ALBERT-LUDWIGS-UNIVERSITÄT
FREIBURG
INSTITUT FÜR INFORMATIK

Institut für Rechnernetze und Telematik
Prof. Dr. Christian Schindelhauer

Bachelor-Arbeit zum Thema:

„Radio Drivers for Sensor Motes“

Damian Spyra

Betreuer:
Faisal Aslam

Diese Arbeit wurde eingereicht als Teilleistung zur Erlangung des Bachelor-Grades an der Fakultät für Angewandte Wissenschaften, Albert-Ludwigs-Universität Freiburg, 2008

Erklärung

Hiermit erkläre ich, dass ich diese Abschlussarbeit selbständig verfasst habe, keine anderen als die angegebenen Quellen/Hilfsmittel verwendet habe und alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten Schriften entnommen wurden, als solche kenntlich gemacht habe. Darüber hinaus erkläre ich, dass diese Abschlussarbeit nicht, auch nicht auszugsweise, bereits für eine andere Prüfung angefertigt wurde.

Ort, Datum

Zusammenfassung

In dieser Bachelorarbeit wurde ein Treiber für ein CC1000 Funkmodul der Firma Chipcon entwickelt, der die drahtlose Kommunikation zwischen Sensoreinheiten in einem Sensornetzwerk gewährleisten soll. Ziel der Arbeit war die Entwicklung eines Treibers, der später als ein Teil einer selbstentwickelten Java-Virtual-Machine auf einem Prozessor-Board dienen soll. Als Hardware-Plattform wurde das Mica2-Board der Firma Crossbow verwendet, das über ein Chipcon CC1000 Funkmodul und einen ATmega128L Mikrocontroller verfügt. Weiterhin beinhaltet diese Arbeit eine Beschreibung der technischen Funktionalität des CC1000 Funkmoduls sowie der Hardware-Komponenten mit welchen es kommuniziert. Für die Konfiguration und Initialisierung der Hardware-Komponente wurden Software-Routinen implementiert. Im Weiteren wurde ein Netzwerkprotokoll implementiert, das für die Aufgaben der physikalischen, der Verbindungs- und Teile der MAC-Schicht (Medium-Access-Control-Schicht) zuständig ist.

Abstract

In this bachelor thesis, a driver for the CC1000 radio device by the Chipcon company was developed. The aim of the thesis was the development of a driver, which will serve as part of a custom-built Java Virtual Machine on a processor board. The used hardware platform was a Mica2-Board from Crossbow, which contains a Chipcon CC1000 radio device as well as ATmega128L microcontroller. Furthermore, this thesis contains a description of the technical functionality of the CC1000 radio device and the hardware components, with which it communicates. For configuration and initialization of the hardware components, small software routines have been implemented. Additionaly, a network protocol was implemented, which is responsible for the tasks of the physical, the link and the MAC layer (Medium-Access-Control layer).

Inhaltsverzeichnis

1 Motivation

Die Sensornetzwerke bestehend aus kleinen Sensorknoten, die miteinander kommunizieren können, gehören zu einem sich schnell entwickelnden Feld im Bereich der eingebetteten Systeme. Drahtlos verkettete Sensornetze ermöglichen die informationstechnische Erfassung der physikalischen Welt so, dass die eingebetteten Systeme mit der Umgebung, über Sensoren und Aktuatoren interagieren können. Die Sensorknoten verbinden die Funktion zur Datenerfassung, Berechnung und drahtloser Kommunikation auf einem kleinen Prozessor-Board zusammen. Diese Fähigkeiten ermöglichen die Datenerfassung auch in schwer zugänglichen Gebieten. Durch die technologiegetriebene Entwicklung der Hardwareumgebung werden neue, schnelle und komplexe Anwendungen möglich. Die Anwendungsgebiete wie Meteorologie, Geologie oder Robotik gehören nur zu den häufigsten Anwendungskreisen der Sensornetzwerke.

Das Gebiet der Sensornetzwerke hat sich in den letzten Jahren sehr stark entwickelt und standardisiert. Die Hardware-Plattformen werden immer billiger während die Software-Umgebung immer kompakter und leichter in der Handhabung für die Entwickler wird. Zur gleichen Zeit findet die Java-Technologie ein immer breiteres Anwendungsgebiet in Sensornetzwerken. Die neuesten Projekte, wie das SunSPOT (siehe Kapitel 1.3) von Sun deuten auf eine schnelle Verbreitung von Java-Applikationen in Sensornetzen hin, die von Jahr zu Jahr größer wird. Die Entwicklung der Anwendungen für eingebettete Systeme, sowie Sensornetzwerke war bisher sehr mühsam. Der Enwickler musste mit der Hardware und deren Programmierung in einer *Low-Level-Sprache* erfahren sein. Die Entwicklung einer Anwendung beinhaltete nicht selten die Programmierung einzelner Register und Hardware-Interrupts. Eine Möglichkeit die sich mit der Entwicklung und dem Einsatz einer *Java-Virtual-Machine* in eingebetteten Systemen offenbart, ist eine Abstraktion der Hardware-Details und deren einfache Nutzung durch fertige Bibliotheken. Auf diese Art könnte sich der Entwickler nur auf die Anwendungsfunktionen und deren Eigenschaften fokusieren. Die Hochsprache Java ist außerdem sehr weit verbreitet und leicht zu erlernen. Zu den bedeutendsten Vorteilen von Java zählt auch nicht zuletzt die Plattformunabhängigkeit. Dennoch benötigen vollständige Implementationen eines eingebetteten Systems eine Grundlage in einer *Low-Level-Sprache* wie C, um darauf aufbauen zu können.

1.1 Zielsetzung

Das Ziel meiner Arbeit ist die Entwicklung eines Treibers für das Crossbow CC1000 Funkmodul eines Sensorknotens der Mica2-Plattform. Es ist ein Teil eines größeren Projektes, in dem eine komplette *Java-Virtual-Machine* für diese Platform entworfen werden soll. Das Ziel des Projektes ist die Entwicklung der Java-Applikationen auf einem Sensor-Board zu ermöglichen. Dadurch soll die Portabilität und einfache Erweitbarkeit der Aplikationen gewährleistet werden. Die Java-Sprache gehört zur den *„höheren“* Programmiersprachen. Demzufolge kann sie keine *„low level“*-Operationen wie Hardwaresteuerung bedienen. Aus diesem Grund ist es nötig einen Teil der Software-Plattform, welcher die niedrigste physikalische Schicht umfasst, in einer Sprache zu entwickeln, die sich besser zu diesem Zweck eignet.

Aus diesem Grund wird der Treiber für das CC1000 Funkmodul zum größten Teil in der Sprache C geschrieben. Der Aufgabenbereich des Treibers umfasst die Konfiguration und

Initialisierung der Hardwarekomponenten, was die Emplementierung der grundlegendsten Methoden zum Schreiben und Lesen aus einem Register ermöglicht.

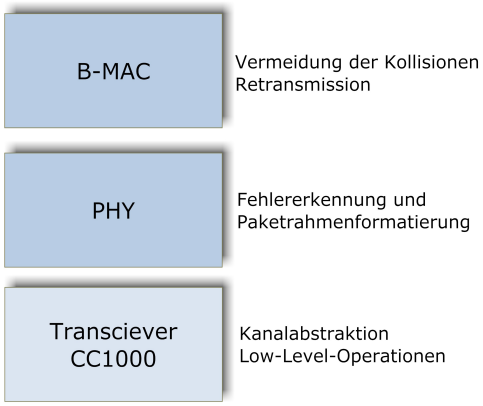

Abbildung 1: Aufbau des Funkmodultreiber.

Mit Hilfe dieser Methoden werden kleine Konfigurations-Routinen für das CC1000 Funkmodul entworfen. Der erste Teil der Implementierung soll nur die nötigsten *„low-level“*-Operationen behandeln. Folglich soll das komplette Konfigurationsinterface sowie das Dateninterface des CC1000 Funkmoduls zum Mikrocontroller bereitgestellt werden. Dadurch wird die Steuerung des Funkmoduls durch das Konfigurationsinterface und das Senden der einzelnen Bytes durch das Dateninterface zum Funkmodul ermöglicht. Als nächstes soll ein kleines B-MAC Protokoll entworfen werden. Dieses wird die gesamte Sendeprozedur, Datenpaketformatierung und einige Teile der MAC-Schicht umfassen.

1.2 Aufbau der Arbeit

Im folgenden Kapitel 2 wird die Hardware-Plattform, für den Funkmodul-Treiber vorgestellt. Hier werden die einzelnen Komponenten der Mica2-Plattform beschrieben. Das Kapitel 3 wird von der Kommunikation zwischen CC1000 Funkmodul und Mikrocontroller dominiert. Hier wird gezielt das Konfidurations- und das Dateninterface zwischen den beiden Komponeneten genauer besprochen. Im Kapitel 4 werden alle wichtige Einstellungen des CC1000 Funkmoduls der Firma Chipcon besprochen. Das fünfte Kapitel wird dem Softwarekonzept gewidmet. In diesem Kapitel werden die gesamten Softwarelösungen vorgestellt, welche von den einfachsten Routinen zur Registerverwaltung bis zum Datenpaket-Format und Routinen, die für die Verbindungs-Schicht verantwortlich sind, reichen. Hier wird auch die Initialisierung und Verwendung einiger Schnittstellen (*SPI*, *serielle Schnittstelle*), die zur Kommunikation zwischen Hardware-Komponenten benötigt werden, beschrieben. Experimente und deren Ergebnisse, die zur Feinabstimmung der Sende-Routine beigetragen haben, werden im Kapitel 6 präsentiert. In diesem Kapitel wird die komplette Software ausführlich auf die Zuverlässigkeit getestet.

1.3 Verwandte Arbeit

Es gibt bereits Projekte im Bereich der Sensornetzwerke, wie zum Beispiel der *„SunSPOT©“* von SUN („SPOT“ steht für *Small-Programmable-Object-Technologie*). SunSPOT ist eine batteriebetriebene, Java-basierte Sensor-Plattform, die mit einer *Java-Virtual-Machine* „Squawk©“ arbeitet. Die Entwickler von SUN arbeiten zwar mit anderen Hardwarekomponenten als die Mica2-Plattform, sie benutzen aber die *Java-Virtual-Machine* „Squawk©“, welche offiziell als Java-kompatibel zertifiziert wurde. Sie haben auch die gesamte JVM sowie Bibliotheken und Treiber für die Netzwerk-Schicht als *„Open-Source“* frei geöffnet. So entsteht eine Möglichkeit einzelne Teile des SunSpot-Codes zu importieren und an die Bedürfnisse dieser Arbeit anzupassen. Die SunSPOT Netzwerkarchitektur basiert auf dem 802.15.4 Standard. Der 802.15.4 Standard unterstützt die physikalische Schicht und MAC-Schicht. Darauf wird die Netzwerk-Schicht mit dem AODV Routingprotokoll (Ad-hoc On-demand Distance Vector) aufgebaut. Die SunSPOT Hardware-Plattform benutzt das CC2420 Radio-Modul. Die Hardware des CC2420 Moduls unterstützt den 802.15.4 Standard [3], so dass keine Unterstützung mehr seitens der Software nötig ist.

Da die in dieser Arbeit verwendete Plattform über ein älteres Funkmodul (CC1000) verfügt, ist es nicht möglich die Teillösungen für die Verbindungs- und MAC-Schicht von SunSPOT zu übernehmen. Die Hardware des älteren CC1000 Moduls bedient weder die physikalische Schicht (Paketrahmen, Bit-Fehler-Prüfung), noch die MAC-Schicht (Carrier-Sense-Multiple-Access-with-Collision-Avoidance). Aus diesem Grund ist es nötig, die zwei Netzwerkschichten in den CC1000 Modul-Treiber zu implementieren. Im Rahmen dieser Arbeit wurde jedoch ein B-MAC Protokoll (Berkeley Media Access Control) [7] entwickelt.

2 Hardware

Als Hardwarevorlage für die Treiberentwicklung dient das Mica2 Prozessor-Board von der Firma Crossbow. Ein Mica2 Prozessor-Board besteht aus einem Mikrocontroller auf dem die Software (in diesem Fall ein Radiotreiber) laufen soll. Die restlichen Komponenten der Mica2-Plattform sind: ein *Funkmodul*, welches die drahtlose Kommunikation zwischen mehreren Prozessor-Boards ermöglicht, ein *Flash-Speicher* für die gesammelten Sensordaten, ein *Erweiterungsinterface* für die Datenkommunikation mit einem PC und eine *Silicon-Serial-Number*, die zur Identifizierung eines Boards verwendet werden kann. Für die Verbindung mit einem PC wird ein Gateway-Board mit einem USB-Interface verwendet.

2.1 Mica2-Plattform

Das Prozessor-Board besteht aus fünf Hauptteilen:

1. Mikrocontroller
2. Flash-Speicher
3. Funkmodul
4. 51-Pin Erweiterungsinterface
5. Silicon-Serial-Number

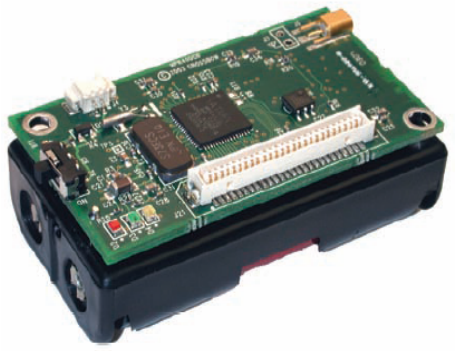

Abbildung 2: Eine Prozessor-Board-Einheit der Mica2-Plattform. *Quelle: [6]*

Mica2 ist in verschiedenen Versionen verfügbar, die sich hauptsächlich nur mit dem Funkmodul und demzufolge auch mit der Sendereichweite unterscheiden.

- **MPR400CB** Frequenz 868/916 MHz, Übertragungsrate bis zu 38,4 Kbits/s (Manchester-Kodierung), Reichweite 167 m.
- **MPR410CB** Frequenz 433 MHz, Übertragungsrate bis zu 38,4 Kbits/s (Manchester-Kodierung), Reichweite 330 m.
- **MPR420CB** Frequenz 315 MHz, Übertragungsrate bis zu 38,4 Kbits/s (Manchester-Kodierung), Reichweite 330 m.

2.1.1 Mikrocontroller

Die in dieser Arbeit verwendete Mica2-MPR400CB Plattform basiert auf einem Atmel ATmega128L Mikrocontroller. Die integrierten Schnittstellen des Mikrocontrollers (digital I/O, I2C, SPI, USART) ermöglichen eine einfache Verbindung mit anderen peripheren Geräten. Der Mikrocontroller ist die Basis des Prozessor-Boards. Mit dem Mikrocontroller werden sämtliche Prozesse und Anwendungen gesteuert. Hier findet die Übertragung und Verarbeitung der aufgenommenen Daten statt. Der Mikrocontroller ist auch für die Steuerung der anderen Komponenten zuständig, zudem ist ATmega128L mit mehreren stromsparenden „Power-Down-Modi“ ausgestattet. Dadurch können je nach Anwendung andere Komponenten, die mit dem Mikrocontroller in Verbindung stehen in den Schlafmodus versetzt werden. Diese Stromspar-Modi haben eine positive Auswirkung auf die Betriebszeit eines Sensor-Boards, das batteriebetrieben und ohne Wartung längere Zeitperioden im Betrieb verbringen kann. Die Taktfrequenz des Mikrocontrollers lässt sich softwaremässig anpassen.

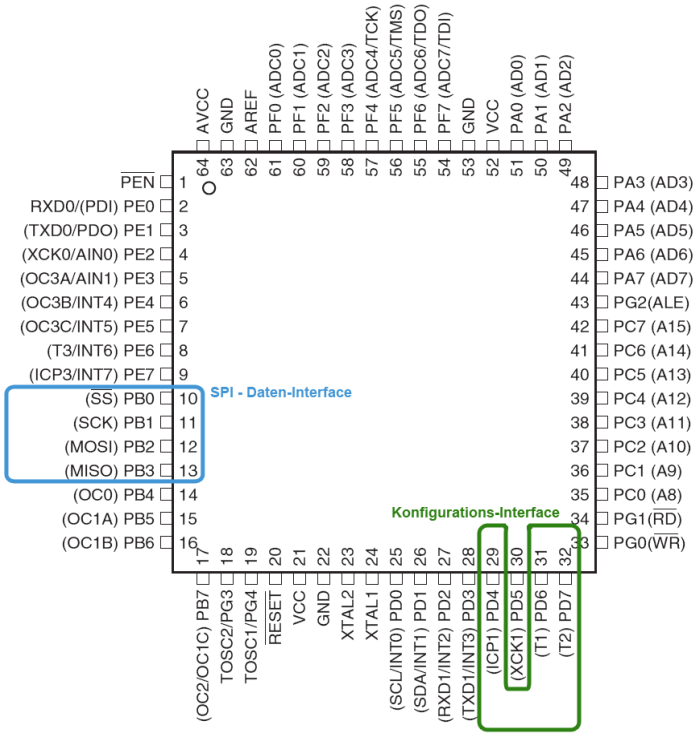

Abbildung 3: Die Pinbelegung des ATmega128L Mikrocontrollers. *Quelle: [1]*

Eigenschaften des Atmel-ATmega128L Mikrocontrollers:

- Hohe Taktfrequenz bis zu 8 MHz
- Speicher
  - 128KByte großer Flash-Programmierspeicher für das Betriebssystem und zusätzliche Applikationen. Wiederbeschreibbarkeit liegt laut Hersteller bei 10.000 Zyklen

  - 4KByte EEPROM für häufig benutzte Daten wie Initialisierungsdaten, die selten überschrieben werden sollen. Wiederbeschreibbarkeit liegt laut Hersteller bei 100.000 Zyklen
  - 4KByte großes SRAM für die Datensammlung, Auswertung und Übertragung
  - Externes Speicher-Interface erlaubt die Erweiterung der Speicherkapazität um 64 KByte

- Peripherie – optional können digitale I/O je nach Anwendung aktiviert oder deaktiviert werden

  - Zwei voneinander unabhängige serielle USART-Interfaces (Universal(Synchron/Asynchron) Receiver/Transmitter)
  - JTAG, ermöglicht die Speicherprogrammierung und *„on chip debugging“*
  - I2C *„Two wire serial Interface“*, eine zweidrahtige, byteorientierte, serielle Schnittstelle
  - SPI *„Serial-Peripheral-Interface“* kann in zwei Modi arbeiten (Master/Slave). Es dient zur Kommunikation mit anderen Modulen (Abbildung 3).
  - 8-Kanal 10-Bit Auflösung ADC
  - Mehrere PWM-Ausgänge (Pulse-Width-Modulation) mit 8- oder 16-Bit Auflösung

2.1.2 Flash-Datenspeicher

Die Mica2-Plattform ist mit einem 512 kByte Flash-Datenspeicher (*„Flash-Logger“*) ausgestattet, der zur Speicherung der Sensordaten gedacht ist. Der Speicher-Chip verfügt über ein SPI-Interface, das aber auf der Mica2-Plattform nicht benutzt wird (der SPI-Bus wird bereits vom Funkmodul benutzt). Stattdessen wird der Flash-Speicher durch ein USART-Interface am ATmega128L Mikrocontroller mit ihm verbunden.

Die wichtigsten Eigenschaften des Flash-Datenspeichers:

- Niedrige Versorgungsspannung 2.5V - 3.6V oder 2.7V - 3.6V für den Batteriebetrieb geeignet
- 4 Mbit Größe
- Serial Peripheral Interface (SPI) mit Taktung bis 20 MHz möglich
- Zwei 264 Byte große S-Ram-Puffer ermöglichen die ungestörte Datenerfassung auch bei eingeleiteter Programmierung
- Energieverbrauch von 4 mA beim Lesen, 15 mA beim Schreiben und 2 μA im Standby-Modus

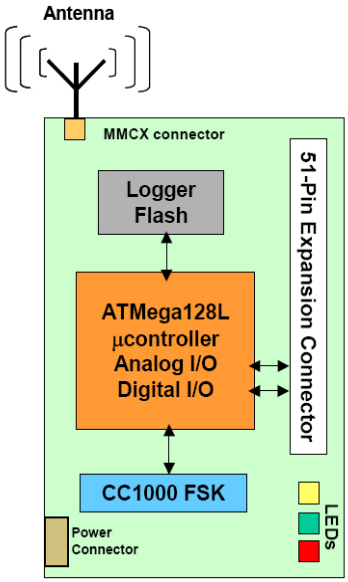

Abbildung 4: Komponentendiagramm des Mica2-Prozessor-Boards. *Quelle: [6]*

2.1.3 Funkmodul

Das Funkmodul ist für die Datenübertragung zwischen Sensorknoten verantwortlich. Das Funkmodul ist der größte Energieverbraucher eines Sensorknotens. Die Erzeugung einer elektromagnetischen Welle ist beim Senden sehr energieaufwendig. Auf der anderen Seite muss der Empfänger viele Komponenten, die für die Modulation und Demodulation des analogen Signals zuständig sind, unterhalten. Aus diesem Grund ist der Stromverbrauch eine signifikante Größe für den Betrieb des gesamten Sensorknotens. Die Mica2-Plattform benutzt dafür ein CC1000 Funkmodul von der Firma Chipcon. Dieses benutzt Frequency-Shift-Keying Modulation mit bis zu 76.8 Kbits/s mit der Non-Return-To-Zero-Kodierung (Verwendung der Manchester-Kodierung verursacht eine Halbierung der Übertragungsrate auf 38,4 Kbits/s). Das CC1000 Funkmodul zeichnet sich durch niedrige Versorgungsspannung und Stromverbrauch aus. Zu den wichtigsten Eigenschaften des Chipcon CC1000 Funkmoduls gehören:

- Simple „Single-Chip-Baumform“: es sind nur sehr wenige externe Elemente für den Betrieb nötig
- Niedrige Versorgungsspannung zwischen 2,1-3,6 V. Das Modul ist somit für die Batteriebetrieb geeignet
- Eine in 250 Hz Schritten programmierbare Ausgangsfrequenz. Diese deckt den gesamten Bereich von 300 bis 1000 MHz ab. Mögliche Bänder sind: 315, 433, 868 und 915 MHz [1]
- Programmierbare Ausgangs-Feldstärke von -20 bis zu 10 dBm
- RSSI-Ausgang für die Bestimmung der Empfangsfeldstärke

[1] In Europa zugelassene ISM-Bänder sind von 433,05 bis 434,79 MHz und 868 bis 870 MHz

- Kodierungsmöglichkeiten:
  - Synchron-NRZ „*Non-Return-To-Zero*“ Modus – taktsynchrone Datenübertragung ohne Kodierung mit 0,6 bis zu 76,8 Kbits/s
  - Synchron-Manchester Modus – taktsynchrone Datenübertragung mit Manchester-Kodierung 0,3 bis zu 38,4 Kbits/s
  - Transparent-Asynchron-UART Modus – asynchrone Datenübertragung ohne Codierung mit 0,6 bis zu 76,8 Kbits/s

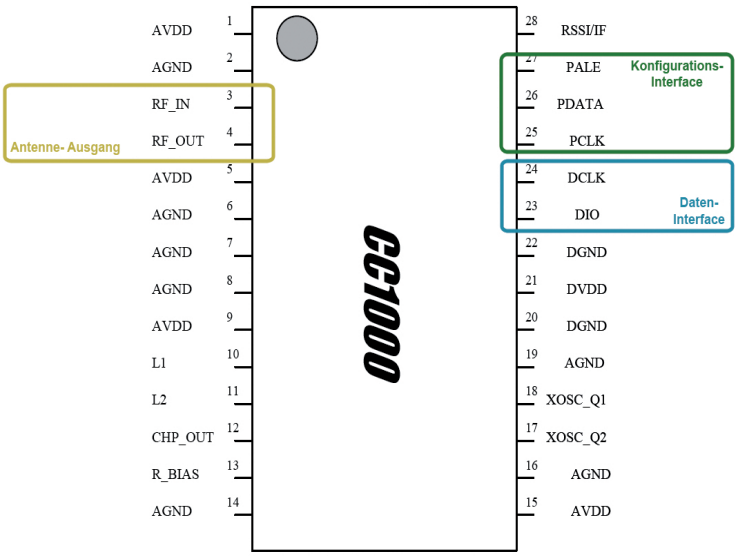

Abbildung 5: Pinbelegung des Chipcon CC1000 Funkmoduls. *Quelle: [4]*

2.1.4 Silicon-Serial-Number

Maxim DS2401 ist ein Chip mit gespeicherter Seriennummer, der auf der Platine einer elektronischen Einheit eingesetzt ist. Der Vorteil dieser Lösung im Vergleich zu einer auf der Platine aufgedruckten Seriennummer ist, die Lesbarkeit der Seriennummer durch die Software. So dient eine *Silicon-Serial-Number* als ein digitales Label für eine Einheit. Es besteht die Möglichkeit diese Nummer im Netzwerkbetrieb als Identifizierung (Network-Node-ID) zu nutzen, weil sie einmalig und damit eindeutig ist.

2.1.5 Das 51-Pin Erweiterungsinterface

Das 51-Pin Erweiterungsinterface der Firma *Hirose* ist eine Schnittstelle mit dessen Hilfe die Mica2-Plattform mit einem anderen Gerät verbunden werden kann. Diese Schnittstelle macht zusammen mit einem Gateway-Board eine Verbindung zu einem PC möglich. Durch diese Verbindung kann die Mica2-Plattform programmiert werden. Es ist auch möglich mit verschiedenen PC-Applikationen zu kommunizieren.

2.2 MIB520CB Serial Gateway

Das Gateway-Board dient zur Verbindung eines Sensor-Boards mit anderen Geräten. Das Gateway-Board ermöglicht den Datentransport zwischen einem PC und einem Sensor-Board durch das USB-Interface. Dadurch wird eine Grundlage für die Entwicklung verschiedener Applikationen geschaffen.

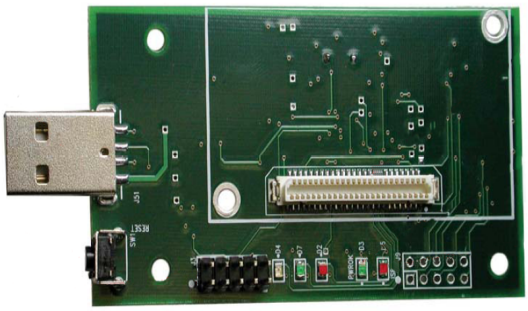

Abbildung 6: MIB520CB Gateway-Board für das Mica2-Prozessor-Board. *Quelle: [6]*

3 CC1000 Radiomodul und ATmega128L Mikrocontroller

Der ATmega128L Mikrocontroller verfügt über mehrere Interfaces, die als Verbindung zu anderen Modulen verwendet werden können. Eines dieser Module ist das Chipcon CC1000 Funkmodul. Die Verbindung zwischen dem CC1000 Funkmodul und dem Mikrocontroller wird mit Hilfe bestimmter Pins des Mikrocontrollers hergestellt. Diese können nach Bedarf entweder als Ausgang oder Eingang definiert werden. Als erstes muss das Richtungsregister eingestellt werden. Mit Hilfe eines Pins können nur dann Daten geschrieben werden, wenn dessen Richtungsregister auf *Ausgang* (logisch 1) gesetzt wurde. Dementsprechend ist es möglich nur dann mit einem Pin zu lesen, wenn sein Richtungsregister auf *Eingang* logisch 0 gesetzt ist.

In laufenden Systemen sollte das Chipcon CC1000 Funkmodul mit dem Atmega128L Mikrocontroller kommunizieren und von ihm konfiguriert werden. Der Mikrocontroller muss in der Lage sein, das Funkmodul durch das Konfigurationsinterface, bestehend aus **PALE**-Pin (*Program-Address-Latch-Enable*), **PCLK**-Pin und **PDATA**-Pin, zu konfigurieren und ihn in verschiedene Betriebs-Modi setzen zu können. Der Mikrocontroller muss demnach über ein synchrones Dateninterface verfügen, das mit den Dateninterface-Pins (**DIO**, **DCLK**) des CC1000 Funkmoduls kommunizieren kann (siehe Abbildung 5).

3.1 Das Interface zum CC1000 Radiomodul

Die Kommunikation zwischen Funkmodul und Mikrocontroller erfolgt duch folgende Pins des CC1000 Funkmoduls:

- Der Mikrocontroller benutzt drei Ausgangspins für das Konfigurationsinterface:
  1. **PDATA** ist mit einem *bi-direktionalen* Pin verbunden werden, der den Konfigurationsregistern das Lesen und Schreiben ermöglicht
  2. **PCLK** ist mit einem Pin verbunden, der den Takt von der Seite des Mikrocontrollers bereitstellen kann
  3. **PALE** ist mit einem einfachen Input-/Output-Pin verbunden
- Dateninterface zwischen Mikrocontroller und CC1000:
  1. **DCLK** ist mit einem Eingangspin des Mikrocontrollers verbunden und stellt den Datentakt bereit
  2. **DIO** ist mit einem *bi-direktionalen* Pin verbunden und sorgt für die Datenübertragung in zwei Richtungen

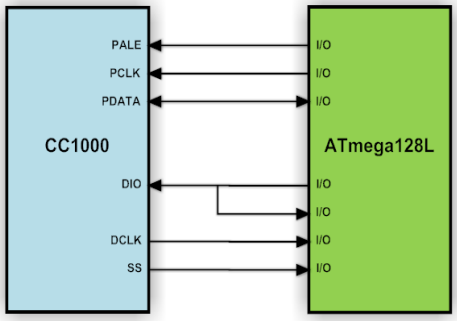

Abbildung 7: Verbindung des Chipcon CC1000 Funkmoduls mit dem ATmega128L Mikrocontroller.

Um eine reibungslose Kommunikation zu ermöglichen, muss das Richtungsregister wie in der Abbildung 7 konfiguriert werden. Der **PALE**- und der **PCLK**-Pin am Funkmodul werden als Eingang definiert und mit dem *PD4*-Pin (ICP1 Timer/Counter) und *PD6*-Pin (T1 Timer/Counter, Clock-Input) an *Port D* verbunden. Diese zwei *Port D*-Pins werden beim Mikrocontroller auf Ausgang gesetzt. Das heißt, dass der Mikrocontroller bei der Übertragung der Daten den Takt bestimmt sowie den gesamten Verlauf der Operation kontrolliert.

3.2 Das Dateninterface

Im folgenden Kapitel wird das Dateninterface zur Datenübertragung zwischen ATmega128L Mikrocontroller und Chipcon CC1000 Funkmodul beschrieben. Der Datenaustausch erfolgt über einen SPI-Bus. SPI (*Serial-Peripheral-Interface*) wurde von *Motorola* entwickelt und beschreibt einen seriellen synchronen Bus, der es ermöglicht mehrere periphere Geräte an den Mikrocontroller anzubinden. Es besteht auch die Möglichkeit mehrere Mikrocontroller miteinander zu verbinden.

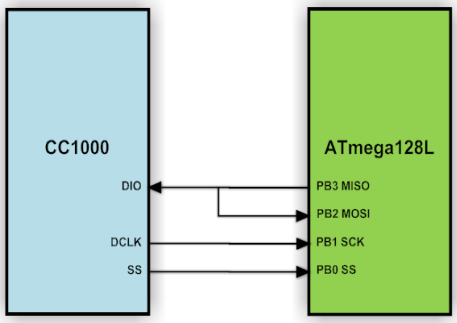

Abbildung 8: Das Dateninterface des CC1000 Funkmoduls.

3.2.1 SPI – Serielle Peripherie Schnittstelle

SPI ist ein *Voll-Duplex-Bus*, der die Datenübertragung durch ein SPI-Bus in beide Richtungen gleichzeitig ermöglicht. SPI kann in den zwei Modi *Master* oder *Slave* arbeiten. Jeder *Slave* besitzt einen *Slave-Select* $\overline{\textbf{SS}}$ oder *Chip-Select* **CS** Pin. Diese sind meistens *„Low-Aktiv“*. Solange ein *Slave* auf dem $\overline{\textbf{SS}}$ Pin nicht selektiert ist, befinden sich das Taktsignal des **DCLK**-Pins und das Datensignal des **DIO**-Pins im sogenannten *„Tri-State-Modus“*. Durch einen hochohmigen Widerstand werden also keine Daten oder Takte an den SPI-Bus durchgelassen. Der *Master* entscheidet durch Setzen von $\overline{\textbf{SS}}$ oder *Chip-Select* **CS** auf *„low“* (logisch 0) mit welchem *Slave*-Gerät er selbst kommunizieren will. Der *Master* leitet also die Datenübertragung ein und stellt den Takt bereit.

Folgende Pins sind für den Datenfluss auf der Seite des CC1000-Funk-Moduls verantwortlich. **DIO** (*Data-Input/-Output*) ist mit einem *bi-directional* **IO** Pin mit dem Mikrocontroller verbunden. **DCLK** ist ein Ausgang, der den Takt für die Datenübertragung bereitstellt. Wenn die synchrone Manchester-Kodierung oder die Non-Return-to-Zero-Kodierung verwendet wird, sollte der **DCLK** mit einem Pin verbunden werden, der einen *Interrupt* beim Mikrocontroller auslösen kann. Im *Sende-Modus* sollte der *Interrupt* auf der fallenden **DCLK** Flanke ausgelöst werden, im *Empfangs-Modus* auf der steigenden **DCLK** Flanke. Innerhalb des *Interrupts* sollten am **DIO** Pin die zu empfangenden Daten ausgelesen werden. Datenaustausch zwischen CC1000 Funkmodul und ATmega128L Mikrocontroller findet immer in Non-Return-to-Zero-kodierter Form statt. Die Manchester-Kodierung/-Dekodierung in der drahtlosen Kommunikation wird von der Hardware des CC1000 Funkmoduls übernommen. Die im gewählten Kodierungsmodus kodierten Daten werden dann drahtlos gesendet. Der Datenfluss wird mit Hilfe der Leitungen **MISO** (*Master-In-Slave-Out*) und **MOSI** (*Master-Out-Slave-In*) vom SPI-Bus an den **DIO**-Eingangspin des Funkmoduls übertragen. Da das Chipcon CC1000 Funkmodul im Master-Modus arbeiten soll, ist es wichtig die **MISO**- und **MOSI**-Pinrichtung richtig zu setzen. PB3 Pin **MISO** (*Master-In-Slave-Out*) muss beim Mikrocontroller ein Ausgang und PB2 Pin **MOSI** (*Master-Out-Slave-In*) ein Eingang sein. Synchron zum Taktsignal des *Master* werden die Daten an die Leitung gesendet. Ein *„Paket“*, welches durch SPI gesendet wird hat immer 8 Bits. Zuerst wird der *MSB-Bit* (*Most-Significant-Bit*) in das Register geschrieben, danach werden die folgenden Bits übertragen. Es wird nur ein Register zum Senden und Empfangen der Daten benutzt. Die Sendedaten liegen also im gleichen Register (**SPDR**-*SPI-Data-Register*) wie die empfangenen Daten. Wird etwas in das **SPDR**-Daten-Register geschrieben, wird sofort eine Übertragung eingeleitet. Wenn die Übertragung komplett beendet wurde, wird

ein „Flag“ im **SPDR**-Register gesetzt (Bit 7, *SPIE - SPI-Interrupt-Flag*) und gegebenenfalls ein Interrupt ausgelöst.

3.2.2 SPI – die Polarität und *„Clock-Phase“*

SPI lässt einige besondere Konfigurationsmöglichkeiten zu. So kann konfiguriert werden in welchem logischen Zustand sich der Takt während des Ruhestands des Mikrocontrollers befindet. Man kann ebenfalls einstellen wann (auf der *steigenden* oder *fallenden* Flanke) ein Datenbit übernommen wird. Für die Polarität und *„Clock-Phase“* bei dem ATmega128L Mikrocontroller sind **CPOL** und **CPHA** zuständig.

Beschreibung	CPOL	CPHA
Ruhestand-Tak „low“. Datenübertragung auf der fallenden Flanke	0	0
Ruhestand-Tak „high“. Datenübertragung auf der steigenden Flanke	1	0
Ruhestand-Tak „low“. Datenübertragung auf der steigenden Flanke	0	1
Ruhestand-Tak „high“ Datenübertragung auf der fallenden Flanke	1	1

Tabelle 1: Mögliche Zustände der Polaritäts- und *Clock-Phase*-Register. *Quelle: [1]*

Die Übertragungsgeschwindigkeit ist von der verwendeten Codierung abhängig. Im Fall der Mica2-Plattform gibt es zwei Kodierungsmöglichkeiten. Die erste davon ist NRZ *„Non-Return-to- Zero“* und taktsynchrone Datenübertragung mit Manchester-Kodierung. Bei der Verwendung der NRZ-Kodierung werden die Daten mit einer Geschwindigkeit gleich der Funkfrequenz mit bis zu 76,8 kbits/s übertragen. Für die Manchester-Kodierung beträgt die Datenrate nur die Hälfte der Funkfreqenz, somit 38,4 kbits/s.

3.2.3 SPI – Steuerungs-Register

Der Atmega128L Mikrocotroller besitzt drei Register, die für die Steuerung und Konfiguration des SPI-Busses zuständig sind. Das Datenregister **SPDR** ist ein Register, in welches die zu übertragenden Daten geschrieben werden können. Gleichzeitig können die empfangenen Daten aus diesem Register ausgelesen werden. Neben dem Datenregister gibt es zwei weitere Register: das Konfigurationsregister **SPCR** und das Statusregister **SPSR**.

Der SPI-Bus muss konfiguriert werden, bevor er benutzt werden kann. Während der Initialisierung des SPI-Buses werden die zwei 7-Bit langen Register **SPCR** und **SPSR** benutzt. Dieser Vorgang wird genauer in Kapitel 5 beschrieben.

3.2.4 SPI – Übertragungs-Frequenz

Sowohl das Funkmodul als auch der Mikrocontroller verfügen über eigene Quarz-Taktgeber. Es ist darauf zu achten, dass beide mit synchroner Frequenz arbeiten, damit keine Registerinhalte „übersehen“ oder doppelt gelesen werden. Nur wenn die Mikrocontroller- und Funkmodul-Frequenzen für die Berechnung der Baudrate richtig definiert wurden (7.372800 MHz Quarz-Frequenz) können die Daten korrekt übertragen und auf dem PC korrekt ausgegeben werden.

3.3 Das Konfigurationsinterface

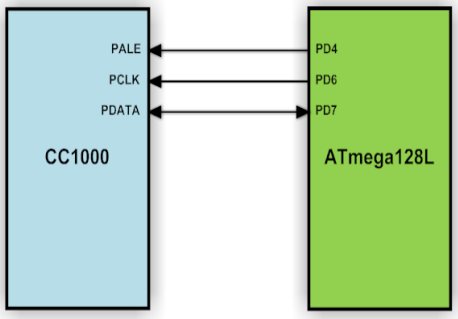

Abbildung 9: Das Konfigurationsinterface des CC1000 Funkmoduls.

Für die Konfiguration des Funkmoduls durch den Mikrocontroller wird kein vordefiniertes Interface beim Mikrocontroller benutzt. Stattdessen wird zum diesem Zweck ein I2C-änlicher 3-Drath-Bus auf den Mikrocontroller-Pins abgebildet. Auf diese Art sind drei oben genannte Funkmodul-Pins (**PALE**, **PCLK** und **PDATA**) mit bestimmten Pins des Mikrocontrollers verbunden. Es gibt drei Pins an *PORT D*: *PD4* für **PALE**, *PD6* für **PCLK** und *PD7* für **PDATA**. Der Mikrocontroller hat durch diese Pins den Zugriff zu allen Konfigurations-Registern des CC1000 Moduls. Es kann die Register entweder neu beschreiben oder auslesen. Auf diese Weise werden sowohl der Zustand als auch alle möglichen Parameter des Moduls verändert und justiert. Der Mikrocontroller kann mit Hilfe des Konfigurations-Interfaces alle notwendigen Einstellungen am Funkmodul vornehmen. Dazu zählen:

- Sende- oder Empfangsmodus
- Sendestärke
- Sendeparameter wie:
  - Betriebsfrequenz
  - FSK Frequenzabweichung zwischen einer Überlagerungsoszillatorfrequenz und einer Trägerfrequenz
  - Referenzfrequenz des Kristall-Oszillators
- Energiemodus („*Schlaf*“- und „*Aufwach*“-Modus)
- Ein-/Ausschaltung des Kristall-Oszillators
- Datenrate und das Datenformat (NRZ-, Manchester-Kodierung oder UART Interface)
- PLL Frequenz-Synthesizer-Modus
- Optionaler RSSI/IF Empfangsfeldstärke-Ausgang

Die aktuelle Konfiguration des Chipcon CC1000 Moduls wird im RAM-Speicher gehalten. Deshalb muss das ganze Modul jedes Mal komplett neu konfiguriert werden, wenn ein

Parameter geändert werden soll. Während die Konfiguration und die Datenübertragung stattfindet, ist der Mikrocontroller im *Master-Modus*. Die Pins *PD4* und *PD6* werden als Ausgänge an **PALE** und **PCLK** konfiguriert. Der an **PDATA** gebundene Pin *PD7* ist ein Ein-/Ausgangs-Pin, mit Hilfe dessen die Konfigurationsdaten aus dem Fukmodul ausgelesen oder neu beschrieben werden können.

3.3.1 Die Schreiboperation

Die Schreib- und Leseoprationen werden nach einem bestimmten Muster und in einer bestimmten Reihenfolge durchgeführt. Während jeder Schreiboperation werden 16 Bits an die PDATA gesendet. Zuerst werden die sieben *(A6:0)* Adress-Bits gesendet. *A6* ist der *MSB* (Most-Significant-Bit) und wird als erstes Bit übertragen. Danach wird das Lese-/Schreib-Bit gesendet und anschließend werden die Konfigurationsdaten an das Funkmodul übertragen.

Die Prozedur einer Schreiboperation:

1. Ansteuern des **PALE**-Pins durch den Mikrocontroller auf logisch Null (PALE low)
2. Senden der Adresse-Bits
3. Senden eines Schreib-Bits 1
4. Ansteuern des **PALE**-Pins durch den Mikrocontroller auf logisch Eins (PALE high)
5. Ansteuern des **PDATA**-Pins als Eingang 0 und Übertragung der Daten

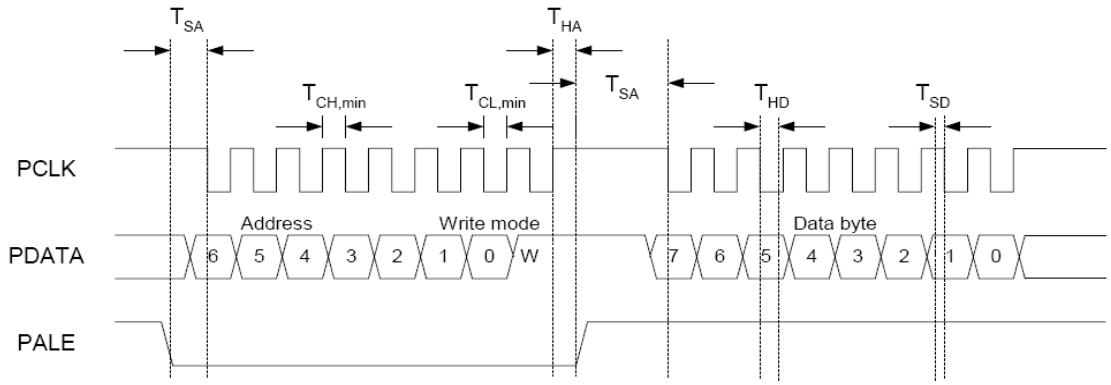

Abbildung 10: Eine Schreib-Operation am Konfigurationsinterface des CC1000 Funkmoduls. *Quelle: [4]*

3.3.2 Die Leseoperation

Die Prozedur einer Lese-Operation:

1. Ansteuern des **PALE**-Pins durch den Mikrocontroller auf logisch Null (PALE low)
2. Senden der Adresse-Bits
3. Senden eines Schreib-Bits 0
4. Ansteuern des **PALE**-Pins durch den Mikrocontroller auf logisch Eins (PALE high)
5. Ansteuern des **PDATA**-Pins als Ausgang 1 und Auslesen der Daten

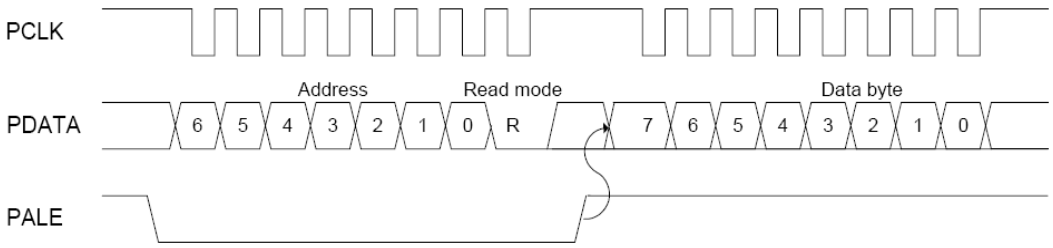

Abbildung 11: Eine Lese-Operation am Konfigurationsinterface des CC1000 Funkmoduls. *Quelle: [4]*

Der Mikrocontroller kann also die Konfigurationsdaten durch das selbe Interface schreiben und auslesen. Die aktuelle Konfiguration wird im RAM-Speicher gehalten und während des *Power-Down-Modus* beibehalten. Wurde die Stromzufuhr komplett getrennt, muss das Funkmodul neu konfiguriert werden.

Adresse	Name	Wert
0x00	MAIN	0x11 (RX-Modus)
0x01	FREQ_2A	0x75 (Frequenzbereich)
0x02	FREQ_1A	0xA0
0x03	FREQ_0A	0x00
0x04	FREQ_2B	0x58
0x05	FREQ_1B	0x33
0x06	FREQ_0B	0x13
0x07	FSEP1	0x01
0x08	FSEP0	0xAB
0x09	CURRENT	0x8C (RX Current)
0x0A	FRONT_END	0x32
0x0B	PA_POW	0x80
0x0C	PLL	0x40 (RX PLL)
0x0D	LOCK	0x10
0x0E	CAL	0x26
0x0F	MODEM2	0x90
0x10	MODEM1	0x6F
0x11	MODEM0	0x43
0x12	MATCH	0x10
0x13	FSCTRL	0x01
0x14	Reserviert	0x00
0x15	Reserviert	0x00
0x16	Reserviert	0x00
0x17	Reserviert	0x00
0x18	Reserviert	0x00
0x19	Reserviert	0x00
0x1A	Reserviert	0x00
0x1B	Reserviert	0x00
0x1C	PRESCALER	0x00
0x40	TEST6	0x10
0x41	TEST5	0x08
0x42	TEST4	0x3F
0x43	TEST3	0x04
0x44	TEST2	0x10
0x45	TEST1	0x3F
0x46	TEST0	0x30

Tabelle 2: Adressen der Konfigurationsregister des Chipcon CC1000 Funkmoduls.

4 Konfiguration des CC1000 Moduls

Im folgenden Kapitel werden die Einstellungsmöglichkeiten des Chipcon CC1000 Moduls besprochen. Das Funkmodul verfügt über mehrere Einstellungen, welche die verwendete Kodierung der drahtlos übertragenen Daten, die Übertragungsfrequenz oder die Stärke des Ausgangssignals ansteuern können. Die fehlerfreie Konfiguration des CC1000 Funkmoduls ist nicht nur für die drahtlose Kommunikation zwischen Sensor-Einheiten wichtig, sondern sie gewährleistet auch eine reibungslose Datenübertragung zwischen Mikrocontroller und Funkmodul.

4.1 Bit-Synchronisation

Die Abbildung 12 stellt die Funktionsweise des digitalen Demodulators des CC1000 Moduls dar. Das empfangene Signal wird zuerst vom **„Sampler“** erfasst. Danach legt der **„Frequency-Detector“** die momentane Frequenz des Signals fest. Nachdem das Signal empfangen wurde und seine Frequenz bestimmt, wird es durch den **„Decimator“** abgeschwächt (*„dezimiert“*) und im **„Data-Filter“** gefiltert. Nach diesem Vorgang werden die Daten zum **„Average-Filter“** weitergereicht.

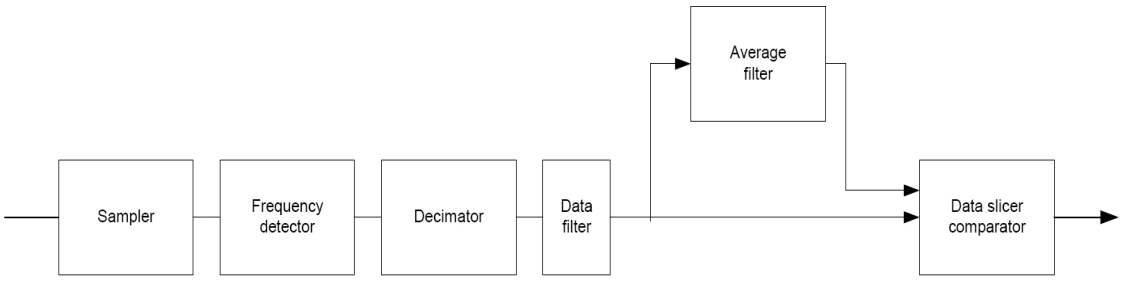

Abbildung 12: Block-Diagramm des Demodulators.
Quelle: [4]

Der **„Average-Filter“** (*ungewichteter Mittelwertfilter*) berechnet den Mittelwert der empfangenen Daten. Der **„Average-Filter“** erlaubt dem CC1000-Modul mit nicht DC-balancierten Daten zu arbeiten. Wenn der Filter sich in freiem Lauf befindet (Abbildung 13) und Daten empfängt, ist es wichtig, dass jedes Datenpaket mit einer Bitfolge aus Einsen und Nullen (*High-* und *Low-Bits*) in gleicher Anzahl anfängt. Alle Kodierungs-Modi (*Non-Return-to-Zero-* oder *Manchester-Kodierung*) brauchen für den internen **„Data- Slicer“** eine balancierte Präambel zum richtigen Abgleich der vom **„Average-Filter“** ankommenden Daten.

Die Präambel sollte folgende Gestalt haben:

- Non-Return-to-Zero-Kodierung – ein einzelnes Präambel-Byte sollte gleich 01010101 sein
- Die gleiche Bit-Folge wird mit der Manchester-Kodierung folgende Sequenz erzeugen: 0110011001100110

Diese Abfolge ist wichtig für den **„Bit-Synchronizer“** zur richtigen Synchronisation der empfangenen Daten.

Bevor die empfangenen Daten richtig ausgelesen werden können, muss der **„Average-Filter“** den Mittelwert aus der Präambel bestimmen. Dieser Mittelwert wird für den korrekten Empfang der Daten benutzt. Danach sollte der **„Average-Filter“** gesperrt werden. Das heißt, dass ab diesem Zeitpunkt der berechnete Mittelwert nicht mehr verändert wird. Die Sperre kann entweder automatisch oder manuell durchgeführt werden. Im „Polling-System“, in welchem der Empfänger sich nur dann einschalten sollte, wenn auch eine Datenübertragung erkannt wird, soll die automatische **„Average-Filter“**-Sperre benutzt werden (siehe Abbildung 15). Wenn der Empfänger die ganze Zeit aktiv nach einer Präambel sucht, sollte der Filter nach der erfolgreichen Erkennung der Präambel gesperrt werden (siehe Abbildung 14). Der Sperr-Modus des **„Average-Filter“** kann im **MODEM1**-Register eingestellt werden. Durch die geeignete Einstellung der Bits **MODEM1**[3] (**LOCK_AVG_MODE**) und **MODEM1**[4] (**LOCK_AVG_IN**) kann der gewünschte **„Average-Filter“**-Modus ausgewählt werden. Wenn das **MODEM1**[3]-Bit (**LOCK_AVG_MODE**=1) auf Eins gesetzt, und der **„Average-Filter“** gesperrt wird, wird auch der berechnete Wert im Energie-Sparmodus (*Power-Down-Modus*) oder Sende-Modus beibehalten. Erst nach einem *Reset* durch das Setzen des **MODEM1**[0]-Bits (**MODEM_RESET_N**) im **MODEM1**-Register auf Null, oder einem *Restart* des kompletten Systems wird der **„Average-Filter“** auf die Standardwerte zurückgesetzt.

Die minimale Länge einer Präambel ist von der aktuellen Kodierung und der Einschwingzeit abhängig (Tabelle 3).

4.1.1 Manchester-Kodierung

Im synchronen Manchester-Modus sollte sich der **„Average-Filter“** in „freiem Lauf“ befinden (siehe Abbildung 13). Der Filter muss in diesem Modus nicht manuell gesperrt werden. Jede Nachricht (Datenpaket) muss mit einer Präambel beginnen, so dass der **„Average-Filter“** das exakte DC-Niveau der Daten bestimmen kann. Auch die genaue Datenrate wird anhand der Präambel durch den **„Bit-Synchronizer“** synchronisiert.

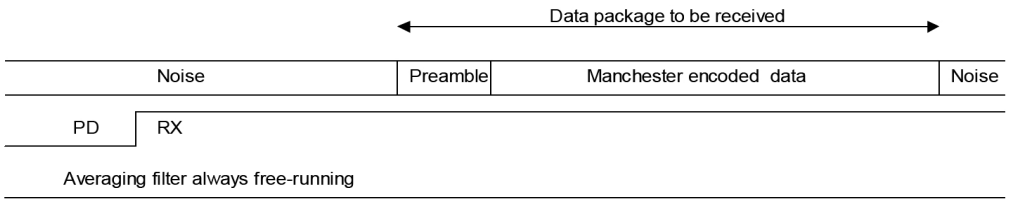

Abbildung 13: „Average Filter“ in „freiem“ Lauf.
Quelle: [4]

4.1.2 Non-Return-to-Zero-Kodierung oder Transparent-Asynchron-UART-Modus

Wenn die NRZ-Kodierung oder der Transparent-Asynchron-UART-Modus benutzt wird, muss der **„Average-Filter“** entweder automatisch oder manuell gesperrt werden. Man muss bedenken, dass die „automatische Sperrung“ nicht „intelligent“ ist. Das heißt, sie erkennt in keiner Wiese die gültigen Daten, sondern sperrt lediglich den Filter nach einer bestimmten Anzahl der *Präambel-Chips* wenn das Funkmodul in den Empfangs-Modus versetzt wird.

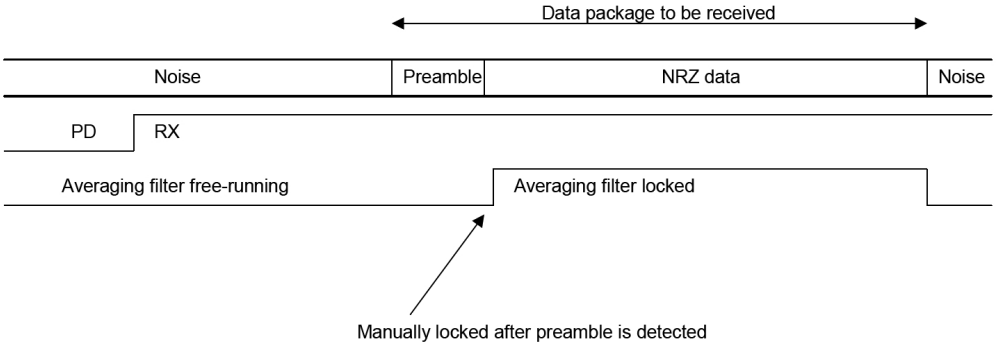

Abbildung 14: Manuelles Sperren des **„Average-Filter“**. *Quelle: [4]*

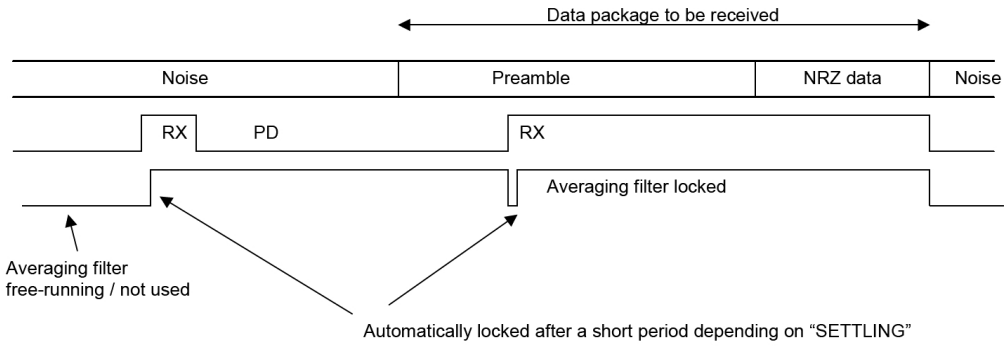

Abbildung 15: Automatisches Sperren des **„Average-Filter“**. *Quelle: [4]*

4.2 Frequenzprogrammierung

FREQ-Frequenzwort – Der Frequenzsynthesizer **PLL** wird durch zwei Frequenz-Wörter kontrolliert. *Frequenz-Wort A* und *Frequenz-Wort B* können für zwei verschiedene Frequenzbereiche programmiert werden. So kann das *Frequenz-Wort A* für die Empfangs-Frequenz (lokale Oszillator-Frequenz) und das *Frequenz-Wort B* für die Sende-Frequenz verantwortlich sein. Dadurch wird ein schnelles Schalten zwischen der Sende- und Empfangs-Frequenz ermöglicht. Aus welchem *Frequenz-Wort* die akuelle Betriebs-

Settling`MODEM1.SETTLING[1:0]`	Free-running-Manchester-mode `MODEM1.LOCK_AVG_MODE=1` `MODEM1.LOCK_AVG_IN=0`
00	23
01	34
10	55
11	98

Tabelle 3: Minimale Anzahl der Präambel-*Chips* für den **„Average-Filter“** in Manchester-Modus. *Quelle: [4]*

Frequenz angesteuert werden sollte wird duch das **F_REG**-Bit im **MAIN**-Register kontrolliert.

Jedes *Frequenz-Wort* ist 24 Bits (3 Bytes) lang. Das *Frequenz-Wort A* findet seinen Platz in den 3 folgenden Registern: **FREQ_2A**, **FREQ_1A** und **FREQ_0A**. Das *Frequenz-Wort B* wird demzufolge in **FREQ_2B**, **FREQ_1B** und **FREQ_0B** untergebracht. Das *Frequenz-Wort* kann mit folgender Formel berechnet werden:

$$f_{vco} = f_{ref} * \frac{FREQ + FSEP * TXDATA + 8192}{16384}$$

TXDATA bezeichnet Sende-Modus *1* oder Empfangs-Modus *0* bezüglich Datentransfer an der **DIO**-Leitung. Die *Referenz-Frequenz* f_{ref} ist die Frequenz des Kristall-Oszillators geteilt duch den Wert aus **REFDIV**-Bit im **PLL**-Register.

$$f_{ref} = \frac{f_{xosc}}{\mathrm{REFDIV}}$$

Mit **REFDIV**-Bit sollte eine Zahl zwischen 2 und 14 ausgewählt werden so, dass:

$$1.0\ \mathrm{MHz} \leqslant f_{ref} \leqslant 2.46\ \mathrm{MHz}$$

f_{vco} ist die Frequenz des lokalen Oszillators (*LO*) im Empfangs-Modus. Die *LO*-Frequenz muss entweder $f_{RF} - f_{IF}$ oder $f_{RF} + f_{IF}$ betragen. Diese entsprechen der unteren oder oberen Schranke des *LO*-Frequenz-Bereiches (Die Daten an der **DIO**-Leitung werden invertiert, falls die untere Schranke des *LO*-Frequenz-Bereiches benutzt wird).

f_0 beschreibt die untere Schranke der Frequenz, die im Sende-Modus verwendet wird (*FSK-Frequenz*). Die obere Schranke der *FSK*-Frequenz ist durch folgende Formel gegeben: $f_1 = f_0 + f_{sep}$. Die Separations-Frequenz f_{sep} wird durch das 11-Bit lange *Separations-Wort* im **FSEP1 : FSEP0** gesteuert.

$$fsep = f_{ref} + \frac{FSEP}{16384}$$

Es besteht die Möglichkeit die **PLL**-Frequenz zu überwachen. Durch setzen des **ALARM_DISABLE**-Bits im **PLL**-Register auf 0 wird eine automatische Frequenz-Alarm-Funktion (**PLL.ALARM_H** und **PLL.ALARM_L**) generiert. Diese zwei Bits im **PLL**-Register weisen darauf hin, dass die **PLL**-Frequenz sich dem Frequenz-Limit nähert und der **PLL** erneut kalibriert werden sollte.

Es wird empfohlen, dass das **LOCK_CONTINOUS**-Bit im **LOCK**-Register eingeschaltet wird (logisch 1), während die Frequenz zwischen Sende- und Empfangs-Modus umgeschaltet wird. Falls die **PLL**-Sperre nicht erreicht wird, muss eine neue Kalibration eingeleitet werden.

4.2.1 Die optimale Empfangs-Frequenz-Einstellung für ISM-Bänder

Die ISM-Bänder (*„Industrial, Scientific, and Medical Band“*) bezeichnen Frequenzbereiche, die durch Hochfrequenz-Geräte in Industrie, Forschung und Gesundheitswesen benutzt werden. Ausgewählte ISM-Bänder werden auch für die lizenzfreie Audio-, Video- und Datenübertragung verwendet (WLAN, Bluetooth). Die empfohlenen RX-Frequenz-Synthesizer-Einstellungen für das CC1000 Modul werden entweder im Datenblatt für das CC1000 Funkmodul [4] oder in einer Applikations-Notiz [10] vorgeschlagen. Gleichermaßen gute Einstellungen, die die optimale Empfindlichkeit für jede Frequenz garantieren, können auch aus dem **SmartRF**©-Studio ausgelesen werden.

Die eingestellte Übertragungsfrequenz im Bereich 868 MHz wurde an einem *Agilent E4405B ESA-E SERIES* Spectrum-Analyzer (9kHz-13.2GHz) überprüft.

4.2.2 Der spannungsgeregelte Oszillator *VCO*

Der spannungsgeregelte Oszillator **VCO** (*Voltage-Controlled-Oscillator*) kontrolliert den Frequenzbereich der Schaltung. Im Fall einer rapiden Temperatur- oder Spannungsänderung kann es vorkommen, dass der VCO von der Referenzfrequenz des Quarz-Oszillators abgewichen ist. Die Phasenregelschleife **PLL** (Frequenzsynthesizer) (*Phase-Locked-Loop*) wird mit einem *Phasendetector* den **VCO**-Oszillator wieder an die Referenzfrequenz anpassen.

4.2.3 *VCO* – Current

Der aktuelle Frequenz-Wert des spannungsgeregelten Oszillators **VCO** lässt sich manuell ansteuern und sollte bezüglich der Betriebsfrequenz (Sende-(*TX*) oder dem Empfangs-Modus (*RX*)) programmiert werden. Die drei **VCO_CURRENT**-Bits im **CURRENT**-Register

sind für die Stromstärke verantwortlich. Diese sollten für Sende- und Empfangs-Modus getrennt eingestellt werden [9].

4.3 Energieverbrauch-Verwaltung

CC1000 bietet eine sehr flexible Energieverbrauch-Verwaltung. Die gesamte Energieverwaltung wird durch das **MAIN**-Register organisiert. Wie in der Tabelle 4 dargestellt, lassen sich alle Energieverbrauchs-Modi durch einzelne Bits aus dem **MAIN**-Register ansteuern. Mögliche Modi sind:

- Sende-Modus – die Stromstärke bei 868 Mhz reicht von 8,6 mA (-20 dBm) bis über 26.7 mA (+10 dBm)
- Empfangs-Modus – 9,6 mA bei 868 Mhz
- Power-Down-Modus – 0,2 μA Stromstärke
- Schlaf-Modus – 30 μA Stromstärke

4.4 Received-Signal-Strength-Indication-Pin (*RSSI*)

Das Chipcon CC1000 Funkmodul hat einen eingebauten analogen RSSI/IF Empfangsfeldstärke-Ausgang. An diesem Pin kann der Mikrocontroller die Ausgangssignalstärke überwachen. **IF_RSSI**-Bit im **FRONT_END**-Register schaltet den RSSI-Ausgang ein. Das RSSI-Signal wird an den **RSSI/IF**-Pin geleitet. Ausgangssignal (Spannung) und Signalstärke sind dabei invers proportional. Das heißt, je höher die Spannung, desto schwächer ist das Signal am **RSSI/IF**-Pin. Die RSSI-Spannung kann durch einen A/D-Eingang am Mikrocontroller gemessen werden. Somit kann der RSSI/IF-Ausgang nur dann mit einem Mikrocontroller verbunden werden, falls dieser (wie ATmega128L) über einen ADC-Eingang verfügt. Das CC1000 Modul hat eine Sensitivität von -105 dBm.

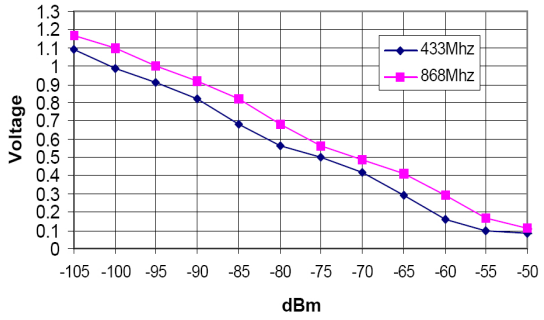

Abbildung 16: Das Verhältnis der Ausgangsspannung zur empfangenen Signalstärke am RSSI-Ausgang. *Quelle: [4]*

4.5 Programmierung der Ausgangs-Feldstärke

Das CC1000 Funkmodul erlaubt eine Steuerung der Ausgangs-Feldstärke in Schritten, die bis zu 1 dB klein sein können. Die Ausgangs-Feldstärke wird durch das **PA_POW**-Register kontrolliert. Im Schlaf-Modus (*Power-Down-Mode*) sollte das **PA_POW**-Register auf 0x00 gesetzt werden, um jegliche Energiezufuhr abzuschalten und unnötigen Energieverlust zu vermeiden. Alle mögliche Einstellungs-Werte für das **PA_POW**-Register sind im Datenblatt des CC1000 [4] in der Tabelle 11 auf Seite 32 zu finden.

4.6 CC1000-Hardware-Beschränkungen

Obwohl das CC1000 Funkmodul in einer sehr breiten Palette von Anwendungen verbreitet ist, hat es auch seine Schwächen. Das CC1000 gehört nicht zu den jüngsten Entwicklungen auf dem Markt, dementsprechend verfügt es auch über keinerlei Zusatzhardware. Seine Hardware unterstützt zwar die Daten-Kodierung (Manchester-, Non-Return-To-Zero-Kodierung), sie unterstützt aber keine Verbindungs-Schicht oder physikalische Schicht. Zu den kleineren Schwächen gehört die lange Einschaltzeit wenn das Radiomodul aus dem Schlafmodus zum Senden oder Empfangen „aufwacht“. Diese beträgt 2,25 ms (2 ms Oszillator Start-Up-Zeit, 250 μs PLL *Sperre*-Zeit). Die Umschalt-Zeit zwischen Sende- und Empfangs-Modus ist mit 200 μs wesentlich länger als bei anderen Produkten der selben Preisklasse. Die Reichweite des CC1000 ist mit ca. 160 Metern im Freien für die meisten kleinen Anwendungen zwar ausreichend, kann aber mit etwas neueren CC1100 nicht mehr mithalten. Die größte Schwäche des CC1000 Moduls ist, dass es nicht mit 2,4 GHz GFSK (*Gaussian-Frequency-Shift-Keying*) arbeiten kann. Folglich unterstützt die CC1000 Hardware nicht den 802.15.4 Standard.

Die oben beschriebene Hardware-Beschränkungen des CC1000 Funkmoduls müssten für die Integration in einer 802.15.4 Netzwerk-Umgebung auf der Software-Seite behoben werden.

5 Software

In diesem Teil meiner Arbeit werden die Software-Lösungen dargestellt, welche zur Entwicklung der Treiber für das Chipcon CC1000 Funkmodul verwendet wurden. In der ersten Phase des Projektes wurde ausschließlich die Programmiersprache C mit einem avr-gcc Compiler benutzt. Der avr-gcc ist ein freier Compiler, der in der Sprache C oder C++ geschriebene Programme zu ausführbaren Programmen compilieren kann. Diese können danach problemlos auf dem Mikrocontroller der AVR-Familie ausgeführt werden. Als Entwicklungsumgebung wurde **AVR-Studio**©Version 4.13 benutzt.

Der Grund für die Entscheidung, die *„Low-Level-Programmierung“* in der Sprache C umzusetzen ist die Lesbarkeit des Codes. Außerdem ist es wesentlich einfacher für den Programmierer sich gewisse Aspekte der AVR-Programmierung in C anzueignen als in Assembler. Den einzigen Nachteil stellt die Geschwindigkeit des auszuführenden Codes dar. Dieser Unterschied zwischen Assembler und C ist aber in den meisten Fällen sehr gering. Für die entsprechenden Aufgaben kann der Assembler-Code in C Code einfach und schnell eingebunden werden.

5.1 „Low-Level-Communication-Layer“

Der Softwareentwurf soll auf zwei Schichten verteilt werden. Der *Low-Level-Teil* wird in C entworfen. Dieser Teil wird nur die wichtigsten Funktionen zum Steuern und Konfigurieren der Hardware der Mica2-Plattform beinhalten. Obwohl C eine *höhere* Sprache als Assembler ist, kann sie die Hardware gut bedienen und ist für Zwecke dieser Arbeit ausreichend. Der zweite Teil wird die physikalische Schicht sowie die MAC-Schicht beinhalten. Der komplette Treiber wird schließlich auf einem MOTE in einer Java-Umgebung zum Einsatz kommen. Die Mica2-Plattform hat einen sehr begrenzten Speicher auf dem eine Java-Virtual-Machine implementiert werden soll. Aus diesem Grund sollte der Treiber für das Funkmodul so speichereffizient wie möglich gestaltet werden.

5.2 Konfiguration des CC1000 Funkmoduls

Das Chipcon CC1000 Funkmodul wird konfiguriert, indem seine 22 Konfigurationsregister mit bestimmten Parametern (16 Bit langen Datensätzen, siehe Kapitel 3.3.1) beschrieben werden. Zuerst müssen die einzelnen Parameter für die Konfiguration bestimmt werden. Dazu wurde **SmartRF**©-Studio verwendet (siehe Abbildung 17). Es ist ein kleines Programm der Firma Chipcon, mit Hilfe dessen die Einstellungen für die Konfigurationsregister des Funkmoduls generiert werden können.

Um die Konfigurationsregister des CC1000 lesen und beschreiben zu können, wurden zwei Methoden implementiert. Diese zwei *Routinen* sind die hierarschich *niedrigsten* im gesamten Programm, weil sie elementare Aufgaben (Beschreiben und Lesen der Register) durchführen. Das Beschreiben eines Registers wird mit der `cc1000_write(addr, val)` Methode durchgeführt. Diese bekommt als Argumente die Adresse `addr` und den Wert `val`, mit dem ein Register beschrieben werden soll. Die Methode zum Auslesen eines CC1000-Konfigurationsregisters `cc1000_read(addr)` bekommt nur die Adresse `addr` eines Registers als Argument. Aus diesem Register wird dann sein aktueller Wert ausgelesen.

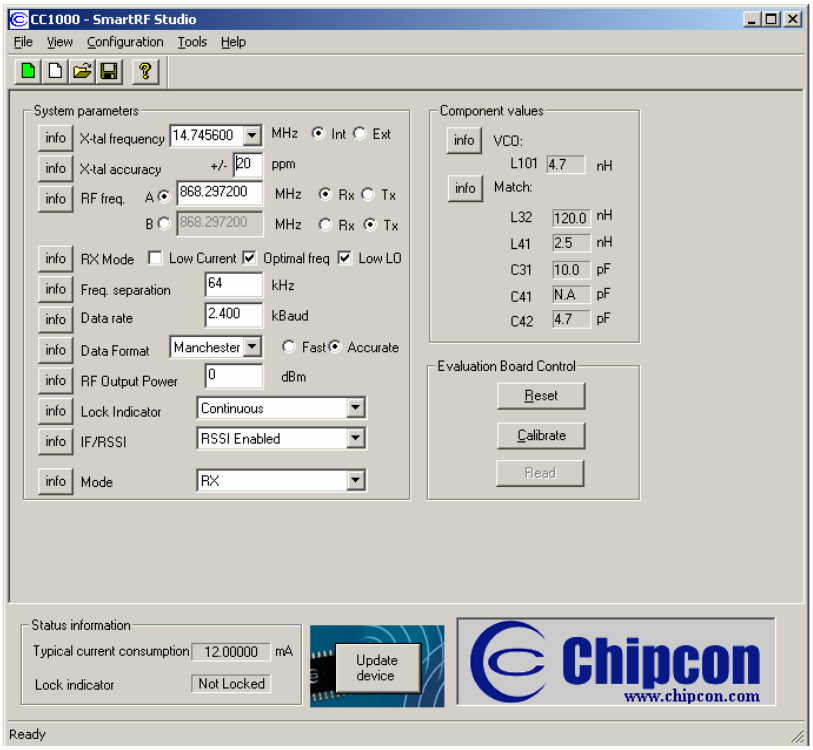

Abbildung 17: SmartRF©Studio Interface.

Die genaue Lese- und Schreib-Prozedur wurde **in Abschnitt 3.3** erklärt. Eine Veranschaulichung der Schreib-Operation findet sich in **Abbildung 10**, der Lese-Operation in **Abbildung 11**.

5.3 CC1000 Routinen

Mit Hilfe von `cc1000_write(addr, val)` und `cc1000_read(addr)` Routinen können nun bestimmte Werte in die Register des Funkmoduls geschrieben und aus diesen gelesen werden. Eine Zusammenfassung aller Register des CC1000, welche für dessen Konfiguration zuständig sind, ist in Tabelle 2 dargestellt. Je nachdem welcher Wert in ein bestimmtes Register geschrieben wird, wird sich das Modul anders verhalten. Als Beispiel sei das **MAIN**-Register genannt. Um das CC1000 in den Sende-Modus zu setzen, muss ein bestimmtes Byte in das **MAIN**-Register geschrieben werden. Durch `cc1000_write(0x00, 0xE1)` wird ein 11100001 Byte in die 0x00 Register-Adresse (**MAIN**-Register) geschrieben. Jedes Bit steuert eine Funktion; so schaltet das siebte Bit = 1 (**MAIN**[7] = 1) den TX-Modus ein. Das nächste Bit = 1 (**MAIN**[6] = 1) bedeutet, dass für den TX-Modus **Frequency-Register-B** benutzt werden soll. Dies verdeutlicht, dass jedes Bit im gesamten Byte wichtig ist. Wenn nur ein Bit versehentlich falsch gesetzt wird, wird die gesamte Funktion des Moduls möglicherweise gefährdet. Die folgenden Unterabschnitte von 5.3.1 bis 5.3.5 stellen die *höheren Routinen* zur Konfiguration des CC1000 Funkmoduls vor.

Register	Name	Standard	Aktiv	Beschreibung
MAIN[7]	RXTX	-	-	RX/TX - Modus, 0: RX; 1: TX
MAIN[6]	F_REG	-	-	Auswahl des Frequenz-Registers, 0: Register A; 1: Register B
MAIN[5]	RX_PD	-	H	Power-Down-Modus für LNA, Mixer, IF, Demodulator, für RX-Teil
MAIN[4]	TX_PD	-	H	Power-Down-Modus für den TX-Teil, PA
MAIN[3]	FS_PD	-	H	Power-Down-Modus für den Frequency-Synthesiser
MAIN[2]	CORE_PD	-	H	Power-Down-Modus für den Oscillator-Kern
MAIN[1]	BIAS_PD	-	H	Power-Down-Modus für BIAS und den Oscillator-Buffer
MAIN[0]	RESET_N	-	L	Reset ist aktiv-low. Eine Null auf MAIN[0] setzt alle andere Register als MAIN auf standard Werte zurück. MAIN-Register hat keine standard Einstellung und muss durch das Konfigurationsinterface eingestellt werden. Nach dem alle Register auf standard Wert zurückgesetz worden sind, muss auf “high“ gesetzt werden

Tabelle 4: Aufbau und Funktionen des MAIN-Registers. *Quelle: [4]*

5.3.1 Reset-Routine

Reset-Routine `cc1000_reset(void)` setzt alle CC1000 Konfigurationsregister auf ihre Standardwerte zurück. Wenn das gesamte System eingeschaltet wird, sollte als erstes die Reset-Routine durchgeführt werden. Danach können die Register mit den aktuellen Konfigurationswerten gefüllt werden.

5.3.2 Kalibrations-Routine

Die Kalibrationsroutine `cc1000_calibrate()` leitet die Kalibration des CC1000 ein und wartet bis diese beendet wird. Während der Kalibration wird der Energieverbrauch verringert, um unerwünschte Transmissionen im Sende-Modus zu vermeinden. Nachdem das Funkmodul mit einer Betriebsfrequenz initialisiert wurde (`cc1000_set(freq)`), wird die Kalibration des spannungsgeregelten Oszillators VCO durch Setzen des **CAL_START**-Bits im **CAL**-Register eingeleitet. VCO sollte gemäß dem Betriebsfrequenz-Modus (Sende- oder Empfangs-Modus) und der Ausgangsleitsung (**PA_POW**-Register) programmiert werden. Die VCO-Kalibration wird durch das **CAL**-Register (*Calibration-Control- und Status-Register*) kontrolliert. Das Ende der Kalibration wird durch das **CAL_COMPLETE**-Bit angezeigt. Die Kalibrationszeit ist von der internen Phasenregelschleife PLL-Frequenz abhän-

gig. Die niedrigste erlaubte Frequenz beträgt 1 MHz. Es ist möglich das **CAL_COMPLETE**-Bit abzufragen oder 34 ms zu warten (bei 1 MHz PLL-Frequenz) bis die Kalibration beendet ist. Das **CAL_START**-Bit muss auf Null gesetzt werden, sobald die VCO-Kalibration beendet ist. Das Funkmodul sollte im Sende-Modus und im Empfangs-Modus kalibriert sein. Nach der erfolgreichen Durchführung der `cc1000_reset(void)` und `cc1000_calibrate()` Routinen ist das CC1000 Modul betriebsbereit.

5.3.3 Initialisierung

Die `cc1000_init(freq)` Methode führt alle beim Initialisieren des Funkmoduls nötigen Operationen aus. Zuerst werden die Pins des Konfigurationsinterfaces konfiguriert. Danach werden alle Register des CC1000 Moduls auf die Standardwerte zurück gesetzt (**RESET_N**-Bit = 1 im **MAIN**-Register). Der Kristall-Oszillator wird eingeschaltet. Danach wird 2 ms lang abgewartet, bis sich der Oszillator stabilisiert hat. Jetzt werden alle Konfigurationsregister des CC1000 mit bestimmten Parametern für eine Betriebsfrequenz beschrieben. Anschließend wird die Kalibration (`cc1000_calibrate()`) durchgeführt und das Modul wird in den *RX*-Empfangsmodus gesetzt. Optional ist es möglich hier die Initialisierung der seriellen Schnittstelle (*UART*) und des *SPI*-Busses einzuschließen).

5.3.4 WakeUp-Routine

Während der *Wakeup-Routine* `cc1000_wakeup(void)`, bestehend aus `cc1000_on(void)`, wird das CC1000 Modul aus dem *Power-Down-Modus* aufgeweckt. Um den Energiekonsum niedrig zu halten wird nur der Kristall-Oszillator eingeschaltet. Danach muss 2 bis 5 ms lang abgewartet werden bis sich der Oszillator stabilisiert hat. Dann können die Übertragungsfunktionen des Funkmoduls eingestellt und benutzt werden. Das Modul wird am Ende der Methode in den Empfangsmodus versetzt.

5.3.5 Transmit-, Receive-, PowerDown- und Sleep-Modus

Die `cc1000_mode(mode)` Methode ermöglicht die Einstellung des Funkmoduls in vier Modi: *TX*-Sende-Modus, *RX*-Empfangs-Modus, *PD*-PowerDown-Modus und *SLEEP*-Modus. Das Umschalten zwischen *TX*- und *RX*-Modus wird durch das Programmieren der **MAIN**-, **PLL**- und **CURRENT**-Register eingeleitet. Der *SLEEP*-Modus unterscheidet sich vom *PD*-PowerDown-Modus in der Empfangsbereitschaft. Im *PD*-PowerDown-Modus müssen alle Bits im **PA_POW**-Register auf Null gesetzt werden, so dass der Energieverbrauch minimiert wird. In diesem Zustand kann das Funkmodul nichts empfangen.

5.4 Datenkommunikation zwischen CC1000 und ATmega128L

Bevor die SPI-Datenleitung zwischen CC1000 und ATmega128L benutzt werden kann, wird sie initialisiert. Die Initialisierungsmethode besteht aus der Einstellung der zur Datenübertragung benutzten Ausgänge der Pins und der Einstellung des SPI-Kontrollregisters. Der *Port B* hat 8 Anschlüsse, von welchen vier durch die SPI-Leitung benutzt werden. Es sind *PB0*-Pin (**$\overline{\text{SS}}$**), *PB1*-Pin (**SCK** – *Bus-Serial-Clock*), *PB2*-Pin (**MOSI** – *SPI Bus Master Output/Slave Input*), *PB3*-Pin (**MISO** – *SPI Bus Master Input/Slave Output*). Diese vier

Pins werden durch setzen des **DDRB**-Registers (Richtungsregister für *Port B*) und der Pins **DDB0**,**DDB1** und **DDB2** als Eingänge definiert (auf logisch 0 gesetzt). Das **DDB3**-Register wird als Ausgang gesetzt (logisch 1). Als nächstes wird das Kontrollregister **SPCR** gesetzt. Hier werden Polarität **CPOL** und *„Clock-Phase“* **CPHA** auf logisch 0 eingestellt (näheres in Kapitel 3.2.2) und SPI-Interface **SPE** sowie SPI-Interrupt **SPIE** auf logisch 1 gesetzt und somit eingeschaltet. Am Ende der Initialisierung wird das CC1000 Funkmodul in den Empfangsmodus versetzt (cc1000_mode(CC1000_MODE_RX)).

```
char SPI_receive(void)
{
/*wait for reception complete*/
  while (!(SPSR & (1 << SPIF)));
  data_r = SPDR;
  return data_r;
}
```

```
void SPI_transmit(char data_t)
{
  SPDR = data_t;

/*wait for transmission complete*/
  while (!(SPSR & (1 << SPIF)));
}
```

Abbildung 18: Eine Sende- und Empfangsoperation durch das SPI-Interface in C.

Wenn ein Byte in das **SPDR**-Register (SPI-Datenregister) geschoben wird, wird die Übertragung eingeleitet. Davor muss sich das Funkmodul im Übertragungsmodus befinden (cc1000_mode(CC1000_MODE_TX)). Das **SPDR**-Datenregister kann jederzeit auf eingehende Daten abgefragt werden. Dies ermöglicht das **SPSR**-Register (SPI-Statusregister). Das *Most-Significant-Bit* **SPIF** wird auf Eins gesetzt, wenn sich eingehende Daten im **SPDR**-Register befinden. Anderenfalls wird **SPIF** Null sein. Die Datensynchronisation wird von der Seite des CC1000 mit dem „Clock“ **DCLK** am „Clock-Eingang“ des Mikrocontroller **SCK** bereitgestellt. Der *SPI-Clock-Interrupt* wird jede 416 μsec (8 Bits) ausgelöst. Während des Interrupts können dann die empfangenen Daten ausgelesen werden.

5.5 Serielle Schnittstelle RS232

Die serielle Schnittstelle RS232 wurde ausschließlich zu Debugging-Zwecken aufgebaut. Diese benutzt das USB-Interface des MIB520CB Gateways für Mica2-Plattform. Als Interface wurde das Programm **Docklight**©verwendet. Die empfangenen Daten können im ASCII-, HEX-, Decimal- und Binary-Format ausgegeben werden. Die benötigten Einstellungen sind Baudrate, Parität sowie die Anzahl der Daten- und Stop-Bits. Nachdem diese

Werte an jene im eigenen Programm angepasst werden, können die gesendeten Daten am PC empfangen werden.

Zuerst muss die „USART“-Schnittstelle (Universal-Synchron/Asynchron-Receiver/Transmitter) des ATmega128L initialisiert werden. Dafür werden die Kontroll- und Statusregister **UCSR0A, UCSR0B, UCSR0C** und die Baudrateregister **UBRR0, UBRR0L, UBRR0H** benutzt. Die Baudrate und der Wert für das *Baud-Rate-Register* **UBRR0** muss anhand des aktuellen CPU-Taktes berechnet werden. Die Berechnung ist vom angewandten Übertragungsmodus abhängig. In diesem Fall ist es der asynchrone, normale Modus (das erste Bit[1] (U2X0) im USART Kontroll- und Statusregister **UCSR0A** muss gleich 0 sein). Die Formeln für **Baudrate**- und **UBRR**-Register in diesem Modus sehen wie folgt aus: $BAUD = \frac{f_{osc}}{16(\mathbf{UBRR}+1)}$ und $\mathbf{UBRR} = \frac{f_{osc}}{16BAUD} - 1$ wobei f_{osc} den CPU-Takt beschreibt [1]. Danach müssen die **UBRR0L** (Einstellung der „low“ Baudrate) und **UBRR0H** (Einstellung der „high“ Baudrate) gemäß **UBRR0**-Register gesetzt werden. Als nächstes werden USART-Empfangsmodus, USART-Transmitmodus, Empfangs- und Transmit-Interrupt im **UCSR0B**-Register eingeschaltet (**TXEN0, RXEN0, TXCIE0** und **RXCIE0** = 1). Zuletzt wird das **UCSR0C**-Register eingestellt. Hier wird die Parität und die Anzahl der Daten- und Stop-Bits festgelegt. Wenn die USART-Schnittstelle mit obigen Werten initialisiert worden ist, können die zu sendenden Daten durch das **UDR0**-Datenregister gesendet und empfangen werden.

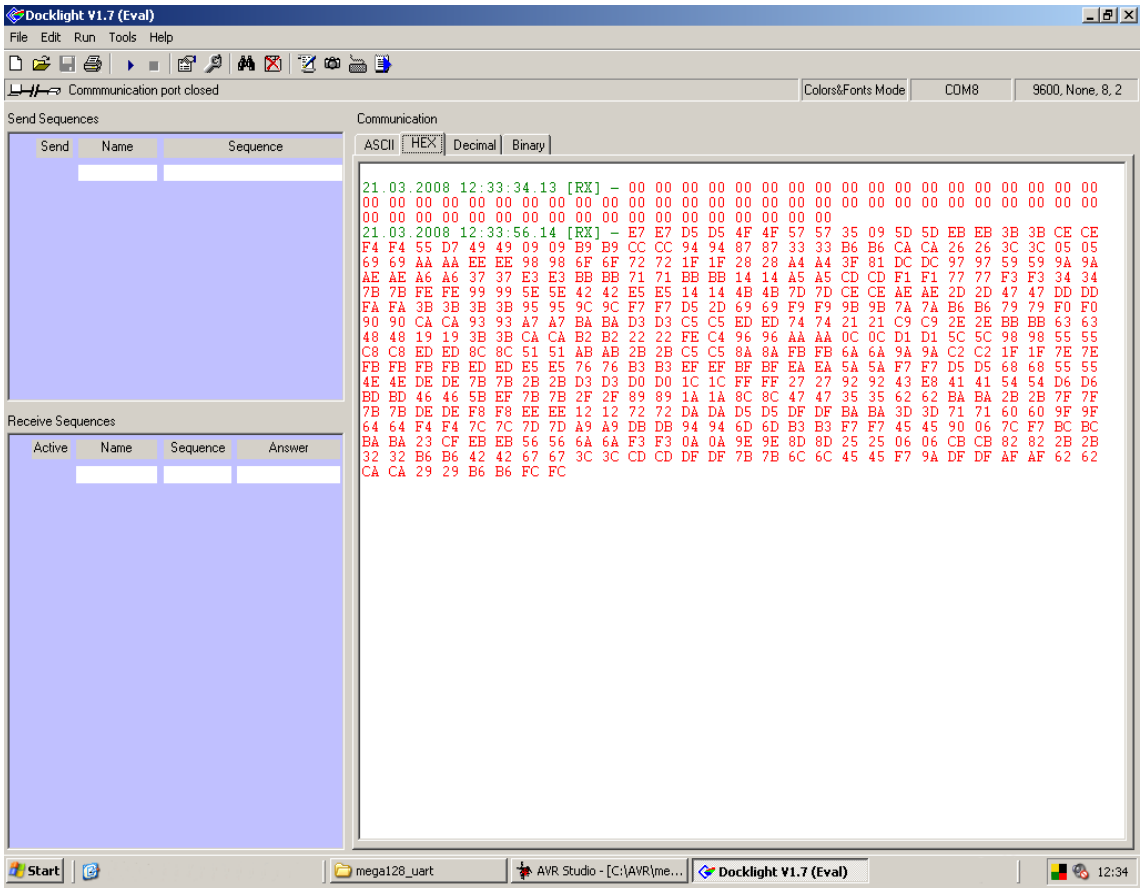

Abbildung 19: Das **Docklight**©Interface.

5.6 Received-Signal-Strength-Indication-Ausgang (*RSSI*)

Das CC1000 Funkmodul verfügt über einen analogen RSSI-Ausgang, welcher zum Messen des aktuellen Rauschniveaus verwendet wird. Dieser, wie im Kapitel 4.4 beschrieben wurde, ist mit einem **ADC**-Pin (Analog/Digital-Wandler) am Mikrocontroller verbunden. Als erstes muss das ADC-Interface ausgeschaltet werden. Dies erfolgt wenn das siebte Bit im **ADCSRA**[7]-Register (Kontroll- und Status-Register) auf 0 gesetzt wird. Folglich kann der *„Prescaler“* eingestellt werden. Durch die Einstellung der *„Prescaler“* (**ADPS2**-, **ADPS1**- und **ADPS0**-Pin im **ADCSRA**-Register) wird der Teilungs-Faktor zwischen Quarz-Oscillator-Frequenz und Eingangsfrequenz vom RSSI-Ausgang [1] festgelegt. Für Bedürfnisse dieser Arbeit wurde der Teilungs-Faktor zwei gewählt (**ADPS2**=0, **ADPS1**=0 und **ADPS0**=0). Anschließend kann das ADC-Interface durch das Setzen des **ADEN**-, **ADIE**-, **ADIF**- und **ADSC**-Pins im **ADCSRA**-Register auf logisch 1, eingeschaltet werden. Der **ADEN**-Pin (**ADCSRA**[7] – das siebte Bit im ADC-Status-Register) schaltet das ADC-Interface ein. Der nächste **ADIE**-Pin (**ADCSRA**[3] – das dritte Bit im ADC-Status-Register) aktiviert den *ADC-Conversion-Complete-Interrupt*. Der **ADIF**-Pin (**ADCSRA**[4]) ist ein *Interrupt-Flag* für den **ADIE**-Pin. Der letzte **ADSC**-Pin (**ADCSRA**[6]) leitet die Konvertierung ein.

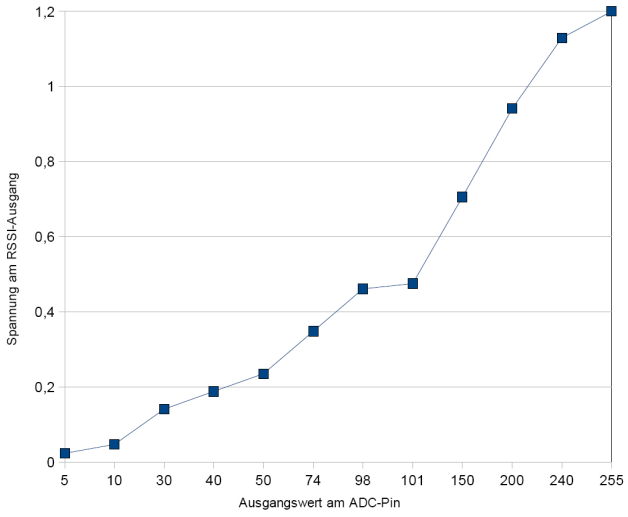

Abbildung 20: Das Verhältnis zwischen dem Wert am digitalen ADC-Ausgang des Mikrocontrollers und der Spannung am RSSI-Ausgang des CC1000 Funkmoduls.

Der RSSI-Spannungswert am ADC-Ausgang wird innerhalb des ADC-Interrupts ausgelesen und in einer Variable gespeichert. Nachdem der **ADSC**-Pin (**ADCSRA**[6]=1) auf logisch 1 gesetzt wurde, wird gewartet bis sich der **ADSC**-Pin auf logisch 0 ändert. Das bedeutet, dass der Interrupt zu Ende ist und der ADC-Wert ausgelesen werden kann. Darauf wird das ADC-Interface ausgeschaltet. Der ADC-Wert befindet sich jetzt in der Variable, die ausgegeben werden kann [1].

Die Werte, die man in digitaler 8-Bit Form am ADC-Ausgang bekommt, können leicht in die dBm-Form umgerechnet werden. Der RSSI-Spannungs-Bereich befindet sich zwischen

0 und 1,2 Volt [4]. Für diesen Spannungsbereich gibt es 255 Schritt-Werte (ohne Null), die am ADC-Ausgang ausgelesen werden können. So ergibt sich durch Umrechnen $A = \frac{1.2}{255} = 0,0047058824$. Wenn der ADC-Ausgangs-Wert mit diesem Wert multipliziert wird, ergibt sich der entsprechende RSSI-Spannungs-Wert. Mit diesem RSSI-Wert kann nun das Dämpfungsmaß in dBm aus der Tabelle 16 im Kapitel 4.4 ausgelesen werden.

5.7 Die Verbindungs-Schicht und die physikalische Schicht

Im weiteren Teil werden die höheren Schichten der Treiberarchitektur beschrieben. Es handelt sich um die physikalische, Verbindungs- und die MAC-Schicht. Diese werden auf der Kernbasis, welche die Hardware-Steuerung bedient, aufgebaut.

Paketrahmenformat: ein einzelnes Datenpaket besteht aus einem festgelegten Informations-Rahmen. Dieser kann sich je nach Netzwerk-Standard im Paket-Format unterscheiden. Zu den wichtigsten Bestandteilen eines Daten-Rahmens gehört die Präambel. Wie schon in Kapitel 4.1 angesprochen, wird diese beim *Data-Slicer* zur Bestimmung des Niveaus der Abgleichs-Schwelle für den *„Average-Filter“* benutzt. Die Länge der Präambel ist von der verwendeten Kodierung und der Empfindlichkeit des Moduls abhängig. Nach der Präambel sollte ein Synchronisations-Byte gesendet werden. Dieses Byte signalisiert den Anfang des gesamten Rahmens. Das Synchronisations-Byte wird auch zur Synchronisierung der empfangenen Daten gebraucht. Nach dem Synchronisations-Byte wird das Empfangsbestätigungs-Byte (Ack-acknowledgement) gesendet. Dieses wird als Empfangsbestätigung beim Sender erwartet. Daraufhin wird die CRC-Summe, die für die Überprüfung der Korrektheit der empfangenen Daten verantwortlich ist, übertragen. Im Anschluß werden die Datengröße und die Paketdaten, die bis zu 30 Byte groß sein können, gesendet.

Päambel	SYNC	ACK	CRC	Size	Data
32 Byte	1 Byte	1 Byte	1 Byte	1 Byte	0-30 Byte

Abbildung 21: Das Datenpaket-Format eines einzelnen Informations-Rahmens.

5.8 Deterministischer, endlicher Zustandsautomat

Der Algorithmus zum Senden und Empfangen drahtlos versendeter Daten kann durch einen endlichen Zustandsautomaten modelliert werden. Zu den Hauptzuständen des Zustandsautomaten gehören:

- Empfangs-Zustände:
  1. `RX_IDLE`,
  2. `RX_SYNC`,
  3. `RX_ACK`,
  4. `RX_CRC`,
  5. `RX_SIZE`,
  6. `RX_DATA`,
  7. `RX_TX_ACK`
- Sende-Zustände:
  1. `TX_PRE`,
  2. `TX_ACK`,
  3. `TX_CRC`,
  4. `TX_SIZE`,
  5. `TX_DATA`,
  6. `TX_DONE`,
  7. `TX_WAIT_FOR_ACK`

Der endliche Zustandsautomat wird mit der Methode `cc1000_state()` innerhalb des SPI-Interrupts aufgerufen. Der Methoden-Status ändert sich, wenn eine bestimmte Datenfolge im **SPDR** SPI-Datenregister empfangen wird. So wird die `cc1000_state()`-Methode solange im Zustand `RX_IDLE` aufgerufen, bis eine Byte-Folge einer festgelegten Länge aus nacheinanderfolgenden Präambel-Bytes empfangen wird.

5.8.1 Senden mit dem Funkmodul

Das Senden der Daten mit dem SPI-Bus ist sehr zeitabhängig. Der SPI-Ausgang ist *„Single-Buffered“*, was dazu führt, dass auf jegliche Verzögerung zwischen Interrupt-Antritt und neuer Beschreibung des SPI-Datenregisters (**SPDR**) geachtet werden muss. Eine Abweichung der SPI-Sende-Routine von 25 μs kann dazu führen, dass die neue Schreibe-Operation auf dem SPI-Datenregister (**SPDR**) ignoriert wird. Stattdessen wird der alte Inhalt des Registers versandt. Die doppelt versendeten Daten-Bytes verursachen inkorrekte Datentransmissionen.

Das Senden wird mit Hilfe eines **SCK**-Pins umgesetzt. Der **SCK**-Pin löst einen Interrupt am Mikrocontroller aus. Innerhalb des Interrupts werden Daten versendet. Eine Sende-Routine versetzt zuerst das CC1000 Modul in den Sende-Modus. Danach wird der aktuelle Zustand des endlichen Automaten ***„State-Machine“*** mit `state = TX_PRE` initialisiert. In diesem Zustand wird ein Präambel-Byte in das **SPDR**-Register geschrieben und

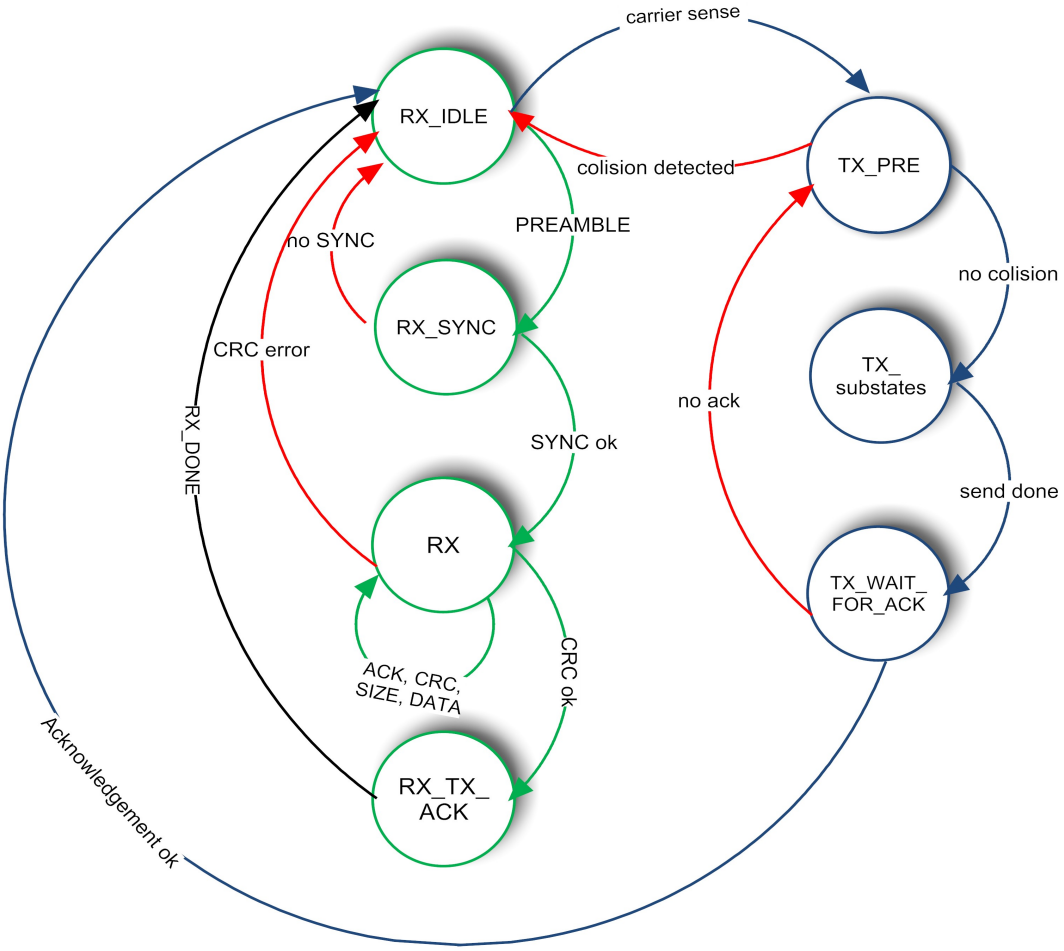

Abbildung 22: Zustandsdiagramm des endlichen Zustandsautomaten.

der Präambel-Byte-Zähler um Eins erhöht. Der `TX_PRE`-Zustand wird so oft aufgerufen, bis die festgelegte Anzahl der Präambel-Bytes versendet wurde. Beim Senden ist es erforderlich, das erste Byte manuell in das SPI-Datenregister **SPDR** zu schieben, damit ein Interrupt am Mikrocontroller ausgelöst wird und der endliche Zustandsautomat dadurch eingeschaltet wird. Das erste Byte (`0xAA`) wird somit in das SPI-Datenregister geschoben und der Präambel-Zähler auf Eins gesetzt.

Wenn der endliche Zustandsautomat eingeschaltet wird, wird er Byte-weise den kompletten Paketrahmen versenden. Zuerst wird die vollständige Präambel versendet. Nach dem letzten Präambel-Byte wird das Synchronisations-Byte in das SPI-Datenregister geschoben (Zustand: `TX_PRE`). Darauf folgend werden nacheinander das Bestätigungs-Byte (Zustand: `TX_ACK`), die CRC-Summe (Zustand: `TX_CRC`) und die Größe der zu versendenden Daten (Zustand: `TX_SIZE`) gesendet. Im nächsten Schritt werden die Daten übertragen (Zustand: `TX_DATA`). Anschließend wird ein sogenanntes „Flush-Byte“ versendet, welches das Ende des Paketrahmens signalisiert (Zustand: `TX_DONE`). Schließlich setzt sich der Sender in den Empfangs-Modus und wartet auf eine Empfangsbestätigung (`ACK_BYTE`, *„Acknowledgement“* engl. die Bestätigung) seines Übertragungspartners (Zustand: `TX_WAIT_FOR_ACK`). Sollte innerhalb eines festen Zeitrahmens keine Empfangs-

bestätigung beim Sender ankommen, wird das gesamte Datenpaket erneut versendet.

5.8.2 Empfangen mit dem Funkmodul

Auch das Empfangen der Daten durch das CC1000 Funkmodul erfolgt mit Hilfe eines **SCK**-Pins. Der **SCK**-Pin löst einen Interrupt am Mikrocontroller aus. Innerhalb des Interrupts werden Daten empfangen.

Im Empfangsmodus wird der endliche Zustandsautomat ***„State-Machine“*** im `(state = RX_IDLE)` initialisiert. In diesem Zustand wird jedes Mal, wenn ein SPI-Interrupt auftritt, der SPI-Datenregister **SPDR** abgefragt. Wenn das gerade empfangene Byte gleich der gesendeten Präambel ist (`0xAA` oder `0x55` falls die empfangene Byte-Folge verschoben ist), wird ein Präambel-Zähler inkrementiert. Falls das neu empfangene Byte sich von dem Päambel-Byte unterscheidet, wird geprüft, ob eine festgelegte Anzahl an aufeinander folgenden Präambel-Bytes bereits gelesen worden ist (oft wird etwa $\frac{1}{4}$ der gesendeten Präambel gezählt). Falls der Präambel-Zähler hoch genug ist, wurde die Präambel richtig erkannt, und der neue Zustand kann nun gesetzt werden. Im `(state = RX_SYNC)` Zustand wird zunächst nach dem Synchronisations-Byte gesucht. Da die empfangene Datenfolge beliebig verschoben werden kann, wird es oft nötig, innerhalb von zwei Bytes nach dem Synchronisations-Byte zu suchen. Das erste Byte, das sich vom Präambel-Byte unterscheidet wird in einer Variable gespeichert und mit dem nächten Byte, das empfangen wird zu einer 16 Bit-Folge zusammengesetzt. Solange das aktuelle Byte ungleich dem Synchronisations-Byte (`0x33` - **00110011**) ist, wird das Byte um eine Stelle nach links verschoben. Die Anzahl der Stellen um die das Byte verschoben wurde, wird als *offset* mitgezählt. Angenommen das erste empfangene Byte, das ungleich Präambel-Byte ist, ist gleich `0x51` - **01010001**. Das nächste Byte, das empfangen wurde ist gleich `0x9E` - **10011110**. Da das erste Byte ungleich dem Synchronisations-Byte (`0x33`) ist, muss angenommen werden, dass das Synchronisations-Byte sich zwischen dem aktuellen und dem nächsten Byte befindet.

Die zwei empfangenen Bytes (`0x51` und `0x9E`) werden nun zu einer 16 Bit-Folge zusammengesetzt:

1. 0**1010001.1**0011110
2. 01**010001.10**011110
3. 010**10001.100**11110
4. 0101**0001.1001**1110
5. 01010**001.10011**110 = `0x33`

Das Synchronisations-Byte wurde nach fünf Verschiebungen gefunden. Der *Offset*-Wert beträgt damit fünf, und wird für den weiteren Verlauf der Empfangs-Routine übernommen.

Nachdem das Synchronisations-Byte gefunden wurde, sind die zu empfangenden Daten synchronisiert und können nun richtig empfangen werden. Nachdem das Datenpaket vollständig empfangen wurde, wird die CRC-Summe geprüft. Über die CRC-Summe im Paket soll überprüft werden, ob es zu Störungen gekommen ist.

5.9 Berkeley Media Access Control *B-MAC*

Die Berkeley-Medienzugriffskontrolle wurde 2004 von Joseph Polastre, Jason Hill und David Culle an der *University of California, Berkeley* entwickelt. Das **B-MAC**-Protokoll beschäftigt sich mit der Trägerprüfung (*Clear-Channel-Assessment*, eng. „Beurteilung der Kanalfreiheit“) und beschränkt sich dabei nur auf einen kleinen Tei der **MAC** Medienzugriffskontrolle-Schicht. **B-MAC** ist also nur ein Verbindungs-Protokoll auf dem die weiteren *„Netzwerk-Services“* wie die Organization, Synchronization und Routing aufgebaut werden können.

Die **B-MAC**-Eigenschaften:

1. Kollisionsvermeidung :
   - CCA Clear Channel Assessment
   - Backoff
2. Zuverlässigkeit :
   - Link-Layer Acknowledgement
3. Energieeffizienz :
   - LPL Low Power Listening
   - Adaptives Preamble Sampling (Präambel-Länge)
4. Einfaches Design und geringe Grösse

5.10 *B-MAC* Design und Implementation

Im folgenden Kapitel werden die Bestandteile des *B-MAC*-Protokolls besprochen. Ferner wird die Implemetation des *CCA*-Algorithmus (Clear-Channel-Assessment) und der Empfangsbestätigung behandelt. Die im weiteren Teil des Kapitels besprochene Backoff- und Low-Power-Listening-Funktion wird nur teilweise umgesetzt.

5.10.1 *CCA* Clear-Channel-Assessment

Das **B-MAC**-Protokoll basiert auf einem Rauschunterdrückungsverfahren. Ein Rauschpegel ist in der drahtlosen Kommunikation allgegenwärtig. Dieser steigt jedoch erheblich an, wenn in der Umgebung eines Sensor-Knotens zur gleichen Zeit eine Datenübertragung stattfindet. Dieses Rauschen wird durch das **B-MAC**-Protokoll überwacht und ausgewertet. Da das aktuelle Rauschen sich mit der Umgebung und den Wetterbedingungen ändern kann, ist es nötig, den Rauschgrenzwert an das neue Rauschniveau anzupassen. Aus diesem Grund wird der aktuelle Rauschpegel ständig aktualisiert.

Die Abbildung 23 stellt den Pegelniveau-Unterschied beim Senden eines einzelnen Datenpakets dar. Das oberste Diagramm zeigt eine Aufnahme der Empfangsfeldstärke aus dem

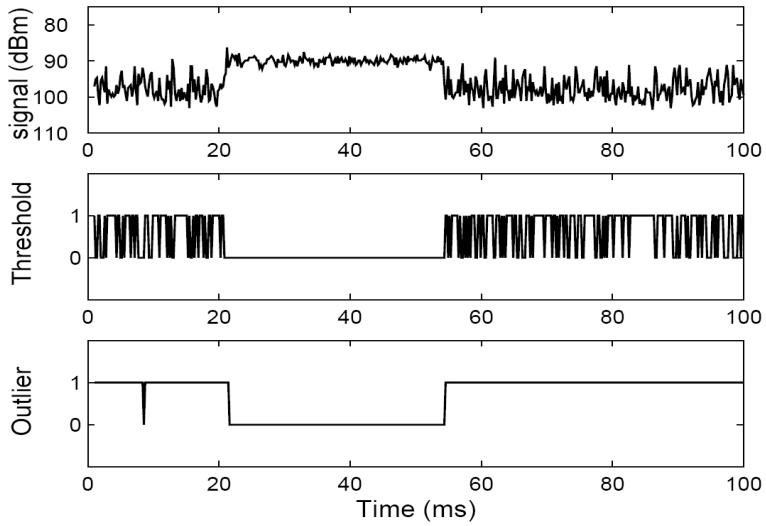

Abbildung 23: Rauschniveau-Unterschied zwischen Transmission eines einzelnen Datenpakets und freiem Umgebungsrauschen. *Quelle: [7]*

RSSI-Ausgang des Funkmoduls. Das mittlere Diagram zeigt den durch den Grenzwert des CCAs *Clear-Channel-Assessment* abgeschnittenen Rauschpegel. Dabei muss beachtet werden, dass Ausgangssignal (die Spannung) und Signalstärke am RSSI-Ausgang invers proportional sind. Das heißt, je höher die Spannung, desto schwächer ist das Signal am RSSI/IF-Pin. Das letzte Diagramm zeigt die Entscheidungsrichtlinie des CCA-Algorithmus. Eine 0 bedeutet der Kanal ist besetzt, eine 1 steht für einen freien Kanal. Eine Sende-Operation eines einzelnen Daten-Pakets der Größe 36 Bytes mit dem CC1000 Funkmodul nimmt zwischen 22 und 54 ms [7] in Anspruch.

Der Algorithmus prüft fünfmal den Kanal ab. Das dabei erfasste Rauschniveau wird in einer FIFO-Schlange gespeichert. Wie in Abbildung 23 zu sehen ist, hat eine Transmission ein konstantes Rauschniveau, während beim Rauschen eine große Streuung der Werte vorhanden ist. B-MAC sucht in dem Probevolumen nach Ausreißern, die unterhalb des aktuellen Grenzwertes liegen. Befinden sich im Puffer Proben, die sich von den anderen derartig unterscheiden, bedeutet dies, dass der Kanal aktuell frei ist. Es ist nämlich nicht möglich, bei einem gültigen Datenpaket einen Ausreißer unterhalb des aktuellen Grenzwertes zu erhalten. Gibt es nach einer fünfmaligen Abfrage des Kanals keine auffälligen Ausreißer in dem Probevolumen, gilt der Kanal als besetzt.

Nach jeder Sende-Operation, wenn der Kanal frei ist, oder keine gültigen Daten im Empfangs-Modus empfangen werden, wird die Signalstärke am RSSI-Ausgang des CC1000 erfasst und mit dem derzeitigen Rauschniveau verglichen. Für die Robustheit der Berechnung des neuen Rauschniveaus wird zuerst der Median der Proben in der FIFO-Schlange berechnet. Anschließend wird durch den exponentiell gewichteten Durchschnitts-Grenzwert mit dem α Koeffizient das neue Rauschniveau aktualisiert. Die besten Ergebnisse werden mit $\alpha = 0.06$ und einer FIFO-Schlange der Länge $n = 10$ erzielt [7] (siehe Algorithmus 1 und Algorithmus 2).

Algorithmus 1

- **Gegeben:** die Größe der FIFO-Schlange, n, und der exponentiell gewichtete Durchschnitts-Grenzwert mit dem α Koeffizient = x
- **Ergebniss:** noise_level - das aktuelle Rauschniveau

```
1  void update_noise_value (char value)
2  {
3    q = new FIFO queue of size n;
4
5    while(radio ist on)
6    {
7      if(end of packet transmission or timer is fired)
8      {
9        if(radio is idle)
10       {
11         data = value;
12         dequeue(q);
13         enqueue(q ,data);
14         m = median(q);
15         noise_level = noise_level (1 - x)  + x m;
16       }
17     }
18   }
19 }
```

Algorithmus 2

- **Gegeben:** das aktuelle Rauschniveau (noise_level)
- **Ergebniss:** true, wenn der Kanal frei ist, false sonst

```
1  char check_noise_value(void)
2  {
3    rssi_on();
4
5    for(i = 0; i<s; i++)
6    {
7      data = RSSI;
8      if(data > noise_level)
9      {
10       update_noise_value(data);
11       return true;
12     }
13   }
14   rssi_off();
15   return false;
16 }
```

5.10.2 Backoff

Nachdem ein Knoten den Sende-Kanal als besetzt erkannt hat, muss er eine bestimmte Zeitspanne (*Backoff-Zeit*) abwarten, bis eine neue Kanal-Prüfung erfolgen kann. Die *Backoff-Zeit* sichert den nötigen Grad an Kollisionsfreiheit. Die *Init-Backoff-Zeit* wird benutzt, falls der Kanal bei der ersten Prüfung belegt ist. Wenn die *Init-Backoff-Zeit* abgelaufen ist, wird die CCA (*Clear-Channel-Assessment*) erneut eingeleitet. Sollte der Sende-Kanal bei der zweiten Abfrage immer noch besetzt sein, wird eine längere *Con-Backoff-Zeit* (congestion, eng. Stauung) verwendet.

5.10.3 *ACK* Link-Layer-Acknowledgement

Unmittelbar nach dem Empfang eines einzelnen Datenpakets wird eine Bestätigung (ACK-Byte) vom Empfänger zurück zum Sender gesendet. Zuerst muss der Empfänger vom Empfangs-Modus in den Sende-Modus wechseln. Dieser Vorgang nimmt etwa 250 μs in Anspruch. Danach wird eine kurze Präambel, die aus 4 Bytes (eine Hexadezimaldarstellung eines Präambel-Bytes: 0xAA) besteht, gesendet. Im nächsten Schritt werden zwei ACK-Bytes übertragen (eine Hexadezimaldarstellung eines ACK [2] -Bytes: 0x66). Das versenden eines Bytes dauert genau 416 μs. Jede 416 μs wird ein Interrupt ausgelöst, währenddessen acht Bits empfangen oder gesendet werden (gleichzeitiges Empfangen und Senden ist möglich, vgl. Kapitel 3.2.1). Der Sender muss dagegen nur vom Sende-Modus auf Empfangs-Modus wechseln und auf eine kurze Präambel warten. Falls diese erkannt wird, sollte nach dem *ACK*-Byte gesucht werden, indem wie im Fall des *SYNC*-Byte (vgl. Kapitel 5.8.2) je zwei nacheinander folgende Bytes verschoben werden.

Die Gesamtzeit, die der Empfänger benötigt um eine *ACK*-Bestätigung zu senden, beträgt $250\ \mu s + 6 \cdot 416\ \mu s = 2746\ \mu s$= 2,746 ms. Die Wartezeit des Senders wurde anfangs mit 2,746 *ms* initialisiert. Dies entspricht 6 Empfangs-Interrupts währenddessen 6 Bytes empfangen werden können. Demzufolge sollte die Zeitschranke groß genug sein, um eine verlässliche *ACK*-Bestätigungsübertragung zu gewährleisten.

5.10.4 *LPL* Low-Power-Listening

Da einer der wichtigsten Ziele der B-MAC-Architektur die Energieerhaltung ist, ist es nötig die Zeit, in der ein Empfänger sich im Leerlauf befindet, zu minimieren. *LPL* (*Low-Power-Listening*) nutzt den Schlaf-Modus des CC1000 Funkmoduls um die Energieersparnis zu maximieren. Das Funkmodul wird nur in festen Zeitabständen geweckt. Zu diesem Zeitpunkt überprüft das *CCA* den Kanal nach Funkaktivitäten. Sollten keine Funkaktivitäten erkannt werden, wird das Funkmodul in das Schlaf-Modus versetzt. Andernfalls wird das Funkmodul in Empfangs-Modus gesetzt und bleibt nur solange „wach“, wie es nötig ist, um ein einzelnes Datenpaket vollständig zu empfangen. Danach wird das CC1000 Funkmodul erneut in den Schlaf-Modus versetzt. Wenn eine Aktivität *falsch-positiv* identifiziert wird und demzufoge kein gültiges Datenpaket empfangen wird, wird das Modul nach einem *Timeout* wieder in den Schlaf-Modus versetzt. Die Anzahl der *falsch-positiven*-CCA-Entscheidungen ist kritisch für einen energiesparenden Betrieb. Zu viele verursachen einen Leerlauf beim Empfänger, der anstatt gültigen Daten nur das Rauschen empfängt und damit eigene Energie verschwendet.

[2] ACK – eine Abkürzung für Acknowledgement eng. Bestätigung

Da das Funkmodul nur in gewissen Zeitabständen den Kanal abfragt, sollte es sicher sein, dass es auch innerhalb dieser Zeit eine Funk-Aktivität erkennt, wenn eine solche stattfindet. Wenn der Kanal jede 100 ms geprüft wird, muss das Datenpaket mit einer genügend langen Präambel anfangen (minimum 100 ms), damit die Paketpräambel die „Schlafzeit“ des Moduls überlappen kann.

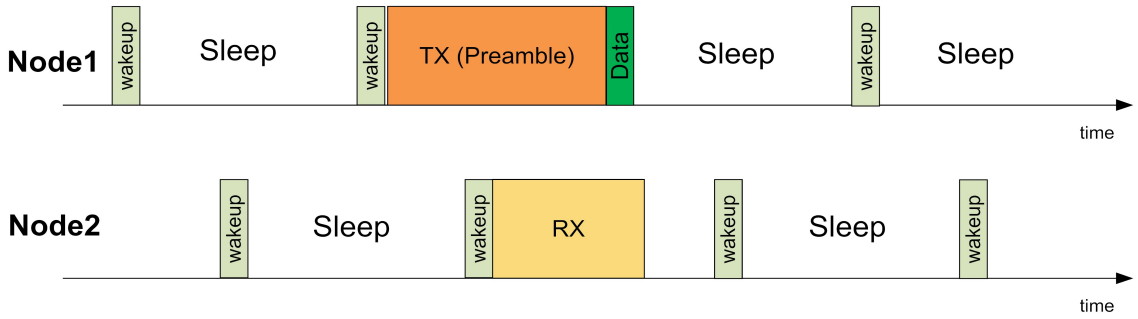

Abbildung 24: Die „Wakeup“-Zyklen beim Low-Power-Listening.

6 Experimente

In diesem Abschnitt werden mehrere Experimente und vergleichende Auswertungen der Übetragungstests beschrieben. Das Ziel der Testversuche ist die Feinabstimmung der zeitkritischen Elemente, die für die Effektivität der Übertragung-Routine verantwortlich sind.

6.1 Test 1: Senden und Empfangen ohne Bestätigung

Bei den ersten Sende-Versuchen von hundert Datenpaketen von einem Sender-Knoten zu einem 1 Meter entfernten Empfänger-Knoten hat sich ein relativ großer Prozentsatz an empfangenen Nachrichten bemerkbar gemacht. Während jedes Versuchs kamen beim Empfänger zwischen 90% und 100% der gesendeten Nachrichten an. Um eine aussagekräftige Größe für den Paketverlust zu bekommen, wurden die ersten Übertragungsversuche mit einer ausgeschaltenen Empfangsbestätigung und Retransmissions-Routine durchgeführt.

6.2 Test 2: Empfangbestätigung und Retransmission

Während der nächsten Übertragungsversuche sollte eine optimale Zeitschranke für die Empfangsbestätigung-Routine gefunden werden. Die Bestätigungswartezeit ist ein wichtiger Faktor für eine effektive und zuverlässige Übertragung einer Antwort zum Sender. Der Sender überträgt zuerst ein einziges Dantenpaket zum Empfänger. Falls der Sender keine Bestätigung empfangen hat, beginnt er eine Retransmission des Pakets. Während des Tests

Zeit	TX	RX	Retransmission	Rausch *ACK*	Paketverlust
2,746 ms	1100	2547	1457	0	10
3,578 ms	1100	1074	53	26	79
5,242 ms	1100	1080	23	20	43
10,234 ms	1100	1069	32	35	63

Tabelle 5: Ergebnisse um den Einfluß der Länge der Empfangsbestätigung-Wartezeitschranke auf die Effizienz zu untersuchen. Das *TX* steht für die Anzahl der bestätigten Paketübertragungen, das *RX* ist die Anzahl der vollständig beim Empfänger angekommenen Pakete.

wurden für jede der vier Zeitschranken 1100 Nachrichten übertragen. Anfangs wurde die Wartezeit für eine Empfangsbestätigung im `TX_WAIT_FOR_ACK`-Zustand mit 250 $\mu s + 6 \cdot 416\ \mu s = 2746\ \mu s = 2{,}746\ ms$ initialisiert. Es ist genau die selbe Zeit, die der Empfänger für die Umschaltung zum Sende-Modus und eine Übertragung einer aus sechs Bytes bestehender Bestätigung braucht. Beim ersten Versuch betrug die Menge der tatsächlich beim Empfänger ankommenden Nachrichten bis zu 200% der bestätigten Transmissionen. Wenn man sich die Menge der vollständig empfangenen Nachrichten genauer anschaut, kommt man schnell zu dem Schluß, dass die Ursache in der zu knapp gelegten Bestätigungswartezeit liegen muss. Die Tabelle 5 zeigt ein Verhältnis zwischen der Menge an Paketen, die erfolgreich mit einer Bestätigung gesendet wurden, der Anzahl der empfangenen Pakete und der Zahl der Pakete die nochmal gesendet werden mussten mit vier verschiedenen Bestätigungswartezeiten.

Für die Bestimmung der optimalen Wartezeit auf die Empfangsbestätigung wurden vier Tests mit folgender Wartezeiten durchgeführt:

1. $250\ \mu s + 6 \cdot 416\ \mu s = 2746\ \mu s = 2{,}746\ ms$
2. $250\ \mu s + 8 \cdot 416\ \mu s = 5242\ \mu s = 3{,}578\ ms$
3. $250\ \mu s + 12 \cdot 416\ \mu s = 5242\ \mu s = 5{,}242\ ms$
4. $250\ \mu s + 24 \cdot 416\ \mu s = 5242\ \mu s = 10{,}234\ ms$

In der Tabelle 5 ist direkt zu sehen, dass die Wartezeit von 2,746 *ms* zu kurz ist, um jede Empfangsbestätigung zuverlässig zu empfangen. Die besten Ergebnisse wurden mit der Zeitschranke mit der Dauer 3,578 *ms* bis 5,242 *ms* erzielt, was 8 bis 12 SPI-Interrupt-Schritten inklusive TX-RX Umschaltzeit entspricht. Demzufolge werden sich kommende Tests auf einen Vergleich zwischen den zwei Zeitschranken unter unterschiedlichen Bedingungen beschränken. Man sollte bedenken, dass je größer die Wartezeitschranke für eine Empfangsbestätigung ist, desto höher ist die Wahrscheinlichkeit, eine *Rauschen-Bestätigung* zu empfangen.

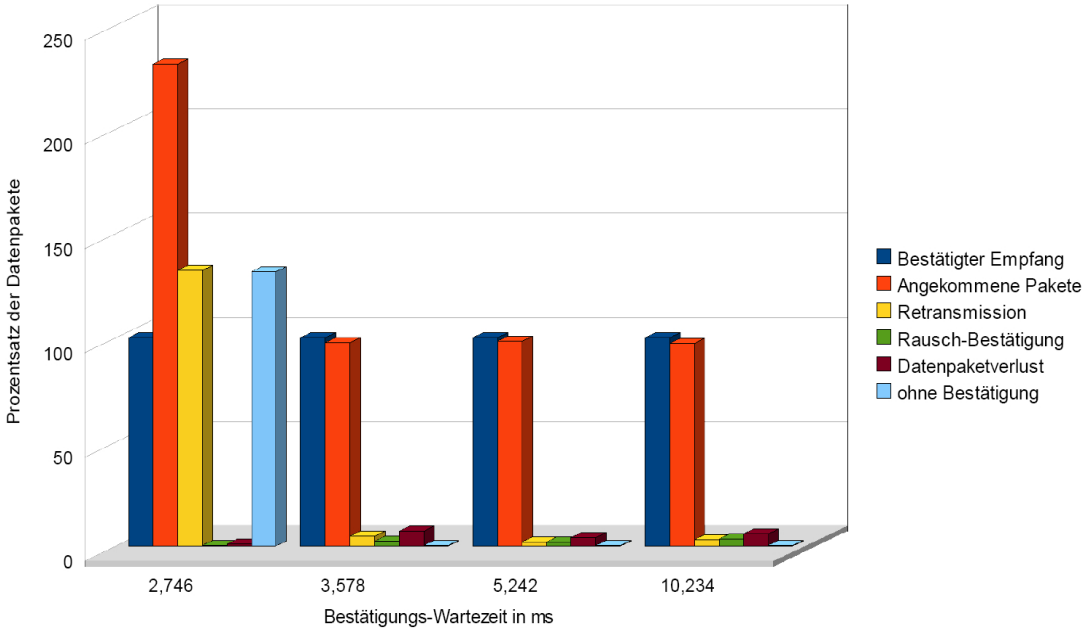

Abbildung 25: Ein Vergleichstest bei der Übetragung von 1100 Datenpakete mit der Bestätigungswartezeit von 2,746, 3,578 ms, 5,242 ms und 10,234 ms (entspricht 6, 8, 12 und 24 SPI-Interrupts inklusive TX-RX Umschaltzeit). Innerhalb eines einzelnen SPI-Interrupt-Schrittes können bis zu 8 Bits (1 Byte) versendet oder empfangen werden.

Entfernung	TX	RX	Retransmission	Rausch *ACK*	Paketverlust
1 Meter	2000	1976	56	24	80
5 Meter	2000	1984	21	16	37
10 Meter	2000	1991	9	9	18

Tabelle 6: Ergebnisse eines Tests für die Effektivität der Retransmission im geschlossenen Raum mit **3,578** *ms* Bestätigungs-Wartezeit.

Entfernung	TX	RX	Retransmission	Rausch *ACK*	Paketverlust
1 Meter	2000	1985	28	15	43
5 Meter	2000	1981	36	19	55
10 Meter	2000	1989	16	11	27

Tabelle 7: Ergebnisse eines Tests für die Effektivität der Retransmission im geschlossenen Raum mit **5,242** *ms* Bestätigungs-Wartezeit.

6.2.1 Test 2.1: Versenden und Empfangen im geschlossenen Raum

In diesem Test wurden 100 Datenpakete in zwanzig Schritten vom Sender zum Empfänger mit eingeschalteter Empfangsbestätigung gesendet. Die Transmissions-Routine des CC1000 Funkmoduls sollte solange neue Pakete senden, bis sie für jede einzelne von hundert Sende-Operationen eine Bestätigung erhalten hat. Der Versuch wurde mit zwei unterschiedlichen Bestätigungs-Wartezeiten (3,578 *ms* und 5,242 *ms*) über eine Distanz von einem, fünf bzw. zehn Metern ausgeführt.

Wie Tabelle 6 und Tabelle 7 zeigen, sinkt die Anzahl der Paketverluste und Retransmissionen überraschend mit steigender Entfernung. Sonst gibt es nur kleine Unterschiede zwischen 3,578 *ms* und 5,242 *ms* Wartezeit bei der drahtlosen Übetragung im geschlossenen Raum. Tabelle 6 und Tabelle 7 zeigen das Verhältnis zwischen der Anzahl der Nachrichten, die mit einer Empfangsbestätigung übertragen wurden, der Zahl der tatsächlich beim Empfänger angekommenen Nachrichten, der Summe der Nachrichten, die nochmal gesendet wurden, der Menge der Nachrichten, die auf dem Weg zum Empfänger verloren gegangen sind und der Anzahl der *falschen* Empfangsbestätigungen, die der Sender empfangen hat. Die Ergebnisse in Tabelle 6 stehen für die Bestätigungs-Wartezeit gleich 3,578 *ms*. Dementsprechend beziehen sich die Ergebnisse in Tabelle 7 auf die Übertragungs-Versuche mit der Bestätigungs-Wartezeit gleich 5,242 *ms*.

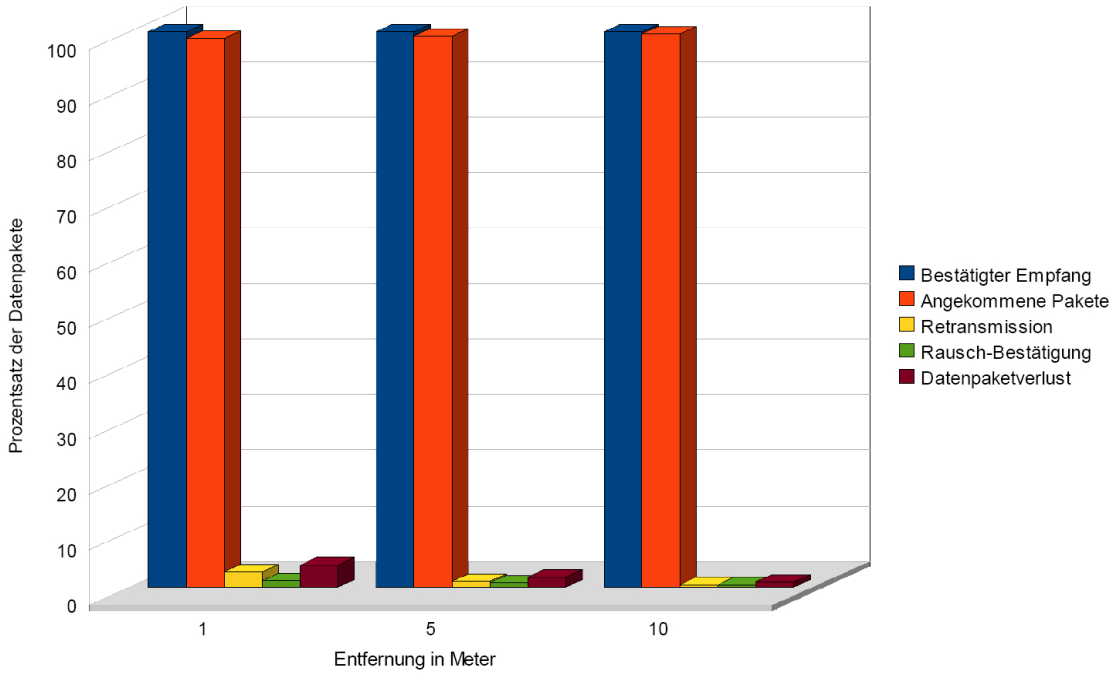

Abbildung 26: Übertragung im geschlossenen Raum mit 3,578 ms Bestätigungs-Wartezeit.

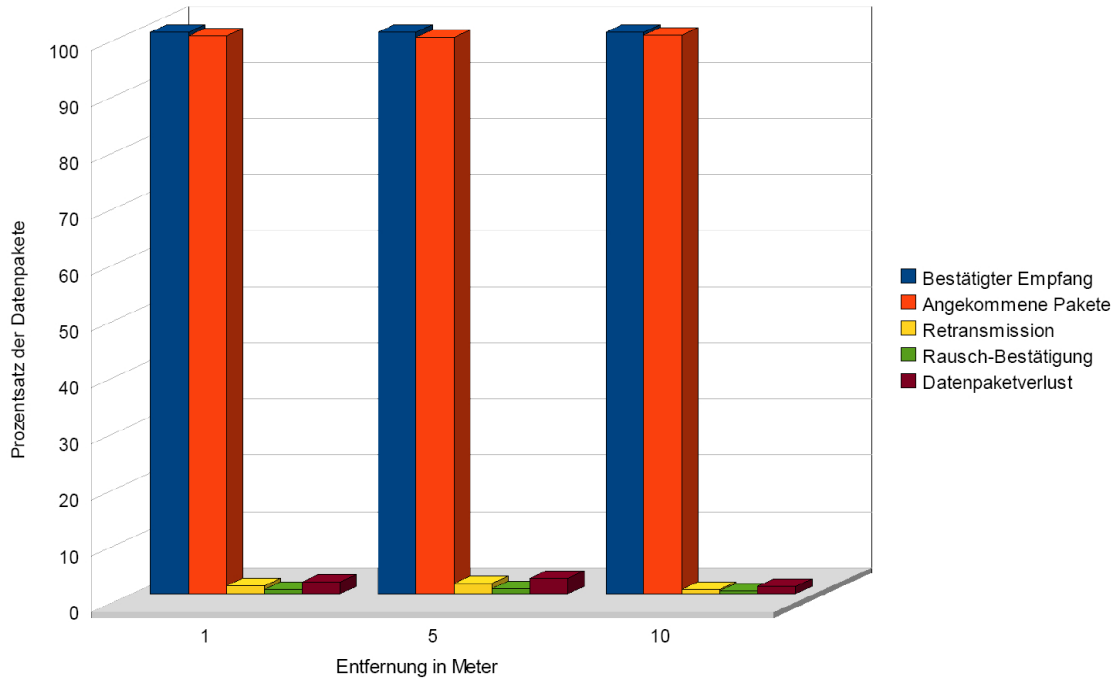

Abbildung 27: Übertragung im geschlossenen Raum mit 5,242 ms Bestätigungs-Wartezeit.

Entfernung	TX	RX	Retransmission	Rausch *ACK*	Paketverlust
5 Meter	2000	1987	12	13	25
37 Meter	2000	1987	21	15	34
65 Meter	2000	2166	518	0	372

Tabelle 8: Ergebnisse eines Tests für die Effektivität der Retransmission im Freien mit **3,578** *ms* Bestätigungs-Wartezeit.

Entfernung	TX	RX	Retransmission	Rausch *ACK*	Paketverlust
5 Meter	2000	1971	15	29	44
37 Meter	2000	1991	12	9	21
65 Meter	2000	2042	272	0	238

Tabelle 9: Ergebnisse eines Tests für die Effektivität der Retransmission im Freien mit **5,242** *ms* Bestätigungs-Wartezeit.

6.2.2 Test 2.2: Versenden und Empfangen im Freien

Der zweite Testversuch fand auf dem Campusgelände der Fakultät für Angewandte Wissenschaften der Universität Freiburg statt. Zwischen den Knoten befanden sich keine Hindernisse, die die drathlose Komminikation beeinträchtigen könnten. Es wurden erneut 2000 Datenpakete mit jeweils 100 Nachrichten in 20 Schritten über die Entfernung von fünf, 37 und 65 Meter übertragen. Abbildung 29 und Abbildung 30 zeigen den Zusammenhang zwischen der Anzahl der Sende-Operationen mit Retransmission, der Anzahl der verlorenen Pakete und der Entfernung, über welche die Datenübertagung stattfand.

Hier ist der Zusammenhang der Entfernung mit dem Verhältnis der gesendeten und empfangenen Datenpakete mit Bestätigungs-Wartezeit von 3,578 *ms* und 5,242 *ms* sehr deutlich sichtbar. Während sich bei der Übertragung über fünf bis 37 Meter die Zahl der Retransmissionen mit Bestätigungs-Wartezeit von 3,578 *ms* dicht innerhalb von 0,6% bis 1,05% der Gesamtmenge an gesendeten Datenpaketen (2000 Pakate) befindet, werden bei einer Entfernung von 65 Metern 25,9% der Nachrichten nochmals versendet. Die Zahl der überschüssigen Pakete, deren Empfangsbestätigung innerhalb der Bestätigungs-Wartezeit von 3,578 *ms* beim Sender nicht angekommen ist, obwohl sie vollständig empfangen wurden ergibt 8,3% bei einer Distanz von 65 Metern. Bei einer Distanz von fünf bis 37 Metern ist diese Anzahl auf die negativen -0,65% gesunken. Das heißt, dass der Sender für genau 13 Datenpakete eine Empfangsbestätigung erhalten hat, die aber nie vom Empfänger gesendet wurden. Es handelt sich hier aber um eine relativ kleine Anzahl der *Rausch-Bestätigungen*. Bei 65 Meter Entfernung beträgt der Datenpaketverlust 18,6% der Gesamtmenge an gesendeten Datenpaketen (2000 Pakate). Bei dem Knotenabstand von fünf bis 37 Metern gingen nur zwischen 1,25% und 1,7% Pakete verloren, was weitgehend akzeptabel ist. Der Anteil der nötigen Retransmissionen mit Bestätigungs-Wartezeit von 5,242 *ms* über fünf bis 37 Meter beläuft sich auf 0,75% und 0,6%. Bei der Distanz von 65 Meter machten die Retransmissionen nur 13,6% der Gesamtmenge der Pakete aus. Das ist knapp über der Hälfte der Retransmissionenen über 65 Meter Entfernung mit der Bestätigungs-Wartezeit von 3,578 *ms*. Die Größe der Rausch-Bestätigungen bei der Übertragung über 5 bis 37 Meter ergab 1,45% und 0,45% der Gesamtmenge an gesendeten Nachrichten.

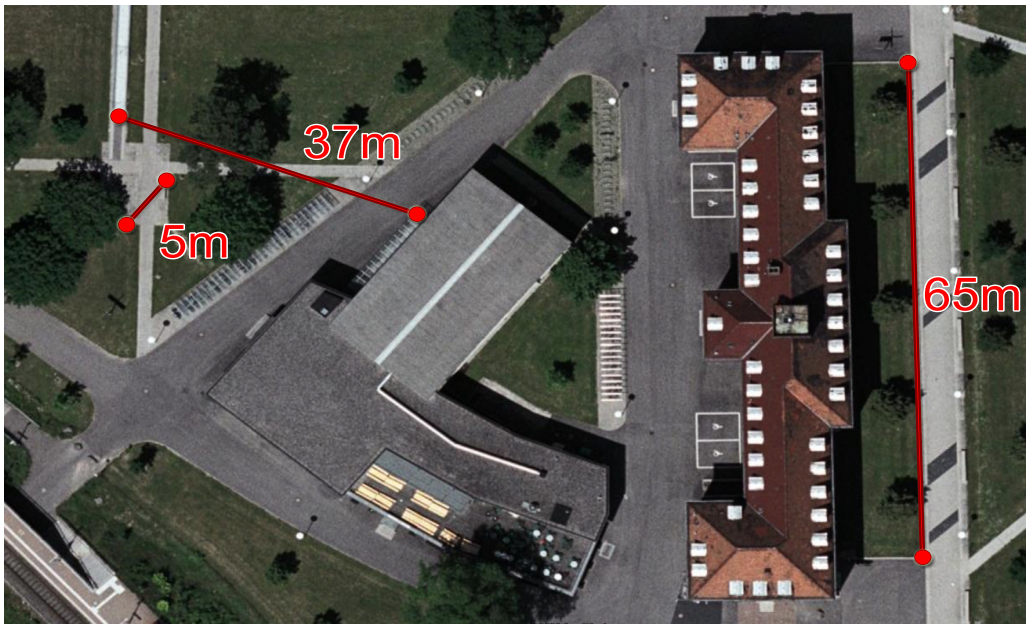

Abbildung 28: Übertragungstests auf dem Campusgelände der Fakultät für Angewandte Wissenschaften der Universität Freiburg. *Quelle: http://maps.google.de/*

Mit 65 Meter Distanz für 2,1% der empangenen Nachrichten kam die Bestätigung bei dem Sender nicht an. So ist bei dem Empfänger ein Überschuß von 2,1% der Nachrichten entstanden. Der Datenpaketverlust beziffert sich mit Bestätigungs-Wartezeit von 5,242 *ms* und Entfernung von 5 bis 37 Meter auf 2,2% und 1,05% der Gesamtmenge der Übertragung (2000 Paketen). Bei einer drahtlosen Übetragung Bestätigungs-Wartezeit von 5,242 *ms* und Entfernung von 65 Meter muss mit einem Paketverlust von 11,9% gerechnet werden.

6.3 Test 3: *„Clear-Channel-Assessment-Deadlock“*

Weil zu diesem Zeitpunkt keine Timer-Fuktionalität beretgestellt wurde, war der Einbau der *„Backoff-Funktion“* nicht möglich. Aus diesem Grund wurde ein *„Deadlock“* erwartet, falls mindestens zwei Knoten gleichzeitig den Sende-Vorgang einleiten. Während eines Versuchs mit zwei Sendern, die gleichzeitig ihre Daten zu Übertragen anfangen sollten, hat sich gezeigt, dass beide Knoten nicht in der Lage waren ihre Nachrichten zu versenden. Der CCA-Algorithmus prüft vor jeder Sende-Operation die Kanalfreiheit. Der Kanal gilt nur dann als besetzt, falls das Rauschniveau entsprechend hoch ist. Das kann nur dann auftreten, wenn ein Knoten sich aktuell im Sende-Modus befindet. Nach jeder Sende-Operation wird aber der Sende-Modus für eine kurze Zeit auf den Empfangs-Modus gewechselt. In dieser Zeit versucht der Sender eine Empfangsbestätigung für soeben gesendete Nachricht zu erhalten. Währenddessen gilt der Kanal für jeden anderen Knoten, der gerade mit der Übertragung seiner Nachrichten beginnt als frei. Da in den früheren Tests die Dauer der Wartezeitschranke mit zwischen 3,578 *ms* und 5,242 *ms* relativ kurz war, ist es nicht möglich innerhalb dieser Zeit ein komplettes Datenpaket zu versenden.

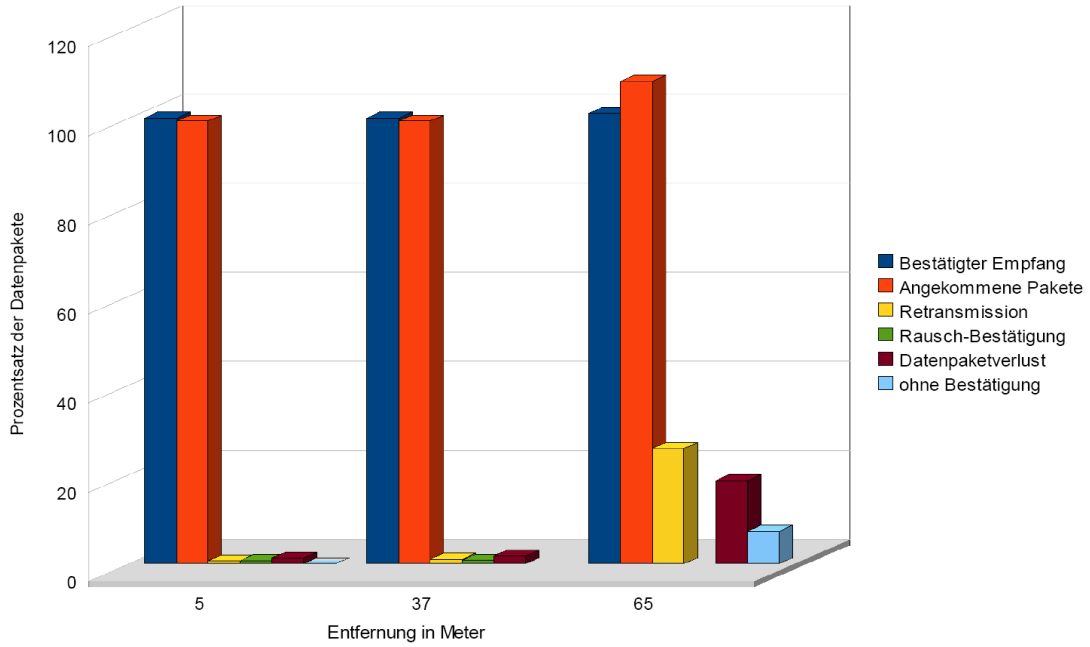

Abbildung 29: Übertragung im Freien mit 3,578 *ms* Bestätigungs-Wartezeit.

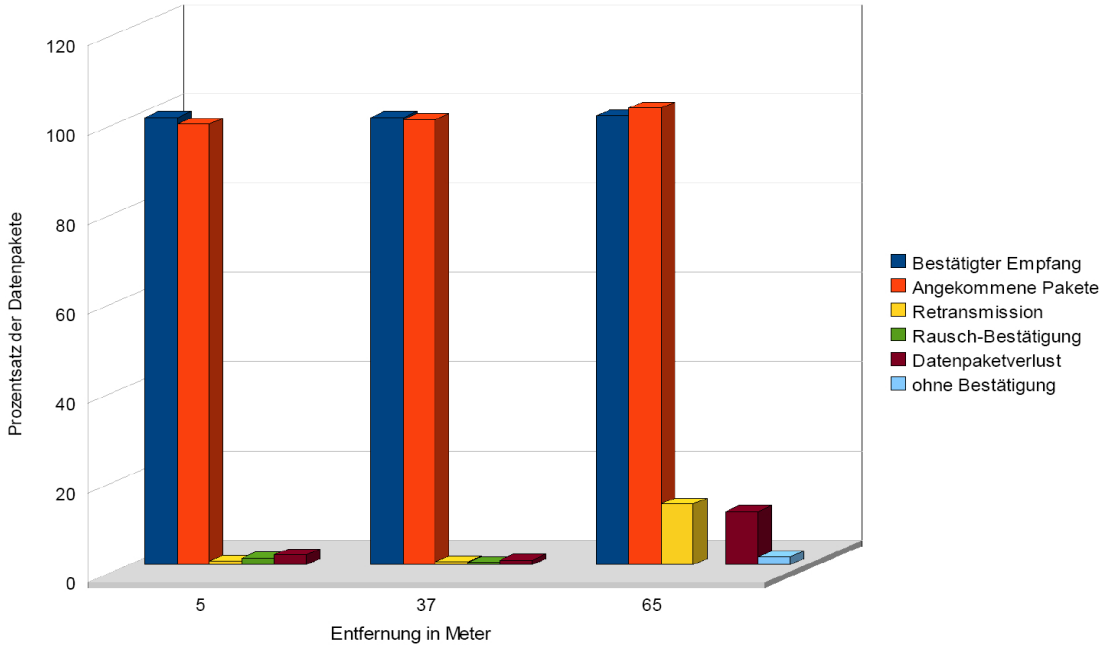

Abbildung 30: Übertragung im Freien mit 5,242 *ms* Bestätigungs-Wartezeit.

7 Zusammenfassung

Ziel dieser Arbeit war die Entwicklung eines Software-Treibers für das CC1000 Funkmodul auf der Mica2-Plattform. Nach einem Überblick über die Mica2-Plattform und der Beschreibung der für die Treiberentwicklung relevanten Hardware-Komponenten, wurde die Implementierung des Software-Teils vorgestellt. Die hardwarespezifischen Konfigurations-Routinen und deren Umsetzung wurden eingehend beschrieben. Ein großer Teil der Entwicklung war der Entwurf eines Netzwerkprotokolls, das für die Aufgaben der physikalischen, der Verbindungs- und Teile der MAC-Schicht (Medium-Access-Control-Schicht) verantwortlich ist. Duch das Netzwerkprotokoll werden die Datenpakete in einem bestimmten Datenformat und nach einem bestimmten Muster versendet und empfangen.

Die vorgestellte Anwendung für das Crossbow CC1000 Funkmodul und den ATmega128L Mikrocontroller wurde in einem auf drei Monate beschränkten Zeitrahmen zu einem funktionierenden Prototypen entwickelt. Nach dem Entwicklungsprozess wurde der Treiber-Prototyp ausfühlich getestet. Die Anwendung ist in der Lage alle möglichen Einstellungen des CC1000 Funkmoduls zu bedienen. Das Funkmodul kann initialisiert, kalibriert und vom Treiber betriebsbereit innerhalb der 868 MHz Funkfrequenz eingestellt werden. Die Datenübetragung wurde während zahlreicher Versuche auf die einwandfreie Funktionalität intensiv getestet. Diese konnte dank der Empfangsbestätigung nachgewiesen werden.

Die sekundäre Aufgabe der Testversuche war die Optimierung der Empfangbestätigungs-Wartezeit beim Sender-Knoten. Demzufolge wurde eine optimale Zeitschranke für die Empfangbestätigungs-Wartezeit zwischen 3,578 *ms* und 5,242 *ms* festgestellt. Überraschenderweise ist bei der Übertragung über kurze Distanz, sowohl im geschlossenen Raum als auch im Freien eine erhöhte Anzahl von fehlerhaften Bestätigungen durch Rauschen zu verzeichnen.

7.1 Ausblick

In der jetzigen Form der durchgeführten Implementiertung ist noch Raum für Verbesserungen. Wie bereits erwähnt, ist die Empfangsbestätigung über kurze Distanz nicht so effektiv wie erwartet. Dieser Effekt könnte genauer untersucht und optimiert werden. Mit der Implementierung der *Timer*-Routinen könnte das B-MAC Protokoll vollständig implementiert werden. Fehlend sind ausschliesslich die korrekte Implementierung der Schlaf- und Aufwach-Intervalle des Sensorknotens.

Abbildungsverzeichnis

Tabellenverzeichnis

Literatur

[1] Atmel. *8-bit AVR Microcontroller with 128K Bytes In-System Programmable Flash*. Atmel Corporation, http://www.atmel.com/dyn/resources/prod_documents/doc2467.pdf, 2467p-avr-08/07 edition, Aug 2007. zugegriffen am 09.06.2008.

[2] Christian Büch. SPI-Serial Peripheral Interface. Technical report, Universität Koblenz-Landau, http://www.uni-koblenz.de/~physik/informatik/MCU/SPI.pdf, 2006. zugegriffen am 09.06.2008.

[3] Chipcon. *2.4 GHz IEEE 802.15.4 / ZigBee-ready RF Transceiver*. Chipcon Products from Texas Instruments, http://focus.ti.com/lit/ds/symlink/cc2420.pdf, swrs041b edition, Mar 2007. zugegriffen am 09.06.2008.

[4] Chipcon. *Single Chip Very Low Power RF Transceiver*. Chipcon Products from Texas Instruments, http://focus.ti.com/docs/prod/folders/print/cc1000.html, swrs048a, revision a edition, Feb 2007. zugegriffen am 09.06.2008.

[5] Chipcon. *SmartRF® Studio User Manual Rev.6.9.0*. Chipcon Products from Texas Instruments, http://focus.ti.com/lit/ug/swru070e/swru070e.pdf, rev. 6.9.0 edition, 2008. zugegriffen am 09.06.2008.

[6] Crossbow. *MPR/MIB User´s manual*. Crossbow Technology, Inc., http://www.xbow.com/support/Support_pdf_files/MPR-MIB_Series_Users_Manual.pdf, doc. 7430-0021-08 rev. a edition, 2002-2007. zugegriffen am 09.06.2008.

[7] Joseph Polastre/Jason Hill/David Culler. Versatile low power media access for wireless sensor networks. In *SenSys '04: Proceedings of the 2nd international conference on Embedded networked sensor systems*, pages 95–107, http://doi.acm.org/10.1145/1031495.1031508, 2004. ACM. zugegriffen am 09.06.2008.

[8] Jaein Jeong/Sukun Kim. CS262A Advanced Topics in Computer Systems:DOT3 Radio Stack. Technical report, University of California at Berkeley Electrical Engineering and Computer Science, http://www.cs.berkeley.edu/~jaein/papers/cs262_paper_dot3radio.pdf, Dec 2002. zugegriffen am 09.06.2008.

[9] Karl H. Torvmark. *Application Note CC1000/CC1050 Microcontroller interfacing*. Chipcon Products from Texas Instruments, http://www.lierda.com/upload/product/down_path/061225/06409390011670218l5.pdf, swra082, rev.3.0 edition, Nov 2002. zugegriffen am 09.06.2008.

[10] Svein Vetti. *Programming the CC1000 frequency for best sensitivity*. Chipcon Products from Texas Instruments, http://focus.ti.com/lit/an/swra080/swra080.pdf, swra080 edition, 2008. zugegriffen am 09.06.2008.