



---

Berna

---



**BFS**  
Bern Formula Student

---

# Telemetrie für Projekt Bern Formula Students

Semesterarbeit im Rahmen des Moduls Projekt 2

Studiengang: Bachelor of Science Informatik

Autoren: Pascal Bohni, Roger Jaggi

Betreuer: Dr. Andreas Danuser

Datum: 16.06.2015

Das Bild von der Titelseite entstammt der Website der Bern Formula Student, erreichbar unter <http://www.bernformulastudent.ch>.

## Versionen

Version	Datum	Status	Bemerkungen
0.1	20.03.2015	Entwurf	Dokument erstellt
0.2	21.05.2015	Entwurf	Konzepte dokumentiert
0.3	12.06.2015	Entwurf	I2C Bridge, IoT Gateway
0.4	14.06.2015	Entwurf	Anhänge eingefügt
1.0	16.06.2015	Definitiv	Definitive Version

# Management Summary

Im Rahmen des Modules Projekt 2 wurde durch Roger Jaggi und Pascal Bohni unter der Leitung von Andreas Danuser ein Telemetrie-Konzept für die Bern Formula Student erarbeitet. Bern Formula Student (kurz BFS) ist ein Studentenprojekt, um mit einem eigens konstruierten Elektro-Rennauto an den bekannten Formula Student Rennen teilzunehmen.

Während des Projektes wurden folgende Telemetrie-Komponenten erarbeitet:

- Prototyp I2C-Bridge für SMING, benötigt für den kontaktlosen Temperatursensor
- Prototyp CAN-to-MQTT Bridge, benötigt zum Übermitteln der Daten der drei CAN-Busse des Fahrzeuges über das MQTT-Protokoll.
- Prototyp IoT Gateway, benötigt zum Übermitteln der MQTT-Daten in die Cloud

Jeder Prototyp ist soweit lauffähig, so dass bereits erste Funktionsdemos gemacht werden konnten.

Die Weiterführung der Arbeit wird Anfangs Juli stattfinden, wenn zeitgleich seitens des BFS-Teams das Rennauto zusammengebaut wird. Für eine komplette Telemetrie wird es nötig sein, alle Komponenten in das Auto einzubauen, einen weiteren Funktionstest zu machen sowie die Anzeige für die Monitore in der Boxengasse zu gestalten.

# Inhaltsverzeichnis

<b>Management Summary</b>	i
<b>1. Aufgabenstellung</b>	1
<b>2. Aufbau</b>	2
<b>3. IoT Gateway</b>	3
3.1. Netmodule NetBox NB1600 . . . . .	4
3.2. CAN-Cape . . . . .	4
3.3. BLED112 . . . . .	5
<b>4. SMING</b>	6
4.1. BLE-I2C Bridge . . . . .	7
4.2. Implementation . . . . .	8
<b>Selbstständigkeitserklärung</b>	9
<b>Literaturverzeichnis</b>	10
<b>Abbildungsverzeichnis</b>	11
<b>A. BeagleBone Hardware Spezifikation</b>	12
<b>B. Setup Beaglebone Black mit CAN Cape</b>	13
B.1. Install latest Kernel image . . . . .	13
B.2. Add default route temporary until next restart . . . . .	13
B.3. Add default route permanent . . . . .	13
B.4. Configure NAT on Ubuntu Host PC . . . . .	14
B.5. Install TowerTech TT3201 rev 5 on BeagleBone Debian . . . . .	15
<b>C. Beispiel Bluegiga Anbindung</b>	17
<b>D. CAN-Bus-Anbindung</b>	18
D.1. CAN-Tutorial . . . . .	18
D.2. CAN Hardware . . . . .	18
D.3. CAN Simulations-Software . . . . .	18
<b>E. Liste der genutzen Attribut UUIDs des SMING</b>	19
<b>F. Zeitplan</b>	20

# 1. Aufgabenstellung

Von Seiten Bern Formula Student wurde folgende Zielvorstellung an die Telemetrie formuliert:

*Grundsätzlich sollen möglichst alle Daten übermittelt werden, welche auf den drei CAN-Bus-Systemen des Fahrzeugs übertragen werden.*

Zusätzlich wurden weitere Messpunkte definiert, welche über die Smings übermittelt werden sollen.

Folgende Messpunkte sollen übermittelt werden:

- **Eingangsgrößen Fahrer**

- Fahrpedalstellung *CAN 1*
- Bremspedalstellung *CAN 1*
- Bremsdruck *CAN 1*
- Lenkwinkel *CAN 1*

- **Motor und Kühlsystem**

- Soll-Moment *CAN 2/3*
- Ist-Moment *CAN 2/3*
- Soll-Drehzahl *CAN 2/3*
- Ist-Drehzahl *CAN 2/3*
- Motortemperatur *CAN 2/3*
- Kühlmitteltemperatur *CAN 1*

- **Batterie**

- HV-Spannung *CAN 1*
- HV-Strom *CAN 1*
- HV-Batterie Ladestand *CAN 1*
- GLV Batterie Spannung *Wireless über Sming*

- **Fahrzeug**

- Geschwindigkeit *CAN 1*
- Beschleunigung *CAN 1*
- Drehrate *CAN 1*
- Reifentemperatur *Wireless über Sming*
- Bremstemperatur *Wireless über Sming*
- Bodentemperatur *Wireless über Sming*
- Lufttemperatur bei Kühlereintritt *Wireless über Sming*
- Lufttemperatur bei Kühleraustritt *Wireless über Sming*
- Dämpferweg (optional) *Wireless über Sming*

Die Anbindung "Wireless über Sming" bind die Messpunkte bei welchen von unserer Seite Sensoren evaluiert werden welche ans Sming angeschlossen werden können.

Erweitert wurde die Aufgabenstellung von Andreas Danuser durch die Konstruktionsvorgabe, dass die einzelnen Komponenten der Telemetrie möglichst universell, sprich als eigenständiges Modul auch in einem komplett anderen Einsatzzweck verwendbar sein sollen.

## 2. Aufbau

Anhand der Aufgabenstellung wurde eine Architektur entworfen, die aus mehreren Modulen besteht. Die einzelnen Module werden in den nächsten Kapiteln beschrieben. Die Abbildung 2.1 zeigt die gewählte Architektur.

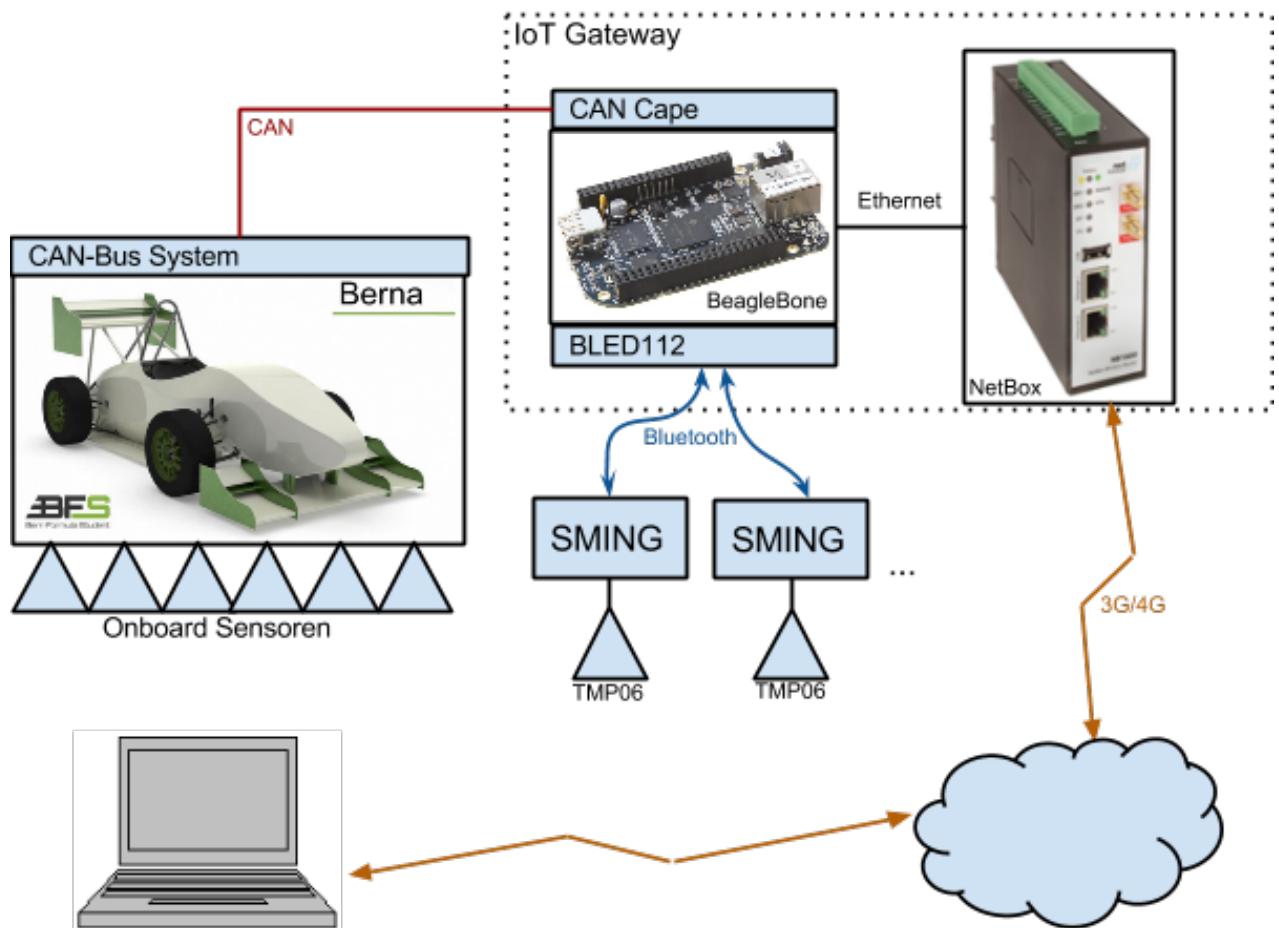


Abbildung 2.1.: Gesamtarchitektur Telemetrie

### 3. IoT Gateway

Das IoT Gateway dient dazu einzelne Messpunkte zusammen zufassen und die Daten danach ins Internet zu übertragen. Als Gateway Hardware nutzen wir die bekannte Embedded Platform Beagle Bone Black. Dies ermöglicht uns auch vorhandene Erweiterungen zu nutzen. Das Gateway arbeitet mit zwei verschiedenen Methoden um die einzelnen Messpunkte abfragen zu können. Einerseits dem CAN-Bus, welcher in der Automobiltechnik sehr verbreitet ist und anderseits eine Bluetooth Smart Schnittstelle um die Sming Messknoten ansprechen zu können. Die Übertragung ins Internet wird mithilfe eines UMTS-Routers gemacht.

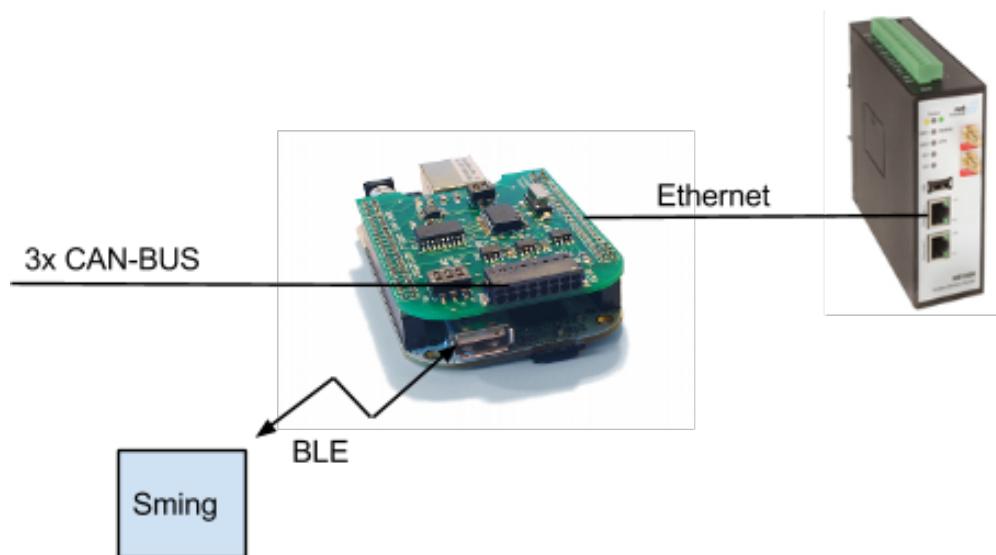


Abbildung 3.1.: Aufbau des IoT Gateways

### 3.1. Netmodule NetBox NB1600

Als UMTS-Router setzen wir auf den Netmodule Router NB1600. Dieser kam schon in unserem Schwerpunkt Mobile Computing beim Semesterprojekt SSmojeßum Einsatz. Für unseren Einsatzzweck als reine Bridge zwischen Ethernet und UMTS-Netz wird er nicht ausgelastet. Der NB1600 könnte nähmlich auch noch GPS empfangen und als Wlan-Hotspot dienen. Daher besteht hier noch eine Optimierungsmöglichkeit durch den Einsatz eines geeigneten USB Dongles der direkt vom Beagle Bone aus auf UMTS zugreifen kann. Dieser zu evaluieren hätte aber den Zeitrahmen für das Projekt gesprengt.



Abbildung 3.2.: Foto NetModule NB1600

### 3.2. CAN-Cape

Der CAN-Bus wird im Bern Formula Student Projekt genutzt um die Steuerbefehle innerhalb des Autos zu übertragen. Es werden drei separate Verdrathungen geführt. Wir empfangen die Daten der 3 CAN-Busse über das Beagle Bone Black CAN Cape von Tower Tech. Da die ganze Steuerung des Autos über die CAN-Busse läuft haben wir nur lesenden Zugriff. Die Installationsanleitung zum Cape befindet sich im Anhang.

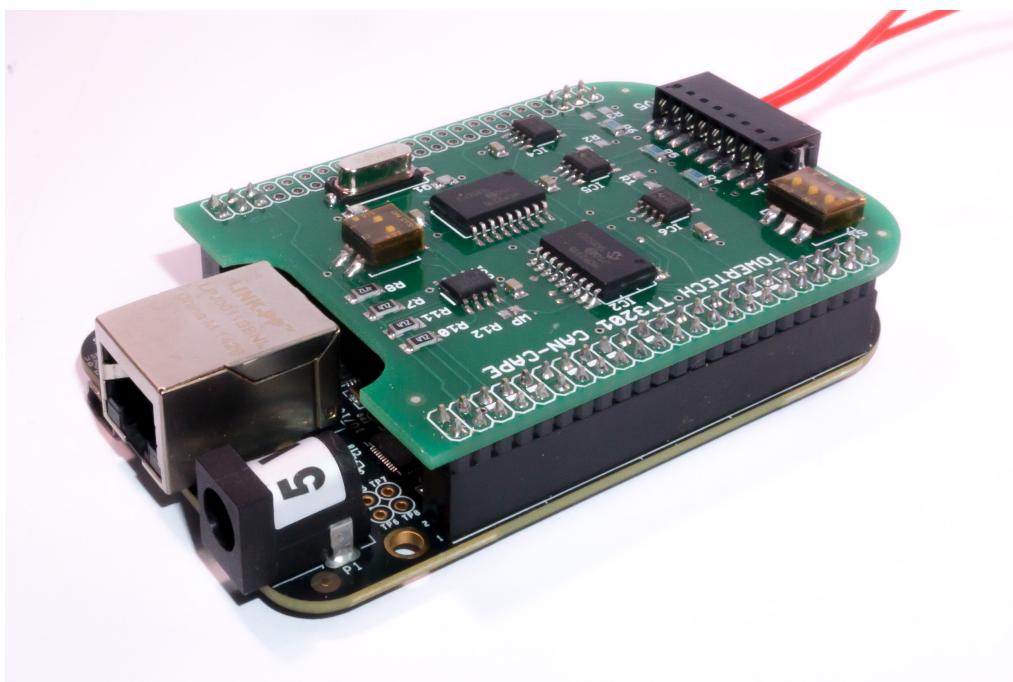


Abbildung 3.3.: Foto CanCape auf Beagle Bone Black

### 3.3. BLED112

Die Schnittstelle zu den Sensoren an den SMINGS ist der USB Dongle BLED112 von Bluegiga. Dieser Mikrocontroller integriert den ganzen Bluetooth Smart Stack, der dank einer API über eine serielle Schnittstelle angesprochen werden kann. Damit ist es nicht nur möglich, die Bluetooth Advertisements der SMINGS zu empfangen, es kann auch mit bis 3<sup>1</sup> Bluetooth 4.0 Geräten verbunden und kommuniziert werden.

Der Bluegiga Chip ist nebst dem auch ein Konkurrenzprodukt zum nRF51 Chip von Nordic, der auf dem Sming zum Einsatz kommt. Wie auch der nRF51 ist der Bluegiga programmierbar, wodurch auch mit diesem autonome smarte Elektronikprojekte realisiert werden können.

Im Anhang findet sich eine Beispiel-Befehlsabfolge, wie sie aktuell zum Verbinden mit dem Smings zum Einsatz kommt. Weitere Informationen wie API Dokumentation, Windows Treiber<sup>2</sup>, etc. finden sich auf folgender Seite: <https://www.bluegiga.com/en-US/products/bled112-bluetooth-smart-dongle/#documentation>



Abbildung 3.4.: Foto BLED112

---

<sup>1</sup>Die maximale Beschränkung von 3 wurde aus Praxistests mit den Smings festgestellt  
<sup>2</sup>Bei Linux/Mac sind die Treiber bereits im Kernel vorhanden

## 4. SMING

Um weitere Sensoren einfach ohne Verdrahtung ins Messsystem aufzunehmen, setzen wir auf das Sming. Das Sming wurde durch Daniel Meer in seiner Master Thesis entwickelt. Er beschreibt in der Thesis das Sming wie folgt: *Der TXW51 ist ein kleiner und energiesparender Sensorknoten, der als Basis für zukünftige Projekte verwendet werden kann. Er kommuniziert über Bluetooth Smart und enthält einen Sensor zur Messung der Beschleunigung. Die Firmware kann einfach für neue Anforderungen modifiziert werden.*[1]

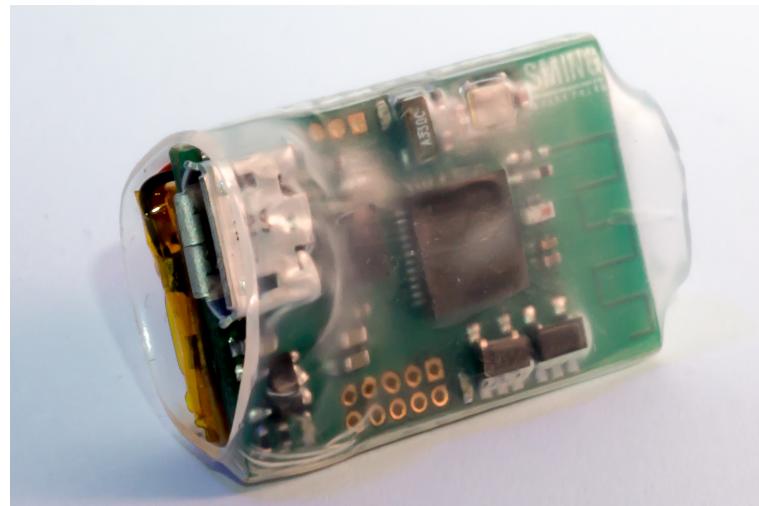


Abbildung 4.1.: Foto Sming mit Akku bestückt

Für unser Projekt ist vor allem die Erweiterbarkeit des Sming sehr interessant. Dies weil beliebige Sensoren über I2C angesprochen werden können.

Um die angepasste Firmware zu testen, nutzen wir das von Bluegiga erhältlich BLE-Gui. Dies ermöglichte uns die Arbeit aufzuteilen, so dass auch ohne programmieren des Gateways die Änderungen an der Firmware getestet werden konnten. Um die bestehende Software zu testen, war dies zum Teil aber nicht optimal, weil, um einen höheren Datendurchsatz zu erreichen, wurden die Messwerte des Beschleunigungssensor in einer Charakteristik zusammengefasst.

## 4.1. BLE-I2C Bridge

Das Sming verfügt über den Erweiterungsheader die Möglichkeit I2C Sensoren anzusprechen. Um diese Funktion zu nutzen, muss die Firmware des Smings erweitert werden. Dies ist nicht Straight-Forward, weil für die Entwicklung in C und das Deployment auf den NRF51 spezielle Hardware und Software erforderlich ist. Um dieses Problem zu umgehen, wurde die BLE-I2C Bridge entwickelt. Sie soll dazu dienen, dass vom IoT-Gateway via Sming I2C Sensoren und Aktoren angesprochen werden können. Somit muss für einen neuen Sensor die Firmware der Smings nicht mehr angerührt werden. Dafür muss aber das IoT-Gateway programmiert werden. Dieses lässt sich je nach Gateway Typ leicht über Ethernet oder USB programmieren. Auch die Softwareentwicklung in JavaScript sollte den meisten Entwickler in der Informatik-Branche weniger schwer fallen als dies bei C der Fall ist.

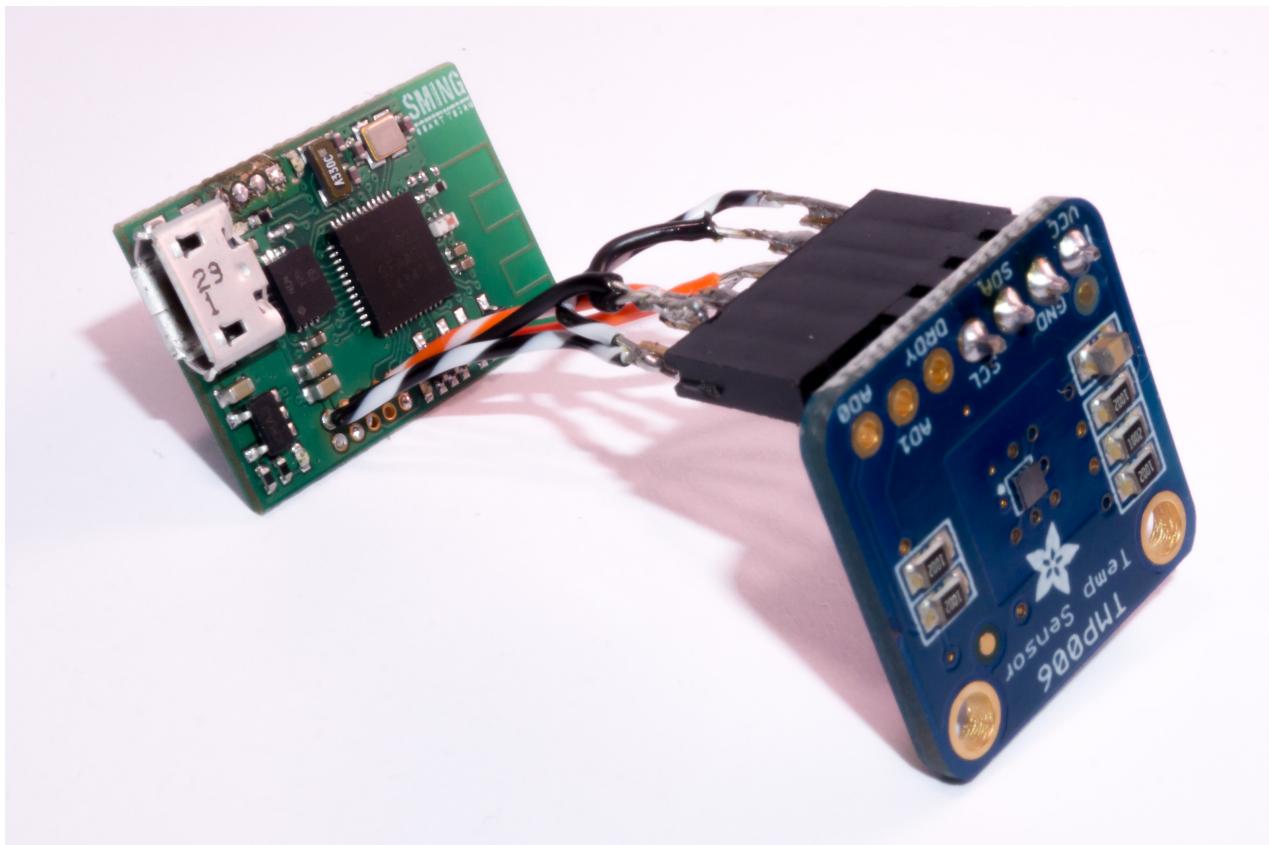


Abbildung 4.2.: Sming mit über I2C Bridge abfragbaren TMP06 Sensor

## 4.2. Implementation

Für die BLE-I2C Bridge wurde ein eigener BLE-Service aufgesetzt. Dieser wird in den nachfolgenden Tabellen beschrieben:

Name	I2C Service
Beschreibung	Ein Service für I2C Devices über BLE anzusprechen
UUID	0x8EDF0500-67E5-DB83-F85B-A1E2AB1C9E7A

Folgende Charakteristiken stehen bei dem Service zur Verfügung:

Name	I2C Device Adress
Beschreibung	Wird verwendet um die I2C Adresse des Sensor (oder Aktor) im Sming einzustellen.
UUID	0x8EDF0501-67E5-DB83-F85B-A1E2AB1C9E7A
Anzahl Byte	1
Zugriffsrechte	R/W
Wertebereich	0x00 .. 0x7F
Wertebereich	0x00

Name	I2C Device Register
Beschreibung	Wird verwendet um das I2C Register des Sensor (oder Aktor) im Sming einzustellen, auf welches geschrieben oder von welchem gelesen werden soll.
UUID	0x8EDF0502-67E5-DB83-F85B-A1E2AB1C9E7A
Anzahl Byte	1
Zugriffsrechte	R/W
Wertebereich	0x00 .. 0xFF
Wertebereich	0x00

Name	I2C Read Length
Beschreibung	Wird verwendet um die Anzahl Bytes einzustellen, welche von dem eingestellten Register gelesen werden.
UUID	0x8EDF0503-67E5-DB83-F85B-A1E2AB1C9E7A
Anzahl Byte	1
Zugriffsrechte	R/W
Wertebereich	0x00 .. 0xFF
Wertebereich	0x00

Name	I2C Read Length
Beschreibung	Wird verwendet um die eingestellte Anzahl Bytes von dem eingestellten Register zu lesen oder die in der Charakteristik mitgegebene Anzahl Bytes auf das angegebene Register zu schreiben.
UUID	0x8EDF0504-67E5-DB83-F85B-A1E2AB1C9E7A
Anzahl Byte	1 .. 10
Zugriffsrechte	R/W
Wertebereich	0x00 .. 0xFF
Wertebereich	0x00

# Selbständigkeitserklärung

Wir bestätigen, dass wir die vorliegende Arbeit selbstständig und ohne Benutzung anderer als der im Literaturverzeichnis angegebenen Quellen und Hilfsmittel angefertigt haben. Sämtliche Textstellen, die nicht von uns stammen, sind als Zitate gekennzeichnet und mit dem genauen Hinweis auf ihre Herkunft versehen.

Ort, Datum: Bern, 16.06.2015

Namen Vornamen: Pascal Bohni Roger Jaggi

Unterschriften: ..... .

# Literaturverzeichnis

- [1] D. Meer, *Master Thesis*. BFH, 2015.

# Abbildungsverzeichnis

2.1. Gesamtarchitektur Telemetrie . . . . .	2
3.1. Aufbau des IoT Gateways . . . . .	3
3.2. Foto NetModule NB1600 . . . . .	4
3.3. Foto CanCape auf Beagle Bone Black . . . . .	4
3.4. Foto BLED112 . . . . .	5
4.1. Foto Sming mit Akku bestückt . . . . .	6
4.2. Sming mit über I2C Bridge abfragbaren TMP06 Sensor . . . . .	7
B.1. Installation CAN-Cape . . . . .	13
F.1. Zeitplan Projekt 2 . . . . .	20

## A. BeagleBone Hardware Spezifikation

- Prozessor: AM335x 1GHz ARM® Cortex-A8
- RAM: 512MB DDR3 RAM
- HDD: 4GB 8-bit eMMC on-board flash storage
- GPU: 3D graphics accelerator + NEON floating-point accelerator
- 2x PRU 32-bit microcontrollers
- USB client for power and communications (Ethernet over USB)
- USB host
- Ethernet
- HDMI
- 2x 46 pin headers (GPIO, I2C, SPI, UART, CAN, etc.)
- Betriebssysteme:
  - Debian
  - Android
  - Ubuntu
  - Cloud9 IDE on Node.js w/ BoneScript library
  - plus jedes weitere ARM kompatible Betriebssystem

## B. Setup Beaglebone Black mit CAN Cape



Abbildung B.1.: Installation CAN-Cape

### B.1. Install latest Kernel image

See [http://elinux.org/BeagleBoardDebian#Install\\_Latest\\_Kernel\\_Image](http://elinux.org/BeagleBoardDebian#Install_Latest_Kernel_Image)

## B.2. Add default route temporary until next restart

```
ip route add default via 192.168.7.1
```

### B.3. Add default route permanent

### B.3.1. Repair file

New content for /etc/init.d/led\_aging.sh

```
#!/bin/sh -e
### BEGIN INIT INFO
# Provides: led_aging.sh
# Required-Start: $local_fs
# Required-Stop: $local_fs
# Default-Start: 2 3 4 5
# Default-Stop: 0 1 6
# Short-Description: Start LED aging
```

```

# Description:      Starts LED aging (whatever that is)
### END INIT INFO

x=$(/bin/ps -ef | /bin/grep "[l]ed_acc")
if [ ! -n "$x" -a -x /usr/bin/led_acc ]; then
    /usr/bin/led_acc &
fi

```

### B.3.2. Make the default gateway script executable

```
chmod +x /etc/network/if-up.d/defaultgw
```

### B.3.3. Install post networking script

```
touch /etc/init.d/initnet && nano /etc/init.d/initnet
```

Insert following content:

```

#!/bin/sh
### BEGIN INIT INFO
# Provides: initnet
# Required-Start: $all
# Required-Stop: $all
# Default-Start: 2 3 4 5
# Default-Stop: 0 1 6
# Short-Description: Short script description
# Description: Longer script description.
### END INIT INFO
case "$1" in
    start)
        /sbin/ip route add default via 192.168.7.1
        #/sbin/route add default gw 192.168.7.1 dev usb
        /usr/sbin/ntpdate pool.ntp.org
        ;;
    stop)
        #no-op
        ;;
    *)
        #no-op
        ;;
esac

exit 0

# Make it executable
chmod +x /etc/init.d/initnet
# Test the new Service
insserv -n /etc/init.d/initnet
# Activate new service
insserv /etc/init.d/initnet

```

## B.4. Configure NAT on Ubuntu Host PC

Add a File named beaglebonenat (without file extension) in /etc/network/if-up.d and change settings to meet your environment

```

#!/bin/sh -e
#
# case "$IFACE" in
#   eth3)
#     /bin/echo > /proc/sys/net/ipv4/ip_forward 1
#     /sbin/iptables -t nat -A POSTROUTING -s 192.168.7.0/24 -o wlan0 -j MASQUERADE
#   ;;
# esac

```

Make the file executable:

```
chmod +x /etc/network/if-up.d/beaglebonenat
```

## B.5. Install TowerTech TT3201 rev 5 on BeagleBone Debian

### B.5.1. Get kernel headers

be sure you have the kernel headers. If your kernel is 3.8.13-bone70, install as follow:

```
apt-get update
apt-get install linux-headers-3.8.13-bone70
```

When paket on apt-get install is not found, install Linux Headers for v3.8.13-bone71

```
wget https://rcn-ee.net/deb/wheezy-armhf/v3.8.13-bone71/linux-headers-3.8.13-bone71_1w
dpkg -i linux-headers-3.8.13-bone71_1wheezy_armhf.deb
```

### B.5.2. Invoke the build script

```
sh build.sh
```

The installed modules will be in `/lib/modules/`uname -r`/extra` and the firmware in `/lib/firmware/TT3201*`

### B.5.3. Configure Beaglebone Cape Manager

Edit `/etc/default/capemgr` to add the TT3201, it should look like this:

```
# Default settings for capemgr. This file is sourced by /bin/sh from
# /etc/init.d/capemgr.sh

# Options to pass to capemgr
CAPE="TT3201-001:05"
```

### B.5.4. Disable HDMI Cape on BeagleBone Black

Disable the HDMI port by editing `uEnv.txt` on your `/boot` partition. It should have an entry like this

```
##Disable HDMI
cape_disable=capemgr.disable_partno=BB-BONELT-HDMI,BB-BONELT-HDMIN
```

### B.5.5. Reboot and check cape detection

Reboot and your cape should be detected automatically, check with

```
cat /proc/cmdline | grep -q "L Bone-Black-HDMI" && echo "ERROR: hdmi NOT disabled"

cat /sys/devices/bone_capemgr.*/slots | grep TT3201
8: ff:P-O-L Override Board Name,05 ,Override Manuf,TT3201-001

ip link | grep can[0-9]
3: can0: mtu 16 qdisc pfifo_fast state UNKNOWN mode DEFAULT qlen 10
4: can1: mtu 16 qdisc noop state DOWN mode DEFAULT qlen 10
5: can2: mtu 16 qdisc noop state DOWN mode DEFAULT qlen 10
```

### B.5.6. Check CAN detection

```
# cat /sys/devices/bone_capemgr.*/slots | grep TT3201
0: 54:P---L TT3201 CAN Cape,01,TowerTech,TT3201-001

# ip link | grep can[0-9]
3: can0: mtu 16 qdisc pfifo_fast state UNKNOWN mode DEFAULT qlen 10
4: can1: mtu 16 qdisc noop state DOWN mode DEFAULT qlen 10
5: can2: mtu 16 qdisc noop state DOWN mode DEFAULT qlen 10

# dmesg | grep mcp251x
[    7.170413] mcp251x spi1.0: mode 0, irq 259, awake 1, clkout 1, oscillator freq 16000000
[    7.186391] mcp251x spi1.0 can1: probed
[    7.257289] mcp251x spi1.1: mode 0, irq 261, awake 1, clkout 0, oscillator freq 16000000
[    7.371486] mcp251x spi1.1 can2: probed

# dmesg | grep c_can
[    6.712145] c_can_platform 481d0000.d_can: invalid resource
[    6.718084] c_can_platform 481d0000.d_can: control memory is not used for raminit
[    6.799197] c_can_platform 481d0000.d_can: c_can_platform device registered (regs=f0000000)
```

### B.5.7. Install CAN tools

```
git clone https://github.com/linux-can/can-utils.git
cd can-utils
make
make install
```

### B.5.8. CAN tools examples

```
ip link set can0 type can bitrate 125000
ip link set can1 type can bitrate 125000

ip link set can0 up
ip link set can1 up

cansend can1 00000104#02aa
candump can0
```

## C. Beispiel Bluegiga Anbindung

Beispiel-Befehlsabfolge, wie sie beim Sming zum Einsatz kommt. Detailbeschrieb der Parameter siehe Bluegiga API 1.3 Doku. Die Parameter-Werte entsprechen denen im Demo-Script und sollten lauffähig sein:

- Suche nach BLE Advertisements
  - bg.api.gapSetScanParameters(0xC8, 0xC8, 0)
  - bg.api.gapDiscover(1)
- Wenn ein Device mit Name TXW51 gefunden, beende Suche kommt als Event mit Class = GenericAccess-Profile siehe Demo-Script ab Zeile 238
  - Suche beenden: bg.api.gapEndProcedure()
- Verbinde zum gefundenen Device anhand seiner MAC-Adresse
  - bg.api.gapConnectDirect( <MAC-Addr>, 1, 60, 76, 100, 9)
- Hole einfach mal komplette Handle-Liste der Bluetooth Attribute (Optimierungspotential da)
  - bg.api.attClientFindInformation( <connectionHandle>, 1, 0xffff)
- Stimme die SMING UUIDs mit denen in der Handle-Liste überein. Merke zu jeder UUID deren Handle sowie UUID der CCID Attributes (siehe später)
- Lese mit den entsprechenden Handle das Attribut (ich habe einfach mal jeweils immer gerade mal alle eingelesen)
  - bg.api.attClientReadByHandle(<connection>, <uuid handle> )
- Zum Aktivieren der Accelerometer-Messungen müssen folgende Attributte geschrieben werden:
  - bg.api.attClientAttributeWrite( <connection>, <uuid handle>, <newValue>)
  - LSM330\_CHAR\_GYRO\_EN = 1
  - LSM330\_CHAR\_ACC\_EN = 1
  - CCIDUUID0x02, 0x29=0x01, 0x00 ( Dies aktiviert das Bluetooth Feature CCID, wodurch das Sming bei Sming seitiger Attributwertänderung automatisch den neuen Wert sendet. Die Messwerte werden so per Push übertragen. Seitens Sming wird somit einfach der Messwert auf das Attribut MEASURE\_CHAR\_DATASTREAM geschrieben. Dies löst aus, dass der Bluetooth Stack des Smings automatisch den neuen Wert als Event zum Bluegiga Dongle überträgt und wir das so mitkriegen)
  - MEASURE\_CHAR\_START = 1
- Das Lesen der Temperatur muss mit einem Polling auf Attribut LSM330\_CHAR\_TEMP\_SAMPLE gemacht werden.
- Beende Verbindung zum Sming
  - bg.api.connectionDisconnect( <connection> )

# **D. CAN-Bus-Anbindung**

## **D.1. CAN-Tutorial**

CAN Infos: [http://www.computer-solutions.co.uk/info/Embedded\\_tutorials/canTutorial.htm](http://www.computer-solutions.co.uk/info/Embedded_tutorials/canTutorial.htm)

### **D.1.1. BeagleBone UART**

<http://beaglebone.cameon.net/home/serial-ports-uart>

## **D.2. CAN Hardware**

### **D.2.1. Seeedstudio Arduino Shield**

[http://www.seeedstudio.com/wiki/CAN-BUS\\_Shield](http://www.seeedstudio.com/wiki/CAN-BUS_Shield) [https://github.com/Seeed-Studio/CAN\\_BUS\\_Shield/blob/master/mcp\\_can.cpp](https://github.com/Seeed-Studio/CAN_BUS_Shield/blob/master/mcp_can.cpp) [http://www.seeedstudio.com/wiki/images/7/78/CAN-BUS\\_Shield\\_v0.9b.pdf](http://www.seeedstudio.com/wiki/images/7/78/CAN-BUS_Shield_v0.9b.pdf)

### **D.2.2. Treiber-Chip**

Empfohlen von BFS, Joel Wenger:

Als Treiber: [http://www.distrelec.ch/Web/Downloads/\\_t/ds/MCP2562FD-E-MF\\_eng\\_tds.pdf](http://www.distrelec.ch/Web/Downloads/_t/ds/MCP2562FD-E-MF_eng_tds.pdf)

Als Controller, der CAN-Bus Spezifikation implementiert hat und per SPI gesteuert wird: [http://www.distrelec.ch/Web/Downloads/\\_t/ds/MCP2515\\_eng\\_tds-2.pdf](http://www.distrelec.ch/Web/Downloads/_t/ds/MCP2515_eng_tds-2.pdf)

### **D.2.3. NetModule CAN Anbindung**

Software: <ftp://share.netmodule.com/router/public/system-software/3.7/3.7.2.104/>

Infos dazu siehe Seite 26: [ftp://share.netmodule.com/router/public/system-software/3.7/3.7.2.104/NB\\_SDK\\_API\\_Manual\\_3.7.2.104.pdf](ftp://share.netmodule.com/router/public/system-software/3.7/3.7.2.104/NB_SDK_API_Manual_3.7.2.104.pdf)

## **D.3. CAN Simulations-Software**

Can-Easy:

<http://automotive.softing.com/de/produkte/caneasy.html> <http://automotive.softing.com/de/produkte/kommunikations-interfaces-can.html>

## E. Liste der genutzen Attribut UUIDs des SMING

DEVICE_INFO_SERVICE	:	"8 EDF0100-67E5-DB83-F85B-A1E2AB1C9E7A"
DEVICE_INFO_CHAR_MANUFACTURER	:	"8 EDF0101-67E5-DB83-F85B-A1E2AB1C9E7A"
DEVICE_INFO_CHAR_MODEL	:	"8 EDF0102-67E5-DB83-F85B-A1E2AB1C9E7A"
DEVICE_INFO_CHAR_SERIAL	:	"8 EDF0103-67E5-DB83-F85B-A1E2AB1C9E7A"
DEVICE_INFO_CHAR_HW_REV	:	"8 EDF0104-67E5-DB83-F85B-A1E2AB1C9E7A"
DEVICE_INFO_CHAR_FW_REV	:	"8 EDF0105-67E5-DB83-F85B-A1E2AB1C9E7A"
DEVICE_INFO_CHAR_DEVICE_NAME	:	"8 EDF0106-67E5-DB83-F85B-A1E2AB1C9E7A"
DEVICE_INFO_CHAR_SAVE_VALUES	:	"8 EDF0107-67E5-DB83-F85B-A1E2AB1C9E7A"
LSM330_SERVICE	:	"8 EDF0200-67E5-DB83-F85B-A1E2AB1C9E7A"
LSM330_CHAR_ACC_EN	:	"8 EDF0201-67E5-DB83-F85B-A1E2AB1C9E7A"
LSM330_CHAR_GYRO_EN	:	"8 EDF0202-67E5-DB83-F85B-A1E2AB1C9E7A"
LSM330_CHAR_TEMP_SAMPLE	:	"8 EDF0203-67E5-DB83-F85B-A1E2AB1C9E7A"
LSM330_CHAR_ACC_FSCALE	:	"8 EDF0204-67E5-DB83-F85B-A1E2AB1C9E7A"
LSM330_CHAR_GYRO_FSCALE	:	"8 EDF0205-67E5-DB83-F85B-A1E2AB1C9E7A"
LSM330_CHAR_ACC_ODR	:	"8 EDF0206-67E5-DB83-F85B-A1E2AB1C9E7A"
LSM330_CHAR_GYRO_ODR	:	"8 EDF0207-67E5-DB83-F85B-A1E2AB1C9E7A"
LSM330_CHAR_TRIGGER_VAL	:	"8 EDF0208-67E5-DB83-F85B-A1E2AB1C9E7A"
LSM330_CHAR_TRIGGER_AXIS	:	"8 EDF0209-67E5-DB83-F85B-A1E2AB1C9E7A"
MEASURE_SERVICE	:	"8 EDF0300-67E5-DB83-F85B-A1E2AB1C9E7A"
MEASURE_CHAR_START	:	"8 EDF0301-67E5-DB83-F85B-A1E2AB1C9E7A"
MEASURE_CHAR_STOP	:	"8 EDF0302-67E5-DB83-F85B-A1E2AB1C9E7A"
MEASURE_CHAR_DURATION	:	"8 EDF0303-67E5-DB83-F85B-A1E2AB1C9E7A"
MEASURE_CHAR_DATASTREAM	:	"8 EDF0304-67E5-DB83-F85B-A1E2AB1C9E7A"

## F. Zeitplan

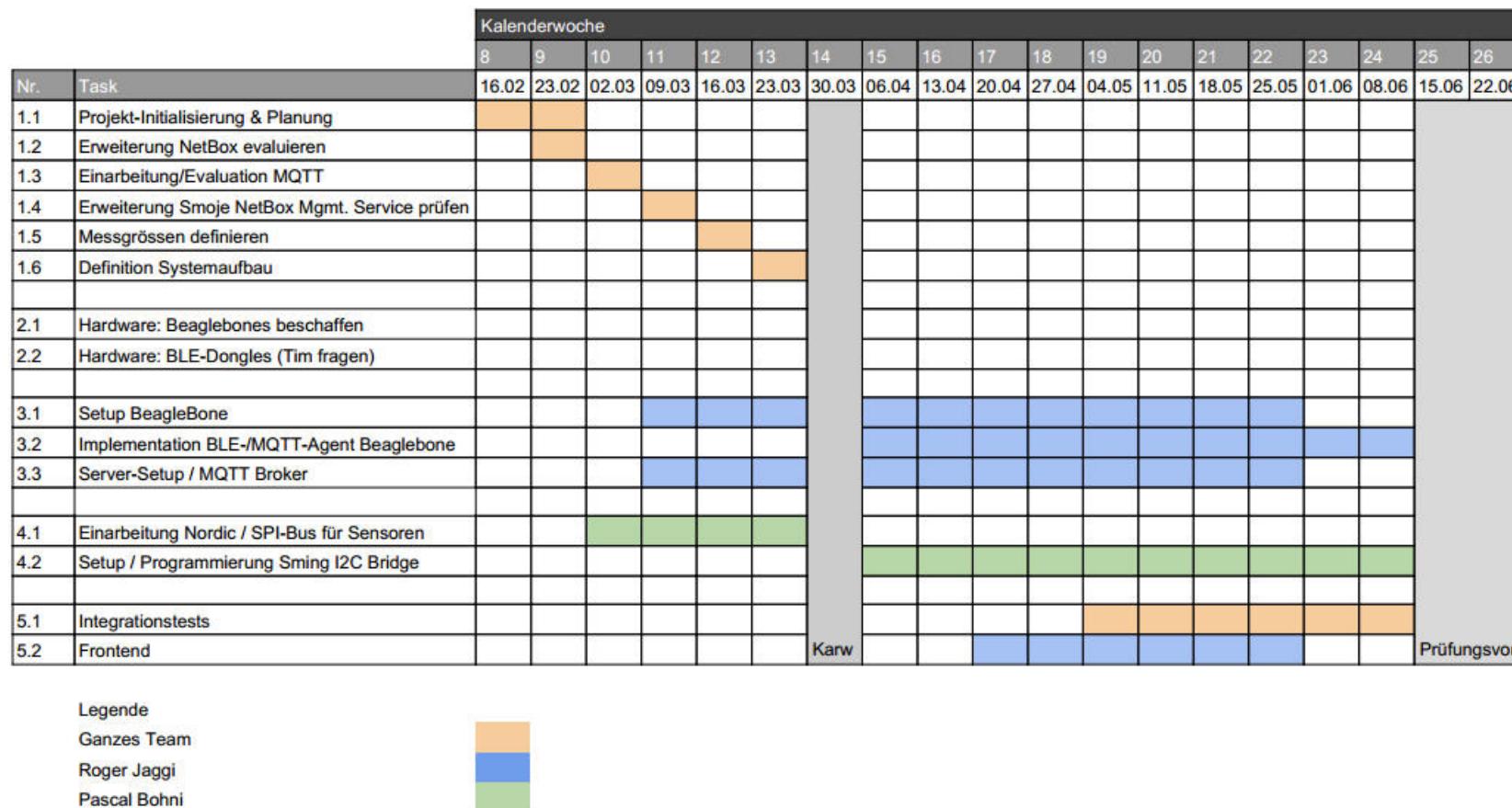


Abbildung F.1.: Zeitplan Projekt 2