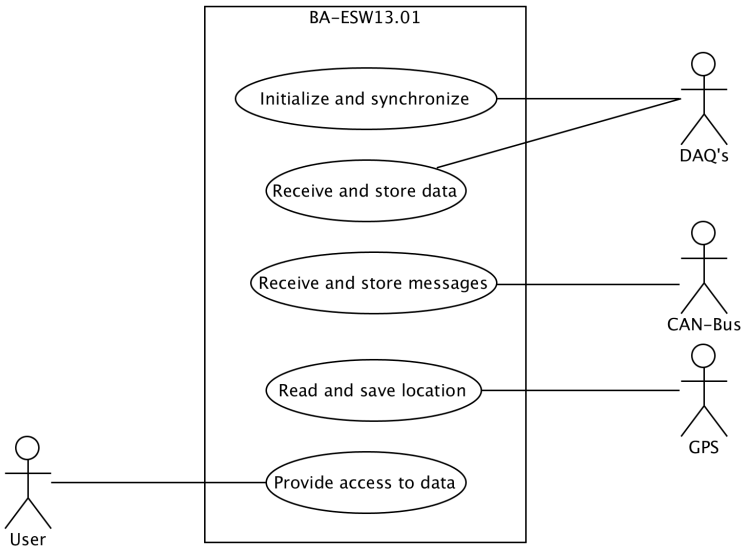
MOBILE DATALOGGER

ANDREAS ZIEGLER



Mobile datalogger for recording decentrally captured dynamic motor vehicle data

Prof. Reto Bonderer

BA-ESW13.01

Electrical Engineering, Embedded Software Engineering

June 2013 – version 1.0

HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL

FHO Fachhochschule Ostschweiz

SPORTEC
MOTORSPORT

Andreas Ziegler: *Mobile Datalogger*, Mobile datalogger for recording decentrally captured dynamic motor vehicle data, © June 2013

SUPERVISORS:
Prof. Reto Bonderer
Gian Danuser B.Sc. FHO

LOCATION:
Rapperswil

TIME FRAME:
June 2013

ABSTRACT

PROBLEM The company Sportec AG from Höri ZH is a professional tuner for cars and is also active in motor sports. To analyze and optimize the driving dynamics of racing cars on racetracks, Sportec AG needs a measurement system. With this system it should be possible to measure high dynamic processes, like length changes of the dampers and acceleration. At the same time also slow processes like the oil temperature should be captured over longer time. The position of the car should be logged by the system with a Global Positioning System (GPS) receiver and the messages which are exchanged over the car internal Controller Area Network (CAN) bus should also be recorded by the system. Because of the rough environment and the high number of disturbing factors in a racing car, the Analog-to-Digital Converters (ADC's) should be placed as close as possible to their sensors. To reach this goal, the system will be realized as a distributed system with Data Acquisition (DAQ) modules, which already exist, and with a central Data Logger (DL).

PROCEEDING In the analysis the specification for the mobile data logger has to be done as the first step, according to the job description. In the next step, the best fitting hardware has to be evaluated and ordered. After the hardware is chosen, the development of the software, starting with the analysis, followed by the design and the implementation, must be done. At the end the whole mobile data logger (hardware and software) has to be tested.

RESULT With the PandaBoard, an embedded platform was chosen on which an normal Linux operating system can run. The used Linux distribution is "Ubuntu Core", a "Ubuntu" version with a very small memory footprint that still allows to install a lot of standard Linux packets.
The mobile data logger software allows five DAQ modules to connect over Wireless Local Area Network (WLAN). Over these WLAN connections, the DAQ modules get configured, their timer reseted and the measurement started and stopped. With IEEE802.15.4 broadcast messages every 250 ms over a XBee module, the five DAQ modules are kept in sync. The sensor data which the DAQ modules are sending over WLAN are received by the mobile data logger software and saved to a file on a SD card. The received positions from the GPS receiver are also saved to a file on a SD card.

ACKNOWLEDGMENTS

First of all, I would like to thank my wife 정춘화 for the patient she had with me during this thesis and also for the support she gave me.

I also want to thank my parents, Fredi and Ursula Ziegler, for helping me to correct my documentation and all the other help they gave me during this thesis.

As next, I want to thank my supervisor, Gian Danuser, for his feedback and help.

I want to thank 김창일 for the hints he gave me.

As last I want to thank Caspar Naef, the labor assistant, for all the orders he did for me and also for helping me with the laboratory infrastructure.

CD CONTENTS

The enclosed CD contains the following directories:

doc	This documentation as pdf file and the LaTeX source code
src	The source code of the mobile data logger software
src/doc	The doxygen documentation of the mobile data logger software
bin	The mobile data logger software
systemtest-src	The source code of the test programs
resources	Available literature as pdf
report	Meeting reports as pdf files and the LaTeX source code

CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACRONYMS

DL Data Logger

DAQ Data Acquisition

ADC's Analog-to-Digital Converters

HSR Hochschule für Technik Rapperswil

IDE Integrated Development Environment

OS Operating System

OOP Object-Oriented Programming

CLI Command Line Interface

CAN Controller Area Network

LIN Local Interconnect Network

UART Universal Asynchronous Receiver Transmitter

MAC Media Access Control

IP Internet Protocol

DHCP Dynamic Host Configuration Protocol

AP Access Point

WLAN Wireless Local Area Network

FSM Finite State Machine

GPS Global Positioning System

GPIO General Purpose Input/Output

1 PREFACE

This document was written for the bachelor thesis BA-ESW13.01 in the department Electrical Engineering at the Hochschule für Technik Rapperswil (HSR) in the subject area Embedded Software Engineering. The client for which this system is developed is the company Sportec AG from Höri ZH.

The company Sportec AG is a professional tuner for cars and is also active in motor sports. To analyze and optimize the driving dynamics of racing cars on racetracks, Sportec AG needs a measurement system. With this system it should be possible to measure high dynamic processes, like length changes of the dampers and acceleration. At the same time also slow processes like the oil temperature should be captured over longer time. The position of the car should be logged by the system with a GPS receiver and the messages which are exchanged over the car internal CAN bus should also be recorded by the system. Because of the rough environment and the high number of disturbing factors in a racing car, the ADC's should be placed as close as possible to their sensors. To reach this goal, the system will be realized as a distributed system with five DAQ modules and a central DL. In figure 1 this distributed measurement system is shown. Four of the DAQ will be positioned in the suspension and one in the engine compartment, as shown in figure 2. With this system, Sportec AG should get real data of the driving dynamic which allows to optimize the drivability.

The five DAQ modules and the DL as distributed system.

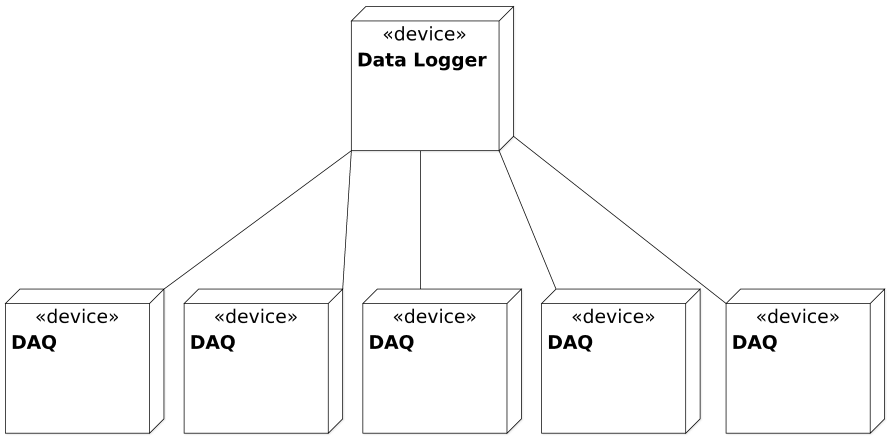


Figure 1: Distributed measurement system

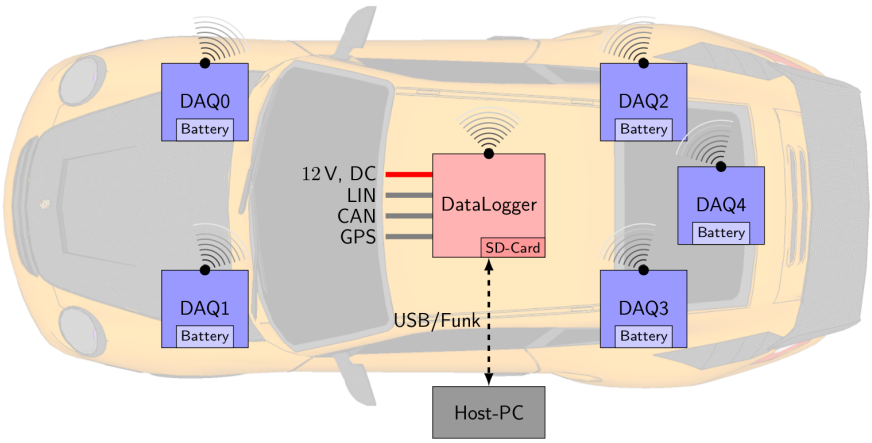


Figure 2: Position of the DAQ modules and the DL

1.1 DATA LOGGER

The DL initializes and configures the DAQ modules so that they are optimal adjusted for the connected sensors. The DL receives and stores the data, which are sent from the DAQ modules.

The DL also has to synchronize the DAQ modules to be able to analyze the captured data.

From the CAN bus, the DL receives the exchanged messages and stores them.

The position, received from a GPS receiver has to be read and also stored.

All the stored data has to be accessible for the user.

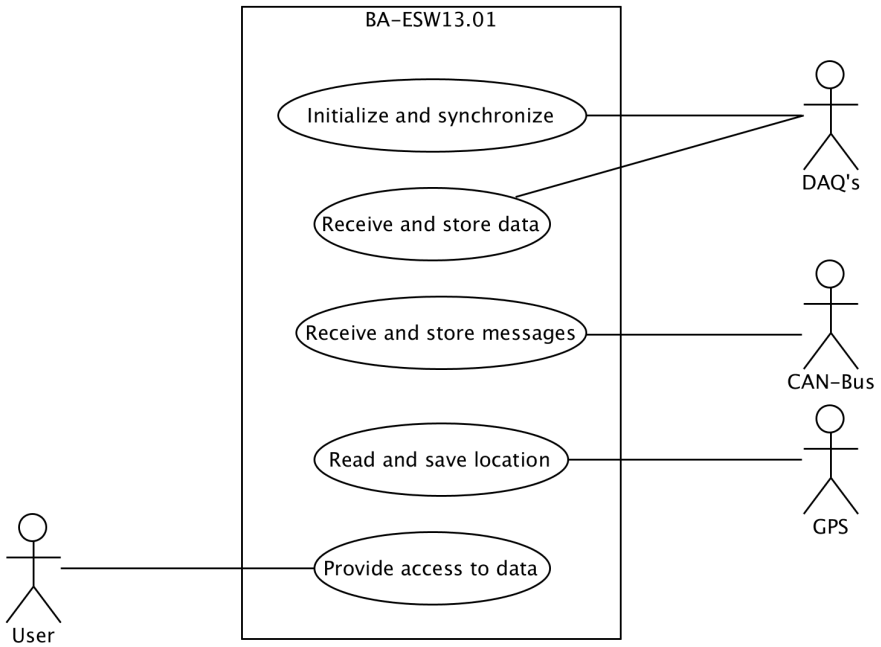


Figure 3: Use case diagram of the DL

HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL
ELEKTROTECHNIK

Embedded Software Engineering

Bachelorarbeit **BA-ESW13.01** **FS 2013**

Aufgabenstellung

für

Andreas Ziegler

Mobiler Datenlogger zur Aufzeichnung dezentral erfasster dynamischer Motorfahrzeugdaten

1. Einführung

Die Firma Sportec AG aus Höri ZH ist ein professioneller Tuner für Pkw und ist auch im Rennsport tätig. Um die Fahrdynamik der Rennwagen auf der Rennstrecke zu analysieren und optimieren, benötigen sie ein umfassendes Messsystem. Mit diesem Messsystem sollen hoch dynamische Vorgänge,

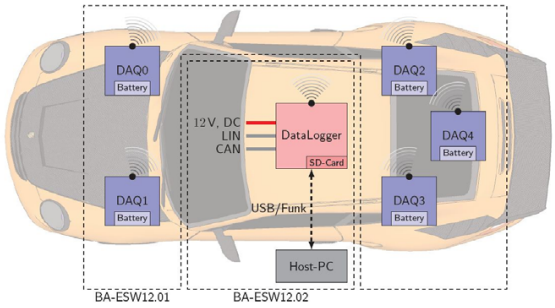


Abbildung 1: Verteilte Messdatenerfassung

wie zum Beispiel die Längenänderung am Stossdämpfer und Beschleunigungen, aber gleichzeitig auch langsame Vorgänge (zum Beispiel die Öltemperatur) über längere Zeit erfasst und abgespeichert werden können. Die Daten, die über den fahrzeugeigenen CAN-Bus ausgetauscht werden, müssen ebenfalls in das Messsystem integriert und aufgezeichnet werden.
In Anbetracht der rauen Umgebungsbedingungen und der vielen Störfaktoren in einem Rennwagen ist es sinnvoll, die A/D-Wandlung möglichst nahe bei den Sensoren zu plazieren. Aus diesem Grund

und wegen dem Verkabelungsaufwand wird das Messsystem als ein verteiltes System aus Datenerfassungsmodulen (DAQ) und einem zentralen Datenlogger ausgelegt. Die Abbildung 1 zeigt das Messsystem mit fünf DAQs. Vier davon sind bei den einzelnen Radaufhängungen und eines im Motorenraum plaziert.
Mit diesem Werkzeug soll Sportec reale Daten über die Fahrdynamik der Rennwagen erlangen. Dadurch können Sie das Fahrverhalten fundiert optimieren.

2. Aufgabenstellung

Das Ziel der Arbeit ist die Entwicklung des zentralen Datenloggers. Der Datenlogger muss alle Daten, welche von den DAQ-Modulen übertragen werden, aufzeichnen können. Zusätzlich sollen Meldungen auf den autointernen Bussen (CAN / LIN) sowie Positionsdaten (GPS) geloggt werden können. Die Konfiguration und die Synchronisation der DAQ-Module übernimmt ebenfalls der Datenlogger.

In einer ersten Phase muss eine geeignete Hardware evaluiert und eingekauft werden. Die Hardware soll den folgenden groben Anforderungen genügen (die Detailspezifikation ist Teil des Pflichtenhefts):

- Spannungsversorgung: 8 V - 24 V, DC Single Supply
- Synchronisation der angeschlossenen DAQs mittels ZigBee
- WLAN mit Accesspoint-Funktionalität für die Kommunikation mit den DAQs
- Handling von mindestens fünf offenen TCP Sockets
- Datenraten von bis zu 1.2 Mbps pro TCP Socket
- GPS für zusätzliche Positionsdaten
- Integration des fahrzeugeigenen CAN- und LIN-Bus
- USB oder Ethernet-Anschluss für Onlinezugriff und Konfiguration der Messung
- Anschlussmöglichkeit für Präsentationsgeräte (PC, Pad, etc.)
- Speicherung der Messdaten auf einer SD-Karte
- Konfiguration der DAQs (Mess-Setup) ebenfalls auf der SD-Karte
- Unterstützung von Embedded-Betriebssystemen (Embedded Linux, etc.)

Bei der Programmierung der Software muss zwischen Analyse und Design unterschieden werden. Im folgenden sind die einzelnen Aufgaben aufgelistet:

- Evaluation und Beschaffung einer geeigneten Hardware, inkl. Inbetriebnahme und Test
- Festlegung der Software-Schnittstellen (API)
- Entwurf der Software-Architektur
- Implementation und Verifikation

3. Ablauf

Zu Beginn der Bachelorarbeit sind ein Projektplan sowie ein detailliertes Pflichtenheft zu erstellen, welches von allen beteiligten Parteien zu genehmigen ist. Anschliessend sind mögliche Lösungsansätze zusammenzutragen, im Vergleich objektiv einander gegenüberzustellen und zu bewerten. Die am besten geeignete Variante soll schliesslich ausgearbeitet, implementiert und detailliert getestet werden.
Bei mehrköpfigen Teams sollen die verschiedenen Tätigkeiten und Teilaufgaben innerhalb des Teams sinnvoll verteilt werden. Die konkrete Aufteilung ist im Projektplan klar und deutlich festzuhalten. Weitere Einzelheiten werden an den regelmässigen Besprechungen festgelegt und sind zu protokollieren.

4. Bericht

Über die Arbeit ist ein Bericht zu verfassen, dessen Textteil 60 Seiten nicht überschreiten soll. Im Bericht müssen alle wesentlichen gemachten Überlegungen, Abklärungen, Berechnungen und Untersuchungen detailliert (in Text und Bild) dokumentiert werden.

Der Bericht muss gut leserlich geschrieben und übersichtlich gegliedert sein. Er soll mindestens die folgenden Kapitel umfassen: Inhaltsverzeichnis, allgemeine (für nicht Fachleute) verständliche Einleitung (Abstract), Original der Aufgabenstellung, kommentierter Zeitplan (Soll) und Arbeitsfortschritt (Ist) inklusive Aufteilung der Arbeiten und Pflichtenheft. Der Aufbau des übrigen Teils des Berichtes mit einer Analyse der Aufgabenstellung und Bewertung der Lösungsmöglichkeiten, einer Konzept- und Realisierungsbeschreibung, sowie der zugrundeliegenden theoretischen Betrachtung ist der Aufgabe entsprechend zu gestalten. Zum Bericht gehören immer auch eine Systembeschreibung mit einer klaren Festlegung der Systemgrenze, Angaben darüber, wie das ganze System zu erstellen ist, sowie eine kommentierte Auflistung der zugehörigen Daten.

Der Bericht ist in 3 Papier-Exemplaren, sowie in elektronischer Form (in Source und als PDF-Dokument) abzugeben. Der Bericht und alle anfallenden Daten sind auf 3 CDs/DVDs zu archivieren und ebenfalls abzugeben.

5. Termine

KW 08	Montag, 18.02.2013	Ausgabe der Aufgabenstellung, Beginn der Arbeit
KW 24	Freitag, 14.06.2013 13:00	Abgabe des Berichts, Ende der Arbeit

6. Besonderes

Während der ganzen Arbeit haben alle Studenten ein stets aktuelles, persönliches Laborjournal in Form eines Heftes zu führen. Darin sind Arbeitszeiten, Tätigkeiten, Erkenntnisse, Beschlüsse, usw. in chronologischer Reihenfolge festzuhalten. Das Laborjournal kann vom Betreuer jederzeit eingesehen und auch zur Gesamtbewertung beigezogen werden. Das Laborjournal ist am Ende der Arbeit zusammen mit dem Bericht abzugeben.

Wenn Programmcode erstellt wird, muss ein Versionsverwaltungstool eingesetzt werden.

Anschaffungen und Leistungen Dritter müssen rechtzeitig angemeldet und im voraus bewilligt werden. Über die wöchentlichen Besprechungen ist ein Kurzprotokoll zu verfassen.

Der Erfolg in der Bachelorarbeit ist stark von der Zusammenarbeit aller Beteiligten abhängig. Eine gute Koordination und Kommunikation ist wichtig und muss von allen Seiten aktiv gefördert werden.

7. Bewertung

Bewertet werden: Arbeits-Methodik inkl. Projektplanung und -durchführung, technischer Inhalt, Resultat, Schlussbericht und Präsentationen. Ebenso werden die individuellen Laborjournale zur Bewertung beigezogen.

8. Organisatorisches

Betreuung der Arbeit:	Prof. Reto Bonderer, Gian Danuser
Industriepartner:	Sportec AG, CH-8181 Höri
Betreuung des Labors:	Caspar Naef
Arbeitsplatz:	Labor 1.206a
Regelmässige Besprechungen:	jeweils am Dienstag um 15:10 Uhr im 1.206a

Rapperswil, Februar 2013 ***R. Bonderer***

Part I

ANALYSIS

3 ANALYSIS

3.1 SPECIFICATION

The results of the problem analysis and additional clarification, as also the deferrals of this thesis are noted in the specification. The specification is in appendix B.

3.2 EVALUATION OF THE EMBEDDED BOARD

In this section, various embedded boards are compared to select the embedded board which fits best to the job definition. In this thesis, the embedded board has to handle different interfaces and protocols at the same time. To be able to handle this tasks, an embedded board with an Operating System (OS) is preferred. The key point of this thesis is to handle five wireless connections at the same time, therefor a board with on board WLAN module is preferred. The different embedded boards are listed in table 1.

3.2.1 *Summary*

The TS-7553 can be excluded from the evaluation because a 250 MHz ARM9 CPU is without much doubt overloaded with handling a OS, five wireless connections, a Dynamic Host Configuration Protocol (DHCP) server and the other required functions. Snowball has an on board GPS receiver which makes it unnecessary to attach an external GPS receiver. Without an extension board, the Snowball hasn't any RS-232 or Universal Asynchronous Receiver Transmitter (UART) connectors and with all the required extension boards it is quite expensive. The Hackberry is with his 4GB NAND flash and a price of only 65 $ quite attractive. The disadvantage is, that the Hackberry doesn't have any bus interface like SPI, I2C or SDIO and there are also none GPIO or buttons.

The PandaBoard is a widely known embedded board with a lot of documentation available. Over SPI or I2C devices like a CAN interface can be attached and with the built-in button, the user could control the board without any special input device.

3.3 POWER SUPPLY

The PandaBoard requires a 5V DC power supply but the specifications define a power supply with a range from 8V DC to 24V DC. To

Module	TS-7553	Snowball
Producer	Technologic Systems Inc.	Calao
Processor	250 Mhz ARM9 CPU	1 GHz ARM A9 CPU
ZigBee	via connector	via RS-232/UART
WLAN (g)	via int. USB	on board
CAN / LIN	on board / via RS-232[1]	via extension Board / RS-232[1]
GPS	via RS-232/UART	on board
USB / Ethernet	yes / yes	yes / yes
(µ)SD	yes	yes
Price	Board: 135 $	Board: 322.57 €
		CAN: 160.7 €
		ZigBee: 115.89 €
Module	PandaBoard	Hackberry
Producer	PandaBoard	Miniand
Processor	1 GHz ARM A9	1.2 GHz ARM A8
ZigBee	via RS-232/UART	via RS-232/UART
WLAN (g)	on board	on board
CAN / LIN	via USB/SPI / via RS-232[1]	via USB / via RS-232[1]
GPS	via RS-232/UART	via USB
USB / Ethernet	yes/yes	yes/yes
(µ)SD	yes	yes
Price	Board: 183 CHF	Board: 65 $
	ZigBee: 50 CHF	ZigBee: 50 CHF
	CAN: 195 €	CAN: 195 €
	GPS: 80 CHF	GPS: 80 CHF

[1] LIN over RS as shown in [20]

Table 1: Comparison of different embedded boards

satisfy this specification, a DC/DC converter is needed. At the time of the order, only one fitting device was available, whose key data are shown in table 2. wer

3.4 OPERATING SYSTEM

The PandaBoard is compatible with different Linux derivatives. In this section various Linux derivatives are compared to select an OS, which fits best to the job definition and to the embedded board. The compared OS are listed in table 3.

Module	NGA10S15050SC
Producer	muRata Ps
Input voltag range	7.0 V DC to 28 V DC
Output voltage	5.0 V DC
Output current	2.0 A
Price	48.10 CHF

Table 2: Key data of the chosen power supply

Name	Ubuntu Server	Ubuntu Desktop
Features	Server Linux System	Desktop Linux System
CPU requirement	300 MHz	700 MHz
Memory requirement	256 MB	512 MB
Space requirement	1 GB	5 GB
Name	Ubuntu Core	Linux Minimal
Features	Minimal Ubuntu System	Minimalistic Linux System
CPU requirement	minimal[1]	minimal[1]
Memory requirement	minimal[1]	minimal[1]
Space requirement	300 MB	4 MB

[1] There are no hardware requirements published because it depends on the utilization of the system.

Table 3: Comparison of different Linux derivatives

3.4.1 *Summary*

Ubuntu is well supported by the PandaBoard and with "apt" Ubuntu has a good packet manager which allows to install a lot of available software. For this thesis, a desktop environment isn't necessary and so Ubuntu Desktop can be excluded from the evaluation. Ubuntu Server is a Command Line Interface (CLI) system which still has a lot of software packages pre-installed, which aren't needed. The Linux derivative with the smallest footprint which is support by the PandaBoard is the "Linux Minimal" system which is based on the "Angstrom Distribution"[1] . To be able to use a lot of packages known from Ubuntu but still have a small footprint, Ubuntu Core is used in this thesis.

1 http://www.angstrom-distribution.org/

3.5 IEEE802.15.4 RADIO MODULE

The PandaBoard has a UART interface on which a IEEE802.15.4 module can be connected. With a connected IEEE802.15.4 module, it allows the PandaBoard to synchronize the DAQ modules with broadcast messages. The different IEEE802.15.4 modules are listed in table 4.

Module	ATZB-24-B0	XB24-ACI-001
Producer	Atmel	Digi
Transmit power	2 mW	1 mW
Serial data rate	250 Kbps	250 Kbps
Price	28 CHF	19 CHF
Module	ZICM357SP2-1-B	ETRX2
Producer	CEL	Telegesis
Transmit power	100 mW	2 mW
Serial data rate	250 Kbps	250 Kbps
Price	33 CHF	31 CHF

Table 4: Comparison of different XBee-modules

3.5.1 *Summary*

With exception of the XB24-ACI-001 from Digi, all the modules are designed as a surface mounted device. Because the XB24-ACI-001 is easy connectable via wires, it's well known for its simple UART communication protocol and there are libraries, which make the development with the modules of Digi easier, the XB24-ACI-001 was chosen as IEEE802.15.4 module.

3.6 CAN MODULE

To log messages on the car internal CAN bus, a CAN bus interface is required. There are two possible well known CAN interface solution, which are compatible with the PandaBoard, shown in table 5.

Module	PCAN-USB	CAN SPI click 5V
Producer	PEAK	MikroElektronika
Interface	USB	SPI
Price	195 €	19 €

Table 5: Comparison of the two CAN interfaces

3.6.1 *Summary*

The desired CAN interface is the "CAN SPI click 5V" because it is much more cost-effective compared to the "PCAN-USB".

3.7 GPS MODULE

Because the UART interface on the PandaBoard is already used by the XBee module, the GPS module has to be connected over USB. The compared GPS modules are shown in table 6.

Module	NL-402U	NL-302U	GM720	GPS-Receiver
Producer	Navilock	Navilock	Navibe	Hama
Interface	USB	USB	USB	USB
Price	68 CHF	64 CHF	50$	40 €

Table 6: Comparison of different GPS interfaces

3.7.1 *Summary*

The products of the company Navilock have a good availability in Switzerland. Next to this the NL-402U from Navilock use a high performance GPS chip from the company u-blox, which is compatible with Linux and supports the NMEA protocols, which was the reason, that the NL-402U was chosen as GPS module.

3.8 PROGRAMMING LANGUAGE

To implement the DL software, multiple programming languages could be used. A selection of possible programming languages is shown in table 7

3.8.1 *Summary*

The design of the DL software is object oriented, which excludes "C" from the choice. The languages "C#" and "Java" doesn't support system interaction on Linux efficient. The best language for system programming under Linux with object oriented programing and multi threading support is "C++11" which will be used for the implementation of the DL software.

Name	C
Advantages	Efficient, good integration in Linux
Disadvantages	No support for OOP
Name	C++11
Advantages	Efficient, good integration in Linux, OOP support
Disadvantages	More complicate
Name	C#
Advantages	A lot of available libraries, OOP support
Disadvantages	Not so efficient, not so good integration in Linux
Name	Java
Advantages	A lot of available libraries, OOP support
Disadvantages	Not so efficient, not so good integration in Linux

Table 7: Selection of possible programming languages

Part II

SET UP

4 SETUP

4.1 SYSTEM

The PandaBoard has a WLAN interface on board. The rest of the components has to be connected as it is shown in figure 4.

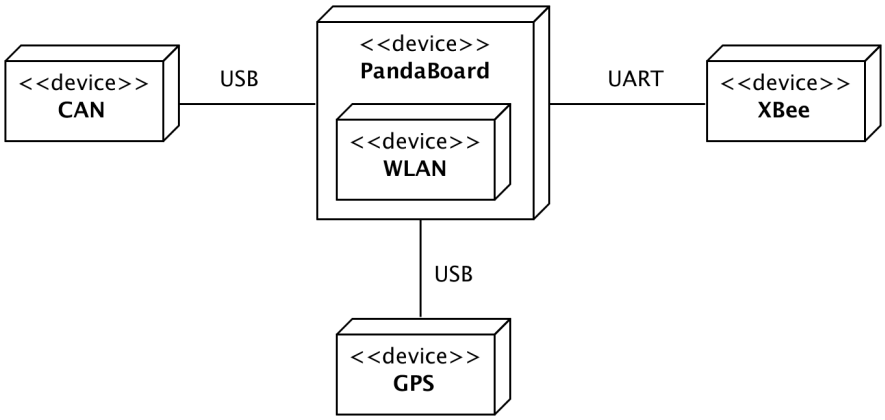


Figure 4: System with it's components

The XBee module is connected over a voltage level shifter to the UART interface #4 of the PandaBoard. The voltage level shifter[1], which is developed by Dirk Grappendorf, is shown in appendix C. The level shifter has to be connected to the PandaBoard as described in table 8. In this thesis the voltage level shifter is realized on a Vero Board, shown in figure 5.

RcvData and TxData is from the sight of the PandaBoard.

PandaBoard J3	—>	Level shifter JP1
Pin 2	(5 V)	Pin 1
Pin 27	(GND)	Pin 2
Pin 8	(RcvData)	Pin 3
Pin 1	(1.8 V)	Pin 4
Pin 6	(TxData)	Pin 5

Table 8: PandaBoard to voltage level shifter connections

1 coyoho-xbeepandaadapter: https://sites.google.com/site/grappendorfnet/projects/coyoho/coyoho-server-hardware

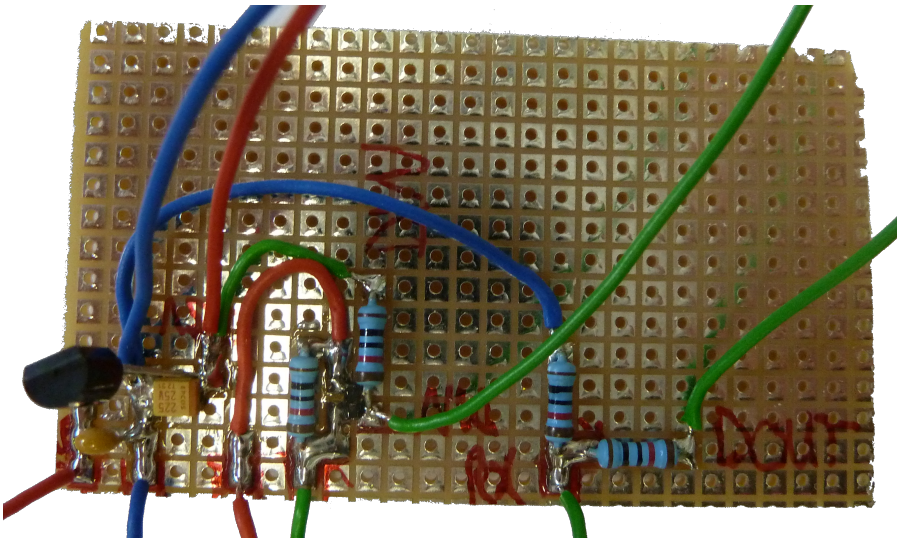


Figure 5: Voltage level shifter realized on a Vero Board

4.2 SETTING UP THE CROSS COMPILER

To reduce the compile time for the Linux kernel and the libraries, a cross compiler can be used. For this thesis, the "arm hf crosscompiler for Linux"[2] from Linaro was used. This package contains the Linaro GCC, the Linaro GDB, standard includes and standard libraries for the "arm hf" architecture. To install this cross compiler, it first has to be downloaded. This can be done with the command:

```
1 wget https://launchpad.net/linaro-toolchain-binaries/trunk/2013.03/+download/gcc-
      linaro-arm-linux-gnueabihf-4.7-2013.03-20130313_linux.tar.bz2
```

With the command:

```
1 tar -xvf gcc-linaro-arm-linux-gnueabihf-4.7-2013.03-20130313_linux.tar.bz2
```

the package will be extracted and with the command:

```
1 mv gcc-linaro-arm-linux-gnueabihf-4.7-2013.03-20130313_linux /opt
```

the directory is moved to a convenient location. Now the path to the cross compiler can be set in a makefile or in a Integrated Development Environment (IDE).

How the libraries, which are needed for this thesis, are built and installed in the cross compiler environment is described in section 4.3.7.

4.3 SETTING UP UBUNTU CORE ON THE PANDABOARD

To set up Ubuntu Core the following steps are required:

2 https://launchpad.net/linaro-toolchain-binaries/trunk/2013.03/+download/gcc-linaro-arm-linux-gnueabihf-4.7-2013.03-20130313_linux.tar.bz2

- Preparation of the SD card
- Basic Ubuntu Core installation
- Modification of Ubuntu Core
- Update of Ubuntu Core and installation of the required packages
- Customization of the kernel
- Configuration of the required packages
- Compilation of the required libraries
- Creation of the autostart script
- Installing the DL software
- System start

4.3.1 *Preparation of the SD-Card*

The commands in this and the following sections have all to be executed in a shell on a Linux system.

First a SD card has to be inserted into the Linux host and should not be mounted. To figure out the device name of the SD card, the following command can be used in a Linux shell:

```
sudo fdisk -l
```

The device name which belongs to the SD card is the disk which comes close to the size of the SD card and might look like this:

```
Disk /dev/sdb: 63.9 GB, 63864569856 bytes, 124735488 sectors
```

With the name, which is now known, existing partitions can be deleted before the needed partitions are created. With

```
sudo fdisk /dev/sdb
```

the program "fdisk" is started and with the following steps, existing partitions can be deleted:

```
Command (m for help):  d
Partition number (1-4): 1
Command (m for help):  d
Partition number (1-4): 2
Command (m for help):  w
```

On the now empty SD card, the first sector should first be erased.

```
sudo dd if=/dev/zero of=/dev/sdb bs=1024 count=1
```

Now the two partitions have to be created. With the following steps this can be done:

```
sudo fdisk /dev/sdb
Command (m for help): n
Command action
e extended
p primary partition (1-4)
p
Partition number (1-4): 1
First cylinder (1-123, default 1):
Using default value 1
Last cylinder or +size or +sizeM or +sizeK (1-123, default 123): +64M
Command (m for help): n
Command action
e extended
p primary partition (1-4)
p
Partition number (1-4): 2
First cylinder (10-123, default 10):
Using default value 10
Last cylinder or +size or +sizeM or +sizeK (10-123, default 123):
Using default value 123
```

The last step to create the partitions on the SD card correctly is to set the type of the first partition. This can be done with the following commands:

```
Command (m for help): t
Partition number (1-4): 1
Hex code (type L to list codes): c
Changed system type of partition 1 to c (W95 FAT32 (LBA))
* You have to format 1st partitions with vfat32 filesystem.
Command (m for help): a
Partition number (1-4): 1
```

With

```
Command (m for help): w
The partition table has been altered!

Calling ioctl() to re-read partition table.

WARNING: If you have created or modified any DOS 6.x
partitions, please see the fdisk manual page for additional
information.
Syncing disks.
```

the partition configuration is written to the SD card.

Now the partitioning is done and the partitions have to be formatted. With the commands

```
# sudo mkfs.vfat -F 32 -n boot /dev/sdb1
# sudo mkfs.ext3 -L rootfs /dev/sdb2
```

this can be done. The two new created partitions have to be mounted, to be able to have access to them. With the commands

```
mkdir /tmp/boot
mkdir /tmp/rootfs
sudo mount /dev/sdc1 /tmp/boot
sudo mount /dev/sdc2 /tmp/rootfs
```

two temporary folders are created and the two partitions are mounted to this folders.

4.3.2 *Basic Ubuntu Core installation*

Firstly, a temporary folder should be created to work in.

```
TMPDIR=‘mktemp -d‘
cd $TMPDIR
```

Now the file “boot.script” has to be created with the following content:

```
fatload mmc 0:1 0x80000000 uImage
setenv bootargs rw vram=32M fixrtc mem=1G@0x80000000 root=/dev/mmcblk0p2 console=
    ttyO2,115200n8 rootwait
bootm 0x80000000
```

With

```
mkimage -A arm -T script -C none -n "Boot Image" -d boot.script boot.scr
```

the file “boot.scr” is created. Now the Ubuntu release, the MLO, u-boot.bin and the kernel can be downloaded with the following commands:

```
wget http://cdimage.ubuntu.com/ubuntu-core/releases/12.10/release/ubuntu-core
    -12.10-core-armhf.tar.gz
wget -O MLO        http://ports.ubuntu.com/ubuntu-ports/dists/precise/main/
    installer-armhf/current/images/omap4/netboot/MLO
wget -O u-boot.bin http://ports.ubuntu.com/ubuntu-ports/dists/precise/main/
    installer-armhf/current/images/omap4/netboot/u-boot.bin
wget -O uImage     http://ports.ubuntu.com/ubuntu-ports/dists/precise/main/
    installer-armhf/current/images/omap4/netboot/uImage
```

MLO, u-boot.bin and boot.scr can now be copied to the mounted boot-partition:

```
cp MLO u-boot.bin uImage boot.scr /tmp/boot
```

Now the filesystem can be untared onto the SD card with:

```
cd /tmp/rootfs
sudo tar --numeric-owner -xzvf $TMPDIR/ubuntu-core-12.10-core-armhf.tar.gz
```

Because the temporary folder is not longer required, it can be deleted with:

```
cd
rm -rf $TMPDIR
```

4.3.3 *Modification of Ubuntu Core*

To be able to work with the installed Ubuntu Core, some modifications have to be done:

- Changing the console on serial port to a terminal
- Allow to login as “root” without the need of a password
- Getting network up and running

To be able to login and work on the Ubuntu Core system over the serial port, two files have to be added.

On the SD card's rootfs partition we go to the "init" folder with:

```
cd /tmp/rootfs
cd etc/init
```

In this directory, a new file with the extension ".conf" has to be created. The file, here "serial-auto-detect-console.conf" needs to have the following content:

```
# serial-auto-detect-console - starts getty on serial console
#
# This service starts a getty on the serial port given in the 'console' kernel
    argument.
#

start on runlevel [23]
stop on runlevel [!23]

respawn

exec /bin/sh /bin/serial-console
```

Now, a shell script has to be created, which parses the kernel command line, reads the "console" argument and launches the TTY. The file should have the same name as in its invocation in the "exec" line of the "serial-auto-detect-console.conf" file. Therefor a file named "/bin/serial-console" with "r+x" (read and execute) rights has to be created with the following content:

```
for arg in $(cat /proc/cmdline)
do
    case $arg in
        console=*)
            tty=${arg#console=}
            tty=${tty#/dev/}

            case $tty in
                tty[a-zA-Z]* )
                    PORT=${tty%%,*}

                    # check for service which do something on this port
                    if [ -f /etc/init/$PORT.conf ];then continue;fi

                    tmp=${tty##$PORT,}
                    SPEED=${tmp%%n*}
                    BITS=${tmp##${SPEED}n}

                    # 8bit serial is default
                    [ -z $BITS ] && BITS=8
                    [ 8 -eq $BITS ] && GETTY_ARGS="$GETTY_ARGS -8 "

                    [ -z $SPEED ] && SPEED='115200,57600,38400,19200,9600'

                    GETTY_ARGS="$GETTY_ARGS $SPEED $PORT"
                    exec /sbin/getty $GETTY_ARGS
            esc
    esac
done
```

To be able to login as "root" without the need of a password, in the file "/etc/shadow" the "*" character in between the semi-colons has to be deleted. The line should afterwards look like:

```
root::15259:...
```

Because the used network module on the PandaBoard doesn't have a fixed Media Access Control (MAC) address, it generates on every boot a random MAC address. To be able to get an Internet Protocol (IP) address from a DHCP server, we have to set up the network interface manually with a registered MAC address after booting. For this the file "/etc/network/interfaces" should look like this:

```
auto lo
iface lo inet loopback
auto eth0
allow-hotplug eth0
iface eth0 inet dhcp
hwaddress ether 2e:60:1c:45:3c:0a
```

After booting the PandaBoard with the SD card and login, the network can be started with the command:

```
/usr/lib/klibc/bin/ipconfig eth0
```

and Ubuntu Core obtain an IP address from the DHCP server.

To be able to connect with the internet, in the file "/etc/resolv.conf", a name server has to be defined:

```
nameserver 152.96.21.10
```

4.3.4 *Update of Ubuntu Core and installation of the required packages*

First, we have to add some repositories to "/etc/apt/sources.list":

```
# Ubuntu repositories
deb http://ports.ubuntu.com/ubuntu-ports/ precise universe multiverse
deb http://ports.ubuntu.com/ubuntu-ports/ precise-updates universe multiverse
deb http://ports.ubuntu.com/ubuntu-ports/ precise-security universe multiverse

# TI release PPA
deb http://ppa.launchpad.net/tiomap-dev/release/ubuntu precise main
deb-src http://ppa.launchpad.net/tiomap-dev/release/ubuntu precise main
```

With

```
apt-get update
apt-get dist-upgrade --yes
```

Ubuntu Core will be updated.

Additionally, at least the three packages "net-tools", "hostapd" and "udhcpd" have to be installed, to be able to operate an Access Point (AP) on the PandaBoard. This three package can be installed with:

```
apt-get install net-tools
apt-get install hostapd
apt-get install udhcpd
```

Other packages can be installed in the same way.

4.3.5 *Customization of the kernel*

Because the WLAN chip, the driver for the GPS receiver and the CAN module, which are used by the PandaBoard aren't activated in the standard Linux kernel, it is necessary to compile a customized kernel. To decrease the compilation time, this can be done on Linux desktop PC with a cross compiler like the one from Linaro[3].

First, the kernel source has to be downloaded with:

```
git clone http://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git
```

after it is downloaded, the location can be changed into the directory and the version 3.5-rc7 of the kernel can be checked out with:

```
cd linux
git checkout v3.5-rc7
```

In this version of the kernel, it is necessary to patch the kernel with the patch "0001a-omap4-pandaboard-wlan-fix.patch"[4] to solve an issue with the WLAN interface.

```
patch -p1 < 0001a-omap4-pandaboard-wlan-fix.patch
```

To build the kernel, it is important to make a good kernel configuration. For this a standard configuration[5] for the PandaBoard can be used and modified. The configuration file has to be copied into the kernel sources directory and renamed with:

```
cp Config.3.5-rc7.1 linux/.config
```

To modify the configuration "menuconfig" can be started with:

```
ARCH=arm make menuconfig
```

A screen as shown in figure 6 should come up.

It is important, that the configuration is changed as follow:

```
Device Drivers  --->
 [*] Network device support  --->
         [*]   Wireless LAN  --->
                 <M>   TI Wireless LAN support  --->
                         <M>   TI wl12xx support
                         {M}   TI wlcore support
                                 <M>     TI wlcore SDIO support
```

```
[*] Networking support  --->
        --- Wireless
        <M>   cfg80211 - wireless configuration API
                        [*]     nl80211 testmode command
```

```
Device Drivers  --->
 [*] USB support  --->
   <*> Support for Host-side USB
           <M> USB Modem (CDC ACM) support
```

3 https://launchpad.net/linaro-toolchain-binaries/trunk/2013.03/+download/gcc-linaro-arm-linux-gnueabihf-4.7-2013.03-20130313_linux.tar.bz2
4 http://elinux.org/images/8/8d/0001a-omap4-pandaboard-wlan-fix.patch
5 http://elinux.org/images/1/14/Config.3.5-rc7.1

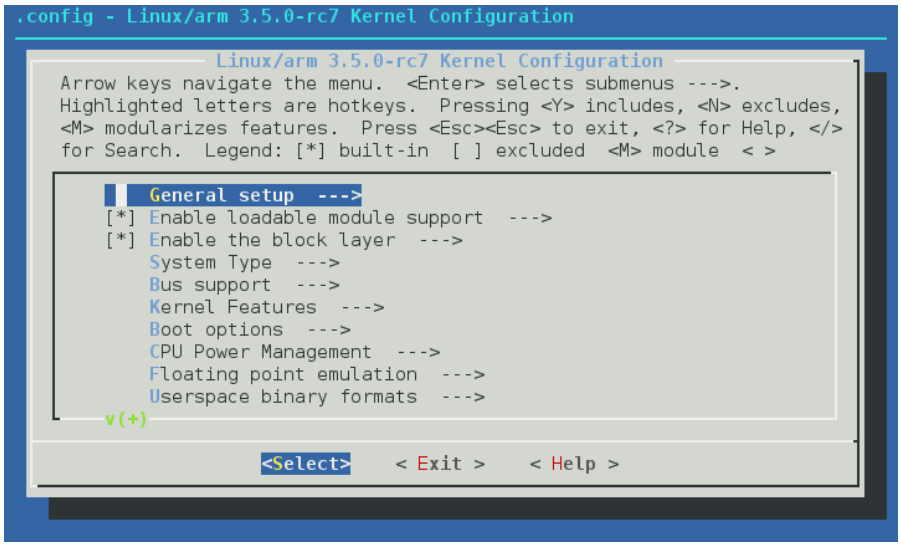


Figure 6: Menuconfig

```
[*] Networking support  --->
  <M> CAN bus subsystem support --->
    <M> Raw CAN Protocol (raw access with CAN-ID filtering)
    <M> Broadcast Manager CAN Protocol (with content filtering)
    CAN Device Drivers --->
      <M> Virtual Local CAN Interface (vcan)
      <M> Platform CAN drivers with Netlink support
        [*] CAN bit-timing calculation
        <M> Microchip MCP251x SPI CAN sontrollers
      CAN USB interfaces --->
        <M> PEAK PCAN-USB/USB Pro interfaces
```

this enables the driver for the WLAN interface, the driver for the GPS receiver and the driver for the CAN interface.

After the configuration is done, the kernel can be compiled with:

```
make ARCH=arm CROSS_COMPILE=/path/to/the/cross_compiler uImage
```

and the modules with:

```
make ARCH=arm CROSS_COMPILE=/path/to/the/cross_compiler modules
```

In this and the following sections, "/path/to/the/cross_compiler" has to be replaced with the real path to the cross compiler. With the installed cross compiler from Linaro in /opt/ the path is for example: "/opt/gcc-linaro-arm-linux-gnueabihf-4.7-2013.03-20130313_linux/bin/arm-linux-gnueabihf-"

With

```
INSTALL_MOD_PATH=../modules make ARCH=arm CROSS_COMPILE=/path/to/the/
    cross_compiler modules_install
```

the modules are afterward located in "../modules/lib/modules/3.5.0-rc7-dirty/".

Now the compiled kernel and the modules can be copied to the partitions on the SD card.

```
cp arch/arm/boot/uImage /tmp/boot
sudo cp -r ../modules/lib/modules/3.5.0-rc7-dirty /tmp/rootfs/lib/modules
```

As the last step, the required firmware for the WLAN module has to be downloaded and copied to the SD card:

```
git clone git://git.kernel.org/pub/scm/linux/kernel/git/dwmw2/linux-firmware.git
sudo cp -r linux-firmware/ti-connectivity/ /tmp/rootfs/lib/firmware/
```

To check if the WLAN module is recognized correctly, the following command can be used on the PandaBoard:

```
ifconfig -a
```

If the WLAN interface #0 appears, the WLAN module is recognized successfully.

4.3.6 *Configuration of the required packages*

The package "hostapd", which provides the AP functionality has to be configured. For this the two files "/etc/default/hostapd" and "/etc/hostapd.conf" has to be modified.

"/etc/default/hostapd" should look like:

```
# Defaults for hostapd initscript
#
# See /usr/share/doc/hostapd/README.Debian for information about alternative
# methods of managing hostapd.
#
# Uncomment and set DAEMON_CONF to the absolute path of a hostapd configuration
# file and hostapd will be started during system boot. An example configuration
# file can be found at /usr/share/doc/hostapd/examples/hostapd.conf.gz
#
DAEMON_CONF="/etc/hostapd.conf"
RUN_DAEMON=YES
```

This tells "hostapd", that the configuration file is "/etc/hostapd.conf" and that it runs as a daemon.

"/etc/hostapd.conf" has to be changed, that it looks like:

```
interface=wlan0
driver=nl80211
ssid=DESSportec1
hw_mode=g
channel=8
```

This configures "hostapd", that it runs in the IEEE802.11g mode, on channel 8 with the SSID "DESSportec1".

That the DHCP server runs the right way, the file "/etc/uhdcpd.conf" has to be modified, that it looks like:

```
# The start and end of the IP lease block

start           192.168.1.1     #default: 192.168.0.20
end             192.168.1.5     #default: 192.168.0.254


# The interface that udhcpd will use
```

```
8
9 interface       wlan0           #default: eth0
```

This defines the start and the end of the IP lease block and sets the WLAN interface as the interface on which the DHCP server will run.

That the DHCP server runs as a daemon, the file "/etc/default/udhcpd" has to be modified, that it looks like:

```
1 # Comment the following line to enable
2 DHCPD_ENABLED="yes"
3
4 # Options to pass to busybox' udhcpd.
5 #
6 # -S    Log to syslog
7 # -f    run in foreground
8
9 DHCPD_OPTS="-S"
```

The WLAN interface also needs an IP address, that it can receive packages from the other nodes. For this the following lines has to be added to "/etc/network/interfaces":

```
1 auto wlan0
2 iface wlan0 inet static
3 address 192.168.1.100
```

and with

```
1 ifconfig wlan0 up
```

the WLAN interface gets started.

4.3.7 *Compilation of the required libraries*

The data logger software requires three libraries to work correctly. The first is the library "libconfig" written by Mark Lindner.

Libconfig is a simple library for processing structured configuration files. This file format is more compact and more readable than XML. And unlike XML, it is type-aware, so it is not necessary to do string parsing in application code.

Libconfig is very compact - a fraction of the size of the expat XML parser library. This makes it well-suited for memory-constrained systems like handheld devices.

The library includes bindings for both the C and C++ languages. It works on POSIX-compliant UNIX systems (GNU/Linux, Mac OS X, Solaris, FreeBSD) and Windows (2000, XP and later).[18]

The second library, which is used is "libxbee". It is a C/C++ library to aid the use of Digi XBee radios in API mode.[11] As discussed in the analysis, the XBee radio module is the chosen 802.15.4 module.

"NMEA Library" is the third used library. We present open source and free library in 'C' programming language for work with NMEA protocol. Small and easy to use.[8]

LIBCONFIG To compile "libconfig" for the PandaBoard, it first has to be downloaded[6]. After downloading the archive it has to be extracted and the folder has to be entered with:

```
wget www.hyperrealm.com/libconfig/libconfig-1.4.9.tar.gz
tar -xvf libconfig-1.4.9.tar.gz
cd libconfig-1.4.9
```

With the two commands:

```
export PATH=$PATH:/path/to/the/cross_compiler
./configure --host=arm-linux-gnueabihf
```

in the libconfig folder, the library will be compiled for the PandaBoard. The now generated library files have to be copied to the library folder on the PandaBoard and also to the library folder in the cross compiler environment. On the PandaBoard, the libraries have to be copied to "/usr/lib/" and the header files to "/usr/include/". This can be done with commands like:

```
sudo cp *.h /tmp/rootfs/usr/include/
sudo cp *.o /tmp/rootfs/usr/lib/
sudo cp .libs/*.so* /tmp/rootfs/usr/lib
```

This commands have to be executed in the sub directory "lib". To copy the header files and the libraries to the cross compile environment, the commands:

```
sudo cp *.h /path/to/the/cross_compiler/include/
sudo cp *.o /path/to/the/cross_compiler/lib/
sudo cp .libs/*.so* /path/to/the/cross_compiler/lib
```

can be executed.

LIBXBEE To get the source code of the "libxbee" library, "git" can be used, to clone the whole repository:

```
git clone https://code.google.com/p/libxbee.libxbee-v3/
```

after the libxbee repository is downloaded, the directory has to be entered and a first configuration must be generated with:

```
cd libxbee.libxbee-v3
make configure
```

Now a default configuration exists which has to be changed, that the library can be compiled for the PandaBoard with the cross compiler. For this, two options have to be set in the file "config.mx":

```
CROSS_COMPILE?= /path/to/the/cross_compiler
OPTIONS+=       XBEE_NO_RTSCTS
```

With the first option, the cross compiler for the PandaBoard is defined. The option "XBEE_NO_RTSCTS" disables the flow control of the UART connection. This is important because the XBee module is

6 www.hyperrealm.com/libconfig/libconfig-1.4.9.tar.gz

connected to the PandaBoard only with its data connections. Without this option the software would tell, that no connection could be established because a timeout occurred.

The librarie can now be compiled with:

```
make
```

The compiled library files are now in the subfolder "lib". From there they have to be copied to the cross compiler environment and also to the PandaBoard include directory. With:

```
cp libxbee.* /path/to/the/cross_compiler/lib/
cp libxbeep.* /path/to/the/cross_compiler/lib/
cp ../xbee.h /path/to/the/cross_compiler/include/
cp ../xbeep.h /path/to/the/cross_compiler/include/
```

the library object files and the header files are copied to the cross compiler environment. On the PandaBoard, the libraries have to be copied to "/usr/lib/" and the header files to "/usr/include/". This can be done with the commands:

```
sudo cp ../xbee.h /tmp/rootfs/usr/include/
sudo cp ../xbeep.h /tmp/rootfs/usr/include/
sudo cp ./libxbee.* /tmp/rootfs/usr/lib/
sudo cp ./libxbeep.* /tmp/rootfs/usr/lib/
```

LIBNMEA The "NMEA library" sources can be downloaded from SourceForge[7] and the archive extracted with the command:

```
unzip nmealib-0.5.3.zip
```

After changing into the directory with the command:

```
cd nmealib
```

in the Makefile, the compiler path "CC" has to be changed to the path of the cross compiler:

```
CC = /path/to/the/crosscompiler
```

Now the library can be compiled with the following command:

```
make all
```

In the last step, the header and the created library files have to be copied to the PandaBoard and also to the cross compiler environment. This can be done with the commands:

```
cp -r build/nmea_gcc /mnt/rootfs/usr/lib
cp lib/libnmea.a /mnt/rootfs/usr/lib
cp -r include/nmea /mnt/rootfs/usr/include

sudo cp -r build/nmea_gcc /path/to/the/cross_compiler/lib/
sudo cp lib/libnmea.a /path/to/the/cross_compiler/lib/
sudo cp -r include/nmea /path/to/the/cross_compiler/include/
```

7 http://downloads.sourceforge.net/nmea/nmealib-0.5.3.zip?use_mirror=switch

4.3.8 *Creation of the autostart script*

That the mobile data logger software can run correctly, some system settings in Ubuntu Core have to be done:

- Change of GPIO #113 to "Mux Mode" #3, that the button can be used.
- Activating the GPIO #113 with the direction "in", triggering on rising edges.
- Providing write and read access to the GPIO #113, where the button is attached.
- Providing write and read access to the UART #4, where the XBee module is connected.
- Providing read access to the GPS receiver.
- Starting the DL software in the "real-time" mode.

This is realized with an autostart script. This script has to be in "/etc/init.d/autostart.sh" with the following content:

```
1  echo 0x011B > /sys/kernel/debug/omap_mux/abe_mcbsp2_fsx
2  echo 113 > /sys/class/gpio/export
3  echo in > /sys/class/gpio/gpio113/direction
4  echo rising > /sys/class/gpio/gpio113/edge
5
6  chmod 666 /sys/class/gpio/gpio113/value
7  chmod 666 /dev/ttyO3
8  chmod 444 /dev/ttyACM0
9
10 chrt 99 /usr/bin/BA-ESW13.01
```

That this autostart script works correctly, it needs execution privileges which can be achieved with the command:

```
1 chmod +x /etc/init.d/autostart.sh
```

After this script can be executed, Ubuntu Core has to be informed about the new script. This can be done with:

```
1 update-rc.d autostart.sh defaults
```

Now this script will automatic be started when Ubuntu Core boots.

4.3.9 *Installing the DL software*

To install the DL software, the program "BA-ESW13.01" from the directory "bin" of the enclosed CD has to be copied to the directory "/usr/bin" on the PandaBoard.

Part III

DESIGN

5 DESIGN

5.1 ENCAPSULATION

The classes of the data logger software are encapsulated in four categories. The category "system" contains the classes "DLCtrl", "DAQCtrl", "Time" and "ConfigCtrl" which form the kernel of the data logger software. In the category "com" are all the classes included, which are responsible for the communication between the data logger and the DAQ modules, the XBee module and the GPS receiver. The category "data" covers the class "SaveData" which handles the saving of the data and the class "RecvBuffer" which is a circular buffer. The encapsulation concept is shown in figure 7.

The classes organized in categories.

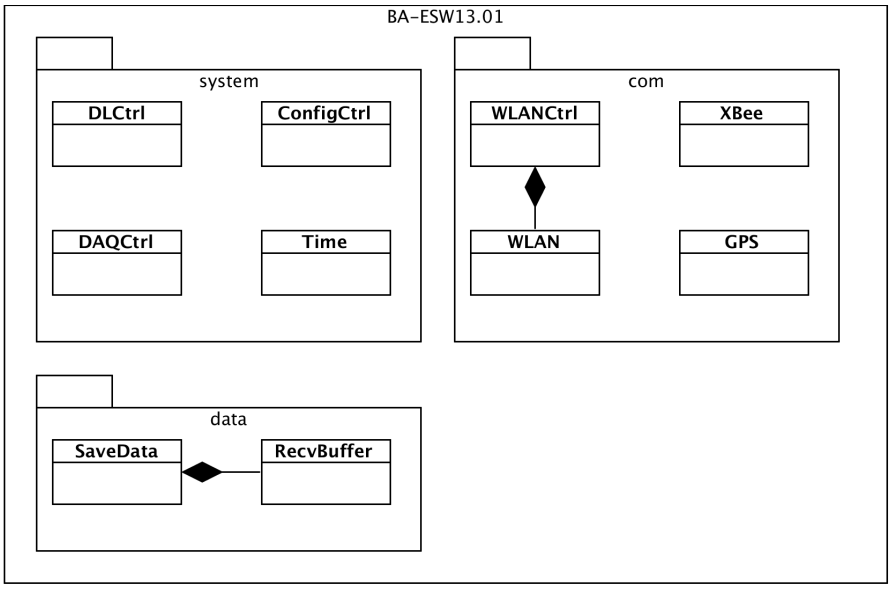


Figure 7: Encapsulation

5.2 CLASS DIAGRAM

The class diagram is shown in figure 8 and describes the structure and the integration of the classes. To keep a better overview some parameter are omitted. The description of the various classes will follow in the next sections.

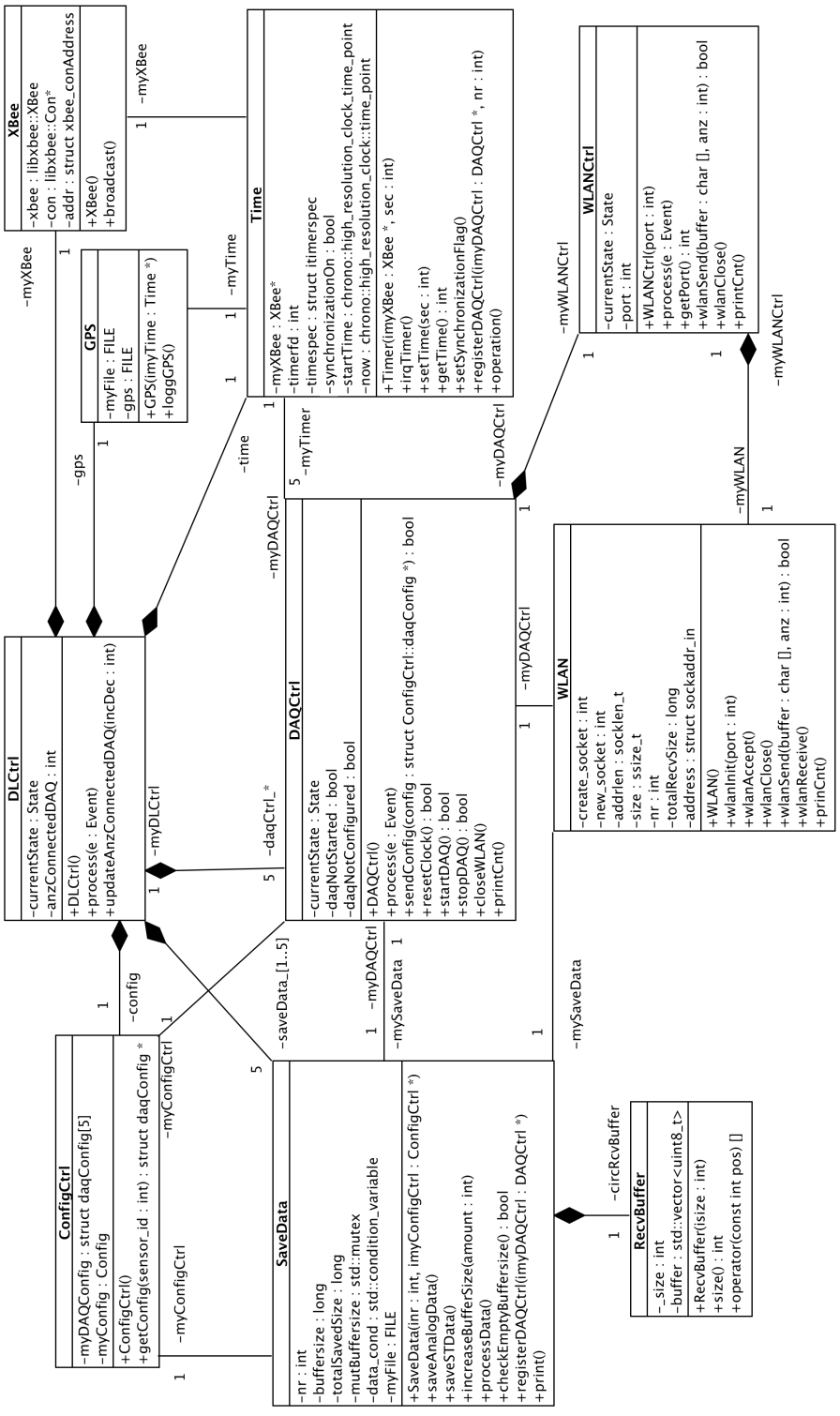


Figure 8: Class diagram

5.3 THREAD PRIORITIES

To provide the fastest possible hardware access time, the DL software gets started in the "real-time" mode with the highest priority level 99 by the command:

```
1 chrt 99 /usr/bin/BA-ESW13.01
```

Next to this, the synchronization of the DAQ modules, the receiving of the sensor data, the processing of the received data and the receiving of the positions from the GPS receiver and saving them are executed with different thread priorities to guarantee a smooth operation of the DL software.

The different threads with their priority are listed in table 9.

Function	Name of the class & method	Priority level
Synchronization	Time::*irqTimer()*	99
Receiving	WLAN::*wlanReceive()*	90
Processing	SaveData::*processData()*	80
GPS	GPS::*loggGPS()*	default

Table 9: The different threads with their priority

5.4 START UP AND PROGRAM EXIT

The button triggers an interrupt, hence no polling loops are required.

When the data logger software gets started, it first waits until the button is pushed. If the user pushed the button, an object of the class "DLCtrl" is created. After the object of the class "DLCtrl" is created, the event "evDLInitFinished" gets triggered, which turns the DL into the state "sDLrunning". If the button is pushed again, the event "evDLButtonPushed" gets triggered and the DL switches to the state "sDLstopped". The DL software waits until all the received data could be saved and after the program stops. A life cycle of the DL software in an activity diagram is shown in figure 9.

5.5 DL CONTROL - DLCTRL

The class "DLCtrl" is the main class, which controls the whole data logger. An object of the class "DLCtrl" is created in the main function, from there the object respective its FSM is controlled. From the class "DLCtrl" the other objects of the classes "DAQCtrl", "ConfigCtrl", "SaveData" and "Time" are created. Also the FSM of the objects of the class "DAQCtrl" are started from the "DLCtrl" class. Next to the FSM the class "DLCtrl" provides the method *updateAnzConnectedDAQ* which other classes can call to keep the number of connected DAQ modules actually.

UPDATEANZCONNECTEDDAQ This method simply increases the variable *anzConnectedDAQ* by one, if it is called with the parameter *INCREASE* or decreases this variable by one if the parameter is *DECREASE*.

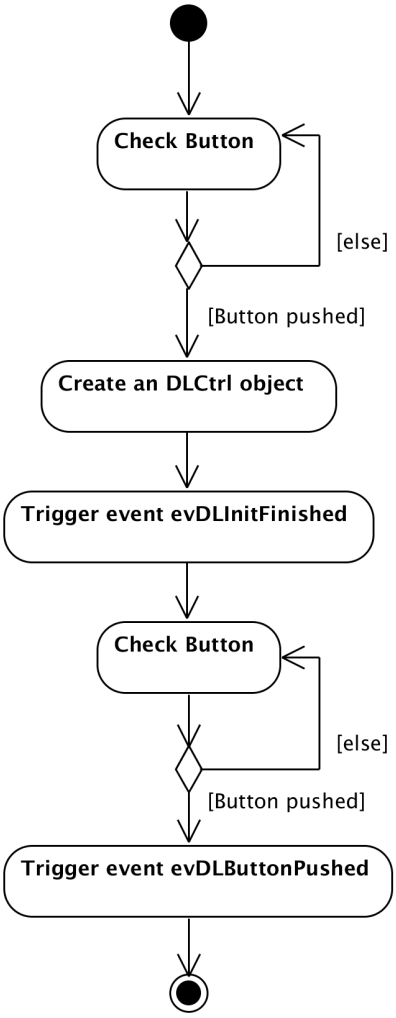


Figure 9: Start up routine

5.5.1 *DLCtrl FSM*

With the creation of an object of the class "DLCtrl", the configuration for the DAQ-modules is loaded, five objects of the class "DAQCtrl" and five objects of the class "SaveData" are created. A "Time" object is also created which initializes a timer. After this initialization, the FSM of "DLCtrl" goes to the state "sDLinit". When the event "evDLInitFinished" occurs, which is triggered by the function *main()*, the methods *processData()* of the class "SaveData" are started, each in a separate thread, the method *irqTimer()* from the class "Time" is started in an own thread and the FSM of the objects of the class "DAQCtrl", one for every WLAN connection to a DAQ module gets started, each running in a separate thread. Over this WLAN connections the DAQ modules get configured, their clock reseted and the measurement started and stopped. The "DLCtrl" FSM then changes to the state "sDLrunning" where it stays until the event "evDLButtonPushed" occurs. This

event is triggered by the function *main()*, when the button get pushed. When the event "evDLButtonPushed" occurs the FSM turns into the state "sDLstopped" and the event "evDAQStop" gets triggered, to stop the DAQ modules. When all the DAQ modules have stopped the measurement and the received data could be saved, the program exits. The FSM of the class "DLCtrl" is shown in figure 10.

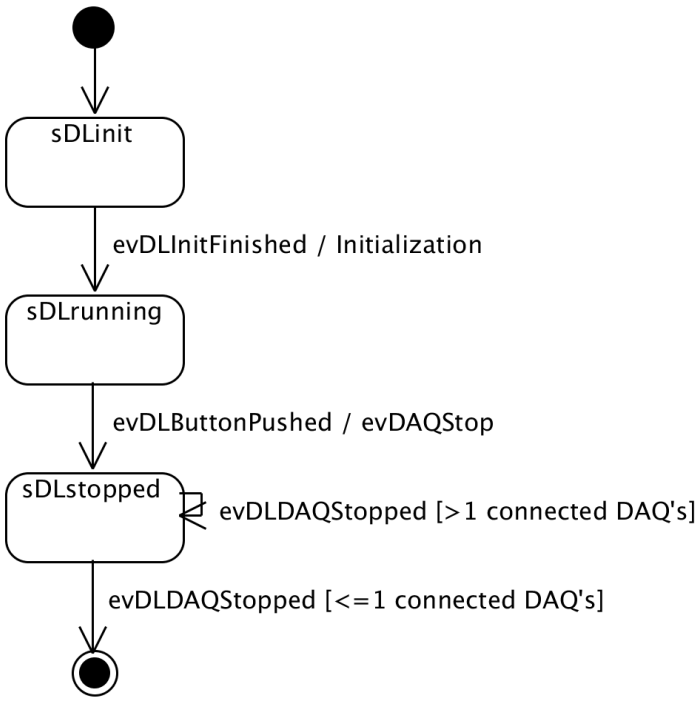


Figure 10: FSM of the class DLCtrl

5.6 CONFIGURATION - CONFIGCTRL

The "ConfigCtrl" class initializes *myDAQConfig*, an array of objects of the structure *daqConfig*. Next to this, the class "ConfigCtrl" provides the method to load the configurations out of the configuration file and to save them in *myDAQConfig* and the method to return a *daqConfig* structure to the requesting objects.

LOADCONFIG This method opens the configuration file "DAQ.cfg" reads its content, parse it and saves the settings for each DAQ module into the corresponding element of the array *myDAQConfig* with the help of the library "libconfig".

GETCONFIG *getConfig()* returns the pointer to the desired structure in the array *myDAQConfig* to the caller.

DAQCONFIG The structure *daqConfig*, shown in figure 11, in which the configuration for a DAQ module is stored, contains two arrays.

The first array contains six elements of the type structure *sensor_a* for the analog sensors and the second array contains two elements of the type structure *sensor_st* for the Schmitt trigger sensors. Next to this two arrays of structures, *daqConfig* also holds the variable *configuredSensors*.

SENSOR SETTINGS The structure *sensor_a* represents the configuration for the analog sensors and contains the variables *sensor_id* and *sample_rate*. In the structure *sensor_st* the variables *sensor_id*, *high_voltage* and *low_voltage* represent the settings for the Schmitt trigger sensors. Table 10 gives an overview over this sensor settings.

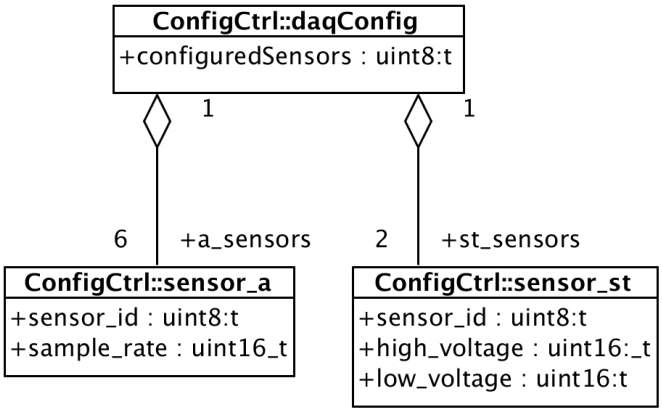


Figure 11: *daqConfig* structure

Structure name	Variable	Description
sensor_a	sensor_id	Sensor ID
sensor_a	sample_rate	Sample rate of the sensor
sensor_st	sensor_id	Sensor ID
sensor_st	high_voltage	Threshold voltage (high)
sensor_st	low_voltage	Threshold voltage (low)

Table 10: Sensor settings

CONFIGUREDSENSORS The variable *configuredSensors* has a bit pattern, which indicates which sensors will be configured and which not. Table 11 shows some examples of the bit pattern rule.

CONFIGURATION FILE The format of the configuration file is explained in appendix D.

Sensor ID	Bit pattern
1	0x0000'0001
4	0x0000'1000
5, 6	0x0011'0000

Table 11: Examples which show the rule of the bit pattern

5.7 DAQ CONTROL - DAQCTRL

The class "DAQCtrl" represents the FSM which controls a DAQ module. There are five objects of the class "DAQCtrl", one for every DAQ module. All of this five objects are owned by the main class "DLCtrl" and their FSM also gets started from there. Next to the FSM, the class "DAQCtrl" also provides the methods to send the configuration, to reset the clock, to start and stop the measurement on the DAQ module, to close the WLAN connection and to print the amount of received data.

START UP The start up routine of a DAQ module, shown in figure 12, sends the configuration to the DAQ module, resets the clock, synchronizes the DAQ modules and starts the measurement.

Each of the method process() with an event "Start" as parameter, in the figures 12, 13 and 14 is executed in a separate thread.

RESTART If the WLAN connection to a DAQ module is lost, the active socket gets closed and with the call of the method *wlanAccept()* from the class "WLAN", the socket is again ready for an incoming connection. An incoming connection would be accepted, afterwards the clock of the DAQ module is reseted the DAQ modules synced and the measurement on the DAQ module is started. The whole process is shown in figure 13.

STOP If the user pushes the button during the measurement, the measurement on the DAQ module gets stopped. After the class "WLAN" notifies a lost connection, the receive buffer gets checked if it is empty, by calling the method *checkEmptyBuffersize()* from the class "SaveData". If the receive buffer is empty, the class "DLCtrl" gets informed, otherwise the left data will be saved and the buffer is checked again later. If the amount of connected DAQ modules is lower or equal to one and the class "DLCtrl" gets notified by the event "evDLDAQStopped", the amount of received and saved data will be printed and afterwards the program stops. This whole process is shown in figure 14.

SENDCONFIG To send the configuration to the DAQ module, first the size of the send buffer has to be calculated. After that, the configuration for the two kind of sensors are copied and before sending it, the header is created and also copied to the send buffer. For send-

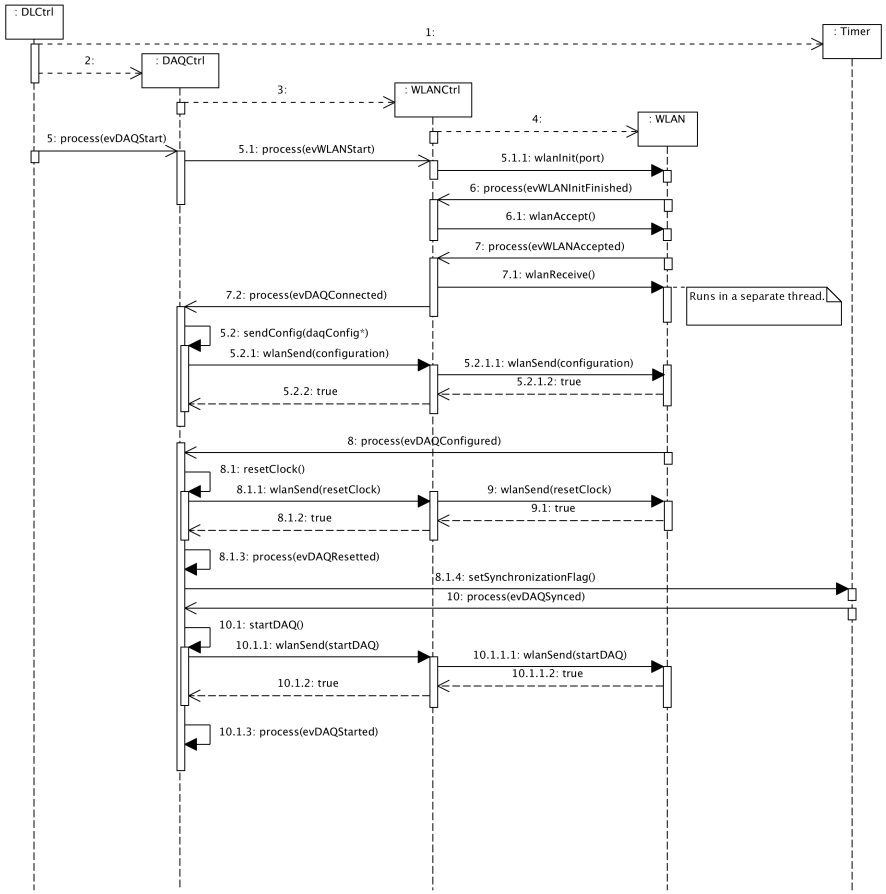


Figure 12: Start up of a DAQ module

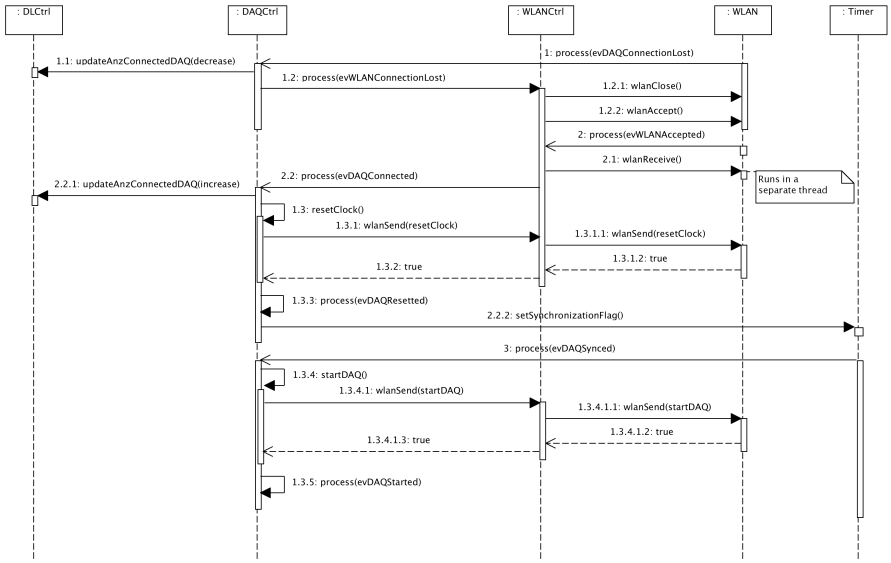


Figure 13: Restart of a DAQ module

ing the configuration to the DAQ module, the method *wlanSend()* from the corresponding "WLANCtrl" object is executed. This method itself

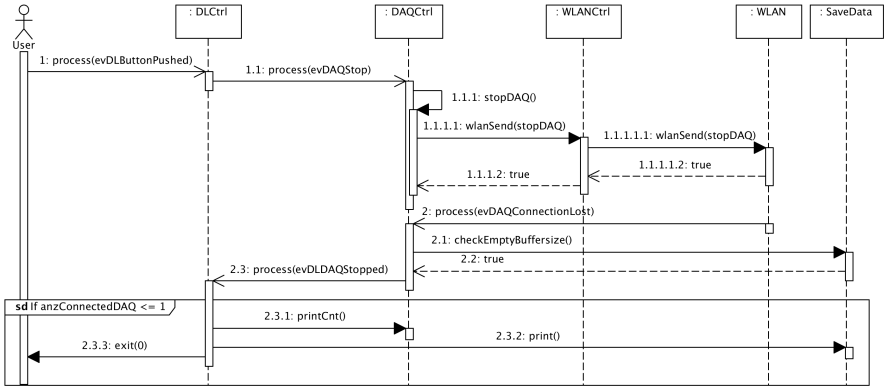


Figure 14: Stop of a DAQ module

calls the method *wlanSend()* from the corresponding "WLAN" object. This configuration transfer is shown in figure 15.

The transfer process to send the configuration to a DAQ.

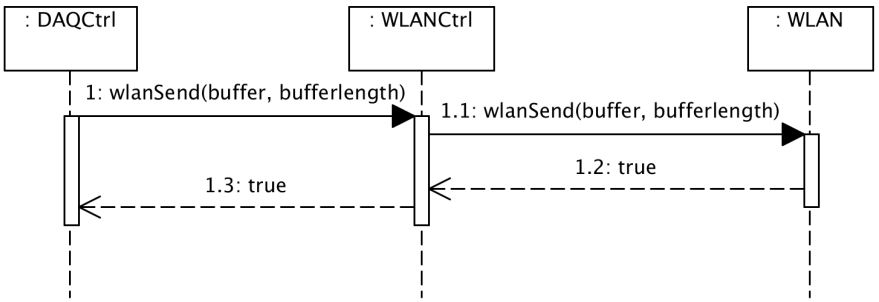


Figure 15: Configuration transfer

RESETCLOCK To reset the clock of a DAQ module, the corresponding command is sent to the DAQ module, by executing the *wlanSend()* method of the corresponding "WLANCtrl" object.

STARTDAQ/STOPDAQ Also to start and stop the measurement on a DAQ module, the method *wlanSend()*, from the class "WLANCtrl" is used to send the corresponding commands.

WLANCLOSE *wlanClose()* calls the same named method from the class "WLANCtrl" to close the corresponding WLAN connection.

PRINTCNT This method simply calls the same named method *printCnt()* of the class "WLANCtrl" to print the amount of data which is received.

5.7.1 *DAQCtrl FSM*

When an object of the class "DAQCtrl" is created, the FSM stays in the state "sDAQidle" until it gets started by the event "evDAQStart". Af-

ter this, the FSM changes to the state "sDAQconnect" and waits for the event "evDAQConnected" which is triggered by the class "WLAN". This changes the FSM to the state "sDAQconfig", where the configuration is sent to the DAQ module with the method *sendConfig()* provided by the same class. If the configuration failed, the class "WLAN", which receives the messages from the DAQ module, triggers the event "evDAQMissConfigured", which forces the class "DAQCtrl" to send the configuration again. If the class "WLAN" triggers the event "evDAQConfigured" after it receives an acknowledgement that the configuration was successful, the FSM changes to the state "sDAQresetClock" where the method *resetClock()* from the same class is executed, to reset the clock of the DAQ module. After the clock is reseted, the synchronization flag in the class "Time" is set with the method *setSynchronizationFlag()* provided by the class "Time" to synchronize the DAQ modules and the FSM changes to the state "sDAQtimeSync". After a synchronisation happened and if the measurement on the DAQ-module is not started yet, the event "evDAQSynced" changes the FSM to the state "sDAQstart" where the method *startDAQ()* sends the start command to the DAQ module. After the measurement on the DAQ module is started the FSM changes to the state "sDAQtimeSync", stays there and every 250 ms a time synchronisation happens. If the class "WLAN" triggers the event "evDAQConnectionLost" during the state "sDAQstart" or "sDAQtimeSync" the FSM changes to the state "sDAQconnect" and triggers the event "evWLANConnectionLost". When the event "evDAQStop" occurs in the state "sDAQstart" or "sDAQtimeSync", the own method *stopDAQ()* will be called and the FSM turns to the state "sDAQstop". If the event "evDAQStop" occurs in one of the states "sDAQconnect", "sDAQconfig", "sDAQresetClock" or "sDAQtimeSync" and the DAQ module was not started yet, the own method *closeWLAN()* is called and the FSM turns to the state "sDAQstop". If the event "evDAQConnectionLost" occurs during the state "sDAQstop" and all the received data could be saved, the FSM switches to the final state. To simplify further adaptations, a restart of the WLAN connection to the DAQ module is implemented, if the event "evDAQStarted" occurs in the state "sDAQstop". The FSM of the class "DAQCtrl" is shown in figure 16.

5.8 WLAN CONTROL - WLANCTRL

An object of the class "WLANCtrl" controls and maintains a WLAN connection with a FSM, shown in figure 17 and provides the methods to get the number of the port, to send messages to the corresponding DAQ module, to close the WLAN connection and to print the amount of received data. There are five WLAN connections, one per DAQ module. Each object of the class "WLANCtrl" is owned by its corresponding "DAQCtrl" object.

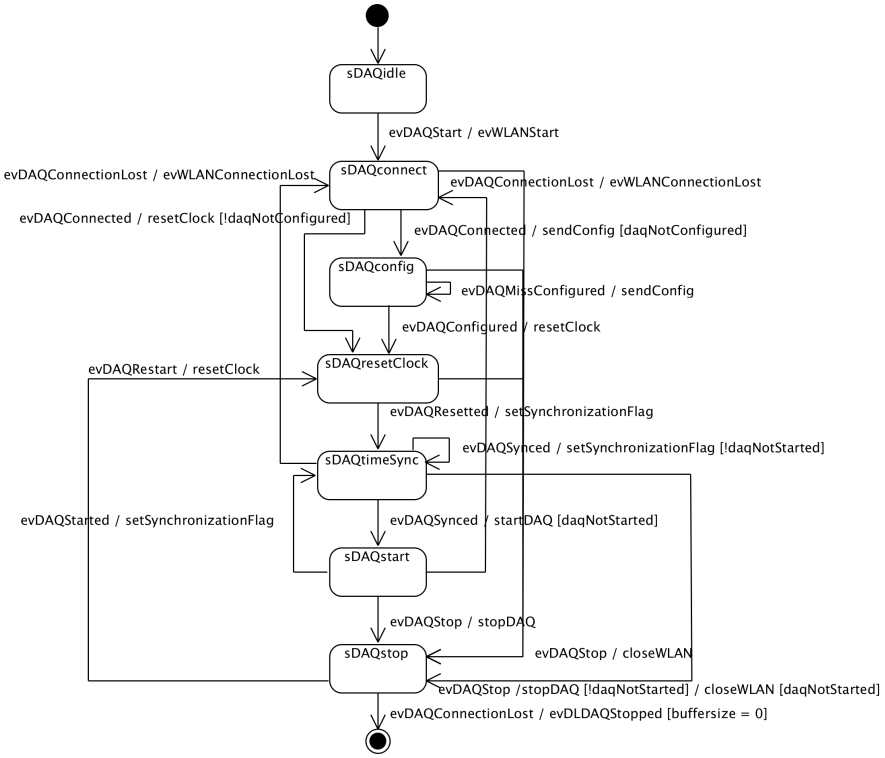


Figure 16: FSM of the class DAQCtrl

GETPORT This method provides the information of the port number of the WLAN connection and returns this if its called.

WLANSEND The method *wlanSend()* which provides the functionality to send messages to the DAQ module receives the message in a char array and its size, both via parameter and forward them to the method *wlanSend()* of the corresponding "WLAN" class object.

WLANCLOSE *wlanClose()* calls the same named method from the class "WLAN" to close the corresponding WLAN connection.

PRINTCNT *printCnt()* calls the method *printCnt()* of the class "WLAN" to print the amount of received data.

5.8.1 *WLANCtrl FSM*

With the creation of an object of the class WLANCtrl, the FSM is in the state "sWLANidle" until the corresponding "DAQCtrl" object triggers the event "evWLANStart". This calls the method *wlanInit()* of the class "WLAN" to initialize the WLAN connection and changes the FSM to the state "sWLANinit". If the WLAN connection is initialized successfully, the class "WLAN" triggers the event "evWLANInitFinished" which calls the method *wlanAccept()* from the class "WLAN"

and changes the FSM into the state "sWLANaccept". There it waits for the event "evWLANAccepted" which indicates, that an incoming connection was accepted by the class "WLAN". With an accepted connection the event "evDAQConnected" will be triggered, the method *wlanReceive()* is executed in a new thread and the FSM changes and stays in the state "sWLANconnected". Now messages can be sent to and received from the DAQ module, and the FSM can change back to the state "sWLANaccept" if the connection gets lost. This would also call the methods *wlanClose()* and *wlanAccept()* from the class "WLAN".

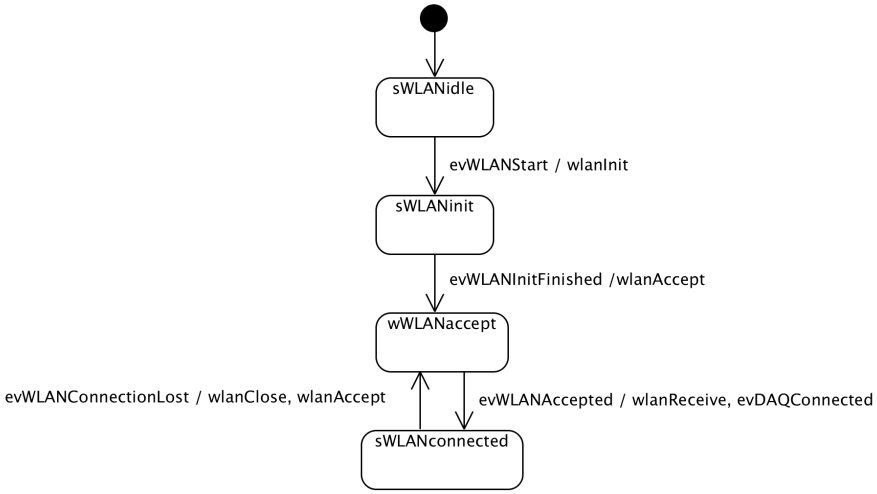


Figure 17: FSM of the class WLANCtrl

5.9 WLAN

The class "WLAN" provides the basic functionalities to establish a WLAN connection and to send and receive messages over this connection. There exist five objects of the class "WLAN" each owned by its corresponding object of the class "WLANCtrl".

WLANINIT The method *wlanInit()* of the class "WLAN" creates and binds a socket to a chosen port. If the binding of the socket was successful, the event "evWLANInitFinished" is triggered.

WLANACCEPT *wlanAccept()* listens for a connection on the created socket and accepts it, if there is one and triggers the event "evWLANAccepted".

WLANCLOSE To close the WLAN connection, the method *wlanClose()* closes the created socket if it is not already closed and triggers the event "evDAQConnectionLost".

WLANSEND The method *wlanSend()* takes a char array and its length as input parameter and writes this message to the socket.

PRINTCNT This method prints the amount of received data.

5.9.1 *Receiving messages*

To receive messages from the DAQ module, the method *wlanReceive()* is executed in the state "sWLANconnected" in the FSM of the class "WLANCtrl", as a separate thread. The received data is temporary saved to a circular receiving buffer, owned by the class "SaveData". If for more than 60 seconds no data could be received, the event "evDAQConnectionLost" is triggered to indicate a lost connection.

If a message is received, the header is extracted, to decide what kind of message it is and how to handle it:

- A received *ConfigSuccessful* header triggers the event "evDAQConfigured".
- If a *ConfigUnsuccessful* header is received, the event "evDAQMissConfigured" is triggered.
- If a *AnalogData* header is received, the corresponding "SaveData" object is informed about the amount of data which is received by calling the method *increaseBufferSize()* with the amount of data as parameter.
- A received *STData* header also informs the corresponding "SaveData" object about the amount of data which is received by calling the method *increaseBufferSize()* with the amount of data as parameter.

The whole process is shown in figure 18.

5.10 SAVING DATA - SAVEDATA

With the creation of an object of the class "SaveData" a "DAQ<1..5>.csv" file gets created, in which later the received sensor data will be saved.

SAVEANALOGDATA/SAVESTDATA The two methods which save the sensor data to the files "DAQ<1..5>.csv", *saveAnalogData()* and *saveSTData()* first read the configuration. The configuration is needed to calculate the time stamp for every measurement point with the time difference t_{diff} between the samples because a received data packet contains only the time stamp for the first measurement point in the packet. From the configuration the sample rate f_s of each configured sensor is known and so the time difference between the sam-

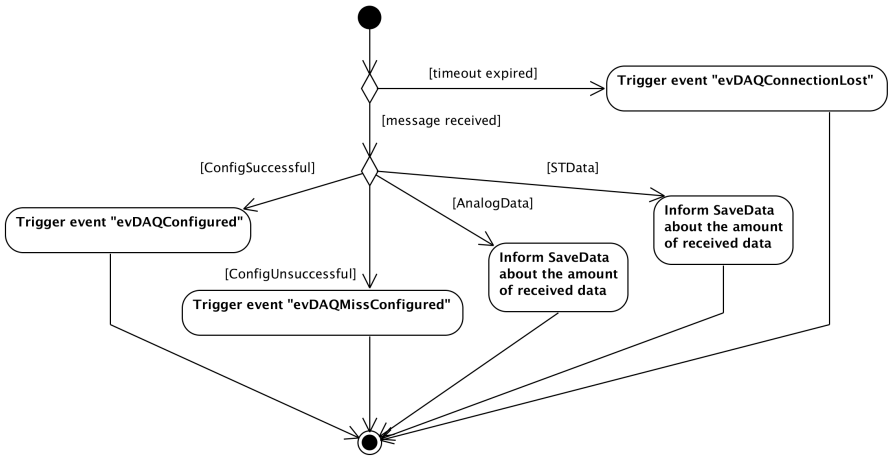


Figure 18: Activity diagram of the method *wlanReceive()*

ples, t_{diff}, with a resolution of 100µs, can be calculated with the equation:

$$t_{diff} = \frac{10000}{f_s} \tag{1}$$

The time stamp for every measurement point is then calculated with the equation:

$$t_n = t_{first\ measurement\ point} + n * t_{diff} \tag{2}$$

where

n Number of the measurement point in the packet

In the final step, the data is written to the "DAQ<1..5>.csv" file in the following format:

For the analog sensor data:

```
1       analog,sensor<n>,time,analog data;
2       analog,sensor<n>,time,analog data;
3       ..
```

and for the analog sensor data with Schmitt trigger data:

```
1       analog,sensor<n>,time,analog data,ST,ST data;
2       analog,sensor<n>,time,analog data,ST,ST data;
3       ..
```

INCREASEBUFFERSIZE The method *wlanReceive()* from the class "WLAN" calls this method *increaseBufferSize()* to inform the class "SaveData" about the amount of received data which is saved in the circular buffer and ready to be processed.

PROCESSDATA This method is called from the class "DLCtrl" in a own thread. It processes the data received and saved to the circular receive buffer by the method *wlanReceive()* from the class "WLAN". This method runs during the entire life of the program and checks, if there is data in the circular receive buffer. If this is the case, the received data gets processed until the receive buffer is empty. In the case, that the circular receive buffer contains no data, the thread goes to sleep until there is data in the circular receive buffer.

If there is data in the circular receive buffer, first the header get extracted, to decide, how the data has to be processed:

- If the received message contains analog sensor data, a *AnalogData* header is received. The time stamp and the analog data is then passed to the method *saveAnalogData()*.
- A received *STData* header announces Schmitt trigger and analog sensor date. The method *saveSTData()* will be called with the data and the time stamp as argument.

REGISTERDAQCTRL Because the "SaveData" objects are created before the objects of the class "DAQCtrl" exist, the association with the corresponding "DAQCtrl" object can't be done with the constructor and has to be done later. This method *registerDAQCtrl()* let the "DAQCtrl" objects building this association.

CHECKEMPTYBUFFERSIZE This method *checkEmptyBuffersize()* is used by the class "DAQCtrl" to check if all the received data could be saved. If there isn't any data in the circular receive buffer anymore, "true" is returned. Otherwise, the method goes to sleep for 100 ms to give the method *processData()* time to save the remaining data and "false" is returned after the event "evDAQConnectionLost" is triggered.

PRINT This method prints the number of the corresponding DAQ module, the amount of data in the circular receive buffer and the amount of data which could be saved.

5.11 CIRCULAR RECEIVE BUFFER - RECVBUFFER

The received data from the DAQ modules can't be stored directly to a file in the method *wlanReceive()* from the class "WLAN" because timing problems would occur and data would be lost. Therefore the received data is first saved temporary in a circular receive buffer. The method *processData()* from the class "SaveData" later reads the data from the circular receive buffer and saves it to a file, when enough

hardware resources are available. This class "RecvBuffer" provides this circular receive buffer and the access to it.

ACCESS VIA [ ] OPERATOR The [ ] operator returns the reference to the item of the desired position in the circular buffer, if the desired position $p_{desired}$ is smaller then twice the size of the buffer s_{buffer} ($p_{desired} \leqslant 2 * s_{buffer}$). This allows a single overflow, which is controlled by the methods, which access the circular buffer.

5.12 XBEE

The class "XBee" sets up the connection to the XBee module over the UART interface #4 on the PandaBoard and defines the broadcast address as the destination address for messages. This class also provides the functionality to send broadcast messages over the IEEE802.15.4 protocol which is needed to synchronize the DAQ modules.

BROADCAST This method sends the command to initiate a broadcast message over the IEEE802.15.4 protocol to the XBee module. If the broadcast message couldn't be sent, an error message will be printed.

5.13 TIMER AND RELATIVE TIME - TIME

The class "Time" starts a timer with the desired time period and after a delay of 5.1ms a time point for the system start t_{start} is captured. t_{start} is required to measure relative time points for the GPS and CAN messages. A delay of 5.1ms is needed because the test in section 6.1.2 has shown this delay between the broadcast sending command in the DL software and the interrupt on the DAQ modules. The class "Time" also contains the methods to get and set the time period of the timer, to activate and deactivate the synchronization functionality, the timer interrupt service routine and the method which returns the relative time.

IRQTIMER After the timer time period expires, the method *irqTimer()* checks if the synchronization was activated by the class "DAQCtrl". If this is the case, the method *broadcast()* from the class "XBee" is called to send a broadcast message. Afterwards the synchronization is deactivated and the events "evDAQSynced" will be triggered.

SETTIMERPERIOD With this method the time period for the timer can be set newly.

GETTIMERPERIOD This method returns the actual time period of the timer.

SETSYNCHRONIZATIONFLAG That the method *irqTimer()* sends a broadcast by calling the method *broadcast()* from the class "XBee", to synchronize the DAQ modules, the synchronization has to be activated. This method *setSynchronizationFlag()* does this by setting the flag *synchronizationOn* "true".

CLEARSYNCHRONIZATIONFLAG This method sets the flag *synchronizationOn* "false", which deactivates the synchronization broadcast.

GETTIME The whole measurement system has its own relative time. The synchronization broadcasts over the IEEE802.15.4 protocol keep the DAQ modules in sync and a high resolution clock in the DL software provides an accurate relative system time. This method *getTime()* calculates and returns the time difference between the time point, when it is called t_{now}, and the system start time t_{start} in a resolution of 100µs. This difference is the actual relative time $t_{now\ relative}$.

$$t_{now\ relative} = t_{now} - t_{start} \tag{3}$$

REGISTERDAQCTRL Because the "Time" object is created before the objects of the class "DAQCtrl" exist, the association with the corresponding "DAQCtrl" object can't be done with the constructor and has to be done later. This method *registerDAQCtrl()* let the "DAQCtrl" objects building this association.

5.14 GPS

With the creation of an object of the class "GPS" a file descriptor for the GPS receiver and one for the file "GPS.csv" are created. In the file "GPS.csv" the position, together with the relative time $t_{now\ relative}$, will be saved.

The class "GPS" handles the connected GPS receiver and the file "GPS.csv". Next to this the class "GPS" provides the method *loggGPS()*, which reads, parses and saves the position received from the GPS receiver, to the file "GPS.csv".

LOGGGPS This method *loggGPS()* reads messages from the GPS receiver until the GPS receiver is not sending anymore, parses this messages with the "nmea library" and saves the position with the relative time from the method *getTime()* of the class "Time", to the file "GPS.csv".

Part IV

TEST

6 TEST

The tests of the data logger system are divided in three parts:

- The hardware dependent part.
- The non hardware dependent part.
- The long time test with the complete system.

6.1 HARDWARE DEPENDENT TESTS

In the hardware dependent test, the WLAN sending- and receiving functionality, the IEEE802.15.4 broadcast functionality and the GPS functionality was tested.

6.1.1 *WLAN sending- and receiving functionality test*

To test if the data logger receives all the packets, which are sent to it and if it also sends all the packets it should, the software Wireshark[1] was used to sniff the WLAN packets. An extract of the Wireshark logs is shown in figure 19.

Sending of the configuration, the configuration acknowledge, the reset of a DAQ clock and the start of a DAQ measurement.

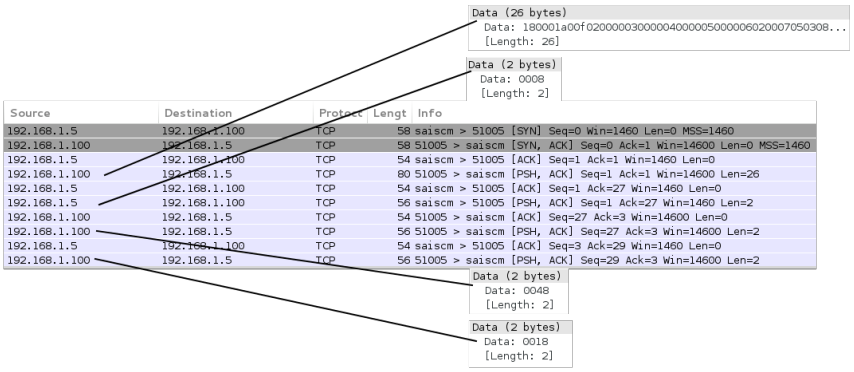


Figure 19: Wirehark logs

6.1.2 *IEEE802.15.4 broadcast functionality*

To test the broadcast functionality the IEEE802.15.4 interrupt on the DAQ PCB and the output of a GPIO output on the PandaBoard was measured. Figure 20 shows that the broadcasts occurs frequently every 250 ms.

1 The world's foremost network protocol analyzer www.wireshark.org

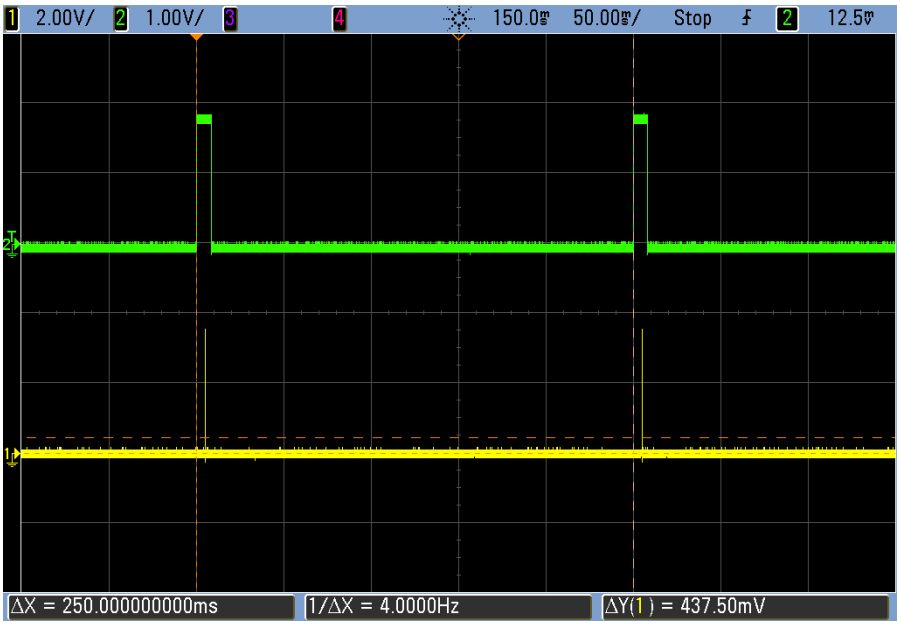


Figure 20: Time difference between the broadcasts

In figure 21 the measured time difference between the GPIO output on the PandaBoard and the IEEE802.15.4 interrupt on a DAQ is shown. This test has pointed out, that there is a delay of 5.1ms and a jitter of 100µs.

The output of the GPIO on the PandaBoard is green and the interrupt signal on a DAQ module is yellow.

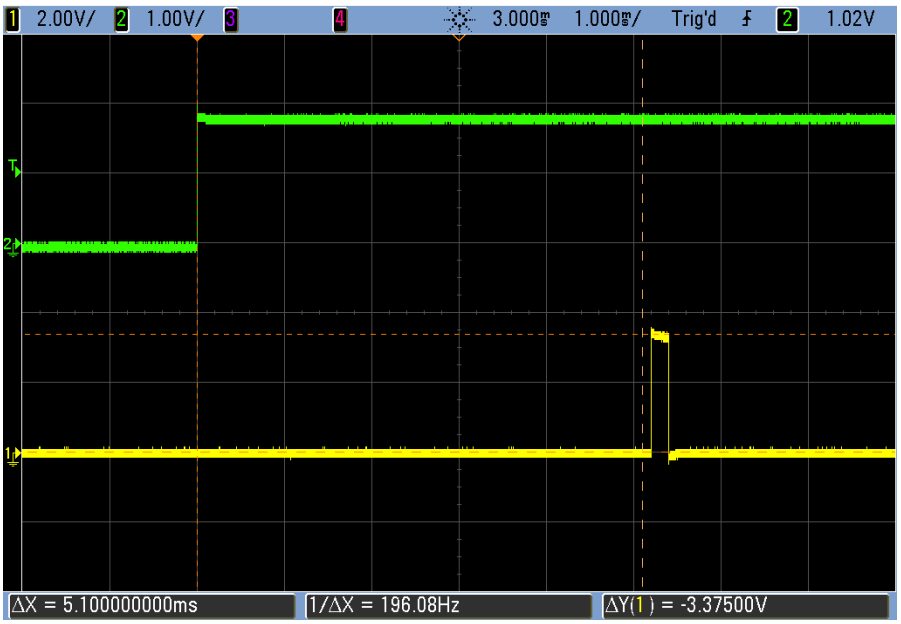


Figure 21: Delay between the GPIO output and the IEEE802.15.4 interrupt and its jitter

6.1.3 *GPS functionality*

The GPS receiver was tested outdoor under Linux because in the laboratory no GPS signal could be received. With "gpsd"[2] and "Xgps"[3] the GPS receiver was successfully tested on its functionality. The screen of of the tool "xgps" is shown in figure 22.

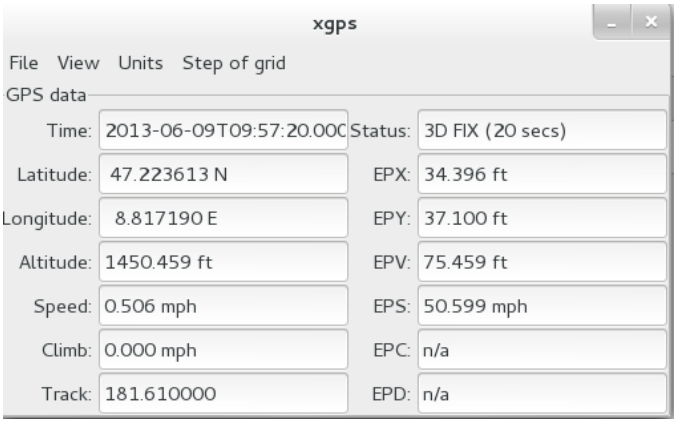


Figure 22: xgps screen

6.2 NON HARDWARE DEPENDENT TESTS

Except the test for the saving data functionality, the non hardware dependent tests were made with googletest[4]. The following software components were tested with googletest:

- DLCtrl: FSM
- DAQCtrl: FSM
- WLANCtrl: FSM
- ConfigCtrl: Loading of the configuration

The output of the googletest framework, which shows the successful tests is shown below:

```
[==========] Running 4 tests from 4 test cases.
[----------] Global test environment set-up.
[----------] 1 test from ConfigCtrlTest
[ RUN      ] ConfigCtrlTest.ReadConfig
[       OK ] ConfigCtrlTest.ReadConfig (0 ms)
[----------] 1 test from ConfigCtrlTest (0 ms total)

[----------] 1 test from DAQCtrlTest
[ RUN      ] DAQCtrlTest.FSM
[       OK ] DAQCtrlTest.FSM (0 ms)
```

2 A GPS service daemon: http://gpsd.berlios.de/
3 The xgps client: http://gpsd.berlios.de/xgps-sample.html
4 Google C++ Testing Framework https://code.google.com/p/googletest/

```
[----------] 1 test from DAQCtrlTest (0 ms total)

[----------] 1 test from DLCtrlTest
[ RUN      ] DLCtrlTest.FSM
[       OK ] DLCtrlTest.FSM (0 ms)
[----------] 1 test from DLCtrlTest (0 ms total)

[----------] 1 test from WLANCtrlTest
[ RUN      ] WLANCtrlTest.FSM
[       OK ] WLANCtrlTest.FSM (0 ms)
[----------] 1 test from WLANCtrlTest (0 ms total)

[----------] Global test environment tear-down
[==========] 4 tests from 4 test cases ran. (1 ms total)
[  PASSED  ] 4 tests.
```

To test the correct saving of the data to files, two test cases were done:

- Creating new files, if there isn't any file.
- Adding new data to existing files, if there are already files.

For both cases, the result was checked manually and has shown, that the tests were successful.

6.3 LONG TIME TEST

Long time tests has shown, that the WLAN modules used by the DAQ modules are quite instable and crashes often, specially with high sample rates. This fact made the accomplishment of seriously long time test impossible.

Nevertheless, a long time test has shown, that with low sample rates, data was correctly received and saved over a period of 2 hours.

7 CONCLUSION

7.1 RESULT

The PandaBoard as the chosen hardware platform with the connected modules satisfy all the must interface criteria and nice-to-have interface criteria, defined by the specification. Ubuntu Core, which is used as OS provides the required software packages with a small memory footprint compared to a normal Ubuntu version. With the packages "hostapd" and "udhcpd" the PandaBoard provides AP functionality.

The tests has shown, that the implemented software parts as well as the whole software run correctly.

The configuration file "DAQ.cfg" is loaded without problems and its settings are saved to the array of configuration structures *myDAQConfig*.

Each configuration is sent to its corresponding DAQ module the right way and the acknowledgment is received from all the five DAQ modules.

The clocks of the DAQ modules are set to zero and the measurement on the DAQ modules can be started from the DL. A broadcast is sent every 250ms to keep the DAQ modules synced.

If the measurement on a DAQ module is started, the DL receives the data of all the sensors and saves them in the file "DAQ<1..5>.csv" in a defined format without data loss.

With the "PCAN-USB" a CAN interface is evaluated, with which the DL can be connected with a CAN bus.

The messages received from the GPS receiver are parsed and the positions together with the actual relative time $t_{now\ relative}$ are saved to the file "GPS.csv".

7.2 CHALLENGES

The WLAN modules on the DAQ modules are quite instable, this fact made it sometimes hard to check if the DL software works the right way. Also a seriously long time test couldn't be done because of this problem.

An other issue was the synchronization over the ZigBee protocol. The radio module on the DAQ modules has no protocol stack (IEEE802.15.4 or ZigBee) implemented. This made it hard, to figure out, why the radio modules just seldom reacted on broadcasts sent from the ZigBee XBee module. As it turned out, a IEEE802.15.4 XBee

module with the channel fixed to 0x0F has to be used instead of the ZigBee XBee module, that the radio modules receive the broadcast messages.

With the first implementation of the methods *wlanReceive()* from the class "WLAN" and *SaveAnalogData()*, *SaveSTData()* from the class "SaveData" data loss occurred. The problem was, that receiving the data from the network socket and saving the data to a file in one step took too long and so other data which was received couldn't be processed anymore. The solution was to implement a circular buffer, in which the method *wlanReceive()* saves the received data temporary and from which the method *processData()* from the class "SaveData" reads it and afterwards saves it to a file, when enough resources are available. With this separation of the data handling the peaks of the system utilization could be eliminated and data loss occurs not any longer.

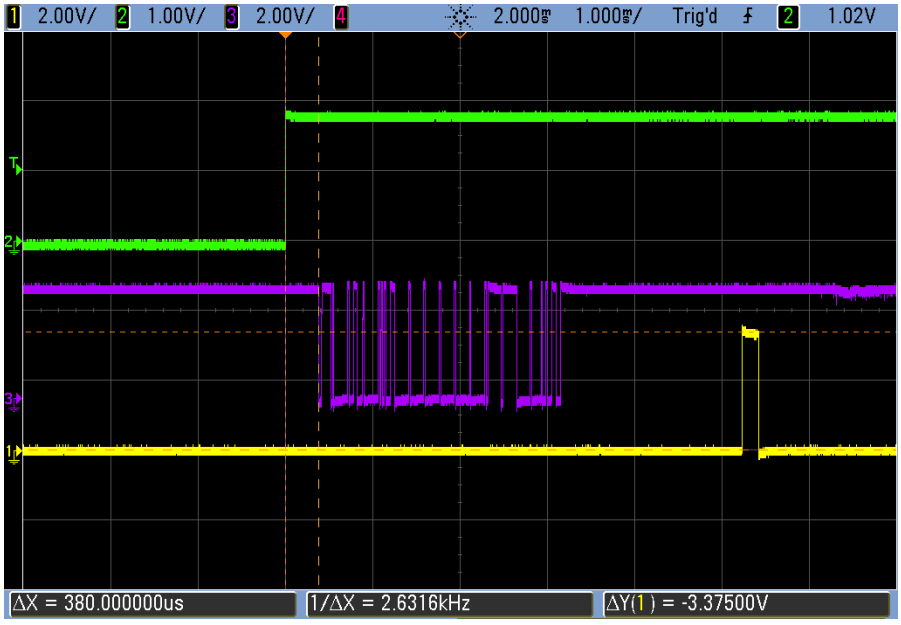
8 PROSPECTS

8.1 COVER

During this thesis the PandaBoard was used without a cover. That the DL can be installed and used in a car, a suitable cover has to be found.

8.2 SYNCHRONIZATION

To decrease the delay between the IEEE802.15.4 broadcasts and the interrupts on the DAQ modules and to reduce the jitter, another IEEE802.15.4 radio module with a faster interface has to be considered. The broadcast test has pointed out, that the delay between the broadcast command in the software and the command transmission start on the UART interface is 380µs. This delay is stable and small compared to the delay between the receiving of the command by the XBee module and the interrupt on the DAQ modules. The described delays are shown in figure 23.



Green is the GPIO signal, purple the UART signal and yellow the interrupt signal on a DAQ module.

Figure 23: Delay between the broadcast command and the interrupt

8.3 CAN

Due to time constraints, the cost saving "CAN SPI click 5V" CAN interface couldn't put into operation. Also a suitable CAN test environ-

ment couldn't be found, so that the implementation of the CAN software part isn't realized yet. With the "PCAN-USB" a CAN interface solution is evaluated, which can be used to connect the PandaBoard to the CAN bus.

8.4 LIN

As described in [20], a Local Interconnect Network (LIN) interface could be realized over an additional UART interface. Because all the UART interfaces on the PandaBoard are already used an USB to UART is needed.

8.5 REMOTE ACCESS

With a web server on the PandaBoard online access to the collected data could be provided. Over this also the configuration of DAQ modules could be changed. This way the SD card wouldn't need to be removed from the PandaBoard anymore to change the settings of the DAQ modules or to access the data. Maybe the received data could also be monitored live.

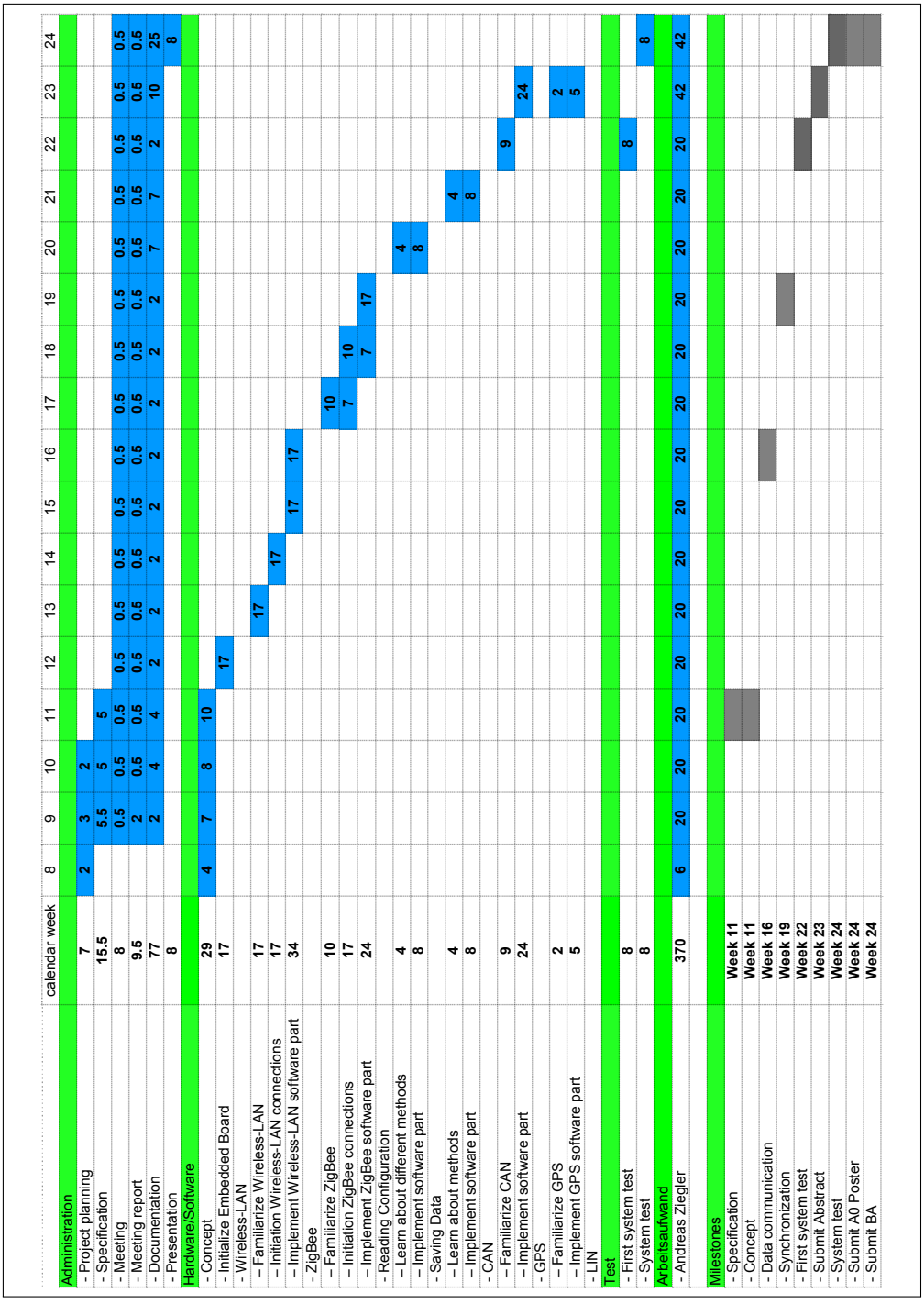
Part V

APPENDIX

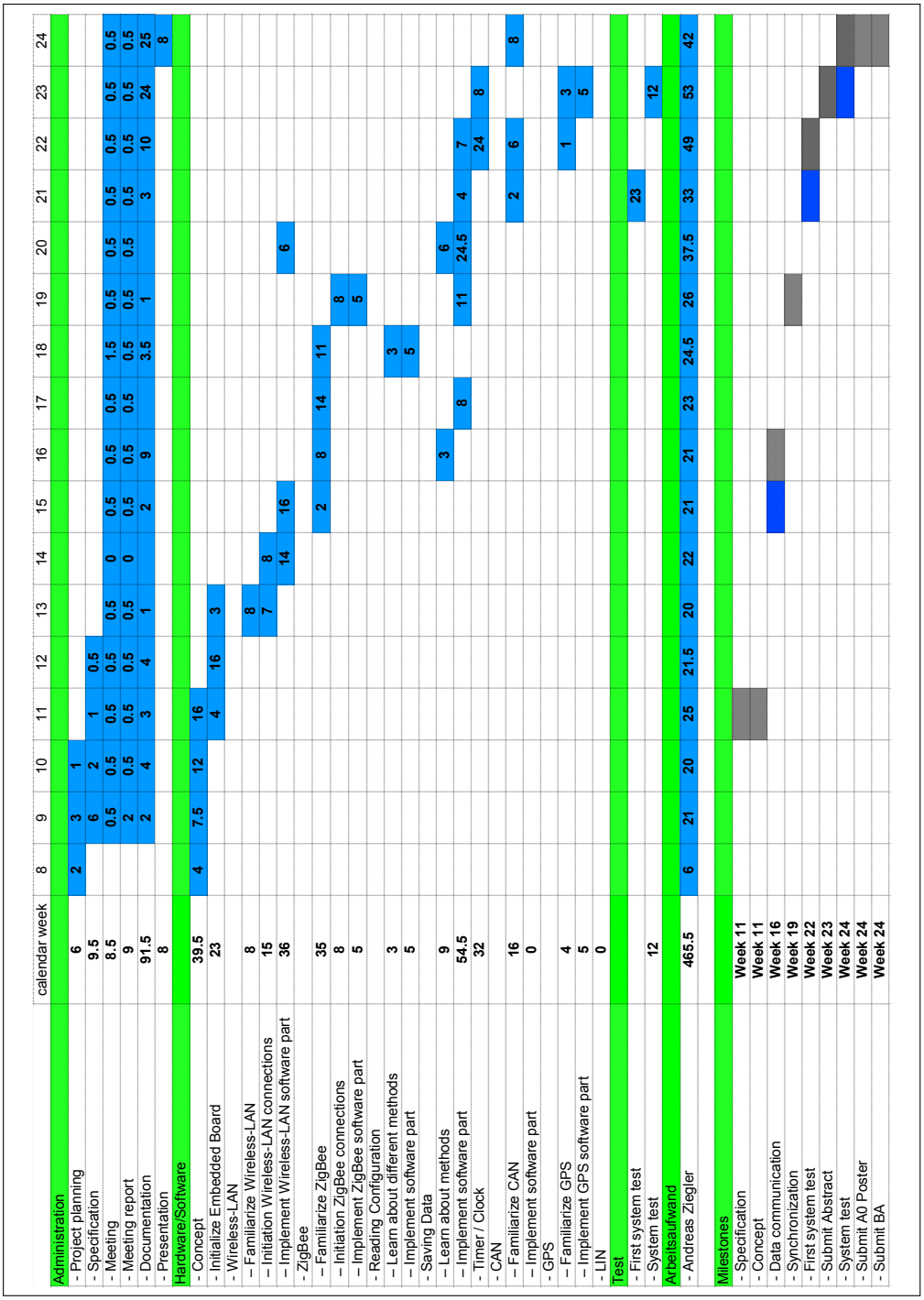
A

TIMETABLES

A.1 TIMETABLE (SHOULD)



A.2 TIMETABLE (IS)



A.3 COMPARISON

The WLAN part of the software was earlier finished then expected. Problems between the XBee and the radio module on the DAQ modules delayed the initialization of the first IEEE802.15.4 connection. The first system test could be done one week before but has shown diffi-

culties with the timing of the data saving part. The realization with a circular receive buffer solved this issue but delayed this part. During the thesis, the part Timer/Clock was added to the timetable. Problems with the real-time operation under Linux for the Timer/Clock, retarded the CAN and GPS parts. Because of the absentee of a test environment for the CAN part, the implementation of the GPS part was chosen before the CAN part.

B

SPECIFICATIONS

PFLICHTENHEFT

Mobiler Datenlogger zur Aufzeichnung dezentral erfasster dynamischer Motorfahrzeugdaten

Autor: Andreas Ziegler

Betreuer: Gian Danuser
Examinator: Prof. Reto Bonderer

Aufgabenstellung: BA-ESW13.01
Themengebiet: Elektrotechnik, Embedded Software Engineering

Version: 0.4
19. März 2013

HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL
FHO Fachhochschule Ostschweiz

Inhaltsverzeichnis

Änderungsindex

Version	Änderungen	Name	Datum
0.4	Sportec bei den Punkten 4 und 5 entfernt	AZ	19.03.2013
0.3	Aufzeichnung starten/stoppen, Speicherort der Daten	AZ	14.03.2013
0.2	LIN-Bus, Mess-Setup, Zielgruppe, Betriebsbedingungen, Ethernet-Schnittstelle	AZ	06.03.2013
0.1	Erste Verion	AZ	28.02.2013

1 Zielbestimmung

Das Ziel dieser Bachelor-Arbeit (BA) ist die Entwicklung eines mobilen Datenlogger zur Aufzeichnung dezentral erfasster dynamischer Motorfahrzeugdaten.

1.1 Musskriterien

- Der Datenlogger muss über die in Abschnitt 3.1.1 beschriebenen Schnittstellen verfügen.
- Der Datenlogger konfiguriert die DAQ-Module gemäss einer Konfigurationsdatei, welche sich auf der (μ)SD-Karte befindet.
- Der Datenlogger übernimmt die Synchronisation der 5 DAQ-Modulen über ZigBee.
- Das Aufzeichnen der Daten kann mit einem Taster gestartet und gestoppt werden.
- Der Datenlogger muss alle Daten, welche von den DAQ-Modulen gesendet und vom Datenlogger empfangen werden auf der SD-Karte aufzeichnen.

1.2 Kannkriterien

- Der Datenlogger muss über die in Abschnitt 3.1.2 beschriebenen Schnittstellen verfügen.
- Der Datenlogger loggt die Meldungen auf dem autointernen CAN-Buss.
- Die Position wird mit der Hilfe eines GPS-Modules aufgezeichnet.

1.3 Wunschkriterien

Die Umsetzung der folgenden Punkte wird während der Arbeit, je nach vorhandenen Ressourcen entschieden.

- Der Datenlogger muss über die in Abschnitt 3.1.3 beschriebenen Schnittstellen verfügen.
- Der Datenlogger loggt die Meldungen auf dem autointernen LIN-Bus.
- Onlinezugriff und Konfiguration der Messung über USB oder Ethernet-Anschluss.
- Anschlussmöglichkeit für Präsentationsgeräte (PC, Pad, etc.).

1.4 Abgrenzungskriterien

Die hier aufgeführten Kriterien werden in der BA nicht bearbeitet:

- Visualisierung und Auswertung der Rohdaten.
- Das Produkt muss nicht serientauglich sein (Es ist ein Prototyp).
- Die Befestigung des Datenloggers im PKW wird zu einem späteren Zeitpunkt entwickelt.

2 Einsatz

2.1 Anwendungsbereiche

Der Datenlogger soll in PKWs der Firma Sportec eingebaut werden, um zusammen mit 5 DAQ-Modulen, die Fahrdynamik auf der Rennstrecke zu analysieren und zu optimieren.

2.2 Zielgruppen

Die Zielgruppe des Produktes sind Personen, welche Anwendungsprogramme für dieses System entwickeln.

2.3 Betriebsbedingungen

Die physikalische Umgebung weist Störfaktoren wie Vibrationen und EMV-Störungen nach Kfz-EMV Richtlinien auf.

3 Umgebung

3.1 Schnittstellen

3.1.1 Muss-Schnittstellen

Stromversorgung

- Spannungseingang: 8V bis 24 V DC

Kommunikation

- 1 x Wireless-Lan g (802.11g) mit Accesspoint-Funktionalität, 5 TCP-Sockets mit 1.2 Mbps pro Socket
- 1 x ZigBee
- 1 x USB
- 1 x Ethernet
- 1 x SD-Karte

3.1.2 Kann-Schnittstellen

Kommunikation

- 1 x CAN

Positionsbestimmung

- 1 x GPS

3.1.3 Wunsch-Schnittstellen

Kommunikation

- 1 x LIN

4 Beteiligte Personen

Name	**Funktion**	**Zugehörigkeit**	**eMail-Adresse**
Prof. Reto Bonderer	Auftraggeber	HSR	reto.bonderer@hsr.ch
Gian Danuser	Betreuer	HSR	gian.danuser@hsr.ch
Andreas Ziegler	Auftragsnehmer	Student HSR	andreas.ziegler@hsr.ch

5 Freigabe

Auftraggeber: Prof. Reto Bonderer, HSR

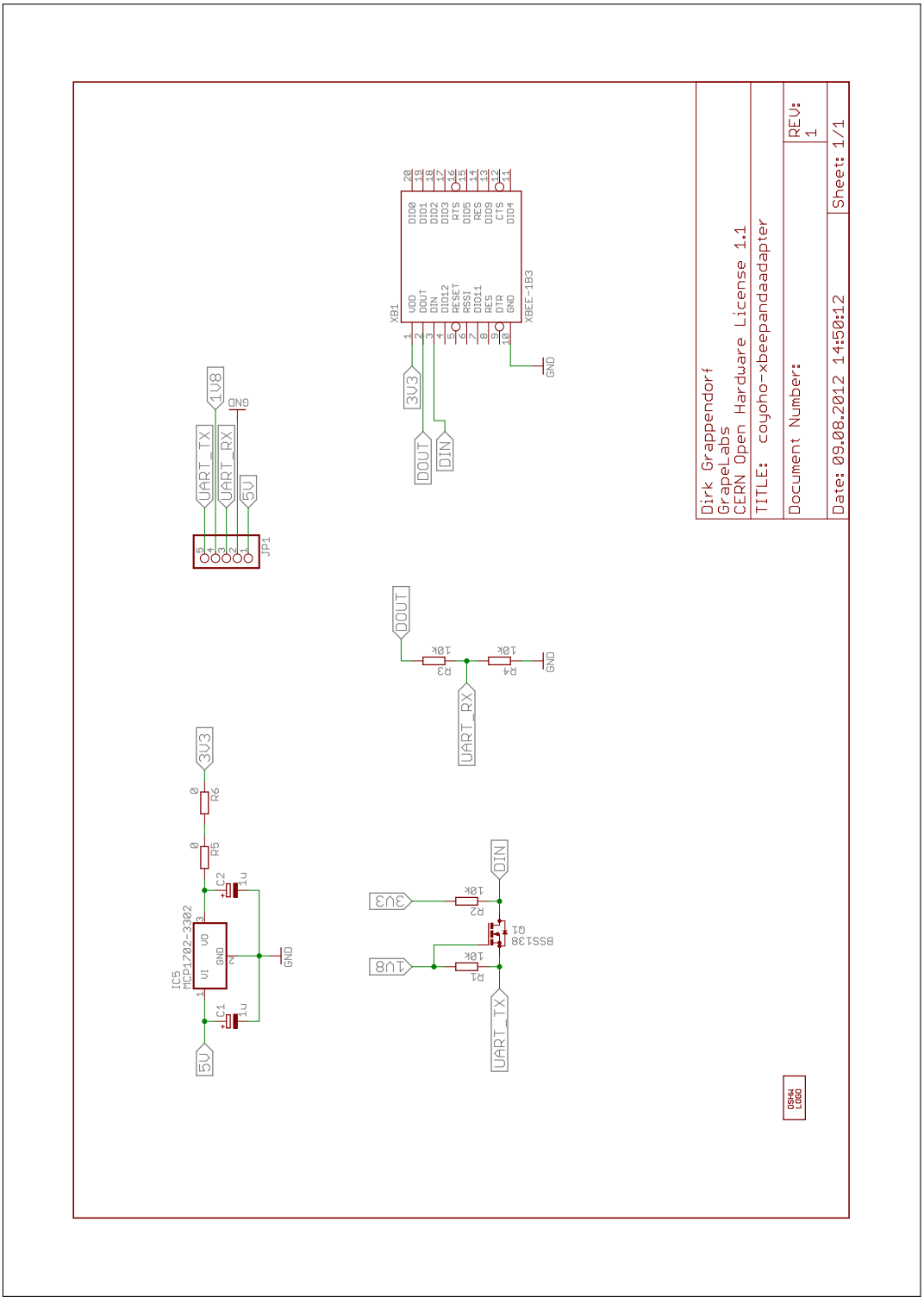
Rapperswil, 25.3.2013 .....
Ort, Datum Unterschrift

Auftragnehmer: Andreas Ziegler, Student HSR

Rapperswil, 25.03.2013 .....
Ort, Datum Unterschrift

C

VOLTAGE LEVEL SHIFTER



D

CONFIGURATION FILE

D.1 FORMAT

The configuration file has to have the name “DAQ.cfg”. The file structure of the configuration file is shown below:

```
// Basic store information:
name = "DAQ configuration";

daq1 =
{
 analogSensors = ( { id = 1;
                     sampleRate = 10000; },
                   { id = 2;
                     sampleRate = 9000; },
                   { id = 3;
                     sampleRate = 900; },
                   { id = 4;
                     sampleRate = 90 },
                   { id = 5;
                     sampleRate = 8; },
                   { id = 6;
                     sampleRate = 2; }
                 );
 stSensors = ( { id = 7;
                 highVoltage = 5;
                 lowVoltage = 3; },
               { id = 8;
                 highVoltage = 5;
                 lowVoltage = 3; }
             );
};

daq2 =
{
 analogSensors = ( { id = 1;
                     sampleRate = 10000; },
                   { id = 2;
                     sampleRate = 9000; },
                   { id = 3;
                     sampleRate = 900; },
                   { id = 4;
                     sampleRate = 90 },
                   { id = 5;
                     sampleRate = 8; },
                   { id = 6;
                     sampleRate = 2; }
                 );
 stSensors = ( { id = 7;
                 highVoltage = 5;
                 lowVoltage = 3; },
               { id = 8;
                 highVoltage = 5;
                 lowVoltage = 3; }
             );
};

daq3 =
```

```
{
 analogSensors = ( { id = 1;
                     sampleRate = 10000; },
                   { id = 2;
                     sampleRate = 9000; },
                   { id = 3;
                     sampleRate = 900; },
                   { id = 4;
                     sampleRate = 90 },
                   { id = 5;
                     sampleRate = 8; },
                   { id = 6;
                     sampleRate = 2; }
                 );
 stSensors = ( { id = 7;
                 highVoltage = 5;
                 lowVoltage = 3; },
               { id = 8;
                 highVoltage = 5;
                 lowVoltage = 3; }
             );
};

daq4 =
{
 analogSensors = ( { id = 1;
                     sampleRate = 10000; },
                   { id = 2;
                     sampleRate = 9000; },
                   { id = 3;
                     sampleRate = 900; },
                   { id = 4;
                     sampleRate = 90 },
                   { id = 5;
                     sampleRate = 8; },
                   { id = 6;
                     sampleRate = 2; }
                 );
 stSensors = ( { id = 7;
                 highVoltage = 5;
                 lowVoltage = 3; },
               { id = 8;
                 highVoltage = 5;
                 lowVoltage = 3; }
             );
};
%
daq5 =
{
 analogSensors = ( { id = 1;
                     sampleRate = 10000; },
                   { id = 2;
                     sampleRate = 9000; },
                   { id = 3;
                     sampleRate = 900; },
                   { id = 4;
                     sampleRate = 90 },
                   { id = 5;
                     sampleRate = 8; },
                   { id = 6;
                     sampleRate = 2; }
                 );
 stSensors = ( { id = 7;
                 highVoltage = 5;
                 lowVoltage = 3; },
               { id = 8;
```

```
119                 highVoltage = 5;
120                 lowVoltage = 3; }
121             );
122 };
```

Every DAQ module has six analog- and two Schmitt trigger sensors. To configure an analog sensor, the sample rate has to be defined. The correct configuration for a Schmitt trigger sensor contains the two threshold voltages, “highVoltage” and “lowVoltage”.

To deactivate an analog sensor there are two ways: The first possibility is to set the id to “0”, the other possibility is, to set the sample rate to “0”. To deactivate a Schmitt trigger sensor, the id has to be set to “0”.

BIBLIOGRAPHY

[1] CDC ACM, March 2010. URL http://wiki.openmoko.org/wiki/CDC_ACM.

[2] SD Configuration, July 2010. URL http://omappedia.org/wiki/SD_Configuration.

[3] How to configure and us CAN bus, January 2011. URL https://developer.ridgerun.com/wiki/index.php/How_to_configure_and_use_CAN_bus.

[4] How to use GPIO signals, October 2012. URL https://developer.ridgerun.com/wiki/index.php/How_to_use_GPIO_signals.

[5] Panda How to kernel 3 5 rcx, July 2012. URL http://elinux.org/Panda_How_to_kernel_3_5_rcx.

[6] Mac80211 based open source architecture, September 2012. URL http://www.omappedia.com/wiki/Mac80211_based_open_source_architecture#Testing_WLAN_in_SoftAP_mode.

[7] Cross Compile Linux Kernel for Pandaboard (armel and armhf), February 2013. URL http://tkcheng.wordpress.com/2013/02/23/cross-compile-linux-kernel-for-pandaboard-armel-and-armhf/.

[8] NMEA Library, 2013. URL http://nmea.sourceforge.net/.

[9] OMAP Ubuntu Core, February 2013. URL http://www.omappedia.com/wiki/OMAP_Ubuntu_Core.

[10] Andrew. 5V-3.3V bidirectional level converter, April 2009. URL http://www.rocketnumbernine.com/2009/04/10/5v-33v-bidirectional-level-converter.

[11] attie.co.uk. libxbee. URL http://attie.co.uk/libxbee.

[12] Prof. Dr. Heide Balzert. *Lehrbuch der Objektmodellierung*. Elsevier GmbH, 2005.

[13] Murat Belge. Gpio 113 default pinmux should be gpio input (Mode 3), April 2012. URL https://bugs.launchpad.net/linaro-ubuntu/+bug/988497.

[14] Gian Danuser and Sven Eicher. Mobiles Datenerfassungsmodul zur Messung der Fahrdynamik von Motorfahrzeugen. Bachelor thesis, Hochschule fuer Technik Rapperswil, June 2012.

[15] Dirk Grappendorf. XBee adapter board. URL https://sites.google.com/site/grappendorfnet/projects/coyoho/coyoho-server-hardware.

[16] Jeff. Socket with recv-timeout: What is wrong with this code?, December 2008. URL http://stackoverflow.com/questions/393276/socket-with-recv-timeout-what-is-wrong-with-this-code.

[17] kristoff. Using the RS232 DB9-connector on the pandaboard for GPIO, November 2012. URL http://villazeebries.krbonne.net/hamstuff/?p=270.

[18] Mark Lindner. libconfig - C/C++ Configuration File Library, 2013. URL http://www.hyperrealm.com/libconfig/.

[19] nims11. Hostapd: The Linux Way to create Virtual Wifi Access Point, April 2012. URL http://nims11.wordpress.com/2012/04/27/hostapd-the-linux-way-to-create-virtual-wifi-access-point/.

[20] Pavel Pisa, Rotislav Losovy, Oliver Hartkopp, and Michal Sojka. Uart-based Lin-bus Support for Linux with Socketcan Interface. *OSADM*, 14:8, 2012.

[21] W. Richard Stevens, Bill Fenner, and Andrew M. Rudoff. *UNIX Network Programming*. Addison-Wesley Professional Computing Series, 2008.

[22] Bjarne Stroustrup. *The C++ Programming Language*. Addison-Wesley, 2013.

[23] Anthony Williams. *C++ Concurrency in Action*. Manning Publications Co., 2012.

[24] Juergen Wolf. *Linux-UNIX-Programmierung*. Galileo Computing, 2006.

COLOPHON

This document was typeset using the typographical look-and-feel `classicthesis` developed by André Miede. The style was inspired by Robert Bringhurst's seminal book on typography "*The Elements of Typographic Style*". `classicthesis` is available for both LaTeX and LyX:

http://code.google.com/p/classicthesis/

DECLARATION

I declare that I have prepared the bachelor thesis "Mobile datalogger for recording decentrally captured dynamic motor vehicle data" without illegal help. I also declare that contributions by other authors which are used in the thesis or led to the ideas behind the thesis are properly referenced in written form. I am aware that a bachelor thesis , developed under guidance, is part of the examination and may not be commercially used or transferred to a third party without written permission from my supervisory professor.

Rapperswil, June 2013

Andreas Ziegler