
# Zwischenbericht für das Forschungsseminar Sensornetze Sommersemester 2016

im Master-Studiengang Angewandte Informationstechnologien

eingereicht von:

Kai Richter und
Alexander Gräb

Betreuer: Herr Prof. Dr.-Ing. Jörg Vogt

Eingereicht: 31.08.2016

# Inhalt

# Abbildungsverzeichnis

# Tabellenverzeichnis

# Abkürzungsverzeichnis

**WPAN** Wireless Personal Area Networks

**6LoWPAN** IPv6 over Low power Wireless Personal Area Networks

**CoAP** Constrained Application Protocol

**RSSI** Received Signal Strength Indicator

# 1 Eingesetzte Hardware

Die hier eingesetzte Hardware wird unterschieden in Border-Router und Sensor-Knoten (Nodes). Die Kommunikation zwischen den Geräten findet via IEEE 802.15.4 statt, einem Funkstandard für Wireless Personal Area Networks (WPAN). Als Protokoll für die Datenübertragung kommt IPv6 zum Einsatz. Der Vorteil des Einsatzes von IPv6 ist die leichtere Integration der Sensornetze in das Internet und dass auf die vielfältigen und bewährten Technologien aus der IP-Welt zurückgegriffen werden kann. IPv6 erfordert jedoch relativ viel Speicher und Rechenleistung für die Anwendung in Sensornetzen. Dies führt wiederum zu einem relativ hohen Energieverbrauch. Energie sparen ist jedoch ein wichtiges Thema, da die Nodes häufig mit Batterien bzw. Akkus betrieben werden. Daher wird tatsächlich IPv6 over Low power Wireless Personal Area Networks (6LoWPAN) als Protokoll verwendet. Dabei handelt es sich um eine leichtgewichtige Form von IPv6, welche optimiert ist für den Einsatz auf Geräten mit stark beschränkten Ressourcen.

Die Nodes übertragen beliebige Messwerte (Temperatur, Luftfäuchtigkeit, etc.) an eine Station oder dienen als Aktoren, welche von einer Station ferngesteuert werden. Eine Station ist dabei ein beliebiger Rechner in einem IPv6-Netzwerk. Der Border-Router dient als Übergangspunkt für die Nodes, welche über IEEE 802.15.4 kommunizieren, in ein „richtiges“ IPv6-Netzwerk, welches bspw. mit dem Internet verbunden ist. Der Border-Router wird mit Hilfe einer UART-Bridge über USB mit einem beliebigen Rechner verbunden, welcher als Router dienen kann (siehe Abb. 1.1).

Zum Einsatz kommt der 8-Bit Mikrocontroller ATmega128RFA1 von Atmel. Er verfügt über 128 KByte ROM, 16 KByte Arbeitsspeicher, einer integrierten IEEE 802.15.4 Funkeinheit und kann mit einem CPU-Takt von bis zu 16 MHz betrieben werden. Die Mikrocontroller wurden in Form eines Funkmoduls deRFmega128-22A00 von dresden elektronik[1] angeschafft. Das Modul enthält unter anderem einen 16 MHz Quarzoszillator als Referenztaktgeber für die Funkeinheit des ATmega128RFA1, welcher aber auch als Taktgeber für die CPU verwendet werden kann, einem 32,768 kHz Taktgeber für den Deep-Sleep-Modus[2] und eine Chip-Antenne[3].

Im Falle der Nodes wird das Funkmodul auf ein Bord des Typs deRFnode von dresden elektronik aufgesteckt (siehe Abb. 1.1). Dieses Bord verfügt unter anderem über Spannungsregler, um das Funkmodul mit einer stabilen Spannung aus einer von drei möglichen Spannungsquellen[4] zu versorgen, einiger Sensoren, wie einem Temperatur-, Helligkeits-, Beschleunigungs- und Spannungssensor, einer USB-Schnittstelle und einem Batteriefach

[1] `https://www.dresden-elektronik.de`

[2] Ein sehr tiefer Schlafmodus, bei welchem der ATmega128RFA1 nur noch etwa 250 nA verbraucht.

[3] Eine aufgelötete Antenne in SMB-bauweise.

[4] Das Bord lässt sich entweder über USB, einem, über eine Buchse angeschlossenen, externen Netzteil oder Batterien/Akkus mit Strom versorgen.

für drei Batterien respektive Akkus im Format AA.

Beim Border-Router ist das Funkmodul auf ein Breakout-Bord des Typs deRFbreakout Board von dresden elektronik aufgesteckt (siehe Abb. 1.1). Das deRFbreakout Board erleichtert den Zugang zu den Anschlüssen des Funkmoduls bzw. des Mikrocontrollers über Schraubklemmen und verfügt außerdem über Steckverbinder für z. B. JTAG (einer Schnittstelle zum Programmieren des Mikrocontrollers).

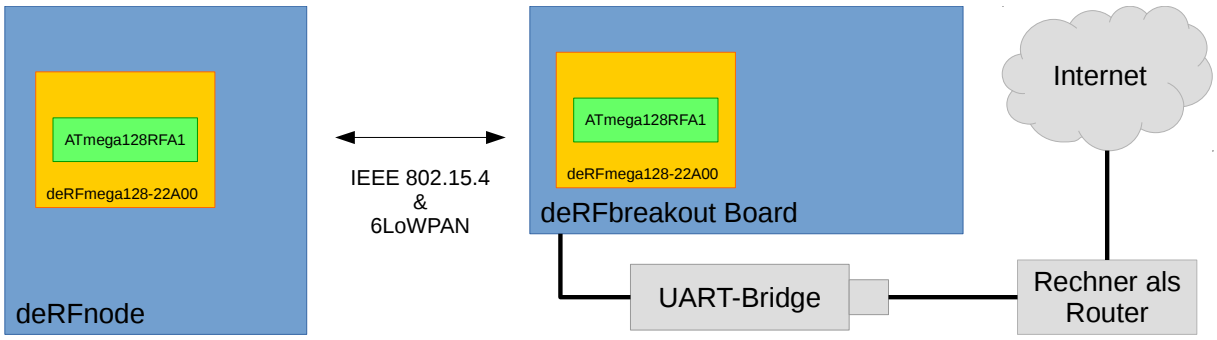


Abb. 1.1: Der Aufbau der Hardwarekomponenten

# 2 Contiki 3.0 kompilieren und flashen

Um Contiki 3.0 verwenden zu können, muss der Quellcode beschafft, die entsprechende Konfiguration vorgenommen und der Quellcode kompiliert werden. Anschließend kann das kompilierte Programm auf den Mikrocontroller geflasht werden. Das Flashen bezeichnet den Vorgang, bei welchem das Programm mit Hilfe eines Programmiergerätes, wie z. B. dem AVR Dragon, in den Flash-Speicher des Mikrocontrollers geschrieben wird.

## 2.1 Verwenden der Original-Quellen

Der original Quellcode von Contiki 3.0 wird in einem GitHub-Repository[5] verwaltet. Die hier aufgeführten Kommandos wurden auf einem Linux System getestet und müssen für andere Systeme entsprechend adaptiert werden. Der Quellcode wird stetig weiter entwickelt. Daher ist es möglich, dass die hier beschriebenen Instruktionen für einen neueren Stand[6] der Quellen ebenfalls angepasst werden müssen. Ferner wird vorrausgesetzt, dass Git, der Crosscompiler gcc-avr, die AVR C-Bibliothek und Avrdude installiert sind. Auf einem Debian 8 (Jessie) GNU/Linux System lassen sich diese als Nutzer root mit einem einzigen Kommando installieren: `aptitude install git gcc-avr avr-libc avrdude` Folgende Beschreibung orientiert sich an [8].

```
1 mkdir contiki3
2 cd contiki3
3 git init
4 git clone --recursive https://github.com/contiki-os/contiki
```

Es wird ein Verzeichnis contiki3 angelegt (Zeile 1), in welchem ein neues Git-Repository initialisert wird (Zeile 2 u. 3). Anschließend wird der Inhalt des original Contiki 3.0 Repositorys von GitHub in das neu initialisiere Git-Repository geklont (Zeile 4). Im Verzeichnis contiki3 befindet sich nun ein weiteres Verzeichnis contiki. Alle weiteren Pfadangaben in diesem Kapitel sind relativ zu diesem Verzeichnis.
Für den Einsatz des gewünschten Programmiergerätes müssen die Konstanten `AVRDUDE_PROGRAMMER` und `AVRDUDE_PORT` in der Datei platform/avr-atmega128rfa1/Makefile.avr-atmega128rfa1 angepasst werden. Für den AVR Dragon sehen die Anpassungen wiefolgt aus:

```
...
AVRDUDE_PROGRAMMER=dragon_jtag
...
AVRDUDE_PORT=usb
...
```

[5] `https://github.com/contiki-os/contiki`
[6] Hier verwendet wurde der Stand des Quellcodes vom 21.05.2016.

Um den AVR Dragon unter Linux ohne root-Rechte nutzen zu können, sei der Wiki-Artikel unter [3] empfohlen.
In die Datei `platform/avr-atmega128rfa1/contiki-conf.h` muss noch folgende Zeile, etwa nach der Definition der letzen `UIP_CONF_*`-Konstante, eingefügt werden:

```
#define UIP_CONF_BUFFER_SIZE 240
```

Der Quellcode von Contiki enthält einige Beispiel-Projekte, welche zum Testen oder als Ausgangspunkt für eigene Projekte verwendet werden können.

```
1 mkdir project
2 cd project/
3 ln -s ../examples/ipv6/rpl-border-router/ border-router
4 cd border-router
5 make TARGET=avr-atmega128rfa1 savetarget
6 make border-router
7 cd ..
8 cp -r ../examples/er-rest-example/ er-rest-example
9 cd er-rest-example
10 make TARGET=avr-atmega128rfa1 savetarget
11 make er-example-server
```

Im Wurzelverzeichnis contiki wird dazu ein neues Verzeichnis project angelegt (Zeile 1). In diesem wird ein symbolischer Link border-router auf das Beispiel-Projekt rpl-border-router angelegt (Zeile 2 und 3). Das Projekt wird einmalig für die Zielplattform avr-atmega128rfa1 konfiguriert und kompiliert (Zeile 4 bis 6). Anschließend wird das Beispiel-Projekt er-rest-example kopiert[7] und ebenfalls für die Zielplattform avr-atmega128rfa1 konfiguriert und kompiliert (Zeile 7 bis 11). Nach jeder Änderung am Quellcode eines Projektes, muss dieses erneut mit `make` kompiliert werden. Wurden Dateien außerhalb des Projekt-Verzeichnisses geändert, müssen unter Umständen beide Projekte neu kompiliert (und geflasht (siehe Kap. 2.1.1)) werden.

### 2.1.1 Flashen des Mikrocontrollers

Um den Mikrocontroller auf der entsprechenden Hardware (Border-Router oder Node) mit der entsprechenden Software zu flashen, muss die Stromversorgung der Hardware aktiv sein[8] und das Programmiergerät mit der Hardware und dem Computer verbunden sein. Für den Border-Router muss innerhalb des Projekt-Verzeichnisses das Kommando `make border-router.u` ausgeführt werden. Vor dem Flash-Vorgang wird das Projekt neu kompiliert, wenn seit dem letzen Aufruf Änderungen am Quellcode vorgenommen wurden.

[7] Um das original Projekt zu erhalten, da hier üblicherweise viele Modifikationen im Qullcode vorgenommen werden.

[8] Beim Border-Router erfolgt die Stromversorgung über die UART-Bridge, sobald diese mit einem Computer verbunden ist. Bei der Node muss der Jumper JP2 auf die entsprechende Stromquelle Batterie/Akku oder USB/externes Netzteil eingestellt sein.

Daher genügt dieser Befehl, um Änderungen am Qullcode zu übersetzen und den Mikrocontroller gleich zu flashen. Analog funktioniert dies für eine Node, indem im Verzeichnis er-rest-example der Befehl `make er-example-server.u` aufgerufen wird. Unter Verwendung des AVR Dragons, müssen die Kommandos mit root-Rechten ausgeführt werden, es sei denn, die Anweisungen in [3] wurden befolgt.

## 2.2 Verwenden der Quellen der HTW Dresden

Die HTW Dresden verwaltet ihre Änderungen an den Quellen von Contiki in einem eigenen GitHub-Repository[9]. Dieses Repository war ursprünglich ein Fork[10] der Original-Quellen von Contiki 2.7. Im GitHub-Repository der HTW Dresden wurde erneut ein Fork der Original-Quellen von Contiki 3.0 mit Stand vom 21.05.2016 erstellt. Dafür wurde der separate Branch release-3-0 angelegt. Ein Git(Hub)-Repository kann ein oder mehrere Branches umfassen, in welchem verschiedene Stände eines Projektes unabhängig voneinander gepflegt werden können. Derzeit ist der Branch sn-2.7, in welchem die HTW Dresden eigenen Modifikationen am Contiki 2.7 Quellcode verwaltet werden, als Standard-Branch festgelegt. Dies gilt es unbedingt zu beachten, wenn mit den Quellen von Contiki 3.0 gearbeitet werden soll.

### 2.2.1 Erstellen eines lokalen Klons

Analog zu Kap. 2.1, kann ein lokaler Klon der Contiki 3.0 Quellen aus dem GitHub-Repository der HTW Dresden erzeugt werden:

```
1 mkdir contiki3
2 cd contiki3
3 git clone -b release-3-0 --recursive https://github.com/HTWDD-SN/contiki
```

Man beachte die Angabe des korrekten Branches mit `-b release-3-0`.

### 2.2.2 Kooperatives Arbeiten mit dem HTW Dresden GitHub-Repository

Sollen die eigenen Änderungen an den Contiki 3.0 Quellen wieder in das GitHub-Repository der HTW Dresden geschrieben werden bzw. sollen zwischenzeitlich durch die Kooperationspartner getätigte Änderungen im GitHub-Repository der HTW Dresden mit dem eigenen lokalen Repository zusammengeführt werden, empfiehlt sich das im Folgendem beschriebene Vorgehen.

#### Vorbereitung

Ein neuer Kooperationspartner erstellt zunächst eine Kopie des HTW Dresden GitHub-Repositorys als lokales Git-Repository und verknüpft sein lokales Git-Repository dabei mit

[9] `https://github.com/HTWDD-SN/contiki`

[10] Mit GitHub lassen sich Kopien von vorhandenen Projekten erstellen, welche dann unabhängig vom Original weiterentwickelt werden können.

einem Remote-Repository, in diesem Fall also das GitHub-Repository der HTW Dresden:

```
1 mkdir contiki3
2 cd contiki3
3 git init
4 git remote add origin https://github.com/HTWDD-SN/contiki
5 git fetch
6 git checkout release-3-0
7 git config user.name "Max␣Mustermann␣<s12345@htw-dresden.de>"
```

In den Zeilen 1 bis 3 wird in einem neu erstellten Verzeichnis ein lokales Git-Repository initiiert. Das lokale Git-Repository wird anschließend mit dem Remote-Repository verknüpft, welches als origin[11] bezeichnet wird (Zeile 4). Der Befehl in Zeile 5 lädt den aktuellen Stand des Remote-Repositorys, also des GitHub-Repositorys der HTW Dresden, herunter. Bis hier hin existiert nur das versteckte Verzeichnis .git als Unterverzeichnis von contiki3. In Zeile 6 wird der Stand des Branches release-3-0 zur Bearbeitung in den sog. *working tree* kopiert. Von nun an befindet sich in dem Verzeichnis contiki3 die typische Dateistruktur des Contiki-Quellcodes. Mit diesen Dateien wird gearbeitet. In Zeile 7 wird der Name des Kooperationspartners festgelegt, welchem das lokale Git-Repository gehört. Dieser Name erscheint bei jedem Commit[12], den der Kooperationspartner macht.

### Git-Repository mit den Änderungen der Kooperationspartner aktualisieren

Bei der Arbeit im Team sollten die Änderungen am Contiki-Quellcode, welche durch die Kooperationspartner vorgenommen wurden, regelmäßig mit dem Stand des eigenen lokalen Git-Repositorys zusammengeführt werden. Dies ist auch Vorraussetzung dafür, dass eigene Commits in das Remote-Repository geschrieben werden können. Sind Änderungen durch Kooperationspartner zu erwarten, sollte man sein lokales Git-Repository vor dem Start der eigenen Arbeit stets aktualisieren:

```
1 git fetch
2 git merge origin/release-3-0
```

Über den Befehl in Zeile 1 werden die neusten Änderungen aus dem Remote-Repository in das lokale Repository herunter geladen. In Zeile 2 werden diese Änderungen mit dem Stand des working trees (also mit den eigenen Änderungen) zusammen geführt.

### Eigene Änderungen in das Remote-Repository schreiben (einchecken)

Nach getaener Arbeit sollten die Änderungen im working tree des lokalen Git-Repositorys in das Remote-Repository der HTW Dresden auf GitHub eingecheckt werden:

[11] Da mehrere Remote-Repositories verwendet werden können, benötigt jedes einen eigenen Namen zur Unterscheidung. Bei einem Remote-Repository wird dieses häufig origin genannt.

[12] Das Committen bezeichnet den Vorgang, bei welchem ein Kooperationspartner seine Änderungen am Projekt im Repository eincheckt.

```
1 git fetch
2 git merge origin/release-3-0
3 git commit -a
4 git push origin release-3-0
```

Die Zeilen 1 bis 2 sind notwendig, wenn zwischenzeitlich Änderungen durch andere Kooperationspartner im Remote-Repository eingecheckt wurden. Git führt Änderungen weitestgehend automatisch zusammen, selbst, wenn ein und dieselbe Datei modifiziert wurde.[13] Nur in Ausnahmefällen treten dabei Konflikte auf (z. B. wenn dieselbe Zeile in derselben Datei modifiziert wurde), welche manuell behoben werden müssen. Der Befehl in Zeile 3 checkt die Änderungen, zunächst in das lokale Git-Repository, ein. Dabei öffnet sich der Standardtexteditor (z. B. vim oder nano), in welchem eine kurze Beschreibung der aktuellen Änderungen bzw. des Commits eingegeben werden sollte. Die Option *-a* sorgt dafür, dass automatisch alle geänderten und gelöschten (nicht aber neue) Dateien beachtet werden. Neue Dateien werden Git über den Befehl `git add pfad/zur/datei` bekannt gemacht. Git informiert den Nutzer darüber, welche Dateien geändert, gelöscht oder neu sind. So kann ein Commit abgebrochen werden, wenn bspw. vergessen wurde, eine neue Datei mit `git add` hinzuzufügen. Der Abbruch erfolgt, indem der Texteditor ohne zu speichern verlassen wird. Schließlich werden die Änderungen in das Remote-Repository geschrieben (Zeile 4). Der letzte Schritt setzt die Kenntnis der Zugangsdaten des GitHub-Accounts der HTW Dresden vorraus.

### 2.2.3 Kompilieren und flashen der Software

Das Kompilieren und Flashen der Software erfolgt analog zur Beschreibung in Kap. 2.1.1. Derzeit existieren die Projekte project/border-router, für den Border-Router und project/er-rest-example, für die Node.

Um die Software des Border-Routers zu kompilieren und zu flashen, muss der Border-Router über die UART-Bridge mit dem Computer (für die Stromversorung) und dem Programmiergerät verbunden werden, bevor der entsprechende `make`-Befehl im Projekt-Verzeichnis abgesetzt werden kann:

```
1 cd project/border-router
2 make border-router.u
```

Ausgehend vom Wurzelverzeichnis des working trees wird in das Projekt-Verzeichnis gewechselt (Zeile 1). Der Befehl in Zeile 2 kompiliert die Software und flasht diese anschließend auf den Mikrocontroller.

[13] D. h., es wurde eine Datei im working tree bearbeitet. Zwischenzeitlich hat ein Kooperationspartner dieselbe Datei bearbeitet und seine Änderungen bereits in das Remote-Repository eingecheckt.

Ein ähnliches Vorgehen gilt für eine Node. Wieder ausgehend vom Wurzelverzeichnis des working trees werden folgende Befehle abgesetzt:

```
1 cd project/er-rest-example
2 make er-example-server.u
```

Hinweis: In der Datei platform/avr-atmega128rfa1/Makefile.avr-atmega128rfa1 werden die Einstellungen für das verwendete Programmiergerät getroffen (vgl. Kap. 2.1). Der derzeitige Stand der Quellen im Branch release-3-0 geht von der Verwendung des Programmiergerätes AVR Dragon mit der JTAG-Schnittstelle aus.

## 2.3 Setzen der korrekten Fuse-Bits

Als Fuses wird eine Gruppe von Bits bezeichnet, welche in einem speziellen Bereich des Flash-Speichers des Mikrocontrollers stehen und der grundlegenden Konfiguration des Mikrocontrollers dienen. Beim Setzen der Fuses muss mit äußerster Vorsicht vorgegangen werden, da fehlerhaft gesetzte Fuse-Bits den Mikrocontroller unbrauchbar machen können. Weitere Informationen sind in [9] zu finden.

Die Fuses sind entscheidend dafür, mit welcher Taktfrequenz die CPU des Atmega128RFA1 arbeitet. Diese muss mit der Taktfrequenz übereinstimmen, welche im Quellcode von Contiki definiert wurde. Die Taktfrequenz wird im Quellcode über die Konstante F_CPU in Hz angegeben. F_CPU wird definiert in platform/avr-atmega128rfa1/contiki-conf.h oder in platform/avr-atmega128rfa1/Makefile.avr-atmega128rfa1.

Die Autoren verwenden den 16 MHz Referenz-Taktgeber des Funkmoduls (Transceiver Crystal Oscillator) als Taktgeber für die CPU, da er ohnehin für die Funkeinheit aktiv sein muss und einen viel genaueren und stabileren Takt liefert, als der voreingestellte interne RC Oszillator (Calibrated Internal RC Oscillator). Für diesen Taktgeber können per Register verschiedene Vorteiler aktiviert werden, um den Takt für die CPU bspw. auf 8 MHz herunter zu setzen (siehe Kap. 4.3). Weitere Informationen zu den Taktquellen finden sich in [11].

Die Fuse-Bits des Atmega128RFA1 sind in drei Gruppen zu je 8 Bits unterteilt, welche als Low, High und Extended bezeichnet werden. (Das Low-Fuse-Byte ist dabei für die Auswahl der Taktquelle verantwortlich.) Die Autoren verwenden für die Border-Router und die Nodes die Fuses E:FE, H:91 und L:D7. Dies ist die hexadezimal kodierte Angabe der Fuse-Bits, wobei der Buchstabe vor dem Doppelpunkt für Extended, High oder Low steht.

### Auslesen der Fuses

Die Fuses werden bei jedem Flash-Vorgang (vgl. Kap. 2.1.1) angezeigt. Eine andere Möglichkeit besteht darin, die Fuses mit Avrdude anzeigen zu lassen:

```
$ avrdude -c dragon_jtag -p m128rfa1 -n

avrdude: jtagmkII_initialize(): warning: OCDEN fuse not programmed, single-byte
    EEPROM updates not possible
avrdude: AVR device initialized and ready to accept instructions

Reading | ################################################## | 100% 0.00s

avrdude: Device signature = 0x1ea701

avrdude: safemode: Fuses OK (E:FE, H:91, L:D7)

avrdude done. Thank you.
```

Die Option `-c dragon_jtag` gibt an, dass das Programmiergerät AVR Dragon mit der JTAG-Schnittstelle verwendet wird. Bei Verwendung eines anderen Programmiergerätes bzw. Schnittstelle, muss `dragon_jtag` entsprechend ersetzt werden.

### Schreiben der Fuses

Folgender Befehl schreibt die Fuses:

```
1 avrdude -c dragon_jtag -p m128rfa1 -U lfuse:w:0xd7:m -U hfuse:w:0x91:m -U
    efuse:w:0xfe:m
```

Die Option `-U` wird hier mehrmals angewendet, um nacheinander die Low-Fuses (lfuse), High-Fuses (hfuse) und Extended-Fuses (efuse) zu setzen.

# 3 Inbetriebnahme und Test der Verbindung

Dieses Kapitel beschreibt die Inbetriebnahme der Hardware mit Contiki 3.0 aus den GitHub-Quellen der HTW Dresden und wie die Verbindung zwischen Border-Router und Node(s) mit einfachen Mitteln überprüft werden kann. Vorraussetzung ist, dass der Quellcode von Contiki 3.0 gemäß Kap. 2.2 beschafft wurde und die Projekte border-router und er-rest-example kompiliert und auf die entsprechende Hardware geflasht wurde. Desweiteren sollten die Fuses korrekt gesetzt sein (siehe Kap. 2.3).

## 3.1 tunslip6 kompilieren

tunslip6 erstellt eine virtuelle Netzwerkschnittstelle mit einer IPv6-Adresse im System, über welche der Datenaustausch mit dem Border-Router und allen mit dem Border-Router verbundenen Nodes, möglich ist. Der Computer, welcher tunslip6 ausführt, wird somit Teil des Sensor-Netzwerkes. Im Wurzelverzeichnis des Quellcodes von Contiki befindet sich das Unterverzeichnis tools. Dort befindet sich der Quellcode von tunslip6, welcher mit `make` leicht kompiliert werden kann:

```
1 cd tools
2 make tunslip6
```

## 3.2 tunslip6 ausführen

Zuvor muss der Border-Router mit dem Computer, welcher tunslip6 ausführen soll, verbunden sein. Auch ist darauf zu achten, dass die UART-Bridge an den korrekten UART-Port des Atmega128RFA1 angeschlossen ist. Der Atmega128RFA1 kennt den UART-Port 0 (Anschlüsse TXD0 und RXD0) und UART-Port 1 (Anschlüsse TXD1 und RXD1). Contiki 3.0 aus den Quellen der HTW Dresden sowie Contiki 3.0 aus den Original-Quellen nutzen beide UART-Port 0[14]. Die korrekten Anschlüsse sind dem Datenblatt des deRF-breakout Boards zu entnehmen. tunslip6 wird als root gestartet:

```
1 sudo ./tunslip6 2002:8d38:831a:aaaa::1/64 -s /dev/ttyUSB0 -B 38400 -t tap0 -v4
      -L
```

Die IPv6-Adresse der virtuellen Netzwerkschnittstelle lautet im Beispiel 2002:8d38:831a:aaaa::1. Dies ist die Adresse, welche der tunslip6-ausführende Computer haben wird. Mit der Option `-s` wird die Gerätedatei angegeben, welche der UART-Bridge im System zugewiesen wurde. Mit `-B` wird die Baudrate angegeben, über welche der Atmega128RFA1 und die UART-Bridge miteinander kommunizieren. Die Option `-t` gibt an, wie der Name der virtuellen Netzwerkschnittstelle lauten soll. Mit `-v4` wird das Level der Ausgabe (die

[14] Der UART-Port 0 der Funkmodule deRFmega128-22A00 lässt sich wohl bei einigen Modellen mit der Nr. 1229 nicht nutzen (Quelle: E-Mail von Herrn Jens Paul vom 10.06.2016). Hier muss auf UART-Port 1 gewechselt werden und die Software-Konfiguration entsprechend angepasst werden.

Gesprächigkeit) angegeben und `-L` fügt den Ausgaben Zeitstempel hinzu. Eine Hilfe zur Benutzung lässt sich außerdem mit `./tunslip6 -help` anzeigen.

Dies Ausgabe von tunslip6 sieht in etwa wiefolt aus:

```
19:52:26 ********SLIP started on ``/dev/ttyUSB0''
0000.015 opened tun device ``/dev/tap0''
0000.015 ifconfig tap0 inet `hostname` mtu 1500 up
0000.041 ifconfig tap0 add 2002:8d38:831a:aaaa::1/64
0000.058 ifconfig tap0 add fe80::8c38:831a:aaaa:1/64
0000.075 ifconfig tap0

tap0 Link encap:UNSPEC Hardware Adresse
    00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00
          inet Adresse:127.0.1.1 P-z-P:127.0.1.1 Maske:255.255.255.255
          inet6-Adresse: fe80::8c38:831a:aaaa:1/64 Gueltigkeitsbereich:Verbindung
          inet6-Adresse: 2002:8d38:831a:aaaa::1/64 Gueltigkeitsbereich:Global
          UP PUNKTZUPUNKT RUNNING NOARP MULTICAST MTU:1500
              Metrik:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          Kollisionen:0 Sendewarteschlangenlaenge:500
          RX bytes:0 (0.0 B) TX bytes:0 (0.0 B)

0000.095 Packet from TUN of length 48 - write SLIP
0000.944 *** Address:2002:8d38:831a:aaaa::1 => 2002:8d38:831a:aaaa
Got configuration message of type P
0000.969 Got configuration message of type P
Setting prefix 2002:8d38:831a:aaaa::
0000.969 Setting prefix 2002:8d38:831a:aaaa::
Server IPv6 addresses:
0001.967 Server IPv6 addresses:
 2002:8d38:831a:aaaa:21:2eff:ff00:1965
0001.967 2002:8d38:831a:aaaa:21:2eff:ff00:1965
 fe80::21:2eff:ff00:1965
0001.967 fe80::21:2eff:ff00:1965
0004.082 Packet from TUN of length 48 - write SLIP
0008.092 Packet from TUN of length 48 - write SLIP
...
```

Die Zeile unterhalb von `Server IPv6 addresses:` verrät, dass die IPv6-Adresse des Border-Routers 2002:8d38:831a:aaaa:21:2eff:ff00:1965 lautet. Der Border-Router sollte nun auf Ping-Anfragen reagieren (es werden vier Anfragen (`-c 4`) an die Adresse des Border-Routers gestellt):

```
1 ping6 -c 4 2002:8d38:831a:aaaa:21:2eff:ff00:1965
```

## 3.3 Ping-Anfragen an die Nodes senden

Nach der Inbetriebnahme des Border-Routers (und dem Start des Programms tunslip6), können die Nodes in Betrieb genommen werden. Es vergeht ein Augenblick, bis eine Node

die Nachbarschaft zum Border-Router erkannt hat. Eine Diagnose-LED kann verwendet werden, um dies anzuzeigen (siehe Kap. 4.4.3). Die Nodes können auch vor dem Border-Router in Betrieb genommen werden, jedoch dauert es dann nach der Erfahrung der Autoren länger, bis die Kommunikation hergestellt werden kann.

Um die IP-Adressen der Nodes zu ermitteln, kann die IP-Adresse des Border-Router in einem Webbrowser eingegeben werden. Der Webbrowser muss auf einem Rechner ausgeführt werden, von welchem aus das Subnetz des Border-Routers erreichbar ist. Z. B. derselbe Rechner, welcher auch tunslip6 ausführt. Ausgehend vom Beispiel in Kap. 3.2 lautet die URL dann: `http://[2002:8d38:831a:aaaa:21:2eff:ff00:1965]/`

Alternativ kann der Inhalt des im Border-Router integrierten Miniwebservers auch auf der Kommandozeile mit Tools wie wget abgerufen werden:
`wget 'http://[2002:8d38:831a:aaaa:21:2eff:ff00:1965]/' -O -`

Die Ausgabe des Webbrowsers könnte wiefolgt aussehen:

```
Neighbors

fe80::21:2eff:ff00:389a
fe80::221:2eff:ff00:225d

Routes

2002:8d38:831a:aaaa:21:2eff:ff00:389a/128 (via fe80::21:2eff:ff00:389a) 1257s
2002:8d38:831a:aaaa:221:2eff:ff00:225d/128 (via fe80::221:2eff:ff00:225d) 1192s
```

Im Beispiel sind zwei Nodes mit den IP-Adressen 2002:8d38:831a:aaaa:21:2eff:ff00:389a und 2002:8d38:831a:aaaa:221:2eff:ff00:225d mit dem Border-Router verbunden.

Die Verbindung zu den Nodes lässt sich nun mit Ping überprüfen. Z. B.:

```
1 ping6 -c 4 2002:8d38:831a:aaaa:21:2eff:ff00:389a
```

Übrigens entsprechen die letzten sechs Stellen der IP-Adresse den letzten sechs Stellen der MAC-Adresse der Node. Die MAC-Adressen beginnen stets mit 00. So lautet die MAC-Adresse der ersten Node im Beispiel 00-21-2E-FF-FF-00-38-9A. Die MAC-Adresse einer Node (oder eines Border-Router) ist auf der Rückseite des Funkmoduls deRFmega128-22A00 in kleiner Schrift vermerkt.

Vorraussetzung dafür, dass eine Node oder ein Border-Router seine IPv6-Adresse generieren kann ist, dass die individuelle MAC-Adresse des Funkmoduls im EEPROM des Mikrocontrollers geschrieben wurde (vgl. [7] und [2]).[15]

[15] Bei der gesamten Hardware der Autoren war dieser Schritt nicht notwendig, da er bereits von den Vorbesitzern der Hardware vorgenommen wurde.

## 3.4 CoAP-Anfragen an die Nodes senden

Das Constrained Application Protocol (CoAP) ist eine Art leichtgewichtiges HTTP für Systeme mit stark beschränkten Ressourcen (vgl. [4]). Es wird bspw. benutzt, um Sensorwerte von den Nodes abzurufen.

Um CoAP-Anfragen an eine Node zu stellen, wird ein CoAP-Client benötigt:

```
1 mkdir coap-client
2 cd coap-client
3 wget http://downloads.sourceforge.net/project/libcoap/coap-18/libcoap-4.1.1.tar.
      gz
4 tar xf libcoap-4.1.1.tar.gz
5 autoconf
6 ./configure
7 make
```

In Zeile 1 bis 3 wird ein neues Verzeichnis coap-client erstellt, in welches ein tar.gz-Archiv mit libcoap in der Version 4.1.1[16] herunter geladen wird. In Zeile 4 bis 7 wird das Archiv entpackt und das CoAP-Client-Programm kompiliert. Das Programm befindet sich dann im Unterverzeichnis examples. Mit folgendem Befehl kann eine CoAP-Anfrage an eine Node gestellt werden (Zeile umbrochen):

```
./coap-client -m get coap://[2002:8d38:831a:aaaa:221:2eff:ff00:225d]/.well
-known/core
```

Die Ausgabe sieht inetwa wiefolgt aus:

```
v:1 t:0 tkl:0 c:1 id:35437
</.well-known/core>;ct=40,</testv:1 t:0 tkl:0 c:1 id:35438
/hello>;title="Hello world: ?lenv:1 t:0 tkl:0 c:1 id:35439
=0..";rt="Text",</test/push>;titv:1 t:0 tkl:0 c:1 id:35440
le="Periodic demo";obs
```

[16] Unter `https://sourceforge.net/projects/libcoap/files/` kann der Download-Link zur aktuellen Version abgerufen werden.

# 4 Modifikationen in den Quellen von Contiki 3.0 gegenüber der Original-Quellen

Dieses Kapitel fasst die Änderungen zusammen, welche im Sommersemester 2016 gegenüber der Original-Quellen von Contiki 3.0 gemacht wurden. Am Ende jedes Unterkapitels steht die Referenz auf den entsprechenden Commit und der zugehörige GitHub-Link, über welchen sich die Details der Änderungen übersichtlich angezeigen lassen.

## 4.1 Anlegen der Projekte border-router und er-rest-example

Der Quellcode von Contiki enthält eine Reihe von Beispiel-Projekten, welche sich im Unterverzeichnis examples befinden. Das Projekt examples/ipv6/rpl-border-router dient als Ausgangsbasis für die Software der Border-Router und das Projekt examples/er-rest-example ist die Basis für die Software des Nodes. Zur besseren Strukturierung wurde ein neues Unterverzeichnis project im working tree erstellt. Dort wurde ein symbolischer Link border-router auf examples/ipv6/rpl-border-router erstellt und eine Kopie des Projektes examples/er-rest-example. Mit Hilfe dieser Projekte kann die Verbindung getestet werden. So können Ping-Anfragen an die Node gesendet werden oder Anfragen über das CoAP.
Desweiteren wurden Änderungen vorgenommen, damit sich der Quellcode für die Zielplattform avr-atmega128rfa1 übersetzen lässt und damit der AVR Dragon mit der JTAG-Schnittstelle für die Programmierung verwendet werden kann.

Commit a58e2b6
`https://github.com/HTWDD-SN/contiki/commit/a58e2b6bdb2d6f9ad7aa98413333f34c5e1ff788`

## 4.2 ContikiMac

ContikiMac ist ein sog. Radio-Duty-Cycling-Protokoll, dessen Sinn es ist, Energie zu sparen (vgl. [5]). Im Folgenden werden alle Änderungen beschrieben, welche den Betrieb von ContikiMac mit der Plattform avr-atmega128rfa1 ermöglichen und die Funktion von ContikiMac verbessern.

### 4.2.1 Aktivieren von ContikiMac

Durch Präprozessor-Anweisungen (`#if`, `#elif` und `#endif`) in der Datei platform/avr-atmega128rfa1/contiki-conf.h wird geregelt, ob und welches Radio-Duty-Cycling-Protokoll eingesetzt werden soll. ContikiMac wurde auf diese Weise aktiviert. Desweiteren mussten einige Fehler behoben werden, damit die Software kompiliert.
Mit diesen Änderungen ist ContikiMac zwar aktiviert, die Qualität der Verbindung aber noch sehr schlecht.

Commit fb6b016

`https://github.com/HTWDD-SN/contiki/commit/fb6b0167ebbdea6b999fc2f1a65be3acf4a11a1e`

### 4.2.2 Deaktivieren von Fast-Sleep

Als eine Ursache für die schlechte Verbindungsqualität wurde Fast-Sleep ausgemacht. Das Prinzip von ContikiMac ist es, Energie zu sparen, indem die Funkeinheit so lange wie möglich deaktiviert bleibt. Fast-Sleep ist ein Mechanismus, der es erlaubt, das Funkmodul unter bestimmten Umständen noch schneller wieder zu deaktivieren (vgl. [1]).

Commit 1f7cd85

`https://github.com/HTWDD-SN/contiki/commit/1f7cd8555db329d4e0ea7c0556afa8489222d65f`

### 4.2.3 Verbesserung des CCA-Checks

Von Haus aus nutzt ContikiMac den sog. Received Signal Strength Indicator (RSSI), um zu ermitteln, ob Daten für den Empfang von anderen Geräten gesendet werden. D. h., erst, wenn die Stärke des eingehenden Signals einen definierten Schwellwert überschreitet, wird ein eingehendes Signal detektiert. Wird dieser Schwellwert zu hoch definiert, führt dies zur einschränkung der Reichweite. Wird er zu niedrig definiert, führt jedes Rauschen zur Detektion eines Signals und damit zur insgesamt längeren Aktivzeit der Funkeinheit, was wiederum den Energieverbrauch anhebt.
Daher wurde die Verwendung des Carrier-Sense-Mode implementiert. Dabei handelt es sich um eine Spezialfunktion des ATmega128RFA1, bei welcher versucht wird, ein Signal zu demodulieren. Kann ein Signal demoduliert werden, werden auch tatsächlich verwertbare Daten übertragen. Bei dieser Methode kommt es praktisch zu keiner Einschränkung der Reichweite.

Commit 5de45ce

`https://github.com/HTWDD-SN/contiki/commit/5de45ced705e0a12a7cc91e864af7ba0a6301b80`

### 4.2.4 Workaround für lange Inaktivität der Funkeinheit

Mit Hilfe einer Diagnose-LED (siehe Kap. 4.4.1) wurde festgestellt, dass die Funkeinheit sporadisch und unter ungeklären Umständen viel zu lange deaktiviert bleibt. Die Funktion `static char powercycle(struct rtimer *t, void *ptr)` in der Datei core/net/mac/contikimac/contikimac.c steuert die An- und Auszeiten der Funkeinheit. Das Auskommentieren zweier Code-Zeilen behebt das Problem. Offen bleibt es, eventuelle Nebenwirkungen zu klären.

Um die Debugging-Funktionen von Contiki sinnvoll nutzen zu können, wurde der Takt der CPU außerdem wieder auf 16 MHz angehoben (siehe Kap. 4.3 und Kap. 4.5).

Commit 8ed6d5c

`https://github.com/HTWDD-SN/contiki/commit/8ed6d5c5a14dcd2fa96fe51484c6f23284f9da2c`

## 4.3 Prescaler für CPU-Takt

Wenn die Konstante *SET_CLOCK_PRESCALER_REGISTER* gesetzt ist, wird der Wert des Registers CLKPR in der Low-Level-Initialisierungs-Routine von Contiki auf den Wert dieser Konstante gesetzt. Damit lassen sich verschiedene Taktteiler aktivieren, welche den Takt der Taktquelle durch einen bestimmten Faktor teilen, bevor dieser an die CPU weitergeleitet wird. Die Autoren verwenden den 16 MHz Referenz-Taktgeber der Funkeinheit (siehe Kap. 2.3). Um die CPU dennoch mit 8 MHz zu takten, wird der Takt durch zwei geteilt.[17] Dazu wurde die Konstante *SET_CLOCK_PRESCALER_REGISTER* mit dem Wert 1 definiert, um das Register CLKPR auf eben diesen Wert zu setzen. Für die Bedeutung der Werte im Register CLKPR siehe [11] und im Datenblatt des Atmega128RFA1 auf S. 157 (Datenblatt-Version vom September 2014).

Commit 5de45ce

`https://github.com/HTWDD-SN/contiki/commit/5de45ced705e0a12a7cc91e864af7ba0a6301b80`

## 4.4 LEDs als Indikatoren

Das Bord deRFnode, welches für die Nodes verwendet wird, verfügt über drei LEDs mit den Bezeichnungen D1 bis D3 (siehe Abb. 4.2), welche zu Diagnosezwecken eingesetzt werden können.

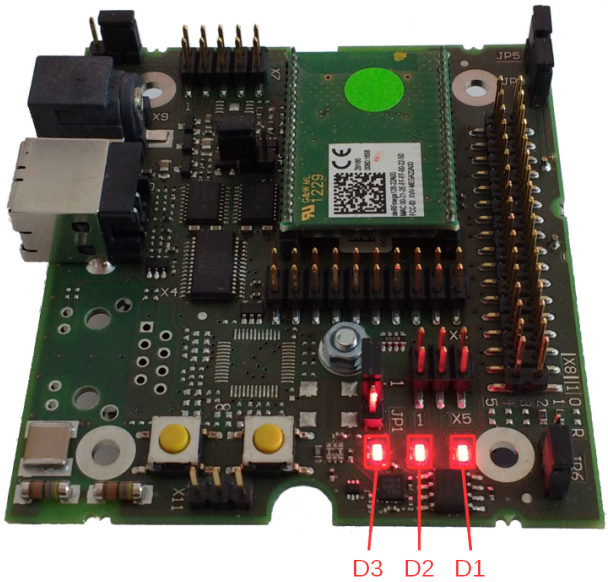


Abb. 4.2: Die LEDs des Bords deRFnode von dresden elektronik

Die LEDs werden via Definition bestimmter Konstanten aktiviert. Diese Konstanten sollten nur im Software-Projekt für die Nodes definiert werden, also z. B. im Projekt er-rest-

[17] Ist der Takt der CPU geringer, verbraucht der Mikrocontroller im Allgemeinen weniger Energie.

example (für den Border-Router ergäbe dies keinen Sinn).

Wenn möglichst viel Energie gespart werden soll, sollten die LEDs nicht verwendet werden. Die Implementierungen für alle drei Indikator-LEDs wurden gemeinsam mit dem Commit 1f98d95 eingereicht.
`https://github.com/HTWDD-SN/contiki/commit/1f98d959fdd85b5b78c0dc74e7e498e80bf45056`

### 4.4.1 LED zum Anzeigen des Status der Funkeinheit

Die LED D1 kann verwendet werden, um den Status der Funkeinheit anzuzeigen, d. h., ob diese aktiviert oder deaktiviert ist. Wenn ContikiMac aktiviert ist, blinkt die LED im schnellen Rhythmus. Dies begründet sich mit der Funktionsweise von ContikiMac (siehe [5]).
Um diese Funktion zu aktivieren, muss die Konstante `RADIO_INDICATOR_LED_ON_PORT_G5` mit einem beliebiegen Wert definiert sein. Dies wurde für die Nodes über die Zeile `#define RADIO_INDICATOR_LED_ON_PORT_G5 1` in der Datei project/er-rest-example/project-conf.h erledigt.

### 4.4.2 LED zum Anzeigen eingegangener Ping-Anfragen

Die LED D2 kann verwendet werden, um eingehende Ping-Requests anzuzeigen, d. h., wenn ein Ping-Request auf die IP-Adresse der Node abgesetzt wurde. Die LED leuchtet nur für eine sehr kurze Zeit auf und nur bei Ping-Requests, nicht beim Antworten (Ping-Response).
Um diese Funktion zu nutzen, muss die Konstante `INCOMING_PING6_REQUESTS_INDICATOR_LED_ON_PORT_E3` definiert sein. Auch diese Funktion wurde für die Nodes, analog zu Kap. 4.4.1, im Projekt er-rest-example aktiviert.

### 4.4.3 LED zum Anzeigen einer gefundenen Nachbarschaft

Die LED D3 kann genutzt werden, um eine gefundene Nachbarschaft anzuzeigen. Die LED beginnt zu leuchten, sobald die Node mit Strom versorgt wird und erlischt, wenn eine Nachbarschaft gefunden wurde. Dies kann z. B. ein Border-Router sein, welcher sich in Reichweite befindet. Erst, wenn eine Nachbarschaft gefunden wurden (LED ist aus), ist die Node über ihre IP-Adresse erreichbar.
Um diese Funktion zu nutzen, muss die Konstante `NEIGHBOR_FOUND_INDICATOR_LED_ON_PORT_E4` definiert sein. Auch dies wurde wieder, analog zu Kap. 4.4.1 und Kap. 4.4.2, für das Projekt er-rest-example getan.

## 4.5 Verwendung der USB-Schnittstelle des Boards deRFnode für das Debugging

Das Board deRFnode verfügt über eine USB-Schnittstelle, welche verschiedene Funktionen erfüllt (siehe Datenblatt zur deRFnode). Sie kann verwendet werden, um Debug-Meldungen von Contiki mit einem Computer zu empfangen (und versorgt die Node dabei gleichzeitig mit Strom). Contiki nutzt normalerweise eine der beiden UART-Schnittstellen des Atmega128RFA1, um Debug-Meldungen auszugeben. Mit Hilfe von Quellcode von dresden elektronik, können die Debug-Meldungen über die USB-Schnittstelle der deRFnode umgeleitet werden. Der Code von dresden elektronik befindet sich in den neu hinzugefügten Dateien usb-de-rfnode.c und usb.h (beide im Unterverzeichnis platform/avr-atmega128rfa1). Ferner werden die Debug-Meldungen nur über diese Schnittstelle umgeleitet, wenn die Stromversorgung der Node über USB erfolgt. Um dies zu ermitteln, wurde weiterer Quell-Code in Form der Dateien adc.c und adc.h (beide im Unterverzeichnis platform/avr-atmega128rfa1/dev) hinzugefügt. Alle vier Dateien wurden aus dem Branch sn-2.7 übernommen.

Ist diese Funktion aktiviert, müssen die Debug-Meldungen ausgelesen werden, wenn die Node über USB an einem Computer angeschlossen ist, da das Programm solange blockiert wird, bis die Meldungen ausgelesen wurden.

Um die Funktion nutzen zu können, müssen die beiden Konstanten `DE_RF_NODE` und `DE_RF_NODE_USB_DEBUG` definiert sein. Dies macht nur Sinn für Software-Projekte, welche das Board deRFnode nutzen. Die Konstanten wurden daher in der Datei project/er-rest-example/project-conf.h definiert. Bei der Ausgabe umfangreicher Debug-Meldungen kann es vorkommen, dass die Node eingehende Daten nicht mehr schnell genug verarbeiten kann. Dies führte bspw. zu hohen Verlusten bei Ping-Anfragen. Daher wurde der CPU-Takt wieder auf 16 MHz angehoben (siehe Commit aus Kap. 4.2.4).

Commit e1ce544
`https://github.com/HTWDD-SN/contiki/commit/e1ce544e1e11849307c602761d4b56af27be3e71`

### Debug-Meldungen empfangen

Ist das Board deRFnode über USB mit einem Computer verbunden, können die Debug-Meldungen über folgende Schritte empfangen werden (vgl. [6]):

```
1 modprobe ftdi_sio
2 echo 1cf1 001d > /sys/bus/usb-serial/drivers/ftdi_sio/new_id
3 cat /dev/ttyUSB0
```

Die Befehle werden als root ausgeführt. In Zeile 1 wird das Kernel-Modul ftdi_sio geladen. Anschließend wird dem Kernel-Modul die Vendor- und Device-ID des Ftdi-Chips bekannt

gemacht (Zeile 2). Die IDs lassen sich mit dem Befehl `dmesg` oder `lsusb` herausfinden.[18] Es wird eine neue Gerätedatei /dev/ttyUSBn erstellt, wobei n für die nächste freie Nummer steht. Unter der Annahme, dass die Gerätedatei die Bezeichnung /dev/ttyUSB0 trägt, werden die Debug-Meldungen schließlich ausgelesen (Zeile 3). Der Befehl kann mit Strg+C abgebrochen werden.

### Debug-Meldungen senden

Debug-Meldungen können für verschiedene Teile von Contiki im Quellcode aktiviert werden. Dazu muss die Zeile `#define DEBUG 0` in der entsprechenden Quellcode-Datei ausgemacht werden und zu `#define DEBUG 1` geändert werden. Eine solche Code-Zeile existiert bspw. in der Hauptdatei er-example-server.c für das Projekt er-rest-example respektive in der Datei border-router.c für das Projekt border-router. Weitere Beispiele sind die Dateien core/net/ipv6/uip-icmp6.c und core/net/mac/contikimac/contikimac.c. Sind Debug-Meldungen in ersterer Datei aktiviert, kann das Empfangen und Antworten auf Ping-Requests mitverfolgt werden. In letzterer Datei können ContikiMac-spezifische Debug-Meldungen aktiviert werden.

Eigene Debug-Meldungen können mit dem Makro PRINTF und ähnlich lautende Makros[19] ausgegeben werden. Es sei ein genauerer Blick in den Quellcode empfohlen, um die Funktionsweise dieser Makros zu verstehen.

[18] Die deRFnode Boards der Autoren hatten durchweg dieselbe ID, nämlich wie im Beispiel 1cf1 001d.
[19] Zwei weitere Varianten des Makros lauten z. B. PRINTDEBUG und PRINTD.

# 5 Test- und Messergebnisse

## 5.1 Ermittlung der Verbindungsqualität mittels Ping-Tests

Ein Großteil des Semestern wurde damit zugebracht, die Verbindungsqualität bei aktiviertem ContikiMac auf ein akzeptables Niveau zu bringen. Dabei kam häufig das Netzwerkdiagnoseprogramm Ping zum Einsatz. Die Verlustrate[20] wurde dabei als Maß für die Verbindungsqualität herangezogen.

Belief sich die Verlustrate ohne ContikiMac und mit einem Abstand von mehereren Metern und Hindernissen zwischen Border-Router und Node, auf $< 10\%$, lag diese mit aktiviertem ContikiMac und nur wenigen cm Abstand ohne Hindernisse zwischen Border-Router und Node, bei ca. 50%. Die Verbindungsqualität unter Contiki 3.0 mit aktiviertem ContikiMac konnte durch folgende Maßnahmen verbessert werden:

- Verwenden des Referenztaktgebers des Funkmoduls (Transceiver Crystal Oscillator) als Taktgeber für die CPU (siehe Kap. 2.3).
- Der tatsächliche CPU-Takt muss mit der Konfiguration von Contiki übereinstimmen (siehe Kap. 2.3 und Kap. 4.3).
- Die Änderungen aus Kap. 4.2.

Die im Folgendem dargelegten Ergebnisse wurden mit dem Softwarestand des Commits 8ed6d5c aus den GitHub-Repositorys der HTW Dresden erzielt. Der Versuch mit demselben Softwarestand kann nachvollzogen werden, in dem man den Quellcode wie in Kap. 2.2 beschrieben beschafft und den Commit mittels des Kommandos `git checkout 8ed6d5c` aus-checkt. Die Software muss anschließend kompiliert und auf die Hardware geflasht werden.

### Vorgehen beim Testen

Für den Test kamen ein Border-Router und zwei Nodes zum Einsatz. Der Border-Router war an einen Computer (Raspberry Pi 2 mit Raspbian als Beriebsystem) angeschlossen. Die Nodes befanden sich dicht nebeneinander und in einem Abstand von fünf Metern (Luftlinie und ohne Hindernisse) zueinander. Alle fünf Minuten wurden, zuerst zu der eine Node, dann zu der andere Node, 100 Ping-Anfragen mit der Standardpaketgröße von 64 Bytes gesented. Die Verlustrate wurde für die Auswertung nach jedem Ping-Lauf und getrennt für beide Nodes, in einer Datei protokolliert. Der Test lief insgesamt 16,26 Tage.

[20] Die Verlustrate ist der prozentuale Anteil aller Ping-Anfragen von allen gesendeten Ping-Anfragen, auf die es gar keine Ping-Antwort gab, bzw. die Ping-Antwort innerhalb eines Timeouts nicht eintraf.

### Testergebnisse

Tabelle 5.1 fasst die Testergebnisse zusammen. Da Ausreißer die Testergebnisse stark beeinflussen, wurde neben der durchschnittlichen Verlustrate auch der Median und Histogramme herangezogen (siehe Abb. 5.3 und Abb. 5.4). Wie man erkennt, lag die Verlustrate der Einzeltests in den aller meisten Fällen bei 0%.

| **Gesamtdauer des Tests** | 16,26 Tage | |
|---|---|---|
| **Zeitlicher Abstand der Einzeltests** | 5 Minuten | |
| **Ping-Anfragen je Einzeltest** | 100 Pakete zu je 64 Bytes | |
| **Abstand der Nodes zum Border-Router** | 5 Meter (Luftlinie, keine Hindernisse) | |
| | **Node 1** | **Node 2** |
| **Verlustrate Durchschnitt** | 2,47% | 3,45% |
| **Verlustrate Median** | 0% | 0% |

Tabelle 5.1: Ergebnisse des Ping-Tests

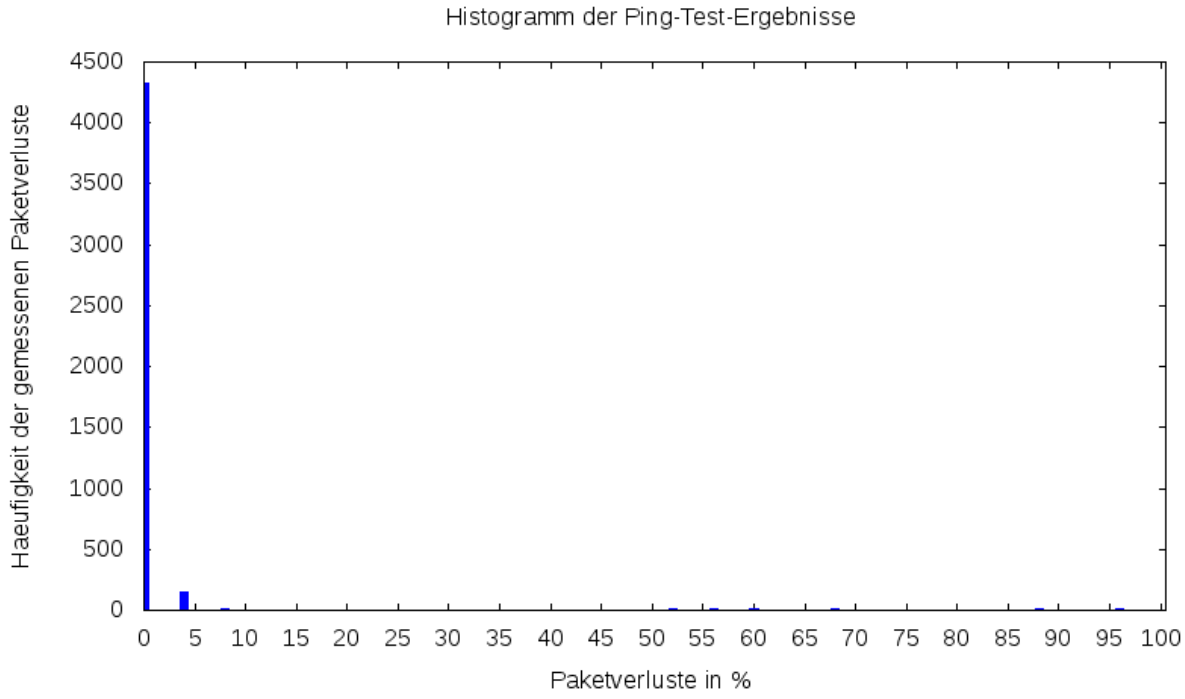


Abb. 5.3: Histogramm des Ping-Tests der Node 1

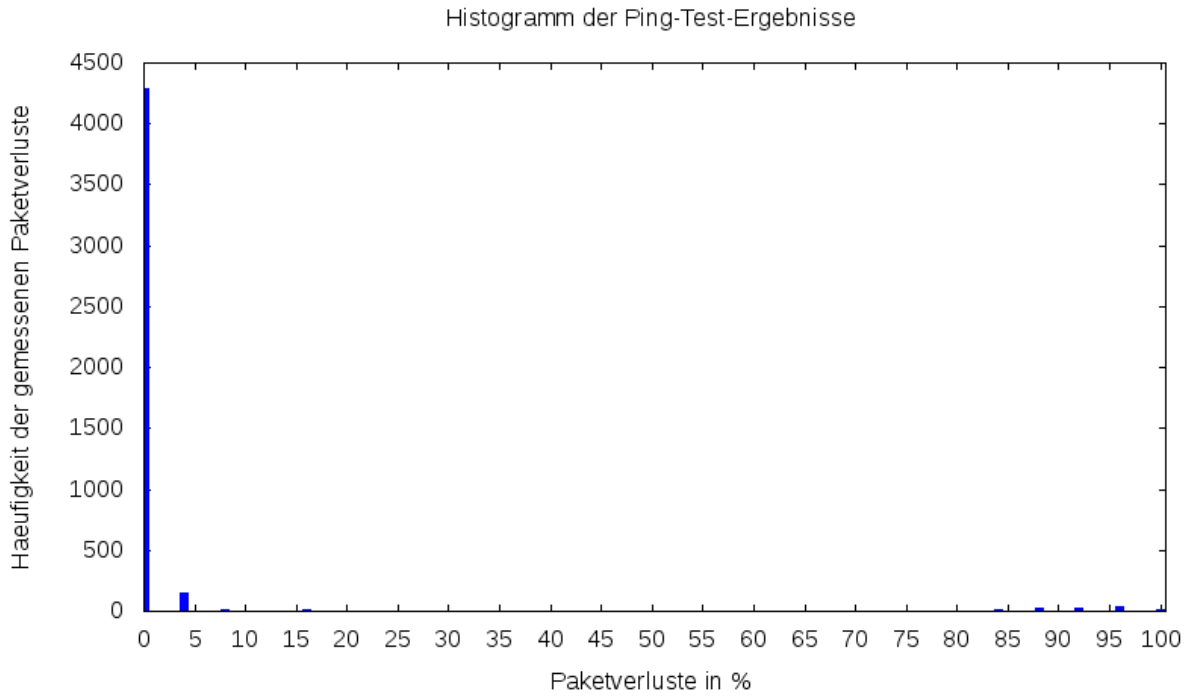


Abb. 5.4: Histogramm des Ping-Tests der Node 2

## 5.2 Reichweite

Die Reichweite der Funkverbindung zwischen Border-Router und Node bei Sichtkontakt wurde in einem Gebäude und im Freien getestet. Während des Tests wurden fortlaufend Ping-Anfragen der Standardpaketgröße von 64 Bytes an die Node gesendet. Wenn die regelmäßigen Ping-Antworten der Node ausblieben (sporadische Paketverluste wurden ignoriert), wurde der Test abgebrochen. ContikiMac war wärend der Tests aktiviert.

Es hat sich gezeigt, dass die Ausrichtung der Geräte aufeinander eine große Rolle spielt. Vermutlich, weil sich die ausgesendeten Funktwellen der verbauten Chip-Antennen des Moduls deRFmega128-22A00 nicht gleichmäßig ausbreiten. Die beste Verbindung wurde bei der Ausrichtung der Hardware gemäß Abb. 5.5 erzielt.

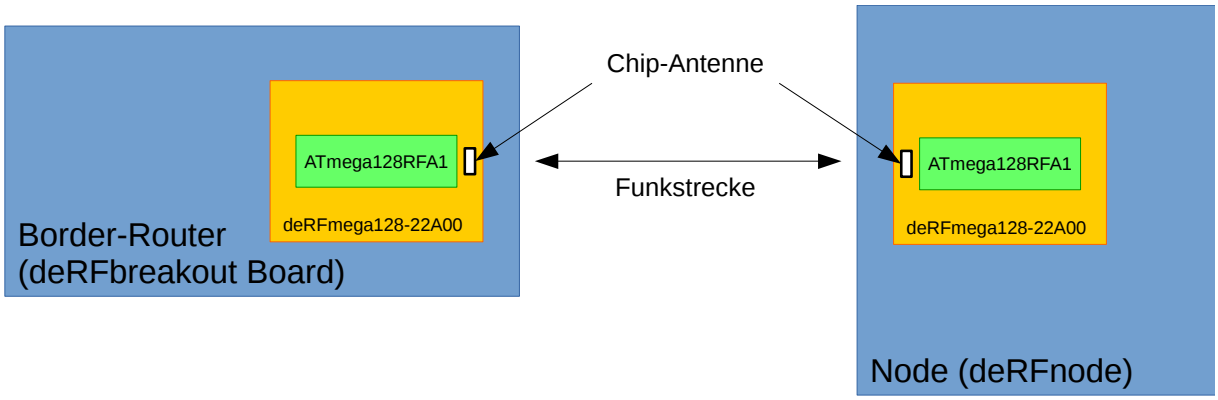


Abb. 5.5: Ausrichtung der Hardware aufeinder für eine optimale Verbindung

### Reichweite im Gebäude

Es wurde in einem langen Gang eines Stahlbetongebäudes mit gut ausgebauter WLAN-Infrastruktur und vielen WLAN-Clients getestet. Der WLAN-Funkverkehr[21] und die Begünstigung von Reflexion von Funkwellen im Gebäude, wirkten sich negativ auf die Funkverbindung zwischen Border-Router und Node aus. Unter Zuhilfenahme eines Gebäudeplans, wurde eine Reichweite von 50 Metern ermittelt. Das Ergebnis ist damit deutlich besser, als ältere Messungen mit Contiki 2.7 und aktiviertem ContikiMac (vgl. [5]).

### Reichweite im Freien

Es wurde in einer Straße (Andreas-Schubert-Straße) nahe der HTW Dresden getestet. Die Entfernung wurde dabei mit einem Smartphone und GPS gemessen. Die Genauigkeit lag daher bei ±5 Metern. Die gemessene Entferung beträgt ca. 95 Meter.

## 5.3 Stromverbrauch

Der Stromverbrauch einer Node im Idealfall, d. h. ohne, dass Kommunikation stattfindet, wurde gemessen, um die Lebensdauer im Akku-Betrieb berechnen zu können. Dazu wurde ein Shunt-Widerstand zwischen Spannungsquelle und der Node eingefügt. Der Spannungsabfall am Shunt-Widerstand wurde mit dem Oszilloskop PicoScope 5203 des Herstellers Pico Technology Ltd. gemessen. Aus dem bekannten Wert des Shunt-Widerstandes und dem gemessenen Spannungsabfall an diesem, kann die Stromstärke mit Hilfe des Ohmschen Gesetzes berechnet werden. Abb. 5.6 zeigt den Versuchsaufbau, wobei die Node mit RLOAD bezeichnet ist.

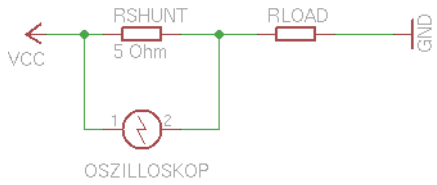


Abb. 5.6: Schaltplan des Versuchsaufbaus zur Bestimmung des Stromverbrauchs

Für die sog. Timebase wurde in der Software des Oszilloskops 100 ms/div ausgewählt. Da es zehn Divider gibt, dauert eine Messung eine Sekunde. Abb. 5.7 zeigt exemplarisch eine solche Messung. An den Spitzen erkennt man deutlich die acht CCA-Phasen pro Sekunde.

[21] IEEE 802.15.4 und WLAN nutzen beide das 2,4 GHz-Band.

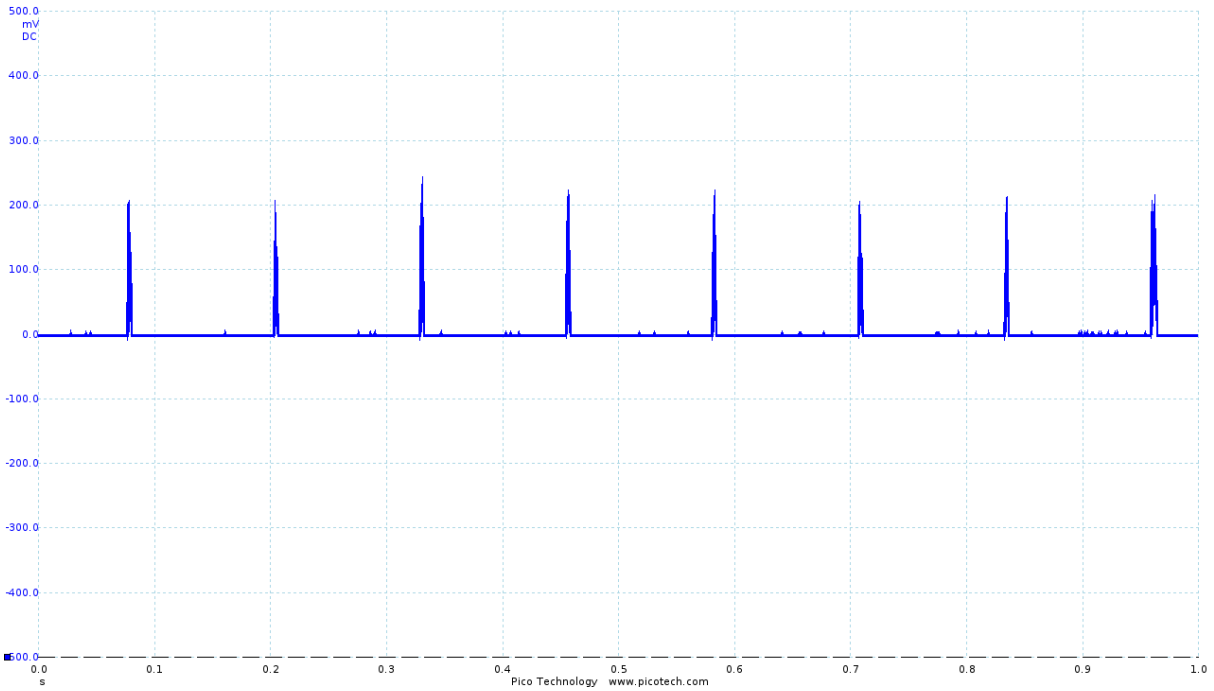


Abb. 5.7: Exemplarisches Messergebnis des Spannungsabfalls am Shunt-Widerstand

Das Oszilloskop führt die Messungen fortlaufend durch und speichert die letzten 32 Messungen in einem Puffer. Insgesamt wurde der Stromverbrauch somit über einen Zeitraum von 32 Sekunden aufgezeichnet. Die Messergebnisse wurde als CSV-Datei exportiert, jede Messung in eine separate Datei. Eine Datei enthält somit den Spannungsverlauf über eine Sekunde mit insgesamt 1000004 Messpunkten (einer je Mikrosekunde). Der Inhalt einer CSV-Datei sieht etwa wiefolgt aus:

```
Time,Channel A
(s),(mV)

0.00000000,−3.93700800
0.00000100,0.00000000
0.00000200,−3.93700800
0.00000300,0.00000000
0.00000400,−3.93700800
0.00000500,0.00000000
0.00000600,0.00000000
0.00000700,0.00000000
0.00000800,0.00000000
0.00000900,0.00000000
0.00001000,−3.93700800
0.00001100,0.00000000
0.00001200,−3.93700800
...
```

Aus jeder einzelnen Datei wurde nun der mittlere Stromverbrauch mit Hilfe des Programms in Anlage 1 numerisch bestimmt. Um den Stromverbrauch schließlich final zu berechnen, wurde der Mittelwert über die 32 Teilergebnisse gebildet.
Sieht man sich die CSV-Daten genauer an, stellt man fest, dass der Spannungsabfall während der Ruhe-Phasen, also dann, wenn keine Spitzen auftreten, meistens $\leq 0$ mV beträgt.

Das Datenblatt des Mikrocontrollers verrät jedoch, dass der Mikrocontroller mindestens 250 $\mu$A (im Deep-Sleep-Modus) benötigt. Das verwendete Oszilloskop ist hier offenbar nicht genau genug. Daher nimmt das Programm aus Anlage 1 mindestens 250 $\mu$A an.

Alle der folgend Beschriebenen Messungen wurden mit dem Softwarestand von Contiki 3.0 vom 27. Juli 2016 (Commit 8ed6d5c) aus der GitHub-Quelle der HTW Dresden durchgeführt.

### Analytische Berechnung des Stromverbrauchs bei 16 MHz

In vergangenen Semestern wurde der Stromverbrauch des Funkmoduls deRFmega128-22A00 bereits analytisch bestimmt (siehe [10] und Forschungsbericht vom Sommersemesters 2015 von Andreas Salwasser[22]). Zum Vergleich soll die analytische Berechnung für das Funkmodul bei 16 MHz wiederholt werden.

Hierfür wurde der Jumper JP5 des Bords deRFnode entfernt und der Shunt-Widerstand mit Oszilloskop angeschlossen. Der Stromverbrauch der Diagnose-LEDs (siehe Kap. 4.4) sollte die Messung nicht beeinflusst haben, da die LEDs über ihre Kathoden mit dem Mikrocontroller verbunden sind. D. h., der Strom fließt von der Spannungsquelle über die LEDs durch den Mikrocontroller hindurch.

Betrachtet man Abb. 5.7, erkennt man ein Muster, welches sich periodisch wiederholt. Da das Muster acht mal in der Sekunde auftritt, beträgt die Periodendauer 125 ms. Eine Periode besteht aus einer langen Ruhephase und einer Spitze, der CCA-Phase (siehe Abb. 5.8).

Zunächst wird die mittlere Spannung der CCA-Phase berechnet:

$$\frac{\begin{array}{l}(1,327 \cdot 7,833 + 0,154 \cdot 57,81 + 0,067 \cdot 78,83 + 0,268 \cdot 114,3 + \\ \quad 0,637 \cdot 63,06 + 0,04 \cdot 82,77 + 0,322 \cdot 114,3 + 0,637 \cdot 63,06) ms \cdot mV\end{array}}{3{,}452 ms} = 50,88789 mV$$

Daraus lässt sich dann mit dem Wert des Shunt-Widerstandes der mittlere Strom der CCA-Phase bestimmen: $\frac{50{,}88789 mV}{5\Omega} = 10,17758 mA$

Nimmt man für die Ruhe-Phasen einen Strom von 250 $\mu$A an, ergibt sich für eine komplette Periode: $\frac{10{,}17758 mA \cdot 3{,}452 ms + 0{,}00025 mA \cdot (125 ms - 3{,}452 ms)}{125 ms} = 0,28131 mA$

[22] http://www2.htw-dresden.de/~wiki_sn/index.php5/Datei:SS15_Bericht_Andreas_Salwasser.pdf

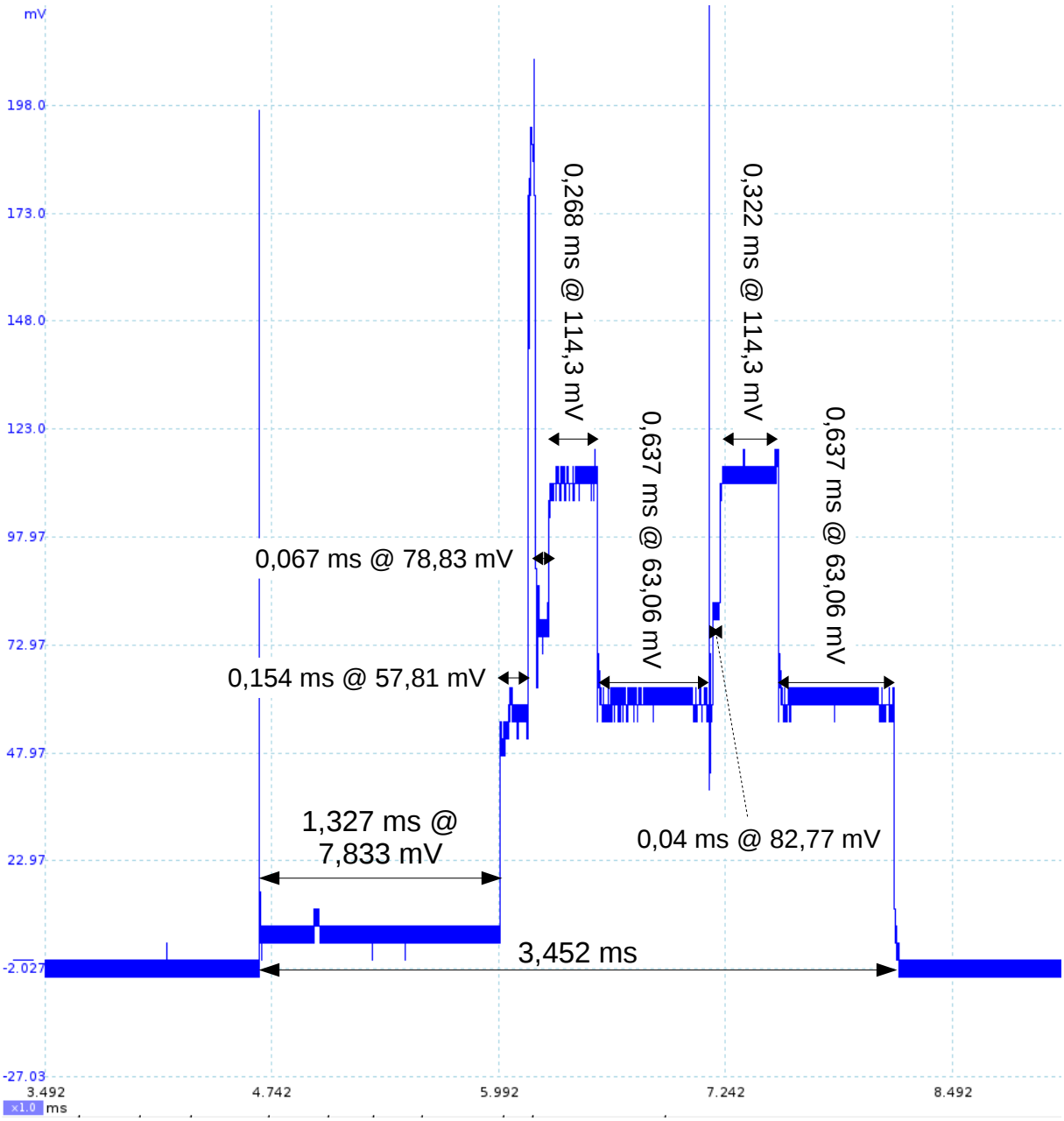


Abb. 5.8: Messung einer CCA-Phase bei 16 MHz

### Stromverbrauch des Funkmoduls deRFmega128-22A00 bei 16 MHz

Der Versuchsaufbau ist derselbe wie für die analytische Berechnung. Der Stromverbrauch wird diesmal jedoch numerisch berechnet, wie zu Begin des Kapitels beschrieben.

### Stromverbrauch des Funkmoduls deRFmega128-22A00 bei 8 MHz

Der Versuchsaufbau ist wieder derselbe wie oben für 16 MHz beschrieben. Um einen CPU-Takt von 8 MHz zu erreichen, wurde der Taktteiler über die Zeile `#define SET_CLOCK_PRESCALER_REGISTER 1` in der Datei platform/avr-atmega128rfa1/contiki-conf.h aktiviert und die Zeile `CONTIKI_PLAT_DEFS = -DF_CPU=16000000UL -DAUTO_CRC_PADDING=2` in der Datei platform/avr-atmega128rfa1/Makefile.avr-atmega128rfa1 geändert in `CONTIKI_`

`PLAT_DEFS = -DF_CPU=8000000UL -DAUTO_CRC_PADDING=2` (vgl. Kap. 4.3). Anschließend wurde das Projekt er-rest-example neu kompiliert und auf den Mikrocontroller geflasht.

### Stromverbrauch der gesamten Node ohne Diagnose-LEDs

Beim oben beschriebenen Vorgehen wird der Stromverbrauch des Funkmoduls deRFmega 128-22A00 ermittelt, nicht aber der gesamten Node. D. h., der Stromverbrauch des Bords deRFnode wird außer Acht gelassen. Um den Stromverbrauch der gesamten Node bei 8 und 16 MHz zu ermitteln, wurde ein Netzteil mit einer Spannung von vier Volt[23] an das Batteriefach des Bords angeschlossen. Zwischen Netzteil und Batteriefach befand sich der Shunt-Widerstand. Die Diagnose-LEDs wurden vor den Messungen in der Datei er-rest-example/project-conf.h deaktiviert (vgl. Kap. 4.4).

### Stromverbrauch der gesamten Node mit Diagnose-LEDs

Oben beschriebene Messungen der gesamten Node wurden bei aktivierter LED für den Status der Funkeinheit (siehe Kap. 4.4.1) wiederholt. Diese Diagnose-LED wurde als einzige LED wieder aktiviert, da es die einzige LED ist, welche im Ruhebetrieb (also ohne stattfindende Kommunikation) aufleuchtet.

### Ergebnis

Tabelle 5.2 auf Seite 28 fasst die Ergebnisse, samt der Einzelergebnisse aus den 32 CSV-Dateien, zusammen. Am Ende der Tabelle (Zeile Avg.) steht jeweils der gemittelte Stromverbrauch für die unterschiedlichen Konfiguarionen.

Was auffällt ist, dass die Werte stark vom analytisch berechneten Wert abweichen. Dies liegt darin begründet, dass der Mikrocontroller ungefähr alle zwei Sekunden während zwei CCA-Phasen nicht in den Schlafmodus wechselt.[24] Ein Beispiel für eine solche Messung ist in Anlage 2 zu finden. Ungefähr jede zweite Messung kommt dem analytisch bestimmten Wert jedoch sehr nahe.
Wie zu erwarten, ist bei den Messungen mit 16 MHz der Stromverbrauch des Funkmoduls geringer, als bei der Messung der gesamten Node und dieser Wert wiederum geringer, als bei der Messung der gesamten Node mit aktivierten Diagnose LEDs. Bei 8 MHz ist der Stromverbrauch für die gesamte Node mit Diagnose-LEDs, entgegen der Erwartungen, am geringsten. Offenbar wurden bei diesem Messlauf besonders viele günstige Werte aufgezeichnet.
Außerdem fällt auf, dass der Stromverbrauch bei 16 MHz geringer ist, als bei 8 MHz. Dies könnte daran liegen, dass Berechnungen bei 16 MHz schneller abgeschlossen sind und der Mikrokontroller dadurch früher wieder in den Schlafmodus wechseln kann. Möglicherwei-

[23] Das entspricht ungefährt der Spannung von drei voll geladenen AA-Akkus.
[24] Das Verhalten tritt auch dann auf, wenn die Änderungen aus Kap. 4.2.4 rückgängig gemacht werden.

se liegt dies aber auch an dem Stromverbrauch des Prescalers, welcher aktiviert werden muss, um den Takt der 16 MHz Taktquelle zu teilen.

| | **nur Funkmodul (Messung über JP5)** | | **gesamte Node (Messung über Batteriefach) ohne Diagnose-LEDs** | | **mit Diagnose-LEDs** | |
|---|---|---|---|---|---|---|
| **Nr.** | **8 MHz** | **16 MHz** | **8 MHz** | **16 MHz** | **8 MHz** | **16 MHz** |
| 1 | 1,71510 | 0,31226 | 0,40232 | 0,34584 | 0,42180 | 0,36064 |
| 2 | 0,37476 | 1,84651 | 0,40294 | 1,95975 | 1,83514 | 1,95912 |
| 3 | 1,71691 | 0,31216 | 1,81653 | 0,34679 | 0,37598 | 0,36108 |
| 4 | 0,37463 | 1,81252 | 0,40311 | 1,98935 | 1,83431 | 1,96203 |
| 5 | 1,71283 | 0,31355 | 1,84893 | 0,30681 | 0,42148 | 0,36015 |
| 6 | 0,37380 | 1,84316 | 0,40204 | 0,34712 | 1,83589 | 1,95897 |
| 7 | 1,66876 | 0,31203 | 1,84372 | 1,48510 | 0,42040 | 0,36113 |
| 8 | 0,37281 | 1,84569 | 0,40228 | 0,27384 | 1,75564 | 1,96047 |
| 9 | 1,68341 | 0,31278 | 1,84557 | 0,70465 | 0,37606 | 1,53078 |
| 10 | 0,33137 | 1,82251 | 0,40078 | 0,34667 | 1,83188 | 0,32068 |
| 11 | 1,70919 | 0,31333 | 1,83472 | 1,99219 | 0,42119 | 0,74694 |
| 12 | 0,37203 | 1,84611 | 0,40165 | 0,34702 | 1,83244 | 0,32851 |
| 13 | 1,70946 | 0,31240 | 0,35523 | 1,96848 | 0,41855 | 1,95555 |
| 14 | 0,98678 | 1,84312 | 1,84715 | 0,34844 | 1,82617 | 0,36218 |
| 15 | 0,40006 | 0,31200 | 0,40128 | 1,98803 | 0,37167 | 1,96276 |
| 16 | 1,71261 | 1,84460 | 1,84446 | 0,34533 | 1,77776 | 0,36265 |
| 17 | 0,37322 | 0,31290 | 0,40065 | 1,93500 | 1,59568 | 1,95612 |
| 18 | 1,70927 | 1,12423 | 1,81529 | 0,34664 | 0,37465 | 0,36219 |
| 19 | 0,37007 | 0,31269 | 0,40253 | 1,99184 | 0,37221 | 1,95355 |
| 20 | 1,70586 | 1,84653 | 1,44735 | 0,34589 | 1,44112 | 0,36022 |
| 21 | 0,37132 | 0,31271 | 0,37896 | 1,99295 | 0,41892 | 1,96342 |
| 22 | 1,71187 | 0,31345 | 1,84332 | 0,34544 | 0,37384 | 1,96333 |
| 23 | 0,37231 | 1,84728 | 0,40262 | 1,99209 | 1,79826 | 0,36411 |
| 24 | 1,67355 | 0,31253 | 1,80159 | 0,34725 | 0,41964 | 1,69274 |
| 25 | 0,37114 | 1,81978 | 0,40134 | 1,96147 | 1,82779 | 0,36343 |
| 26 | 1,70518 | 0,31201 | 1,80903 | 0,34480 | 0,42235 | 0,86964 |
| 27 | 0,37035 | 1,84442 | 0,40021 | 1,99029 | 1,80829 | 0,36043 |
| 28 | 1,70081 | 0,31217 | 1,84149 | 0,62997 | 0,41998 | 1,95755 |
| 29 | 0,32901 | 1,84482 | 0,40147 | 0,78279 | 0,71301 | 0,36056 |
| 30 | 1,70392 | 0,75592 | 1,84463 | 0,34512 | 0,41831 | 1,95831 |
| 31 | 0,32841 | 0,87289 | 0,40112 | 0,33650 | 1,63684 | 0,36070 |
| 32 | 0,64919 | 0,41768 | 1,77753 | 1,98676 | 0,41911 | 1,96184 |
| Avg | 1,02062 | 0,99271 | 1,05693 | 1,02407 | 1,00676 | 1,11443 |

Tabelle 5.2: Ergebnisse der Messungen des Stromverbrauchs der Nodes in mA

Nachfolgend wird die Betriebsdauer der gesamten Node ohne Diagnose-LEDs für 8 MHz und 16 MHz berechnet, wenn die Akkus eine Kapazität von 1900 mAh aufweisen.

Bei 8 MHz: $\frac{1900mAh}{1{,}05693mA} = 1797,65926h = 74,90247$ Tagen

Bei 16 MHz: $\frac{1900mAh}{1{,}02407mA} = 1855,34192h = 77,30591$ Tagen

### Experiment mit einem 150 mAh Akku

Um die Ergebnisse der Berechnungen zu verifizieren, wurde ein 150 mAh Akku mit einer Nennspannung von 3,7 V am Batteriefach der Node angeschlossen. Die Node lief mit einem CPU-Takt von 16 MHz und die Diagnose-LEDs wurden für den Versuch deaktiviert. Die Anordnung von Border-Router und Node war dieselbe wie in Kap. 5.1 beschrieben. Es wurde alle 15 Minuten ein Ping-Lauf mit drei Ping-Anfragen an die Node gestartet und das Ergebnis zur Auswertung protokolliert. Das Experiment wurde abgebrochen, als vortlaufend 100% Paket-Verluste verzeichnet wurden.[25] Bei dieser Konfiguration hat die Node eine erwartete Betriebsdauer von $\frac{150mAh}{1{,}02407mA} = 146,47436h = 6,1031$ Tagen. Tabelle 5.3 fasst das Ergebnis zusammen.

| | |
|---|---|
| Erwartete Betriebsdauer | 6,1031 Tagen |
| Tatsächliche Betriebsdauer | 6,1042 Tagen |

Tabelle 5.3: Betriebsdauer mit einem 150 mAh Akku

Wie man sieht, ist die tatsächliche Betriebsdauer sehr nah an der errechneten. Entgegen der Erwartungen ist die tatsächliche Betriebsdauer sogar geringfügig höher. Da Kommunikation statt fand (Ping-Anfragen alle 15 Minuten), wäre das Gegenteil zu erwarten gewesen. Eine mögliche Erklärung dafür ist, dass die reale Kapazität des, noch recht jungen Akkus, etwas höher ist, als angegeben.

[25] Gelegentlich blieben ein bis zwei Ping-Antworten der Node aus. Bereichts mit dem ersten Auftreten von 100% Paket-Verlust setzte sich dies fort und das Experiment wurde beendet.

# Literatur


[1] Adam Dunkels: *The ContikiMAC Radio Duty Cycling Protocol*. Report. 2011.

[2] Sensornetze Wiki der HTW Dresden: *Ändern der MAC/IPv6 Addresse per Contiki*. URL: http://www2.htw-dresden.de/~wiki_sn/index.php5/%C3%84ndern_der_MAC/IPv6_Addresse_per_Contiki (besucht am 16. 08. 2016).

[3] Sensornetze Wiki der HTW Dresden: *Avr-dragon ohne root*. URL: http://www2.htw-dresden.de/~wiki_sn/index.php5/Avr-dragon_ohne_root (besucht am 04. 08. 2016).

[4] Sensornetze Wiki der HTW Dresden: *CoAP*. URL: http://www2.htw-dresden.de/~wiki_sn/index.php5/CoAP (besucht am 16. 08. 2016).

[5] Sensornetze Wiki der HTW Dresden: *Contiki OS/ContikiMAC*. URL: http://www2.htw-dresden.de/~wiki_sn/index.php5/Contiki_OS/ContikiMAC (besucht am 11. 08. 2016).

[6] Sensornetze Wiki der HTW Dresden: *Contiki OS/Debugging*. URL: http://www2.htw-dresden.de/~wiki_sn/index.php5/Contiki_OS/Debugging (besucht am 14. 08. 2016).

[7] Sensornetze Wiki der HTW Dresden: *Contiki OS/Workflow*. URL: http://www2.htw-dresden.de/~wiki_sn/index.php5/Contiki_OS/Workflow (besucht am 16. 08. 2016).

[8] Sensornetze Wiki der HTW Dresden: *Contiki3-klonen*. URL: http://www2.htw-dresden.de/~wiki_sn/index.php5/Contiki3-klonen (besucht am 04. 08. 2016).

[9] Sensornetze Wiki der HTW Dresden: *Die Fuses des Atmega128RFA1*. URL: http://www2.htw-dresden.de/~wiki_sn/index.php5/Die_Fuses_des_Atmega128RFA1 (besucht am 11. 08. 2016).

[10] Sensornetze Wiki der HTW Dresden: *Strommessungen des Funkmoduls*. URL: http://www2.htw-dresden.de/~wiki_sn/index.php5/Strommessungen_des_Funkmoduls (besucht am 25. 08. 2016).

[11] Sensornetze Wiki der HTW Dresden: *Taktfrequenz des Atmega128RFA1*. URL: http://www2.htw-dresden.de/~wiki_sn/index.php5/Taktfrequenz_des_Atmega128RFA1 (besucht am 11. 08. 2016).

# Anlage

## Anlage 1 Python-Programm zur numerischen Berechnung des Strombedarfs

```
1 #!/usr/bin/python
2
3 # Python-Skript zur numerischen Berechnung des Stromverbrauchs mit
4 # Hilfe eines Shunt-Widerstandes. Das Skript verwertet CSV-Dateien,
5 # welche die Software PicoScope produziert hat.
6
7 import sys
8
9 # Shunt-Widerstand in Ohm.
10 RSHUNT = 5
11
12 # Minimaler Stromverbrauch in mA.
13 MAMIN = 0.00025
14
15 # Minimaler Spannungswert in mV.
16 MVMIN = MAMIN * RSHUNT
17
18 if len(sys.argv) < 2:
19   print 'Usage:', sys.argv[0], 'filename'
20   exit(1)
21
22 print 'File is:', sys.argv[1]
23
24 inpfile = open(sys.argv[1])
25
26 # Ueberspringen der ersten drei Zeilen.
27 for i in range(1, 4): inpfile.readline()
28
29 # Erste verwertbare Zeile lesen und Spannungs-Wert (in mV) extrahieren.
30 line = inpfile.readline()
31 old_v = float(line.split(',', 2)[1])
32
33 # Zeile-Zaehler initialisieren.
34 l = 1
35
36 ma_max = 0
```

```
37 ma_sum = 0
38
39 line = inpfile.readline()
40
41 # Alle weiteren Zeilen der Eingabedatei bis zum Ende konsumieren.
42 while line:
43   l = l + 1
44   new_v = float(line.split(',', 2)[1])
45
46   # >>> Berechnen des aktuellen Stroms.
47   cur_v = (old_v + new_v) / 2
48
49   if cur_v < MVMIN:
50     ma = MAMIN
51   else:
52     ma = cur_v / RSHUNT
53   # <<<
54
55   if ma > ma_max: ma_max = ma
56   ma_sum = ma_sum + ma
57
58   line = inpfile.readline() # Naechste Zeile lesen.
59   old_v = new_v
60
61 # Ergebnis ausgeben.
62 print 'Read', l, 'lines␣from␣file', sys.argv[1] + '.'
63 print 'Min.', MAMIN, 'mA'
64 print 'Max.', ma_max, 'mA'
65 print 'Avg.', ma_sum / l, 'mA'
66
67 inpfile.close()
```

## Anlage 2 Messung ohne Schlaf-Modus zwischen zwei CCA-Phasen