Hochschule
Augsburg University of
Applied Sciences

Embedded Linux

Thema:
Entwurf eines Zugangskontrollsystems
unter Verwendung eines Fingerabdrucks
zur Personenidentifikation

Albert Böswald
albert.boeswald@fh-augsburg.de

Sommersemester 2011

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Überblick

Unter Verwendung eines embedded Linux wurde ein Demonstrationssystem aus verschiedenen Hardwarekomponenten entwickelt, welches den Zutritt zu Räumlichkeiten anhand eines Fingerabdrucks gewährt. Die Elektronik ist in der Lage, Personen anhand Ihres Fingerabdrucks zu identifizieren. Ist die Identifikation einer Person gegen eine existente Datenbank positiv, so treibt das System für einen kurzen Moment eine definierte GPIO Leitung. Diese Leitung ist mit einem Schloss verbunden, welches in diesem Fall öffnet. Die grobe Skizze des Gesamtsystems ist in Abb. 1.1 dargestellt.

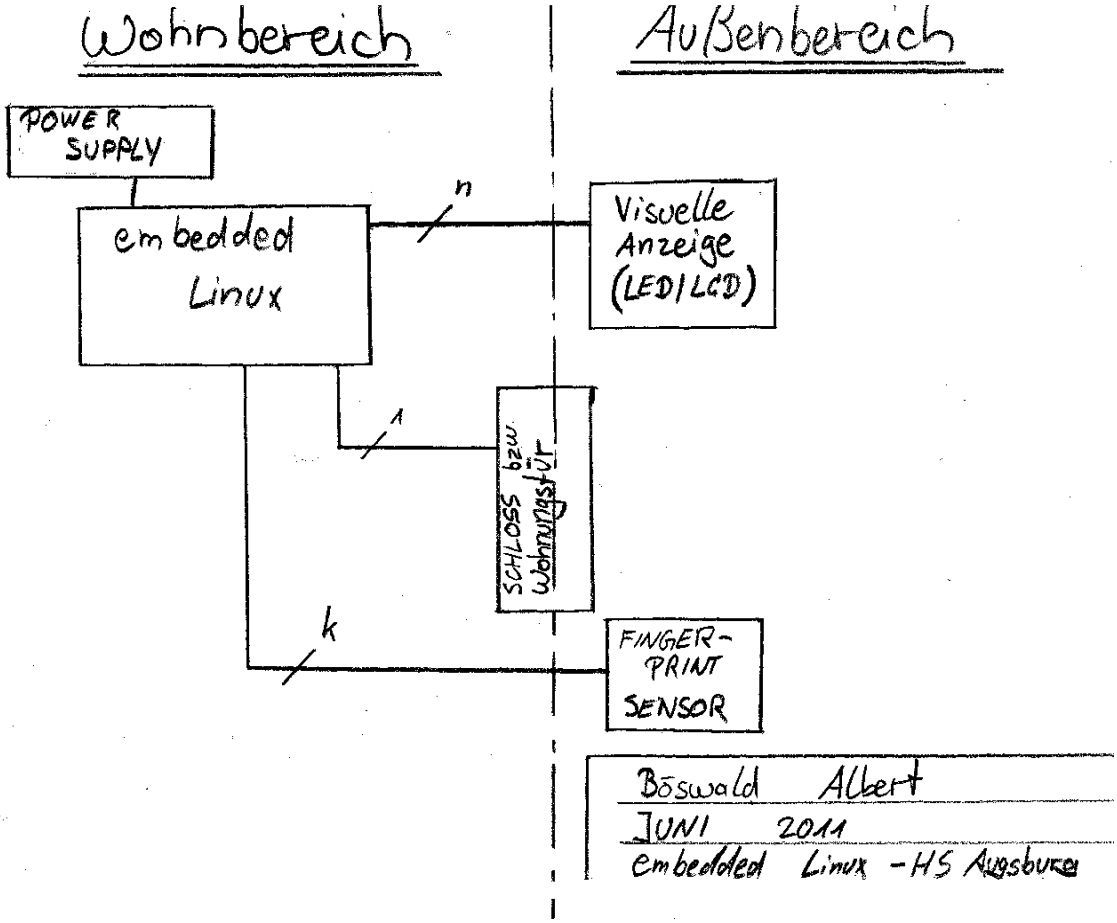

Abb. 1.1: Grobes Blockschaltbild des Gesamtsystems

2 Hardware

2.1 FriendlyARM mini2440

Um das Projekt durchführen zu können, musste ein geeignetes Target für die Aufgabe gefunden werden. Verschiedenste Aspekte können in einem realen Anwendungsfall Entscheidungshilfen geben. Das Preis-Leistungs-Verhältnis des mini2440 von FriendlyARM stach sofort ins Auge.
Auf dem mini2440 ist der Samsung S3C2440A Mikrocontroller integriert. Innerhalb dieses Controllers arbeitet eine ARM920T CPU, deren Takt 400MHz ist.
Vor allem das 3,5Zoll LCD ist für die hier angestrebte Anwendung hervorragend geeignet, denn damit werden komplexere Statusausgaben sehr komfortabel an den Benutzer ausgegeben. Bei vielen embedded Linux Boards stehen nur GPIO Leitungen bzw. LEDs zur Verfügung, die die Statusausgabe erschweren.
Weiter hat das Board eine sehr große Open-Source-Gemeinde und ist weit verbreitet, daher können über diesen Kanal sehr oft Problemlösungen gefunden werden.

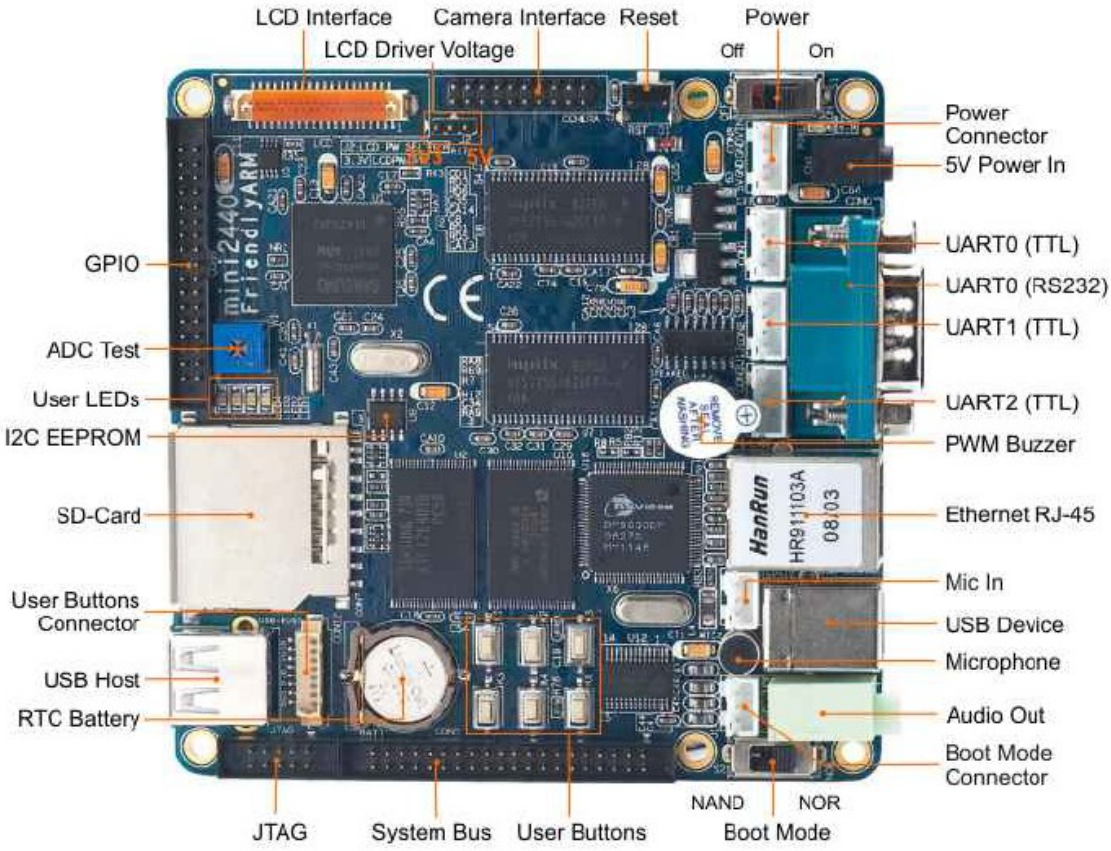

Abb. 2.1: FriendlyARM mini2440 (Abbildung ohne 3,5“ LCD)

2.2 TopSec ID Module

Das TopSec ID Module der MB fingerMetrica GmbH ist eine Fingerabdruckerkennungseinheit mit Anschlussmöglichkeit für verschiedenste Fingerprintsensoren. Es enthält komplexe Algorithmen um eingelesene biometrische Daten in zuverlässiger Form auf dem onboard Flash zu speichern. Aus Sicherheitsgründen ist die Implementierung dieser Algorithmen geschlossen. Ein Rückschluss auf den Fingerabdruck ist, aus den im Flash gespeicherten Daten, nicht möglich.

Alle notwendigen biometrischen Funktionen sind bereits im TopSec ID Module implementiert. Die Enrollment Prozedur fügt einen neuen Finger in die Datenbank des TopSec hinzu, während die Identify Prozedur einen aufgelegten Finger mit eben dieser Datenbank abgleicht. Leider ist bislang kein quelloffener Treiber für dieses Modul bekannt, weswegen im Rahmen dieser Arbeit ein solcher entwickelt wurde (siehe Kapitel 3.1 TopSec Gerätetreiber).

Das TopSec ID Module, ein passender kapazitiver Sensor (siehe Abb. 2.2), sowie ein Interfaceboard zum Abgreifen aller Pins (TopSec SDK, siehe Abb. 2.3) wurden freundlicherweise von der MB fingerMetrica GmbH für dieses Projekt zur Verfügung gestellt.

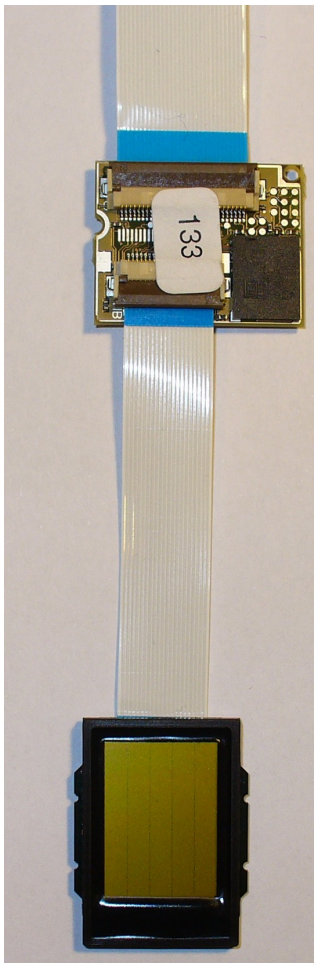

Abb. 2.2: TopSec ID Module mit angeschlossenem kapazitivem Sensor

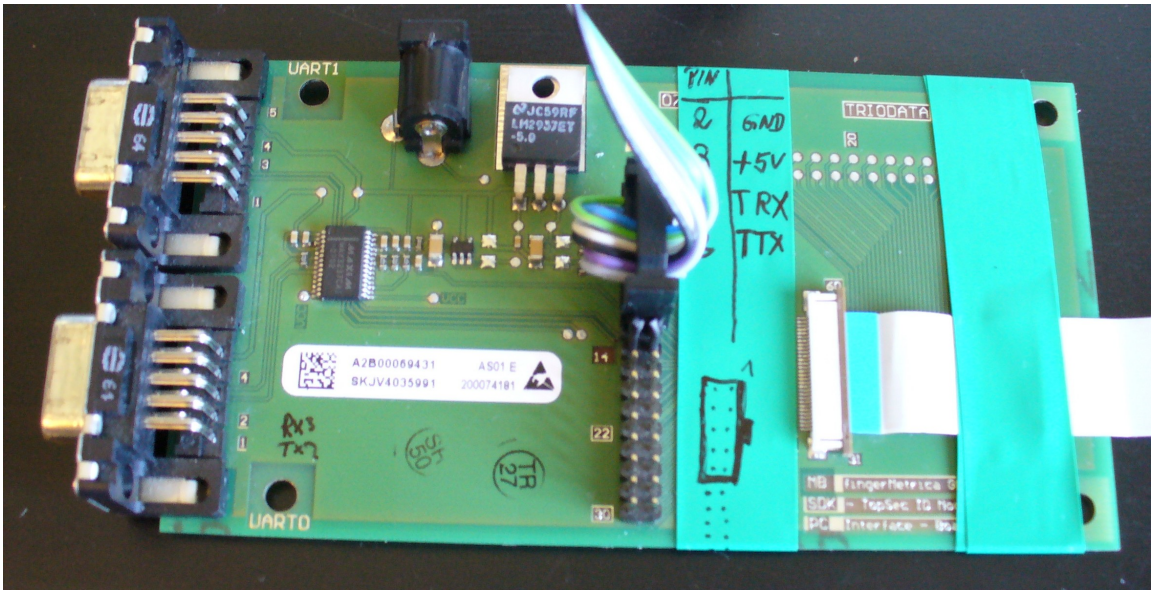

Abb. 2.3: TopSec SDK

Zur Kommunikation zwischen dem TopSec und dem FriendlyARM mini2440 wurde die beidseitig vorhandene UART Schnittstelle verwendet. Da die Fingerprinteinheit mit 5V TTL und das mini2440 mit 3,3V TTL arbeitet, ist auf einem selbst entworfenem Adapterboard (siehe Kapitel 2.4 Adapterboard) eine Hardware zur Pegelanpassung des UART aufgebaut worden.

2.3 Schloss

Ein mechanisches Schloss mit einem Hubmagneten (siehe Abb. 2.4) simuliert das Verhalten der Verriegelungsmechanik einer Haustüre. Liegen 12V über der Spule des Schlosses an, so zieht der Bolzen (Abb. 2.4 [B]) nach oben und ein Federmechanismus öffnet das Schloss. Ein im Schloss integrierter Schalter (Abb. 2.4 [A]) ist in der Lage, den aktuellen Zustand des Schlosses nach außen hin zu melden. Legt man an die beiden Eingänge dieses Schalters sowohl 3,3Volt als auch GND an, so ließe sich über den Ausgang des Schalters jeweils „geschlossen“ oder „offen“ als Zustand des Schlosses an einer GPIO Leitung des mini2440 auslesen. Dieses Verhalten wurde durch Tests bestätigt, jedoch ist es für diese Anwendung nicht relevant, weswegen der Zustandsschalter nicht weiter berücksichtigt wurde.

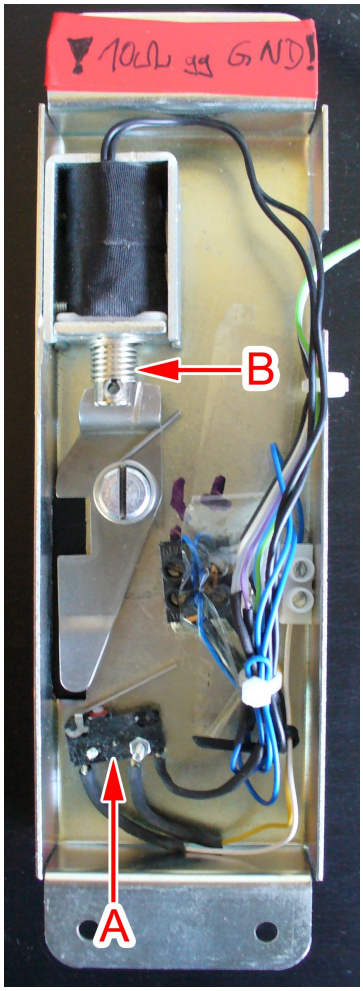

Abb. 2.4: Schloss zur Simulation einer Haustür

Da das Schloss aus verschiedenen Gründen (Leistungsbedarf, Betriebsspannung, etc.) nicht direkt an eine GPIO Leitung des mini2440 angeschlossen werden kann, wurde auch für diese Komponente eine Anpassung auf dem Adapterboard (siehe Kapitel 2.4) aufgebaut.

2.4 Adapterboard

Abb. 2.5 zeigt das für die Anwendung entworfene Adapterboard. Auf der linken Seite des Boards sind GPIO und UART Schnittstellen zum mini2440 angebracht. Die Signale werden auf der Platine gewandelt und auf der rechten Seite mit dem Schloss bzw. der Fingerabdruckeinheit verbunden. Ein zusätzlicher Taster ist in der Mitte des Boards angebracht. In der Regel ist das System stets damit beschäftigt, aktuelle Bildinformationen mit der Datenbank zu vergleichen, um gegebenenfalls die Tür zu öffnen. Der Use-Case „Hinzufügen eines neuen Fingers in die Datenbank“ kann mit Hilfe des Tasters angestoßen werden. Drückt und hält man ihn für etwa eine Sekunde, so wechselt das System den Enroll-Zustand und nimmt einen neuen Finger in die Datenbank auf. Entsprechende Anzeigen am LCD des mini2440 signalisieren dies dem Benutzer.

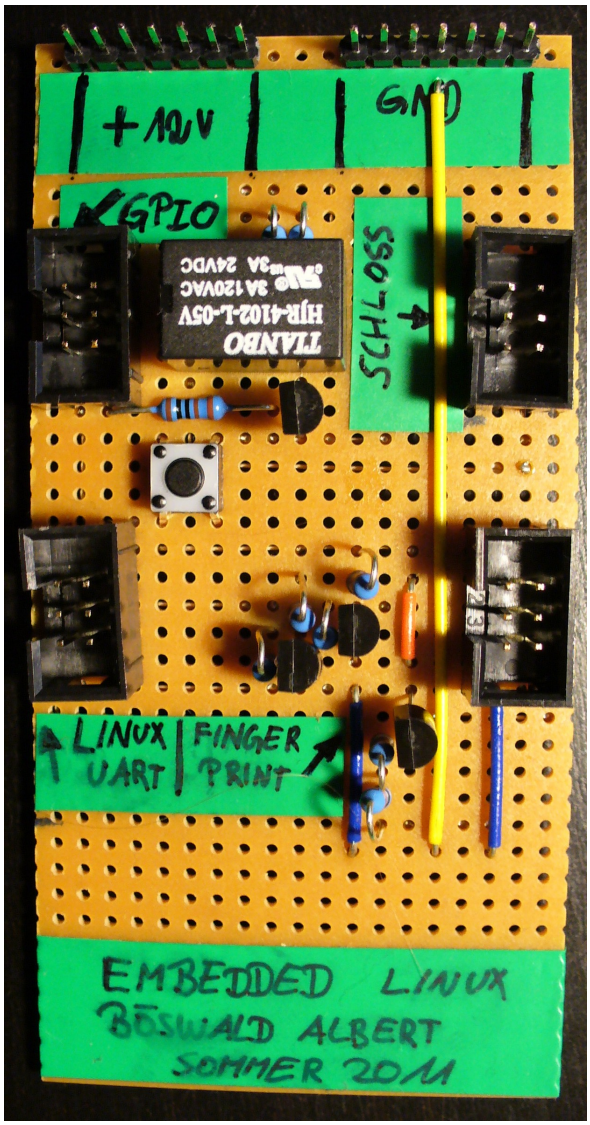

Abb. 2.5: Adapterboard

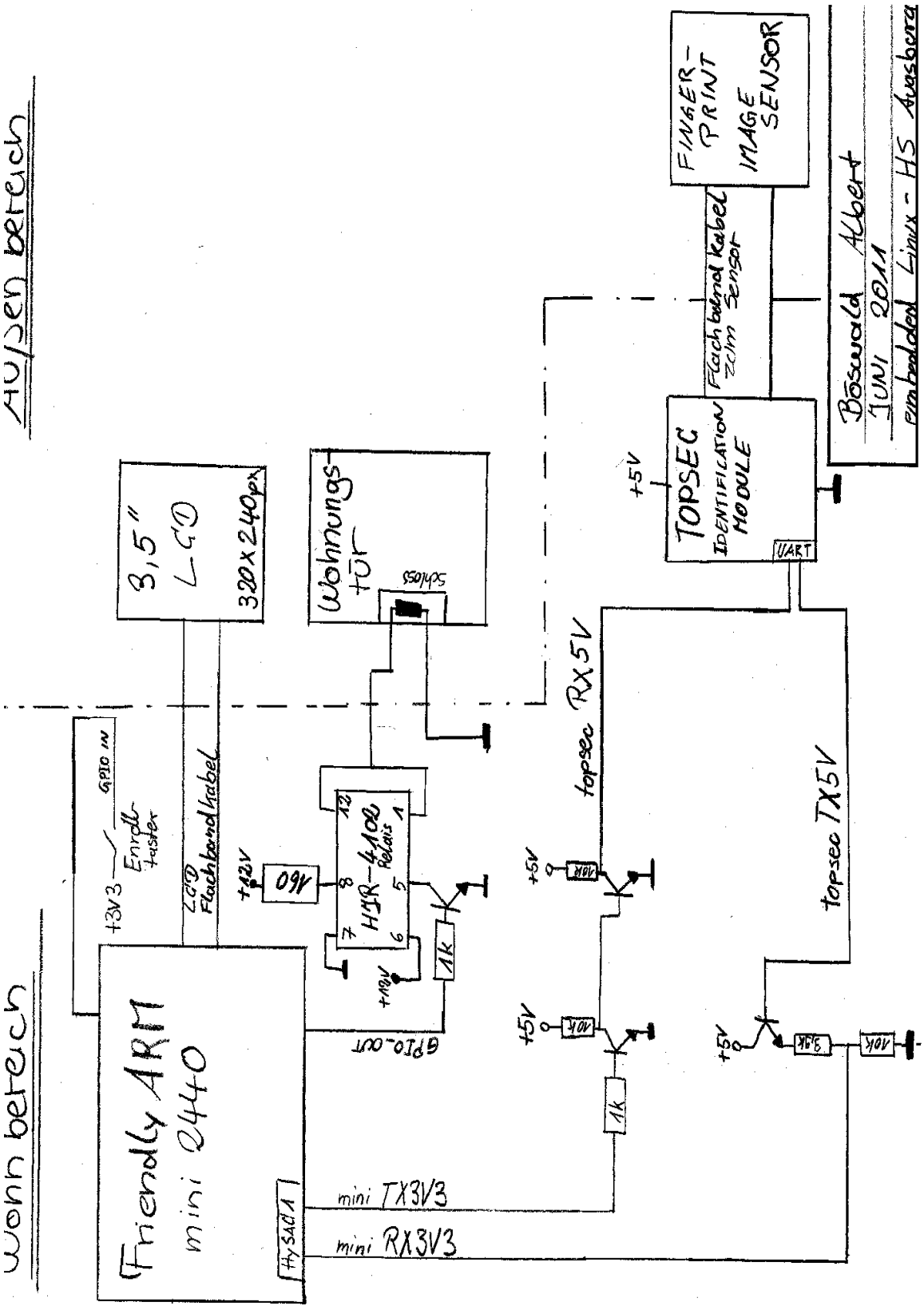

Abb. 2.6: Stromlaufplan und Blockschaltbild des Gesamtsystems

2.5 Gesamtsystem

Verbindet man alle beschriebenen Komponenten aus dem Kapitel 2, so ergibt sich ein Demonstrationsaufbau wie in Abb. 2.7.

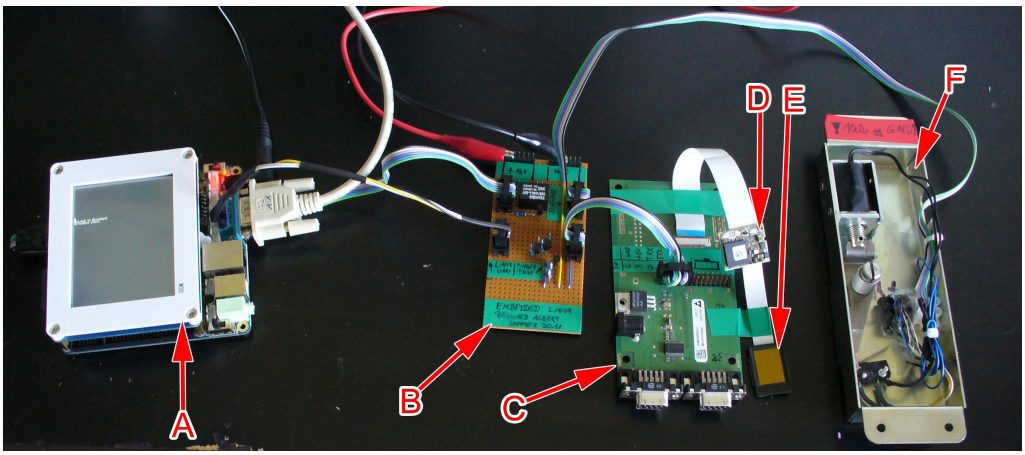

Abb. 2.7: Das aufgebaute Gesamtsystem

Verwendete Hardwarekomponenten:

A) FriendlyARM mini2440
B) Adapterboard
C) TopSec SDK
D) TopSec ID Module
E) Fingerprint Sensor
F) Schloss

3 Software

3.1 TopSec Gerätetreiber

Der Gerätetreiber für das TopSec ID Module wurde an einer x86 Maschine unter Ubuntu entwickelt. Mit einem USB-TTL (5V) Kabel der Firma FTDI-Chip konnte eine Verbindung zu der Hardware unter sehr geringem Aufwand hergestellt werden. Der Gerätetreiber des TopSec wurde wegen seiner hohen Komplexität in drei verschiedenen Schichten entwickelt, welche ab Kapitel 3.1.5 näher erläutert werden.

3.1.1 TopSec ID Module Kommunikationsprotokoll

Jede Kommunikation von und zum TopSec ID Module ist in einzelne Pakete untergliedert. Ein Paket besteht aus einem Header, Daten und einer Prüfsumme. Während Header und Prüfsumme stets eine konstante Länge besitzen kann die Länge des Datenfeldes variieren.

3.1.2 TopSec Packet Header

Der TopSec Packet Header (stets 16 Byte lang) ist nachfolgend dargestellt.

Low Byte	High Byte
Start of Header (0x55)	Version (0x20)
Packet Number	Number of Packets
OpCode	
Data Length	
Connection Status	Connection ID
Reserved 1	Reserved 2
Reserved 3	End of Header (0xAA)
Checksum	

Abb. 3.1: TopSec Packet Header

Name	Länge (Byte)	Inhalt
`Start of Header`	1	0x55 (fixed)
`Version`	1	0x20 (fixed)
`Packet Number`	1	Paketnummer bei Mehrpaketübertragung
`Number of Packets`	1	Summe aller Pakete bei Mehrpaketübertragung
`OpCode`	2	Befehl (von und zum TopSec ID Module)
`Data Length`	2	Länge der nachfolgend zu erwartenden Daten
`Connection Status`	1	Aktiviert den Check der Connection ID
`Connection ID`	1	Die aktuelle Verbindungsnummer
`Reserved1`	1	0x00 (fixed)
`Reserved2`	1	0x00 (fixed)
`Reserved3`	1	0x00 (fixed)
`End of Header`	1	0xAA (fixed)
`Checksum`	2	16-Bit breites XOR der ersten 14Byte des Headers

3.1.3 TopSec Packet Data & Packet Data Checksum

Die Anzahl der Bytes, die im Datenfeld übertragen werden, müssen stets ein vielfaches von zwei sein. Befindet sich im Feld „Data Length“ des Headers 0x00, wird kein Packet Data und keine Packet Checksum übertragen. Die Packet Checksum wird über eine 16-Bit breite XOR-Verknüpfung der Packet Data errechnet. Dieser Vorgang ist ähnlich zur Prüfsummenberechnung beim TopSec Header.

3.1.4 Kommunikation

In der Regel findet die Kommunikation zwischen den Geräten gemäß Abb. 3.2 satt. Der Host sendet einen Befehl an das ID Module, dass den Empfang mit einem Acknowledge Paket quitiert. Das ID Module führt den Befehl aus und meldet sich nach einiger Zeit mit dem Ergebnis der Ausführung zurück.

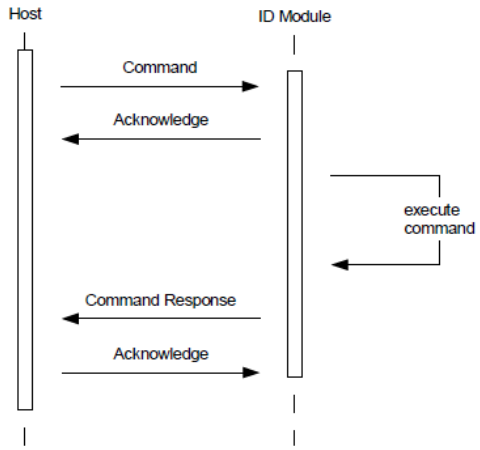

Abb. 3.2: Allgemeine Kommunikationssequenz

Grundsätzlich findet diese Kommunikationsmethode immer Anwendung, doch existieren aufgrund der hohen Flexibilität des TopSec ID Moduls auch äußerst komplexe Kommunikationspfade, die in dem Treiber korrekt abgefangen werden. Dies ist beispielsweise beim Enrollment einer neuen Person in die Datenbank der Fall (siehe Abb. 3.3).

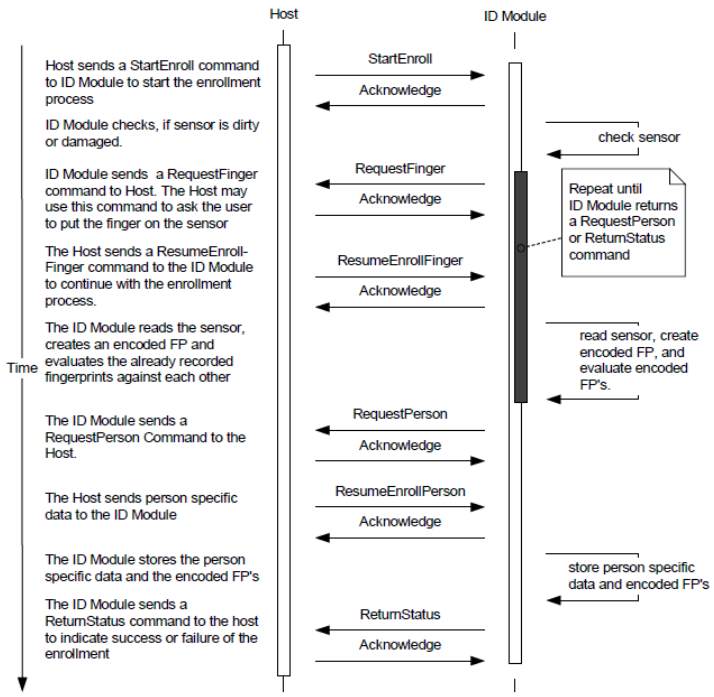

Abb. 3.3: Komplexe Kommunikationssequenz – Einlernen einer neuen Person

3.1.5 Level0 Treiber: UART-Interface

Der Level0 Treiber stellt elementarste Funktionen für die Nutzung der UART unter Linux zur Verfügung. Grundsätzlich bietet dieser Treiber 5 verschiedene Programmschnittstellen zur UART, welche in den Dateien uart.h und uart.c implementiert sind:

Funktionsname	Aufgabe
`uart_open(...);`	- Öffnen der übergebenen UART-Schnittstelle
`uart_close(...);`	- Schließen der übergebenen UART-Schnittstelle
`uart_tx(...);`	- Übertragen von Daten (nicht blockierend)
`uart_rx(...);`	- Empfangen von Daten (nicht blockierend)
`uart_rx_time_block(...);`	- Empfangen von Daten (blockierend für eine bestimmte Zeit)

3.1.6 Level1 Treiber: TopSec-Packet-Interface

Der Level1 TopSec Treiber ist in den Dateien driver_topsec.h und driver_topsec.c implementiert. Die wichtigsten Funktionen können anhand von entsprechenden Übergabeparametern komplette TopSec Pakete zusammenstellen und dieses mit Hilfe des Level0 Treibers über die physikalische Schnittstelle übertragen.

Funktionsname	Aufgabe
`topsec_tx_packet(...);`	- Übertragen eines Pakets
`topsec_rx_packet(...);`	- Empfangen eines Pakets
`topsec_open_connection();`	- Öffnen der TopSec Verbindung
`topsec_close_connection();`	- Schließen der TopSec Verbindung

3.1.7 Level2 Treiber: TopSec-Application-Interface

Denkt man an die Qt Anwendung, die in der höchsten Abstraktionsebene arbeitet, so ist der Level1 Treiber noch immer zu fein strukturiert. Auf dieser Ebene ist es sinnvoll, Funktionen aufzurufen, die eine ganze Kette an Paketen senden und Empfangen und auf eventuelle Fehlerfälle mit entsprechenden Rückgabewerten reagieren. Ebenso abstrakt wie Grafikfunktionen implementiert sind, möchte man auch mit dem TopSec kommunizieren. Diesen Anforderungen ist der Level2 Treiber mit folgenden Funktionen gewachsen.

Funktionsname	Aufgabe
`topsec_enroll_person(...);`	- Einlernen einer neuen Person
`topsec_delete_person(...);`	- Löschen einer Person
`topsec_identify();`	- Identifizieren des Fingers mit der Datenbank
`topsec_init();`	- Initialisieren des TopSec
`topsec_close();`	- Beenden der Verbindung

Die einzelnen Funktionen liefern detaillierte Rückgabewerte, welche präzise Auskunft über die Art des Fehlers geben. Als Beispiel kann die Funktion topsec_identify() unter anderem Folgende Rückgabewerte erzeugen:

#define	Bedeutung
`RET_TOPSEC_TRANSMITERROR_ACKEXPECTED`	Übertragungsfehler.
`ERRORAR_NO_ARCHIVE_EXISTING`	Identifizierung kann nicht stattfinden, weil keine Datenbank vorhanden ist.
`ERRORLEAD_AREA_COVERED_LOW`	Der aufgelegte Finger bedeckt eine zu geringe Sensorfläche
`ERRORCMD_NO_MATCH`	Der Vorgang war erfolgreich, aber es wurde keine passende Referenz in der Datenbank gefunden.
`...`	...

3.2 Toolchain, RootFS und Qt Libraries

Es wird das Konzept des Cross-Kompilierens angewandt. Es bedeutet, dass auf einer Hostarchitektur (für gewöhnlich x86) die Softwarequellen direkt für eine Zielarchitektur (z.B. ARM) kompiliert werden. Das Kompilat ist daher nicht auf der Hostarchitektur ausführbar. Für diese Aufgabe wird ein Compiler benötigt, der auf dem Hostsystem ausgeführt werden kann, aber Programme für eine andere Zielarchitektur erzeugt. Diese Art von Compiler nennt man Cross-Compiler. Natürlich sind wesentlich mehr Schritte als nur das Kompilieren notwendig, um aus einem Quellcode Maschinencode zu generieren. Gerne spricht man aber nur vom Cross-Compiler, während man aber eigentlich eine ganze Sammlung an Cross-Programmen meint. Diese Sammlung nennt man Cross-Compilation-Toolchain.

Hinweis: *Nachfolgendes Vorgehen wurde getestet mit folgenden Versionsständen:*
buildroot Version 2011.05-git-00178-g7e3e8ec
buildroot Version 2011.08-git-00190-g9dc7b73

Buildroot ist in der Lage eine Cross-Compilation-Toolchain sowie ein RootFilesystem zu bauen, welches beim Kernel-boot von diesem gemountet wird.
Dazu wurde zunächst Buildroot aus dem aktuellen GIT Repository heruntergeladen.

Beschaffen von Buildroot und Vorbereiten des Host-Systems

```
sudo apt-get install libncurses5-dev bison g++ flex gettext texinfo patch
git-core libtool autoconf subversion

git clone git://git.buildroot.net/buildroot
```

Starten der Konfiguration von Buildroot:

```
cd buildroot
make menuconfig
```

Um Buildroot für das mini2440 zu konfigurieren, sind einige Einstellungen notwendig.
Die besondere Bedeutung der Flags kann im Internet oder in der Buildroot eigenen Hilfe nachgeschlagen werden.

Name	Wert
Target Architecture	ARM
Target Architecture Variant	arm920t
Toolchain	
Enable MMU support	aktiv
Use Software Floating point by default	aktiv
Enable large file > 2GB support	aktiv
Enable WCHAR support	aktiv
Enable C++ support	aktiv
System Configuration	
Port to run a getty on	ttySAC0
Package selection for the target,	
Graphic libraries and applications (graphic/text),	
Qt	
Compile with debug support	aktiv
Approve free licence	aktiv
JPEG – Use Qt bundeled libjpeg	aktiv
Graphics Drivers	
Linux Framebuffer	aktiv
Transformed	aktiv
multiscreen	aktiv
Mouse Driver	
PC	aktiv
tslib	aktiv
hide the mouse cursor	aktiv

Das Bauen des konfigurierten Buildroot-Systems kann gestartet werden.

```
make
```

3.3 Sourcen der Toolchain

Bevor der Vorgang des Cross-Kompilierens funktionieren kann, muss die Toolchain dem Hostsystem bekannt gemacht werden. Weil oftmals eine Toolchain auf dem System gewechselt werden muss, ist es am Besten man „sourced“ eine Datei, die die Informationen über gewünschte Toolchain bekannt macht:

mini2440_buildroot_toolchain.sh

```
export PATH="$PATH:/home/boeswalda/mini2440/buildroot/buildroot/output/host/usr/bin"
export CROSS_COMPILE=arm-linux-
export ARCH="arm"
```

Vor dem Kompilieren des U-boots und des Kernels ist das Kommando *source /home/boeswald/mini2440/buildroot/mini2440_buildroot_toolchain.sh* in der Bash abzugeben. Die Toolchain ist dann für die aktuelle Bash bekannt.

3.4 Bootloader „Das U-Boot“

Hinweis: *Nachfolgendes Vorgehen wurde getestet mit der U-Boot Version 1.3.2-mini2440 aus dem Repository von http://repo.or.cz/r/u-boot-openmoko/*

Beschaffung des Bootloaders

```
mkdir u-boot
cd u-boot/
git clone git://repo.or.cz/u-boot-openmoko/mini2440.git
```

Das U-Boot soll den Kernel von der SD-Karte laden und ausführen. Dies kann als Standardkonfiguration bereits mit ins U-Boot eingefügt werden. Dazu muss die Datei „include/configs/mini2440.h“ wiefolgt geändert werden:

Diff der mini2440.h aus dem Git Repository mit der verändertern Version:

```
boeswalda@boeswalda-VirtualBox:~/diff mini2440git.h mini2440.h

62c62
< #define CONFIG_MINI2440_OVERCLOCK 1
---
> #define CONFIG_MINI2440_OVERCLOCK 0

138,139c138,139
< #define CONFIG_BOOTDELAY      3
< #define CONFIG_BOOTARGS       "root=/dev/mtdblock3 rootfstype=jffs2
console=ttySAC0,115200"
---
> #define CONFIG_BOOTDELAY      1
> #define CONFIG_BOOTARGS       "console=ttySAC0,115200 root=/dev/mmcblk0p1
rw rootwait mini2440=4tb init=/sbin/init fbcon=rotate:1"

145c145
< #define CONFIG_BOOTCOMMAND    ""
---
> #define CONFIG_BOOTCOMMAND    "mmcinit; ext2load mmc 0:1 0x31000000
uImage; bootm;"

159c159
< #define       CFG_PROMPT              "MINI2440 # "
---
> #define       CFG_PROMPT              "TUEROEFFNER# "

208,209c208,209
< #define CONFIG_USBD_MANUFACTURER      "MINI2440"
< #define CONFIG_USBD_PRODUCT_NAME      "S3C2440 Bootloader " U_BOOT_VERSION
---
> #define CONFIG_USBD_MANUFACTURER      "TUERO"
> #define CONFIG_USBD_PRODUCT_NAME      "TUERO-Bootloader" U_BOOT_VERSION
```

Die wichtigen Veränderungen sind an CONFIG_BOOTCMD (Eine Art Script, die vom U-Boot beim Start nacheinander ausgeführt wird) und an CONFIG_BOOTARGS (Übergabeparameter an den Kernel) vorgenommen worden:

CONFIG_BOOTCMD:

mmcinit	Initialisieren der SD-Karte
ext2load mmc 0:1 0x31000000 uImage	Lädt aus dem EXT2 Dateisystem der SD Karte 0 / Partition 1 die Datei uImage (Kernel) an die Adresse 0x31000000 (SDRAM)
bootm	Startet den Kernel

CONFIG_BOOTARGS:

console=ttySAC0,115200	Kernelausgabe, tty
root=/dev/mmcblk0p1 rw	Pfad zum RootFilesystem
rootwait	Warte (forever) auf das RootFileSystem
mini2440=4tb	LCD Konfiguration
init=/sbin/init	Pfad zum Init Prozess
fbcon=rotate:1	Framebuffer Konfiguration – LCD um 90° drehen

Bauen des Bootloaders

```
source ../../buildroot/mini2440_buildroot_toolchain.sh
make mini2440_config
make
```

Die am Ende des Prozesses entstandene Datei *u-boot.bin* beinhaltet den Bootloader für das Target und muss nun auf dieses übertragen werden.

3.5 Upload des Bootloaders auf das Target

Ist der Bootloader gebaut, so ist er im nächsten Schritt auf das Target zu übertragen. Das mini2440 Development-Board führt eine JTAG-Schnittstelle auf einen 10-poligen Wannenstecker mit 2,00mm Pitch nach Außen (siehe Abb. 3.4). Damit kann z.B. die Software OpenOCD mit einem entsprechenden Interface verwendet werden, um direkt über die CPU die Daten in den NAND- oder NOR Flash zu übertragen.

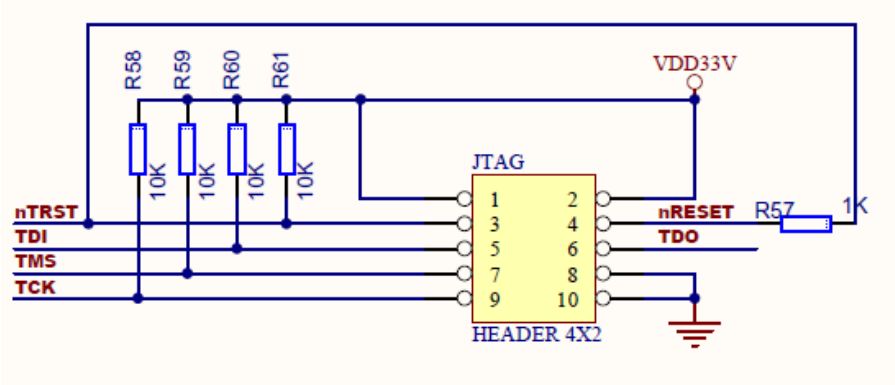

Abb. 3.4: Schematics der JTAG-Schnittstelle

Bei der Auslieferung des mini2440 ist zusätzlich die Software „Supervivi“ als Bootloader im NOR Flash abgelegt. Schaltet man den dafür vorgesehenen Hardwareswitch auf dem Board in NOR Position, so bootet Supervivi und bietet die Möglichkeit über eine USB Verbindung den NAND Flash mit dem U-Boot zu bespielen. Man benötigt dazu:

1) Ein USB A/B Kabel
2) Das geladene Supervivi auf dem Target, sowie eine Consolenverbindung dorthin
3) Die opensource Software „s3c2410_boot_usb“
4) Die Größe der Datei u-boot.bin („ls -la“ hier: 248988 Bytes)

Das Übertragen der Datei in den NAND Flash kann wiefolgt durchgeführt werden:

Supervivi Bootloader

```
boeswald@Si1520Linux:~$ sudo picocom /dev/ttyUSB0 -b 115200
picocom v1.4
[...]
Terminal ready


##### FriendlyARM BIOS 2.0 for 2440 #####
[x] format NAND FLASH for Linux
[v] Download vivi
[k] Download linux kernel
[y] Download root_yaffs image
[a] Absolute User Application
[n] Download Nboot for WinCE
[l] Download WinCE boot-logo
[w] Download WinCE NK.bin
[d] Download & Run
[z] Download zImage into RAM
[g] Boot linux from RAM
[f] Format the nand flash
[b] Boot the system
[s] Set the boot parameters
[u] Backup NAND Flash to HOST through USB(upload)
[r] Restore NAND Flash from HOST through USB
[q] Goto shell of vivi
[i] Version: 1026-2K
Enter your selection:
```

Während das USB Kabel getrennt ist, initiiert man die Upload Sequenz am Target:

```
Enter your selection: q
Supervivi> load flash 0 248988 u
USB host is not connected yet.
```

Steckt man nun das USB Kabel an, zeigt das Target das Warten auf Daten an:

```
USB host is connected. Waiting a download.
```

Am Host wird die Übertragung mit *s3c2440_boot_usb* gestartet:

```
boeswald@Si1520Linux:~/$ sudo ./s3c2410_boot_usb/s3c2410_boot_usb u-boot.bin
csum = 0x213c
send_file: addr = 0x33f80000, len = 0x0003cc9c
Error downloading program
boeswald@Si1520Linux:~/$
```

Am Target wird das erfolgreiche Empfangen des Binaries quitiert.

```
Now, Downloading [ADDRESS:30000000h,TOTAL:248998]
RECEIVED FILE SIZE:  248998 (121KB/S, 2S)
Downloaded file at 0x30000000, size = 248988 bytes
Found block size = 0x00040000
Erasing...    ... done
Writing...    ... done
Written 248988 bytes
Supervivi>
```

Nachdem das Board nun ausgeschaltet, der Bootschalter wieder in NAND Position gebracht und das Board wieder angeschaltet wird, sollte das Target nun vom NAND Flash das U-Boot starten.

3.6 Kernel

Hinweis: *Nachfolgendes Vorgehen wurde getestet mit folgenden Versionsständen:*
kernel.org Version 2.6.38.6
kernel.org Version 2.6.39.3

Beschaffung des Kernels

```
cd mini2440
mkdir kernel
cd kernel/
git clone git://git.kernel.org/pub/scm/linux/kernel/git/stable/linux-
2.6.39.y.git
```

Da FriendlyARM den Supplier des LCDs geändert hat, sind Änderungen am Kernel notwendig, um das W35 LCD (LQ035Q1DG06) zu unterstützen. Dazu muss die Datei *arch/arm/mach-s3c2440/mach-mini2440.c* um ein weiteres Displaytiming erweitert werden:

Erweiterungen in der Datei mach-mini2440.c, in der Struktur
„static struct s3c2410fb_display mini2440_lcd_cfg[] __initdata“

```
[4] = { /* LCD-W35i 3.5" display (LQ035Q1DG06)*/
                _LCD_DECLARE(
                 7, /* clock */
                 320, 68, 66, 4, /* xres, margin_right, margin_left, hsyn */
                 240, 4, 4, 9, /* yres, margin_top, margin_bottom, vsync */
                 60), /* refresh rate */
                 .lcdcon5 = (S3C2410_LCDCON5_FRM565 |
                 S3C2410_LCDCON5_INVVDEN |
                 S3C2410_LCDCON5_INVVFRAME |
                 S3C2410_LCDCON5_INVVLINE |
                 S3C2410_LCDCON5_INVVCLK |
                 S3C2410_LCDCON5_HWSWP),
                 },
```

Sind die Änderungen abgeschlossen, kann mit der Konfiguration des Kernels begonnen werden.

Konfigurieren des Kernels

```
make menuconfig
```

Name	Wert
System Type	Samsung S3C2410...
ARM system type	aktiv
ADC common driver support	aktiv
PWM device support	
S3C2440 and S3C2442 Machines	aktiv
MINI2440 development board	
Kernel Features	
Use the ARM EABI to compile the kernel	aktiv
High Memory Support	inaktiv
Enable KSM for page merging	inaktiv
CPU Power Management	
CPU Frequency scaling	inaktiv
Device Drivers	
Memory Technology Device (MTD) support	aktiv
NAND Device Support	aktiv
Verify NAND page writes	aktiv
NAND Flash support for Samsung S3C SoCs	aktiv
Support for generic platform NAND driver	aktiv
Parallel port support	inaktiv
Block devices	inaktiv
Misc devices	inaktiv
Serial ATA and Parallel ATA drivers	inaktiv
Multiple devices driver support	inaktiv
ISDN support	inaktiv
Telephony support	inaktiv
Input device support	
Keyboards	inaktiv
Mice	inaktiv
Joysticks/Gamepads	inaktiv
Tablets	inaktiv
Touchscreens	
Samsung S3C2410/generic touchscreen input driver	aktiv
<alle anderen Touchscreens>	inaktiv
Miscellaneous devices	inaktiv
Character devices	
TMP Hardware Support	inaktiv
SPI Support	inaktiv
GPIO Support	
/sys/class/gpio... (sysfs interface)	aktiv
<alle anderen GPIO-Optionen>	inaktiv
Graphics Support	
Support for frame buffer devices	aktiv
S3C2410 LCD framebuffer support	aktiv
Bootup logo	aktiv
Staging drivers	inaktiv

Ist die Konfiguration abgeschlossen, kann mit *make* der Kernel erstellt werden.

3.7 Embedded Linux - Bootsequenz

Gewünscht ist es, dass nach dem Systemstart (PowerOn), das Linux hochfährt und die Qt Applikation startet. Dazu ist ein genauerer Blick in die Abläufe beim Bootup notwendig.

- Power On
  Register im Target nehmen Ausgangszustände an und versuchen an definierten Stellen ausführbaren Code zu laden und diesen auch sofort auszuführen. Über den Hardwareschalter NAND/NOR (U-Boot/Supervivi) kann darauf Einfluss genommen werden.

- First Stage Bootloader
  Das U-Boot ist geladen und das Target ist nun in der Lage, komplexere Vorgänge für das weitere Laden des Betriebssystems durchzuführen (z.B. ext2 Dateisystemzugriffe über eine SD-Speicherkarte). Ist der Kernel ins RAM geladen wird er ausgeführt.

- Kernel
  Der Kernel lädt während seines Startvorgangs diverse Hardwaretreiber und mountet auch das RootFilesystem. Letzteres ist essentiell notwendig, da der Kernel nach dem Hochfahren versuchen wird, den Init-Prozess (der Erste im User-Space existierende Prozess) zu starten.

- Init-Prozess
  Der Init-Prozess ist der erste Prozess des Betriebssystems, von dem alle weiteren Prozesse abgeleitet sind. Init kann in der Datei /etc/inittab (RootFilesystem) konfiguriert werden. In dieser Datei wird auch das Aufrufen der /etc/init.d/rcS definiert, die alle Scripte in dem Verzeichnis /etc/init.d/ der Reihe nach ausführt. Möchte man bei Systemstart bestimmte Programme ausführen, so sind diese Scripte zu erweitern.

- /etc/init.d/S60app
  Die Datei /etc/init.d/S60app wurde angelegt und mit folgendem Inhalt gefüllt. Damit startet nach dem Hochfahren des Kernels automatisch die Qt-Anwendung.

/etc/init.d/S60app

```
#!/bin/sh
#
# Start QTAPP
#
echo "Calling QT application in new process"
echo 1 > /sys/devices/platform/s3c24xx_led.5/leds/backlight/brightness
/home/default/application -qws -display transformed:rot90 &
```

3.8 Nokia Qt

Die grundsätzliche Grafikoberfläche wurde mit dem QT-Designer (Abb. 3.5) erstellt. Der Qt-Designer generiert eine *.ui Datei, die den Aufbau der GUI für das Target in einer XML-ähnlichen Struktur abspeichert.

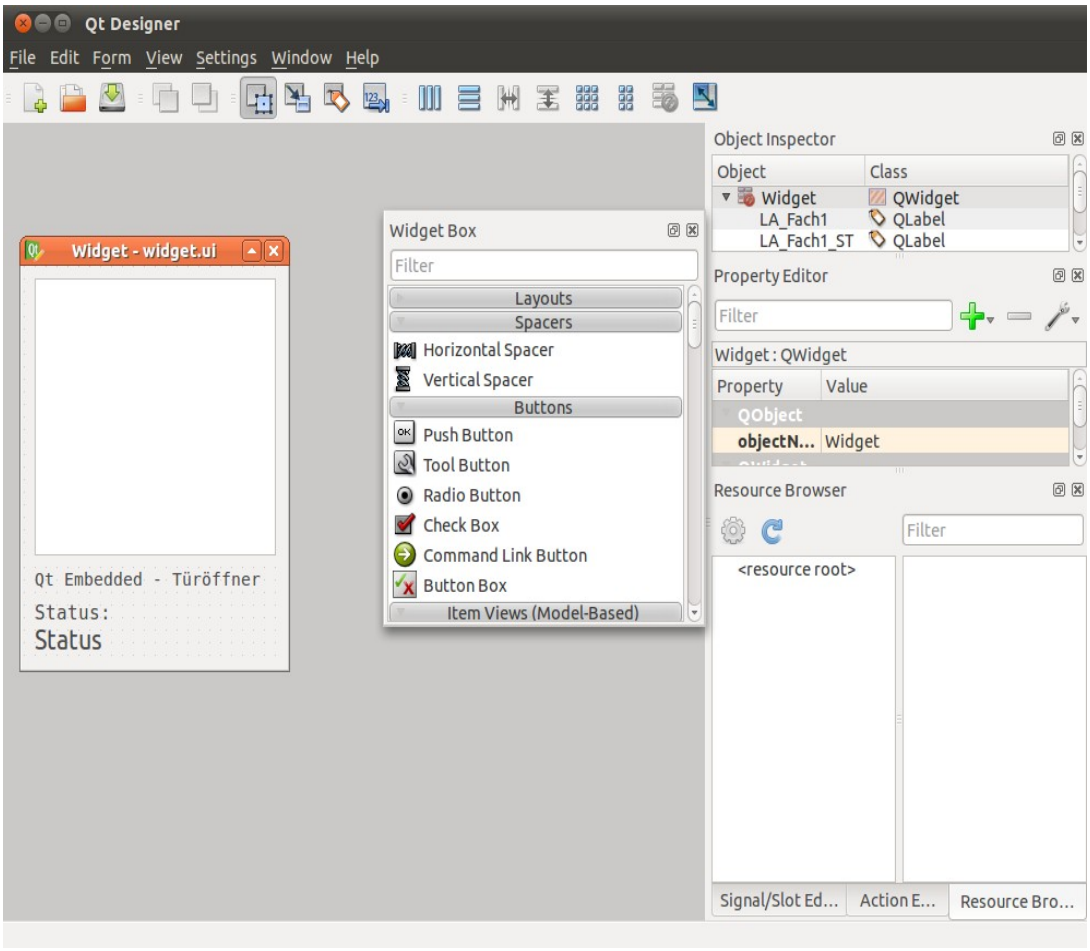

Abb. 3.5: Qt-Designer

Wird das von Buildroot erstellte Cross-Compilation-Tool *qmake* (output/host/usr/bin/qmake) im Verzeichnis der *.ui-Datei auf eine noch nicht existierende *.pro Datei angewandt, erstellt qmake die *.pro Datei. Führt man noch einmal *qmake* (dieses Mal ohne Parameter) aus, wird das aktuelle Verzeichnis nach *.cpp / *.hpp Dateien durchsucht und eine Makefile mit korrekten Abhängigkeiten erstellt. Mit *make* wird die Applikation dann tatsächlich für das Target gebaut.

Erstellen des Projekts aus der *.ui, Ausführen von qmake und bauen der Anwendung

```
buildroot/output/host/usr/bin/qmake meinprojekt.pro
buildroot/output/host/usr/bin/qmake
make
```

3.9 Anwendungssoftware unter Qt

3.9.1 Signal-Slot-Konzept

Beim Entwickeln der grafischen Qt Anwendung stellt man sehr schnell fest, dass man zum Lösen der Aufgabe zwei parallele Prozesse, mindestens aber zwei parallele Threads, benötigt. Ein Thread wird dafür verwendet die Grafik zu bedienen, während der andere Thread die tatsächliche Aufgabe (Kommunikation mit TopSec, Zustandsverwaltung, etc.) übernimmt. Das Konzept der Synchronisierung zweier Threads ist in Qt mittels „Signals and Slots“ gelöst und ähnelt den in UNIX Systemen bekannten Signals (SIGSEV, SIGTERM, etc.). Threads stellen Slots zur Verfügung, welche Signale anderer Teilnehmer verarbeiten können. Die Signale können von einem beliebigen Thread (für ihn selbst nicht blockierend) ausgelöst werden. Bevor dies stattfinden kann, müssen Signale und Slots miteinander verbunden werden, welches mit der Bibliotheksfunktion connect(), zur Laufzeit ausgeführt wird. Die implementierte Signal- Slot- und Threadstruktur der Applikation findet sich in Abb. 3.6. Die Dokumentierung des Statemachinethreads ist, neben Quellcodekommentaren, in einem Zustandsdiagramm in Kapitel 3.9.2 zu sehen.

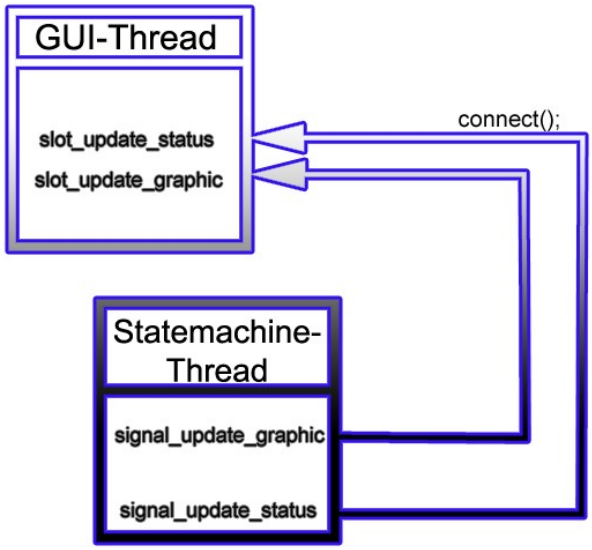

Abb. 3.6: Signal-Slot-Thread-Konzept des Türöffners

3.9.2 Statemachine

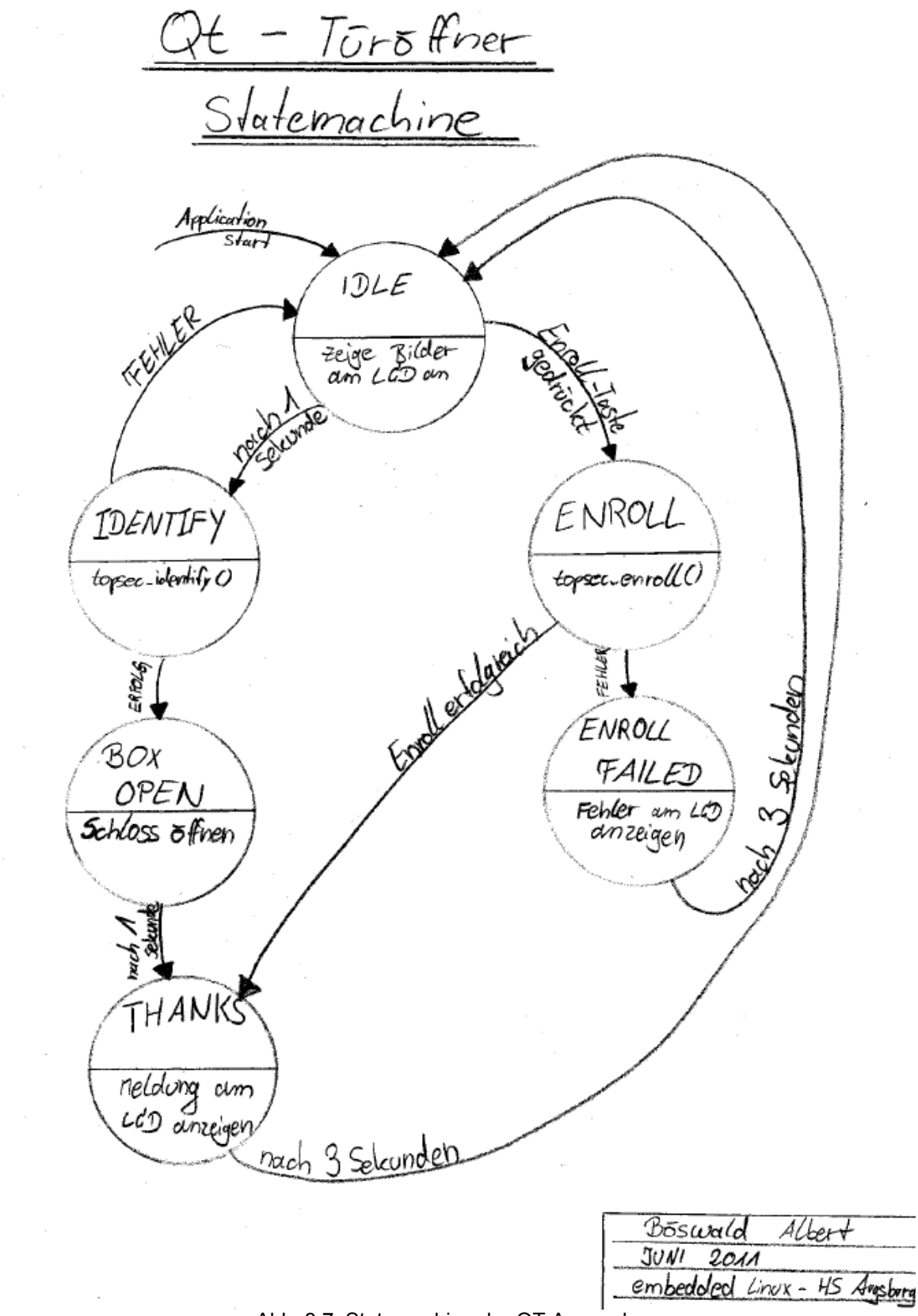

Abb. 3.7: Statemachine der QT Anwendung

4 Quellenangaben

[01] Serial Programming Guide for POSIX Operating Sysms by Michael R. Sweet
http://www.easysw.com/~mike/serial/serial.html
[02] The TopSec ID Module Interface Description Paper V2.701
[03] http://www.mbfingermetrica.de/Module-Sensoren.html
[04] http://www.mikrocontroller.net/articles/Mini2440
[05] http://cchia-cwp.blogspot.com/2010/05/running-buildroot-system.html
[06] http://wiki.linuxmce.org/index.php/Mini2440
[07] http://pauljones.id.au/blog/tag/mini2440/
[08] http://doc.trolltech.com/4.7/signalsandslots.html
[09] http://de.wikipedia.org/wiki/Signal-Slot-Konzept
[10] http://www.ibm.com/developerworks/linux/library/l-linuxboot/
[11] http://www.yolinux.com/TUTORIALS/LinuxTutorialInitProcess.html
[12] http://www.gossamer-threads.com/lists/linux/kernel/1393616

[13] http://www.friendlyarm.net
[14] http://www.denx.de/
[15] http://buildroot.uclibc.org
[16] http://programmers-projects.de/node/16
[17] http://doc.qt.nokia.com/

5 Lizenz dieser Arbeit