Hacking into TomTom GO



Thomas Kleffel, Christian Daniel
und das www.opentom.org-Projekt

Hacking into TomTom GO

Vortrags-Abschnitte

Geschichte des Hacks

Überblick über die Hardware

Umgang mit Treibern im GO-Kernel

GO im Originalzustand...



Navigation– je nach Version durch ganz Europa

G500, G700 und RIDER arbeiten als Bluetooth-Freisprecheinrichtung

Alle Modelle exportieren ihren Speicher als USB-Storage (GO = USB device)

Erweiterungsmöglichkeiten via SDK

OS ist von Haus aus Linux

Möglichkeiten...

MP3-Player

Video-Player

Kopierstation z.B. von Digicam

Rückfahrkamera

Spielkonsole

Wardriving mit WLAN, GPS und Straßen-Karte...

OpenEmbedded, Qtopia, GPE, ...











Reverse Engineering

Linux & GPL

Für Systeme unter Linux sollte Reverse Engineering *eigentlich* nicht nötig sein.

Die GPL verpflichtet den Hersteller zur Veröffentlichung aller zur Entwicklung eigener Funktionalität notwendigen Dinge.

- Quellen für Kernel & GNU-Tools
- Toolchain (Compiler, Libs, ...)
- Systemsspezifische Build-Tools

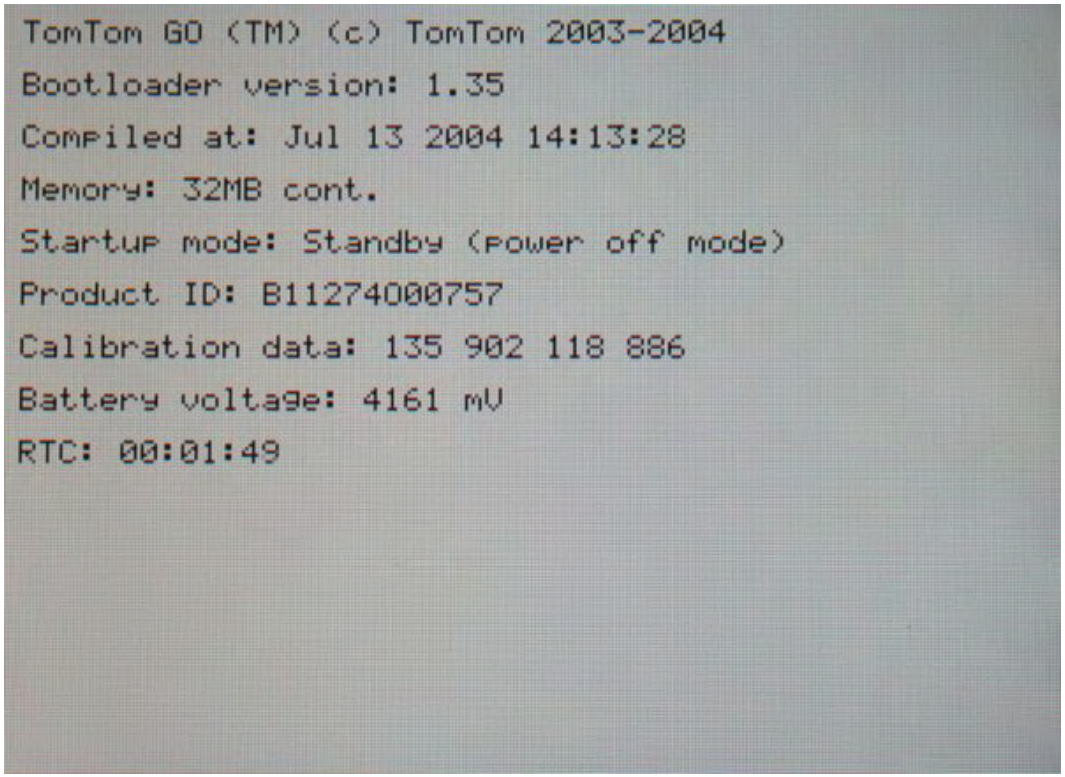
Die Realität zeigt, daß Hersteller diese Prinzipien oft ignorieren... http://www.gpl-violations.org hilft hier gerne etwas aus :-)

Reverse Engineering

Erste Schritte

Durch einfaches „Herumspielen“ konnten dem System schon wertvolle Informationen abgerungen werden:

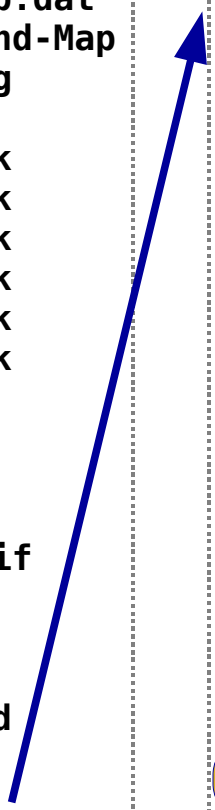
- Bootloader (Taste lange drücken!)
- Debug-Tools / JTAG
- Fehlermeldungen
- Abstürze



Reverse Engineering

Analyse des Dateisystems

```
     33 Jan  1  2004 CurrentMap.dat
   2048 Oct 13 22:36 Deutschland-Map
   2048 Oct 13 22:36 TomTom-Cfg
 773676 Sep  3 16:04 data.chk
 794756 Mar 24  2004 data01.chk
 826460 Mar 24  2004 data02.chk
 683093 Mar 24  2004 data05.chk
 670562 Mar 24  2004 data12.chk
 735281 Apr 12  2004 data13.chk
 712656 Apr 11  2004 data15.chk
     28 Mar 23  2004 deuf.vif
     30 Mar 23  2004 deum.vif
     33 Mar 23  2004 engf.vif
     34 Mar 23  2004 fraf.vif
     37 Jun 10 12:33 install.bif
     30 Mar 23  2004 ita.vif
     35 Mar 23  2004 nldf.vif
 267536 Sep  3 16:04 system
      4 Jun  7 18:08 tomtom.aid
    220 Jan  1  2004 ttgo.bif
1908942 Sep  3 16:04 ttsystem
```

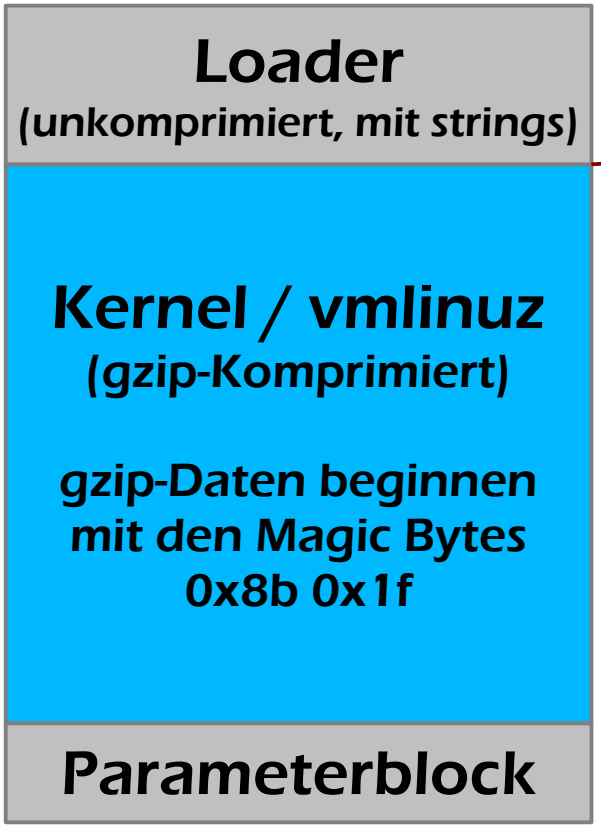


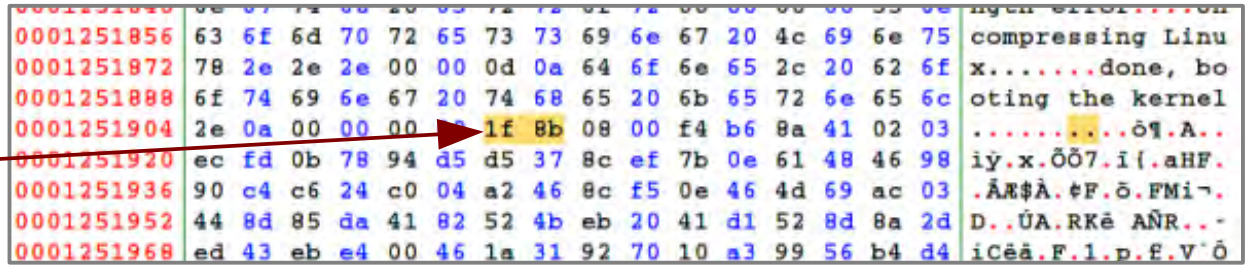
```
# strings ttsystem
[...]
ran out of input data
Malloc error
Memory error
Out of memory
incomplete literal tree
incomplete distance tree
bad gzip magic numbers
internal error, invalid method
Input is encrypted
Multi part input
Input has invalid flags
invalid compressed format (err=1)
invalid compressed format (err=2)
out of memory
invalid compressed format (other)
crc error
length error
Uncompressing Linux...
done, booting the kernel.
[...]
```

Reverse Engineering

Extraktion des Kernels aus ttsystem

zImage:





Alles ab den Magic Bytes wurde in eine neue Datei kopiert und mit gunzip entpackt.

Damit stand der Kernel als vmlinux zur weiteren Analyse zur Verfügung.

Reverse Engineering

Analyse des Kernels mit „strings“ ergab (auszugsweise):

```
root=/dev/ram0 console=ttyS0
Linux version 2.4.18-rmk6-swl5-tt201 (aya@achilles.intra.local)
<3>initrd (0x%08lx - 0x%08lx) extends beyond physical memory - disabling initrd
EXT2-fs: unable to read group descriptors
FAT: logical sector size too small for device (logical sector size = %d)
u.v.m.
```

Konsole auf seriellem Port

Init-Ramdisk

Linux 2.4

FAT für SD-Karten

Ext2 für Init-Ramdisk

GPL-Violation nachgewiesen...

- Die Benutzung von GPL-Code (Linux Kernel) war hiermit hinreichend nachgewiesen.
- Der Hersteller konnte nachdrücklich auf den GPL-Verstoß hingwiesen werden.
- Anhand der Texte im Kernel kann die Verwendung einzelner Features nachgewiesen werden.
- Nachdem TomTom eher träge reagierte, kam es zur Abmahnung eines großen Händlers in Deutschland.
- Nachdem die ganze Juristerei doch etwas Zeit brauchte und bei uns die Neugier nagte, ging es weiter...

Reverse Engineering

Extraktion der Init-Ramdisk

Die Init-Ramdisk wird vom Bootloader in den Speicher geladen und die Startadresse dem Kernel übergeben.

Der Kernel sucht nach Filesystem- oder gzip-Magics.

ttsystem:

```
0000000000  54 54 42 4c 8e e5 12 00 00 00 00 31 1f 8b 08 00  TTBL.å.....1....
0000000016  29 78 65 41 02 03 ec 9d 09 78 14 45 de c6 6b 66  )xeA..ì..x.EÞÆkf
0000000032  42 12 20 42 44 54 3e d1 35 5e c8 19 50 3c 40 50  B. BDT>Ñ5^È.P<@P
0000000048  12 0e 01 4f 14 fd d4 55 d7 1c 33 24 91 24 13 33  ...O.ýÔU×.3$.$.3
```

Offensichtlich befand sich am Anfang von ttsystem ein weiterer gzip-komprimierter Block – dieser enthielt ein ext2-Dateisystem, die Init-Ramdisk.

Reverse Engineering

Analyse der Init-Ramdisk

- Die Init-Ramdisk enthält ein Mini-Linux-System, basierend auf BusyBox.
- Auch die Navigations-Software (/bin/ttn) ist in der Init-Ramdisk enthalten.
- Das Init-Script (/etc/rc) sah einen Debug-Modus mit Shell vor!

```
echo "* Starting ${product}"
if test -f ${debugf}
then
  ${ttnapp} > /dev/console 2>&1 &
  echo "* Starting shell"
  sh
else
  ${ttnapp} > /dev/null 2>&1 &
fi
```

- Die Konsole wird im Kernel auf ttyS0 gesetzt.
- Wir vermuteten also, daß ttyS0 von außen zugänglich sein mußte...

Reverse Engineering

Ein genauer Blick auf die Hardware...



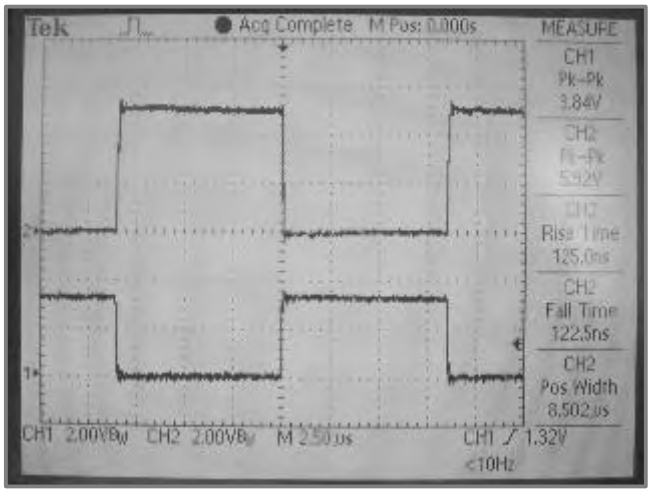


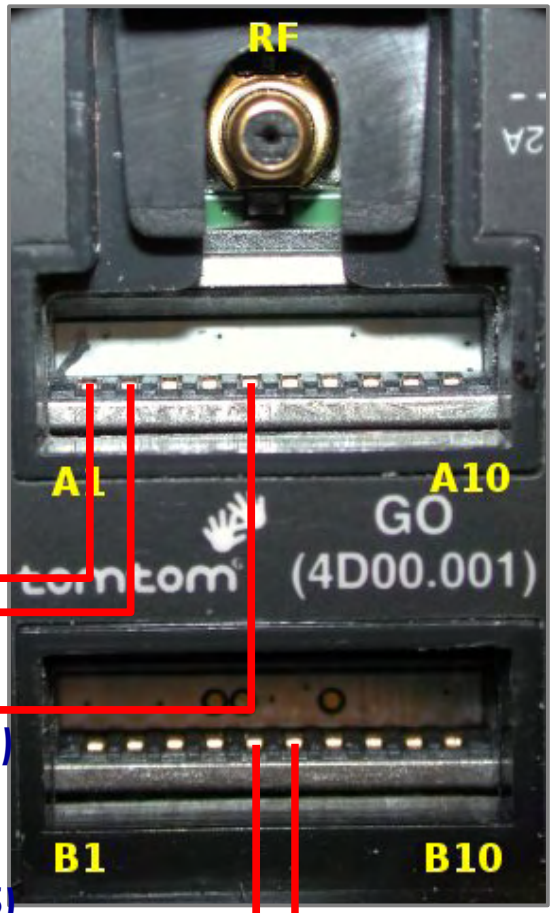
- 5V Stromversorgung
- USB
  (Host/Device in Software(!) umschaltbar)
- Externe GPS-Antenne
- 2x10 Pins für Docking-Station & Carkit

Reverse Engineering

Ein genauer Blick auf die Hardware...

```
Uncompressing
Linux.............................
.... done, booting the kernel.
Linux version 2.4.18-rmk6-swl5-
tt200 (aya@achilles.intra.local)
(gcc version 2.95.2 19991024
(release)) #1 Fri Sep 3 15:03:17
CEST 2004
CPU: SAMSUNG
S3C2410(Arm920T)sid(wb) revision 0
Machine: SAMSUNG ELECTRONICS Co.,
Ltd
On node 0 totalpages: 8192
zone(0): 8192 pages.
zone(1): 0 pages.
zone(2): 0 pages.
TomTom Go Boot, built at 2004-09-
03 15:04:15
* Mounting /proc
* Waiting for SD card to mount
```





Pegelkonverter (3V <-> RS232)

TxD (A1)

RxD (A2)

Ground (A5)



Left audio out (B5)

Right audio out (B6)

Reverse Engineering

Image-Format von ttsystem



- Veränderung der Init-Ramdisk und Einbau in ein bestehendes ttsystem
- Anpassung des Längen-Parameters
- Image wurde vom TTBL nicht akzeptiert
- Was bedeuten die mysteriösen 16 Byte nach Kernel und Init-Ramdisk???
- Trial-and-Error schlugen fehl (kein MD4, MD5 oder ähnliches)
- ...TomTom machte einen Fehler: Ein Bootloader-Update

Reverse Engineering

Die Image-Signaturen

Um den noch bestehenden Verdacht auf MD5 zu prüfen, suchten wir nach typischen Konstanten:

md5.c:

```
MD5STEP (F1, a, b, c, d, in[0] + 0xd76aa478, 7);
MD5STEP (F1, d, a, b, c, in[1] + 0xe8c7b756, 12);
MD5STEP (F1, c, d, a, b, in[2] + 0x242070db, 17);
MD5STEP (F1, b, c, d, a, in[3] + 0xc1bdceee, 22);
MD5STEP (F1, a, b, c, d, in[4] + 0xf57c0faf, 7);
MD5STEP (F1, d, a, b, c, in[5] + 0x4787c62a, 12);
MD5STEP (F1, c, d, a, b, in[6] + 0xa8304613, 17);
MD5STEP (F1, b, c, d, a, in[7] + 0xfd469501, 22);
MD5STEP (F1, a, b, c, d, in[8] + 0x698098d8, 7);
MD5STEP (F1, d, a, b, c, in[9] + 0x8b44f7af, 12);
MD5STEP (F1, c, d, a, b, in[10] + 0xffff5bb1, 17);
MD5STEP (F1, b, c, d, a, in[11] + 0x895cd7be, 22);
MD5STEP (F1, a, b, c, d, in[12] + 0x6b901122, 7);
MD5STEP (F1, d, a, b, c, in[13] + 0xfd987193, 12);
MD5ST
MD5ST
```

```
0000047328  1e ff 2f e1 78 a4 6a d7 56 b7 c7 e8 db 70 20 24  .ÿ/áx¤j×V·ÇèÛp $
0000047344  ee ce bd c1 51 f0 83 0a 2a c6 87 47 13 46 30 a8  îÎ½ÁQð..*Æ.G.F0¨
0000047360  01 95 46 fd d8 98 80 69 51 08 bb 74 42 28 a3 76  ..FýØ..iQ.»tB(£v
0000047376  22 11 90 6b 6d 8e 67 02 72 bc 86 59 21 08 b4 49  "..km.g.r¼.Y!.´I
```

- Der Bootloader enthält offensichtlich eine MD5-Implementation
- Der Hash stimmte trotzdem nicht – MD5 ist also nicht ausreichend

Reverse Engineering

Die Image-Signaturen

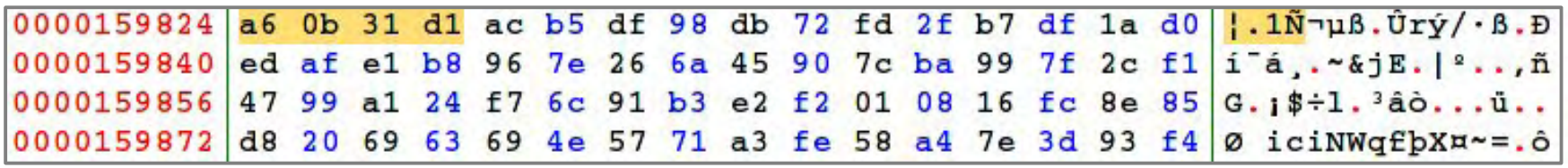
Was einmal gut geklappt hat, funktioniert auch ein zweites Mal: Wir suchten also nach Konstanten für Crypto-Algorithmen...

...und wurden bei Blowfish fündig!

blowfish.c:

```
/* precomputed S boxes */
static const u32 ks0[256] = {

0xD1310BA6,0x98DFB5AC,0x2FFD72DB,0xD01ADFB7,0xB8E1AFED,0x6A267E96,
0xBA7C9045,0xF12C7F99,0x24A19947,0xB3916CF7,0x0801F2E2,0x858EFC16,
0x636920D8,0x71574E69,0xA458FEA3,0xF4933D7E,0x0D95748F,0x728EB658,
0x718BCD58,0x82154AEE,0x7B54A41D,0xC25A59B5,0x9C30D539,0x2AF26013,
[...]
```



Reverse Engineering

Die Image-Signaturen

Die weitergehende Analyse des Bootloaders mittels Disassembler führt zum Blowfish-Key:

```
d8 88 d3 13 ed 83 ba ad 9c f4 1b 50 b3 43 fa dd
```

Mit diesem Schlüssel signierte Images werden vom TomTom-Bootloader akzeptiert.

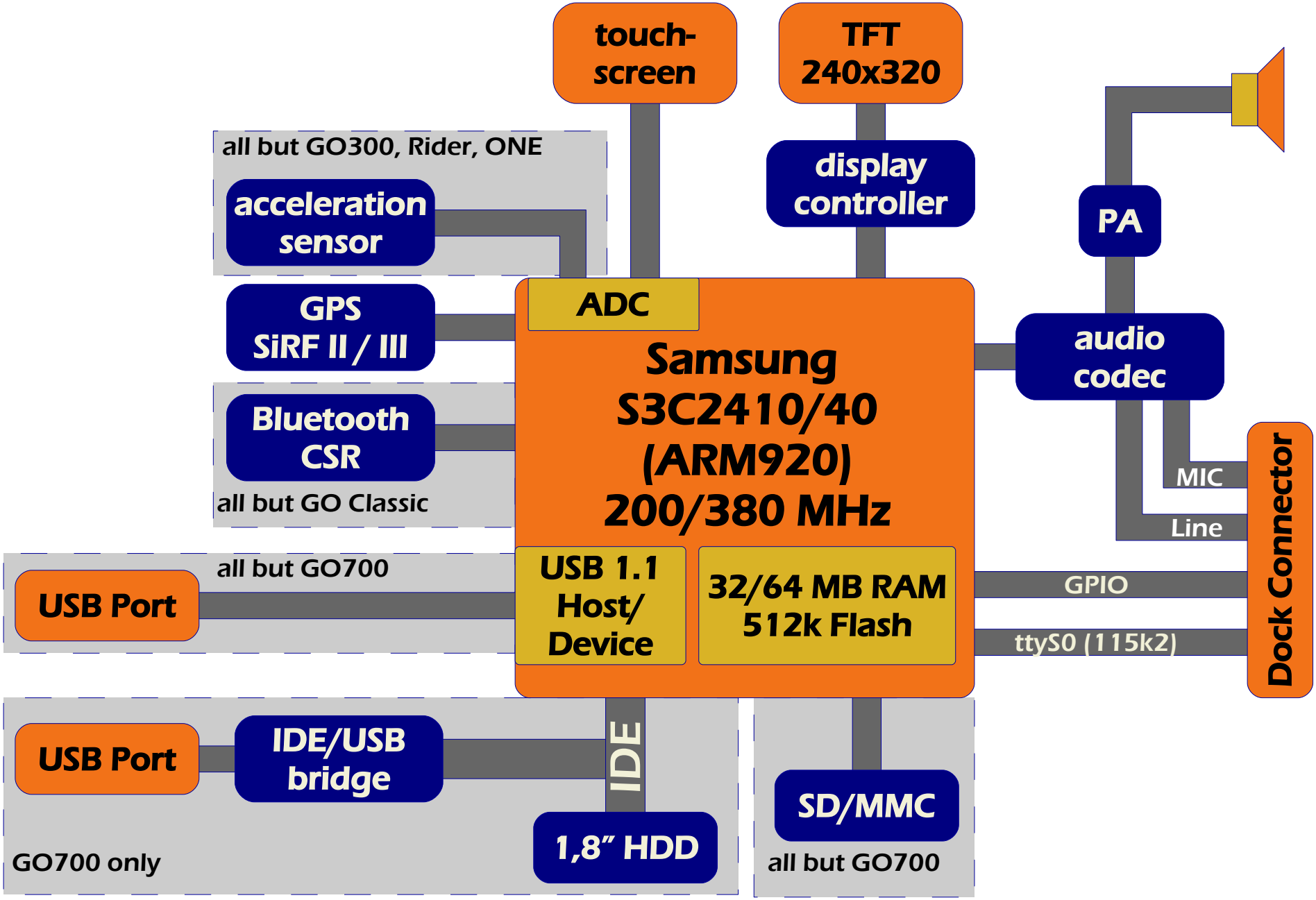


Hardware

Modell-Tabelle

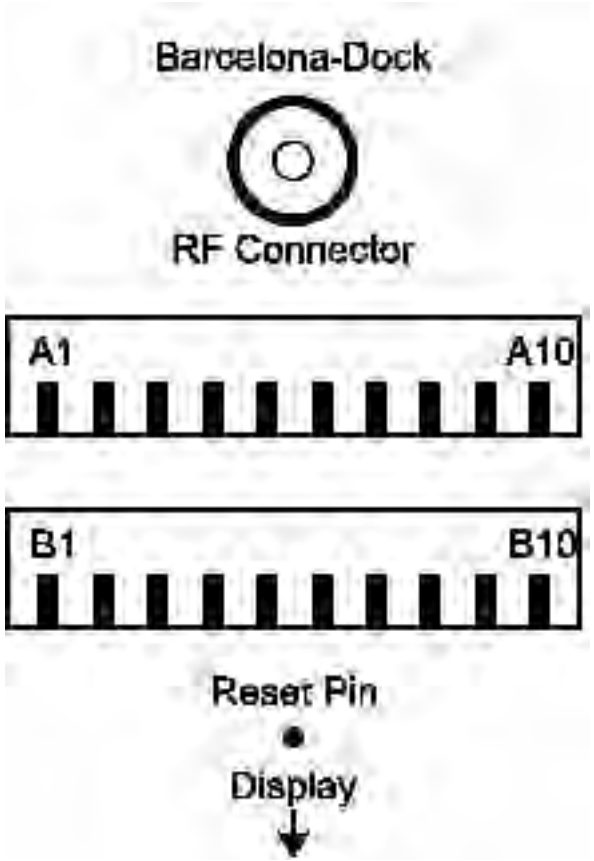
	RAM	Storage	BT	Sound	USB-Host	G-Sense	RC RX
Classic	32MB	SD-Card	–	DA	möglich	vorh.	–
ONE	32MB	SD-Card	ja	DA	möglich	–	–
GO 300	32MB	SD-Card	ja	DA	möglich	–	–
GO 500	32MB	SD-Card	ja	AD/DA	möglich	vorh.	vorh.
GO 700	64MB	HDD	ja	AD/DA	–	vorh.	vorh.
RIDER	64MB	SD-Card	ja	–	möglich	–	–

Hardware


touch-screen
TFT 240x320
all but GO300, Rider, ONE
acceleration sensor
display controller
PA
GPS SiRF II / III
ADC
Samsung S3C2410/40 (ARM920) 200/380 MHz
audio codec
Bluetooth CSR
all but GO Classic
MIC
Line
Dock Connector
all but GO700
USB Port
USB 1.1 Host/ Device
32/64 MB RAM 512k Flash
GPIO
ttyS0 (115k2)
USB Port
IDE/USB bridge
IDE
SD/MMC
1,8" HDD
GO700 only
all but GO700

Hardware

Dock-Port - Classic-Modell

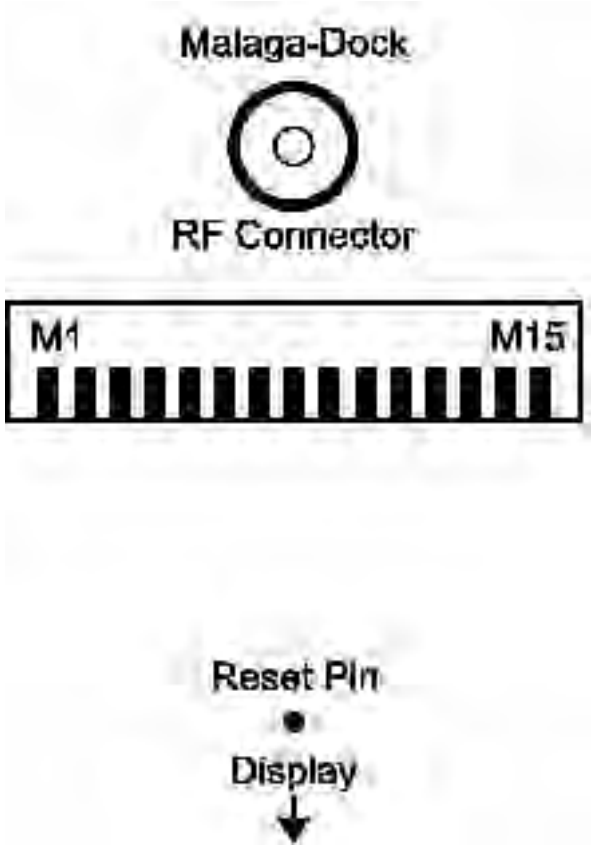


Pin	Signal	Pin	Signal
A1	ttyS0 TxD (3.3V)	B1	Signal ground
A2	ttyS0 RxD (3.3V)	B2	Signal ground
A3	ttyS0 RtS (3.3V)	B3	Dock sense
A4	ttyS0 CtS (3.3V)	B4	Dock sense
A5	Signal ground	B5	Line-Out left
A6	JTAG RST	B6	Line-Out right
A7	JTAG TMS	B7	Car-radio mute
A8	JTAG TCK	B8	Ignition sense
A9	JTAG TDI	B9	+5V from dock
A10	JTAG TDO	B10	+5V from dock

Hardware

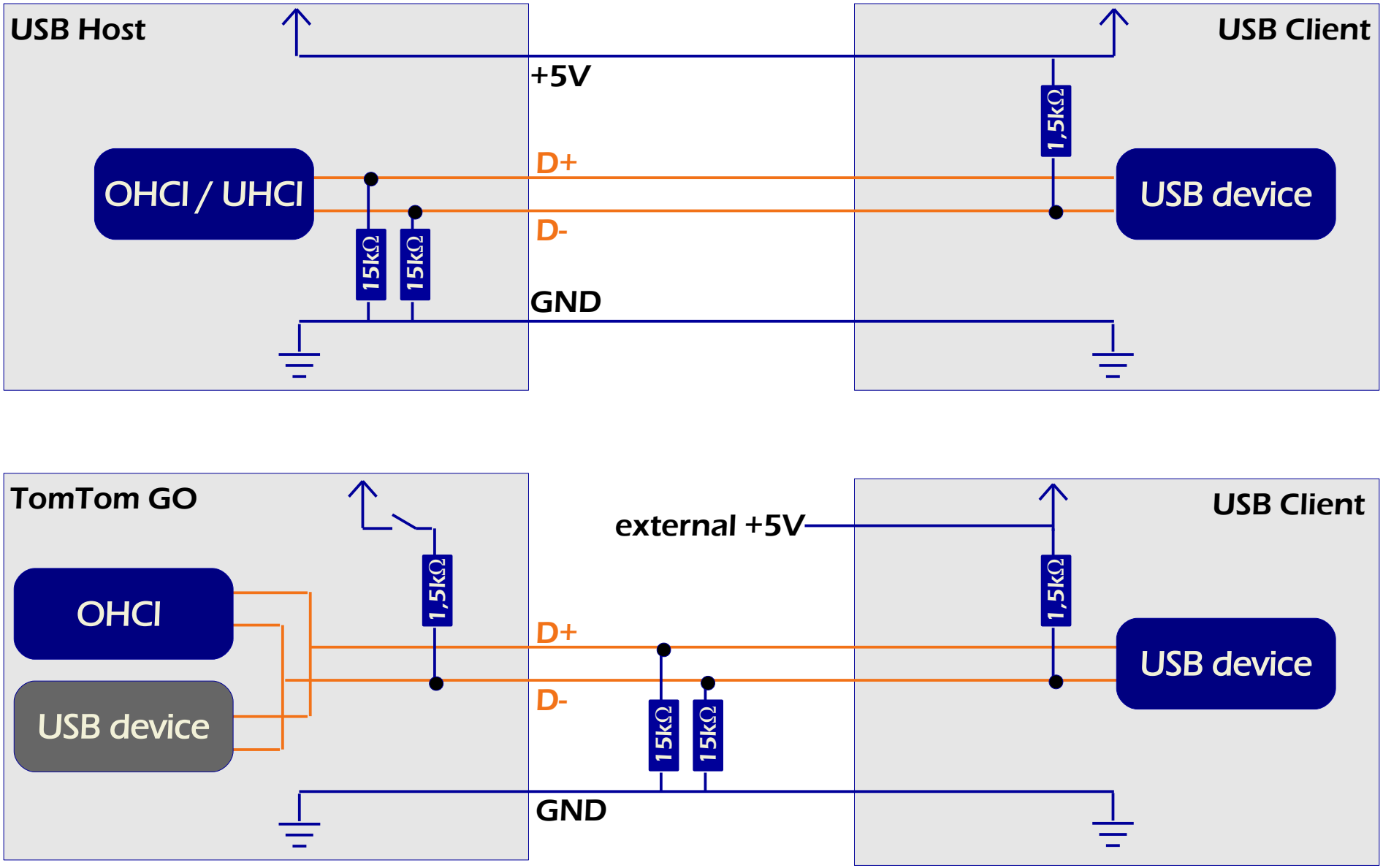
Dock-Port – neuere Geräte

Pin	Signal
M1	Signal ground
M2	Dock sense
M3	Dock sense
M4	Mic-In mono
M5	Line-Out left
M6	Line-Out right
M7	Car-radio mute
M8	Light sense
M9	Ignition sense
M10	ttyS0 TxD (3.3V)
M11	ttyS0 RxD (3.3V)
M12	ttyS0 RtS (3.3V)
M13	ttyS0 CtS (3.3V)
M14	+3.3V from device
M15	+5V from dock



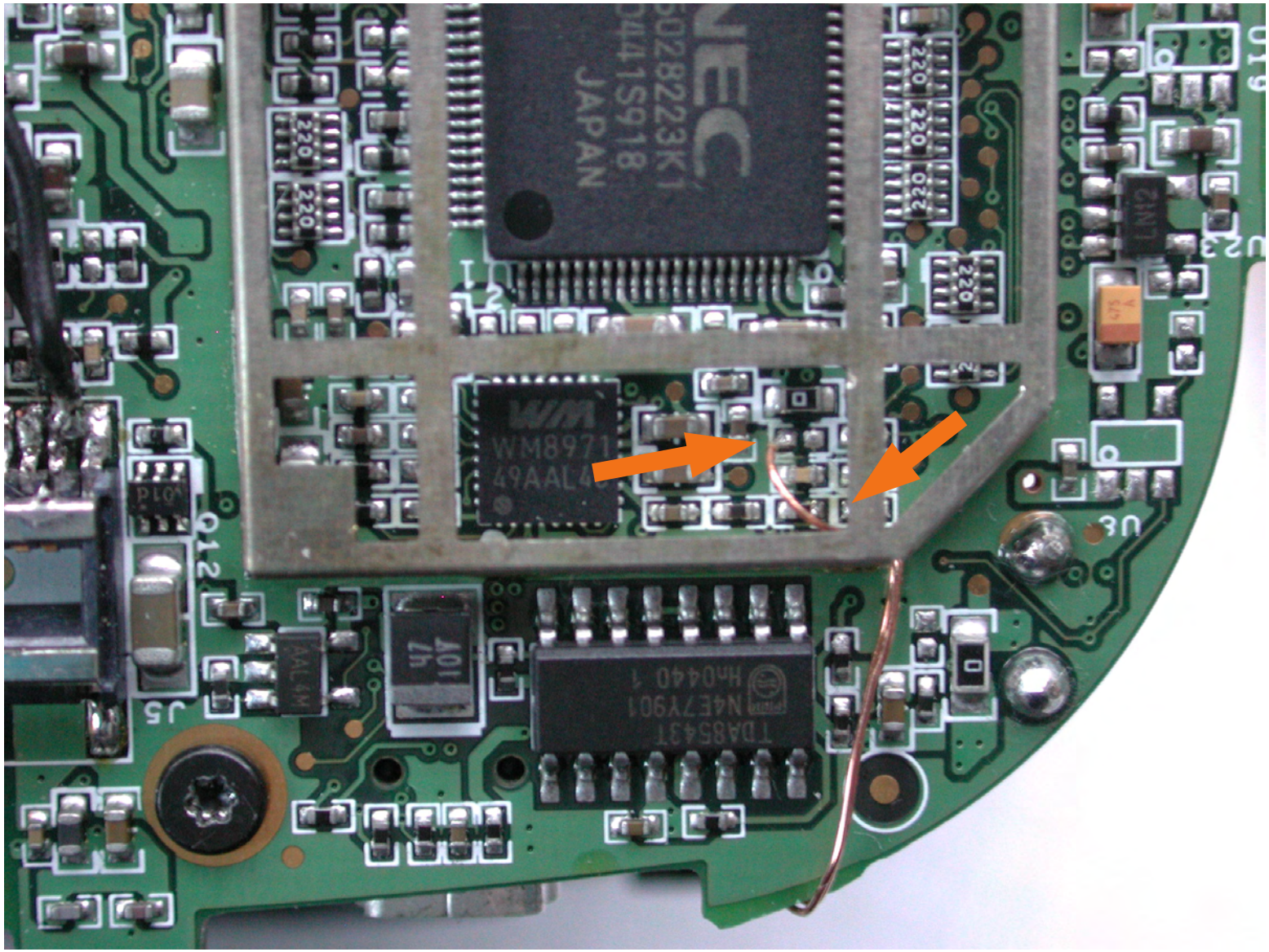
Hardware

USB



Hardware

USB am GO700



Treiber

Display/Framebuffer

Ein ganz normaler Framebuffer-Treiber

allerdings bei fast allen Modellen ist das Display um 90 Grad verdreht eingebaut (daher 240x320)

Zugriff via mmap() auf /dev/fb0

Sound

Für die Freisprecheinrichtung neu entwickelt – 4ms Latenz...

Eigenes CoolSound API

Demnächst gibts auf www.opentom.org einen OSS-Treiber, der ein /dev/dsp exportiert

Treiber

GPS

Beim Classic und GO300 an /dev/ttySAC1,
bei den anderen an /dev/ttySAC2

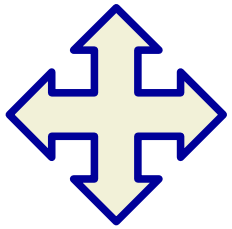
Standard-NMEA GPS-Protokoll, allerdings werden vom seriellen Treiber zusätzlich Timestamps eingefügt:

```
$PTOM105,s,us,*cs
```

G-Sensor

Character-Device 120:0

Sampling-Rate kann über ioctl() eingestellt werden



```
typedef struct {
    unsigned int xData;
    unsigned int yData;
    unsigned int sTimeStamp;
    unsigned int usTimeStamp;
    unsigned int temperature;
} ACCMETER_DATA;
```

Treiber

Touchscreen

Character-Device 254:0

```
typedef struct {
        short pressure;
        short x;
        short y;
        short paddding;
} TS_EVENT;
```

Power-Taste

/dev/input/event0

Event-Device sendet KEY_POWER Tasten-Ereignisse

Fernbedienung

Character-Device 122:0

Kalibrierung über ioctl()

```
typedef struct {
    unsigned int remoteId;
    unsigned char batteryStatus;
    unsigned char keycode;
} RC_EVENT;
```

Was funktioniert denn schon?





ScummVM rockt...



die Rückfahr-Cam auch

Ausblick

Crypto-Key-Server via BT

Radar-Warner :-) (via USB-Receiver)

Anbindung an Motor-Elektronik via CAN

Vielen Dank! Noch Fragen?

www.opentom.org

www.gpl-violations.org

www.maintech.de