
piA-AM35x

Software User Guide

Revision 1.03, Stand 6. Februar 2012



Inhaltsverzeichnis

| | | |
|--------|--|----|
| 1. | Einleitung | 6 |
| 1.1. | Zielgruppe | 6 |
| 1.2. | Hardware | 6 |
| 1.3. | Überblick | 6 |
| 2. | Betriebssoftware | 7 |
| 2.1. | Überblick | 7 |
| 2.2. | Bootvorgang | 7 |
| 2.3. | Linux | 8 |
| 2.4. | Einrichtung der µSD-Karte | 9 |
| 2.4.1. | Partitionierung | 9 |
| 2.4.2. | Bootloader und Kernel | 9 |
| 2.4.3. | Installation der Linux-Systems | 9 |
| 2.5. | Einrichtung für NAND-Boot | 10 |
| 2.5.1. | NAND-Partitionierung | 10 |
| 2.5.2. | NAND-Dateisystem JFFS2 | 10 |
| 2.5.3. | NAND-Installation (manuell) | 10 |
| 2.5.4. | NAND-Installation (automatisiert) | 11 |
| 2.6. | Alternative Betriebssysteme | 11 |
| 3. | U-Boot | 12 |
| 3.1. | Überblick | 12 |
| 3.2. | U-Boot Shell | 12 |
| 3.3. | Default Konfiguration | 12 |
| 3.4. | Anpassung der Bootkonfiguration | 13 |
| 3.4.1. | Bootkonfiguration im NAND | 13 |
| 3.4.2. | Bootkonfiguration mit boot.scr | 13 |
| 4. | Linux – Ångström | 14 |
| 4.1. | Überblick | 14 |
| 4.2. | Userverwaltung | 14 |
| 4.3. | Paketverwaltung | 14 |
| 4.3.1. | Pakete installieren und deinstallieren | 15 |
| 4.3.2. | Informationen über Pakete | 15 |
| 4.4. | Konfiguration | 16 |
| 4.4.1. | Systemlogs | 16 |
| 4.4.2. | Netzwerkkonfiguration | 16 |

| | | |
|----------|--|----|
| 4.4.3. | Besonderheiten von GPRS (nur mit GSM-Modul) | 18 |
| 4.5. | Dateisysteme | 19 |
| 5. | Softwareentwicklung: Linux allgemein | 21 |
| 5.1. | C/C++ Cross-Compiler für ARMv7 | 21 |
| 5.2. | Installation | 21 |
| 5.2.1. | Einrichtung eclipse | 22 |
| 5.2.2. | Eclipse Projekt erstellen und Konfigurieren | 22 |
| 5.3. | Hardwarebibliotheken | 24 |
| 5.4. | Alternative Programmiersprachen | 24 |
| 5.4.1. | Perl & Python | 24 |
| 5.4.2. | Java | 24 |
| 6. | Softwareentwicklung: Hardwarezugriff | 25 |
| 6.1. | Überblick | 25 |
| 6.2. | GPIO | 25 |
| 6.3. | Serielle Schnittstellen | 26 |
| 6.3.1. | GSM-Schnittstelle UART0 | 26 |
| 6.3.2. | Externe RS232/RS485 Schnittstelle UART1 | 26 |
| 6.3.3. | Debugschnittstelle UART2 | 27 |
| 6.3.4. | Serielle Schnittstelle für Tochterkarten UART3 | 27 |
| 6.3.5. | Programmatischer Zugriff auf UARTs | 27 |
| 6.4. | I ² C | 27 |
| 6.4.1. | i2c-tools | 28 |
| 6.4.2. | Zugriff aus C/C++ | 28 |
| 6.4.3. | Hinweise zur Device-ID | 28 |
| 6.5. | Netzwerk | 29 |
| 6.5.1. | Ethernet / IP | 29 |
| 6.5.2. | CAN | 30 |
| 6.6. | USB | 31 |
| 6.7. | Spezielle Devices | 32 |
| 6.7.1. | Temperatursensor | 32 |
| 6.7.2. | Accellerometer | 33 |
| 6.7.3. | RTC | 34 |
| 6.7.3.1. | sysfs Interface | 34 |
| 6.7.3.2. | Userspace Timer | 35 |
| 6.7.4. | Watchdog | 36 |
| 6.7.5. | GSM/GPRS (optional) | 36 |
| 6.7.5.1. | Steuer-IOs des GSM-Moduls | 37 |
| 6.7.5.2. | Beispiel SMS-Versand | 37 |
| 6.7.5.3. | Hinweise zum GPRS-Modus | 38 |

| | |
|--|----|
| 7. Erweiterungsplatinen..... | 39 |
| 7.1. piA-Wireless..... | 39 |
| 7.1.1. RFID | 39 |
| 7.1.2. WLAN/BT | 39 |
| 7.1.2.1. Verbindung zu einem offenen Access Point herstellen | 39 |
| 7.1.2.2. Verbindung zu einem WEP AP | 39 |
| 7.1.2.3. Verbindung zu einem WPA2 AP | 40 |
| 7.1.2.4. Treiber/Firmware | 41 |
| 7.1.3. AD-Wandler | 41 |
| 7.2. Erweiterungsplatine piA LCD | 42 |

Abbildungsverzeichnis

| | |
|---|----|
| Abbildung 1: Ablauf des Bootvorgangs piA-AM3505 | 7 |
| Abbildung 2: eclipse CDT Toolchain Settings | 22 |
| Abbildung 3: eclipse CDT Cross-Compiler Settings..... | 23 |
| Abbildung 4: eclipse CDT Build Settings..... | 23 |
| Abbildung 5: eclipse CDT Discovery Options..... | 24 |
| Abbildung 6: Zugriff auf GPIOs mit gpiolib | 25 |
| Abbildung 7: 3G Sensor, Beschleunigungsachsen | 34 |

Tabellenverzeichnis

| | |
|--|----|
| Tabelle 1: NAND Partitionierung | 10 |
| Tabelle 2: wichtige U-Boot Shell-Kommandos | 12 |
| Tabelle 3: Informationen zum Standard root User | 14 |
| Tabelle 4: Netzwerk-Interfaces..... | 17 |
| Tabelle 5: wichtige pppd Optionen | 18 |
| Tabelle 6: Partitionen und Dateisysteme..... | 20 |
| Tabelle 7: Serielle Schnittstellen | 26 |
| Tabelle 8: GPIOs zur Steuerung der RS232/RS485-Schnittstelle | 27 |
| Tabelle 9: piA-AM35x I ² C Busse und Devices | 29 |
| Tabelle 10: CAN Abschlusswiderstand | 31 |
| Tabelle 11: Watchdog Register | 36 |
| Tabelle 12: GSM-Modul Steuer-GPIOs..... | 37 |
| Tabelle 13: GSM-Modul-Rückmeldungen bei Versand einer SMS | 38 |

Namenskonventionen

| | | |
|-------|---|--|
| NEON™ | - | Floating Point Unit der ARM Cortex-A8 Architektur |
| piA | - | Familie der Singleboard-Computer der Firma pironex |

Änderungsliste

| Datum | Änderung |
|------------|---|
| 31.08.2011 | Neuerstellung |
| 10.01.2012 | Einrichtung der SD-Karte, NAND, Bootkonfigurationen, Recovery, Netzwerkkonfiguration, Updates |
| 13.01.2012 | Korrektur NAND-Bootparameter bessere Beschreibung RS232/485 Mode Selection |
| 06.02.2012 | Abschlusswiderstände CAN/RS485, GPIOs und I ² C Ergänzungen für piAx |

1. Einleitung

1.1. Zielgruppe

Dieses Software User Guide richtet sowohl an reine Anwender der mitgelieferten Software, als auch an Programmierer, die eigene Linux-Software für das Board entwickeln wollen.

1.2. Hardware

Die piA-AM35x Familie umfasst Singleboard-Computer basierend auf ARM Cortex-A8 Prozessoren der Typen OMAP AM3505 oder AM3517 von Texas Instruments mit einem Standard ARMv7-Befehlssatz, Hardware FPU (NEON™) und Grafikkern für beschleunigte 3D-Darstellung (nur AM3517).

Verschiedene optionale Erweiterungen wie GSM/GPRS/UMTS-Modul, WLAN/BT, RFID-Leser, AD-Wandler, LCD-Display-Board werden ebenfalls in entsprechend gekennzeichneten Abschnitten dieser Dokumentation behandelt.

Sofern in dieser Dokumentation die Kurzbezeichnung piA oder piA-AM35x verwendet wird, bezieht sich die Beschreibung auf beide Board-Varianten piA-AM3505 und piAx-AM3517.

1.3. Überblick

Dieses Dokument bezieht sich im Wesentlichen auf grundlegende Eigenheiten, die beim Arbeiten mit dem System beachtet werden müssen, das vorinstallierte Linux-System, dessen Anwendung und den Zugriff auf die verschiedenen Hardware-Subsysteme aus Sicht eines Programmierers.

Sofern eine Beschreibung nur auf eine bestimmte Prozessorvariante zutrifft, wird dies explizit angegeben. Die Verwendung von piA-AM35x zeigt, dass die angegebenen Informationen auf alle Typen zutreffen.

2. Betriebssoftware

2.1. Überblick

Dieser Abschnitt befasst sich mit den verschiedenen Betriebssystemalternativen für das piA-AM35x. Zunächst wird jedoch ein Überblick über den Bootprozess gegeben.

2.2. Bootvorgang

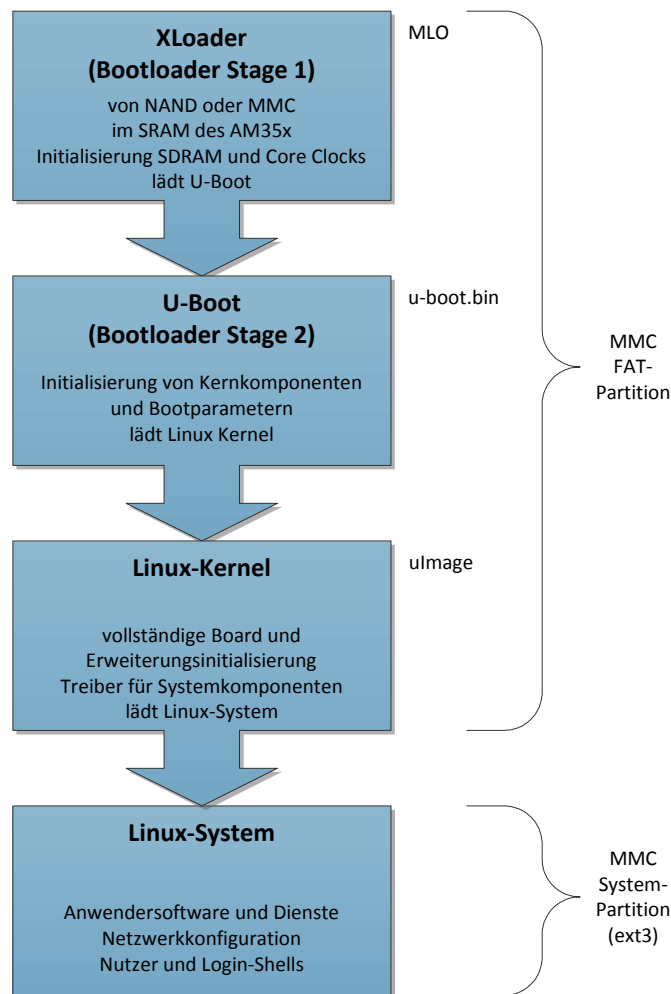


Abbildung 1: Ablauf des Bootvorgangs piA-AM3505

Das System wird im Auslieferungszustand vollständig von der μ SD-Karte geladen. Die Abarbeitung entspricht der Darstellung Abbildung 1.

Die μ SD-Karte ist in 2 Partitionen für Bootloader und System aufgeteilt. Es ist zu beachten, dass der ROM-Code des Prozessors die X-Loader-Datei (MLO) im ersten Block der FAT-Bootpartition erwartet. Sollte eine Kopie der mitgelieferten μ SD-Karte erzeugt werden, muss diese Datei als erste auf die neu formatierte Partition geschrieben werden.

Debug-Ausgaben können über die serielle Schnittstelle UART3 (115200 Baud) erfasst werden¹. Die Ausgaben sollten beim Start des Systems in etwa so aussehen:

¹ Hierfür kann beispielsweise der piUSB-USB Adapter verwendet werden.

```
Texas Instruments X-Loader 1.46 (Aug 17 2011 - 15:00:12)
Starting X-loader on MMC
Reading boot sector

223076 Bytes Read from MMC
Starting OS Bootloader from MMC...
Starting OS Bootloader...

U-Boot 2009.11-dirty (Aug 24 2011 - 10:34:34)

OMAP34xx/35xx-GP ES1.0, CPU-OPP2 L3-165MHz
piA_AM35X + LPDDR/NAND
I2C: ready
DRAM: 256 MB
NAND: 256 MiB
*** Warning - bad CRC or NAND, using default environment

In: serial
Out: serial
Err: serial
Die ID #516000010000000015da39417008023
Net: davinci_emac_initialize
Ethernet PHY: GENERIC @ 0x00
EMAC ID 7c:8e:e4:3c:29:5
DaVinci EMAC
Hit any key to stop autoboot: 3 2 1 0
mmc1 is available
reading boot.scr

** Unable to read "boot.scr" from mmc 0:1 **
reading uImage

3468212 bytes read
Booting from mmc ...
## Booting kernel from Legacy Image at 82000000 ...
Image Name: Angstrom/2.6.37/pia-am35x
Created: 2011-08-24 7:57:59 UTC
Image Type: ARM Linux Kernel Image (uncompressed)
Data Size: 3468148 Bytes = 3.3 MB
Load Address: 80008000
Entry Point: 80008000
Verifying Checksum ... OK
Loading Kernel Image ... OK
OK

Starting kernel ...
...
```

2.3. Linux

Das System wird mit einem auf µSD-Karte installierten Linux System ausgeliefert. Dabei handelt es sich um ein abgewandeltes Ångström² mit angepasstem Linux Kernel (Version 2.6.37).

² <http://www.angstrom-distribution.org>

2.4. Einrichtung der μ SD-Karte

Bei der Einrichtung einer neuen μ SD-Karte ist aufgrund von Eigenheiten des ROM-Bootloaders eine Reihe von Dingen zu beachten.

Die benötigten Tools sind im Paket *pia-utilities.tar.gz* enthalten.

2.4.1. Partitionierung

Die Karte muss mit einem speziellen Cylinder/Sektor-Layout formatiert werden. Hierzu kann unter Linux das Shell-Skript *omap3-mkcard.sh* verwendet werden. Stellt der Kartenleser ein Device */dev/sdh* zur Verfügung, dann sieht der Aufruf wie folgt aus:

```
# ./omap3-mkcard.sh /dev/sdh
```

Wichtig! Es muss darauf geachtet werden, das korrekte Device anzugeben, da keine Nachfrage erfolgt und bei falscher Angabe die System- oder andere Festplatten gelöscht werden können!

Es werden 2 Partitionen *boot* und *Angstrom* erstellt. Erstere wird mit FAT32, die zweite mit EXT3 formatiert.

2.4.2. Bootloader und Kernel

Zum Booten des Systems werden mindestens die beiden Bootloader (*MLO* und *u-boot.bin*) sowie der Linux-Kernel (*uImage*) auf der ersten Partition benötigt. Hierbei ist wichtig, dass die Datei *MLO* vom ROM-Code des Prozessors gelesen wird und dieser sie im ersten Sektor der Partition erwartet.

Dies kann nur durch erneutes Formatieren der Partition sichergestellt werden!

Nach Verwendung von *omap3-mkcard.sh* ist ein erneutes Formatieren nicht nötig, solange noch keine Dateien in das Dateisystem geschrieben worden sind.

Liegen die benötigten Dateien im Verzeichnis */tmp/boot-img/* und wird die μ SD-Karte als */dev/sdh* erkannt, dann kann die Initialisierung der Boot-Partition mit:

```
# mkdosfs -F 32 -n "boot" /dev/sdh
# cd /tmp/boot-img/
# mount -t vfat /dev/sdh1 /mnt
# cp MLO /mnt/
# umount /mnt
# mount -t vfat /dev/sdh1 /mnt
# cp u-boot.bin uImage /mnt/
# umount /mnt
```

erfolgen. (Das *umount* nach dem Kopieren von *MLO* stellt sicher, dass eventuelle Schreibcaches dafür sorgen, dass die Datei nicht in den ersten Sektor geschrieben wird.)

Die optionale *boot.scr* dient der Anpassung der Bootparameter, die von u-boot an den Linux-Kernel übergeben werden (siehe Abschnitt 3.).

2.4.3. Installation der Linux-Systems

Das tar.bz2-Archiv mit dem Linux-rootfs kann auf die zweite Partition kopiert und dort entpackt werden:

```
# mount /dev/sdh2 /mnt
# cp *rootfs.tar.bz2 /mnt/
# cd /mnt
# tar xvf *rootfs.tar.bz2
```

Anschließend können nötige Änderungen an der Konfiguration vorgenommen oder zusätzliche Dateien kopiert werden.

2.5. Einrichtung für NAND-Boot

Das piA-AM35x verfügt über 256 MBytes NAND-Speicher, der für die Installation des Systems verwendet werden kann. Es ist zu beachten, dass der Betrieb aus dem NAND aufgrund der besonderen Art, wie entsprechender Speicher beschrieben und gelesen wird.

Das NAND ist in so genannte Erase Blocks unterteilt. Ein solcher Block hat eine Größe von 128 KBytes und muss vor jedem Überschreiben zunächst eine Löschphase durchlaufen.

2.5.1. NAND-Partitionierung

Das NAND ist in 5 Partitionen aufgeteilt, jeweils für X-Loader (MLO), U-Boot (uboot.bin), U-Boot-Konfiguration (ENV), Linux-Kernel (ulmage) und rootfs:

| Part. | Funktion | Bereich | Größe |
|-------|-----------|-------------------------|------------------------------|
| 1 | MLO | 0x00000000 - 0x0007FFFF | 4 x 128 KBytes |
| 2 | uboot.bin | 0x00080000 - 0x0025FFFF | 15 x 128 KBytes |
| 3 | ENV | 0x00260000 - 0x0027FFFF | 1 x 128 KBytes |
| 4 | ulmage | 0x00280000 - 0x0067FFFF | 32 x 128 KBytes |
| 5 | rootfs | 0x00680000 - 0x0FFFFFFF | 256 MBytes - 64 x 128 KBytes |

Tabelle 1: NAND Partitionierung

Linux erzeugt für jede NAND-Partition mehrere Device-Files, z.B. für direkten Zugriff (`/dev/mtdX`) oder Read Only Zugriff (`/dev/mtdXro`). Um sie als Block-Device, also ähnlich MMC-Partitionen oder USB-Sticks verwenden zu können, ist lediglich `/dev/mtdblkOpX` relevant. X steht hier für die Partitionsnummer entsprechend der Angabe in Tabelle 1.

2.5.2. NAND-Dateisystem JFFS2

Für die Systempartition des Linuxsystems wird ein speziell für NAND-Speicher optimiertes Dateisystem JFFS2 verwendet. Dies kümmert sich um die Organisation der Schreibzugriffe in Erase Blocks und das Handling von Bad Blocks, die bei NAND-Speicher in kleiner Zahl auftreten können.

2.5.3. NAND-Installation (manuell)

Zuerst muss das JFFS2-Image auf die erste Partition der µSD-Karte kopiert werden.

Es ist zu beachten, dass ein 8.3 Namensschema bei der Benennung der Datei verwendet werden sollte! Im unten stehenden Beispiel wurde das Image *Angstrom-pia-...rootfs.jffs2* als *rootfs.img* auf die FAT-Partition kopiert.

Anschließend können die einzelnen Partitionen aus der U-Boot-Shell ins NAND geschrieben werden. Hierbei gehen alle Inhalte, auch eventuelle Änderungen an den Umgebungsvariablen für U-Boot verloren!

1. Vollständiges Löschen des NAND

```
# nand erase
```

2. Installation X-Loader

```
# mmc init
# fatload mmc 0 0x82000000 MLO
# nandeccl hw
# nand erase 0 80000
# nand write 0x82000000 0x00000000 0x00020000
# nand write 0x82000000 0x00020000 0x00020000
# nand write 0x82000000 0x00040000 0x00020000
# nand write 0x82000000 0x00060000 0x00020000
```

3. Installation U-Boot

```
# fatload mmc 0 0x82000000 u-boot.bin
# nandeccl sw
# nand erase 0x00080000 0x001e0000
# nand write 0x82000000 0x00080000 0x00160000
```

4. Installation Linux-Kernel

```
# fatload mmc 0 0x82000000 uImage
# nand erase 0x00280000 0x00400000
# nand write 0x82000000 0x00280000 0x00400000
```

5. Installation rootfs

```
# fatload mmc 0 0x82000000 rootfs.img
# nand write.e 0x82000000 0x00680000 \${filesize}
```

6. Update der U-Boot-Umgebungsvariablen

```
# setenv nandroot /dev/mtdblock4
# setenv nandrootfs jffs2
# saveenv
```

2.5.4. NAND-Installation (automatisiert)

Im Paket *pia-utils.tar.gz* ist ein U-Boot-Skript *boot.scr.installnand* für die automatische Einrichtung des NANDs enthalten, welches beim Starten die oben vollzogenen Befehle automatisch ausführt. Für die Verwendung muss es unter dem Namen *boot.scr* zusammen mit dem JFFS2-Image (*rootfs.img*) auf die erste Partition der µSD-Karte kopiert werden.

Wird das piA von dieser Karte gebootet, dann erfolgt die Installation. Nach Abschluss muss die Karte entfernt werden, damit nach einem erneuten Reboot aus dem NAND gestartet werden kann.

2.6. Alternative Betriebssysteme

TODO

3. U-Boot

3.1. Überblick

„Das U-Boot“ ist ein Bootloader für verschiedene embedded Systeme, der von der Funktionalität vergleichbar ist mit einer Kombination von BIOS + Bootloader auf einem PC. Er kümmert sich um die Konfiguration der Pins, die Auswahl der Bootdevices und das Starten und Konfigurieren des Systemkerns.

3.2. U-Boot Shell

U-Boot hat eine eigene eingebaute Shell, mit der bei Bedarf die Bootkonfiguration angepasst werden kann. Man erreicht diese durch Abbruch des Bootvorgangs mittels Tastendruck im Terminal der seriellen Debug-Schnittstelle.

| Kommando | Funktion |
|-----------------|---|
| help | Kurzhilfe zu allen verfügbaren Kommandos |
| printenv | Listet die gesetzten Umgebungsvariablen für die Bootkonfiguration |
| setenv | Erlaubt das Setzen von Umgebungsvariablen |
| saveenv | Speichert Umgebungsvariablen im NAND |
| boot | Setzt Bootvorgang fort |

Tabelle 2: wichtige U-Boot Shell-Kommandos

3.3. Default Konfiguration

Die Standard-Konfiguration kann innerhalb der U-Boot-Shell ausgegeben werden:

```
PIA_AM35X # printenv

bootcmd=if mmc init; then if run loadbootscrip; then run bootscrip; else if
run loaduimage; then run mmcboot; else run nandboot; fi; fi; else run
nandboot; fi
bootdelay=3
baudrate=115200
bootfile=uImage
loadaddr=0x82000000
console=ttyO2,115200n8
mmccargs=setenv bootargs console=${console} earlyprintk root=/dev/mmcblk0p2 rw
rootfstype=ext3 rootwait
nandargs=setenv bootargs console=${console} root=/dev/mtdblock4 rw
rootfstype=jffs2
loadbootscrip=fatload mmc 0 ${loadaddr} boot.scr
bootscrip=echo Running bootscrip from mmc ...; source ${loadaddr}
loaduimage=fatload mmc 0 ${loadaddr} uImage
mmcboot=echo Booting from mmc ...; run mmccargs; bootm ${loadaddr}
nandboot=echo Booting from nand ...; run nandargs; nand read ${loadaddr}
280000 400000; bootm ${loadaddr}
dieid#=5160000100000000015da39417008023
ethact=DaVinci EMAC
ethaddr=7c:8e:e4:3c:29:5

Environment size: 854/131068 bytes
```

3.4. Anpassung der Bootkonfiguration

Die Umgebungsvariablen von U-Boot können in der Shell mit `setenv` angepasst werden, z.B. würde

```
# setenv bootdelay 10
```

die Haltezeit auf 10 Sekunden erhöhen, bevor der Bootvorgang fortgesetzt wird. Diese Änderungen sind nur in der aktuellen Sitzung aktiv. Ein Neustart des piA lädt wieder die Standardeinstellungen.

Um Änderungen dauerhaft zu machen gibt es 2 Möglichkeiten:

3.4.1. Bootkonfiguration im NAND

Eine Page des NAND-Speichers ist für das dauerhafte Speichern der U-Boot-Umgebung reserviert. U-Boot prüft bei jedem Start, ob dieser Bereich Daten enthält und fügt die dort hinterlegte Konfiguration an die Standardkonfiguration an; gleiche Umgebungsvariablen werden durch die NAND-Daten überschrieben.

Das Speichern der momentanen Konfiguration ins NAND erfolgt durch Aufruf von:

```
# saveenv
```

Um die Konfiguration aus dem NAND zu entfernen, kann

```
# nand erase 0x00260000 0x00020000
```

verwendet werden.

3.4.2. Bootkonfiguration mit `boot.scr`

U-Boot bietet eine Möglichkeit die Booteinstellungen per Script zu ändern. Nach dem setzen der Standard-Variablen und dem anschließenden Aktualisieren der Einstellungen aus dem NAND-Konfigurationsbereich, sucht der Bootloader eine Datei `boot.scr` auf der ersten Partition der µSD-Karte.

Falls diese existiert wird sie anstelle des `boot`-Kommandos ausgeführt. Mit dieser Skriptdatei können beliebige Variablen (neu) gesetzt oder U-Boot-Kommandos wie in der Shell ausgeführt werden.

Die `boot.scr` ist kein reines Textfile. Sie enthält einen binären Header, der mittels `mkimage` generiert wird. Das Tool ist in vielen Distributionen wie Ubuntu im Paket `uboot-mkimage` oder unter einem ähnlichen Namen vorhanden. Alternativ kann die Version aus dem `pia-utils.tar.gz` verwendet werden.

Beispiele für die Verwendung finden sich in den `mkbootscr`-Skripts, die ebenfalls im `pia-utils.tar.gz` zu finden sind.

4. Linux – Ångström

4.1. Überblick

Ångström ist eine speziell für embedded Devices ausgerichtete Linux-Distribution. Sie zeichnet sich insbesondere durch geringen Speicherverbrauch und Optimierung für konkrete ARM-Architekturen aus. Es wird in vielen Software-Paketen auf für den Betrieb überflüssigen Ballast verzichtet.

Die Verzeichnisstruktur entspricht dem LSB-Standard.

Das installierte System enthält die wesentlichen Softwarekomponenten, die für den Betrieb notwendig sind. Weitere können bei Bedarf über ein komfortables Installationstool nachinstalliert werden (siehe Abschnitt 4.3.1).

- Busybox (leichtgewichtige Implementierung der Linux/UNIX-Standardtools), u.a.:
 - Default Shell
 - psutils
 - awk
 - grep
 - halt
 - find
 - wget
- eglibc (Standard-Bibliothek glibc in der embedded-Variante)
- dropbear (SSH-Server)
- lighttpd (leichtgewichtiger HTTP-Server mit vollem Funktionsumfang)
- mc (komfortabler Dateimanager)
- openvpn
- ppp (Netzwerkverbindung über Modem)
- wpa_supplicant (WLAN WPA-Konfigurationsmanagement)

4.2. Userverwaltung

Das vorinstallierte Linux des piA-AM35x kennt im Auslieferungszustand nur den User:

| | |
|-------------------------|-------------|
| Name | root |
| Passwort | <leer> |
| UID | 0 |
| HOME-Verzeichnis | /home/root/ |

Tabelle 3: Informationen zum Standard root User

Zum Anlegen eines zusätzlichen normalen Nutzers mit eingeschränkten Rechten, wird der Linux-übliche Befehl:

```
# adduser <username>
```

verwendet. Die Userkonfiguration findet man in /etc/passwd für Nutzernamen, Standardgruppenzuordnung und Standardshell.

4.3. Paketverwaltung

Ångström verwendet ein dem bekannten Debian-Paketverwaltungssystem (dpkg/apt-get) ähnliches System.

Die verwendete Software ist „opkg“ mit dem gleichnamigen Befehlszeilenkommando, das Paketformat ist „ipk“.

4.3.1. Pakete installieren und deinstallieren

Zum Installieren einer Applikation aus dem Online-Paketfeed, wie z.B. bash, wird folgender Befehl verwendet:

```
# opkg install bash
```

Für die Installation eines lokalen ipk-Pakets muss der komplette Dateiname angegeben werden:

```
# opkg install /tmp/bash_3.2-r7.1_armv7a.ipk
```

Um ein Paket wieder zu entfernen benutzt man:

```
# opkg remove bash
```

Die Online-Paketdatenbank muss gegebenenfalls neu herunter geladen werden. Die geschieht mittels:

```
# opkg update
```

Um alle installierten Pakete zu aktualisieren verwendet man:

```
# opkg update  
# opkg upgrade
```

4.3.2. Informationen über Pakete

Eine Liste der installierten Pakete erhält man mit:

```
# opkg list-installed
```

Eine Liste der online verfügbaren Pakete erhält man mit:

```
# opkg list
```

Informationen zu einem speziellen Paket:

```
# opkg status lighttpd  
Package: lighttpd  
Version: 1.4.26-r9.0.9  
Depends: lighttpd-module-access, lighttpd-module-accesslog, lighttpd-module-  
indexfile, lighttpd-module-dirlisting, lighttpd-module-staticfile, update-  
rc.d, libc6 (>= 2.12), libpcre0 (>= 7.6)  
Provides:  
Status: install ok installed  
Architecture: armv7a  
Conffiles:  
 /etc/lighttpd.conf 734e1477a308848c11d88130634ff8c1  
Installed-Time: 1314174067
```

Welche Abhängigkeiten hat ein Paket?

```
# opkg depends lighttpd  
lighttpd depends on:  
  lighttpd-module-access  
  lighttpd-module-accesslog  
  lighttpd-module-indexfile
```

```
lighttpd-module-dirlisting
lighttpd-module-staticfile
update-rc.d
libc6 (>= 2.12)
libpcre0 (>= 7.6)
```

Zu welchem Paket gehört eine Datei?

```
# opkg search /bin/sh
busybox - 1.18.3-r42.1.9
```

4.4. Konfiguration

Globale Konfigurationsdateien sind Linux typisch im `/etc/`-Ordner abgelegt. Das System verwendet `sysvinit` als `init`-Implementierung. Demzufolge ist die Runlevel-Konfiguration in `/etc/init.d/`, den `/etc/rc.X`-Ordnern und `/etc/inittab` zu finden.

4.4.1. Systemlogs

Das System ist standardmäßig so konfiguriert, dass keine Logdateien auf die Partition geschrieben werden. Der Grund hierfür ist die Verwendung von Flashspeichern (μ SD oder NAND), die eine begrenzte Zahl von Schreibzyklen erlauben.

Das Kernel-Log wird im RAM gepuffert. Man erhält die letzten Zeilen durch Eingabe von:

```
# dmesg
```

Die `syslog` Messages werden ebenfalls im Speicher abgelegt. Die letzten 32 KByte werden durch

```
# logread
```

angezeigt.

File-Logging kann in `/etc/default/busybox-syslog` aktiviert werden. Der Speicherort, Größenrotation, maximale Größe können mit den folgenden Variablen angepasst werden:

```
DESTINATION="file"          # log destinations (buffer file remote)
MARKINT=20                  # interval between --mark-- entries [min]
REDUCE=no                   # reduced-size logging
BUFFERSIZE=64               # buffer: size of circular buffer [kByte]
LOGFILE=/var/log/messages  # file: where to log
```

4.4.2. Netzwerkkonfiguration

Hinweis: Im Auslieferungszustand holt sich das piA die IP-Konfiguration über DHCP, gleichzeitig wird dem Ethernet-Port eine feste IP 192.168.1.51 zugewiesen, diese Konfiguration sollte angepasst werden, da ein nicht vorhandener DHCP-Server im Netz den Bootvorgang sonst unnötig verlangsamt (Netzwerkkonfiguration wartet auf Timeout).

Die Hauptkonfiguration für alle Netzwerkdevices liegt in `/etc/network/`. Hier wird festgelegt, ob ein Device beim Booten automatisch initialisiert und hochgefahren wird, ob DHCP-Server oder statische IP-Adressen verwendet werden sollen.

Jedes Device hat mindestens einen Eintrag der Form in der Datei `/etc/network/interfaces`:

```
iface eth0 inet dhcp
```

„eth0“ ist der Name der Netzwerkschnittstelle. Diesen werden bei gleichem Basistyp von 0 aufsteigend nummeriert.

Soll das Device automatisch beim Booten aktiviert werden, benötigt man eine weitere Zeile:

```
auto eth0
iface eth0 inet dhcp
```

Relevante Device-Namen sind:

| Name | Typ |
|-------|---|
| ethX | Ethernet |
| wlanX | WiFi/WLAN |
| pppX | Point-to-Point, Modem-Verbindung |
| canX | CAN-Adapter |
| lo | Virtuelles lokales Netzwerkinterface mit IP-Adresse 127.0.0.1 |

Tabelle 4: Netzwerk-Interfaces

Eine Liste der momentan aktiven Devices liefert:

```
# ifconfig
```

und aller Devices inklusive der deaktivierten:

```
# ifconfig -a
```

Im Anschluss an die *iface*-Zeile folgt eine genauere Konfiguration der Netzwerkparameter, z.B. IP-Adresse bei statischer IP-Zuweisung. Das folgende Beispiel konfiguriert die eth0 Schnittstelle automatisch beim Booten auf eine statische lokale IP 192.168.1.51 mit einem Standardgateway 192.168.1.1:

```
auto eth0
iface eth0 inet static
    address 192.168.1.51
    network 192.168.1.0
    netmask 255.255.255.0
    gateway 192.168.1.1
    up /bin/starteth0.sh
    down /bin/stopeth0.sh
```

Die up/down-Zeilen geben jeweils ein Shell-Script an, welches automatisch ausgeführt wird, wenn das Device korrekt hoch- bzw. herunter gefahren wurde. Hier können z.B. besondere Routing-Einstellungen vorgenommen oder ein VPN-Tunnel aktiviert werden.

Außerdem werden die Scripts in den Verzeichnissen *if-up.d*, *if-pre-up.d*, *if-down*, *if-pre-down* ausgeführt. Als Beispiel soll */etc/network/if-up.d/ntpdate* dienen, welches nach Aktivierung eines Netzwerkdevices automatisch die lokale Zeit mit einem Internet-NTP-Server abgleicht.

Hinweis: Beim Einloggen über die Shell der seriellen Schnittstelle Debug-Schnittstelle werden die aktuell konfigurierten IP-Adressen ausgegeben.

4.4.3. Besonderheiten von GPRS (nur mit GSM-Modul)

Falls vorhanden, wird das GSM-Modul per Default beim Booten eingeschaltet, eine Internetverbindung ist zu diesem Zeitpunkt noch nicht vorhanden. Dafür muss das Modul zunächst in den GPRS-Modus versetzt werden.

Die entsprechende Konfiguration befindet sich im Verzeichnis `/etc/ppp/`. Sie ist in zwei Hauptbereiche aufgeteilt, der Konfiguration des ppp-Daemons „pppd“, der für den Aufbau und die Aufrechterhaltung der Verbindung zuständig ist und einen Teil, der eine Folge von providerabhängigen AT-Kommandos zum Verbindungsaufbau enthält.

Eine Beispiel-Konfiguration für eine Standardverbindung zum deutschen O2-Netz findet sich in den gprs-Dateien `/etc/ppp/peers:`

```

/dev/tty00 115200
debug kdebug 7
logfile /tmp/gprs.log
connect 'chat -vsf /etc/ppp/chats/gprs'
noccp nobsdcomp noaccomp novj novjccomp noauth
#asyncmap A0000
asyncmap 0
:10.0.0.1
noipdefault defaultroute replacedefaultroute
usepeerdns
#nodetach
user o2
password o2
#ipcp-accept-remote
ipcp-accept-local
#modem
persist
#nolock
    
```

Wichtig sind hier vor allem:

| Option | Beschreibung |
|-------------------------------|---|
| [erste Zeile] | Serielle Schnittstelle und Baudrate |
| noccp, nobsdcomp, ... | Deaktivierung nicht benutzter Kompressionsprotokolle |
| user | Nutzername falls notwendig, darf nicht leer sein |
| password | Passwort falls notwendig, darf nicht leer sein kann alternativ in <code>/etc/ppp/pap-secrets</code> definiert werden |
| persist | Stellt Verbindung nach Disconnect automatisch wieder her |
| debug, kdebug, logfile | spezifiziert Loglevel und Speicherort für Logdatei |
| usepeerdns | verwendet vom PPP-Server gesendete DNS-Konfiguration |
| defaultroute | Setzt eine neue Default-Route über das ppp-Device, falls noch keine Default-Route vorhanden |
| replacedefaultroute | Überschreibt die aktuelle Default-Route bei erfolgreicher Verbindung mit den Einstellungen für das ppp-Device |
| connect ... | Chat-Script für Aufbau der Verbindung |

Tabelle 5: wichtige pppd Optionen

Das chat-Script kümmert sich um die Authentifizierung der SIM-Karte, der Aktivierung des GPRS-Modus und dem Aufbau der ppp-Verbindung:

```

ABORT "BUSY"
    
```

```
ABORT "NO CARRIER"  
ABORT "ERROR"  
ABORT "NO DIALTONE"  
ABORT "NO ANSWER"  
TIMEOUT 90  
"" AT+CPIN=1234 TIMEOUT 10  
ERROR-AT-OK AT+CGDCONT=1,"IP","internet"  
OK AT#GAUTH=1  
OK "ATD*99***1#"   
CONNECT ""
```

Wichtig sind hier die PIN hinter *AT+CPIN* und der Einwahlnamen „internet“ und die Einwahlnummer „*99***1#“. All diese Angaben sind vom Mobilfunk-Provider zu erfahren.

4.5. Dateisysteme

Auf der µSD-Karte sind 2 Partitionen, eine FAT-Bootpartition (Typ 0x0B oder 0x0C) und eine EXT3-Systempartition. Die Bootpartition muss zwingend ein FAT32-Dateisystem ohne Extras (keine langen Dateinamen, nicht größer als 32MB) sein. Diese Partition beinhaltet lediglich die Bootloader und den Linux-Kernel und sollte nicht für andere Zwecke verwendet werden.

EXT3 ist das Standard-Linux-Dateisystem mit Journaling-Funktionalität, es eignet sich gut für MMC-Datenträger.

JFFS2 ist eine Weiterentwicklung des JFFS-Dateisystems. Es ist ausschließlich für Flashspeicher wie dem NAND des piA-AM35x geeignet und geht auf deren Besonderheiten ein. Es kümmert sich um das Handling der erase-Blöcke, erlaubt virtuelles Memory-Mapping, reduziert den IO-Overhead auf ein Minimum³, versucht die Lebensdauer zu maximieren, in dem alle Bereiche des NAND bei Schreiboperationen verwendet werden. Es bietet darüber hinaus eine transparente Kompression, um die Speicherkapazität zu vergrößern und ebenfalls die Zahl der IO-Operationen zu minimieren.

Die RAM-Disks in */var/volatile* und */media/ram/* sollten nur für Informationen verwendet werden, die bei einem Neustart des Systems nicht mehr benötigt werden. Zudem ist zu beachten, dass die maximale Größe von der Menge des Arbeitsspeichers des Systems abhängt. Sie sind besonders für temporäre Log-Files geeignet, da das fortlaufende Schreiben von Log-Dateien die Lebensdauer von Flash basierten Speichermedien drastisch reduzieren kann.

³ Der verwendete NAND-Speicher ist mit 100.000 PROGRAMM/ERASE Cycles spezifiziert.

| Device | Dateisystem | Mountpoint | Verwendung |
|----------------|-------------|----------------|---|
| /dev/mmcblk0p1 | FAT32 | - | Bootpartition µSD |
| /dev/mmcblk0p2 | ext3 | / | Systempartition µSD |
| /dev/mtdblock0 | - | - | reserviert für X-Loader kein Zugriff unter Linux |
| /dev/mtdblock1 | - | - | reserviert für u-boot keing Zugriff unter Linux |
| /dev/mtdblock2 | - | - | reserviert für u-boot Environment kein Zugriff unter Linux |
| /dev/mtdblock3 | - | - | Linux Kernel |
| /dev/mtdblock4 | JFFS2 | / | Systempartition oder zur freien Verfügung bei Verwenden der µSD als Systempartition |
| - | tmpfs | /dev/ | Device Files (im RAM) |
| - | tmpfs | /var/volatile/ | temporäre Files (im RAM) |
| - | tmpfs | /media/ram/ | allgemeine RAM-Disk |

Tabelle 6: Partitionen und Dateisysteme

5. Softwareentwicklung: Linux allgemein

5.1. C/C++ Cross-Compiler für ARMv7

Softwareentwicklung ist grundsätzlich auch auf dem piA-AM35x selbst möglich. Es wird sowohl aus praktischen (Verwendung einer grafischen Entwicklungsumgebung wie eclipse) als auch performancetechnischen Gründen empfohlen, die Programmierung auf einem Linux-PC durchzuführen.

Um auf dem ARM-Zielsystem lauffähigen Code erzeugen zu können, wird ein Cross-Compiler mit Standardbibliotheken, die Toolchain, benötigt, der im SDK enthalten ist. Dieser erlaubt es, auf einem x86 basierten System Quellcode in ARMv7-Binärcode zu übersetzen.

5.2. Installation

Die Toolchain liegt als tar.gz-Archiv vor und muss vor der Verwendung auf dem Entwicklungs-PC entpackt werden. Grundsätzlich kann sie an einen beliebigen Ort auf der Festplatte installiert werden, es wird jedoch empfohlen, den Standardpfad unter `/usr/local/` zu verwenden, da in diesem Fall die Pfadangaben zum Compiler und den include-Dateien der Standardbibliothek über ein einfaches Script automatisch in die Arbeitsumgebung integriert werden können.

Liegt die Datei im `/tmp/`-Verzeichnis, kann sie mit:

```
# cd /  
# tar xvzf /tmp/toolchain_x86_pia_am35x.tar.gz
```

entpackt werden.

Im Verzeichnis `/usr/local/angstrom/arm/` liegt nun die Datei `environment-setup`, die alle benötigten Umgebungsvariablen enthält. Für die Entwicklung reiner Consolen-Anwendungen ist lediglich die `PATH`-Variable interessant. Diese erweitert den Standard-Suchpfad für ausführbare Dateien um den Ort des Cross-Compilers.

Das Einbinden in die aktive Shell erfolgt mit:

```
# source /usr/local/angstrom/arm/environment-setup
```

Sollen diese Anpassungen automatisch bei jedem Login gemacht werden, muss die Zeile in die Startdatei der verwendeten Shell eingetragen werden.

Im Falle von `bash`, `dash` oder einer kompatiblen Implementierung ist dies `$.HOME/.bashrc`.

Die Cross-Compiler und Tools haben für jede Toolchain einen gemeinsamen Prefix. Beim piA-AM35x SDK ist dies `arm-angstrom-linux-gnueabi-`. Soll ein Quelltext „test.c“ für die piA-Plattform übersetzt werden, geht man genauso vor wie bei lokalen Kompilaten. Es muss lediglich der Compiler-Aufruf durch den vollständigen Namen ersetzt werden:

```
# arm-angstrom-linux-gnueabi-gcc -o test test.c
```

statt

```
# gcc -o test test.c
```

Wenn der Rechner für die Entwicklung mit verschiedenen Toolchains eingerichtet ist, sollte beachtet werden, dass sich diese nicht überschreiben, wenn sie dasselbe Prefix verwenden!

5.2.1. Einrichtung eclipse

Eclipse ist eine integrierte Entwicklungsumgebung für eine Vielzahl von Programmiersprachen und –aufgaben. Unter anderem erlaubt die CDT-Variante⁴ der Version Indigo eine komfortable Konfiguration der Toolchain für Crosscompiler-Entwicklung.

Eclipse erlaubt grundsätzlich das Konfigurieren von verschiedenen Umgebungsvariablen innerhalb der Build-Einstellungen. Allerdings müssen diese Anpassungen für jedes neue Projekt vorgenommen werden, was bei einfachen Projekten, in denen lediglich die Anpassung der PATH-Variablen nötig ist, kein Problem ist. Bei komplexeren Anforderungen empfiehlt es sich die Umgebungsvariablen vor dem Start von eclipse in der Shell zu setzen und anschließend aus derselben Shell eclipse aufzurufen.

5.2.2. Eclipse Projekt erstellen und Konfigurieren

Ein neues Cross-Compile-Projekt wird über den normalen Wizzard für ein C oder C++-Projekt erzeugt. Bei der Auswahl der Toolchain wählt man „Cross GCC“⁵, dies kann aber auch im Nachhinein noch angepasst werden (siehe Abbildung 2: eclipse CDT Toolchain Settings).

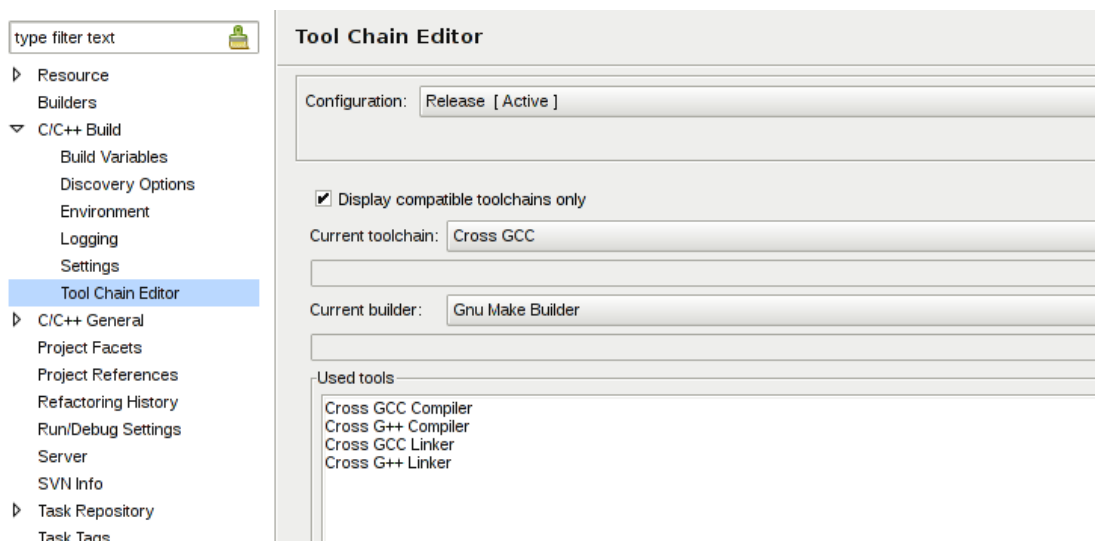


Abbildung 2: eclipse CDT Toolchain Settings

Ein „Cross GCC“ Projekt hat einen zusätzlichen Punkt „Cross Settings“ unter „C/C++ Build → Settings“. Hier kann der Toolchain-Prefix und der Pfad der Compiler-Binaries konfiguriert werden, wie in Abbildung 3 dargestellt.

⁴ <http://www.eclipse.org/downloads/>

⁵ In älteren eclipse Versionen existiert der Cross GCC Toolchain Eintrag nicht. Hier muss die Konfiguration mit einem Standard-GCC erzeugt werden. Danach wird der Compilernamen in den Einstellungen unter „C/C++ Build“ bei „Settings“, „Discovery Options“ und evtl. in den Indexer-Settings angepasst.

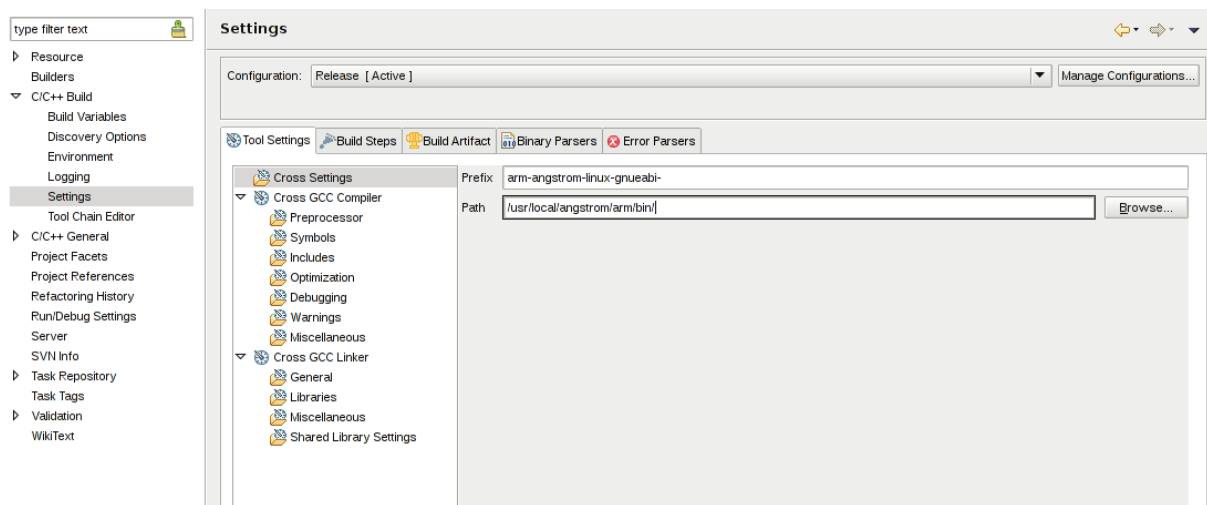


Abbildung 3: eclipse CDT Cross-Compiler Settings

In diesem Dialog wird unter „Build Artifact“ der Typ (Bibliothek, Executable) und Name des zu erzeugenden Binaries eingestellt.

Der Builder sollte auf „extern“ gestellt sein. Eclipse erzeugt die Makefiles automatisch (siehe Abbildung 4).

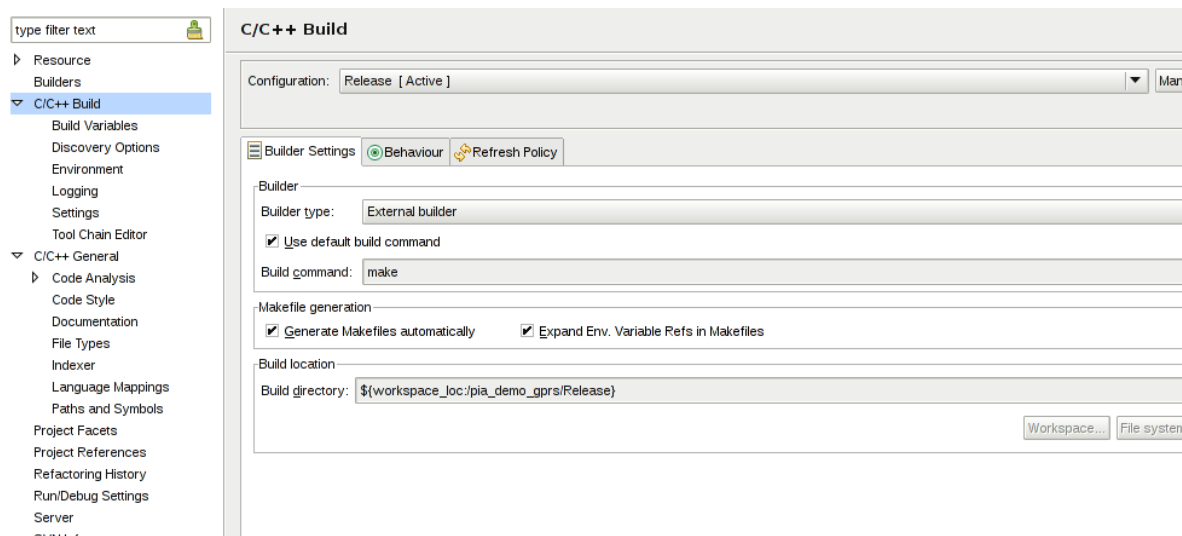


Abbildung 4: eclipse CDT Build Settings

Abschließend wird noch empfohlen die Einstellungen der „Discovery Options“ zu überprüfen (Abbildung 5). Eclipse verwendet diese um automatisch korrekte Include- und Bibliothekspfade vom Compiler abzufragen und in die C/C++-Settings zu übernehmen.

Selbst wenn der Build ohne diese Einstellungen funktioniert, werden sie außerdem für den in eclipse integrierten Indexer, der für die automatische Vervollständigung und Hervorhebung von Syntaxfehlern zuständig ist, verwendet.

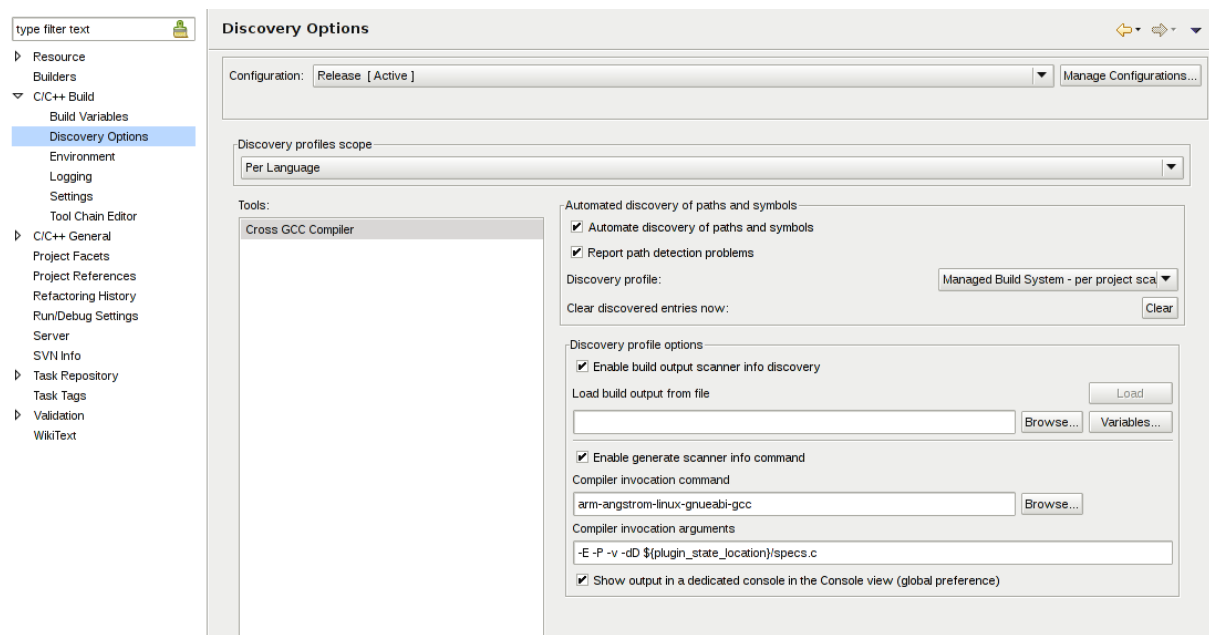


Abbildung 5: eclipse CDT Discovery Options

5.3. Hardwarebibliotheken

Zusammen mit dem piA-AM35x werden eine Reihe von C-Bibliotheken geliefert, die zum einen als Beispiel für den Zugriff auf die Hardwarefunktionen dienen, zum anderen als fertige Programmierschnittstelle verwendet werden können.

Die jeweiligen API-Dokumentationen liegen als Doxygen generierte HTML- und PDF-Dateien vor. Die Implementierungen wurden möglichst einfach gehalten. Sie werden jedoch mit vollständigem Quelltext geliefert, so dass sie für spezielle Anwendungen problemlos erweitert und angepasst werden können.

5.4. Alternative Programmiersprachen

5.4.1. Perl & Python

Interpreter für Perl und Python können über die Online-Paketverwaltung nachinstalliert werden. Die Verwendung dieser Sprachen erfolgt wie auf einem Desktop-PC.

Unter Umständen ist ein direkter Zugriff auf bestimmte Hardwarefunktionen nicht unmittelbar möglich, so dass entsprechende C-Bibliotheken als externe API verwendet werden müssen.

5.4.2. Java

TODO

6. Softwareentwicklung: Hardwarezugriff

6.1. Überblick

6.2. GPIO

Der Linux-Kernel bietet ein gpiolib genanntes Userspace-Interface für die Ansteuerung von GPIOs. Dieses stellt die Steuerung der GPIO-Pins über das /sys-/Dateisystem zur Verfügung. Dies erlaubt einen Zugriff von jeder Programmiersprache, ohne externe Bibliotheken zu benötigen.

GPIOs sind nicht dafür geeignet hochfrequente Ausgangswechsel zu setzen oder Hochfrequente Eingangssignale zu erfassen. Selbst bei Verwendung einer Implementierung mit direktem Hardwarezugriff sind Pegelwechselfrequenzen oberhalb von > 1 MHz nicht per GPIO realisierbar.

Der grundlegende Ablauf ist in Abbildung 6 dargestellt.

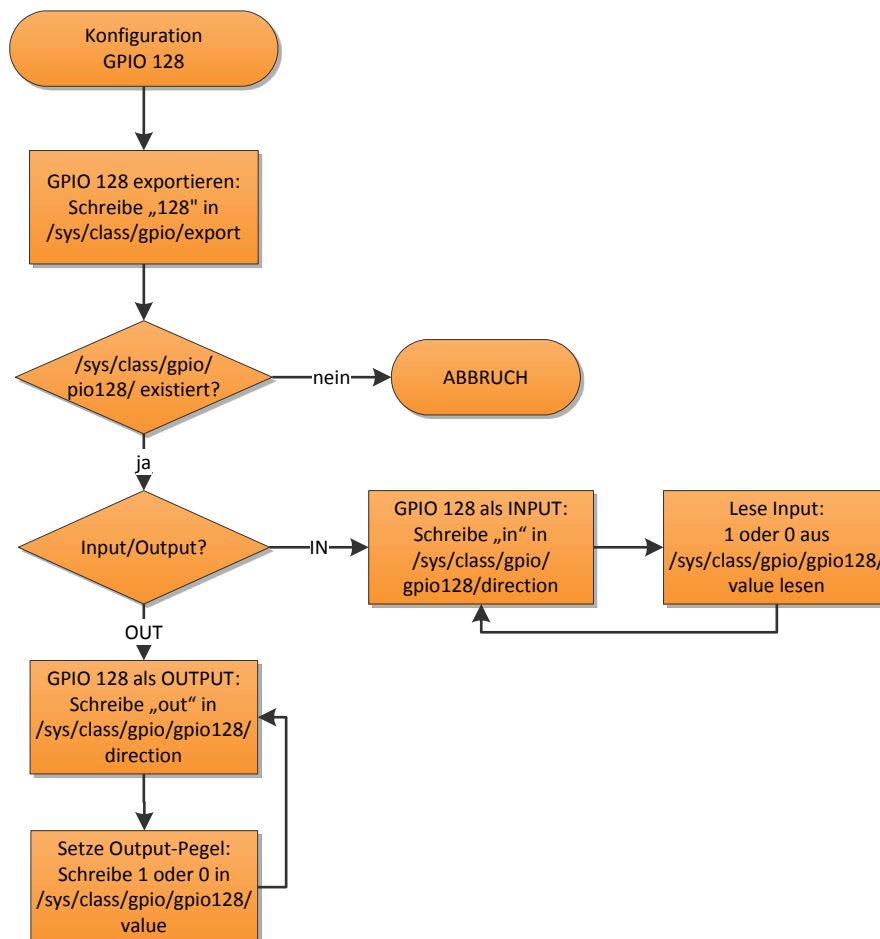


Abbildung 6: Zugriff auf GPIOs mit gpiolib

Ein kurzes Shell-Skript soll als Beispiel dienen, der Zugriff aus anderen Programmiersprachen erfolgt in ähnlicher Weise:

```
#!/bin/sh
GPIO_XY=128
echo -n "Initializing GPIO ${GPIO_XY} as Output, high: "
```

```

echo $GPIO_XY > /sys/class/gpio/export

if [ -e /sys/class/gpio/gpio$GPIO_XY ] ; then
  if [ -e /sys/class/gpio/gpio$GPIO_XY/direction ] ; then
    echo out > /sys/class/gpio/gpio$GPIO_XY/direction
  else
    echo -n "Error while settin direction for ${GPIO_XY}! "
    exit 1
  fi
else
  echo -n "Unable to export GPIO $GPIO_XY"
  exit 1
fi

# finally set output to HIGH
echo 1 > /sys/class/gpio/gpio$GPIO_XY/value

exit 0

```

6.3. Serielle Schnittstellen

Das piA-AM35x bietet vier serielle Schnittstellen, die nachfolgend als UART0 bis UART3 bezeichnet werden.

Jede Schnittstelle erscheint im Linux-System als serielles Device /dev/ttyOX. X ist die Nummer der jeweiligen UART.

| UART | Device | Funktion |
|------|------------|---|
| 0 | /dev/ttyO0 | reserviert für GSM-Modul |
| 1 | /dev/ttyO1 | externes RS232/RS485 Interface |
| 2 | /dev/ttyO2 | Debugschnittstelle |
| 3 | /dev/ttyO3 | Schnittstelle für Tochterkarten, z.B. Bluetooth |

Tabelle 7: Serielle Schnittstellen

6.3.1. GSM-Schnittstelle UART0

UART0 ist für das optionale GSM-Modul reserviert. Es arbeitet mit einer Datenrate von 115200 Baud und kann über AT-Kommandos gesteuert werden. Details zu GSM sind im Abschnitt 6.7.5 zu finden.

6.3.2. Externe RS232/RS485 Schnittstelle UART1

UART1 erlaubt den Betrieb als RS232- oder Halbduplex-RS485-Device. Die Umschaltung des Übertragungsmodus erfolgt über GPIO 128. Für den RS485-Betrieb werden noch weitere GPIOs benötigt.

Der Halbduplexmodus erlaubt keinen konkurrierenden Lese-Schreibzugriff auf die Schnittstelle, weshalb beim Wechsel der Transferrichtung RXEN und DEN wechselseitig umgeschaltet werden müssen. Modus und SLEW müssen nur einmal während der Initialisierung festgelegt werden.

Bei Bedarf lässt sich mittel Jumper auf der Platine bzw. per GPIO ein 120 Ohm Abschlusswiderstand für den RS485-Modus hinzuschalten.

Der Zugriff auf die GPIOs ist im Abschnitt 6.2 beschrieben.

| Name | piA-AM3505 | piAx-AM3517 | Funktion |
|------------------|------------|-------------|--|
| 485/232 | GPIO 128 | GPIO 128 | Modus 0: RS232 1: RS485 |
| SLEW | GPIO 129 | GPIO 129 | RS485: Geschwindigkeitsmodus 0: bis 460 Kbps 1: bis 20 Mbps RS232: nicht benutzt |
| RXEN | GPIO 144 | GPIO 34 | RS485: 0: Receiver disable 1: Receiver enable RS232: 0: Driver disable 1: Driver enable (RX+TX) |
| DEN | GPIO 145 | GPIO 35 | RS485: 0: Driver output disable 1: Driver output enable RS232: 0: Defaultmodus, kein Loopback 1: Loopback TX => RX active |
| RS485_RES | GPIO 27 | GPIO 42 | RS485 Abschlusswiderstand 0: deaktiviert (default) 1: aktiviert |

Tabelle 8: GPIOs zur Steuerung der RS232/RS485-Schnittstelle

Auf die seriellen Schnittstellen kann mit Standardtools zugegriffen werden. Ein einfaches Terminal-Programm picocom ist vorinstalliert. Ein Beispielaufruf für die Schnittstelle UART1 sieht folgendermaßen aus:

```
# picocom -b 115200 /dev/tty01
```

6.3.3. Debugschnittstelle UART2

Die Debugschnittstelle dient der Ausgabe der Bootloader und Linux Kernel Messages. Für die Verbindung mit dem PC kann beispielsweise der piA-USB-Adapter (TODO?) angeschlossen werden.

Nachdem der Bootvorgang abgeschlossen ist, öffnet sich auf dem Device eine Login-Shell.

6.3.4. Serielle Schnittstelle für Tochterkarten UART3

Diese Schnittstelle ist in der Standardkonfiguration deaktiviert. Sie wird ausschließlich für Tochterkarten wie piA PLUS Wireless verwendet.

6.3.5. Programmatischer Zugriff auf UARTs

Ein Beispiel für den asynchronen Zugriff auf die seriellen Schnittstellen mittels C ist die lib-serial Bibliothek aus den SDK-Beispielen.

6.4. I²C

Das piA-AM35x bietet drei I²C-Busse. Einzelne Devices werden über die Bus-Nummer und ihre Device-ID identifiziert. Letztere ist der piA-AM35x Hardware-Dokumentation für die integrierten Devices bzw. den jeweiligen Datenblättern zu entnehmen.

6.4.1. i2c-tools

Für den Zugriff auf den I²C-Bus bietet Linux mit den i2c-tools⁶ eine einfache Bibliothek, die alle gängigen Hardwareimplementierungen unterstützt.

In der Dokumentation zum Temperatursensor (Abschnitt 6.7.1) ist ein kurzes Beispiel für die Verwendung des in i2c-tools enthaltenen Kommandozeilentools angegeben. Für genauere Informationen ist die entsprechende man-page bzw. die Online-Dokumentation zu Rate zu ziehen.

6.4.2. Zugriff aus C/C++

Der programmatische Zugriff aus C/C++ erfordert das Einbinden von „i2c-tools.h“. Eine Bibliothek wird nicht benötigt.

Wie die meisten anderen Busse und Geräte existiert auch bei I²C für jeden Bus ein Device-File unter `/dev/i2c-<busnummer>`.

Soll beispielsweise auf ein Gerät an Bus 2 mit der Device-ID 0x48 (Temperatursensor lm75) zugegriffen werden, muss der Bus zunächst geöffnet und dann das gewünschte Device über einen ioctl-Aufruf selektiert werden:

```
int i2c_handler = open(filename, O_RDWR);
if (i2c_handler < 0)
    // Bus-Fehler
if (ioctl(fd, I2C_SLAVE, dev_addr) < 0)
    // Device-Fehler
```

Anschließend können mit den `i2c_smbus_read_...` bzw. `i2c_smbus_write_...` Funktionen Lese- und Schreibaktionen ausgeführt werden:

```
#define TEMP_REG 0x00

int value = 0;
value = i2c_smbus_read_word_data(fd, TEMP_REG);
```

6.4.3. Hinweise zur Device-ID

I²C-Device-IDs haben eine Länge 7 Bit, das 8. Bit wird auf dem Bus für die Bestimmung von Lese/Schreibrichtung verwendet.

Dokumentationen geben Device-IDs auf zwei verschiedene Arten an. Entweder als einzelne ID, z.B. 0x60 oder als Lese-ID 0xC1 und Schreib-ID 0xC0. Beide Angaben sind äquivalent. Die i2c-tools verwenden die erste Schreibweise, eine rechtsbündige 7 Bit ID.

⁶ http://www.lm-sensors.org/wiki/I2CTools_Documentation

| Device | I ² C-Bus | Device-ID | Beschreibung |
|-------------------------|----------------------|-----------|--|
| Audio | 2 | 0x30 | Steuerung Audio IC (nur piAx-AM3517) |
| Temperatursensor | 2 | 0x48 | Typ LM75, Hardwaremonitoring |
| 3G-Sensor | 2 | 0x4C | Typ MMA76600, 3-Achsen-Beschleunigungssensor |
| Powermanagement | 1 | mehrere | Powermanagement Chip, wird durch Linux Kernel verwaltet |
| DS1374 | 1 | 0x68 | Watchdog + RTC |
| EEPROM | 1 | 0x50-0x57 | EEPROM (nur piAx-AM3517) |
| - | 3 | - | für Tochterkarten, u.a. AD-Wandler, Display, Touch-Control, EEPROM |

Tabelle 9: piA-AM35x I²C Busse und Devices

6.5. Netzwerk

Netzwerkdevices haben eine Sonderrolle unter Linux: es gibt keine zugehörigen Device-Files in `/dev/`, stattdessen wird eine Einheitliche Socket-API für diverse Devices und Protokolle verwendet. Jeder Socket beschreibt einen einzelnen Kommunikationsendpunkt.

Im Web sind umfangreiche Dokumentationen und Tutorials zu den Stichworten „linux socket api“ zu finden. An dieser Stelle soll anhand eines kurzen Beispiels die grundlegende Arbeitsweise verdeutlicht werden. Die Implementierung auf dem piA-AM35x unterscheidet sich dabei in keiner Weise von der auf anderen Geräten.

6.5.1. Ethernet / IP

Das piA hat eine Fast-Ethernetschnittstelle, die als Device `eth0` ins Device eingebunden ist.

Um als Client eine TCP/IP-Verbindung zu einem Server aufzubauen, muss zunächst das Socket erstellt werden:

```
#include <sys/socket.h>
#include <arpa/inet.h>
#include <stdlib.h>
#include <netdb.h>

/* ... */

int sock = 0;
if ((sock = socket(PF_INET, SOCK_STREAM, IPPROTO_TCP)) < 0)
    /* Fehler: Socket konnte nicht initialisiert werden */
```

Dies legt Protokollfamilie IPv4 (`AF_INET`) mit verbindungsorientiertem (`SOCK_STREAM`) Protokoll TCP fest.

In den meisten Fällen ist eine DNS-Auflösung notwendig:

```
#define TARGET "www.google.com"

struct hostent *host;
struct in_addr h_addr;

if ((host = gethostbyname(TARGET)) == NULL) {
    /* Fehler bei Namensauflösung */
    exit(1);
}
```

```
h_addr.s_addr = *((unsigned long *)host->h_addr_list[0]);
printf("nslookup returned: %s\n", inet_ntoa(h_addr));
```

Jetzt kann die Verbindung zum Server aufgebaut werden:

```
static struct sockaddr_in server;
memset(&server, 0, sizeof (server));
server.sin_family = AF_INET;
server.sin_addr = h_addr;
server.sin_port = htons(80); /* Port 80 */
if (connect(sock, (struct sockaddr*)&server, sizeof (server)) < 0) {
    fprintf(stderr, "Failed to connect to server\n");
    exit(1);
}
```

An dieser Stelle hat die erste Kommunikation stattgefunden (TCP SYN-ACK mit Server). Wenn es keinen Fehler gegeben hat kann der eigentliche Datenverkehr gestartet werden:

```
int res = 0;
char *httpreq = "GET / HTTP/1.0\r\nHost: www.google.com\r\nUser-Agent: piA-AM35x Networking Demo\r\n\r\n";

/* Fordere Startseite von www.google.com an */
res = send(sock, httpreq, strlen(httpreq), 0);
if (res < 0) {
    /* Konnte Request nicht senden */
    exit(1);
}

/* Lese Antwort */
char buf[BUFSIZ+1];
memset(buf, 0, sizeof(buf));

while ((res = recv(sock, buf, BUFSIZ, 0)) > 0) {
    /* buf enthält Teil des Response */
    fprintf(stdout, buf);

    /* Stelle sicher, dass die empfangenen Teile \0-terminiert sind */
    memset(buf, 0, tmpres);
}
```

Das Beispiel baut eine Verbindung zu <http://www.google.com> auf, sendet ein http-Request und gibt die Antwort in der Shell aus.

6.5.2. CAN

CAN-Devices werden in Linux ebenfalls als Netzwerkdevices behandelt. Die „Socket CAN“-API erlaubt die Kommunikation auf dem CAN-Bus mittels BSD Sockets.

Mittels Jumper auf dem Board bzw. GPIO per Software kann ein Abschlusswiderstand geschaltet werden:

| Name | piA-AM3505 | piAx-AM3517 | Funktion |
|----------------|------------|-------------|---|
| CAN_RES | GPIO 26 | GPIO 36 | CAN-Abschlusswiderstand 0: aus 1: ein |

Tabelle 10: CAN Abschlusswiderstand

Der ins piA-AM35x integrierte CAN-Controller bindet sich als Device „can0“ ins System ein:

```
# ifconfig -a can0
```

Analog zum Ethernet muss zunächst ein socket registriert werden:

```
int cansocket = socket(PF_CAN, SOCK_RAW, CAN_RAW);
```

Es wird ein RAW CAN-Socket ohne Protokoll erzeugt. Der folgende Code verwendet *bind* zum Öffnen des Sockets, da hier keine Verbindung zu einem zweiten Host aufgebaut wird, sondern die Verknüpfung zum CAN-Bus hergestellt werden soll:

```
struct sockaddr_can addr; /* CAN Adresse mit Message ID */
struct ifreq ifr;
struct can_frame frame;

/* ermittle Interface Index */
strcpy(ifr.ifr_name, "can0");
ioctl(cansocket, SIOCGIFINDEX, &ifr);
addr.can_family = AF_CAN;
addr.can_ifindex = ifr.ifr_ifindex;

/* Verbinde mit CAN-Bus */
bind(cansocket, (struct sockaddr *)&addr, sizeof(addr));

/* Schreibe 11 22 33 44 55 66 77 88 mit ID 0x50 */
frame.can_id = 0x50;
int i, data = 11;
for (i = 0; i < 8; ++i) {
    frame.data[i] = data;
    data += 11;
}
frame.can_dlc = 8; /* Anzahl Datenbytes */

write(cansocket, &frame, sizeof(frame) );

/* ... */

close(cansocket);
```

6.6. USB

Der integrierte µUSB-Port bietet als OTG-Port sowohl Host- als auch Device-Funktionalität. Die Umschaltung erfolgt automatisch durch den Treiber, wenn das korrekte Kabel mit ID-Pin-Kodierung verwendet wird.

Eine Möglichkeit des programmatischen Zugriffs auf den USB-Bus und angeschlossene Devices, ist die Verwendung von libusb⁷. Die Ångström Repositories enthalten Pakete sowohl für Version 0.1 (welche mit der Windowsvariante quellcodekompatibel ist) und 1.0.

6.7. Spezielle Devices

6.7.1. Temperatursensor

Der integrierte Temperatursensor vom Typ LM75 ist per I²C-Bus (Bus 2, ID 0x48, siehe auch Abschnitt 6.4) ins System eingebunden und kann auf verschiedene Arten ausgelesen werden.

Der Linux-Kernel bringt einen Hardwaremonitoring-Treiber für den Sensor mit, so dass die Standard-Tools aus der lm-sensors-Suite⁸ verwendet werden kann. Mit

```
# lsmod
Module                Size  Used by
ppp_async              7196  1
ppp_generic            23409  5 ppp_async
slhc                   5075  1 ppp_generic
g_serial              24224  2
lm75                   3935  0
```

kann zunächst überprüft werden, ob das Treiber-Modul „lm75“ geladen ist.

Dann gibt ein Aufruf von

```
# sensors
lm75-i2c-2-48
Adapter: OMAP I2C adapter
temp1:      +33.0 C (high = +80.0 C, hyst = +75.0 C)
```

Die aktuelle Temperatur und die Alarm-Einstellungen aus.

Programmatisch können dieselben Werte ähnlich dem GPIO-Interface gelesen werden:

```
# cd /sys/class/hwmon/hwmon0/device/
# ls
driver          name          temp1_input    uevent
hwmon           power         temp1_max
modalias        subsystem     temp1_max_hyst
# cat name
lm75
# cat temp1_input
33000
# cat temp1_max
80000
# cat temp1_max_hyst
75000
```

Die Angaben haben eine Auflösung von 0,001 °C. Der Maximal- und der Hysterese-Wert können durch Schreiben in das entsprechende File verändert werden.

⁷ <http://www.libusb.org/>

⁸ <http://lm-sensors.org/>

Als letzte Möglichkeit ist ein direktes Auslesen über den I²C-Bus möglich. Dies funktioniert allerdings nur dann, wenn der Kernel-Treiber nicht geladen ist, da dieser sonst alle Zugriffe auf das Device abfängt.

Zum Entladen des Treibers dient der Befehl:

```
# rmmmod lm75
```

Das Tool `i2cget` ermöglicht den direkten Zugriff auf den I²C-Bus aus der Shell heraus:

```
# i2cget 2 0x48 0 b
WARNING! This program can confuse your I2C bus, cause data loss and worse!
I will read from device file /dev/i2c-2, chip address 0x48, data address
0x00, using read byte data.
Continue? [Y/n]
0x21
```

Es wird auf dem I²C Bus 2 ein Byte der Registeradresse 0 des Devices mit der ID 0x48 gelesen. Der gelesene Wert 0x21 entspricht einer Temperatur von 33°C.

Für den programmatischen Zugriff auf den I²C-Bus kann die mitgelieferte `i2c`-Bibliothek verwendet werden oder die `i2c-tools-API`, wie in Abschnitt 6.4 beschrieben, verwendet werden.

6.7.2. Accellerometer

Der 3G Beschleunigungssensor vom Typ MMA7660 des piA-AM35x ist ebenfalls an den I²C Bus 2 mit Device ID 0x4C angebunden. Die Ansteuerung mit C/C++ erfolgt äquivalent zum Temperatursensor.

Das SDK enthält eine Bibliothek, die ein einfaches Auslesen der Sensorwerte mit Standardeinstellungen (kein Sleep, kein Interrupt, 16 Samples/s) ermöglicht.

Zur genaueren Konfiguration des Sensors sollte das Datenblatt konsultiert werden. Der Sensor bietet einige Zusatzfunktionen, wie z.B. Lageerkennung, Shake-Detection, Auto-Sleep/-Wakeup, Standby oder das genauere Einstellen der Sampling-Rate (1-120 Samples/s) erlauben.

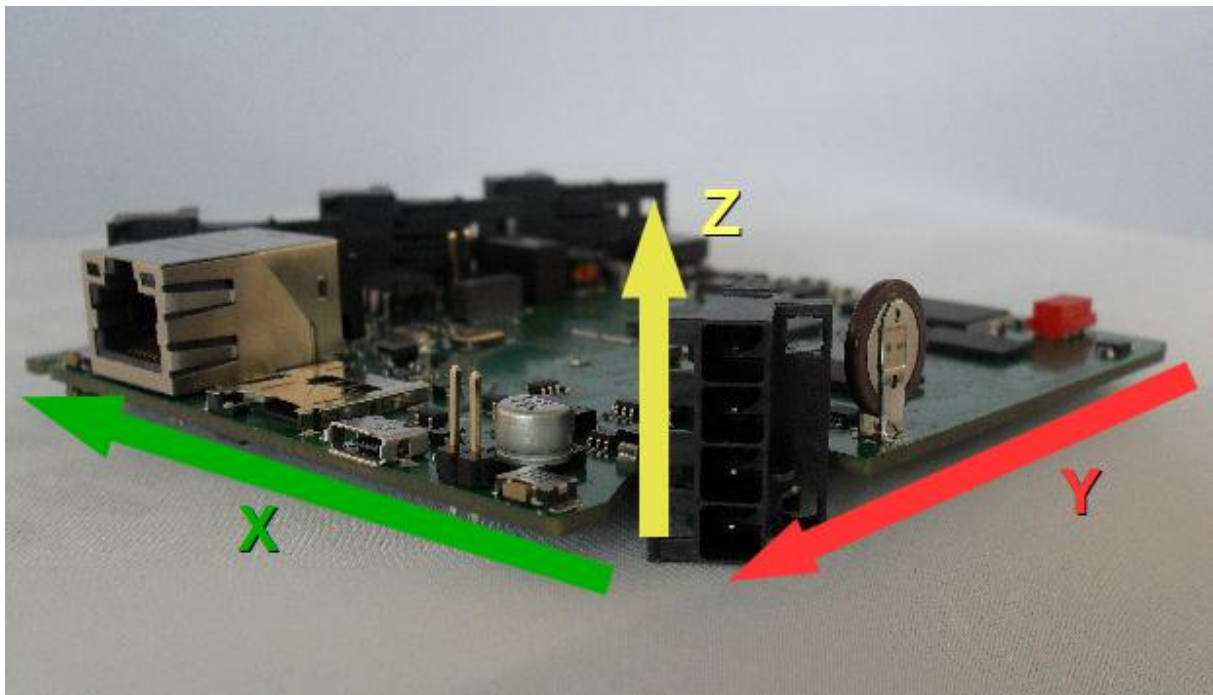


Abbildung 7: 3G Sensor, Beschleunigungsachsen

Abbildung 7 zeigt die Beschleunigungsachsen. Eine Beschleunigung in Pfeilrichtung vergrößert den Sensormesswert.

6.7.3. RTC

Linux führt eine virtuelle System-Uhr, die als Zeitgeber dient. Diese wird bei jedem Systemstart mit dem Datum der Hardware-RTC initialisiert.

Die Hardware-RTC läuft, dank der integrierten Batterie, auch bei ausgeschaltetem System weiter. Trotzdem ist es ratsam die Zeit regelmäßig mit einem NTP-Zeitserver abzugleichen. Das vorinstallierte ntpdate-Paket realisiert dies automatisch, sofern eine Internetverbindung vorhanden ist.

Es muss beachtet werden, dass die RTC Linux typisch in UTC geführt wird, die lokale Zeit wird gegebenenfalls durch einen Zeitzonen-Offset angepasst (Umgebungsvariable „TZ“).

6.7.3.1. sysfs Interface

Informationen und Einstellungen zur RTC sind über das sysfs-Interface zugänglich:

```
# cd /sys/class/rtc/
# ls
rtc0
# cd rtc0/
# ls
date          hctosys      power        time
dev           max_user_freq since_epoch  uevent
device        name         subsystem
```

Zeit und Datum können in einem standardisierten Format aus den Dateien „date“ bzw. „time“ gelesen werden:

```
# cat date
```

```
2011-09-01
# cat time
07:56:16
# date
Tue Sep  1 07:56:18 UTC 2011
# TZ="Europe/Paris" date
Tue Sep  1 09:56:20 CEST 2011
```

6.7.3.2. Userspace Timer

In Linux existieren verschiedene Möglichkeiten, Timer zu implementieren. Die gängigsten Methoden sind nachfolgend aufgelistet. Highlevel Framework-APIs bieten unter Umständen noch andere Interfaces, die letztlich jedoch alle auf denselben Mechanismen basieren.

Die POSIX-Funktion `settimer()` erlaubt das registrieren eines Timers in Userspace-Programmen. Der Prozess wird regelmäßig nach Ablauf der eingestellten Zeitspanne durch ein SIGALRM unterbrochen.

Eine Alternative, die erlaubt `poll/select` auf einen File-Descriptors zu verwenden, ist durch die POSIX-Funktionen `timerfd_create/gettime/settime` gegeben. Der wesentliche Ablauf ist:

- Initialisiere File-Descriptor mit `timerfd_create(...)`
- Setze Parameter mit `timerfs_settime(...)`
- `read()/poll()/...` auf File-Descriptor, der gelesene Wert ist die Zahl der Timerevents seit dem letzten Aufruf.

Die Manpage von `timerfd_create` zeigt ein einfaches Beispiel⁹.

Eine weitere Alternative ist die Verwendung der Eventbibliothek `libevent`¹⁰. Die Prozesse werden nicht unterbrochen und man kann Callback-Funktionen für Timer-Events festlegen:

- `timeout_set(...)` initialisiert event struct für Timer
- `timeout_add(...)` activates a timeout event
- `timeout_del(...)` removes a timeout event

```
#include <event.h>

void timer_callback(int fd, short event, void *arg) {
    struct event *ev = arg;
    struct timeval tv;
    timerclear(&tv);
    tv.tv_sec=1;
    event_add(ev, &tv);
    printf("timer tick\n");
}

int main (int argc, char * argv[]){
    struct event ev;
    struct timeval tv;
    int fd = -1;

    event_init();

    event_set(&ev, fd, EV_TIMEOUT, timer_callback, &ev);
    timerclear(&tv);
    tv.tv_sec=1;
```

⁹ http://www.kernel.org/doc/man-pages/online/pages/man2/timerfd_create.2.html

¹⁰ <http://monkey.org/~provos/libevent/>

```

event_add(&ev, &tv);
event_dispatch();

return 0;
}
    
```

6.7.4. Watchdog

Der Watchdog erlaubt das Zurücksetzen des piA-AM35x, wenn über einen konfigurierbaren Zeitraum kein Signal an den Watchdogbaustein gesendet wird.

Lesen Sie den LSB des WDOG Counter und CTRL-Register:

```

# i2cget -f 1 0x68 0x04 b
0x00
# i2cget -f 1 0x68 0x07 b
0x06
    
```

Setzen Sie den Modus auf Counter enable (Bit 6 = 1), Watchdog on (Bit 5 = 1), Reset on (Bit 3 = 0):

```

# i2cset -f 1 0x68 0x07 0x66 b
    
```

Setzen Sie den Counter auf den neuen Wert 0x20 (20 x 1/4096s) und warten Sie auf RESET:

```

# i2cset -f 1 0x68 0x04 0x20 b
#
    
```

Das piA-AM35x sollte nun neu starten.

Der programmatische Zugriff erfolgt analog zu anderen I²C Devices.

| Register | Beschreibung |
|-------------|--|
| 0x04 | LSB des RESET-Counter im Watchdogmodus entspricht 1 Takt 1/4096s |
| 0x05 | 2. Byte des RESET-Counters |
| 0x06 | 3. Byte des RESET-Counters |
| 0x07 | Control-Register Bit 6: 1 = aktiviere Counter Bit 5: 1 = Watchdog Modus / 0 = Alarm Modus, Counter in Sekunden Bit 3: 0 = wenn Counter 0, dann wird System resettet alle anderen Bits sollten nicht verändert werden |

Tabelle 11: Watchdog Register

6.7.5. GSM/GPRS (optional)

Das optionale GSM-Modul wird über die serielle Schnittstelle mit Telit-AT-Kommandosatz angesprochen. Eine ausführliche Beschreibung der Kommandos und deren Syntax ist dem Telit Software User Guide¹¹ zu entnehmen.

An dieser Stelle soll beispielhaft das Versenden einer SMS vom piA-AM35x mittels picocom dokumentiert werden. Die programmatische Umsetzung kann analog über die serielle Schnittstelle /dev/ttyO0 erfolgen.

¹¹ http://www.telit.com/en/products/gsm-gprs.php?p_id=12&p_ac=show&p=89

6.7.5.1. Steuer-IOs des GSM-Moduls

Das GSM/GPRS-Modul verwendet 3 Steuer-GPIOs, welche die Stromversorgung und den Betriebszustand steuern. Das Bootscript `/etc/init.d/gprs` setzt alle 3 GPIOs auf den Default-Zustand (gepowerd und aktiviert).

| Name | GPIO | Beschreibung |
|----------------|------|--|
| POWERON | 29 | 1 = Power ON 0 = Power OFF |
| ONOFF | 127 | 0 = active 1 = inactive Hinweis: Das Modul befindet sich nach setzen in den inactive Zustand im LowPower-Modus und wird erst durch Zurücksetzen der Stromversorgung über POWERON wieder aktiviert! |
| RESET | 128 | Default: 0 Führt ein Emergency Reset des Moduls aus. Sollte nach Möglichkeit nicht verwendet werden, da der Betriebszustand anschließend undefiniert ist. |

Tabelle 12: GSM-Modul Steuer-GPIOs

6.7.5.2. Beispiel SMS-Versand

Es wird vorausgesetzt, dass zuvor bereits mit

```
AT+CPIN=XXXX
```

die SIM-Karte mit der korrekten PIN aktiviert wurde.

```
# picocom -b 115200 /dev/tty00
picocom v1.4

port is      : /dev/tty00
flowcontrol  : none
baudrate is  : 115200
parity is    : none
databits are : 8
escape is    : C-a
noinit is    : no
noreset is   : no
send_cmd is  : ascii_xfr -s -v -l10
receive_cmd is : rz -vv

Terminal ready
AT
OK
AT+CMGF=1
OK
AT+CMGS="0179TELEFONNUMMER"
> Hallo, dies ist ein Test
+CMGS: 69

OK
```

Dies sendet an die anzugebende Telefonnummer eine SMS mit dem Inhalt "Hallo, dies ist ein Test".

Hinweis: Der SMS-Text wird mit Strg-Z (0x1A) beendet. Antworten des Moduls und mögliche Fehler sind in der folgenden Tabelle aufgeführt:

| Antwort | Bedeutung | Aktion |
|---------------------------------------|---|---|
| +CMGS: <nr> OK | Nachricht erfolgreich gesendet. <nr>=nachrichten referenz nummer | |
| ERROR | Error aufgetreten | Erweiterte Fehlerkodierung einschalten und noch einmal probieren. |
| +CMS ERROR: 330 | SMSC Adresse unbekannt | Korrekte SMSC Adresse eingeben |
| +CMS ERROR: 41 | Device ist nicht im Netzwerk registriert | Signalstärke und Netzwerkregistrierung überprüfen |
| +CMS ERROR: 331 | Kein Netzwerkservice | Signalstärke und Netzwerkregistrierung überprüfen |
| +CMS ERROR: 1 | Ungültige Nummer | Die Telefonnummer existiert nicht. Prüfen und wiederholen |
| +CMS ERROR: 42 | Netzwerk überlastet | Später noch einmal probieren |
| +CMS ERROR: 96 | Wichtige Information fehlt | Telefonnummer prüfen |
| OK | Abbruch durch Nutzer | |

Tabelle 13: GSM-Modul-Rückmeldungen bei Versand einer SMS

6.7.5.3. Hinweise zum GPRS-Modus

Wurde das Modul wie in Abschnitt 4.4.3 im GPRS-Modus konfiguriert ist ein gleichzeitiger Zugriff auf die GSM-Funktionalität nicht vorgesehen.

7. Erweiterungsplatinen

Für das piA-AM35x stehen verschiedenste Erweiterungsplatinen zur Erweiterung des Funktionsumfangs zur Verfügung. Es existieren die Module

- piA-Wireless
- piA-LCD
- piA-ChargeControl
- piA-Motor
- piA-IO

Alle Tochterkarten sind vollständig kompatibel zum Basissystem.

7.1. piA-Wireless

Dieses Zusatzmodul erweitert den Funktionsumfang des piA-AM35x um diverse drahtlose Techniken: WLAN, Bluetooth, RFID. Außerdem sind ein 3-Kanal-AD-Wandler und diverse digitale Ein- und Ausgänge verfügbar.

7.1.1. RFID

TODO

7.1.2. WLAN/BT

Das integrierte WLAN/BT-Modul der Erweiterungskarte piA-Wireless wird automatisch erkannt und als Netzwerkdevice ins Linux-System integriert.

Die Programmierung für WLAN erfolgt dann über die in Abschnitt 6.5 angesprochene Socket-API.

7.1.2.1. Verbindung zu einem offenen Access Point herstellen

Zuerst muss dem WLAN-Interface eine eigene MAC Adresse gegeben werden:

```
ifconfig wlan0 hw ether 08:00:28:00:00:<number>
ifconfig wlan0 up
```

Ein Scan auf der Schnittstelle wlan0 geht mittels „iw-tool“:

```
iw wlan0 scan
```

Danach kann die Verbindung zu dem offenen Netz hergestellt werden:

```
iw wlan0 connect -w OpenLink
dhclient wlan0
ping 192.168.1.1
```

Verbindung trennen:

```
iw wlan0 disconnect
```

7.1.2.2. Verbindung zu einem WEP AP

Verbindung mit einem vordefinierten Schlüssel herstellen:

```
iw wlan0 connect -w OpenLinkWEP key 0:00deadbeef
```

Für die Verbindung mittels wpa_supplicant muss zuerst die Konfigurationsdatei wep.conf angepasst werden:

```
ctrl_interface=/var/run/wpa_supplicant

network={
    ssid="OpenLinkWEP"
    scan_ssid=1
    key_mgmt=NONE
    wep_key0=00deadbeef
}
```

Nun kann man sich per Konsole mit dem Netz verbinden:

```
wpa_supplicant -B -i wlan0 -c wep.conf
wpa_cli status
dhclient wlan0
ping 192.168.1.1
```

7.1.2.3. Verbindung zu einem WPA2 AP

WLAN konfigurieren (MAC Adresse zuweisen):

```
ifconfig wlan0 hw ether 00:11:22:33:44:55
```

Verbindung mittels wpa_supplicant herstellen:

```
wpa_supplicant -i wlan0 -D wext -c /etc/wpa_supplicant.conf -B
```

Status mittels iw-tools anzeigen lassen:

```
iw wlan0 link
```

Für wpa_supplicant muss der ASCII-Zeichen Schlüssel noch in Hexadezimalform umgewandelt werden. Die Berechnung erfolgt mittels wpa_passphrase. Dem Befehl werden die SSID des Access-Points und der entsprechende Schlüssel übergeben:

```
wpa_passphrase MeineSSID MeinSchluessel
```

ergibt

```
network={
    ssid="MeineSSID"
    #psk="MeinSchluessel"
    psk=1f6d1d5393464f5bf78acd7b8573459f63128a4cbdca767f9d6c9890b100dee9
}
```

Diese Werte müssen dann zusätzlich zu der Konfiguration für die WLAN-Verbindung in die wpa_supplicant.conf eingetragen werden.

Für Bluetooth stellt Linux einen eigenen Stack unter der Bezeichnung bluez4 zur Verfügung, der die unterschiedlichsten Bluetooth-Funktionalitäten unterstützt.

TODO: BT

7.1.2.4. Treiber/Firmware

Es werden 2 binäre Firmwaredateien benötigt, welche für Linux unter <http://git.kernel.org/?p=linux/kernel/git/dwmw2/linux-firmware.git> zu finden sind.

WLAN:

ti-connectivity/wl1271-fw.bin
wird im Verzeichnis `/lib/firmware/ti-connectivity/` gesucht

BT:

ti-connectivity/TIInit_7.2.31.bts
im Verzeichnis `/lib/firmware/`

7.1.3. AD-Wandler

Auf der Erweiterungskarte piA-Wireless befindet sich der AD-Wandler AD7993, welcher 4 Kanäle mit jeweils 10bit Auflösung bereitstellt.

Das SDK für das piA-AM35x enthält eine Bibliothek zur Konfiguration und Ansteuerung des AD-Wandlers. Da die Kommunikation des AD-Wandlers über den I²C-Bus erfolgt, muss vor dem Verwenden der mitgelieferten Bibliothek die entsprechende Schnittstelle geöffnet und ein 'Handler' bereitgestellt werden (siehe Abschnitt 6.4.2). Weiterhin benötigt die Bibliothek eine Device-Struktur für die interne Speicherung der Konfiguration.

```
/* AD7993 device structure */
ad7993_tad7993;
```

Der AD-Wandler wird wie folgt mit Standardeinstellungen initialisiert:

```
ad7993_dev_init(&ad7993, i2c_handler);
```

Die Parameter sind die Device-Struktur und der I2 C-Handler.

Nun kann mittels

```
int res = single_measurement(channel);
fprintf(stdout, "Single conversion: vin=%d\n", res);
```

eine einzelne AD-Wandlung gestartet und das Ergebnis ausgegeben werden. Die Funktion `single_measurement()` erwartet als Parameter den zu messenden Kanal als Integer-Wert (z.B. 1 – 4). Der Rückgabewert ist das Ergebnis der Analog-Digital-Konvertierung.

Eine fortlaufende Konvertierung mit einer Frequenz von 15625 Hz ist ebenfalls möglich:

```
/* start conversion with fixed frequency */
start_cycle_measurement("15625", channel);

while(1) {
    /* read result out of conversion result register */
    res = ad7993_read_conv_res();
    ch = identify_channel(res);
    v_in = calculate_voltage(res);
    fprintf(stdout, "channel: %d, V = %d.%d\n", ch, v_in/1000, v_in%1000);
}
```

7.2. Erweiterungsplatine piA LCD

TODO Dokumentation zu Tochterkarten