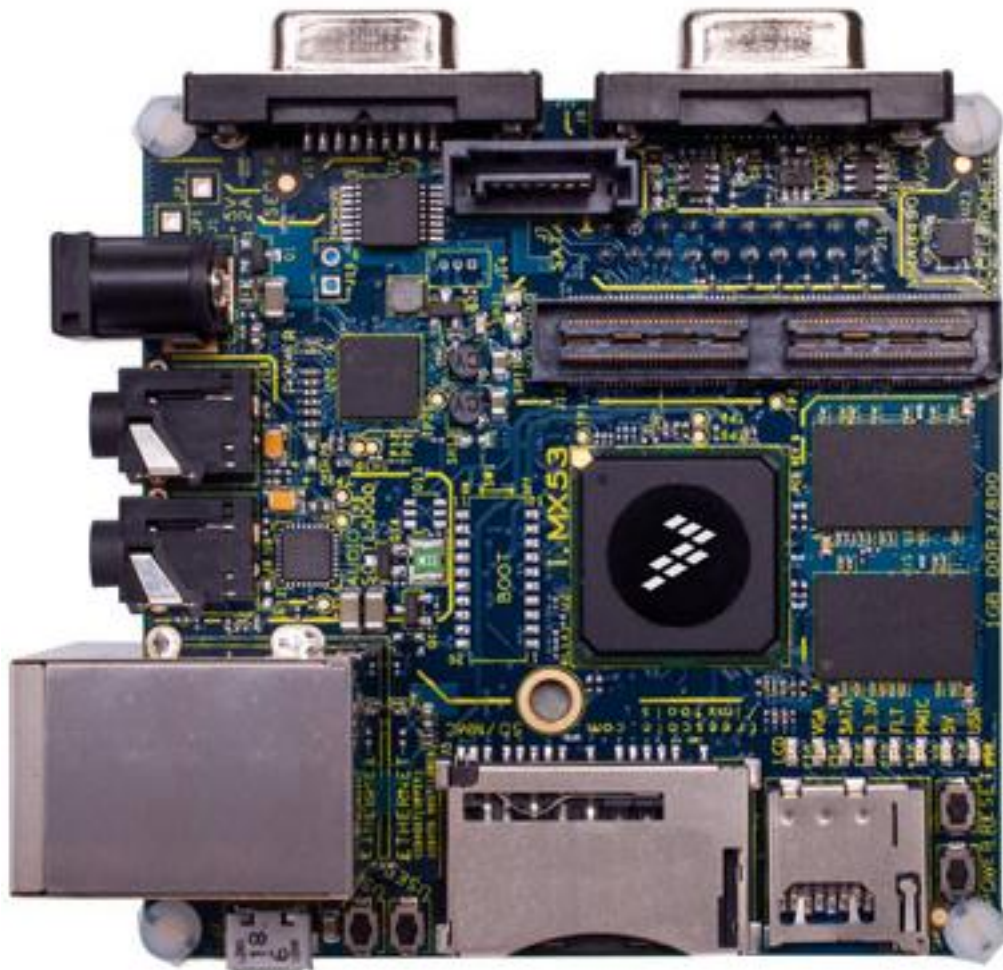


Abbildung 1: QSB

Das Quick Start Board (QSB) ist ein günstiges (\$149) evaluation board für den i.MX535. Unter anderem bietet es eine serielle Konsole, ein Ethernet interface, einen USB host und LVDS/HDMI.

1.3.3 MIC

Mit dem Meego Image Creator (MIC) kann man das root filesystem erstellen, das auf dem Zielgerät laufen soll.

Die folgende Anleitung bezieht sich auf Debian 6 als Host-System. Um die aktuellste Version (z. Zt. 0.24.14) zu installieren, ist eine zusätzliche Paketquelle hinzuzufügen:

```
/etc/apt/sources.list:
```

```
deb http://repo.meego.com/MeeGo/tools/repos/debian/5.0/ /  
apt-get install mic2
```

Fehler wie

```
Compiling /var/lib/python-support/python2.4/mic/appcreate/nand.py ...  
File "/var/lib/python-support/python2.4/mic/appcreate/nand.py", line  
283
```

```
finally:  
    ^
```

```
SyntaxError: invalid syntax
```

die bei der Paketinstallation auftreten können ignoriert werden.

1.3.4 Qemu

Außerdem sollte noch das Paket `qemu-user-static` manuell aus dem Repository installiert werden, zumindest im Hinblick auf die Unterstützung der aktuellen Architektur `armv7hl`:

```
wget "http://ftp.de.debian.org/debian/pool/main/q/qemu/qemu-user-
static_0.14.1+dfsg-3_i386.deb"
dpkg -i qemu-user-static_0.14.1+dfsg-3_i386.deb
```

1.3.5 Device mapper

Bei Einsatz eines eigenen Kernels ist darauf zu achten, das Kernel-Modul `dm-mod` zu kompilieren (`.config: CONFIG_BLK_DEV_DM=y|m`), zu installieren und mit

```
modprobe dm-mod
```

einzubinden.

1.3.6 Bootstrapping

Durch die Option `--build-bootstrap` wird nicht die installierte toolchain verwendet, sondern eine passende erstellt. Mit der Option `--bootstrap=` wird angegeben, wo sie abgelegt werden soll. Sie kann dann später mit Angabe dieser Option genutzt werden.

1.3.7 Zielort

`uboot` kann auf Dateisysteme auf einer μ SD-, SD-Card oder einem S-ATA-Gerät zugreifen, außerdem kann der Linux-Kernel das root filesystem über das Network File System (NFS) einbinden, was für die Entwicklung sehr praktisch ist und deshalb hier verwendet wird.

1.3.8 Erstellen des Dateisystems

Für die GPU-Treiber und das Kompilieren eigener Kernel-Module ist der Linux Target Image Builder (LTIB) erforderlich, es ist unter folgender URL erhältlich:

https://www.freescale.com/webapp/Download?colCode=L2.6.35_11.01_ER_SOURCE&prodCode=IMX53QSB&appType=license&location=null&fsp=1&Parent_nodeId=1298413274207729722207&Parent_pageType=product

Eine Übersicht über die enthaltene Dokumentation bietet [L2.6.35_11.01.00_ER_docs/readme.html](https://www.freescale.com/webapp/Download?colCode=L2.6.35_11.01_ER_docs/readme.html)

Die Installation von LTIB erfolgt nach dem dort aufgeführten Dokument „i.MX53 START Linux BSP User Guide“. Zwar werden die dabei entstehenden kernel und uboot images nicht benötigt, aber dieser Ablauf sorgt dafür, daß die Kernel-Quellen installiert werden.

Die Erstellung des Root-Filesystems wird durch eine sog. Kickstart-Datei (`.ks`) gesteuert.

Mit Version 1.2 wurde Meego auf die Architektur `armv7hl` umgestellt. Dadurch ändert sich die Übergabe von Fließkommazahlen beim Aufruf von Bibliotheksfunktionen. Da Freescale die GPU-Treiber wie leider branchenüblich nur binär bereitstellt, ist man so lange auf `armv7l` festgelegt, bis Freescale die Treiber für das neue ABI veröffentlicht. 1.1.9* ist die letzte Version, die die Architektur `armv7l` unterstützt. Für diese Version habe ich die Kickstart-Datei [ivi-armv7l-1.1.90.8-i.MX53QSB.ks](#) erstellt.

Das Erstellen des Dateisystems erfolgt mit

```
mic-image-creator --bootstrap=/meego/bootstrap1_2 --cache=x-cache --
format=fs --arch=armv7l --config=ivi-armv7l-1.1.90.8-i.MX53QSB.ks
```

Danach muß noch das von mir bereitgestellte Script [GPU.sh](#) in dem Verzeichnis, in dem sich auch die von mir bereitgestellte [xorg.conf](#) befindet, ausgeführt werden, um die Treiber für die GPU zu installieren. Davor sind darin die Variablen `LTIB`, (ggf. `LTIB_PGKS`) und `TARGET` anzupassen.

1.3.9 NFS

Auf dem System, auf dem das root filesystem liegt, ist folgendes einzurichten und den Gegebenheiten des LANs anzupassen:

/etc/exports:

```
/meego-ivi-armv7l-1.1.90.8-i.MX53QSB-1.1.90.20111003.2114
192.168.30.0/255.255.255.0(rw,no_root_squash,async,no_subtree_check)
```

Neustart des NFS-Servers:

```
/etc/init.d/nfs-kernel-server restart
```

Zum testen:

```
sudo mount.nfs 192.168.30.1:/meego/meego-ivi-armv7l-1.1.90.8-i.MX53QSB-
1.1.90.20111003.2114 /mnt -v
```

1.3.10 uboot

Im bootloader uboot können bestimmte Systemparameter im nichtflüchtigen Speicher eingestellt werden, vergleichbar mit dem BIOS eines PCs. Dazu ist innerhalb von drei Sekunden nach einem Reset an der seriellen Konsole (115.200 bps, 8N1, keine Flußkontrolle) eine Taste zu drücken.

Um das ggf. vorhandene HDMI daughter board anzusteuern, sind folgende Variablen anzupassen:

```
set hdmi 'video=mxcdi0fb:RGB24,1280x800M@60'
set bootargs_base 'setenv bootargs console=ttyMxc0,115200 ${hdmi}'
```

Folgende Einstellungen sorgen für die Einbindung des root-Filesystems über NFS:

```
set serverip 192.168.30.1
set nfsroot /meego/meego-ivi-armv7l-1.1.90.8-i.MX53QSB-
1.1.90.20111003.2114
```

Gespeichert werden die Variablen mit

```
saveenv
```

Das Booten erfolgt dann mit

```
run bootargs_base bootargs_nfs; mmc read 0 ${loadaddr} 0x800 0x1800;
bootm
```

Ich habe in der .ks-Datei dafür gesorgt, daß ein Login über die serielle Konsole möglich ist und daß beim ersten Systemstart der GPU-Treiber kompiliert wird.

Benutzername und Passwort sind „meego“.

Während des Bootens an meinem Monitor HP LP3065, der über einen HDMI/DVI-Adapter angeschlossen war, wird der Kernel durch den erfolglosen Versuch, die EDID-Daten vom Monitor auszulesen (Fehlermeldung: sii9022 1-0039: SII9022: read edid fail) derart blockiert, daß sich der Bootvorgang verzögerte, DHCP-Verzögerungen (NETDEV WATCHDOG: eth0 (fec): transmit queue 0 timed out) auftraten und die Erkennung des integrierten Soundchips nicht funktionierte (sgtl5000_hw_read: read reg error : Reg 0x00).

1.4 Installation von Software für andere OSs

Nach dem Austritt von Nokia aus der Meego-Weiterentwicklung hat Intel Pläne veröffentlicht, Meego 2012 durch Tizen abzulösen. Nicht nur im Hinblick darauf sind Möglichkeiten interessant, Programme für andere Betriebssysteme unter Meego zu installieren.

1.4.1 OpenICE

OpenICE ist ein Ubuntu-basiertes OS für den Automotive-Bereich, das von einer Community gepflegt wird.

Tabelle 1: Vergleich Meego/OpenICE

| | App-Frame-Work | Message Bus | Medien-Wiedergabe | Navigation | OBD II | Freisprechen | Erweiterungen |
|---------|-------------------------------|-------------|-------------------|------------|------------------------------|--------------|---------------------------|
| Meego | Qt | D-Bus | GStreamer | navit | - | oFono | Meego Apps |
| OpenICE | nGhost (SDL, XML) ICEPanel | | mplayer | | carman, nobd, NCarInfo | nohands | Applikationen, Plugins |

Im Standard-Meego-Repository nicht vorhandene Paket-Abhängigkeiten bestehen in libtag1, libxml++ und libltdl3. Diese müßten auf jeden Fall gesondert portiert werden.

1.4.2 Web-Applikationen

Als webbasiertes Framework stellt QtWRT einen zukunftssicheren Portierungspfad dar, da Tizen verstärkt auf Webtechnologien setzen wird.

1.4.3 Myriad Alien Dalvik

Von der Myriad Group AG stammt Alien Dalvik, womit es möglich sein soll, Android-Apps unverändert unter Meego laufen zu lassen.

1.5 Fazit

Durch mic-image-creator kann das root filesystem relativ komfortabel erstellt werden. Ein Problem stellt momentan noch die Tatsache dar, daß IVI eigentlich für Touchscreens vorgesehen und deshalb bei Einsatz einer Maus deren Zeiger nicht sichtbar ist. Für den Umstieg auf die aktuelle Version 1.2 ist hinderlich, daß die immer wichtiger werdenden GPU-Treiber von den Herstellern nach wie vor nur binär bereitgestellt werden.

2 CAN-Bus

2.1 Einführung

Das Controller Area Network (CAN) ist ein vornehmlich im Automobil-Bereich eingesetztes Bussystem. Es wurde 1986 von der Robert Bosch GmbH entwickelt.

In den Fahrzeugen der FH Trier soll über das IVI die Fahrzeugelektronik, die über den CAN-Bus erreichbar ist, zugänglich gemacht werden.

2.2 Physische Schicht

Um Gleichtaktstörungen zu unterdrücken, erfolgt die Datenübertragung symmetrisch. Die Busteilnehmer werden parallel am Bus angeschlossen und über eindeutige Identifizierer angesprochen. Kollisionen zwischen ihnen werden mit dem Carrier Sense Multiple Access/Collision Resolution (CSMA/CR)-Verfahren aufgelöst. Liegen auf den Datenleitungen unterschiedliche Pegel, entspricht dies einem „0“-Bit. Dieses ist dominant und überschreibt ggf. gleichzeitig vorhandene rezessive „1“-Bits.

2.3 Logische Schicht

Daten-Frames beginnen mit einem Identifizierer-Segment (CAN 2.0A/B: 11/29 Bit) und haben eine Nutzlast von bis zu 8 Bytes.

2.4 SocketCAN

Um unter Unix ein Standard-API für die Kommunikation zu schaffen, initiierte die VW AG das Open-Source-Projekt SocketCAN. Damit können Busteilnehmer ähnlich wie TCP/IP-Geräte über Sockets angesprochen werden. Diese Sockets werden mit „AF_CAN“ als Protokoll-Familien-Parameter geöffnet.

2.5 FlexCAN

Die FH Trier hat besonderes Interesse an der i.MX53-Familie, da einige Modelle zwei CAN-Controller beinhalten. Dabei handelt es sich um Freescales FlexCAN, einem FullCAN-Controller mit einem FIFO und 64 Mailboxen, die in embedded RAM liegen.

Der SocketCAN-Treiber ist quelloffen und besteht aus 4 Dateien im Verzeichnis `drivers/net/can/flexcan`:

`dev.c`: Parameter-Handling

`drv.c`: Netzwerk-Treiber

`mbm.c`: Message buffer management

`flexcan.h`: Header-Datei

Zu Beginn meiner Arbeit stand das Reference Manual noch nicht zu Verfügung, so daß ich mich an dem für den Vorgänger i.MX35 orientiert habe. Zum Ende der Arbeit hin erschien dann Rev. 1A, deren Kapitel über den FlexCAN-Controller allerdings eine schlechte Qualität aufwies. Ich habe folgende Fehler an Freescale gemeldet:

34.1.2: "Mbytes" should be "MB" for "Message Buffers" 2x

34.2.2.1/2: logic levels missing

34.3: Memory map and register definitions completely absent

34.3.2: Was the meaning of IDE really flipped in regard to iMX35RM.pdf?

Table 34-3: SRR called SR, IDE called ID, RTR called RT, PRIO missing

Table 34-4: Missing OVERRUN cross references 2x

34.3.3: Was the meaning of REM and EXT really flipped in regard to iMX35RM.pdf?

34.4.7: "Mbytes" should be "MB" for "Message Buffers"

34.5.1: Missing Freeze Mode cross reference

2.6 USB-CAN-Controller

Da beim Quick Start Board kein CAN-Controller enthalten ist, habe ich folgende USB-CAN-Controller verwendet:

2.6.1 PCAN-USB



Abbildung 2: PCAN-USB

[http://www.peak-system.com/Produktdetails.49+M5f1e2159727.0.html?&tx_commerce_pi1\[catUid\]=6&tx_commerce_pi1\[showUid\]=16](http://www.peak-system.com/Produktdetails.49+M5f1e2159727.0.html?&tx_commerce_pi1[catUid]=6&tx_commerce_pi1[showUid]=16)

Hersteller: PEAK-System Technik GmbH, Darmstadt

Preis: ab 195 €

Quelltext des SocketCAN-Treibers verfügbar

Cross compilation für das QSB:

```
cd /peak-linux-driver-7.3/driver
make -j2 ARCH=arm CROSS_COMPILE=/opt/freescale/usr/local/gcc-4.4.4-
glibc-2.11.1-multilib-1.0/arm-fsl-linux-gnueabi/bin/arm-none-linux-
gnueabi- KERNEL_LOCATION=/ltib/rpm/BUILD/linux-2.6.35.3 PCI=NO DNG=NO
ISA=NO PCC=NO
```

Start auf dem QSB:

```
./pcan_make_devices 2
insmod pcan.ko bitrate=0x0014
cat /proc/pcan
ifconfig can0 up
```

2.6.2 Tiny-CAN I

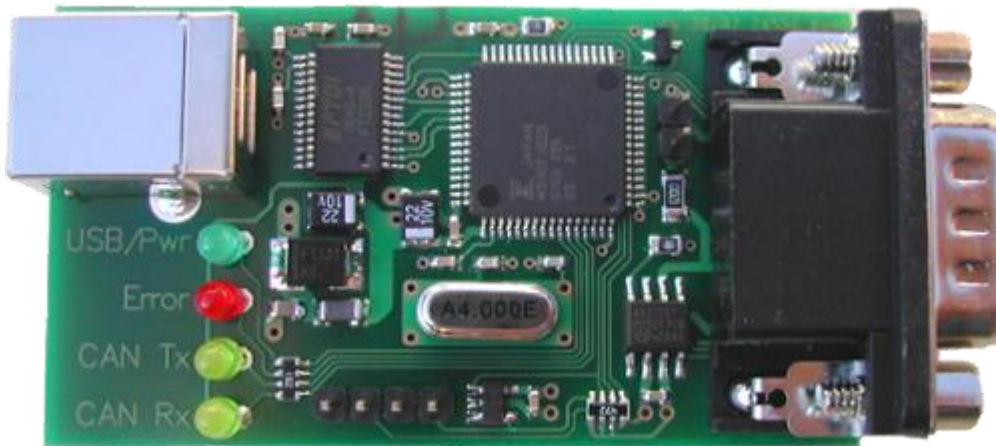


Abbildung 3: Tiny-CAN I

<http://www.mhs-elektronik.de/cgi-bin/mhs.pl?id1=2&id2=1>

Hersteller: MHS GmbH & Co. Elektronik KG, Kößlarn

Preis: 59 €

Ursprüngliche Entwicklung: Dr. Högl, HS Augsburg:

<http://www.hs-augsburg.de/~hhoegl/proj/usb-tiny-can>

Ich habe diesen CAN-Controller als günstige Test-Gegenstelle zur Verwendung unter Windows besorgt. Ein Betrieb mit Socket-CAN wäre evtl. möglich mit

<http://svn.berlios.de/viewvc/socketcan/trunk/can-utils/slcard.c?view=log>

2.6.3 Funktionstest

Um die Kommunikation zu überprüfen, habe ich beide CAN-Controller mit einem terminierten (120 Ω) Kabel verbunden und dann auf dem QSB den bei SocketCAN enthaltenen Paket-Generator und das -Protokoll gestartet:

```
svn co http://svn.berlios.de/svnroot/repos/socketcan/trunk socketcan
cd socketcan/can-utils
make
./cangen can0 &
./candump can0
```

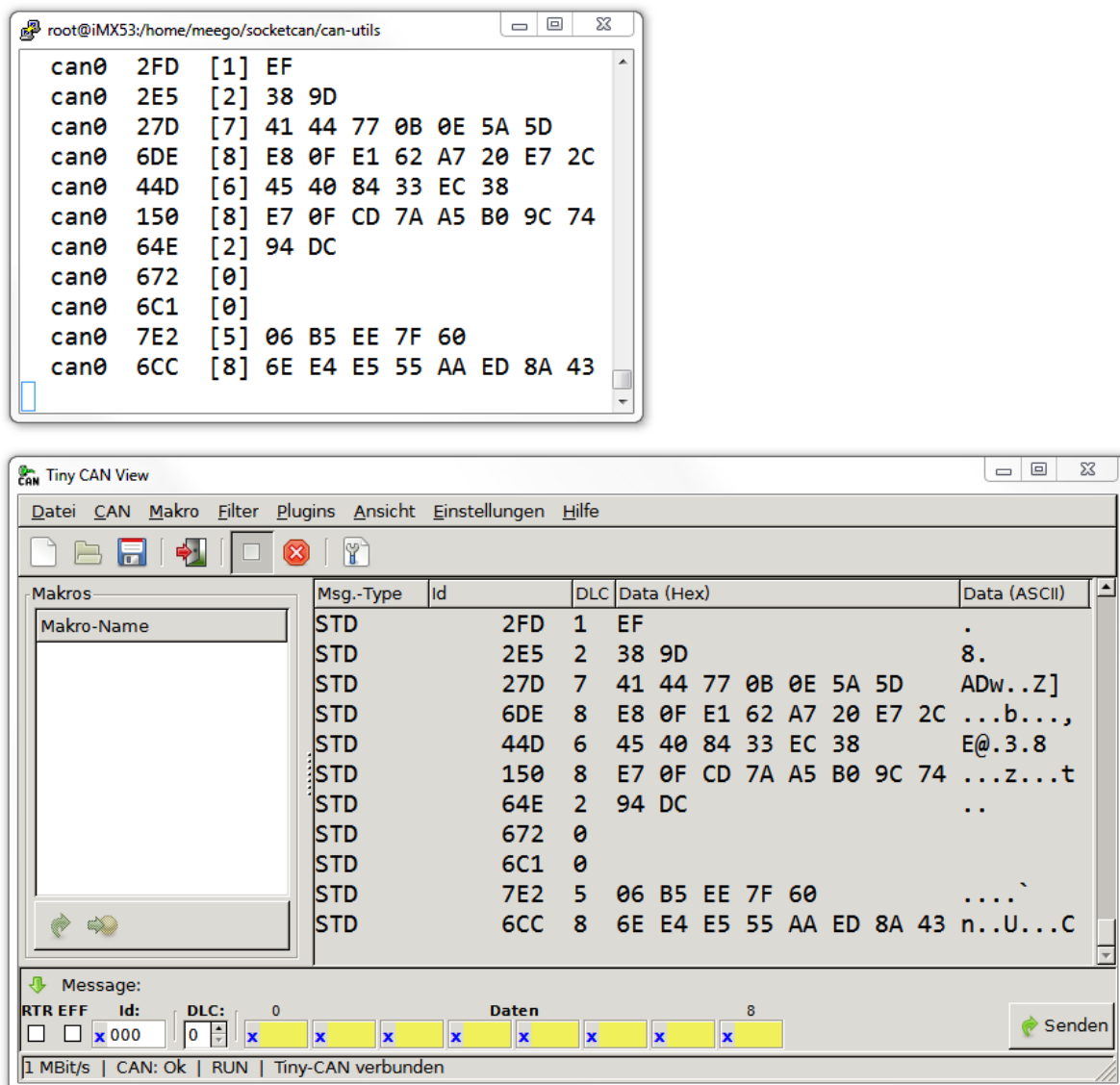


Abbildung 4: CAN-Testschleife

2.7 SPI-CAN-Controller

Eine mögliche Alternative wäre ein OMAP3-basiertes development board mit per SPI angebundenem CAN-Controller auf einem Expansion Board:

http://shop.igep.es/index.php?main_page=product_info&cPath=1&products_id=49

Hersteller: ISEE, Sant Cugat del Valles, Spanien

SocketCAN-Treiber sind vorhanden.

Preis: ab 160+199=359 €

2.8 Paralleler Betrieb mit zwei OSs

2.8.1 Motivation

Im Automotive-Bereich gibt es die Anforderung, daß CAN-Geräte maximal 50 ms nach Stromzufuhr zur Kommunikation bereit sein müssen. Das läßt sich mit einem Linux-Kernel

nicht realisieren, sehr wohl aber mit einem vorher gestarteten Automotive Open System Architecture (AUTOSAR)-Realtime-OS.

2.8.2 Übergabe

Das einfachere Szenario wäre eine Übergabe des CAN-Controllers von AUTOSAR an den Linux-Kernel, nachdem die zeitkritische Kommunikation abgewickelt wurde.

2.8.3 Sharing

Im Falle eines gemeinsamen Zugriffs wurden folgende Problemfelder identifiziert:

- Bus-Parameter
Grundlegende Parameter des CAN-Busses wie die Geschwindigkeit müssten von AUTOSAR gesetzt werden und dürften später von Linux nicht mehr verändert werden.
- Fehlerbehandlung
Die Fehlerbits werden beim Lesen gelöscht und können nicht beschrieben werden.
- IRQs
Die Verteilung an die korrekte Interrupt Service Routine (ISR) müsste sichergestellt werden.
- FIFO
Der First In First Out (FIFO)-Puffer würde besser nur von einem OS verwendet werden.
- Puffer
Auch wenn man den Puffer in der Mitte aufteilen würde, ergäbe sich eine Asymmetrie durch die Register RX14MASK und RX15MASK.

Eine mögliche Lösung wäre eine Virtualisierung des CAN-Controllers. Ein Mechanismus für die Zugriffssteuerung ist in [Sch11] beschrieben. Unterstützung dafür könnte die ARM Trust-Zone liefern [Fre10], deren eigentlicher Zweck es ist, sicherheitsrelevante Routinen vom Rest des Systems zu trennen. ARM arbeitet an einer diesbezüglichen Erweiterung der Referenz:

http://infocenter.arm.com/help/topic/com.arm.doc.ddi0406b_virtualization_extns/index.html

Der Performance-Verlust scheint auch vertretbar zu sein [Bra08].

2.9 Fazit

Auch ohne integrierten Controller ist eine Kommunikation über den CAN-Bus mit dem QSB mit Hilfe eines USB-CAN-Controllers möglich.

Der parallele Zugriff auf einen CAN-Controller von zwei OSs bedarf einer genauen Abstimmung oder einer Virtualisierung.

Literatur

- Sch11. Schneider, Jörn: *A WCET Directed Operating System Service to address Industrial Challenges in Mixed-Criticality Systems*. Eingereicht bei SIES 2011.
- Fre10. Frenzel, Torsten; Lackorzynski, Adam; Warg, Alexander; Härtig, Hermann: *ARM TrustZone as a Virtualization Technique in Embedded Systems*. Twelfth Real-Time Linux Workshop 2010, Nairobi, Kenya, October 2010. http://os.inf.tu-dresden.de/papers_ps/rtlws2010_armtrustzone.pdf
- Bra08. Brakensiek, Jörg; Dröge, Axel; Botteck, Martin; Härtig, Hermann; Lackorzynski, Adam: *Virtualization as an enabler for security in mobile devices*. IIES '08: Proceedings of the 1st workshop on Isolation and integration in embedded systems, Pages 17-22, ISBN 978-1-60558-126-2, Glasgow, Scotland, April 2008.

Anhang

Auf der beigelegten CD-R befindet sich neben den angesprochenen Dateien ein tarball mit dem root filesystem (1,4 GB). Darin habe ich einige zusätzliche Verzeichnisse installiert:

/home/meego

/pcan: Script und Kernel-Modul für PCAN-USB

/socketcan/can-utils: SocketCAN-Werkzeuge

/GPUSDK/bin: Auf dem Target kompilierter OpenGL ES 1.1-Test des Archivs gpu-sdk1008.tgz von der Freescale-Website (eindrucksvoller: bbPinball)